Competitive Security Assessment

ApeX

Nov 24th, 2022

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	ApeX
Platform & Language	Solidity
Codebase	• repo - https://github.com/ApeX-Protocol/periphery/ • audit commit - 4661330a339e1e7e5888ec7d7e457f6ee8c1af53 • final commit - 09e0f3f4f314807b782aeeed09ef862f39e07d0d --- • repo - https://github.com/ApeX-Protocol/apexpro-contracts • audit commit - 2fa3161d72e21908a012cac778bfafc45819e46e • final commit - 2fa3161d72e21908a012cac778bfafc45819e46e (same)
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

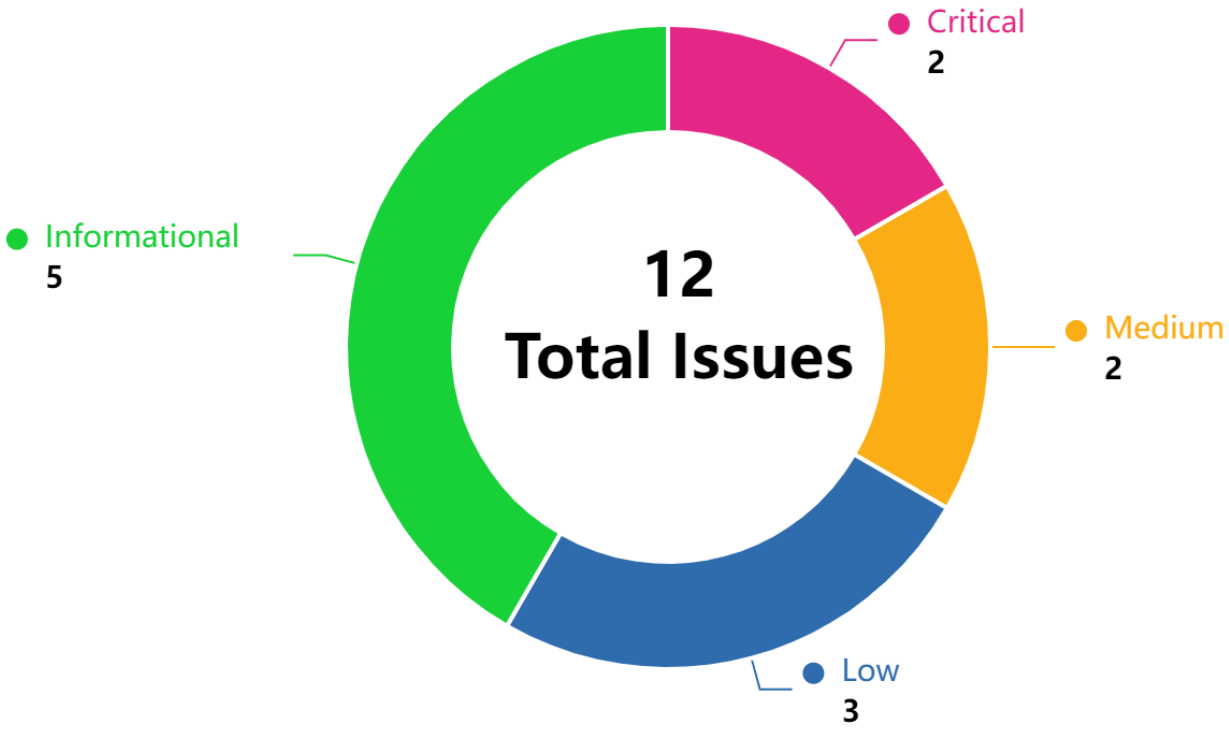
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	2	0	1	0	0	1
Medium	2	0	2	0	0	0
Low	3	0	3	0	0	0
Informational	5	0	1	3	0	1

Audit Scope

File	Commit Hash
banana/BuybackPool.sol	4661330a339e1e7e5888ec7d7e457f6ee8c1af53
banana/interfaces/ITWAMM.sol	4661330a339e1e7e5888ec7d7e457f6ee8c1af53
banana/Banana.sol	4661330a339e1e7e5888ec7d7e457f6ee8c1af53
banana/BananaDistributor.sol	4661330a339e1e7e5888ec7d7e457f6ee8c1af53
banana/BananaClaimable.sol	4661330a339e1e7e5888ec7d7e457f6ee8c1af53
banana/interfaces/ITWAMMPair.sol	4661330a339e1e7e5888ec7d7e457f6ee8c1af53
banana/interfaces/IBanana.sol	4661330a339e1e7e5888ec7d7e457f6ee8c1af53
banana/interfaces/IBananaDistributor.sol	4661330a339e1e7e5888ec7d7e457f6ee8c1af53
contracts/core/MultiSigPool.sol	2fa3161d72e21908a012cac778bfafc45819e46e
contracts/core/SelfSufficientERC20.sol	2fa3161d72e21908a012cac778bfafc45819e46e
contracts/core/MarketMaker.sol	2fa3161d72e21908a012cac778bfafc45819e46e
contracts/interfaces/IStarkEx.sol	2fa3161d72e21908a012cac778bfafc45819e46e
contracts/interfaces/IAggregationRouterV4.sol	2fa3161d72e21908a012cac778bfafc45819e46e
contracts/interfaces/IFactRegister.sol	2fa3161d72e21908a012cac778bfafc45819e46e
contracts/interfaces/IWETH.sol	2fa3161d72e21908a012cac778bfafc45819e46e
contracts/interfaces/IAggregationExecutor.sol	2fa3161d72e21908a012cac778bfafc45819e46e

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
APX-1	Banana Contract `transferFrom` Operation Lacks `Approve` Event Updates	Logical	Informational	Fixed	Hellobloc
APX-2	Contract may permanent broken due to careless constructor parameter	Logical	Informational	Acknowledged	0xxm
APX-3	First minter can break minting of BANA	Logical	Informational	Declined	thereksfour
APX-4	Function parameter Should be Declared as Calldata	Code Style	Informational	Fixed	0xxm
APX-5	Incomplete different signer checker	Logical	Low	Acknowledged	iczc

APX-6	Missing msg.value check in deposit native token	Logical	Medium	Acknowledged	iczc
APX-7	Reset allowance when using `safeApprove`	Race Condition	Low	Acknowledged	0xxm
APX-8	Signature replay for different chains	Signature Forgery or Replay	Critical	Acknowledged	p41m0n, Hellobloc, 0xoyst2r
APX-9	User-provided `exchangeData` is not sufficiently validated.	Logical	Medium	Acknowledged	p41m0n
APX-10	When the user transfers tokens to himself, the amount of tokens is double counted	Logical	Critical	Declined	thereksfour, 0xac
APX-11	`Banana.approve()` can be front-run	Logical	Low	Acknowledged	p41m0n
APX-12	`Banana::_mint` should check `to` to avoid tokens being permanently locked	Logical	Informational	Fixed	yekong, Hellobloc

APX-1:Banana Contract `transferFrom` Operation Lacks `Approve` Event Updates

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	code/banana/contracts/banana/Banana.sol#L107-L117	Fixed	Hellobloc

Code

```
107:    function _spendAllowance(
108:        address from,
109:        address spender,
110:        uint256 value
111:    ) internal virtual {
112:        uint256 currentAllowance = allowance[from][spender];
113:        if (currentAllowance != type(uint256).max) {
114:            require(currentAllowance >= value, "insufficient allowance");
115:            allowance[from][spender] = currentAllowance - value;
116:        }
117:    }
```

Description

Hellobloc : `Banana` contract `transferFrom` operation lacks `Approve` event updates.

```
function _spendAllowance(
        address from,
        address spender,
        uint256 value
    ) internal virtual {
        uint256 currentAllowance = allowance[from][spender];
        if (currentAllowance != type(uint256).max) {
            require(currentAllowance >= value, "insufficient allowance");
            [+]allowance[from][spender] = currentAllowance - value;
            [-]unchecked {
            [-]  _approve(owner, spender, currentAllowance - amount);
            [-]}
        }
    }
```

This can result in missing `Approve` event content, which ultimately causes false recognition by the `dextool`.

Recommendation

Hellobloc : We recommend implementing the `Banana` contract using the inherited `ERC20` contract to ensure code specification.

Client Response

Fixed

APX-2:Contract may permanent broken due to careless constructor parameter

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	code/banana/contracts/banana/BananaDistributor.sol#L30-L49	Acknowledged	0xxm

Code

```
30:     constructor(
31:         address banana_,
32:         address keeper_,
33:         address rewardRecipient_,
34:         uint256 duration_,
35:         uint256 distributeTime_,
36:         uint256 endTime_,
37:         uint256 initReward_,
38:         uint256 delta_
39:     ) {
40:         owner = msg.sender;
41:         banana = banana_;
42:         keeper = keeper_;
43:         rewardRecipient = rewardRecipient_;
44:         duration = duration_;
45:         distributeTime = distributeTime_;
46:         endTime = endTime_;
47:         lastReward = initReward_;
48:         delta = delta_;
49:     }
```

Description

0xxm : variable lastReward can only be initialized in constructor. If it is carelessly set to zero, distribute() will always calculate newReward be 0, and cause transaction revert. Unlike other parameters of constructor can be changed afterwards, this function broken cannot be remedied.

Recommendation

0xxm : Add non-zero check for initReward_ to avoid careless setting the constructor parameter

Client Response

No need to amend. We deploy this contract by script and always set the `initReward` to be nonzero value. On the other hand, less code less gas.

APX-3:First minter can break minting of BANA

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	code/banana/contracts/banana/Banana.sol#L45-L60	Declined	thereksfour

Code

```
45:    function mint(address to, uint256 apeXAmount) external override returns (uint256) {
46:        require(minters[msg.sender], "forbidden");
47:        require(apeXAmount > 0, "zero amount");
48:
49:        uint256 apeXBalance = IERC20(apeXToken).balanceOf(address(this));
50:        uint256 mintAmount;
51:        if (totalSupply == 0) {
52:            mintAmount = apeXAmount * 1000;
53:        } else {
54:            mintAmount = apeXAmount.mulDiv(totalSupply, apeXBalance);
55:        }
56:
57:        TransferHelper.safeTransferFrom(apeXToken, msg.sender, address(this), apeXAmount);
58:        _mint(to, mintAmount);
59:        return mintAmount;
60:    }
```

Description

thereksfour : According to the documentation, BANA is minted by the admin. But in Banana.sol, minter is specified by owner and minter mints BANA. In the mint function, the malicious first minter can manipulate apeXBalance to prevent other minters from minting BANA tokens. Consider the following scenario The malicious first minter calls the mint function and uses 1 wei apeXToken to mint 1000 wei BANA. Then the malicious first minter transfers 1000e18 wei apeXToken to the Banana contract, at this time apeXBalance = 1000e18 + 1 wei. Then other minters call mint function with apeXAmount == 1e18. The number of BANA tokens minted by this minter is 1e18*1000/(1000e18 + 1) = 0. Finally, the malicious first minter can redeem all apeXTokens in the contract.

Recommendation

thereksfour : Uniswap V2 solved this problem by sending the first 1000 LP tokens to the zero address https://github.com/Uniswap/v2-core/blob/master/contracts/UniswapV2Pair.sol#L119-L124

Client Response

No need to amend. The hypothetical scenario in the report is not going to happen in fact. First, minter will be specified to a gnosis safe wallet, not to a malicious minter. Second, even if the first minter really mint 1000 wei BANA and transfer 1000e18 wei apeXToken, the other minters will not mint BANA with apeXAmount == 1e18, it's highly inlikely to happen.

APX-4:Function parameter Should be Declared as Calldata

Category	Severity	Code Reference	Status	Contributor
Code Style	**Informational**	**code/banana/contracts/banana/BananaClaimable.sol#L42 code/banana/contracts/banana/BananaClaimable.sol#L57**	**Fixed**	**0xxm**

Code

```
42:    function claim(

57:    function verify(
```

Description

0xxm : Both `claim` and `verify` functions has bytes arguments `nonce` and `signature`, but one is declared as calldata and another as memory. Function can directly read the parameters from calldata. Setting it to other storage locations may waste gas.

Recommendation

0xxm : Change storage location of `signature` in function `claim` and `verify` as calldata.

Client Response

Fixed

APX-5:Incomplete different signer checker

Category	Severity	Code Reference	Status	Contributor
Logical	Low	code/multisig/contracts/core/MarketMaker.sol#L180 code/multisig/contracts/core/MultiSigPool.sol#L187 code/multisig/contracts/core/MarketMaker.sol#L225 code/multisig/contracts/core/MultiSigPool.sol#L233	Acknowledged	iczc

Code

```
180:    require(allSigners[0] != allSigners[1], "can not be same signer"); // must be different signer

187:    require(allSigners[0] != allSigners[1],"can not be same signer"); // must be different signer

225:    require(allSigners[0] != allSigners[1], "can not be same signer"); // must be different signer

233:    require(expireTime >= block.timestamp,"expired transaction");
```

Description

iczc : Different signer checker only check the first and the second cannot be the same, and allow the same signer if the signer is greater than 2. This results in multi-sign not reaching the actual threshold.

Recommendation

iczc : De-duplicate signers with set, then check if the length is the same as the original.

Client Response

No need to amend. Signers are requied 2/3, (allSigners[0] != allSigners[1]) already means there are at least two different signers, even there are more that 2 signers and have some signer in the allSigners array.

APX-6:Missing msg.value check in deposit native token

Category	Severity	Code Reference	Status	Contributor
Logical	**Medium**	**code/multisig/contracts/core/MultiSigPool.sol#L103**	**Acknowledged**	**iczc**

Code

```
103:  function deposit(
```

Description

iczc : There is no requirement for the amount argument to equal msg.value in deposit native token, which causes the amount to be arbitrarily forged and the emit event is also faked data. This further results in the off-chain infura getting the wrong amount data.

Recommendation

iczc : make sure the amount is equal msg.value in deposit native case.

Client Response

Deposit native token, will call 1inch to swap USDC, if the msg.value is not right, the USDC will be wrong.

APX-7:Reset allowance when using `safeApprove`

Category	Severity	Code Reference	Status	Contributor
Race Condition	Low	code/multisig/contracts/core/MultiSigPool.sol#L128 code/multisig/contracts/core/MultiSigPool.sol#L146 code/multisig/contracts/core/MultiSigPool.sol#L299	Acknowledged	0xxm

Code

```
128:        desc.srcToken.safeApprove(AGGREGATION_ROUTER_V4_ADDRESS, 0);

146:      IERC20(USDC_ADDRESS).safeApprove(STARKEX_ADDRESS, 0);

299:    IERC20(token).safeApprove(FACT_ADDRESS, 0);
```

Description

0xxm : The initial intention of `safeApprove` to only allow allowance change from zero to non-zero is to avoid excessive spend of owner's allowance by front running. However, reset allowance in one transcation doesn't prevent front running at all just a walkaround on `safeApprove`'s check.

Detailed disccussion about `ERC20-approve` issue can be found here: https://github.com/OpenZeppelin/openzeppelin-contracts/issues/438

Recommendation

0xxm : Use `safeIncreaseAllowance` and `safeDecreaseAllowance` instead. Considering all spenders are trusted addresses, it should be acceptable to simply use `approve` to change allowance.

