MythX

REPORT 6054D3D3846CDD00182B6D40

Created	Fri Mar 19 2021 16:39:47 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	josephwharold@protonmail.com

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
abe34b65-e803-46b5-a218-5428f807542e	browser/contracts/MasterSamurai.sol	52

Analysis abe34b65-e803-46b5-a218-5428f807542e MythX

Started	Fri Mar 19 2021 16:39:50 GMT+0000 (Coordinated Universal Time)
Finished	Fri Mar 19 2021 17:25:13 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Remythx
Main Source File	Browser/Contracts/MasterSamurai.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	26	26

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/MasterSamurai.sol

Locations

```
621 * thereby removing any functionality that is only available to the owner.
622 */
623 function renounceOwnership() public onlyOwner {
624 emit OwnershipTransferred(_owner, address(0));
625 _owner = address(0);
626 }
627
628 /**
```

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/MasterSamurai.sol

Locations

```
630 * Can only be called by the current owner.
631 */
632 function transferOwnership(address newOwner) public onlyOwner {
633 _transferOwnership(newOwner);
634 }
635
636 /**
```

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/MasterSamurai.sol

Locations

```
722 * @dev Returns the token decimals.
723 */
724 function decimals() public override view returns (uint8) {
725 return _decimals;
726 }
727
728 /**
```

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/MasterSamurai.sol

Locations

```
729 * @dev Returns the token symbol.
730 */
731 function symbol() public override view returns (string memory) {
732 return _symbol;
733 }
734
735 /**
```

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/MasterSamurai.sol

Locations

```
736 * @dev See {BEP20-totalSupply}.
737 */
738 function totalSupply() public override view returns (uint256) {
739 return _totalSupply;
740 }
741
742 /**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "burnSupply" 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/MasterSamurai.sol

Locations

```
743 * @dev See {BEP20-totalSupply}.
744 */
745 function burnSupply() public view returns (uint256) {
746 return _burnSupply;
747 }
748
749 /**
```

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/MasterSamurai.sol

Locations

```
762 * - the caller must have a balance of at least `amount`.
763 */
764 function transfer(address recipient, uint256 amount) public override returns (bool) {
765 _transfer(_msgSender(), recipient, amount);
766 return true;
767 }
768
769 /**
```

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/MasterSamurai.sol

Locations

```
770 * @dev See {BEP20-allowance}.
771 */
772 function allowance(address owner, address spender) public override view returns (uint256) {
773 return _allowances[owner][spender];
774 }
775
776 /**
```

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/MasterSamurai.sol

Locations

```
781 * - `spender` cannot be the zero address.
782 */
783 function approve(address spender, uint256 amount) public override returns (bool) {
784 _approve(_msgSender(), spender, amount);
785 return true;
786 }
787
788 /**
```

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/MasterSamurai.sol

Locations

```
798 * `amount`.
799 */
800 function transferFrom(
801 address sender,
802 address recipient,
803 uint256 amount
804 ) public override returns (bool) {
805 _transfer(sender, recipient, amount);
806 _approve(
807 sender,
808 _msgSender(),
809 _allowances[sender][_msgSender()].sub(amount, 'BEP20: transfer amount exceeds allowance')
810 );
811 return true;
812 }
813
814 /**
```

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/MasterSamurai.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/MasterSamurai.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/MasterSamurai.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/MasterSamurai.sol

Locations

```
981
982 /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterSamurai).
983 function mint(address _to, uint256 _amount) public onlyOwner {
984 _mint(_to, _amount);
985 _moveDelegates(address(0), _delegates[_to], _amount);
986 }
987
988 /// @dev overrides transfer function to meet tokenomics of SMR
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "updateMultiplierAtDate" 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/MasterSamurai.sol

