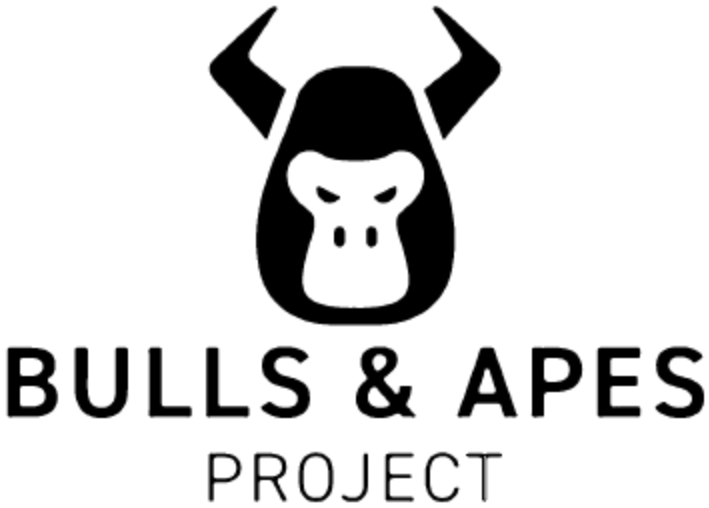
BULLS AND APES PROJECT

NFT

Smart Contract Audit
Final Report



May 28, 2022

Introduction

1. About BULLS AND APES PROJECT

Bulls and Apes Project (B.A.P.) is a generative 3D NFT project that aims to transform the industry by setting a new standard for what NFT collectors should expect and demand.

Starting with de-risking the NFT buying experience for you. We stand behind our products and will back them with an industry-changing 6 Month ETH-Back Guarantee, built right into the smart contract. Since our project is fully funded by our founders we can lock up the mint proceeds until we know you are thrilled to own our products.

Visit https://www.bullsandapesproject.com/ to know more about it.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provides professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-ups with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The BAP team has provided the following doc for the purpose of audit:

1. https://www.bullsandapesproject.com/faqs

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: Bulls and Apes Project
- Contracts Name: BAPGenesis.sol, BAPMethane.sol, BAPOrchestrator.sol, BAPTeenBulls.sol, BAPUtilities.sol, BAPVesting.sol, ERC721A.sol, IERC721A.sol, Migrations.sol
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Link for codebase for audit: https://bitbucket.org/jdmlabs/contracts-bap/src/master/
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level Reference

Every issue in this report were assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	-	-	-
Closed	2	2	3

High Severity Issues

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

   Amended: The issue was fixed by the **BAP** team.

2. **BAPMethane.pay fails to execute**
   Contract: BAPOrchestrator.sol, BAPMethane.sol

   Description:
   The method calls in BAPOrchestrator which is internally called bapMeth.pay reverts, we are not sure how this transfer is supposed to work, since it will transfer contract from itself and we are not minting or transferring any BAPMethane to BAPMethane address.

```
function pay(uint256 paymentAmount, uint256 fee) public {
  require(open, "100:CLOSED");
  require(orchestrator == msg.sender, "500:UNAUTHORIZED");
  require(balanceOf(tx.origin) >= paymentAmount, "300:INSUFFICIENT_METH");
  uint256 _fee = fee;
  // Protect unsingned operation
  if (fee > paymentAmount) {
    _fee = paymentAmount;
  }
  uint256 toBurn = paymentAmount - _fee;
  transfer(treasuryWallet, _fee);
  if (toBurn > 0) {
    _burn(tx.origin, toBurn);
    burned += toBurn;
  } }
```

   Amended: The issue was fixed by the **BAP** team.

Medium Severity Issues

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 don't perform any kind of state update or validation on the orchestrator address, which allows the 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 that 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 validation to check what address is 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.

   Amended: The issue was fixed by the **BAP** team.

2. **Incorrect require check**
   Contract: BAPGenesis.sol

   Description:
   The verify method in the contract returns true if the value of the 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 the intended behavior.

Comment: false flag

Amended: The issue was fixed by the **BAP** team.

Low Severity Issues

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 is too many lists down.

   Recommendation:
   We should never initialize state variables with hardcoded values in the 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 defined state constants which can be used if it needs to be a fixed value.

