Contents

VDICE October 2016 Audit

Audit Prepared by Peter Vessenes, October 2016

Discussion

Vdice requested an audit of the current codebase. The codebase has been in development for some time, worked on most extensively by engineers at Oraclize.

In general, the vdice contracts are intended to provide two layers of earnings: first to "investors", a group of participants that can stake the capital needed by the casino in exchange for a share in the reward (and risk), and the "house", an entity that accrues a share of the investor profits.

Generally, the code is vastly improved from the last audit. Underflows and overflows are checked for carefully, decorators are used liberally, public vs private functions are carefully considered; all these increase the safety and correctness of the code substantially.

Participants in the vdice smart contract bear a small amount of risk from the 'house' being a bad actor, and from a bad actor internal to Oraclize. Some details are contained in the comments below. Vdice and Oraclize have responses about these risks.

Vdice has made no representations about their plans for publishing on the ETC-chain. Accordingly I have not evaluated the contracts for cross-chain replay risks.

In general, there are two risks bettors take with the smart contract, detailed below.

Disclaimer

The audit makes no statements about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bugfree status. The audit documentation is for discussion purposes only.

Oraclize Response

Oraclize and vdice have shared informal responses to most of these questions. Look for them to publish when ready.

Broad Concerns

Many functions use `uint` as a return type. This is appropriate in some circumstances, but can cause problems if there is a risk of underflow. For instance, can investorProfit ever be less than 0? As it stands, a fair amount of code is devoted to worrying about this, by testing for underflows.

If it could underflow through process or error, then the smart contract itself is going to be hard to manage due to the copious error checking.

If instead it were an integer, then most logic checks would work, and you could relax some of the underflow / overflow concerns. This is not an urgent concern, unless you figure out how to underflow the number, but I currently recommend to clients that numbers like this be typed `int` to increase resilience.

The "house" has some control of the contract. The house can trigger the emergency at will, and can also increase the safe gas limit, allowing for some recursive calls to be made, but I do not see significant benefits to the house for a recursive call here.

Oraclize has some previously published attack vectors, in particular, they can replay winning (or losing) bets at random.org until the desired number is returned, and offer that TLS-Notarized proof. This would not be incentivized at a corporate level, that is, I do not believe Oraclize would gain financially from pursuing this strategy. But, it may be incentivized at an individual employee level; consider an Oraclize employee that is a victim of extortion.

Other than that, I have no broad concerns about the codebase.

Critical Errors

I identified no critical errors in the codebase.

Moderate to Low Risk Errors

I believe that line 614 is incorrect, instead it should test if `newHouse` is equal to 0. **update** this has been fixed by Oraclize.

There are few ways the owner can grief the casino – by allowing enough gas to make recursive calls a risk, or by repeatedly recalling the emergency proposal functions. Neither of these seem like major issues. More in the Developer Log Section below.

Events should be clearly observable as events through a naming convention. Standard solidity practice is to make the first letter capitalized. This is insufficient

in my opinion. Instead of `Bet` being the event, I recommend `LOG_BET` or `EMIT_BET` to help developers understand that no logic is being executed.

I identified no other moderate or low risk errors.

Tiny Annoyances

Grammar: 'betted' doesn't read well in English. I might suggest 'bet'. **update** this has been fixed by Oraclize.

Log of Developer Comments and Specific Line By Line Inquiries

L148

Are there circumstances in which you would want to invalidate a bet later, after it's processed? If so, this logic may need some work.

getBankroll

Are there any circumstances in which investorProfit will be less than investorLosses? If so, this will report a lie: it will claim bankroll is 0 when it's actually negative.

If there might be, the function should return an int and the logic should be different. I generally would recommend these types of numbers be 'ints' instead of uints because failure modes are less spectacular when something underflows.

Investorlosses, profits and accrual

I'm a little worried that there's some problem between investorlosses, investorprofits, and accrual of them all when investors change. Are you sure that there is no moment when an unaccrued loss or gain is skipped somehow when investors come into or leave the system for any reason?

update Oraclize responds that there is no such moment.

House Always wins?

Is it your intent to accrue all losses to investors and only accrue earnings to the house? That's how it's currently written in isWinningBet/isLosingBet.

L330

The Owner can change oraclize config any time. I *think* this means if the owner is mining, they can interrupt a bet mid-flight. This is a minor risk, lots of moving parts. Lots of them. **Update** Oraclize states that `setConfig` does not currently do anything. This line should be removed until there is needed functionality at Oraclize.

L384

Lots of underflow checks. Nice. It would be great to somehow log an event that the casino is out of money though.

L449

Why is this necessary?

Throwing during a losing bet

So the callback can throw during a winning or losing bet being processed (L436). This is an interesting idea, what happens to the casino then? Will oraclize re-offer the callback at some point? Is a bet that breaks the bank just 'gone'? **update** Oraclize responds that this a good idea.

L148

If there are circumstances in which you'd want to invalidate a bet later, after it's stored in `bets[]`, you are going to have a variety of problems with the contract.

L192

Check what happens if invalid number but processed?

L482

There's a chance for a rounding error here between the two. Better would be subtracting investorsProfit. **update** Oraclize says that up to 1 wei could leak here and has fixed this very tiny bug.

L535

What about a run on the casino when it's not fully funded? Smaller investors can get out easier I think. If their pro-rata share is less than the contract balance, they'll get out, no? Better would be, well, I don't know about better. That possibility isn't considered here. **update** Oraclize says that losses would be distributed equally through `profitDistribution()`.

L634

Owner can tee up a recursive call with this function. This seems low risk because the public attack surface is low; essentially only `divest` and `bet`.

Emergencies

As it stands, the owner can reset emergency withdrawal forever by just calling the function over and over, making sure money never goes to the emergency address. I'm not sure why this would happen though in the real world.