Competitive Security Assessment

MagpieBurn

Nov 22nd, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

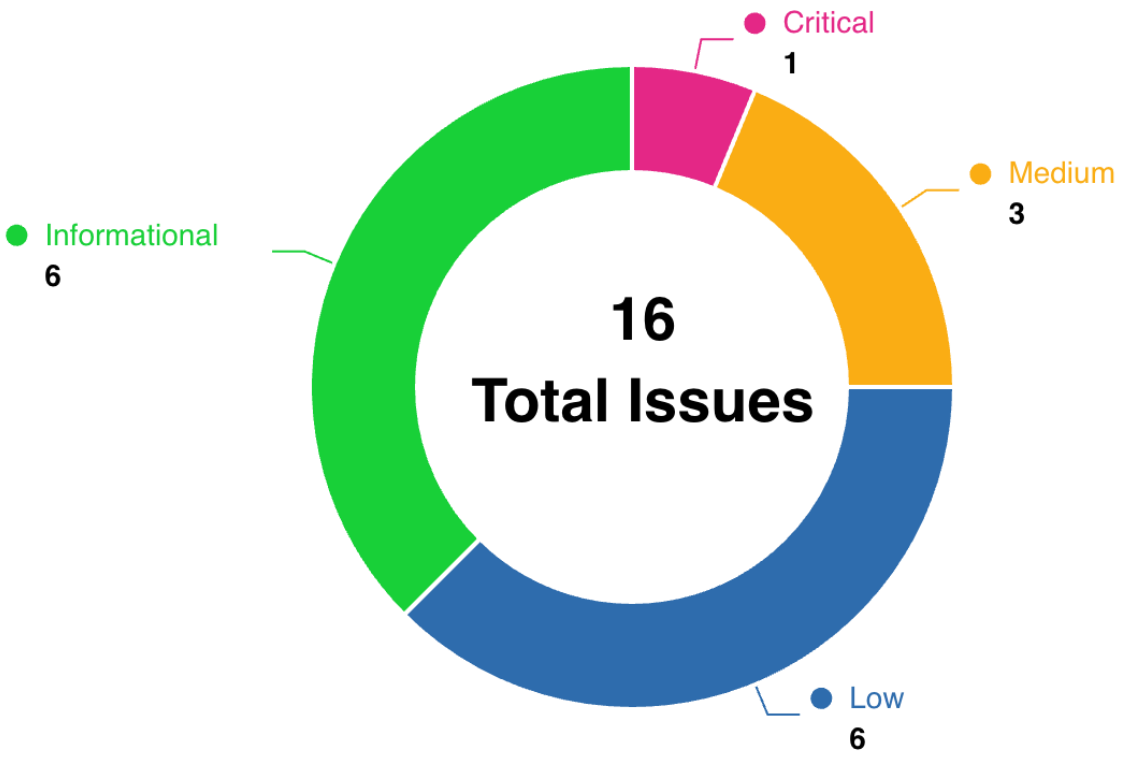
Project Detail

Project Name	MagpieBurn
Platform & Language	Solidity
Codebase	• https://github.com/magpiexyz/magpie_contracts • audit commit - 3e210a9af6a470741e8a9eba775b689475fa357a • final commit - 45890aedbd98af6f9febb55a8f7434abfd35b632
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
./contracts/VLMGP.sol	876280f333570062ec151620db4a7fb4ea385c8646d8d87fc4026877616f491c
./contracts/MGPBurnEventManager.sol	c821179770db7df240c7ed5f54c55c90dd5109ae8912a515de6ee72d88a9128e

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
MAB-1	The burn mechanism implemented via `joinEvent()` and `joinEventFor()` does not burn the mgp tokens	Logical	Critical	Acknowledged	0xzoobi
MAB-2	Lack of Storage Gap in Upgradeable Contracts	DOS	Medium	Fixed	0xzoobi, ginlee
MAB-3	Transfer whitelist not enforced	DOS	Medium	Acknowledged	SerSomeone