   Amended: The issue was fixed by the **BAP** team.

2. **Missing setter validation for mintingMin**
   Contract: BAPMethane.sol

   Description:
   The value of mintingMin can even be set to zero as there are no validations for the value in the setter. A situation like this will allow a method like a 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”

Amended: The issue was fixed by the **BAP** team.

3. **Claim with amount zero**
Contract: BAPVesting.sol

Description:
In the vesting method if meth amount fetched is zero, the method still goes ahead to execute the 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

Amended: The issue was fixed by the **BAP** team.

Recommendation / Informational

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
BAPOrchestrator - 15	address public bapGenesisAddr; address public bapMethAddr; address public bapUtilitiesAddr;
BAPVesting - 24	address public methContractAdress;

   Recommendation:
   Remove unused variables

   Amended: The issue was fixed by the **BAP** team.

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

   Amended: The issue was fixed by the **BAP** team.

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.

   Amended: The issue was fixed by the **BAP** team.

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.

Comment: Follow the naming convention for constants

Amended: The issue was fixed by the **BAP** team.

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.

   Amended: The issue was fixed by the **BAP** team.

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.

   Amended: The issue was fixed by the **BAP** team.

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

   Amended: The issue was fixed by the **BAP** team.

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);
  }
```

Amended: The issue was fixed by the **BAP** team.

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.

Amended: The issue was fixed by the **BAP** team.

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.

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 the start value is wrong.

    Amended: The issue was fixed by the **BAP** team.

12. **Best Practices**

    Description:
    Some of the best practices are not followed throughout the repo, and 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

    Amended: The issue was fixed by the **BAP** team.

Testnet Testing

1. BAPGenesis(0x543D097779eDeAad3D485a47D28ED45689aA549a)
   Postfix - 0x399Cc9688a873C45F5eC1bE4143A6ef618a2302C

   airdrop
   0x09236b19bf37ebd58a58172392187faf194b5acfbfc03a38b903e068be24b945

   setOpen
   0x5bbc135721a812619dec6e86639a14007bff7efa884f4f6638ecabd62d929651

   mint
   0x1cf35fca634381ca52039882bc9acda6834e3a1970767510fb656e1ad8c49175
   0x02dbb0f89e3dc39e8b121b69b81f6b4af24150c36bc3c223beed4f1abfb0420d
   0x5c7001c2af20d8bacba97b38940d06447639e0a2c0de921105f15ded72d81695

   setOrchestrator
