**JMDLabs Contracts Initial Report**

[**Scope of Audit**](#_ypywdf1ix69g)2

[**Check Vulnerabilities**](#_fk11hakqdckb)2

[**Techniques and Methods**](#_4f674zh1p4ou)3

[Issue Categories](#_nxpic713a3e6) 4

[Number of security issues per severity.](#_2idbboidjlj7) 5

[**Introduction**](#_aqhg5kqr9l1u)6

[**Issues Found – Code Review / Manual Testing**](#_jnyy7e310pds) **7**

[High Severity Issues](#_tif7pylvu8h3) 7

[H.1 Missing Method Definition](#_rqevufs9vhax) 7

Medium [Severity Issues](#_tif7pylvu8h3) 7

[M.1 Orchestrator can call refund any number of times 7](#_359d2wv1ti5a)

[M.2 Incorrect require check](#_fd6p89dn2qqt) 8

[Low Severity Issues](#_y65o5x8fwlqc) 8

[L.1 Hardcoded initialization](#_xg8p62iavmvd) 8

[L.2 Missing setter validation 9](#_2yhugd5su7em)

[L.3 Claim with amount zero 9](#_ab3yx6mfvopb)

[Informational Issues](#_b5bkvrn01dit) 10

[I.1 Unused variables 1](#_nhb4vqaltlj9)0

[I.2 Unused imports](#_jnmtaupqx04r) 10

[I.3 Methods can be made external](#_vwqcbkse3zjq) 11

[I.4 State variables that can be constants 1](#_rmc4bdwpt4v0)1

[I.5 Unnecessary require check 12](#_dy5hpsbovnb5)

[I.6 Redundant require check 12](#_1enr6l318a00)

[I.7 View function can be made pure](#_f4gpeasfwzag) 12

[I.8 Refacator methods 13](#_81mmv0hi0f56)

[I.9 Pragma unlocked 15](#_kg8kp6ca1cju)

[I.10 Missing Events](#_etdlf8gv4um) 15

[I.11 Incorrect Error Message 15](#_qodk2jgwmh0l)

[I.12 Best Practices 16](#_agwkvg8kwnsi)

[**Functional Tests 16**](#_v45y4w3h16jf)

[**Automated Tests 18**](#_x7qz3xds3x9t)

[**Results**](#_x8ufajnyxbkt) **22**

[**Closing Summary**](#_ol9yx1kh3q0b) **22**

## **Scope of Audit**

The scope of this audit was to analyze and document the JMDLabs smart contracts codebase for quality, security, and correctness.

## **Check Vulnerabilities**

* Re-entrancy
* Timestamp Dependence
* Gas Limit and Loops
* DoS with Block Gas Limit
* Transaction-Ordering Dependence
* Use of tx.origin
* Exception disorder
* Gasless send
* Balance equality
* Byte array
* Transfer forwards all gas
* ERC20 API violation
* Malicious libraries
* Compiler version not fixed
* Redundant fallback function
* Send instead of transfer
* Style guide violation
* Unchecked external call
* Unchecked math
* Unsafe type inference
* Implicit visibility level

# **Techniques and Methods**

Throughout the audit of smart contracts, care was taken to ensure:

* The overall quality of code.
* Use of best practices.
* Code documentation and comments match logic and expected behavior.
* Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
* Efficient use of gas.
* Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

**Structural Analysis**

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

**Static Analysis**

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

**Code Review / Manual Analysis**

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

**Gas Consumption**

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

**Tools and Platforms used for Audit**

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistical analysis, Theo.

## 

## **Issue Categories**

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

**High Severity Issues**

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract’s performance or functionality, and we recommend these issues be fixed before moving to a live environment.

**Medium Severity Issues**

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

**Low Severity Issues**

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

**Informational** **Issues**

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

## 

## **Number of security issues per severity.**

# 

| **TYPE** | **HIGH** | **MEDIUM** | **LOW** | **INFORMATIONAL** |
| --- | --- | --- | --- | --- |
| **Open** | **1** | **2** | **3** | **12** |
| **Acknowledged** | **0** | **0** | **0** | **0** |
| **Closed** | **0** | **0** | **0** | **0** |

# 

# **Introduction**

During the period of **May 15th, 2022 to May 19th, 2022**.

