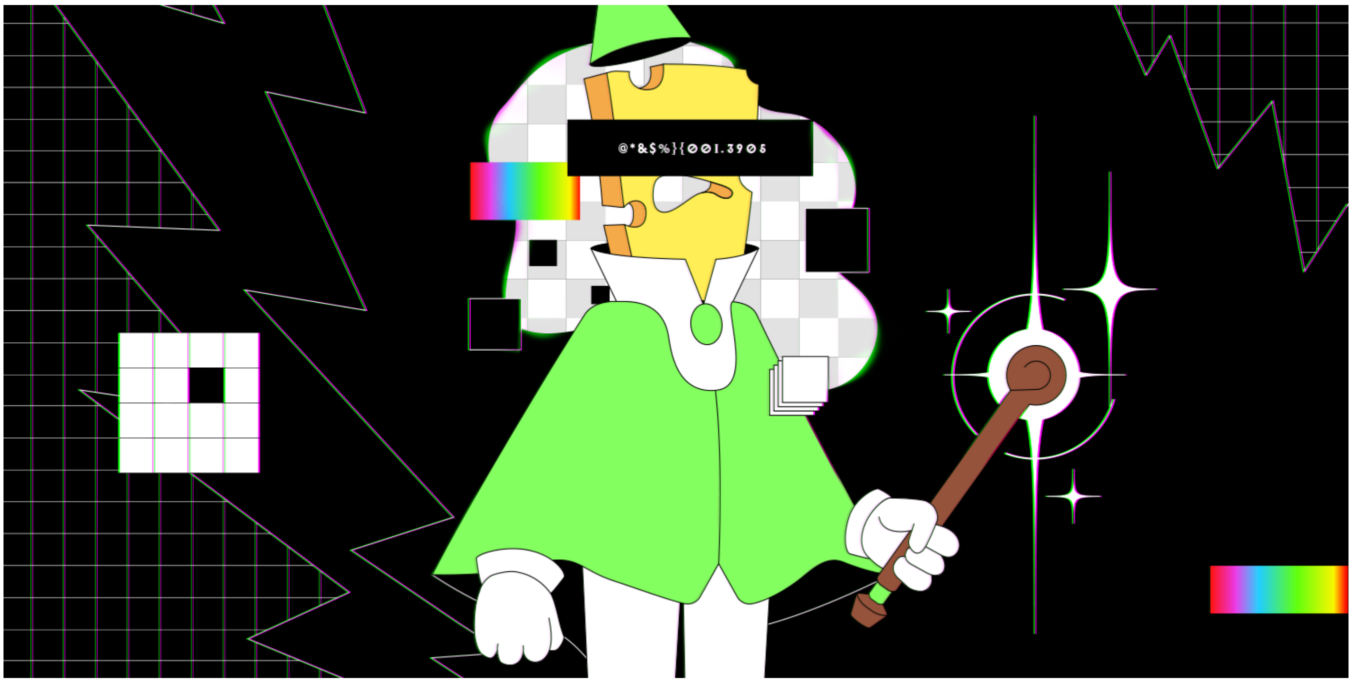
Disclosure: Forking Cheeze Wizards Smart Contracts, All Funds (and Wizards!) are Secure

Some players will get an extra Wizard in the tournament contract we're calling Cheeze Wizards: Unpasteurized

Cheeze Wizards 🧀
Oct 17, 2019 · 6 min read



Well... CRAP!

Within 24 hours of letting our player community know about Cheeze Wizards hitting Ethereum mainnet, one of our community members alerted our security auditors to a bug in the smart contract. The best part of the crypto community is how collaborative people are: A big thanks to @samczsun for responsibly disclosing the issue and making the game better for everyone.

The great news is that as a player, you're mostly unaffected. Sales for the main tournament reopens tomorrow October 18th, your Wizards will be waiting for you: just head to cheezewizards.com and log in with your wallet.