MAB-4	Hook `_beforeTokenTransfer` would revert token transfers regardless of `transferWhitelist` status.	Logical	Medium	Acknowledged	n16h7m4r3
MAB-5	Deprecated `safeApprove()` function in `burnVlmgp()` function	Language Specific	Low	Fixed	0xzoobi, ginlee
MAB-6	Missing zero address checks	Logical	Low	Fixed	ginlee, n16h7m4r3
MAB-7	`OwnableUpgradeable`: Does not implement 2-Step-Process for transferring ownership	Logical	Low	Acknowledged	0xzoobi
MAB-8	Outdated OpenZeppelin Dependencies Leading to Function Selector Clash in TransparentUpgradeableProxy	DOS	Low	Acknowledged	0xzoobi
MAB-9	MGP token holders can access `joinEvent()` and `joinEventFor()` even when the `VLMGP.sol` is paused leading to Denial of Service(DoS) for VLMGP holders	DOS	Low	Fixed	0xzoobi
MAB-10	Missing zero amount check	Code Style	Low	Fixed	n16h7m4r3
MAB-11	There is a risk of centralization of address `blackHole`	Privilege Related	Informational	Fixed	0xac
MAB-12	A appropriate way to judge the size of `uint256` type data	Code Style	Informational	Fixed	0xac
MAB-13	`deActivateEvent` can be called in same block as joinEvent	Logical	Informational	Acknowledged	SerSomeone
MAB-14	The maximum value of `coolDownInSecs` has not been limited	Integer Overflow and Underflow	Informational	Fixed	0xac
MAB-15	`joinEvent` and `joinEventFor` can be called multiple times with `0` amount	DOS	Informational	Acknowledged	SerSomeone
MAB-16	Constructor in the contracts and functions do not emit events.	Code Style	Informational	Fixed	n16h7m4r3

MAB-1:The burn mechanism implemented via `joinEvent()` and `joinEventFor()` does not burn the mgp tokens

Category	Severity	Client Response	Contributor
Logical	Critical	Acknowledged	0xzoobi

Code Reference

- code/contracts/MGPBurnEventManager.sol#L73
- code/contracts/MGPBurnEventManager.sol#L88

```
73:IERC20(mgp).safeTransferFrom(msg.sender, blackHole, _mgpBurnAmount);

88:IERC20(mgp).safeTransferFrom(msg.sender, blackHole, _mgpBurnAmount);
```

Description

0xzoobi : The current implementation of the `joinEvent()` and `joinEventFor()` function transfers tokens to a `blackhole` address (assuming `address(0)` or could be any other address), simulating token burning. However, this approach might create a false sense of burning, as it does not reflect in the `totalSupply()` of the token. This could lead to a discrepancy between the reported total supply and the actual circulating supply of the token. This might affect tokenomics, transparency, and the perception of the project's economic health.

Proof of Concept

Place the test case in `MgpBurnEventManager.spec.ts`

```
        it("totalSupply() of mgp tokens is not updated post burn via join event", async () => {
            await mgpBurnEventManager.startNewEvent("Burn MGP");
            let eventData = await mgpBurnEventManager.eventInfos(1);

            expect(await mgp.balanceOf(player1.address)).to.eq(ether(10000))

            const totalSupplyBeforeBurn = await mgp.totalSupply();

            //player1 burns their mgp tokens
            await mgp.connect(player1.wallet).approve(mgpBurnEventManager.address, mgpBurnAmount);
            await mgpBurnEventManager.connect(player1.wallet).joinEvent(1, mgpBurnAmount);

            const totalSupplyAfterBurn = await mgp.totalSupply();

            // Total Supplt post burn is not updated
            expect(totalSupplyBeforeBurn).to.eq(totalSupplyAfterBurn);
        });
```

Recommendation

0xzoobi : To address this issue, a potential solution involves modifying the ERC-20 token contract to incorporate a burn function, facilitating the permanent removal of tokens from circulation. This solution necessitates the implementation of proper access control measures and checks for the caller's authorization.

```
function burn(uint256 amount) external {
    require(msg.sender == address(MGPBurnEventManager), 'access control issue');
    _burn(msg.sender, amount);
}
```

And modify the `joinEvent()` and `joinEventFor()` to Use Token Burn Function:

