CD SECURITY

AUDIT REPORT

Varonve
July 2024

Prepared by
yotov721
immeas

Introduction

A time-boxed security review of the **Varonve** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **Varonve**

Varonve staking is an NFT staking protocol. A user stakes their Varonve NFT to gain XP. This XP can be used to increaes the rate they with which they gain XP or it can be used to join raffles to win a prize.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 9d0f2482e811c1175185f7aed92270aff028eca9**

Scope

The following smart contracts were in scope of the audit:

- `Staking.sol`

The following number of issues were found, categorized by their severity:

- Critical & High: 14 issues
- Medium: 2 issues
- Low: 0 issues

Findings Summary

ID	Title	Severity
[C-01]	Anyone can spend others XP by calling `spendXP`	Critical
[C-02]	Any user can add XP by calling `addXP`	Critical
[C-03]	`addXP` subtracts XP instead of adding	Critical
[C-04]	User can inflate their XP by abusing `updateXP`	Critical
[C-05]	Calling `returnLoserTicketXPs` will fail	Critical
[C-06]	Several issues in adding player to raffle list	Critical
[C-07]	First-time raffle entries are double counted	Critical
[C-08]	User can buy multiple tickets but pay only for one	Critical
[H-01]	`levelUP` doesn't actually increase the level	High
[H-02]	`spend/addXP` will not add or spend correctly	High
[H-03]	`spendXP` fails if the last NFT lacks balance	High
[H-04]	Level XP multiplier isn't updated on NFT level up	High
[H-05]	`returnLoserTicketXPs` uses wrong price	High
[H-06]	Raffles cannot be created	High
[M-01]	Choosing winner can be manipulated	Medium
[M-02]	Refunds for losing tickets aren't registered	Medium

Detailed Findings

[C-01] Anyone can spend others XP by calling `spendXP`

Severity

Impact: High

Likelihood: High

Description

In `VaronveStaking` there's a call `spendXP`:

```
139:     function spendXP(uint256 amount, address _address) public {
```

Which, as the name suggests, spends XP for the user. This is called from `levelUP` and `buyTicket` to spend cost for the action.

The issue however is that `spendXP` is `public`. Hence it can be called by anyone, letting any user spend XP for any other user.

Recommendations

Consider making `spendXP` internal instead of public:

```
-    function spendXP(uint256 amount, address _address) public {
+    function spendXP(uint256 amount, address _address) internal {
```

[C-02] Any user can add XP by calling `addXP`

Severity

Impact: High

Likelihood: High

Description

Similar to C-01 there is a call `addXP`:

```
159:     function addXP(uint256 amount, address _address) public {
```

This is called by `returnLoserTicketXPs` to return XP to the users.

The issue is however that it is public and can be called by anyone. Hence any user can add XP to their own NFTs.

Recommendations

Consider making `addXP` internal instead of public:

```
-    function addXP(uint256 amount, address _address) public {
+    function addXP(uint256 amount, address _address) internal {
```

[C-03] addXP subtracts XP instead of adding

Severity

Impact: High

Likelihood: High

Description

When `addXP` is called it will add XP back to NFTs:

```
163:             if (stakedNFTs[_address][i].balance >= payPerNFT) {
164:                 stakedNFTs[_address][i].balance -= payPerNFT;
165:                 paid += payPerNFT;
166:             } else {
167:                 stakedNFTs[_address][i].balance = 0;
168:                 paid += stakedNFTs[_address][i].balance;
169:             }
```

However there's a mistake here, on line 164 it subtracts the sum instead of adding it. Causing the NFT to lose XP instead of gaining.

Recommendations

Consider redesigning `addXP` and simply add `payPerNFT` to each NFT:

```
        for (uint256 i = 0; i < stakedNFTs[_address].length; i++) {
-           if (stakedNFTs[_address][i].balance >= payPerNFT) {
-               stakedNFTs[_address][i].balance -= payPerNFT;
-               paid += payPerNFT;
-           } else {
-               stakedNFTs[_address][i].balance = 0;
-               paid += stakedNFTs[_address][i].balance;
-           }
+           stakedNFTs[_address][i].balance += payPerNFT;
+           paid += payPerNFT;
        }
```

As the check was only there to protect from underflow.