So what's the bad news?

The Cheeze Wizards smart contacts were carefully designed so that we have no ability to withdraw the prize pool (or otherwise alter any of the Tournament state) once they are published. As a result, the only way we can fix the bug is by deploying a new contract.

There are wizards belonging to 150 wallet addresses already in the vulnerable smart contract, totaling 175 ETH worth of power. This ~$40,000 is now locked in this contract, and the only way anyone can get it out is by fighting a Cheeze Wizards tournament with the current battle logic.

Last block: 8,759,737
Only a few Wizards were affected they would all be below: #6133
Txhash for reference:
https://etherscan.io/tx/0x0d497ea959406909edad945d332d0aa1ed2a41273c694ad385910720af2f86f3

The good news is that the bug isn't a complete deal-breaker. We can still let the vulnerable tournament run, it'll just be one where fighting dirty is allowed. **We call it Cheeze Wizards: Unpasteurized**.

What is Cheeze Wizards: Unpasteurized?

Unpasteurized is what we are calling the version of the Cheeze Wizards smart contracts we deployed on October 14, 2019.

CW: Unpasteurized includes a bug that can be exploited to steal power from honest players, especially those who access the game using a web interface. However, it occurred to us that this bug may also make the game *more* fun to play for certain classes of technically-minded folks who like to get in touch with their dark side from time to time.

Here's what we mean.

Normal duel:

1. Player A challenges Player B
2. Player B accepts and submits a move commitment

3. Player A responds and submits a move commitment
4. Player A reveals moves
5. Player B reveals moves
6. Duel is resolved on the smart contract
7. Duel animation is generated on cheezewizards.com and the players find out the results.

Exploiting the "Dead Ringer Attack":

1. Player A challenges Player B
2. Player B accepts and submits a move commitment
3. Player A responds and submits a move commitment
4. Player A reveals moves
5. **Player B deliberately times out without revealing (a 90-minute window)**
6. **Player B calls the `resolveTimedOutDuel` (rTOD) function with malformed parameters, providing Player A's Cheeze Wizard in both the slots.**

```
NORMAL: function resolveTimedOutDuel(WIZARD-A, WIZARD-B)

EXPLOIT: function resolveTimedOutDuel(WIZARD-A, WIZARD-A)
                                                ^^^^^^^^
                                              THE BAD PART
```

This will wipe out Wizard A's power while also putting Wizard B in an invalid state. However, calling `resolveTimedOutDuel(WIZARD-B, WIZARD-B)` would fix that bad state. So, Wizard A's power is wiped out, while Wizard B's power is untouched. (Note that there is no power transfer from A to B during the exploit).

Interestingly enough, a malicious third party could also call rTOD to wipe out Wizard A's power. It doesn't necessarily have to be Player B who triggers the exploit!

So… How could that be "fun"?

As described above, it seems like Player A is always the victim here. They reveal their moves correctly and get their power drained. Sure, Player B doesn't get to absorb that power, but they've wiped out their opponent and get to live on to fight another day.

Except…

Player B is taking a big risk by not revealing any moves. By the rules of CW, not revealing your moves (once committed) is an automatic loss. If an *honest* call to rTOD — one that correctly uses Wizard A and Wizard B as arguments — is processed before the malformed call to rTOD, Wizard B will be the one that gets drained… and Wizard A will absorb all of its power!

So we have a game of Wizard Chicken here. If a player suspects that their opponent will try to use the Dead Ringer Attack against them, they can reveal their moves and then it's a race to see who can get their rTOD in first. And the honest player has a bigger reward if they win that race, because they get to absorb their opponent's power; someone using the exploit can only drain their opponent and won't ever increase their power.

Oh, and don't forget that we are running a set of server daemons that will fire honest rTODs automatically. 😼

So… are you game?

Let me be absolutely clear: *CW: Unpasteurized* is not for everyone. You should be fully aware of the risks. We suspect many folks will be running automated scripts to try to play both angles of this issue.