The Fix

```
function joinEvent(uint256 _eventId, uint256 _mgpBurnAmount) external whenNotPaused nonReentrant
    {
        EventInfo storage eventInfo = eventInfos[_eventId];

        if(eventInfo.eventId == 0) revert eventNotExist();
        if(eventInfo.isActive == false) revert eventIsNotActive();

        eventInfo.totalMgpBurned += _mgpBurnAmount;
        userMgpBurnAmountForEvent[msg.sender][_eventId] += _mgpBurnAmount; // User burned MGP amount
for given event id

        -    IERC20(mgp).safeTransferFrom(msg.sender, blackHole, _mgpBurnAmount);
        +    IERC20(mgp).burn(_mgpBurnAmount);

        emit eventJoinedSuccesfully(msg.sender, _eventId, _mgpBurnAmount);
    }
```

Client Response

Acknowledged,We can’t add burn on MGP token since it’s deployed.

MAB-2:Lack of Storage Gap in Upgradeable Contracts

Category	Severity	Client Response	Contributor
DOS	Medium	Fixed	0xzoobi, ginlee

Code Reference

- code/contracts/MGPBurnEventManager.sol#L12
- code/contracts/VLMGP.sol#L28

```
12:contract MGPBurnEventManager is

28:contract VLMGP is IVLMGP, Initializable, ERC20Upgradeable, OwnableUpgradeable, PausableUpgradeabl
e, ReentrancyGuardUpgradeable {
```

Description

0xzoobi : The smart contracts `MGPBurnEventManager` and `VLMGP` which are upgradeable, lack storage gaps. It is considered a best practice in upgradeable contracts to include a state variable named `__gap`. This `__gap` state variable will be used as a reserved space for future upgrades. It allows adding new state variables freely in the future without compromising the storage compatibility with existing deployments.

The size of the `__gap` array is usually calculated so that the amount of storage used by a contract always adds up to the same number (usually 50 storage slots).

Reference: OpenZeppelin's storage gap -https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gap

ginlee : For upgradeable contracts, there must be storage gap to "allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployments". Otherwise it may be very difficult to write new implementation code. Without storage gap, the variable in child contract might be overwritten by the upgraded base contract if new variables are added to the base contract. This could have unintended and very serious consequences to the child contracts. Refer to the bottom part of this article: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable

Recommendation

0xzoobi : It is recommended to add a state variable named `__gap` as a reserved space for future upgrades in every upgradeable contract.

The Fix :

```
uint256[50] private __gap;
```

ginlee : Adding appropriate storage gap at the end of upgradeable contracts such as the below. Please reference OpenZeppelin upgradeable contract templates

```
uint256[50] private __gap;
```

Client Response

Fixed

MAB-3:Transfer whitelist not enforced

Category	Severity	Client Response	Contributor
DOS	Medium	Acknowledged	SerSomeone

Code Reference

- code/contracts/VLMGP.sol#L524

```
524:function _beforeTokenTransfer(
```

Description

SerSomeone : There is a function to toggle transfer whitelist for addresses by owner:

```
    function setWhitelistForTransfer(address _for, bool _status) external onlyOwner {
        transferWhitelist[_for] = _status;

        emit WhitelistSet(_for, _status);
    }
```

However, the whitelist is not enforced and currently and mint/burn/transfer will not be enabled.

```
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        revert TransferNotWhiteListed();
    }
```

Recommendation

SerSomeone : Add a check in `_beforeTokenTransfer` that validates the from and to are whitelisted

Client Response

Acknowledged,not fixed bcs removed _beforeTokenTransfer() hook as it's not used anywhere.

MAB-4:Hook `_beforeTokenTransfer` would revert token transfers regardless of `transferWhitelist` status.