   0xd2bb6bf3d4f0e08490f90df810c203eb10073d3567341af180072016523285b5

   refund
   0x7cd61d5e5b3218823406a4bd3c031342029a18a3f4d35aed6e944b7989885842

   withdrawETH
   0xfc4d7e10b3cc2221b2a8aac81e249d39e047b74c6e41a97aabd2c3fc1f9bd0f2
   0x87d18a3a49b3725904cdef168f5943a45bfaf2d6e70b0e01b323d5b2e0cd0741 (revert)

2. BAPOrchastrator(0x4a106F9455342D199b9241b568D9B28a2C412a20)
   Postfix - 0xCa22c0c68aa891517Da849e23711Ef555C304476

   setUtilitiesContract: 0xbe0e2d91337ddbe3a8882bbbcbccb94c36c5ad9d8b0303b51cf13b2db57bef8e

   setTeenBulls
   0xe1be759706d1850772710c412e0551a98604a171e5cdac5a597db6db98aa9f8f

   setClaimFlag
   0xce68e32836d47083833c200c7b0286ea572516abffd51440f5fb4b456edec695

   claimMeth
   0xb119894460f380d2e4764c2c58f4250dc8356d3c6db797da6129b1dd314341da

0x259bf217c19968b6be17e76b85c2179c99e0943cde6a8fb424a8724e6988cbbb

setWhitelistedAddress
0x6e455e28b9417a4e5e9e3363e1da6356800612d0097fddad7db1e355fe748545

3. BAPMethane(0x67D84E2c472b54f776a676DfBdCfA1F32d20C86b
Postfix - 0xA9F2EffBf178DcF98eBC74cC9CBb2afEE366221c

setOpen
0x2a198b3c3efa863ed667a773e6793e96910c36e2f5c0622d5d143dde330033fd

Airdrop
0x1f59efc88fd301695356f1d9ec58616ce6d05c28ff88e75fcf5e8bb2799d02c6
0xe01c2882be4b331823258760c7dfe4fd469184f8227ec2a82dbde22e1a2c7659

setOrchestrator
0x7861295c0d23b8f6208e297263a6ab24a876e66764d9e0afb08230125f111196
0xa8106c3e4de4e6936ac0c5a1d4d863149c3978291d6aad971fe31545248bcc12

pay
0xc504d81660bea76a100e006d1b979e1123c200fdb34e27e9031aa118abc6d599