Think you've got the guts to handle unpasteurized milk? Suit yourself. We hope you will grow strong bones from this experience. **Join our Discord** and look for the #Unpasteurized channel to share your experience.

Starting now you can buy Unpasteurized Wizards on **unp.cheezewizards.com.** (Buyers beware: these Wizards will be a part of the tournament that has the known exploit!)

Sales on CheezeWizards.com (using the patched contract) will restart next week.

Let's Get Technical

Strap in, we're going deep!

The issue

To handle the case where a user either accidentally or maliciously doesn't submit their revealed moves after committing to a battle, our smart contract allows a user to do a "one-sided reveal" where they submit their moves independently from the other user. This is an exceptional occurrence, only used when one user goes dark. By default, our interface uses a "double reveal", which is more gas efficient.

The rTOD exploit can only be triggered in the case where a duel has a one-sided reveal and has also timed out. It is very easy for a participant of a duel to arrange this: They just need to provide their commitment and then drop off the network.

Once there is a timed-out duel with a single reveal, *any* malicious user can submit a malformed transaction to drain the Wizard that revealed their moves.

Line by line

Assume that Honest Hermione is using Wizard #1000 to battle Byzantine Belinda, who is using Wizard #2000 and uses the Dead Ringer Attack. Both Wizards commit their moves and enter into a duel. Honest Hermione reveals her moves, while Belinda waits for the duel to time out and calls resolveTimedOutDuel(1000, 1000). Let's walk through that function call in the smart contract.

```
function resolveTimedOutDuel(uint256 wizardId1, uint256 wizardId2)
{
  BattleWizard storage wiz1 = wizards[wizardId1];
  BattleWizard storage wiz2 = wizards[wizardId2];

  bytes32 duelId = wiz1.currentDuel;

  require(duelId != 0 && wiz2.currentDuel == duelId);
```

The smart contract checks that both Wizards are taking part in the same duel, but assumes that the two arguments refer to different Wizards, *without explicitly checking*. Unsurprisingly, using the same Wizard ID for both arguments passes the check of having the same duel ID.

```
if (revealedMoves[duelId][wizardId1] != 0)
```

Because the target of this attack (Wizard 1000) has revealed their moves, they pass this check to allow for a power transfer.

```
if (revealedMoves[duelId][wizardId1] != 0) {
  // The first Wizard revealed their moves, but the second
  // one didn't (otherwise it would have been resolved).
  // Transfer all of the power from two to one.

  _updatePower(wiz1, allPower);
  wiz2.power = 0;
}
```

Finally, the smart contract executes a power transfer where it thinks it is giving all power to the winning Wizard, then draining it all from the losing Wizard. However, since *both* `wiz1` and `wiz2` are references to the same Wizard in storage, so we are first doubling its power, and then draining its power to zero. 🤦‍♀️ 🤦‍♂️

Thankfully, this bug is easily fixed by adding a simple require statement at the top of the function to make sure that the two Wizards IDs are different.

```
require(wizardId1 != wizardId2, “Same Wizard”);
```

It's worth noting that these smart contracts have undergone formal security reviews by Sigma Prime and we're confident there are no further issues to keep the tournament from running as expected.

Awesome! How do I get involved?

If you're a hacker looking to get a little dirty (or excited about the opportunity to turn the tables on someone trying to get dirty themselves!), jump over to unp.cheezewizards.com. But don't say we didn't warn you… this is dangerous stuff! There will be upset stomachs and wallets.

For you hackers who prefer to keep it clean, the best way to participate is to join the CheezyVerse! Join the dozens of other teams and build useful tools for players in the main tournament.

If you're not a developer, sign up for an account on cheezewizards.com and just hang tight — we'll email you once sales reopen for the main tournament.

Ethereum Cheeze Wizards Smart Contracts Blockchain Dapper Labs

About Help Legal