BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

Platform for Cryptocurrency Address Collection

Vladislav Bambuch*

Abstract
The goal of this work is to build a platform for collecting and displaying metadata about cryptocurrency addresses from public and also dark web. To achieve this goal, the author uses web parsing technologies written in PHP. Challenges accompanying a website parsing are solved by scaling capabilities of Apache Kafka streaming platform. The modularity of the platform is accomplished by microservice architecture and Docker containerization.
The work creates a unique way how to search for potential crypto criminal activities, that appeared outside of the blockchain world, by building a web page application on top of this platform (that serves for managing the platform and exploring the extracted data). The platform architecture allows adding loosely coupled modules smoothly where the Apache Kafka mediates communication of the modules.
The result of this article is meant to be used for cybercrime detection and prevention. Its users can be law enforcement authorities or other agencies interested in reputations of cryptocurrency addresses.
Keywords: web scraping — cryptocurrencies — crypto crime detection — microservices — apache kafka — data streaming

Supplementary Material: Demonstration Video — Downloadable Code

*xbambu03@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

The goal of this work is to build a web-scraping platform for a collection of criminal related cryptocurrency addresses. The platform introduces high-throughput web mining features using data-streaming techniques and stores the extracted metadata into persistent storage for further analysis. On top of the system is a web-based application that is meant to be used by a law enforcement authorities searching for proof of crypto criminal activities.

Using cryptocurrency transactions, a criminal can exchange money for illegal services [1] extremely quickly due to its pseudonymization features. We can only obtain the information about who is an owner of a cryptocurrency address when the person chooses to expose it to the public. Due to the natural behaviour of Internet websites and their dynamic content changes, we need to look up for this type of information regularly. This topic is worthy of exploration as we do not have sufficient tools freely available that would address the mentioned difficulties.

Currently, there are many websites focusing on displaying cryptocurrency addresses[1], some of them also contain owners for particular addresses subset[2] or general discussions about cryptocurrencies[3]. The

[1] https://bitinfocharts.com
[2] https://www.walletexplorer.com
[3] https://bitcointalk.org

data on those websites can help to clarify what happened outside of blockchain and to map pseudonymous crypto-addresses to real users. There are also websites[4] focusing on collecting addresses seen in fraud emails and other forms of extortion activities. On the other hand, we lack a web application that would unify and link all the data from mentioned websites and provide possibility to search in them. The result of this paper aims to fill this blind spot.

Given the fact that there are many websites with interesting data from Open-source intelligence (OSINT) point of view and do not expose any Application Programming Interface (API), this platform enables smooth integration of parsers for particular websites. Those parsers can be implemented as results of school projects or theses and without dependency on a programming language.

The potential amount of data this platform is going to process is enormous. This requires designing a robust architecture that provides parallel data processing using scalability features and fault-tolerant properties. All modules need to be excessively supervised by a monitoring tool, and its metrics displayed in a coherent dashboard. The monitoring enables alerting when some soft-dependency is broken, but without the need of stopping all modules. This approach follows the microservices architecture design [2].

At the output of this platform lies scraped cryptocurrency metadata that can be searched with a simple web-based application on top of the system. The application also allows maintenance of the underling backend layer and scheduling of repetitive jobs. Law enforcement authorities and, in general, all agencies interested in reputations of specific cryptocurrency addresses can use this system.

This work is a part of the project *Integrated platform for analysis of digital data from security incidents*[5], developed at Brno University of Technology, and it is planned to integrate it with its other components.

2. Challenges The Platform Faces

In this chapter, there are explained several challenges the platform is dealing with and how they are solved.

This paper tackles many obstacles from a data collection point of view, as well as using reliable processing and transformation until the results can be monitored, stored and kept in persistent storage. Each of these obstacles has to align with the intention to implement modules loosely coupled, language-independent, and as general as possible.

The following challenges are discussed in this chapter:

- Scraping unstructured data;
- Rate-limiting;
- Browser fingerprinting;
- Processing vast amount of data;
- Sharing parsed data across processes;
- Parallel task execution;
- Data quality of scraped results.

2.1 Cryptocurrency Websites Parsing

The reason why we want to scrape data from public sources is quite simple. We do not have enough information to successfully find relations between data in the blockchain and real-world entities. It is necessary to mention that the extraction of the data is aligned with OSINT initiative, and therefore, all of the information is publicly available. That being said, privacy issues are not related to this topic.

The idea is to collect as much data as possible and to perform it fast and frequently. The disadvantages to this idea are application security measures, such as rate-limiting, browser fingerprinting or traffic throttling. There are also issues with dynamic structural changes of the websites and currently highly popular isomorphic web applications [3].