setVestingManager
0xc6a3b09e7ea695254ed317dab6c39cef2e51e23c47234fd9b742b822a2ee167e

4. BAPVesting(0x72D2c8c638F647d80535C9f9cad403BC0f75195e)

addVestingSchedule
0x1d6f3ac769d9ede65e2492ad3a4ad3c4faa46fa908690ccf945845e735fb56bf

vesting
0x2251a679ac7fb38de59799fa967b7418166769e185ce4b0348755bf4ba127a4e

5. BAPTeenBulls(0xEaf0CCB0b5a13707e657B8A241fa071ed3a64D86)

setOpen
0x7e34e24e3c5a41b53dbd58cfbd4670b706b2b94af18cb5020ae3e6b9eae37c3a

setOrchestrator
0x7b8b0541bccf1925021b4886dc131feb5b1322bdb20a72d9fef4a1505cc97a27

generateTeenBull(FAILING)
0xba364022ca103d61d534f39a999a21a88411c6c2b14c21a3b4f197c2dd557c86

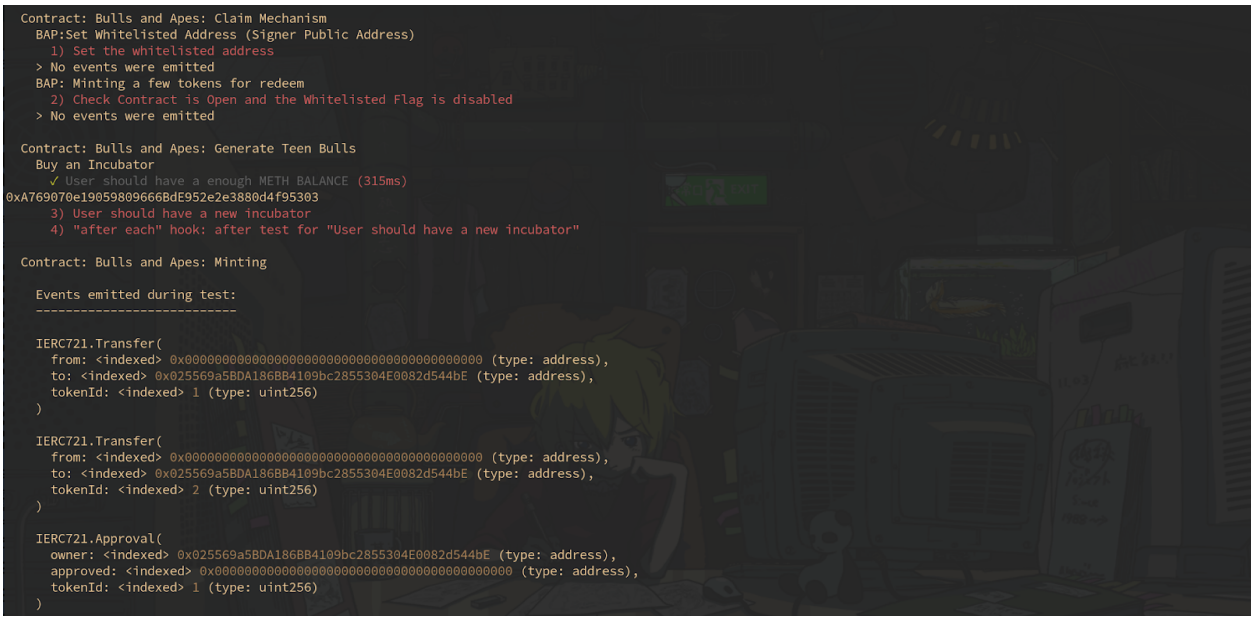
6. BAPUtilities(0x84bD89b302e87D24A53C280bDe71378B6cE02Df4)

   setOpen
   0x8adbebbf0dd2c88b6476b64d668556a6f4ccee05d60ea0985172313781afcbf3

   purchaseIncubator
   0x43d368aa4a4a8842e3ac88625b7ebf11451e2c51bebf7a0fb905dda6a0abea32

   purchaseMergeOrb
   0x1f8dbd927f669eb323f9f1fa70267bc42dd4e6fb44189e2ed924bfb017bc574a

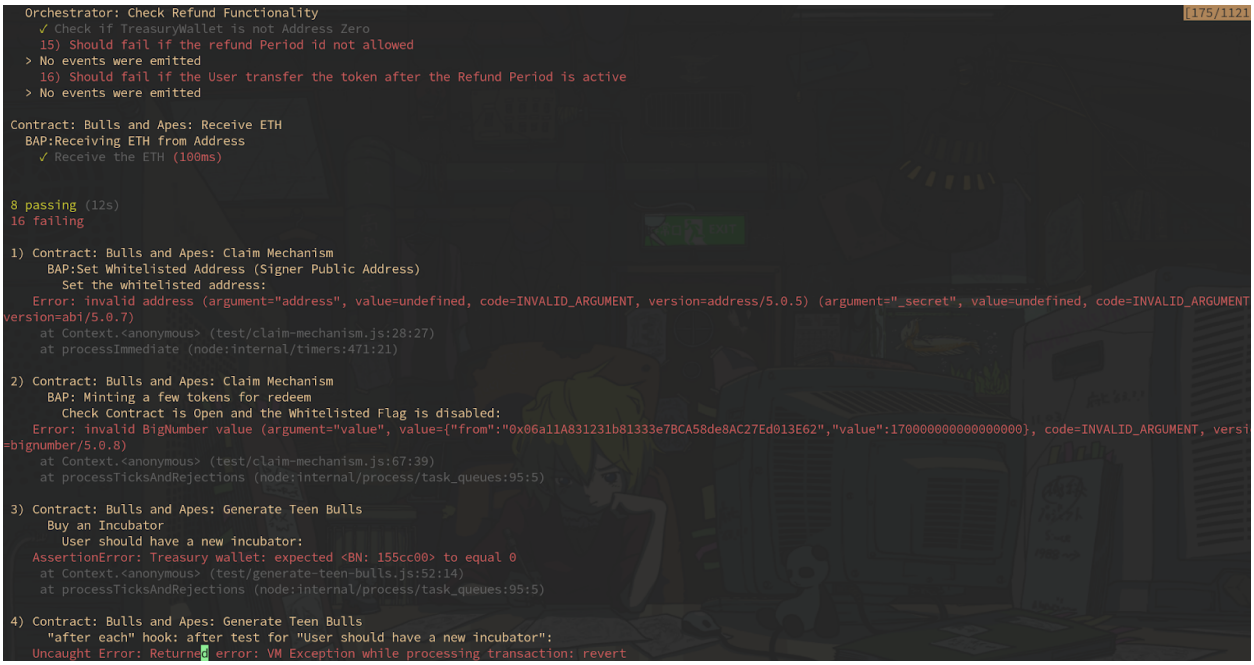
Unit Tests

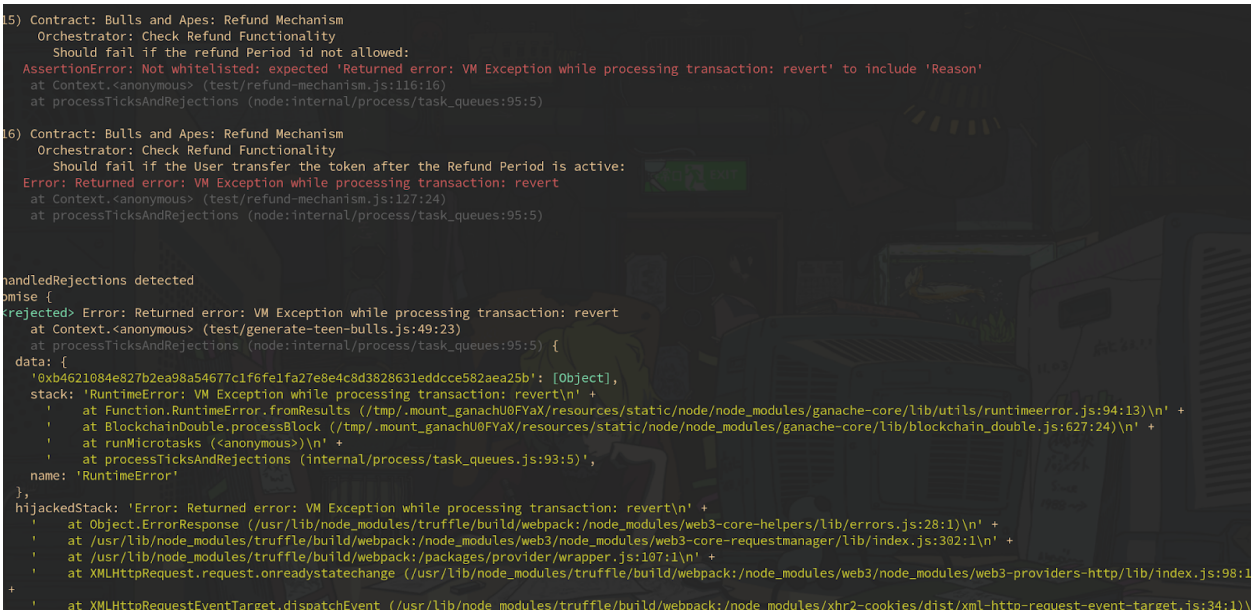


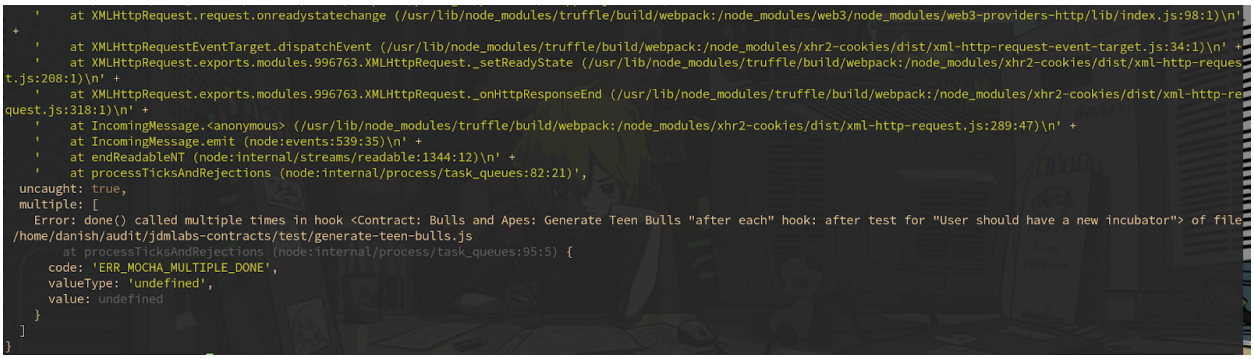
```
  Contract: Bulls and Apes: Claim Mechanism
    BAP:Set Whitelisted Address (Signer Public Address)
      1) Set the whitelisted address
    > No events were emitted
    BAP: Minting a few tokens for redeem
      2) Check Contract is Open and the Whitelisted Flag is disabled
    > No events were emitted