Manmeet performed a security audit for **JMDLabs** smart contracts.

## 

| **Version Number** | **Date** |
| --- | --- |
| 01 | May 15 - May 19 |

# **Issues Found – Code Review / Manual Testing**

## **High Severity Issues**

### H.1 Missing Method Definition

**Contract**: BAPOrchestrator.sol, BAPGenesis.sol

**Description:**

The refund method in Orchestrator tries to call a method ‘*bullsBreedingLeft’* on the contract BAPGenesis. But no such implementation exists on the BAPGenesis contracts hence the call to refund would break,

**Recommendation**:

Implement *bullsBreedingLeft* on BAPGenesis

**Status: Open**

## **Medium Severity Issues**

### M.1 Orchestrator can call refund any number of times

**Contract:** BAPGenesis.sol

**Description**: The check on call of refund is implemented within the orchestrator contract, but here in BAPGenesis we dont perform any kind of state update or validation on the orchestrator address, which allows address set as orchestrator address to call refund any number of times and transfer the same *balance* amount every time it is called. Another concern which increases the chance of that happening is not having any validation on the parameter in setOrchestrator which allows us to even set a user address to be an orchestrator.

| **Line** | **Code/Function** |
| --- | --- |
| 89 | function refund(address depositAddress, uint256 tokenId) |
| 365 | function setOrchestrator(address newOrchestrator) |

**Recommendation:**

Add a validation to check what address being set as Orchestrator or/and implement a state update in the refund logic to check if the refund has already been given, it can even be a call to another contract.

**Status: Open**

### M.2 Incorrect require check

**Contract:** BAPGenesis.sol

**Description**: The *verify* method in the contract returns true if the value of state variable *whitelisted* is false, but the require check which should ideally throw *"Not whitelisted"* in this case passes successfully to the next lines of the method *mint.*

| **Line** | **Code/Function** |
| --- | --- |
| 152 | require(  verify(signature, tokenAmount, tier, walletLimit),  "Not whitelisted"  ); |

**Recommendation:**

Either fix the logic in verify or modify the require check error message if this is an intended behaviour.

**Status: Open**

## **Low Severity Issues**

### L.1 Hardcoded initialization and parameters in method calls

**Contract**: BAPGenesis.sol, BAPMethane.sol, BAPUtilities.sol, BAPVesting.sol

**Description**

Several state variables in the above contracts have been initialized with a hardcoded value in the constructor and there are several internal method calls that use hardcoded parameters.  
The number of instances are too many list down.

**Recommendation**:

We should never initialize state variables with hardcoded value in constructor and definitely use them as internal parameters since it binds the code logic to certain values. We can either pass the values as parameter variables of define state constants which can be used if it needs to be a fixed value.  
  
**Status: Open**

### L.2 Missing setter validation for mintingMin

**Contract**: BAPMethane.sol

**Description**

The value of mintingMin can eve be set to zero as there are no validations for the value in the setter. A situation like this will allow a method like *claim* to execute even with value zero passed as tokenAmount.

| **Line** | **Code/Function** |
| --- | --- |
| 95 | function setMintingMin(uint256 min) public onlyOwner |

**Recommendation**:

Add validation to ensure *mintingMin* is never set to zero or update require checks in *claim* method to check value “>*mintingMin”* instead of “=>*mintingMin”*

**Status: Open**

### L.3 Claim with amount zero

**Contract**: BAPVesting.sol

**Description**

In the *vesting* method if meth amount fetched to be zero, the method still goes ahead to execute *claim* and update other state values with the value zero.

| **Line** | **Code/Function** |
| --- | --- |
| 86 | function vesting() public nonReentrant {  require(vestingWallets[msg.sender].start != 0, "200:UNREGISTERED");  uint256 methAmount = vestingAmount();  methContract.claim(msg.sender, methAmount);  vested[msg.sender] += methAmount;  totalVested += methAmount;  emissionLeft -= methAmount;  } |

