SECURITY REVIEW OF YEET

Summary

Auditors: 0xWeiss (Marc Weiss)

Client: Yeet

Report Delivered: 11 June, 2024

About 0xWeiss

0xWeiss is the Co-Founder and Lead Security Researcher at Enigma Dark. In-house auditor/security engineer in Ambit Finance and Tapioca DAO. Security Researcher at Paladin Blockchain Security and SR at Spearbit DAO. Reach out on Twitter @0xWeisss

Protocol Summary

Protocol Name	Yeet
Language	Solidity
Codebase	https://github.com/0xKingKoala/contracts
Initial Commit	a89aae41116904eea4e1b5293b9d0be219a6bbd5
Post-Fixes Final Commit	89eaf9d8061084842efe299ea48b9ed78040b9b9

Audit Summary

Yeet engaged **0xWeiss** through **Hyacinth** to review the security of its staking contracts.

A 1 week time-boxed security assesment was performed.

At the end, there were 15 issues identified.

All findings have been recorded in the following report. Notice that the examined smart contracts are not resistant to internal exploit.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

Vulnerability Summary

Severity	Total	Pending	Acknowledged	Par. resolved	Resolved
HIGH	**3**	**0**	**0**	**0**	**3**
MEDIUM	**3**	**0**	**1**	**0**	**2**
LOW	**8**	**0**	**0**	**0**	**8**
INF	**0**	**0**	**0**	**0**	**0**

Severity Classification

Severity	Classification
HIGH	Exploitable, causing loss/manipulation of assets or data.
MEDIUM	Risk of future exploits that may or may not impact the smart contract execution.
LOW	Minor code errors that may or may not impact the smart contract execution.
INF	No impact issues. Code improvement

Methodology

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Audit Scope

Contracts

ID	File Path
ST	contracts/Stake.sol
YEE	contracts/Yeet.sol
NFTV	contracts/NFTVesting.sol
RW	contracts/Reward.sol
YT	contracts/YeetToken.sol
YGS	contracts/YeetGameSettings.sol
H	contracts/History.sol
RS	contracts/RewardSettings.sol
YB	contracts/ Yeetback.sol

Findings and Resolutions

ID	Severity	Status
YB-H1	**HIGH**	Resolved
H-H1	**HIGH**	Resolved
YEE-H1	**HIGH**	Resolved
YB -M1	**MEDIUM**	Resolved
YEE -M1	**MEDIUM**	Resolved
YEE -M2	**MEDIUM**	Acknowledged
ST -M1	**MEDIUM**	Resolved
ST -L1	**LOW**	Resolved
YT -L1	**LOW**	Resolved
YB -L1	**LOW**	Resolved
NFTV -L1	**LOW**	Resolved
YEE -L1	**LOW**	Resolved
YB -L2	**LOW**	Resolved
NFTV -L2	**LOW**	Resolved
RW -L1	**LOW**	Resolved

YB-H1 | The same user can be pushed multiple times to the `yeetsInRound` array.

Technical Details

The formula to pick winners of the Yeetback consists of is broken because users are able to be added multiple times to the `yeetsInRound` array.

A user can game his odds and increase them exponentially of wining a spot on the YeetBack by calling `yeet()` multiple times.

A user that calls `yeet()` once with 1ETH as msg.value will get 1 chance to win the round on the YeetBack. But a user that calls `yeet()` with the `_minimumYeetPoint` which can be as low as `0.0001 ether`, will have 10x the chances of winning the round on the YeetBack as the winner is a random number in the array. The more times your address is included, the more chnages of winning you will have:

```
uint winner = randomDataNumber % nrOfYeets; yeet
uint256 winnings = potValue / 10;
address winnerAddress = yeetsInRound[round][winner];
```

Impact

Flawed formula to pick winners on the YeetBack.

Recommendation

Do not allow users to push their address twice or more in the `yeetsInRound` array. Once a user has been added once, do not include him again.

Developer Response

Fixed at PR

H-H1 | History can be overridden and manipulated

Technical Details

Anyone can override and manipulate the history of past, previous and future yeets given that there is an access control missing on the `push()` function.

The capacity equals a 100 previous rounds that can be stored and then it starts overriden the previous rounds. Anyone can craft their own history by passing the parameters directly:

```
function push(History calldata history) public {
        if (endIndex >= capacity) {
            startIndex++;
        }
        histories[endIndex % capacity] = history;
        endIndex++;
    }
```

Impact

Anyone can override and manipulate the history of past, previous and future yeets.

Recommendation

```
- function push(History calldata history) public {
+ function push(History calldata history) public onlyOwner{  //@auditor ow
ner should be the Yeet contract

        if (endIndex >= capacity) {
            startIndex++;
        }
        histories[endIndex % capacity] = history;
        endIndex++;
    }
```

Developer Response

Fixed at PR

YEE – H1 | NFT rewards can be manipulated using flashloans.