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	n16h7m4r3

Code Reference

- code/contracts/VLMGP.sol#L418-L422
- code/contracts/VLMGP.sol#L524-L530

```
418:function setWhitelistForTransfer(address _for, bool _status) external onlyOwner {
419:        transferWhitelist[_for] = _status;
420:
421:        emit WhitelistSet(_for, _status);
422:    }

524:function _beforeTokenTransfer(
525:        address from,
526:        address to,
527:        uint256 amount
528:    ) internal virtual override {
529:        revert TransferNotWhiteListed();
530:    }
```

Description

n16h7m4r3 : The function `setWhitelistForTransfer()` allows owner to set `transferWhitelist` status for wallets to enable `vlMGP` token transfer for the respective wallet. The implemented `_beforeTokenTransfer` hook which is executed before any transfer would always revert token transfers regardless of `transferWhitelist` status.

Recommendation

n16h7m4r3 : Consider updating the `_beforeTokenTransfer` hook as following to allow token transfers of a wallet based on `transferWhitelist` status:

```
    function _beforeTokenTransfer(
          address from,
          address to,
          uint256 amount
      ) internal virtual override {
          if(!transferWhitelist[from]) {
              revert TransferNotWhiteListed();
          }
      }
```

Client Response

Acknowledged,not fixed bcs removed _beforeTokenTransfer() hook as it's not used anywhere.

MAB-5:Deprecated `safeApprove()` function in `burnVlmgp()` function

Category	Severity	Client Response	Contributor
Language Specific	Low	Fixed	0xzoobi, ginlee

Code Reference

- code/contracts/VLMGP.sol#L345

```
345:IERC20(MGP).safeApprove(burnEventManager, _vlmgpAmountToBurn);
```

Description

0xzoobi : The `safeApprove()` function prevents changing an allowance between non-zero values to mitigate a possible front-running attack. It reverts if that is the case. Instead, the `safeIncreaseAllowance` and `safeDecreaseAllowance` functions should be used. Comment from the OZ library for this function: "// `safeApprove()` should only be called when setting an initial allowance, // or when resetting it to zero. To increase and decrease it, use // 'safeIncreaseAllowance()' and 'safeDecreaseAllowance'()"

Impact: If the existing allowance is non-zero (say, for e.g., previously the entire balance was not deposited due to vault balance limit resulting in the allowance being reduced but not made 0), then `safeApprove()` will revert causing the user's token deposits to fail leading to denial-of-service. The condition predicate indicates that this scenario is possible.

A deeper discussion on the deprecation of this function is in OZ issue #2219. The OpenZeppelin ERC20 `safeApprove()` function has been deprecated, as seen in the comments of the OpenZeppelin code.

ginlee : openZeppelin safeApprove is deprecated, it is recommended not to use this method by openzeppelin

```
IERC20(MGP).safeApprove(burnEventManager, _vlmgpAmountToBurn);
```

check out original comments from openzeppelin https://github.com/OpenZeppelin/openzeppelin-contracts/blob/566a774222707e424896c0c390a84dc3c13bdcb2/contracts/token/ERC20/utils/SafeERC20.sol#L38

Recommendation

0xzoobi : As suggested by the OpenZeppelin comment, replace `safeApprove()` with `safeIncreaseAllowance()` and `safeDecreaseAllowance()`.

ginlee : Use approve from ERC20 instead of safeApprove

Client Response

Fixed

MAB-6:Missing zero address checks

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	ginlee, n16h7m4r3

Code Reference

- code/contracts/MGPBurnEventManager.sol#L55-L56
- code/contracts/MGPBurnEventManager.sol#L86
- code/contracts/VLMGP.sol#L108-L109

```
55:blackHole = _blackHole;
56:         mgp = _mgp;

86:userMgpBurnAmountForEvent[_user][_eventId] += _mgpBurnAmount; // User burned MGP amount for given
event id

108:masterMagpie = _masterMagpie;
109:         MGP = IERC20(_mgp);
```

Description

ginlee : No validation for 0 address in function joinEventFor contract MGPBurnEventManager which may lead to unexpected storage updates

n16h7m4r3 : Constructor is missing zero address checks where address is used as a parameter. In many of these instances, the functions do not validate that the passed address is not the address 0. While this does not currently pose a security risk, consider adding checks for the passed addresses being nonzero to prevent unexpected behavior

Recommendation

ginlee : add an input validation for _user

```
        if(_user == address(0)) revert addressNotExist();
```

n16h7m4r3 : Consider adding zero address checks.