  Contract: Bulls and Apes: Generate Teen Bulls
    Buy an Incubator
      ✓ User should have a enough METH BALANCE (315ms)
0xA769070e19059809666BdE952e2e3880d4f95303
      3) User should have a new incubator
      4) "after each" hook: after test for "User should have a new incubator"

  Contract: Bulls and Apes: Minting

    Events emitted during test:
    ---------------------------

    IERC721.Transfer(
      from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
      to: <indexed> 0x025569a5BDA186BB4109bc2855304E0082d544bE (type: address),
      tokenId: <indexed> 1 (type: uint256)
    )

    IERC721.Transfer(
      from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
      to: <indexed> 0x025569a5BDA186BB4109bc2855304E0082d544bE (type: address),
      tokenId: <indexed> 2 (type: uint256)
    )

    IERC721.Approval(
      owner: <indexed> 0x025569a5BDA186BB4109bc2855304E0082d544bE (type: address),
      approved: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
      tokenId: <indexed> 1 (type: uint256)
    )
```

```
    IERC721.Approval(
      owner: <indexed> 0x025569a5BDA186BB4109bc2855304E0082d544bE (type: address),
      approved: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
      tokenId: <indexed> 2 (type: uint256)
    )


    ---------------------------
    5) "after each" hook: after test for "User should have a new incubator"
    BAP:Set Whitelisted Address (Signer Public Address)
      6) Set the whitelisted address
    > No events were emitted
    BAP:Check Whitelist Minting
      ✓ Check Open the Contract and set Whitelist to true (599ms)
      ✓ Should throw if the minter is trying to mint with a wrong signature (973ms)
      ✓ Should throw if the minter is trying to mint with a higher limit (818ms)
      7) Should throw if the minter is trying to mint with a bigger stage than the contract has
    > No events were emitted
      8) Check mint fuctionality with the contract whitelisted
    > No events were emitted
      9) Should allow to change the contract stage and mint in the next stage
    > No events were emitted
      10) Should allow to change the contract stage and mint in the next stage  Total:4
    > No events were emitted
      11) Should allow to change the contract stage and mint in the next stage  Total:5
    > No events were emitted
      ✓ Should fail to change the contract stage and mint in the next stage  Total:5 + 2 (155ms)

  Contract: Bulls and Apes: Refund Mechanism
    12) It should allow to get the refund: Waiting
    > No events were emitted
    BAP:Check name and symbol
      13) should match name and symbol
    > No events were emitted
    BAP:Check owner
      ✓ Check ownership
    BAP: Minting a few tokens
      14) Check Contract is Open and the Whitelisted Flag is disabled
    > No events were emitted
    Orchestrator: Check Refund Functionality
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.







Automated Audit Result

Slither

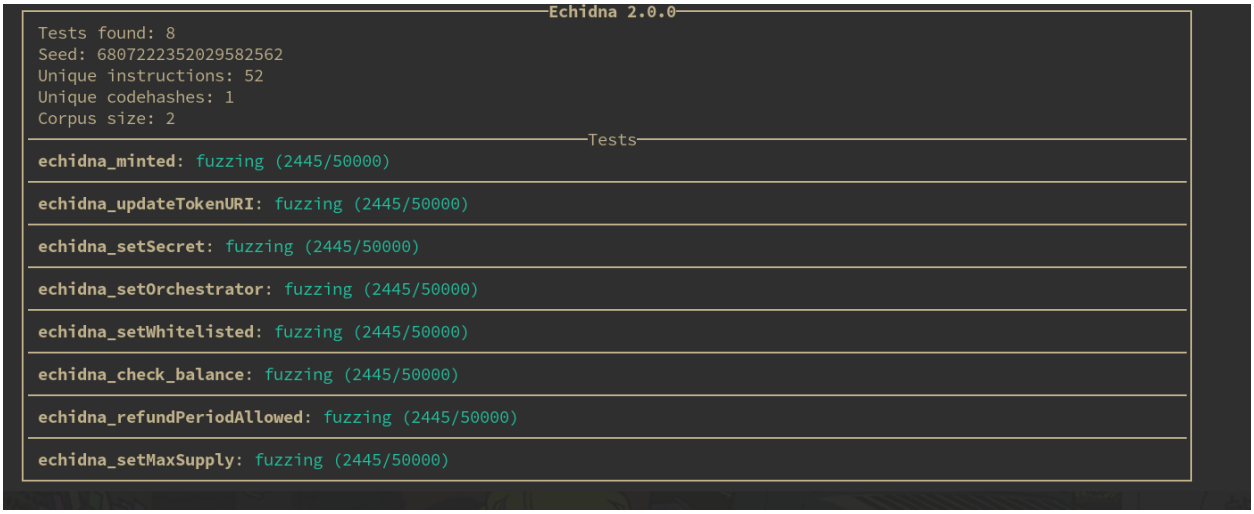
```
Compiled with solc
Number of lines: 725 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 6 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 16
Number of informational issues: 13
Number of low issues: 3
Number of medium issues: 1
Number of high issues: 0

ERCs: ERC20

+------------+-------------+-------+--------------------+--------------+----------+
|    Name    | # functions | ERCS  |     ERC20 info     | Complex code | Features |
+------------+-------------+-------+--------------------+--------------+----------+
| BAPMethane |      48     | ERC20 |     No Minting     |      No      |          |
|            |             |       | Approve Race Cond. |              |          |
|            |             |       |                    |              |          |
+------------+-------------+-------+--------------------+--------------+----------+
```