The limitation of requests a website is receiving can be overcome by IP address pool and performing requests to a single website on behalf of different IP addresses. This behaviour can be accomplished by the help of proxy servers and rotation of addresses they expose.

Browser fingerprinting is a technique employing HTTP headers like `User-Agent`, `Connection`, `Encoding` or `Language` and other connection parameters to find a network entity even if it swaps IP addresses. The `User-Agent` is the essential header in this scenario and therefore changing its value is an effective way how to deal with this challenge [4].

Both of the mentioned principles are handled by a proxy module. Implementation of the module is delivered by project Lemmit that was created as a result of thesis *Automated Web Analysis and Archivation* [5] which is integrated into the platform. The project also consists of archival module executing JavaScript code and extracting fully-loaded Document Object Model (DOM). Integration of this feature into the work suppresses the issue with isomorphic websites.

Every scraper in this work is built using Goutte [6]. Goutte is a library for web crawling and scraping that is written in PHP. In the PHP community, Goutte is some-

[4] https://www.bitcoinabuse.com
[5] https://www.fit.vut.cz/research/project/1063/.en

what of standard for scraping techniques. It is truly easy to load a web page, filter some HTML elements, iterate over them and extract all necessary data. This library cannot handle isomorphic web applications, but it is solved by a combination of the mentioned smart proxy server.

Finally, the structural changes of scraped websites are handed by extensive monitoring capabilities of all requests and expecting responses. Therefore, every detected DOM change triggers an alert.

2.2 The Platform Data Processing

As it was mentioned before, the idea is to collect publicly available metadata as fast as possible and without being blocked by application firewalls. To conquer the problem, the author of this platform introduces massive parallel processing using the Apache Kafka streaming platform. An illustration of how particular Kafka APIs communicate together is present in Figure 1.

The system consists of several loosely coupled modules, performing a single task. Every web-scraping module is subscribed to a data stream and listens for incoming messages – URLs that need to be parsed. When a message is processed, its result is sent to an output data stream by which are other modules notified. By this principle, modules can interact asynchronously and share the same information between multiple simultaneously running processes. The core of the described communication is the Apache Kafka, that is described in this section.

Kafka is a distributed streaming platform, allowing to publish and subscribe to particular data flows. It is used for building reactive event-driven applications and also for building data pipelines for reliable communication between systems. Kafka provides high-speed and fault-tolerant data processing for which has been chosen in many enterprise solutions[6].

In comparison to a traditional message-queuing system, Kafka is capable of storing messages persistently and does not delete them when they are read. That enables reproducing the whole sequence of events if needed and reading one message multiple times using different logic of different data consumers [7]. This technology combines two concepts, *queuing* and *publish-subscribe*, and solves their issues using *Kafka Consumer Group*. That means advantages from both models – *in-order delivery* and *parallel processing* – are merged together [8].

This data streaming platform can be operated as a cluster on multiple servers which proves this technology is an excellent choice for fault tolerance and data replication. One cluster stores stream of records aggregated into categories called *topics*. Each record has a *key*, a *value* and a *timestamp* [9].

One Kafka topic can be distributed across multiple partitions. The number of partitions goes hand in hand with the level of parallelism.

[6] https://kafka.apache.org/powered-by

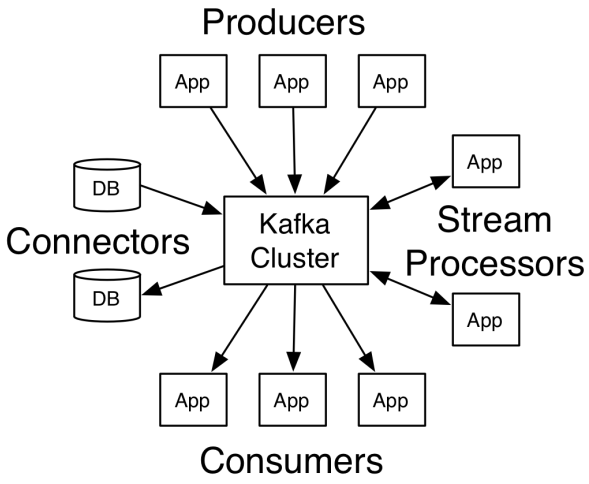

Figure 1. The Kafka architecture diagram is showing how many types of modules can be connected together through this data-streaming platform.