Technical Details

The initial issue starts at the `Yeet` contract, exactly on the `getBoostedValue` function:

```
uint256 boostedValue = getBoostedValue(msg.sender, valueToPot);
```

If we go to the code in the `getBoostedValue` function, we see that the `getNFTBoost()` gets called which returns a boost amount using the current spot balance of NFTs you have in your wallet. After, that number is used on the `return value + (value * nftBoost) / 10000;` calculation to return the boosted value:

```
function getBoostedValue(address sender, uint256 value) public view
returns (uint256) {
        uint256 nftBoost = getNFTBoost(sender);
        return value + (value * nftBoost) / 10000;

 function getNFTBoost(address user) public view returns (uint256) {

        INFTContract nftContract = INFTContract(yeetardsNFTsAddress)
;
        uint256 balance = nftContract.balanceOf(user);

        return nftBoostLookup[balance];
```

This boosted value is then stored on the Rewards contract:

```
rewardsContract.addYeetVolume(msg.sender, boostedValue);
```

which then increases the `userYeetVolume` mapping:

```
userYeetVolume[currentEpoch][user] += amount;
```

which is the mapping direclty correlated to the rewards that the user will be able to claim from the `claim()` function in the `Reward` contract.

Therefore the user can flashloan NFTs from `yeetardsNFTsAddress` in any protocol that allows to flashloan NFTs, and increase their rewards.

Impact

Users can claim more rewards than they should.

Recommendation

Do not use the spot balance of NFTs to calculate rewards, or if you control the NFT address disable transfers. Developer Response

Fixed at PR

YB - M1 | Weak randomness can be ballparked to position yourself across different spots on the array.

Technical Details

There are 3 main arguments that are considered to select the winners on `Yeetback.sol`:

- A random number from pyth oracle cated to uint16, which is topped to `65535` uints.
- block.timestamp
- an `i` which is the current iteration in the for loop from 1 to 10.

Therefore, from the entire calculation, you can know two of the arguments before hand, `i` and `block.timestamp`.

You can know `block.timestamp` in advance because `draftWinners()` can be timed as anyone can call `restart()` on `Yeet.sol` as soon as the round is finsihed, timing the exact `block.timestamp` of the randomization.

```
            uint16 smallNumbers = uint16(randomNumber & 0xFFFF);
            randomNumber >>= 16;

            // Seed a new random number with some entropy from the l
arge random number
            uint randomDataNumber = uint(keccak256(abi.encodePacked(
smallNumbers, i, block.timestamp)));
```

Users can try and game where to position themselves in the array of yeets given that 2 of the arguments to calculate the randomness of the winner can be guessed pre-draft. Having increased chaces of winning.

Impact

Users can try and game where to position themselves in the array of yeets given that 2 of the arguments to calculate the randomness of the winner can be guessed pre-draft. Having increased chaces of winning.

Recommendation

Do not use `block.timestamp`, not as a source of randomnes. It is ok to use another deterministic for of randomnes because you can't guess it before the `yeet()` round is finished (which is when the users can position themselves on the array), but you can guess `block.timestamp`.

Developer Response

Fixed at PR

YEE - M1 | Pausable contracts are in fact not pausable

Technical Details

For safety, Yeet has added whenNotPaused restrictions to multiple contracts But there is no method provided to modify the _paused state If a security event occurs, it cannot be paused at all.

Impact

Pausing functionality on the contracts does not work due to missing an override.

Recommendation

Add the following functions to all the pausable contracts

```
+    function pause() external onlyOwner{
+        _pause();
+    }

+    function unpause() external onlyOwner{
+        _unpause();
+    }
```

Developer Response

Fixed at PR

YEE - M2 | valueToStakers will be lost if there are no stakers

Technical Details

When calling `yeet()` on the `Yeet.sol` contract, at the end, there is a small percentage of the yeeted amount deposited to stakers:

```
stakingContract.depositReward{value: valueToStakers}();
```

Though if no one has staked yet, that `msg.value` will be lost in the contract. This can happen multiple times as long as no one stakes.

```
function depositReward() external payable {
        if (totalSupply == 0) {
            return;
        }
        rewardIndex += (msg.value * MULTIPLIER) / totalSupply;
    }
```

To make sure there is no funds stuck or lost, the protocol team should be the first staker in the `stake.sol` contract before `yeet()` is called once.