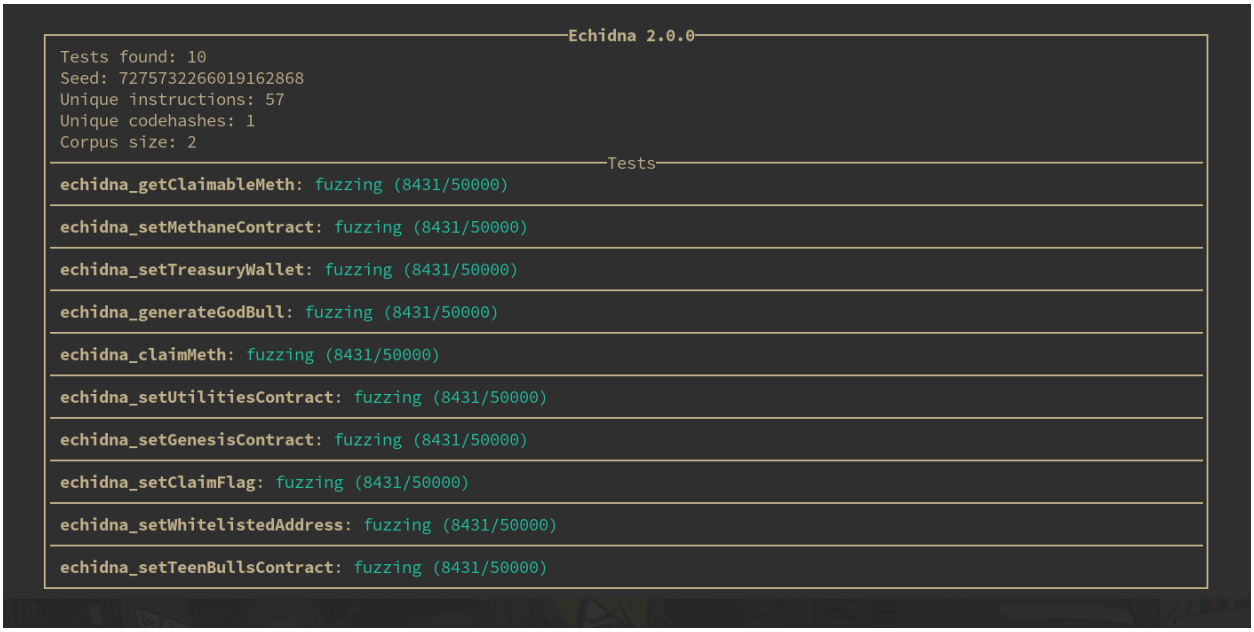
```
Compiled with solc
Number of lines: 832 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 9 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 36
Number of low issues: 13
Number of medium issues: 3
Number of high issues: 0


+----------------------+-------------+------+------------+--------------+--------------------+
|         Name         | # functions | ERCS | ERC20 info | Complex code |      Features      |
+----------------------+-------------+------+------------+--------------+--------------------+
|       SafeMath       |      13     |      |            |      No      |                    |
|  BAPGenesisInterface |      17     |      |            |      No      |    Receive ETH     |
|  BAPMethaneInterface |      6      |      |            |      No      |                    |
| BAPUtilitiesInterface |      3      |      |            |      No      |                    |
| BAPTeenBullsInterface |      2      |      |            |      No      |                    |
|    BAPOrchestrator   |      34     |      |            |      No      |     Ecrecover      |
|                      |             |      |            |              | Tokens interaction |
|                      |             |      |            |              |      Assembly      |
+----------------------+-------------+------+------------+--------------+--------------------+
```

