MythX

REPORT 60503644EE5C650019AF2BD5

Created	Tue Mar 16 2021 04:38:28 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	admin@pineappledefi.com

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
5fbec4ee-9d04-4e15-88d9-a0e0aee47fd3	browser/contracts/MasterChef.sol	41

Analysis 5fbec4ee-9d04-4e15-88d9-a0e0aee47fd3

MythX

Started	Tue Mar 16 2021 04:38:39 GMT+0000 (Coordinated Universal Time)
Finished	Tue Mar 16 2021 04:41:01 GMT+0000 (Coordinated Universal Time)
Mode	Quick
Client Tool	Remythx
Main Source File	Browser/Contracts/MasterChef.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	22	19

ISSUES

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "renounceOwnership" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

browser/contracts/MasterChef.sol

Locations

```
* thereby removing any functionality that is only available to the owner.
*/
function renounceOwnership() public virtual onlyOwner {
emit OwnershipTransferred(_owner, address(0));
_owner = address(0);
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transferOwnership" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
* Can only be called by the current owner.
*/
function transferOwnership(address newOwner) public virtual onlyOwner {
require(newOwner != address(0), "Ownable: new owner is the zero address");
emit OwnershipTransferred(_owner, newOwner);
_owner = newOwner;
}
}
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "symbol" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
* name.
*/
function symbol() public override view returns (string memory) {
return _symbol;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "decimals" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
* @dev Returns the number of decimals used to get its user representation.
*/
function decimals() public override view returns (uint8) {
return _decimals;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "totalSupply" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

browser/contracts/MasterChef.sol

Locations

```
* @dev See {BEP20-totalSupply}.
*/
function totalSupply() public override view returns (uint256) {
return _totalSupply;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transfer" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

browser/contracts/MasterChef.sol

Locations

```
* - the caller must have a balance of at least `amount`.
*/
function transfer(address recipient, uint256 amount) public override returns (bool) {
_transfer(_msgSender(), recipient, amount);
return true;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "allowance" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

browser/contracts/MasterChef.sol

Locations

```
* @dev See {BEP20-allowance}.
*/
function allowance(address owner, address spender) public override view returns (uint256) {
return _allowances[owner][spender];
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "approve" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
* - `spender` cannot be the zero address.
*/
function approve(address spender, uint256 amount) public override returns (bool) {
_approve(_msgSender(), spender, amount);
return true;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transferFrom" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
* `amount`.
*/
function transferFrom (address sender, address recipient, uint256 amount) public override returns (bool) {
_transfer(sender, recipient, amount);
_approve(
sender,
_msgSender(),
_allowances[sender][_msgSender()].sub(amount, 'BEP20: transfer amount exceeds allowance')
);
return true;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "increaseAllowance" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
* - `spender` cannot be the zero address.
*/
function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
_approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
return true;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "decreaseAllowance" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
* `subtractedValue`.
*/
function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
_approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, 'BEP20: decreased allowance below zero'));
return true;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "mint" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
* - `msg.sender` must be the token owner
*/
function mint(uint256 amount) public onlyOwner returns (bool) {
_mint(_msgSender(), amount);
return true;
}

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "mint" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
contract PineappleToken is BEP20('Pineapple', 'PIN') {
/// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
function mint(address _to, uint256 _amount) public onlyOwner {
_mint(_to, _amount);
_moveDelegates(address(0), _delegates[_to], _amount);
}

// Copied and modified from YAM code:
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "add" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
// Add a new lp to the pool. Can only be called by the owner.
// XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner {
require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
if (_withUpdate) {
massUpdatePools();
}
uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
totalAllocPoint = totalAllocPoint.add(_allocPoint);
poolInfo.push(PoolInfo({
lpToken: _lpToken,
allocPoint: _allocPoint,
lastRewardBlock: lastRewardBlock,
accPinPerShare: 0,
depositFeeBP: _depositFeeBP
}));
}

// Update the given pool's PIN allocation point and deposit fee. Can only be called by the owner.
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "set" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```

// Update the given pool's PIN allocation point and deposit fee. Can only be called by the owner.
function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner {
require(_depositFeeBP <= 10000, "set: invalid deposit fee basis points");
if (_withUpdate) {
massUpdatePools();
}
totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
poolInfo[_pid].allocPoint = _allocPoint;
poolInfo[_pid].depositFeeBP = _depositFeeBP;
}

// Return reward multiplier over the given _from to _to block.
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "deposit" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```

// Deposit LP tokens to MasterChef for PIN allocation.
function deposit(uint256 _pid, uint256 _amount) public {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
updatePool(_pid);
if (user.amount > 0) {
uint256 pending = user.amount.mul(pool.accPinPerShare).div(1e12).sub(user.rewardDebt);
if(pending > 0) {
safePinTransfer(msg.sender, pending);
}
}
if(_amount > 0) {
pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
if(pool.depositFeeBP > 0){
uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
pool.lpToken.safeTransfer(feeAddress, depositFee);
user.amount = user.amount.add(_amount).sub(depositFee);
}else{
user.amount = user.amount.add(_amount);
}
}
user.rewardDebt = user.amount.mul(pool.accPinPerShare).div(1e12);
emit Deposit(msg.sender, _pid, _amount);
}

// Withdraw LP tokens from MasterChef.
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "withdraw" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```