**Recommendation**:

The *vesting* call should revert if methAmount is set to zero

**Status: Open**

## **Informational Issues**

### I.1 Unused Variables

**Contract**: BAPGenesis.sol, BAPTeenBulls.sol, BAPOrchestrator.sol, BAPVesting.sol

**Description:**

These contract define and set the given state variables but never use it.

| **Line** | **Code/Function** |
| --- | --- |
| BAPGenesis - 86 | uint256 public mintedAllowedCap = 8810; |
| BAPTeenBulls - 15 | mapping(uint256 => string) private \_tokenURIs; |
| BAPOrchestrator - 15 | address public bapGenesisAddr;  address public bapMethAddr;  address public bapUtilitiesAddr; |
| BAPVesting - 24 | address public methContractAdress; |
| BAPOrchestrator - 289, 313, 337 | SignatureClaimMethStruct memory payload |
| BAPOrchestrator - 161 | function generateTeenBull(bytes memory signature, uint256 **tokenId**) |
| BAPTeenBulls - 16 | address public secret; |

**Recommendation:**

Remove unused variables

**Status: Open**

### I.2 Unused Imports

**Contract:** BAPTeenBulls.sol, ERC721A.sol

**Description:**

The contract contains imports that are not used within the contract and make the contract heavy.

| **Line** | **Code/Function** |
| --- | --- |
| BAPTeenBulls - 5 | import "@openzeppelin/contracts/access/Ownable.sol"; |
| ERC721A - 10 | import '@openzeppelin/contracts/utils/Context.sol'; |

**Recommendation:**

Remove unused imports

**Status: Open**

### I.3 Methods can be made external

**Contract**: BAPGenesis.sol, BAPUtilities.sol

**Description:**

public functions that are never called by the contract should be declared external to save gas.

| **Line** | **Code/Function** |
| --- | --- |
| BAPGenesis - 256 | function contractURI() public |
| BAPGenesis - 260 | function tokenURI(uint256 tokenId) public |
| BAPUtilities - 37 | function purchaseIncubator() public |
| BAPUtilities - 45 | function purchaseMergerOrb() public |

**Recommendation:**

Use the external attribute for functions never called from the contract.

**Status: Open**

### I.4 State variables than can be constants Contract: BAPGenesis.sol, BAPOrchestrator.sol

**Description:** State variables are stored in the storage slots which where reading and writing are gas intensive operations while constants become a part of the contract bytecode and are cheaper to use.

| **Line** | **Code/Function** |
| --- | --- |
| BAPGenesis - 32 | uint256 public maxBreedings = 3; |
| BAPGenesis - 34 | uint256 public mintedAllowedCap = 8810; |
| BAPOrchestrator - 25 | uint256 timeCounter = 1 days; |

**Recommendation:**

We should make state variables whose values we are sure will never be changed should be defined as a constant.

**Status: Open**

### I.5 Unnecessary require check Contract: BAPMethane.sol

**Description:** *treasuryWallet* can never be set to zero since it is initialized in the constructor which will revert of zero address is passed and there is a zero address check in the setter

| **Line** | **Code/Function** |
| --- | --- |
| 99 | function setOpen(bool \_open) public onlyOwner {  require(treasuryWallet != address(0), "100:NOT\_READY");  open = \_open;  } |

**Recommendation:**

Remove unnecessary checks to save gas consumption.

**Status: Open**

### I.6 Redundant require check Contract: BAPMethane.sol

**Description:** *verifyOrigin* checks for if the caller is Orchestrator or vestingManager which is enough to forbid direct calls from players, but the method executes redundant check to see if *msg.sender == tx.origin.*

| **Line** | **Code/Function** |
| --- | --- |
| 86 | function verifyOrigin() internal view returns (bool) |

**Recommendation:**

Remove redundant checks to save gas consumption.