Impact

`valueToStakers` will be lost if there are no stakers

Recommendation

To make sure there is no funds stuck or lost, the protocol team should be the first staker in the `stake.sol` contract before `yeet()` is called once

Developer Response

Acknowledged

ST - M1 | User's stakes can be DOS and be unable to withdraw

Technical Details

Every time a user stakes, it is pushed to the array:

```
vestings[msg.sender].push(Vesting(unStakeAmount, start, end));
```

When the same user tries to unstake it does loop through all the stakes to pop the correct one from the array.

If a user, or a contract integrating with Yeet, stakes a lot of times, the `_unstake` function can run out of gas, prohibiting users from withdrawing a single position.

Impact

Users stakes can be DOS and be unable to withdraw if they stake multiple times

Recommendation

Add a maximum amount of times a user can stake. A 30 time limit would be enough so that the function does not run out of gas.

```
   function stake(uint256 amount) external {
        require(amount > 0, "Amount must be greater than 0");
+       require(stakedTimes[msg.sender] < STAKING_LIMIT, "Amount must be g
reater than 0");
        _updateRewards(msg.sender);

        balanceOf[msg.sender] += amount;
        totalSupply += amount;
+       stakedTimes[msg.sender]++ ;
        stakingToken.transferFrom(msg.sender, address(this), amount);
        emit Stake(msg.sender, amount);
    }
```

Developer Response

Fixed at PR

YT - L1 | Ownable is implemented but has no use case

Technical Details

The `YeetToken` contract inherits `Ownable2Step` from OpenZeppelin, but it is not being used.

```
contract YeetToken is ERC20, Ownable2Step {
    constructor()  Ownable(msg.sender) ERC20("$YEET", "YEET") {
        _mint(msg.sender, 100_000_000 * 10 ** uint(decimals()));
    }
}
```

Impact

The `Ownable2Step` inherited contract is not being used

Recommendation

Either remove the `Ownable2Step` inheritance or override some functions like `mint()` and use the modifier.

Developer Response

Fixed at PR

YB - L1 | If `msg.value > fee` the rest, it is not being re-imbursed to the user in `addYeetback`

Technical Details

On the `addYeetback`, when more `msg.value` is sent than the entropy fee, the amount is not re-imbursed to the user, and it can't be withdrawn from the `Yeetback` contract, thus the value will be stuck in the contract.

```
 function addYeetback(bytes32 userRandomNumber, uint256 round, uint2
56 amount) payable public onlyOwner {
        require(userRandomNumber != bytes32(0), "Invalid number");
        require(round != 0, "Invalid round");
        require(amount != 0, "Invalid amount");
        potForRound[round] = amount; //ok

        uint256 fee = entropy.getFee(entropyProvider);
        if (msg.value < fee) {
            revert ("Yeet: Not enough value to pay for entropy fee")
;
        }
        uint64 sequenceNumber = entropy.requestWithCallback{value: f
ee}(
            entropyProvider,
            userRandomNumber
        );
        sequenceToRound[sequenceNumber] = round;
        emit RandomNumberRequested(sequenceNumber);
        emit YeetbackAdded(round, msg.value);
    }
```

Impact

The extra `msg.value` is not being re-imbursed to the user in `addYeetback`

Recommendation

Add an edge case to when the `msg.value` that is not being sent to pyth is re-imbursed to the user.

Developer Response

Fixed at PR

NFTV - L1 | Arithmetic error when claiming

Technical Details

When claiming tokens for an NFT on the `NFTVesting` contract, it is being checked that the vesting period has not started nor finished.

```
require(block.timestamp >= startTime, "NFTVesting: vesting period has not started yet"); require(block.timestamp <= endTime, "NFTVesting: vesting period has ended");
```

But, there is a one-off error when checking whether the vesting period has ended as it should not include the `endtime` as a valid timestamp to claim.

Impact

Users will be able to claim when the vestion period has ended

Recommendation

Update the requirement:

```
- require(block.timestamp <= endTime, "NFTVesting: vesting period has ended");
+ require(block.timestamp < endTime, "NFTVesting: vesting period has ended");
```

Developer Response

Fixed at PR

RW - L1 | Superfluous `_shouldEndEpoch` call.

Technical Details

On the `_endEpoch` function there is a check that `_shouldEndEpoch` should be true for the function to be executed:

```
function _endEpoch() private {
        require(_shouldEndEpoch(), "Epoch not ended");
```

Though the only place where `_endEpoch` gets called is inside the `addYeetVolume` function which already has this requirement.

```
function addYeetVolume(address user, uint256 amount) external onlyYe
etOwner {
        require(amount > 0, "Amount must be greater than 0");
        require(user != address(0), "Invalid user address");

        if (_shouldEndEpoch()) {
            _endEpoch();
        }
```

Impact

Superfluous `_shouldEndEpoch` call

Recommendation

Remove the `require(_shouldEndEpoch(), "Epoch not ended");` check from the `_endEpoch` function.