Locations

```
1367 }
1368
1369 function updateMultiplierAtDate(uint256 multiplierNumber, uint256 date) public onlyOwner {
1370 require(date * 1 seconds > now, "Cannot update multiplier for passed date");
1371 // nb block before multiplier update (delayBlock) = (updateStartDate - now) / 3 seconds
1372 // multiplier update start at block = delayBlock + actual block number
1373 uint256 startUpdateAtBlock = (date * 1 seconds - now) / (3 * 1 seconds) + block.number;
1374 multiplierBlockUpdate.push(Multiplier(multiplierNumber, startUpdateAtBlock));
1375 }
1376
1377 // Return reward multiplier over the given _from to _to block.
```

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/MasterSamurai.sol

Locations

```
1439 // Add a new lp to the pool. Can only be called by the owner.
1440 // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
1441 function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner {
1442 require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
1443 if (_withUpdate) {
1444 massUpdatePools();
1445 }
1446 checkPoolDuplicate(_lpToken);
1447 uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1448 totalAllocPoint = totalAllocPoint.add(_allocPoint);
1449 poolInfo.push(PoolInfo({
1450 lpToken: _lpToken,
1451 allocPoint: _allocPoint,
1452 lastRewardBlock: lastRewardBlock,
1453 accSmrPerShare: 0,
1454 depositFeeBP : _depositFeeBP
1455 }));
1456 }
1457
1458 // Update the given pool's SMR allocation point. 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/MasterSamurai.sol

Locations

```
1457
1458 // Update the given pool's SMR allocation point. Can only be called by the owner.
1459 function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner {
1460 if (_withUpdate) {
1461 massUpdatePools();
1462 }
1463 uint256 prevAllocPoint = poolInfo[_pid].allocPoint;
1464 poolInfo[_pid].allocPoint = _allocPoint;
1465 poolInfo[_pid].depositFeeBP = _depositFeeBP;
1466 if (prevAllocPoint != _allocPoint) {
1467 totalAllocPoint = totalAllocPoint.sub(prevAllocPoint).add(_allocPoint);
1468 }
1469 }
1470
1471 // View function to see pending SMRs on frontend.
```

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/MasterSamurai.sol

Locations

```

// Deposit LP tokens to MasterSamurai for SMR allocation.
function deposit(uint256 _pid, uint256 _amount) public validatePool(_pid) {
PoolInfo storage pool = poolInfo[_pid];
UserInfo storage user = userInfo[_pid][msg.sender];
updatePool(_pid);
if (user.amount > 0) {
uint256 pending = user.amount.mul(pool.accSmrPerShare).div(1e12).sub(user.rewardDebt);
if(pending > 0) {
safeSmrTransfer(msg.sender, pending);
}
}
if (_amount > 0) {
pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
if (address(pool.lpToken) == address(smr)) {
uint256 transferTax = _amount.mul(2).div(100);
_amount = _amount.sub(transferTax);
}
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.accSmrPerShare).div(1e12);
emit Deposit(msg.sender, _pid, _amount);
}

// Withdraw LP tokens from MasterSamurai.
```

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/MasterSamurai.sol

Locations

```

// Withdraw LP tokens from MasterSamurai.
function withdraw(uint256 _pid, uint256 _amount) public validatePool(_pid) {
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.accSmrPerShare).div(1e12).sub(user.rewardDebt);
if(pending > 0) {
safeSmrTransfer(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.accSmrPerShare).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/MasterSamurai.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];
pool.lpToken.safeTransfer(address(msg.sender), user.amount);
emit EmergencyWithdraw(msg.sender, _pid, user.amount);
user.amount = 0;
user.rewardDebt = 0;
}

function getPoolInfo(uint256 _pid) public view
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "getPoolInfo" 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/MasterSamurai.sol

Locations

```
1572 }
1573
1574 function getPoolInfo(uint256 _pid) public view
1575 returns(address lpToken, uint256 allocPoint, uint256 lastRewardBlock, uint256 accSmrPerShare, uint16 depositFeeBP) {
1576 return (address(poolInfo[_pid].lpToken),
1577 poolInfo[_pid].allocPoint,
1578 poolInfo[_pid].lastRewardBlock,
1579 poolInfo[_pid].accSmrPerShare,
1580 poolInfo[_pid].depositFeeBP);
1581 }
1582
1583 // Safe smr transfer function, just in case if rounding error causes pool to not have enough SMRs.
```