2.3 Storing Crawling Progress

Every website that is supposed to be scraped is crawled at first, and all required URLs are extracted. Individual URL addresses are stored with additional metadata. For example, if a page has been parsed already or if a page is the last one from a sequence of pages. This metadata can be used for running scrapers on appropriate pages or for other robots to know where to start a new round of crawling a website.

Storing the crawling process provides better visibility on performing tasks and speeds up the whole data flow.

3. The Platform Architecture Overview

This chapter contains detailed information about the architecture of the platform and what technologies are used to build loosely coupled, highly modular, data-streaming system.

The platform (depicted in Figure 2) consists of the following modules:

- Web crawlers – collect URL addresses from websites, store them into a database and shares them with other modules through Kafka topics.
- Web scrapers – parse web pages according to crawled URL addresses and extract interesting

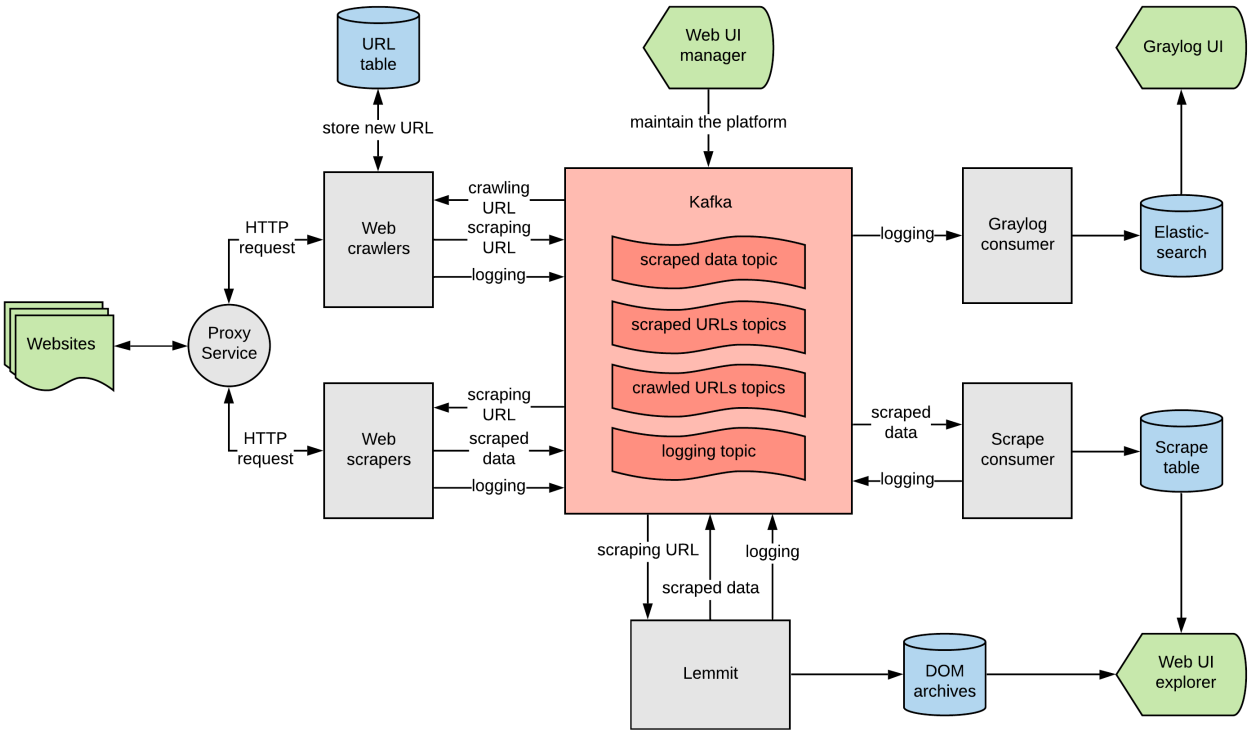

Figure 2. The platform architecture diagram. The blue elements are databases, the green are associated with a web browser, the grey are data-processing modules and the red one is the core – Apache Kafka.

metadata. The data are streamed back to Kafka for further processing.

- Scrape consumer – consumes resulting metadata from all scrapers and stores them into the database with a unified schema.
- Proxy service – allows making HTTP requests to a single website from multiple IP addresses at the same time. It ensures application firewalls will not block the scrapers. The module is provided by the project *Automated Web Analysis and Archivation* [5].
- Apache Kafka with Zookeeper[7] – the core module of the entire system. It is a scalable, robust and fault-tolerant streaming platform that assures all the modules can communicate in a simple and unified way.
- PostgreSQL – stores the resulting scrapes and information about processing statuses of all websites in *URL table*, *DOM archives* and *Scrape table*.
- Graylog monitoring tool – web-based monitoring tool for all parts of the platform.
- Lemmit – it gets URL addresses from Kafka topics so that archives whole DOM structure of a webpage. The DOM structure is used as evidence that data were present on a particular webpage at the time of scraping [5].
- Web UI – allows an admin to manage the platform, to run parsing jobs manually and to schedule them. It also empowers a user to inspect the scraped data in order to see cryptocurrency activities that appeared outside of blockchain world. Every scraped information is linked to the proof mentioned above.