Developer Response

Fixed at PR

ST - L1 | CEI pattern is not followed

Technical Details

The `stake()` function does not follow the check-effects-iterations pattern as it first updates the balance from the user and then it sends their tokens:

```
        function stake(uint256 amount) external {
        require(amount > 0, "Amount must be greater than 0");
        _updateRewards(msg.sender);

        balanceOf[msg.sender] += amount;
        totalSupply += amount;

        stakingToken.transferFrom(msg.sender, address(this), amount)
;
        emit Stake(msg.sender, amount);
    }
```

Impact

CEI pattern is not followed

Recommendation

Send the tokens from the user before and then update their balances

```
        function stake(uint256 amount) external {
        require(amount > 0, "Amount must be greater than 0");
        _updateRewards(msg.sender);

+        stakingToken.transferFrom(msg.sender, address(this), amount);

        balanceOf[msg.sender] += amount;
        totalSupply += amount;

-        stakingToken.transferFrom(msg.sender, address(this), amount);
        emit Stake(msg.sender, amount);
    }
```

Developer Response

Fixed at PR

YEE – L1 | isRoundFinished can return an invalid state

Technical Details

The `isRoundFinished` function returns whether the round has been finished is `block.timestamp > _endOfYeetTime` , but it does not count that when `block.timestamp == _endOfYeetTime`, the round will also be finished:

```
 function isRoundFinished() public view returns (bool) {
        return block.timestamp > _endOfYeetTime;
    }
```

Impact

`isRoundFinished` can return an invalid state

Recommendation

Update the function as following:

```
 function isRoundFinished() public view returns (bool) {
-        return block.timestamp > _endOfYeetTime;
+        return block.timestamp >= _endOfYeetTime;

    }
```

Developer Response

Fixed at PR

YB – L2 | Invalid state when calling `claim()`

Technical Details

On the `claim()` function, when a user claims their winnings, the winning amount is never reset to 0, which will be incorrect because the user will have claimed all their rewards:

```
  function claim(uint256 round) public nonReentrant {
        require(round != 0, "Yeetback: Invalid round");
        Winner storage winner = winners[round][msg.sender];
        require(winner.amount != 0, "Yeetback: No winnings to claim"
);
        require(!winner.claimed, "Yeetback: Already claimed");

        winner.claimed = true;

        (bool success,) = payable(msg.sender).call{value: winner.amo
unt}("");
        require(success, "Transfer failed.");
        emit Claimed(round, msg.sender, winner.amount);
    }
```

Impact

The winner amount in `claim()` is not reset to 0 after claiming.

Recommendation

```
  function claim(uint256 round) public nonReentrant {
        require(round != 0, "Yeetback: Invalid round");
        Winner storage winner = winners[round][msg.sender];
        require(winner.amount != 0, "Yeetback: No winnings to claim");
        require(!winner.claimed, "Yeetback: Already claimed");

        winner.claimed = true;
+       winner.amount = 0;

        (bool success,) = payable(msg.sender).call{value: winner.amount}("
");
        require(success, "Transfer failed.");
        emit Claimed(round, msg.sender, winner.amount);
    }
```

Developer Response

Fixed at PR

NFTV – L2 | Superfluos Ownership check

Technical Details

On the `claim()` function in `NFTVesting` you can find an initial check that requires the msg.sender to be the owenr of the tokenId you want to claim the rewards from:

```
require(nftContract.ownerOf(tokenId) == msg.sender, "NFTVesting: not the
owner of the NFT");
```

After the check a redundant fetch is done when fetching the owner again at:

```
address owner = nftContract.ownerOf(tokenId);
```

Recommendation

Do remove the superfluous owner check:

```
  function claim(uint256 tokenId) public {

        require(nftContract.ownerOf(tokenId) == msg.sender, "NFTVesting: n
ot the owner of the NFT");

        uint256 endTime = startTime + vestingPeriod;
        require(block.timestamp >= startTime, "NFTVesting: vesting period
has not started yet");
        require(block.timestamp <= endTime, "NFTVesting: vesting period ha
s ended");

        require(claimable != 0, "Nothing to claim");
        require(token.balanceOf(address(this)) >= claimable, "Not enough t
okens in contract");

        claimed[tokenId] += claimable;

-       address owner = nftContract.ownerOf(tokenId);
-       token.transfer(owner, claimable);
+       token.transfer(msg.sender, claimable);
        emit Claimed(tokenId, claimable);
    }
```

Developer Response

Fixed at PR

DISCLAIMER

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Marc Weiss to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk.

My position is that each company and individual are responsible for their own due diligence and continuous security. My goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze. Therefore, I do not guarantee the explicit security of the audited smart contract, regardless of the verdict.