SMART CONTRACT AUDIT

DAO Maker

PROJECT: DAOMAKER

METHODOLOGY

Main tests list:

- Best code practices
- ERC20/BEP20 compliance (if applicable)
- Logical bugs
- General Denial Of Service(DOS)
- Locked ether
- Private data leaks
- Using components with known vulns
- Weak PRNG
- Unsed vars
- Uncheked call return method
- Code with no effects
- Function visibility
- Use of deprecated functions
- Authorization issues
- Re-entrancy
- Arithmetic Over/Under Flows
- Hidden Malicious Code
- External Contract Referencing
- Short Address/ Parameter Attack
- Race Conditions / Front Running
- Uninitialized Storage Pointers
- Floating Points and Precision
- Signatures Replay
- Pool Asset Security (backdoors in the underlying ERC-20)

STRUCTURE OF CONTRACT
FARM.SOL

Recommended to add onlyManager modifier instead of duplicating requirement in each method.

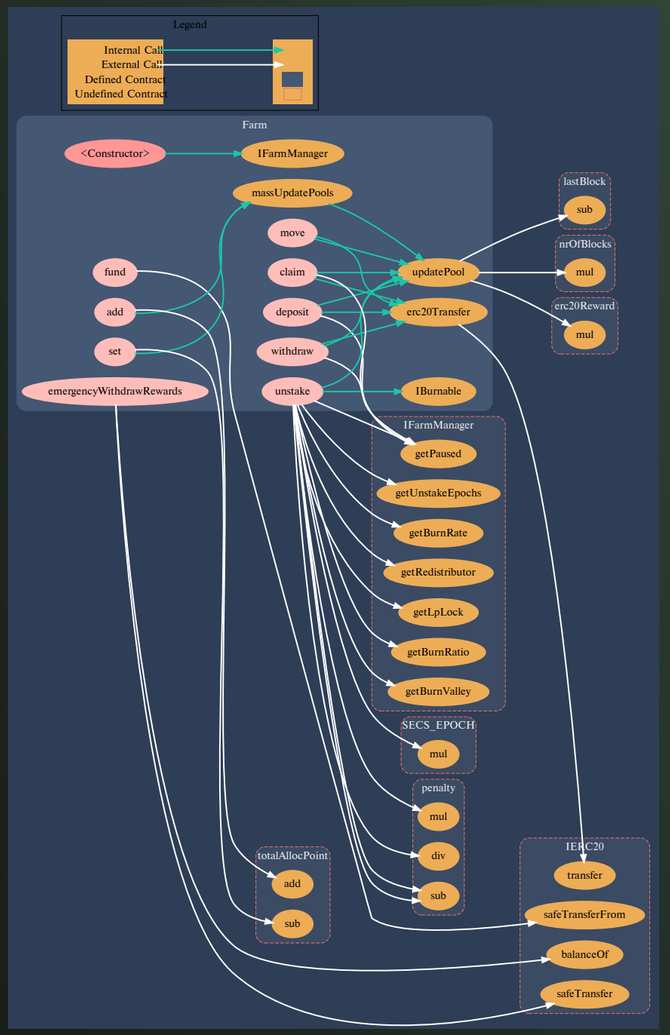
Contract methods analysis

function fund(address _funder, uint256 _amount)
Vulnerabilities not detected

function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate)
Vulnerabilities not detected

function add(uint256 _allocPoint, IERC20 _stakingToken, bool _isLP, bool _isBurnable, bool _withUpdate)
Vulnerabilities not detected

function massUpdatePools()
Vulnerabilities not detected



Pic. 1.1.
Farm.sol

function updatePool(uint256 _pid)
1e36 perscision is only compatibale for tokens with 18 decimals. So if you are going to use tokens with another decimals, perscision should be 10**(2*token_decimals)

function move(uint256 _pid, address _mover)
1e36 perscision is only compatibale for tokens with 18 decimals. So if you are going to use tokens with another decimals, perscision should be 10**(2*token_decimals)

function deposit(uint256 _pid, address _depositor, uint256 _amount)
1e36 perscision is only compatibale for tokens with 18 decimals. So if you are going to use tokens with another decimals, perscision should be 10**(2*token_decimals)

function withdraw(uint256 _pid)
1e36 perscision is only compatibale for tokens with 18 decimals. So if you are going to use tokens with another decimals, perscision should be 10**(2*token_decimals)

function unstake(uint256 _pid)
Vulnerabilities not detected

function claim(uint256 _pid)
1e36 perscision is only compatibale for tokens with 18 decimals. So if you are going to use tokens with another decimals, perscision should be 10**(2*token_decimals)

function erc20Transfer(address _to, uint256 _amount)
Recommended to use safeTrasnfer

function emergencyWithdrawRewards(address _receiver)
Recommended to delegate this method to funders. Each funder should have an ability to withdraw his part of yet unclaimed rewards. Basically funders shouldn't be able to withdraw anything until owner calls e.g confirm method and allows them to withdraw. Even if owner's wallet is compromised all tokens won't be stollen because the belong to different funders.