Client Response

Fixed

MAB-7: `OwnableUpgradeable`: Does not implement 2-Step-Process for transferring ownership

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	0xzoobi

Code Reference

- code/contracts/MGPBurnEventManager.sol#L6
- code/contracts/VLMGP.sol#L7

```
6:import { OwnableUpgradeable } from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

7:import { OwnableUpgradeable } from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
```

Description

0xzoobi : The contracts `VLMGP.sol` and `MGPBurnEventManager.sol` does not implement a 2-Step-Process for transferring ownership. So ownership of the contract can easily be lost when making a mistake when transferring ownership.

Since the privileged roles have critical function roles assigned to them. Assigning the ownership to a wrong user can be disastrous. So Consider using the `Ownable2StepUpgradeable` contract from OZ (https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/Ownable2StepUpgradeable.sol) instead.

The way it works is there is a `transferOwnership()` to transfer the ownership and `acceptOwnership()` to accept the ownership. Refer the above `Ownable2StepUpgradeable.sol` for more details.

Recommendation

0xzoobi : Implement 2-Step-Process for transferring ownership via `Ownable2StepUpgradeable.sol`

Client Response

Acknowledged,No need now suggested by project owner

MAB-8:Outdated OpenZeppelin Dependencies Leading to Function Selector Clash in TransparentUpgradeableProxy

Category	Severity	Client Response	Contributor
DOS	Low	Acknowledged	0xzoobi

Code Reference

- code/package.json#L33-L34

```
33:"@openzeppelin/contracts": "4.7.3",
34:    "@openzeppelin/contracts-upgradeable": "4.7.3",
```

Description

0xzoobi : The current version of openzeppelin-contracts in use is `4.7.3`, while the latest version available is `5.0.0`.
To maintain the highest level of security, it is recommended to keep all libraries, including openzeppelin-contracts, up-to-date.
You can refer to security advisories for openzeppelin-contracts at: https://github.com/OpenZeppelin/openzeppelin-contracts/security
The issue marked with `CVE-2023-30541` can potentially lead to a `Denial of Service`. This can occur when a function in the implementation contract becomes inaccessible due to a clash between its selector and one of the proxy's own selectors. For more details, please visit: https://github.com/advisories/GHSA-mx2q-35m2-x2rh
To fix this issue and other associated issues related to openzeppelin-contracts, upgrade to the latest version.

Recommendation

0xzoobi : Update to the latest version of openzeppelin-contracts.
Reference - https://github.com/OpenZeppelin/openzeppelin-contracts/releases

Client Response

Acknowledged,After updating openZappelin dependency it's required add some other changes. so would like to still with current dependencies.

MAB-9:MGP token holders can access `joinEvent()` and `joinEventFor()` even when the `VLMGP.sol` is paused leading to Denial of Service(DoS) for VLMGP holders

Category	Severity	Client Response	Contributor
DOS	Low	Fixed	0xzoobi

Code Reference

- code/contracts/MGPBurnEventManager.sol#L63
- code/contracts/MGPBurnEventManager.sol#L78
- code/contracts/VLMGP.sol#L333

```
63:function joinEvent(uint256 _eventId, uint256 _mgpBurnAmount) external whenNotPaused nonReentrant

78:function joinEventFor(address _user, uint256 _eventId, uint256 _mgpBurnAmount) external whenNotPaused nonReentrant

333:function burnVlmgp( uint256 _vlmgpAmountToBurn, uint256 _vlmgpBurnEventId ) external whenNotPaused nonReentrant
```

Description

0xzoobi : The `VLMGP.sol` and `MGPBurnEventManager.sol` contracts both inherit from the OZ Pausable contract, which allows admins/owners to pause each contract in the event of an emergency. However, Imagine **only the `VLMGP.sol` contract** is paused due to an emergency or to fix an issue, As a result, users are unable to call the `burnVlmgp()` function, preventing them from burning their VLMGP tokens for MGP tokens to participate in events.