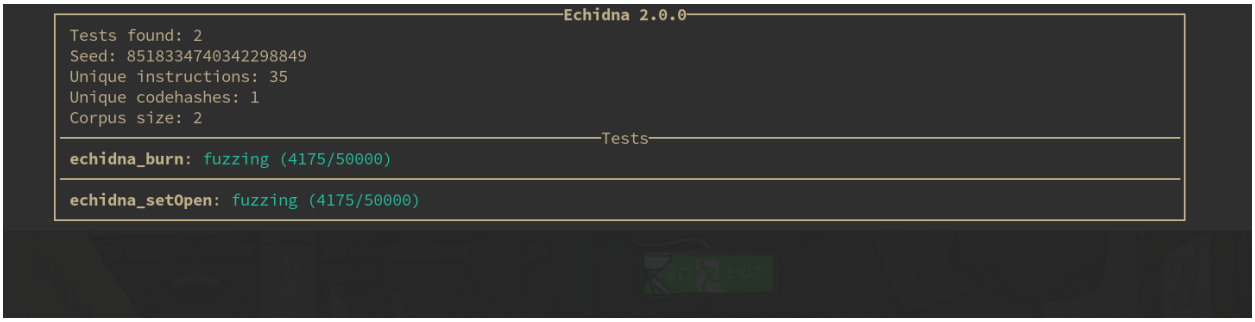
```
Compiled with solc
Number of lines: 1446 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 12 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 11
Number of informational issues: 45
Number of low issues: 20
Number of medium issues: 2
Number of high issues: 0

ERCs: ERC721, ERC165

+-----------------+-------------+---------------+------------+--------------+--------------+
|       Name      | # functions |      ERCS     | ERC20 info | Complex code |   Features   |
+-----------------+-------------+---------------+------------+--------------+--------------+
| IERC721Receiver |      1      |               |            |      No      |              |
|     Address     |      11     |               |            |      No      |   Send ETH   |
|                 |             |               |            |              | Delegatecall |
|                 |             |               |            |              |   Assembly   |
|     Strings     |      4      |               |            |     Yes      |              |
|   BAPTeenBulls  |      68     | ERC165,ERC721 |            |     Yes      |   Assembly   |
+-----------------+-------------+---------------+------------+--------------+--------------+
BAPTeenBulls_flat.sol analyzed (12 contracts)
```

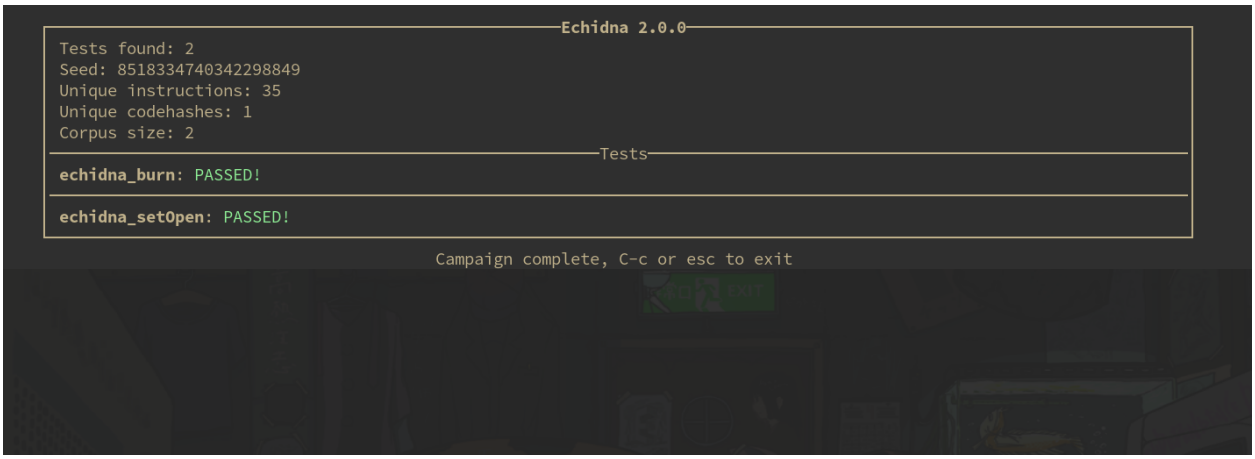
This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Echidna









Concluding Remarks

While conducting the audits of the BAP smart contracts, it was observed that the contracts contain High, Medium, and Low severity issues.

Our auditors suggest that the Hight, Medium, and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Notes

- ***The BAP team has fixed the issues based on the auditor's recommendation.***

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the BULLS AND APES PROJECT platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes