Augur REP Token Critical Vulnerability Disclosure

OPENZEPPELIN SECURITY | JULY 28, 2017

Security Audits

> Thank you for your interest in this post! We're undergoing a rebranding process, so please excuse us if some names are out of date.

Two weeks ago, we finished our audit of the Serpent compiler and sent it privately to the Augur team. We found the Serpent project to be of very low quality, with 8 critical severity vulnerabilities.

A few days later, we found one of those critical vulnerabilities affected the production REP token. In a nutshell, **an out-of-bounds write on the token's `reputation` array allowed an attacker to modify the `creation timestamp`, making it believe the crowdsale was still ongoing, and disabling all token transfers**. This critical severity vulnerability, if exploited, could have halted the whole REP economy, worth over $200 million.

The Complete Story

We contacted the Augur team on July 13th 2017 to privately notify them about the issue. It is important to note that the critical vulnerability was in the Serpent compiler's code, not in Augur's code. We proposed a mitigation plan to reduce damages to the Augur project, which was accepted by their team. It included:

1. **Writing a new REP smart contract in Solidity**, based on OpenZeppelin's reusable components.
2. **Auditing the new REP token** contract.

5. **Freezing the old REP token** (by exploiting the vulnerability ourselves) on July 25th 2017.
6. **Migrating the balances** of the frozen REP token into the new REP token.

The new token was developed by the Augur team using OpenZeppelin, and audited by the Zeppelin Solutions team. Deployment, migration and related scripts were also audited by the Zeppelin Solutions team.

REP Vulnerability Explained

To understand how the old REP token could be frozen, we first need to understand the Serpent critical severity vulnerabilities used in the attack.

First, Serpent contracts can overwrite storage locations when accessing arrays out of bounds. This means that if a Serpent contract attempts to access an array at a position greater than the array's length, Serpent won't stop it.

Second, the Serpent language is untyped. It allows any operation to be performed on any data. Every value is a 256-bit sequence which can be used as an address, a contract, an integer, or an array. Moreover, it performs no checks on the data sent by a user on a transaction.

This got us thinking if there was a way to force the REP contract to overwrite a storage location by sending unexpected data in one of its function parameters. Turns out there is.

Here's the old REP token contract source code in case you want to guess how the attack works yourself with the information given so far.

To perform the attack, one must send the REP contract a transfer call, crafting the transaction so that the `receiver` parameter is greater than the length of the `reputation` array (`2**160`). The choice of that length seems reasonable at first sight: after all, ethereum addresses only have 160 bits, right?

But remember Serpent doesn't check types nor array bounds, and the `receiver` parameter is something an attacker has full control of. And given the `receiver` parameter is used to index the `reputation` array in line 60, we can use that to modify the contract's other storage locations.

```
transfer(2**160+4, 1)
```

would have ended up changing the REP token's `decimals` storage variable to 19 (from 18). The REP-freezing transaction is, then:

```
transfer(2**160+6, 10**10)
```

This will increase the `creation` storage variable by `10**10`, making the contract believe that the crowdsale starts in ~314 years, via the `stillCreating` macro defined in line 140. This makes all further `transfer` calls fail due to the check in line 55.

It's important to note that the REP balance needed to perform this attack has a value of 0.0000002145 USD at the time of writing. A single transaction had the power to freeze a $200M+ economy, because of two compiler bugs.

Conclusions

Serpent should not be considered safe to use unless its many problems are fixed. We recommend all projects using contracts written in Serpent to migrate to Solidity using a similar mitigation plan as the one proposed above. Contact us if you need help with this. We also think a formal audit of the Solidity compiler should be performed, and are currently talking with the Ethereum Foundation to make this happen.

As with any software component, smart contracts are prone to vulnerabilities. We believe our industry needs better mechanisms to secure smart contract code.

Related Posts

Beefy Zap Audit

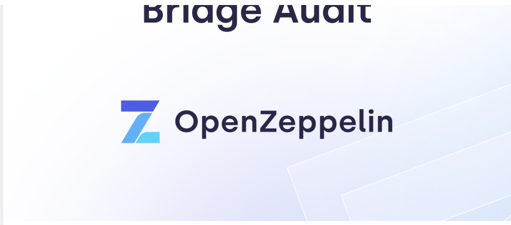
BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits



Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin

Defender Platform

Secure Code & Audit
Secure Deploy
Threat Monitoring
Incident Response
Operation and Automation

Services

Smart Contract Security Audit
Incident Response
Zero Knowledge Proof Practice

Learn

Docs
Ethernaut CTF
Blog

Company

About us
Jobs
Blog

Contracts Library

Docs

Privacy | Terms of Use