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/MasterSamurai.sol

Locations

```
1594
1595 // Update dev address by the previous dev.
1596 function dev(address _devaddr) public {
1597 require(msg.sender == devaddr, "dev: wut?");
1598 devaddr = _devaddr;
1599 }
1600
1601 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/MasterSamurai.sol

Locations

```
1599 }
1600
1601 function setFeeAddress(address _feeAddress) public {
1602 require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
1603 feeAddress = _feeAddress;
1604 }
1605
1606 function setStartRewardsDate(uint256 _startRewardsDate) public onlyOwner{
```

MEDIUM

SWC-128

Loop over unbounded data structure.

Gas consumption in function "updateEmissionRate" in contract "MasterSamurai" 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/MasterSamurai.sol`

Locations

```
1410
1411 uint256 newEmissionRate = smrPerBlock;
1412 for (uint256 index = lastReductionPeriodIndex; index < currentIndex; ++index) {
1413 newEmissionRate = newEmissionRate.mul(1e4 - EMISSION_REDUCTION_RATE_PER_PERIOD).div(1e4);
1414 }
```

MEDIUM

SWC-128

Loop over unbounded data structure.

Gas consumption in function "checkPoolDuplicate" in contract "MasterSamurai" 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/MasterSamurai.sol`

Locations

```
1432 function checkPoolDuplicate(IBEP20 _lpToken) public view {
1433 uint256 length = poolInfo.length;
1434 for (uint256 _pid = 0; _pid < length; _pid++) {
1435 require(poolInfo[_pid].lpToken != _lpToken, "add: existing pool");
1436 }
```

MEDIUM

SWC-128

Loop over unbounded data structure.

Gas consumption in function "massUpdatePools" in contract "MasterSamurai" 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/MasterSamurai.sol`

Locations

```
1486 function massUpdatePools() public {
1487 uint256 length = poolInfo.length;
1488 for (uint256 pid = 0; pid < length; ++pid) {
1489 updatePool(pid);
1490 }
```

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/MasterSamurai.sol`

Locations

```
1 pragma solidity >=0.4.0;
2
3 /**
```

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/MasterSamurai.sol`

Locations

```
187 }
188
189 pragma solidity >=0.4.0;
190
191 interface IBEP20 {
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.2"". 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/MasterSamurai.sol`

Locations

```
284 }
285
286 pragma solidity ^0.6.2;
287
288 /**
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.0"". 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/MasterSamurai.sol`

Locations

```
444 }
445
446 pragma solidity ^0.6.0;
447
```

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/MasterSamurai.sol`

Locations

```
543 }
544
545 pragma solidity >=0.4.0;
546
547 /*
```

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/MasterSamurai.sol`

Locations

```
570 }
571
572 pragma solidity >=0.4.0;
573
```

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/MasterSamurai.sol`

Locations

```
644 }
645
646 pragma solidity >=0.4.0;
647
```

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/MasterSamurai.sol`

Locations

```
1138 returns (uint256)
1139 {
1140 require(blockNumber < block.number, "SMR::getPriorVotes: not yet determined");
1141
1142 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/MasterSamurai.sol`

Locations

```
1211 internal
1212 {
1213 uint32 blockNumber = safe32(block.number, "SMR::_writeCheckpoint: block number exceeds 32 bits");
1214
1215 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/MasterSamurai.sol`

Locations

```
1371 // nb block before multiplier update (delayBlock) = (updateStartDate - now) / 3 seconds
1372 // multiplier update start at block = delayBlock + actual block number
1373 uint256 startUpdateAtBlock = (date * 1 seconds - now) / (3 * 1 seconds) + block.number;
1374 multiplierBlockUpdate.push(Multiplier(multiplierNumber, startUpdateAtBlock));
1375 }
```

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/MasterSamurai.sol`

Locations