[C-04] User can inflate their XP by abusing `updateXP`

Severity

Impact: High

Likelihood: High

Description

A lot of calls in `VaronveStaking` uses the `updateXP` modifier which syncs the XP state of the user:

```
242:                NFTs[i].balance += (
243:                    (((block.timestamp -
NFTs[i].lastBalanceUpdateTime) /
244:                        period) *
245:                        ((xpPerPeriod * NFTs[i].NFTXPMultiplier) +
246:                            ((NFTs.length - 1) * bonusPerNFTStake)))
247:                );
```

Which essentially takes the time between last update and now and then applies some modifiers.

The issue is that `lastBalanceUpdateTime` is never updated. Hence for each call to this it will add XP since the time the user first staked the NFT.

One way of abusing this is to first stake one NFT, then after a while, repeatedly stake and unstake a second NFT. This will cause `updateRewardSingleNFT` to be called multiple times and greatly inflate the XP for the first NFT.

Recommendations

Consider saving the `lastBalanceUpdateTime` at the end of `updateRewardSingleNFT`:

```
            if (NFTs[i].NFTID == id) {
                NFTs[i].balance += (
                    (((block.timestamp - NFTs[i].lastBalanceUpdateTime) /
                        period) *
                        ((xpPerPeriod * NFTs[i].NFTXPMultiplier) +
                            ((NFTs.length - 1) * bonusPerNFTStake)))
                );
+               NFTs[i].lastBalanceUpdateTime = block.timestamp;
                break;
```

[C-05] Calling `returnLoserTicketXPs` will fail

Severity

Impact: High

Likelihood: High

Description

When a raffle finishes the owner will call `returnLoserTicketXPs` which will return the XP to the participants that didn't win.

This will in turn call `addXP` to do the XP state changes:

```
159:     function addXP(uint256 amount, address _address) public {
160:         uint256 paid = 0;
161:         uint256 payPerNFT = amount / stakedNFTs[msg.sender].length;
```

The issue is that `returnLoserTicketXPs` can only be called by owner:

```
402:     function returnLoserTicketXPs(uint256 id) public onlyOwner {
```

This the `msg.sender` can only be the `owner` address which it's unlikely that they will have an NFT staked, thus `stakedNFTs[msg.sender].length` will be `0` and the division will revert. Thus the XP can never be returned back to the stakers. At best the `owner` could stake an NFT, but then the calculation would be wrong as they could need to stake different numbers of NFT to match what the users have.

However, fixing this in the easy way, changing `msg.sender` to `_address` opens up a pretty dangerous griefing possibility. Since a user could unstake their NFT before the call to `returnLoserTicketXPs`. Causing the same divide by zero DoS.

Recommendations

Consider using the NFT balance of the `_address` and checking that the user still has NFTs staked:

```
-       uint256 payPerNFT = amount / stakedNFTs[msg.sender].length;
+       if(stakedNFTs[_address].length == 0) {
+           return;
+       }
+       uint256 payPerNFT = amount / stakedNFTs[_address].length;
```

This will forfeit the XP if the user unstakes before the returns are added.

[C-06] Several issues in adding player to raffle list

Severity

Impact: High

Likelihood: High

Description

1. The check for duplicate address entries iterated over all entries every single time it checks. This makes the gas cost higher the more people join the raffle. The more players join, the higher the gas cost. Eventually gas costs could become so expensive that they could not fit in a single block, or too expensive for users, leading to DoS.

2. Adding element to a storage array like this would not work and revert - this is not possible because the index is not instantiated because storage arrays are not fixed size. Only existing elements can be set like this in for storage arrays.

```
    function addJoinerToList(address _address, uint256 id) internal {
        address[] storage list = joinedAddresses[id];
        for (uint256 i = 0; i < list.length; i++) {
            if (list[i] == _address) {
                break;
            } else {
=>              list[list.length] = _address;
            }
        }
    }
```

3. Another problem is that the user would be added to the list multiple times since when the loop iterates if the `_address` is not at index it would add it on the iteration. For example if the list has 50 addresses and the current is brand new it would add it 50 times.