// Withdraw LP tokens from MasterChef.
function withdraw(uint256 _pid, uint256 _amount) public {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
require(user.amount >= _amount, "withdraw: not good");
updatePool(_pid);
uint256 pending = user.amount.mul(pool.accPinPerShare).div(1e12).sub(user.rewardDebt);
if(pending > 0) {
safePinTransfer(msg.sender, pending);
}
if(_amount > 0) {
user.amount = user.amount.sub(_amount);
pool.lpToken.safeTransfer(address(msg.sender), _amount);
}
user.rewardDebt = user.amount.mul(pool.accPinPerShare).div(1e12);
emit Withdraw(msg.sender, _pid, _amount);
}

// Withdraw without caring about rewards. EMERGENCY ONLY.
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "emergencyWithdraw" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/contracts/MasterChef.sol`

Locations

```

// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
uint256 amount = user.amount;
user.amount = 0;
user.rewardDebt = 0;
pool.lpToken.safeTransfer(address(msg.sender), amount);
emit EmergencyWithdraw(msg.sender, _pid, amount);
}

// Safe pin transfer function, just in case if rounding error causes pool to not have enough PINs.
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "dev" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

browser/contracts/MasterChef.sol

Locations

```
// Update dev address by the previous dev.
function dev(address _devaddr) public {
require(msg.sender == devaddr, "dev: wut?");
devaddr = _devaddr;
}

function setFeeAddress(address _feeAddress) public{
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "setFeeAddress" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

browser/contracts/MasterChef.sol

Locations

```
}

function setFeeAddress(address _feeAddress) public{
require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
feeAddress = _feeAddress;
}

//Pancake has to add hidden dummy pools inorder to alter the emission, here we make it simple and transparent to all.
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "updateEmissionRate" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

browser/contracts/MasterChef.sol

Locations

```

//Pancake has to add hidden dummy pools inorder to alter the emission, here we make it simple and transparent to all.
function updateEmissionRate(uint256 _pinPerBlock) public onlyOwner {
massUpdatePools();
pinPerBlock = _pinPerBlock;
}
}
```

MEDIUM

SWC-128

Loop over unbounded data structure.

Gas consumption in function "massUpdatePools" in contract "MasterChef" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
function massUpdatePools() public {
uint256 length = poolInfo.length;
for (uint256 pid = 0; pid < length; ++pid) {
updatePool(pid);
}
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/**
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.6.4"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
// File: contracts/libs/IBEP20.sol

pragma solidity >=0.6.4;

interface IBEP20 {
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.6.2<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`browser/contracts/MasterChef.sol`

Locations

```


pragma solidity >=0.6.2 <0.8.0;

/**
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`browser/contracts/MasterChef.sol`

Locations

```


pragma solidity >=0.6.0 <0.8.0;

```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`browser/contracts/MasterChef.sol`

Locations

```


pragma solidity >=0.6.0 <0.8.0;

/*
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`browser/contracts/MasterChef.sol`

Locations

```


pragma solidity >=0.6.0 <0.8.0;

/**
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is "">=0.4.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`browser/contracts/MasterChef.sol`

Locations

```


pragma solidity >=0.4.0;

```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
returns (uint256)
{
require(blockNumber < block.number, "PIN::getPriorVotes: not yet determined");

uint32 nCheckpoints = numCheckpoints[account];
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
internal
{
uint32 blockNumber = safe32(block.number, "PIN::_writeCheckpoint: block number exceeds 32 bits");

if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
massUpdatePools();
}
uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
totalAllocPoint = totalAllocPoint.add(_allocPoint);
poolInfo.push(PoolInfo({
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
massUpdatePools();
}
uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
totalAllocPoint = totalAllocPoint.add(_allocPoint);
poolInfo.push(PoolInfo({
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
uint256 accPinPerShare = pool.accPinPerShare;
uint256 lpSupply = pool.lpToken.balanceOf(address(this));
if (block.number > pool.lastRewardBlock && lpSupply != 0) {
uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
uint256 pinReward = multiplier.mul(pinPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
uint256 lpSupply = pool.lpToken.balanceOf(address(this));
if (block.number > pool.lastRewardBlock && lpSupply != 0) {
uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
uint256 pinReward = multiplier.mul(pinPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
accPinPerShare = accPinPerShare.add(pinReward.mul(1e12).div(lpSupply));
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
function updatePool(uint256 _pid) public {
PoolInfo storage pool = poolInfo[_pid];
if (block.number <= pool.lastRewardBlock) {
return;
}
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
uint256 lpSupply = pool.lpToken.balanceOf(address(this));
if (lpSupply == 0 || pool.allocPoint == 0) {
pool.lastRewardBlock = block.number;
return;
}
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
return;
}
uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
uint256 pinReward = multiplier.mul(pinPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
pin.mint(devaddr, pinReward.div(10));
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
pin.mint(address(this), pinReward);
pool.accPinPerShare = pool.accPinPerShare.add(pinReward.mul(1e12).div(lpSupply));
pool.lastRewardBlock = block.number;
}
```

LOW

SWC-128

Potentially unbounded data structure passed to builtin.

Gas consumption in function "delegateBySig" in contract "PineappleToken" depends on the size of data structures that may grow unboundedly. Specifically the "1-st" argument to builtin "keccak256" may be able to grow unboundedly causing the builtin to consume more gas than the block gas limit, effectively causing a denial-of-service condition.Consider that an attacker might attempt to cause this condition on purpose.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
abi.encode(
DOMAIN_TYPEHASH,
keccak256(bytes(name())),
getChainId(),
address(this)
```

LOW

SWC-128

Loop over unbounded data structure.

Gas consumption in function "getPriorVotes" in contract "PineappleToken" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

`browser/contracts/MasterChef.sol`

Locations

```
uint32 lower = 0;
uint32 upper = nCheckpoints - 1;
while (upper > lower) {
uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
Checkpoint memory cp = checkpoints[account][center];
```