Security Audit of Autocompounder_v3

Conclusion

Audit was made by “BlockSec” team.
The auditor telegram is `@vladimirdir`

Found notes (see below)

LEVEL	NUMBER
CRITICAL	TODO
MAJOR	TODO
WARNING	TODO
COMMENTS	TODO

TODO

```
In the final contract were not found:
- Backdoors for investor funds withdrawal by anyone.
- Bugs allowing to steal money from the contract.

All found security problems were fixed.

The final contract is deployed here - TODO
```

Scope

https://github.com/Recompounder-v3/Contracts (https://github.com/Recompounder-v3/Contracts)

bec76ee02f5f3c0ddec7b41e23d44c3dad121f55

TODO: IT WAS CHANGED BY THE LAST CODE FROM ARCHIVE

I used 6b9aa9746132578995b95898a5a1b63798200261 for this report

Methodology

1. Blind audit. Try to understand the structure of the code.
2. Find info in internet.
3. Ask quiestions to developers.
4. Draw the scheme of cross-contracts interactions.
5. Write user-stories, usage cases.
6. Run static analyzers

Find problems with:

- backdoors
- bugs
- math
- potential leaking of funds
- potential locking of the contract
- validate arguments and events
- others

Result

Critical

Major

1. User provided contract address called

Polygon/adapters/Sushiswap/SushiswapAdapter.sol:746

```
    // Pool actions, requires impersonation via delegatecall
    function deposit(address _adapter, uint256 poolId, uint256 amount) external
        IAdapter adapter = IAdapter(_adapter);
        IERC20 lpToken = adapter.lockableToken(poolId);
        lpToken.approve(address(sushiFarm), uint256(amount));
        sushiFarm.deposit(poolId, amount, address(this));
    }
```

.trsnfer

adapters/Alchemix/AlchemixAdapter.sol:841
adapters/dodo/DODOAdapter.sol:671

Recommendation

Use SafeERC20 everywhere.
Check that return-value is expected.

Warning

1. Return value is ignored

```
MasterVampire._deposit(uint256,uint256) (MasterVampire.sol#443-460) performs a
        -sharesNoFee = (change.mul(pool.totalShares)).div(depositsBefore) (Maste
        -newshares = (sharesNoFee.mul(feeAdjusted.sub(pool.entranceFee))).div(fe
```

Recommendation.

TODO

Status.

NEW

2. Zero checks in address set.

```
MasterContract.constructor(address)._drainAddress (MasterContract.sol#363) lack
                - drainAddress = _drainAddress (MasterContract.sol#365)
MasterContract.updateCustodian(address)._custodianAddress (MasterContract.sol#4
                - custodianAddress = _custodianAddress (MasterContract.sol#406)
MasterContract.updateService(address)._serviceAddress (MasterContract.sol#409)
                - serviceAddress = _serviceAddress (MasterContract.sol#410)
MasterContract.updateDrainAddress(address)._drainAddress (MasterContract.sol#44
                - drainAddress = _drainAddress (MasterContract.sol#445)

Timelock.constructor(address,uint256).admin_ (Timelock.sol#171) lacks a zero-ch
                - admin = admin_ (Timelock.sol#175)
Timelock.setPendingAdmin(address).pendingAdmin_ (Timelock.sol#199) lacks a zero
                - pendingAdmin = pendingAdmin_ (Timelock.sol#207)
Timelock.executeTransaction(address,uint256,string,bytes,uint256).target (Timel
                - (success,returnData) = target.call{value: _value}(callData) (T
```

3. Potential front-running due to misuse of `min_amount`

At

- Binance-Smart-Chain/adapters/autofarm/Auto_BeltAdapter.sol:757
- Binance-Smart-Chain/adapters/belt/BeltAdapter.sol:742
  min_amount is not used.

Also in execution branches:

- adapters/Alchemix/AlchemixAdapter.sol:847
- adapters/Alchemix/AlchemixAdapter.sol:901
- adapters/Alchemix/AlchemixAdapter.sol:907

mon_amount is not used, I do not understand why.

Also in some places they called `min_out` in other `min_amount`. Better to be consistent.

4. Huge rate of logic-duplication.

Increases the space to make a mistake `*= N`.
Very simple idea is implemented in dozens thsousands of lines! I beleive it could be 1000-2000.