**Status: Open**

### I.7 View function can be made pure

**Contract**: BAPOrchestrator.sol

**Description:**

When a function is neither reading or writing from state of the contract it should be marked as pure.

| **Line** | **Code/Function** |
| --- | --- |
| 89 | function dailyRewards(bool godBull) |

**Recommendation:**

Label the *dailyRewards* function as pure instead of view

**Status: Open**

### I.8 Refactor methods

**Contract**: BAPGenesis.sol, BAPOrchestrator.sol

**Description:**

The following methods can be refactored to save gas, avoid unnecessary variable declarations and reduce contract size.

**Recommendation:**

| BAPGenesis - 300 (merge the internal and external methods and eliminate parameter) |
| --- |
| function \_refundPeriodAllowed(uint256 tokenId)  internal  view  returns (bool)  {  require(  block.timestamp >= genesisTimestamp + 31 days &&  block.timestamp <= genesisTimestamp + 180 days  );  return true;  }  function refundPeriodAllowed(uint256 tokenId) external view returns (bool) {  return \_refundPeriodAllowed(tokenId);  } |
| function refundPeriodAllowed()  external  view  returns (bool)  {  require(  block.timestamp >= genesisTimestamp + 31 days &&  block.timestamp <= genesisTimestamp + 180 days  );  return true;  } |

| BAPOrchestrator - |
| --- |
| function claimMeth(  bytes memory signature,  uint256 bullsCount,  uint256 godsCount,  uint256[] memory bulls,  uint256[] memory gods  ) external nonReentrant {  require(  verifyClaimMeth(signature, bullsCount, godsCount),  "Claim Meth Signature not valid"  );  require(bullsCount == bulls.length, "Invalid Bulls Array");  require(godsCount == gods.length, "Invalid Bulls Array");  uint256 amount = 0;  for (uint256 index = 0; index < bullsCount; index++) {  amount += claimRewardsFromToken(bulls[index], false);  }  for (uint256 index = 0; index < godsCount; index++) {  amount += claimRewardsFromToken(gods[index], true);  }  bapMeth.claim(\_msgSender(), amount);  } |
| function claimMeth(  bytes memory signature,  uint256[] memory bulls,  uint256[] memory gods  ) external nonReentrant {  uint256 bullsCount = bulls.length;  uint256 godsCount = gods.length;  require(  verifyClaimMeth(signature, bullsCount, godsCount),  "Claim Meth Signature not valid"  );  uint256 amount = 0;  for (uint256 index = 0; index < bullsCount; index++) {  amount += claimRewardsFromToken(bulls[index], false);  }  for (uint256 index = 0; index < godsCount; index++) {  amount += claimRewardsFromToken(gods[index], true);  }  bapMeth.claim(\_msgSender(), amount);  } |

**Status: Open**

### I.9 Pragma Unlocked

**Contract**: BAPUtilities.sol, BAPTeenBulls.sol, BAPOrchestrator.sol, BAPGenesis.sol, Migrations.sol

**Description:**

Every Solidity file specifies in the header a version number of the format. The caret symbol before the version number implies an unlocked pragma, meaning that the compiler will use the specified version and above. This range of versions might cause some unexpected version related issues.

**Recommendation:**

Fix the solidity version by removing the caret symbol the specified version numbers.**Status: Open**

### I.10 Missing events in setters

**Description:**

The setters created in all the contracts within the repo almost never include an event which is emitted when the values are set.

**Recommendation:**Create and emit events for every setter.**Status: Open**

### I.11 Incorrect error message

**Contract:** BAPVesting.sol

**Description:**require check in *addVestingSchedule* throws zero address when you pass start value less than block.timestamp  
  
**Recommendation:**Change the error message to convey that start value is wrong.

### I.12 Best Practices

**Description:**

Some of the best practices are not followed throughout the repo, some simple recommendations can be implemented.

**Recommendation:**1. Internal method names should be preceded with ‘\_’ to differentiate it from public and external methods, internal methods should also be written towards the end of the contract.

