LEAF:

Ring Explorer

Write Up v0.1

Author : Harnick Khera
Email: Harnickk@gmail.com
Repo: https://www.github.com/hephyrius/ringexplorer

<u>Project Details</u>:

<u>Document Purpose:</u>

The aim of this document is to provide a brief, high level description of what this project is doing/attempting to do and how it intends to do this.

<u>Project Brief:</u>

The aim of the proposed ring explorer is to provide a user friendly way of exploring transactional data related to the loopring protocol. The gist of the proposed project could be summarised as a blockchain agnostic "Etherscan" for the rings of the loopring protocol.

<u>Timescale</u>**:**

The ring explorer does not have a timescale in the traditional sense. This is because the Ring explorer is highly dependant on the development progression of the Protocol. ideal roadmap for such a project would be for features to coincide with the release of major protocol updates and new blockchain deployments.

Taking this into account, the first milestone on a theoretical roadmap, would be a working and relatively feature rich implementation to coincide with the release of Protocol v1.6.

Subsequent milestones would depend on the development progress of the QTUM and Neo protocol implementations. With the aim of having the explorer ready for use with these implementations, on or near the date that these protocols are to be incorporated into the Loopr Waller.

Prototype Design Considerations:

Data Preface:

The data used by the ring explorer is at the core of the project. Acquiring and storing the data is an important procedure. Within the prototype this is done as a one off operation using the Etherscan API. In a live environment, the acquisition of data would need to be done in a block-by-block fashion.

Ethereum Ring Data:

For the Ethereum implementation, this can done via two methods. The first is by querying the Etherscan api every minute and updating the database based on new entries. The second is by running a GETH or Parity node, and monitoring for new event emissions for deployed contracts.

In terms of operation costs, using the etherscan API is the better choice as this reduces the requirement of having an AWS/Azure instance running a full node. However, in terms of raw speed, having a node readily available and listening for contract events is the better option. This is something that will need to be decided upon later down the line.

Qtum/Neo Ring Data:

This needs to be researched further. An initial concept related to these chains would be to run Nodes and query them regularly in order to populate and update the Sql databases.

Database:

The current prototype implementation makes use of a Mysql database for ring data storage and querying. The database stores all the data in a single table called "web_ring". In future it may be prudent to split up this table into several smaller tables and link them via foreign keys, as the current number of columns, 22, is hard to keep track of. A potential restructure would group together Orders, Block metadata and ringminer details in separate tables.

For now the implementation seems to be working without issue, as reading and writing is working. The true test in this regard will be throughput when the site is under load. If the database becomes a bottleneck then an alternative might need to be found.

The database does not intend to have any sensitive data. All the data held within the database is publicly available on the blockchain. This means that any leaks of the database are non-critical. Despite this basic security methodologies still need to be adopted. This includes but is not limited to Sanitizing input data wherever possible.

Frontend/Backend:

The prototype offers three distinct pages, The Homepage.html, AllRingsData.html and RingBasic.html. The prototype homepage allows the user to search for a ring by looking up its ring index.

The Ring Basic page shows the user the RingMined event outputs of a ring index. In future it will show more, such as the addresses for each order holder. The All Ring Data page provides a table with ring metadata for all stored database rings.

The backend of the ring explorer is powered by Django. Django enforces a very rigid MVC-style framework which is excellent for what the ring explorer is trying to achieve.

The prototype makes use of a Bootstrap frontend. The frontend itself is used as a template by the backend in order to offer dynamic content to the end user. Each html page is a Template. These templates contain elements which Django can take control of and dynamically update.