```
1388 function checkForMultiplierUpdate() internal{
1389 if (multiplierBlockUpdate.length > lastMultiplierUpdateIndex + 1){
1390 if (block.number >= multiplierBlockUpdate[lastMultiplierUpdateIndex + 1].startBlock){
1391 lastMultiplierUpdateIndex = lastMultiplierUpdateIndex +1;
1392 actualMultiplier = multiplierBlockUpdate[lastMultiplierUpdateIndex];
```

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/MasterSamurai.sol`

Locations

```
// Reduce emission rate by 3% every 9,600 blocks ~ 8hours
function updateEmissionRate() internal {
if(startBlock > 0 && block.number <= startBlock){
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/MasterSamurai.sol`

Locations

```
}

uint256 currentIndex = block.number.sub(startBlock).div(EMISSION_REDUCTION_PERIOD_BLOCKS);
if (currentIndex <= lastReductionPeriodIndex) {
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/MasterSamurai.sol`

Locations

```
}
checkPoolDuplicate(_lpToken);
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/MasterSamurai.sol`

Locations

```
1445 }
1446 checkPoolDuplicate(_lpToken);
1447 uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1448 totalAllocPoint = totalAllocPoint.add(_allocPoint);
1449 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/MasterSamurai.sol`

Locations

```
1475 uint256 accSmrPerShare = pool.accSmrPerShare;
1476 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
1477 if (block.number > pool.lastRewardBlock && lpSupply != 0) {
1478 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
1479 uint256 smrReward = multiplier.mul(smrPerBlock).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/MasterSamurai.sol`

Locations

```
1476 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
1477 if (block.number > pool.lastRewardBlock && lpSupply != 0) {
1478 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
1479 uint256 smrReward = multiplier.mul(smrPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
1480 accSmrPerShare = accSmrPerShare.add(smrReward.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/MasterSamurai.sol`

Locations

```
1498
1499 PoolInfo storage pool = poolInfo[_pid];
1500 if (block.number <= pool.lastRewardBlock) {
1501 return;
1502 }
```

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/MasterSamurai.sol`

Locations

```
1503 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
1504 if (lpSupply == 0) {
1505 pool.lastRewardBlock = block.number;
1506 return;
1507 }
```

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/MasterSamurai.sol`

Locations

```
1506 return;
1507 }
1508 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
1509 uint256 smrReward = multiplier.mul(smrPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
1510 smr.mint(devaddr, smrReward.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/MasterSamurai.sol`

Locations

```
1511 smr.mint(address(this), smrReward);
1512 pool.accSmrPerShare = pool.accSmrPerShare.add(smrReward.mul(1e12).div(lpSupply));
1513 pool.lastRewardBlock = block.number;
1514 }
```

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/MasterSamurai.sol`

Locations

```
1605
1606 function setStartRewardsDate(uint256 _startRewardsDate) public onlyOwner{
1607 require(startBlock == 0 || startBlock < block.number, "rewardsStartDate passed !");
1608 require(_startRewardsDate * 1 seconds >= now, "");
1609 startBlock = block.number.add((_startRewardsDate * 1 seconds - now).div(3 * 1 seconds));
```

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/MasterSamurai.sol`

Locations

```
1607 require(startBlock == 0 || startBlock < block.number, "rewardsStartDate passed !");
1608 require(_startRewardsDate * 1 seconds >= now, "");
1609 startBlock = block.number.add((_startRewardsDate * 1 seconds - now).div(3 * 1 seconds));
1610 }
1611 }
```

LOW

SWC-128

Loop over unbounded data structure.

Gas consumption in function "sqrt" in contract "SafeMath" 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/MasterSamurai.sol`

Locations

```
z = y;
uint256 x = y / 2 + 1;
while (x < z) {
z = x;
x = (y / x + x) / 2;
```

LOW

SWC-128

Potentially unbounded data structure passed to builtin.

Gas consumption in function "delegateBySig" in contract "SmrToken" 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/MasterSamurai.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 "SmrToken" 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/MasterSamurai.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];
```