Comments

5. Potential events misordering.

```
        External calls:
        - (success,returnData) = target.call{value: _value}(callData) (Timelock
        Event emitted after the call(s):
        - ExecuteTransaction(txHash,target,_value,signature,data,eta) (Timelock
```

```
        Event emitted after the call(s):
        - Deposit(msg.sender,_pid,_amount) (MasterContract.sol#453)
```

```
        Event emitted after the call(s):
        - ExecuteTransaction(txHash,target,_value,signature,data,eta) (Timelock
```

```
        Event emitted after the call(s):
        - Withdraw(msg.sender,_pid,_amount) (MasterContract.sol#515)
```

```
        Event emitted after the call(s):
        - WithdrawAndUnzap(msg.sender,_pid,_amount,token,lpTokens) (MasterContra
```

```
        Event emitted after the call(s):
        - ZapAndDeposit(msg.sender,_pid,_amount,token,lpTokens) (MasterContract
```

Recommendation.

Emit event before external call.

Status.

NOT_ISSUE

9. Use reentryGuard.

Use reentryGuard on every external method to be sure no reentry will happen.

Recommendation.

Use reentryGuard.

Status.

NEW

11. Methods should be declared external.

```
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (MasterContract.sol#295-298)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (MasterContract.sol#304-308)
add(Target,uint256,uint256,uint256) should be declared external:
        - MasterContract.add(Target,uint256,uint256,uint256) (MasterContract.so
addBulk(Target,uint256[],uint256[],uint256[]) should be declared external:
        - MasterContract.addBulk(Target,uint256[],uint256[],uint256[]) (MasterC
updateCustodian(address) should be declared external:
        - MasterContract.updateCustodian(address) (MasterContract.sol#405-407)
updateService(address) should be declared external:
        - MasterContract.updateService(address) (MasterContract.sol#409-411)
updateProtection(bool) should be declared external:
        - MasterContract.updateProtection(bool) (MasterContract.sol#413-415)
poolUpdate(uint256,uint256) should be declared external:
        - MasterContract.poolUpdate(uint256,uint256) (MasterContract.sol#417-42
updateTargetInfo(uint256,address,uint256,bool) should be declared external:
        - MasterContract.updateTargetInfo(uint256,address,uint256,bool) (Master
updateDrainModifier(uint256,uint256) should be declared external:
        - MasterContract.updateDrainModifier(uint256,uint256) (MasterContract.s
updateEntranceFee(uint256,uint256) should be declared external:
        - MasterContract.updateEntranceFee(uint256,uint256) (MasterContract.sol
updateDrainAddress(address) should be declared external:
        - MasterContract.updateDrainAddress(address) (MasterContract.sol#444-44
deposit(uint256,uint256) should be declared external:
        - MasterContract.deposit(uint256,uint256) (MasterContract.sol#448-454)
zapAndDeposit(address,address[],uint256,uint256,uint256) should be declared ext
        - MasterContract.zapAndDeposit(address,address[],uint256,uint256,uint25
withdrawAndUnzap(address,address[],uint256,uint256,uint256) should be declared 
        - MasterContract.withdrawAndUnzap(address,address[],uint256,uint256,uin
withdraw(uint256,uint256) should be declared external:
        - MasterContract.withdraw(uint256,uint256) (MasterContract.sol#501-516)
drain(uint256) should be declared external:
        - MasterContract.drain(uint256) (MasterContract.sol#518-546)
setDelay(uint256) should be declared external:
        - Timelock.setDelay(uint256) (Timelock.sol#182-189)
acceptAdmin() should be declared external:
        - Timelock.acceptAdmin() (Timelock.sol#191-197)
setPendingAdmin(address) should be declared external:
        - Timelock.setPendingAdmin(address) (Timelock.sol#199-210)
queueTransaction(address,uint256,string,bytes,uint256) should be declared exter
        - Timelock.queueTransaction(address,uint256,string,bytes,uint256) (Time
cancelTransaction(address,uint256,string,bytes,uint256) should be declared exte
        - Timelock.cancelTransaction(address,uint256,string,bytes,uint256) (Tim
executeTransaction(address,uint256,string,bytes,uint256) should be declared ext
        - Timelock.executeTransaction(address,uint256,string,bytes,uint256) (Ti
```

Recommendation.

Status.

12. Child does not implement the parent.

does not implement functions

Recommendation.

Status.

14. Language style.

```
Parameter ITarget.zapAny(address,address[],uint256,uint256,uint256,address).min
Parameter ITarget._zapAny(address,address[],uint256,uint256,uint256,address).mi
Parameter ITarget.unzapAny(address,address[],uint256,uint256,uint256,address).m
Parameter ITarget._unzapAny(address,address[],uint256,uint256,uint256,address).
Parameter MasterContract.zapAndDeposit(address,address[],uint256,uint256,uint25
Parameter MasterContract.withdrawAndUnzap(address,address[],uint256,uint256,uin
Constant MasterContract.feeAdjusted (MasterContract.sol#328) is not in UPPER_CA
Variable Timelock.admin_initialized (Timelock.sol#166) is not in mixedCase
```

Recommendation.