This situation creates an imbalance that enables a subset of users to exploit the system, potentially gaining unfair advantages and initiating a Denial of Service (DoS) attack on users who have staked in the `VLMGP` contract. This unequal treatment erodes the trust of MGP token stakers in the protocol, as they find themselves subjected to unfair circumstances despite being active participants in staking.

Recommendation

0xzoobi : The best solution to fix this issue is to synchronize the pause across all contracts. This can be done by creating a parent contract from which the Pausable state is read. For large projects, this approach also improves the user

experience (UX) for admins, as they can pause all contracts with a single click instead of calling the pause function on all the contracts.

Client Response

Fixed,Fixed by adding paused checks for mgpBurnEventManager when trigger burnVlmgp from VLMGP.

MAB-10:Missing zero amount check

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	n16h7m4r3

Code Reference

- code/contracts/MGPBurnEventManager.sol#L63-L76
- code/contracts/MGPBurnEventManager.sol#L78-L91

```
63:function joinEvent(uint256 _eventId, uint256 _mgpBurnAmount) external whenNotPaused nonReentrant
64:     {
65:         EventInfo storage eventInfo = eventInfos[_eventId];
66:
67:         if(eventInfo.eventId == 0) revert eventNotExist();
68:         if(eventInfo.isActive == false) revert eventIsNotActive();
69:
70:         eventInfo.totalMgpBurned += _mgpBurnAmount;
71:         userMgpBurnAmountForEvent[msg.sender][_eventId] += _mgpBurnAmount; // User burned MGP amo
unt for given event id
72:
73:         IERC20(mgp).safeTransferFrom(msg.sender, blackHole, _mgpBurnAmount);
74:
75:         emit eventJoinedSuccesfully(msg.sender, _eventId, _mgpBurnAmount);
76:     }

78:function joinEventFor(address _user, uint256 _eventId, uint256 _mgpBurnAmount) external whenNotPa
used nonReentrant
79:     {
80:         EventInfo storage eventInfo = eventInfos[_eventId];
81:
82:         if(eventInfo.eventId == 0) revert eventNotExist();
83:         if(eventInfo.isActive == false) revert eventIsNotActive();
84:
85:         eventInfo.totalMgpBurned += _mgpBurnAmount;
86:         userMgpBurnAmountForEvent[_user][_eventId] += _mgpBurnAmount; // User burned MGP amount f
or given event id
87:
88:         IERC20(mgp).safeTransferFrom(msg.sender, blackHole, _mgpBurnAmount);
89:
90:         emit eventJoinedSuccesfully(_user, _eventId, _mgpBurnAmount);
91:     }
```

Description

n16h7m4r3 : The functions `joinEvent()` and `joinEventFor()` allows zero token transfers for joining an event.

Recommendation

n16h7m4r3 : Add the following to the functions `joinEvent()` and `joinEventFor()` to avoid zero token transfers.

```
require(_mgpBurnAmount > 0);
```

Client Response

Fixed

MAB-11:There is a risk of centralization of address `blackHole`

Category	Severity	Client Response	Contributor
Privilege Related	Informational	Fixed	0xac

Code Reference

- code/contracts/MGPBurnEventManager.sol#L51-L57

```
51:function __MGPBurnEventManager_init(address _blackHole, address _mgp) public initializer {
52:         __Ownable_init();
53:         __ReentrancyGuard_init();
54:         __Pausable_init();
55:         blackHole = _blackHole;
56:         mgp = _mgp;
57:     }
```

Description

0xac : The `blackHole` address may be set to an account address controlled by the project party, which will allow this address to obtain all the tokens that should be burned.

```
    function __MGPBurnEventManager_init(address _blackHole, address _mgp) public initializer {
        __Ownable_init();
        __ReentrancyGuard_init();
        __Pausable_init();
        blackHole = _blackHole;
        mgp = _mgp;
    }
```

Recommendation

0xac : It is recommended to use hard coding to set this address.

```
address public constant blackHole = address(0); // or orther address
```