2. Define modifier *onlyOrchestrator* before initialize block**Status: Open**

# Functional Tests (Goerli)

**BAPGenesis(0x543D097779eDeAad3D485a47D28ED45689aA549a)**

*airdrop*

0x09236b19bf37ebd58a58172392187faf194b5acfbfc03a38b903e068be24b945

*setOpen*

0x5bbc135721a812619dec6e86639a14007bff7efa884f4f6638ecabd62d929651

mint

0x1cf35fca634381ca52039882bc9acda6834e3a1970767510fb656e1ad8c49175

0x02dbb0f89e3dc39e8b121b69b81f6b4af24150c36bc3c223beed4f1abfb0420d

setOrchestrator

0xd2bb6bf3d4f0e08490f90df810c203eb10073d3567341af180072016523285b5

refund

0x7cd61d5e5b3218823406a4bd3c031342029a18a3f4d35aed6e944b7989885842

withdrawETH

0xfc4d7e10b3cc2221b2a8aac81e249d39e047b74c6e41a97aabd2c3fc1f9bd0f2

0x87d18a3a49b3725904cdef168f5943a45bfaf2d6e70b0e01b323d5b2e0cd0741 (revert)

**BAPOrchastrator(0x4a106F9455342D199b9241b568D9B28a2C412a20)**

*setUtilitiesContract*: 0xbe0e2d91337ddbe3a8882bbbcbccb94c36c5ad9d8b0303b51cf13b2db57bef8e

*setTeenBulls*

0xe1be759706d1850772710c412e0551a98604a171e5cdac5a597db6db98aa9f8f

**BAPMethane(0x67D84E2c472b54f776a676DfBdCfA1F32d20C86b**

*setOpen*

0x2a198b3c3efa863ed667a773e6793e96910c36e2f5c0622d5d143dde330033fd

*Airdrop*

0x1f59efc88fd301695356f1d9ec58616ce6d05c28ff88e75fcf5e8bb2799d02c6

0xe01c2882be4b331823258760c7dfe4fd469184f8227ec2a82dbde22e1a2c7659

*setOrchestrator*

0x7861295c0d23b8f6208e297263a6ab24a876e66764d9e0afb08230125f111196

0xa8106c3e4de4e6936ac0c5a1d4d863149c3978291d6aad971fe31545248bcc12

*pay*

0xc504d81660bea76a100e006d1b979e1123c200fdb34e27e9031aa118abc6d599  
  
*setVestingManager*0xc6a3b09e7ea695254ed317dab6c39cef2e51e23c47234fd9b742b822a2ee167e

**BAPVesting(0x72D2c8c638F647d80535C9f9cad403BC0f75195e)**

*addVestingSchedule*

0x1d6f3ac769d9ede65e2492ad3a4ad3c4faa46fa908690ccf945845e735fb56bf   
  
*vesting*  
0x2251a679ac7fb38de59799fa967b7418166769e185ce4b0348755bf4ba127a4e   
  
**BAPTeenBulls(0xEaf0CCB0b5a13707e657B8A241fa071ed3a64D86)**

*setOpen*

0x7e34e24e3c5a41b53dbd58cfbd4670b706b2b94af18cb5020ae3e6b9eae37c3a

setOrchestrator

0x7b8b0541bccf1925021b4886dc131feb5b1322bdb20a72d9fef4a1505cc97a27

generateTeenBull(FAILING)

0xba364022ca103d61d534f39a999a21a88411c6c2b14c21a3b4f197c2dd557c86  
  
**BAPUtilities(0x84bD89b302e87D24A53C280bDe71378B6cE02Df4)***setOpen*

0x8adbebbf0dd2c88b6476b64d668556a6f4ccee05d60ea0985172313781afcbf3

*purchaseIncubator*

0x43d368aa4a4a8842e3ac88625b7ebf11451e2c51bebf7a0fb905dda6a0abea32

*purchaseMergeOrb*

0x1f8dbd927f669eb323f9f1fa70267bc42dd4e6fb44189e2ed924bfb017bc574a

# Automated Tests **Slither:**

# 

# Results

A few major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

# **Closing Summary**

Overall, smart contracts are well written and adhere to guidelines.

Numerous issues were discovered in the initial audit. It is recommended to kindly go through the above-mentioned details and fix the code accordingly.

# Disclaimer

Indiviual audit is not a security warranty, investment advice, or an endorsement of the **JMDLabs platform**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that JMDLabs put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.