Follow language naming style.

15. Multiple calls are executed in the same transaction.

Polygon/adapters/Sushiswap/SushiswapAdapter.sol:743

```
    function deposit(address _adapter, uint256 poolId, uint256 amount) external
        IAdapter adapter = IAdapter(_adapter);
        IERC20 lpToken = adapter.lockableToken(poolId);
        lpToken.approve(address(sushiFarm), uint256(amount));
        sushiFarm.deposit(poolId, amount, address(this));//zz
    }
```

a lot of places

quickAdapter.sol:737
quickAdapter.sol:725

Recommendation

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

17. Multiplication after division.

MasterContract._deposit(uint256,uint256) (MasterContract.sol#456-473) performs a multiplication on the result of a division:
-sharesNoFee = (change.mul(pool.totalShares)).div(depositsBefore) (MasterContract.sol#466)
-newshares = (sharesNoFee.mul(feeAdjusted.sub(pool.entranceFee))).div(feeAdjusted) (MasterContract.sol#467)

Recommendation.

do not do

```
x = (a / b) * c
```

do

```
x = (a * c) / b
```

Status.

NOT_ISSUE

18. Unused local variable.

Unused local variable “amountOut”
Binance-Smart-Chain/adapters/belt/BeltAdapter.sol:746

Unused local variable “amount”
Binance-Smart-Chain/adapters/belt/BeltAdapter.sol:962

19. `feeAdjusted`

MasterContract.sol:328
uint256 constant feeAdjusted = 10000;
probably rename to BASE

20. add asserts

adapters/Alchemix/AlchemixAdapter.sol:794

```
if (lp.token0() == sellToken) {
    ...
} else {
    assert(lp.token1() == sellToken);
}
```

The same here

- adapters/dodo/DODOAdapter.sol:646
- adapters/dodo/DODOAdapter.sol:661

21. replace `require(false, "Bad Zap");`

with `revert(msg)`

22. Use SafeMath.

In some places division `/` is used. Better to use SafeMath for zero-check. https://ethereum.stackexchange.com/questions/3010/how-does-ethereum-cope-with-division-of-prime-numbers (https://ethereum.stackexchange.com/questions/3010/how-does-ethereum-cope-with-division-of-prime-numbers)

Questions

1. Various reentry weak places

MasterContract.sol:428-429

```
        State variables written after the call(s):
        - poolInfo[_pid].target = Target(_target) (MasterContract.sol#428)
        - poolInfo[_pid].targetPoolId = _targetPoolId (MasterContract.sol#429)
```

```
        State variables written after the call(s):
        - pool.totalShares = pool.totalShares.add(newshares) (MasterContract.so
        - pool.totalDeposits = depositsAfter (MasterContract.sol#471)
        - user.shares = user.shares.add(newshares) (MasterContract.sol#469)
```

```
        State variables written after the call(s):
        - pool.totalShares = pool.totalShares.sub(sharesToBurn) (MasterContract
        - pool.totalDeposits = depositsAfter (MasterContract.sol#513)
        - user.shares = user.shares.sub(sharesToBurn) (MasterContract.sol#511)
        State variables written after the call(s):
        - pool.totalShares = pool.totalShares.sub(sharesToBurn) (MasterContract
        - pool.totalDeposits = depositsAfter (MasterContract.sol#495)
        - user.shares = user.shares.sub(sharesToBurn) (MasterContract.sol#493)
```

```
        State variables written after the call(s):
        - _deposit(_pid,lpTokens) (MasterContract.sol#480)
                - pool.totalShares = pool.totalShares.add(newshares) (MasterCont
                - pool.totalDeposits = depositsAfter (MasterContract.sol#471)
```

Recommendation

Be careful.
Write the code in a way to avoid reentry as much as possible.

2. Convert to lockableToken in updateTargetInfo

function updateTargetInfo
MasterContract.sol:423
`pool.target.lockableToken(pool.targetPoolId).balanceOf(address(this))`
shold we convert withdrawn tokens to new target tokens?

3. unclear `this`

MasterContract.sol:241-244

```
        function _msgData() internal view virtual returns (bytes memory) {
            this; // silence state mutability warning without generating byteco
            return msg.data;
        }
```

4. User provided address method return address call method

> An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract.
> Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in
> place.