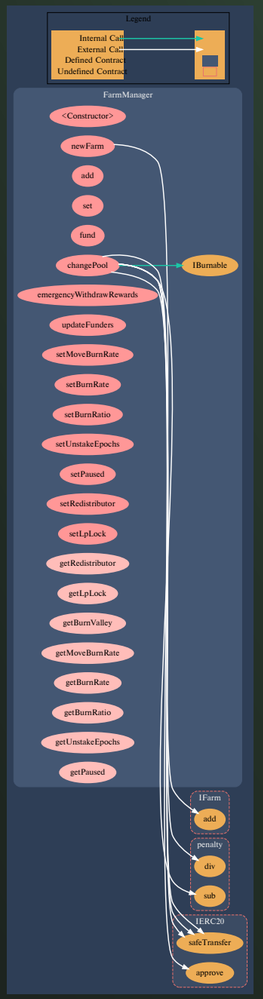
function emergencyWithdrawStake(uint256 _pid)
1e36 perscision is only compatibale for tokens with 18 decimals. So if you are going to use tokens with another decimals, perscision should be 10**(2*token_decimals)

STRUCTURE OF CONTRACT
FARMMANAGER.SOL

Contract methods analysis

function newFarm(IFarm farm)
In case if owner's wallet is comprosied a malicious farm can be added. Farm methods will be simillar to all existing farms, but but the inside logic will allow third parties to withdraw for example staking tokens. What we recommend doing here is to create farms via clones(Clones.sol contract from openzepplein). First of all it's much cheaper than deploying contract with constructor and secondly attacker won't be able to deploy malicious farm, because he can only clone the existing one. Method should emit an event

massClaimNaturalTokens(address[] calldata receivers)
Vulnerabilities not detected



Pic. 1.2.
FarmManager.sol

function add(uint allocPoint, IERC20 stakingToken, bool isLP, bool isBurnable)
Malicious stake token can be added with huge allocation. So in case attacker will have an ability to add a custom staking token with big AllocPoint, he will own 100% token in the pool and because of allocPoint will be able to drain all the rewards from the farm.
We recommend to remove ability to add custom staking tokens(all required tokens should be added at deployment). Method should emit an event

function set(uint[] calldata allocPoints, uint _fid, bool _withUpdate)
We recommend to limit the allocPoint, basing on existing allocPoints in the farm. Method should emit an event

function fund(uint _fid, uint256 _amount)
Method should emit an event.

function changePool(uint _currentFid, uint _nextFid, uint _pid)
Recommended to rename method to changeFarm in order to fit it's logic. Method should emit an event.

function emergencyWithdrawRewards(uint256 _fid)
Recommended to delegate this method to funders. Each funder should have an ability to withdraw his part of yet unclaimed rewards. Basically funders shouldn't be able to withdraw anything until owner calls e.g confirm method and allows them to withdraw. Even if owner's wallet is compromised all tokens won't be stollen because the belong to different funders. Method should emit an event.

function updateFunders(address funder, bool _set)
Method should emit an event.

unction setMoveBurnRate(uint256 _moveBurnRate)
Method should emit an event.

function setBurnRate(uint256 _burnRate)
Method should emit an event.

function setBurnRatio(uint256 _burnRatio)
Method should emit an event.

function setUnstakeEpochs(uint256 _unstakeEpochs)
Method should emit an event.

function setPaused(bool _paused)
Method should emit an event.

function setRedistributor(address _redistributor)
Method should emit an event.

function setLpLock(address _lpLock)
Method should emit an event.

function getRedistributor() external view returns(address)
Method should emit an event.

function getLpLock() external view returns(address)
Method should emit an event.

function getBurnValley() external view returns(address)
Method should emit an event.

function getMoveBurnRate() external view returns(uint256)
Method should emit an event.

function getBurnRate() external view returns(uint256)
Method should emit an event.

function getBurnRatio() external view returns(uint256)
Method should emit an event.

function getUnstakeEpochs() external view returns(uint256)
Method should emit an event.

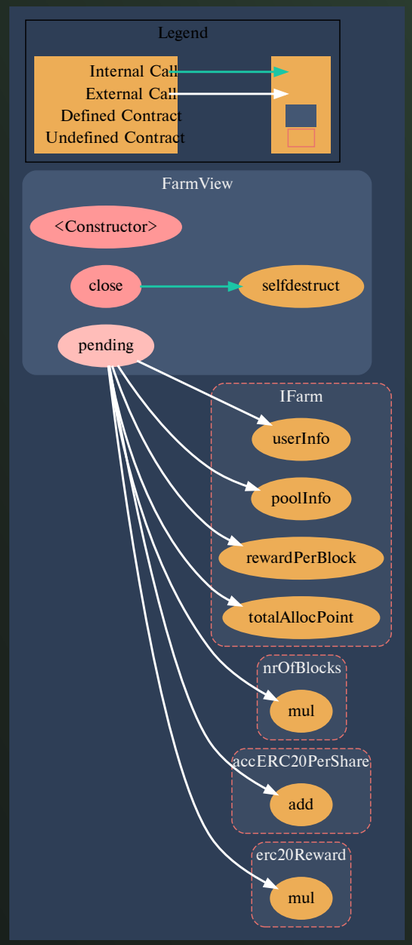
function getPaused() external view returns(bool)
Method should emit an event.

STRUCTURE OF CONTRACT
FARMVIEW.SOL

Contract methods analysis

function pending(IFarm farm, uint256 _pid, address _user) external view returns (uint256)
Vulnerabilities not detected

function close()
Vulnerabilities not detected



Pic. 1.3.
FarmView.sol

GET IN TOUCH

info@smartstate.tech
smartstate.tech