Recommendations

Add a mapping storing raffleId => address => bool indicating whether the address has already entered the raffle and if he has do not add him in the `joinedAddresses[raffleId]` list again. This would fix the multiple address additions, DoS and invalid index problems.

```
+  mapping(uint256 => mapping(address => bool)) joinedRaffle;
+  error AlreadyJoined();

   function addJoinerToList(address _address, uint256 id) internal {
+    if (joinedRaffle[id][_address]) revert AlreadyJoined();
      address[] storage list = joinedAddresses[id];
-      for (uint256 i = 0; i < list.length; i++) {
-          if (list[i] == _address) {
-              break;
-          } else {
-              list[list.length] = _address;
-          }
-      }
+    list.push(_address);
+    joinedRaffle[id][_address] = true;
   }
```

[C-07] First-time raffle entries are double counted

Severity

Impact: High

Likelihood: High

Description

The protocol offers users the ability to enter a raffle. They can buy multiple tickets for a single raffle. When they enter for the first time a ticket instance is assigned for the user indicating the amount of tickets he owns. The initial amount of tickets bought is assigned directly to the user tickets struct.

```
    function buyTicket(uint256 raffleid, uint256 amount)
        public
        updateXP(msg.sender)
        nonReentrant
    {
        require(raffleid <= raffles.length, "Raffle ID Does not exist");
        raffle storage choosenRaffle = raffles[raffleid - 1]; // points
raffle at the index (raffleid-1).
        require(
            block.timestamp >= choosenRaffle.startTime,
            "Raffle has not started"
        );
        require(block.timestamp <= choosenRaffle.endTime, "Raffle is
ended");
        require(
            showRewards(msg.sender) >= amount * choosenRaffle.price,
            "Your balance is not enough."
        );
        require(amount > 0, "Amount must be greater than zero");

        if (ticketsBought[msg.sender][raffleid].raffleID != raffleid) {
            ticketsBought[msg.sender][raffleid] = ticket(
                msg.sender,
                raffleid,
 =>             amount
            );
        }

        spendXP(choosenRaffle.price, msg.sender);
        choosenRaffle.totalTicketsBought += amount;
=>      ticketsBought[msg.sender][raffleid].amount += amount;
        addJoinerToList(msg.sender, raffleid);

        emit JoinedGiveaway(msg.sender, amount, raffleid);
    }
```

However later in the function the ticket amount is added again effectively doubling the amount of thickets the user bought, making it unfair for other users.

Recommendations

When a user enters for the first time instantiate his amount with 0, because it is incremented after that

```
    function buyTicket(uint256 raffleid, uint256 amount)
      ...SNIP...
        if (ticketsBought[msg.sender][raffleid].raffleID != raffleid) {
            ticketsBought[msg.sender][raffleid] = ticket(
                msg.sender,
                raffleid,
-               amount
+               0
            );
        }

        spendXP(choosenRaffle.price, msg.sender);
        choosenRaffle.totalTicketsBought += amount;
        ticketsBought[msg.sender][raffleid].amount += amount;
        addJoinerToList(msg.sender, raffleid);

        emit JoinedGiveaway(msg.sender, amount, raffleid);
    }
```

[C-08] User can buy multiple tickets but pay only for one

Severity

Impact: High

Likelihood: High

Description

The protocols offers users the ability to spend XP in exchange to enter a raffle. Each raffle has a ticket price and a user may buy multiple tickets for one raffle. When a user calls `BuyTicket` tickets there is a check that verifies the user has enough balance for the amount of tickets.

```
    function buyTicket(uint256 raffleid, uint256 amount)
        public
        updateXP(msg.sender)
        nonReentrant
    {
        require(raffleid <= raffles.length, "Raffle ID Does not exist");
```

```
        raffle storage choosenRaffle = raffles[raffleid - 1]; // points
raffle at the index (raffleid-1).
        require(
            block.timestamp >= choosenRaffle.startTime,
            "Raffle has not started"
        );
        require(block.timestamp <= choosenRaffle.endTime, "Raffle is
ended");
        require(
=>          showRewards(msg.sender) >= amount * choosenRaffle.price,
            "Your balance is not enough."
        );
...SNIP...
```