Polygon/adapters/quickswap/quickAdapter.sol:734

```
    function claimReward(address _adapter, uint256 poolId) external override {
        IAdapter adapter = IAdapter(_adapter);
        IQuickStake lp = IQuickStake(adapter.poolAddress(poolId));
        lp.getReward();
    }
```

Polygon/adapters/quickswap/quickAdapter.sol:728

```
    function withdraw(address _adapter, uint256 poolId, uint256 amount) externa
        IAdapter adapter = IAdapter(_adapter);
        IQuickStake lp = IQuickStake(adapter.poolAddress(poolId));
        lp.withdraw(amount);
    }
```

Polygon/adapters/quickswap/quickAdapter.sol:725

```
    function deposit(address _adapter, uint256 poolId, uint256 amount) external
        IAdapter adapter = IAdapter(_adapter);
        address lp = adapter.poolAddress(poolId);
        adapter.lockableToken(poolId).safeApprove(lp, uint256(0));
        adapter.lockableToken(poolId).safeApprove(lp, amount);
        IQuickStake(lp).stake(amount);
    }
```

various places

```
    // Pool actions, requires impersonation via delegatecall
    function deposit(address _adapter, uint256 poolId, uint256 amount) external
        IAdapter adapter = IAdapter(_adapter);
        address lp = adapter.poolAddress(poolId);
        adapter.lockableToken(poolId).safeApprove(lp, uint256(0));
        adapter.lockableToken(poolId).safeApprove(lp, amount);
        IHawkStake(lp).stake(amount);
    }

    function withdraw(address _adapter, uint256 poolId, uint256 amount) externa
        IAdapter adapter = IAdapter(_adapter);
        IHawkStake lp = IHawkStake(adapter.poolAddress(poolId));
        lp.withdraw(amount);
    }

    function claimReward(address _adapter, uint256 poolId) external override {
        IAdapter adapter = IAdapter(_adapter);
        IHawkStake lp = IHawkStake(adapter.poolAddress(poolId));
        lp.getReward();
    }
```

5. Dangerous `_functionCallWithValue`

various places

```
(bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
```

6. Tricky code

Binance-Smart-Chain/adapters/autofarm/Auto_BeltAdapter.sol:757

```
        if (poolId == 338) {
            poolId = 9;
        }
        if (poolId == 339) {
            poolId = 7;
        }

        if (poolId == 340) {
            poolId = 8;
        }

        if (poolId == 341) {
            poolId = 3;
        }
```

```
            if (sellToken == DAI) {
                IERC20(sellToken).safeIncreaseAllowance(beltStableAddress, amou
                uint256[4] memory uamounts = [amount, 0, 0, 0];
                ILiquidityBelt(beltStableAddress).add_liquidity(uamounts, uint2
                uint256 amountOut = stableLP.balanceOf(address(this));
                IERC20(stableLP).safeTransfer(to, amountOut);
                return amountOut;
            }
```

```
if (poolId == 0 || poolId == 4) {
```

7. Why so much `require(false, "not implemented");`

```
require(false, "not implemented");
```

8. Discuss `functionCallWithValue` usage.

Not really clear.

9. Discuss contracts state sync on update address

Various places where interacted contracts address changed, state should be updated.

10. Unclear branch

adapters/ hemix/AlchemixAdapter.sol:784

```
                if (sellToken == token0 || sellToken == token1) {
                    amountOut = amount.div(2);
                }
```

11. Unclear statement.

adapters/Alchemix/AlchemixAdapter.sol:804

```
amount = amount.div(2);
```

```
                if (sellToken == token0 || sellToken == token1) {
                    amount = amount.mul(2);
                }
```

11. Alchemix._zapAny execution branches.

Discuss
adapters/Alchemix/AlchemixAdapter.sol:775

12. Unclear sellableRewardAmount

A lot of methods with

```
  sellableRewardAmount = uint256(-1);
```

But

```
                  uint256 claimedReward = rewardToken.balanceOf(address(this));
                  uint256 sellableAmount = target.sellableRewardAmount();
                  if(sellableAmount < claimedReward) { // target is drained empty
                      claimedReward = sellableAmount;
                  }
                  if(claimedReward == 0) {
                      return;
                  }
                  uint256 feeAmount = claimedReward.mul(pool.drainModifier).div(f
                  claimedReward = claimedReward.sub(feeAmount);
```

13. Unsafe approve

IERC20(originToken).safeApprove(address(pairLP), uint256(-1));
Then originToken of the pool (collected by fees) could be affected.

14.

You do

```
                      lp0.swap(amountOutput, uint(0), address(this), new bytes(0)
```

no

But

adapters/vesper/VesperAdapter.sol:763

also whole