Client Response

Fixed

MAB-12:A appropriate way to judge the size of `uint256` type data

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	0xac

Code Reference

- code/contracts/VLMGP.sol#L105
- code/contracts/VLMGP.sol#L433

```
105:if (_maxSlots <= 0)

433:if(_coolDownSecs <= 0) revert InvalidCoolDownPeriod();
```

Description

0xac : Because the value of `uint256` parameter is no less than 0. This judgment statement is not rigorous.

```
if (_maxSlots <= 0)
    revert MaxSlotShouldNotZero();
```

Recommendation

0xac : It is recommended to change the judgment statement to the following form.

```
if (_maxSlots == 0)
    revert MaxSlotShouldNotZero();
```

Client Response

Fixed

MAB-13: `deActivateEvent` can be called in same block as joinEvent

Category	Severity	Client Response	Contributor
Logical	Informational	Acknowledged	SerSomeone

Code Reference

- code/contracts/MGPBurnEventManager.sol#L68

```
68:if(eventInfo.isActive == false) revert eventIsNotActive();
```

Description

SerSomeone : Since `deActivateEvent` can be called in same block as `joinEvent` users will not know that an event is deactivated and will essentially pay for a non existing event. This is not a user error because the user essentially joins an activated event, pays, then it gets deactivated in the same block.

Recommendation

SerSomeone : Make deactivating an event a two step process which have at least a block gap between them

Client Response

Acknowledged,Event can only deActivate by admin when decided to deActivate it. there is check's for event activation status if user tried to join is deActivated.

MAB-14:The maximum value of `coolDownInSecs` has not been limited

Category	Severity	Client Response	Contributor
Integer Overflow and Underflow	Informational	Fixed	0xac

Code Reference

- code/contracts/VLMGP.sol#L96-L111
- code/contracts/VLMGP.sol#L240-L254
- code/contracts/VLMGP.sol#L432-L437

```
96:function __vlMGP_init_(
97:        address _masterMagpie,
98:        uint256 _maxSlots,
99:        address _mgp,
100:        uint256 _coolDownInSecs
101:    ) public initializer {
102:        __Ownable_init();
103:        __Pausable_init();
104:        __ERC20_init("Vote Locked MGP", "vlMGP");
105:        if (_maxSlots <= 0)
106:            revert MaxSlotShouldNotZero();
107:        maxSlot = _maxSlots;
108:        masterMagpie = _masterMagpie;
109:        MGP = IERC20(_mgp);
110:        coolDownInSecs = _coolDownInSecs;
111:    }

240:function expectedPenaltyAmount(uint256 _slotIndex) public view returns(uint256 penaltyAmount, ui
nt256 amontToUser) {
241:        UserUnlocking storage slot = userUnlockings[msg.sender][_slotIndex];
242:
243:        uint256 coolDownAmount = slot.amountInCoolDown;
244:        uint256 baseAmountToUser = slot.amountInCoolDown / 5;
245:        uint256 waitingAmount = coolDownAmount - baseAmountToUser;
246:
247:        uint256 unlockFactor = 1e12;
248:        if((block.timestamp - slot.startTime) <= (slot.endTime - slot.startTime))
249:            unlockFactor = ((block.timestamp - slot.startTime) * 1e12 / (slot.endTime - slot.sta
rtTime)) ** 2 / 1e12;
250:
251:        uint256 unlockAmount = waitingAmount * unlockFactor / 1e12;
252:        amontToUser = baseAmountToUser + unlockAmount;
253:        penaltyAmount = coolDownAmount - amontToUser;
254:    }

432:function setCoolDownInSecs(uint256 _coolDownSecs) external onlyOwner {
433:        if(_coolDownSecs <= 0) revert InvalidCoolDownPeriod();
434:        coolDownInSecs = _coolDownSecs;
435:
436:        emit CoolDownInSecsUpdated(_coolDownSecs);
437:    }
```