However when a user is charged for the tickets he is charged only for one ticket.

```
...SNIP...

=>      spendXP(choosenRaffle.price, msg.sender);
        choosenRaffle.totalTicketsBought += amount;
        ticketsBought[msg.sender][raffleid].amount += amount;
        addJoinerToList(msg.sender, raffleid);

        emit JoinedGiveaway(msg.sender, amount, raffleid);
    }
```

Recommendations

Charge users for all the tickets they buy.

```
-       spendXP(choosenRaffle.price, msg.sender);
+       spendXP(choosenRaffle.price * amount, msg.sender);
```

[H-01] `levelUP` doesn't actually increase the level

Severity

Impact: Medium

Likelihood: High

Description

When calling `levelUP` a user looks to spend XP to level up their NFT:

```
122:    function levelUP(uint256 id)
123:        public
124:        nonReentrant
125:        isStakerOfAll(viewStakedNFTs(msg.sender))
126:        updateXP(msg.sender)
127:    {
128:        if (stakedNFTs[msg.sender][getIndexOfItem(id)].NFTLevel == 1)
{
129:            spendXP(levelTwoPrice, msg.sender);
130:            stakedNFTs[msg.sender][getIndexOfItem(id)].balance++;
131:        } else if (stakedNFTs[msg.sender][getIndexOfItem(id)].NFTLevel
== 2) {
132:            spendXP(levelThreePrice, msg.sender);
133:            stakedNFTs[msg.sender][getIndexOfItem(id)].balance++;
134:        } else {
135:            revert("Your NFT reached max level.");
136:        }
137:    }
```

However, the call never actually increases the level. As you can see on lines 130 and 133 it increases `balance` instead of `NFTLevel`.

Hence the user is still spending their XP but their NFTs level is never increased and the balance increase is negligee.

Recommendations

Consider increasing `NFTLevel` instead of `balance`:

```
-           stakedNFTs[msg.sender][getIndexOfItem(id)].balance++;
+           stakedNFTs[msg.sender][getIndexOfItem(id)].NFTLevel++;
        } else if (stakedNFTs[msg.sender][getIndexOfItem(id)].NFTLevel ==
2) {
            spendXP(levelThreePrice, msg.sender);
-           stakedNFTs[msg.sender][getIndexOfItem(id)].balance++;
+           stakedNFTs[msg.sender][getIndexOfItem(id)].NFTLevel++;
```

[H-02] `spend/addXP` will not add or spend correctly

Severity

Impact: High

Likelihood: Medium

Description

Using `spendXP` as an example, the pattern in `addXP` but our recommendation there is to change the pattern, hence it applies more to `spendXP`:

If the balance of the NFT is not enough just the balance of the NFT is removed, to prevent underflow:

```
150:                stakedNFTs[_address][i].balance = 0;
151:                spent += stakedNFTs[_address][i].balance;
```

This is however done in the wrong order. Since the `balance` is set to `0` first the spent will not increase.

This then impacts the calculation at the end where the delta is accounted for:

```
        stakedNFTs[_address][stakedNFTs[_address].length - 1]
            .balance -= (amount - spent);
```

`amount - spent` will be to high here while the `balance` of the NFT was actually spent.

Imagine this scenario:

A user has two NFTs, with `20_000` and `10_000` XP. They spend `22_000` XP. `amountPerNFT` will then be `11_000`. After the first iteration of the first NFT, its balance will be `9_000` and `spent` will be `11_000`. The second iteration will go into the else clause. The balance will be set to zero (effectively spending `10_000` XP), `spent` will be increased by `0`.

Then at the last calculation `amount` was `22_000`, `spent` is `11_000`. This means that it will try to remove `amount - spent = 11_000` more, even though what is actually spent is `11_000 + 10_000 = 21_000`.

Recommendations

Consider removing the balance before setting it to 0 in `spendXP`. Also consider redesigning the design used here. Another possible way is removing a proportional amount from each NFT (`(amount * nft balance)/total rewards`, Then iterating over the NFTs again from the top removing the delta (`amount - spent`) from the first possible one.

In `addXP` use the recommendation in C-03. Simply add `payPerNFT` to each NFT.

[H-03] `spendXP` fails if the last NFT lacks balance

Severity

Impact: High

Likelihood: Medium

Description

When spending XP in `spendXP` an equal amount `amountPerNFT` is subtracted from each NFTs XP:

```
143:         uint256 spent = 0;
144:         uint256 amountPerNFT = amount / stakedNFTs[_address].length;
145:         for (uint256 i = 0; i < stakedNFTs[_address].length; i++) {
146:             if (stakedNFTs[_address][i].balance >= amountPerNFT) {
147:                 stakedNFTs[_address][i].balance -= amountPerNFT;
148:                 spent += amountPerNFT;
149:             } else {
150:                 stakedNFTs[_address][i].balance = 0;
151:                 spent += stakedNFTs[_address][i].balance;
152:             }
153:         }
154:
155:         stakedNFTs[_address][stakedNFTs[_address].length - 1]
156:             .balance -= (amount - spent);
```

If the NFT doesn't have that much XP all of its XP is removed (see H-02 for another issue with this). This sets it to balance `0`. Then at the end, the delta between the amount actually deducted and the amount that should be deducted (`amount - spent`) is removed from the last NFT.

The issue here is that if the last NFT didn't have enough XP, it's balance will be `0` and `amount - spent` will be `!= 0`. Thus this will always revert when the last NFT doesn't have enough XP. The last NFT is also always the latest one which will always have the least XP making this more likely.

Recommendations

Similar to the recommendation in H-02: Consider redesigning the design used here. Another possible way is removing a proportional amount from each NFT `(amount * nft balance)/total rewards`, Then iterating over the NFTs again from the top removing the delta (`amount - spent`) from the first possible one.

[H-04] Level XP multiplier isn't updated on NFT level up

Severity

Impact: High

Likelihood: Medium

Description

Users have the ability to stake their NFTs to get XP in exchange. They can then use that XP to enter a raffle or level up their NFTs. Depending on the level of the NFT it has different multiplier bonuses. The problem is that when an NFT is level up the multiplier bonus is NOT updated.

```
    function levelUP(uint256 id)
        public
```

```
        nonReentrant
        isStakerOfAll(viewStakedNFTs(msg.sender))
        updateXP(msg.sender)
    {
        if (stakedNFTs[msg.sender][getIndexOfItem(id)].NFTLevel == 1) {
            spendXP(levelTwoPrice, msg.sender);
            stakedNFTs[msg.sender][getIndexOfItem(id)].balance++;
        } else if (stakedNFTs[msg.sender][getIndexOfItem(id)].NFTLevel ==
2) {
            spendXP(levelThreePrice, msg.sender);
            stakedNFTs[msg.sender][getIndexOfItem(id)].balance++;
        } else {
            revert("Your NFT reached max level.");
        }
    }
```

There are state variables for the NFT level which are not used anywhere:

```
    uint256 levelTwoMultiplier = 2;
    uint256 LevelThreeMultiplier = 3;
```

Recommendations

Update `NFTXPMultiplier` when leveling up NFT:

```
    function levelUP(uint256 id)
        public
        nonReentrant
        isStakerOfAll(viewStakedNFTs(msg.sender))
        updateXP(msg.sender)
    {
        if (stakedNFTs[msg.sender][getIndexOfItem(id)].NFTLevel == 1) {
            spendXP(levelTwoPrice, msg.sender);
            stakedNFTs[msg.sender][getIndexOfItem(id)].balance++;
+           stakedNFTs[msg.sender][getIndexOfItem(id)].NFTXPMultiplier =
levelTwoMultiplier;
        } else if (stakedNFTs[msg.sender][getIndexOfItem(id)].NFTLevel ==
2) {
            spendXP(levelThreePrice, msg.sender);
            stakedNFTs[msg.sender][getIndexOfItem(id)].balance++;
+           stakedNFTs[msg.sender][getIndexOfItem(id)].NFTXPMultiplier =
LevelThreeMultiplier;
        } else {
            revert("Your NFT reached max level.");
        }
    }
```