Client Response

It's low risk. Because our contract is already deployed, so stay the same.

APX-8:Signature replay for different chains

Category	Severity	Code Reference	Status	Contributor
Signature Forgery or Replay	**Critical**	**code/multisig/contracts/core/MultiSigPool.sol#L190**	**Acknowledged**	**p41m0n, Hellobloc, 0xoyst2r**

Code

```
190:    bytes32 operationHash = keccak256(abi.encodePacked("ETHER", to, amount, expireTime, orderId, address(this)));
```

Description

p41m0n : According to line 144, the contract is deployed on multiple chains. However signatures used in `withdrawErc20()` and `withdrawETH()` do not contain a `chainid` field. Thus a valid signature can be replayed on another chain.

NOTE: This bug exists on other files, please check dev bros.

Hellobloc : The signed message in the current code lacks the important chainid information, which makes the contract vulnerable to replay attacks at different chains.

```
  bytes32 operationHash = keccak256(abi.encodePacked("ETHER", to, amount, expireTime, orderId, address(this)));
```

0xoyst2r : The `chainid` is missed in the signed message, this means the contract is vulnerable to the replay attacks on a different chain.

Recommendation

p41m0n : Add `block.chainid` into `keccak256(abi.encodePacked("ETHER", to, amount, expireTime, orderId, address(this)));`.
Hellobloc : We recommend following the recommendations of SWC-121 as follows.
In order to protect against signature replay attacks consider the following recommendations:

- Store every message hash that has been processed by the smart contract. When new messages are received check against the already existing ones and only proceed with the business logic if it's a new message hash.
- Include the address of the contract and chainid that processes the message. This ensures that the message can only be used in a single contract and single chain.
- Under no circumstances generate the message hash including the signature. The ecrecover function is susceptible to signature malleability (see also SWC-117).

0xoyst2r : Add `block.chainid` chainId information into the signed message.

Client Response

No need to amend. The contract addresses are different on different chains, address(this) already avoid replay.

APX-9:User-provided `exchangeData` is not sufficiently validated.

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	code/multisig/contracts/core/MultiSigPool.sol#L117-L121	Acknowledged	p41m0n

Code

```
117:      (, IAggregationRouterV4.SwapDescription memory desc,) = abi.decode(exchangeData[4:], (address,
IAggregationRouterV4.SwapDescription, bytes));
118:      require(token == desc.srcToken, 'mismatch token and desc.srcToken');
119:      require(USDC_ADDRESS == address(desc.dstToken), 'invalid desc.dstToken');
120:      require(amount == desc.amount, 'mismatch amount and desc.amount');
121:      require(address(this) == desc.dstReceiver, 'invalid desc.dstReceiver');
```

Description

p41m0n : `exchangeData` is the calldata that will be forwarded to 1inch `AggregationRouterV4` to do asset swap. It's user-controlled and should be carefully checked.

1. `exchangeData[0:4]` which is used as the selector to call `AggregationRouterV4` is not checked. Attackers can force `MultiSigPool` to invoke other methods of `AggregationRouterV4`. Then checks in line 118~121 can be bypassed as the function definition is different.
2. The first parameter of `IAggregationRouterV4.swap()` should be `AggregationExecutor` caller contract, such as 1inch: Aggregation Executor 2. It's not checked here. Attacker can deploy its own executor to change origional logic of 1inch swap and can drain any asset sent to `AggregationRouterV4`.

Recommendation

p41m0n : 1. Assert that the selector should be equal to `IAggregationRouterV4.swap.selector`. 2. Check the first variable decoded from `exchangeData` is offical 1inch Aggregation Executor contract.

Client Response

It's low risk. Also there is `require (afterSwapBalance == beforeSwapBalance.add(returnAmount),"swap incorrect");` to make sure the swap is done after the call.