Description

0xac : The value of `coolDownInSecs` would be set by `VLMGP.__vlMGP_init_()` and `VLMGP.setCoolDownInSecs()` functions. However, these functions do not limit the maximum value of `coolDownInSecs`. If the value of `coolDownInSecs` is greater than 2**128, an overflow problem would raise in the `VLMGP.expectedPenaltyAmount()` function while caculating the `(slot.endTime - slot.startTime)) ** 2`.

```
    function expectedPenaltyAmount(uint256 _slotIndex) public view returns(uint256 penaltyAmount, ui
nt256 amontToUser) {
        UserUnlocking storage slot = userUnlockings[msg.sender][_slotIndex];

        uint256 coolDownAmount = slot.amountInCoolDown;
        uint256 baseAmountToUser = slot.amountInCoolDown / 5;
        uint256 waitingAmount = coolDownAmount - baseAmountToUser;

        uint256 unlockFactor = 1e12;
        if((block.timestamp - slot.startTime) <= (slot.endTime - slot.startTime))
            unlockFactor = ((block.timestamp - slot.startTime) * 1e12 / (slot.endTime - slot.startTi
me)) ** 2 / 1e12;

        uint256 unlockAmount = waitingAmount * unlockFactor / 1e12;
        amontToUser = baseAmountToUser + unlockAmount;
        penaltyAmount = coolDownAmount - amontToUser;
    }
```

Recommendation

0xac : It is recommended to add a limit for setting the value of `coolDownInSecs`.

```
    function __vlMGP_init_(
        address _masterMagpie,
        uint256 _maxSlots,
        address _mgp,
        uint256 _coolDownInSecs
    ) public initializer {
        __Ownable_init();
        __Pausable_init();
        __ERC20_init("Vote Locked MGP", "vlMGP");
        if (_maxSlots <= 0)
            revert MaxSlotShouldNotZero();
        maxSlot = _maxSlots;
        masterMagpie = _masterMagpie;
        MGP = IERC20(_mgp);

        if (_coolDownInSecs > 2**128)
            revert();
        coolDownInSecs = _coolDownInSecs;
    }



    function setCoolDownInSecs(uint256 _coolDownSecs) external onlyOwner {
        if(_coolDownSecs <= 0) revert InvalidCoolDownPeriod();
        if (_coolDownSecs > 2**128)
            revert();
        coolDownInSecs = _coolDownSecs;

        emit CoolDownInSecsUpdated(_coolDownSecs);
    }
```

Client Response

Fixed

MAB-15: `joinEvent` and `joinEventFor` can be called multiple times with `0` amount

Category	Severity	Client Response	Contributor
DOS	Informational	Acknowledged	SerSomeone

Code Reference

- code/contracts/MGPBurnEventManager.sol#L75

```
75:emit eventJoinedSuccesfully(msg.sender, _eventId, _mgpBurnAmount);
```

Description

SerSomeone : `joinEvent` and `joinEventFor` emit events stating that the caller joined an event.
However there is no check for the amount of tokens used to join the event. Therefore a malicious actor can call `joinEvent` multiple times even if they don't have any tokens in their balance which will emit a lot of events, especially on L2 chains (Arb is supported according to README) which gas is very cheap

Recommendation

SerSomeone : Put a minimum threshold for the amount of tokens transferred

Client Response

Acknowledged,There is check for not allowing user to join event with 0 amount.

MAB-16:Constructor in the contracts and functions do not emit events.

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	n16h7m4r3

Code Reference

- code/contracts/VLMGP.sol#L107-L110
- code/contracts/VLMGP.sol#L456-L466

```
107:maxSlot = _maxSlots;
108:        masterMagpie = _masterMagpie;
109:        MGP = IERC20(_mgp);
110:        coolDownInSecs = _coolDownInSecs;

456:function setWombatBribeManager(address _bribeManager) external onlyOwner {
457:        wombatBribeManager = _bribeManager;
458:    }
459:
460:    function setReferralStorage(address _referralStorage) external onlyOwner {
461:        referralStorage = _referralStorage;
462:    }
463:
464:     function setMgpBurnEventManager(address _burnEventMAnager) external onlyOwner {
465:        burnEventManager = _burnEventMAnager;
466:    }
```

Description

n16h7m4r3 : Events help external applications keep track of state changes of the contract, the external application would not register the state change when updated.

Recommendation

n16h7m4r3 : Consider emitting the corresponding events for the respective state changes.

Client Response

Fixed

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.