[H-05] `returnLoserTicketXPs` uses wrong price

Severity

Impact: High

Likelihood: Medium

Description

`returnLoserTicketXPs` is called by the owner to return the XP for the ticket owners that didn't win the raffle:

```
402:    function returnLoserTicketXPs(uint256 id) public onlyOwner {
403:        uint256 index = id - 1;
404:        require(losersPaidBack[id] == false, "Ticket prices already
paid back");
405:        require(raffles[index].endTime >= block.timestamp);
406:        uint256 valueToReturn;
407:        for (uint256 i = 0; i < joinedAddresses[id].length; i++) {
408:            if (isWinnerOf[joinedAddresses[id][i]][id] == false) {
409:                valueToReturn =
410:                    ticketsBought[joinedAddresses[id][i]][id].amount *
411:                    ((9 * raffles[id].price) / 10);
412:                addXP(valueToReturn, joinedAddresses[id][i]);
413:            } else if (isWinnerOf[joinedAddresses[id][i]][id] == true)
{
414:                valueToReturn =
415:                    (ticketsBought[joinedAddresses[id][i]][id].amount
- 1) *
416:                    ((9 * raffles[id].price) / 10);
417:                addXP(valueToReturn, joinedAddresses[id][i]);
418:            }
419:        }
420:    }
```

`raffles` uses indexes on `id - 1` as seen on line 403.

The issue is that later `raffles[id]` is used. This will cause the price from a different raffle to be used.

Recommendations

Consider using `index` everywhere:

```
                valueToReturn =
                    ticketsBought[joinedAddresses[id][i]][id].amount *
-                   ((9 * raffles[id].price) / 10);
+                   ((9 * raffles[index].price) / 10);
                addXP(valueToReturn, joinedAddresses[id][i]);
```

```
            } else if (isWinnerOf[joinedAddresses[id][i]][id] == true) {
                valueToReturn =
                    (ticketsBought[joinedAddresses[id][i]][id].amount - 1)
*
-                   ((9 * raffles[id].price) / 10);
+                   ((9 * raffles[index].price) / 10);
```

[H-06] Raffles cannot be created

Severity

Impact: Medium

Likelihood: High

Description

Similar to C-06 in solidity you cannot add elements to an array by assigning that index:

```
323:        raffles[raffleCounter - 1] = raffle(
324:            raffleCounter,
325:            _raffleName,
326:            _rewardAmount,
327:            block.timestamp,
328:            _endTime,
329:            _price,
330:            0,
331:            _image
332:        );
```

This causes an `array out-of-bounds` panic.

Recommendations

We recommend you change to use push:

```
-       raffles[raffleCounter - 1] = raffle(
+       raffles.push(raffle(
            raffleCounter,
            _raffleName,
            _rewardAmount,
            block.timestamp,
            _endTime,
            _price,
            0,
            _image
-       );
```

```
+       ));
        raffleCounter++;
```

[M-01] Choosing winner can be manipulated

Severity

Impact: Medium

Likelihood: Medium

Description

The protocols offers users the ability to enter a raffle by buying tickets. The problem is that the winners of the raffle are input by the owner.

```
    function insertWinnerAddresses(address[] memory _raffleWinners,
uint256 id)
        public
        onlyOwner
    {
        for (uint256 i = 0; i < _raffleWinners.length; i++) { // iterate
over input array
            isWinnerOf[_raffleWinners[i]][id] = true; // set winner to
true
        }
    }
```

This is problematic since the owner can chose winners

Recommendations

Use Chainlink VRF instead to chose the winners.

[M-02] Refunds for losing tickets aren't registered

Severity

Impact: High

Likelihood: Low

Description

When users buy tickets for a raffle and don't win, 90 % of the loosing tickets value is paid back to users in the form of XP. This is done be calling the `returnLoserTicketXPs` function. The function checks if the loosing tickets for the raffle are already paid and if they are the transaction reverts.

```
require(losersPaidBack[id] == false, "Ticket prices already paid back");
```

The problem is that the `losersPaidBack` is never set to `true` when the refund is done.

Recommendations

Set `losersPaidBack` to `true` when they are repaid:

```
    function returnLoserTicketXPs(uint256 id) public onlyOwner {
...SNIP...
        for (uint256 i = 0; i < joinedAddresses[id].length; i++) {
...SNIP...
        }
+     losersPaidBack[id] = true;
    }
```