APX-10:When the user transfers tokens to himself, the amount of tokens is double counted

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	code/multisig/contracts/core/SelfSufficientERC20.sol#L120-L121	Declined	thereksfour, 0xac

Code

```
120:        _balances[recipient] = recipient_balance + amount;
121:        emit Transfer(sender, recipient, amount);
```

Description

thereksfour : In the _transfer function, if sender == recipient, the amount of tokens will be double counted. Consider the following scenario. User A has 100 tokens and User A transfers 100 tokens to himself. In the following calculation, sender_balance == recipient_balance == 100 _balances[sender] = 100 - 100 = 0 _balances[recipient] = 100 + 100 = 200 At this point, user A has 200 tokens.
0xac : While the attacker calls the `transfer` function and the recipient address is himself, his balance would increase and the `_totalSupply` of token would not increase. The increase amount of attacker's balance is equal to the amount of transfer.

Recommendation

thereksfour : Change to

```
        _balances[sender] = sender_balance - amount;
-       _balances[recipient] = recipient_balance + amount;
+       _balances[recipient] = _balances[recipient] + amount;
```

0xac : To avoid this problem, suggesting to ensure the `to address` is not equal to the `from address`.

```
require(to != from);
```

Client Response

This contract is only for testnet, not in mainnet.

APX-11: `Banana.approve()` can be front-run

Category	Severity	Code Reference	Status	Contributor
Logical	Low	code/banana/contracts/banana/Banana.sol#L102	Acknowledged	p41m0n

Code

```
102:    function approve(address spender, uint256 value) external override returns (bool) {
```

Description

p41m0n : The ERC20 `approve()` is vulnerable to front-run attack, which allows the spender to front-run and take more tokens than the owner protend to approve.
Find more on:

- https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit

Recommendation

p41m0n : Implement increaseAllowance/decreaseAllowance method.
Another good choice to fix is to extend OpenZeppelin's ERC20.sol.

Client Response

It's low risk. And many other tokens are using the approve implement the same way. We stay the same.

APX-12: `Banana::_mint` should check `to` to avoid tokens being permanently locked

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	code/banana/contracts/banana/Banana.sol#L45-L60 code/banana/contracts/banana/Banana.sol#L119-L123	Fixed	yekong, Hellobloc

Code

```
45:    function mint(address to, uint256 apeXAmount) external override returns (uint256) {
46:        require(minters[msg.sender], "forbidden");
47:        require(apeXAmount > 0, "zero amount");
48:
49:        uint256 apeXBalance = IERC20(apeXToken).balanceOf(address(this));
50:        uint256 mintAmount;
51:        if (totalSupply == 0) {
52:            mintAmount = apeXAmount * 1000;
53:        } else {
54:            mintAmount = apeXAmount.mulDiv(totalSupply, apeXBalance);
55:        }
56:
57:        TransferHelper.safeTransferFrom(apeXToken, msg.sender, address(this), apeXAmount);
58:        _mint(to, mintAmount);
59:        return mintAmount;
60:    }

119:     function _mint(address to, uint256 value) internal {
120:         totalSupply = totalSupply + value;
121:         balanceOf[to] = balanceOf[to] + value;
122:         emit Transfer(address(0), to, value);
123:     }
```

Description

yekong : The mint function does not check the 0 address of the incoming address, and the newly issued token may enter the 0 address,Pledged apeXToken is permanently locked
Hellobloc : `Banana` contract's `_mint` lacks zero address checksum for `to` addresses

```
function _mint(address to, uint256 value) internal {
        totalSupply = totalSupply + value;
        balanceOf[to] = balanceOf[to] + value;
        emit Transfer(address(0), to, value);
    }
```

This can result in `mint` events having the same event content as `burn`, which ultimately causes misidentification by the `dex tool` under the chain.

Recommendation

yekong : Add 0 address check
Hellobloc : We recommend implementing the `Banana` contract using the inherited `ERC20` contract to ensure code specification.

Client Response

Accepted and 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.