3.1 Platform Modularity

Platform is written in PHP with use of Laravel framework. In this work, the framework is employed for web scraping, communication with the PostgreSQL database and for building CLI commands that manage separate modules. Scheduling features in the platform are implemented by Laravel Scheduler.

Even though there is used single programming language almost for the whole platform, it does not mean all future extensions has to be written in it. The idea is that all core modules are language independent and communicate through a unified API. It means a specific web scraper module can be implemented by any programming language and reuse common modules if it meets the API requirements. This principle allows fast prototyping of new scraping modules without the need to understand the complexity of the entire system. That being said, the problematic part of website

[7] https://zookeeper.apache.org

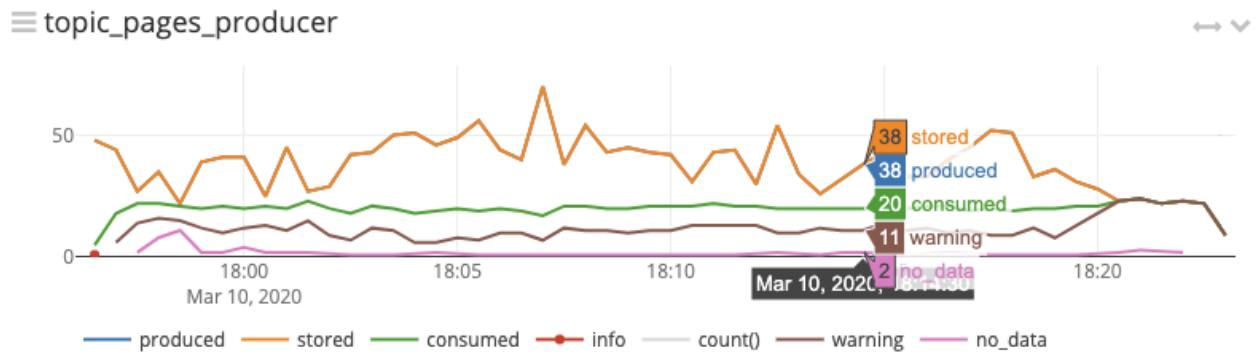

Figure 3. Monitoring example of one of bitcointalk.com scrapers. This graph was taken from a dashboard containing eleven, quite similar bitcointalk-monitoring graphs.

scraping is fully covered by this work and the more straightforward parts can be added seamlessly.

The archiving and proxy modules, database or monitoring tool, can also be smoothly changed to different implementation or other technology. This enables the possibility to keep track of technology alternatives and swap them if needed. The architecture with loosely coupled parts allows doing that without unnecessary changes.

Platform modularity is powered by Docker containers. The modules are divided into several categories, that can be seen in the platform diagram 2:

- Web crawlers/scrapers;
- Apache Kafka;
- PostgreSQL database;
- Graylog with supplementary databases;
- Web server powering the web application;
- Proxy server;
- Lemmit.

Every mentioned category runs in a separate Docker container and in the case of Web crawlers/scrapers it is expected to have tens of containers running simultaneously. All the modules are managed by Docker Compose that allows defining container dependencies, internal network communication and many other properties[8].

3.2 Data Layer Architecture

In this work, the author uses PostgreSQL database engine for the following purposes:

- saving all extracted data and their metadata in a unified structure,
- for metadata about individual web pages,
- for additional information about cryptocurrencies and websites categorization.

[8]https://docs.docker.com/compose

PostgreSQL is associated with *URL table*, *Scrape table* and *DOM archives* in the diagram 2.

The data layer also contains MongoDB and Elasticsearch technologies. Both of them are related to Graylog monitoring tool where MongoDB is used to store configuration files and Elasticsearch keeps all the logs produces by the platform. The Graylog-related databases correspond to *Elasticsearch* module in 2.

3.3 Unified Database Schema for All Parsers

It is crucial to keep data quality of scraped content at the highest possible level. Otherwise, any additional processing can be extremely difficult.

The database schema for storing the scraped data consists of particular tables:

- Owners – contains re-identified owners of crypto wallets;
- Identities – there are stored metadata about a page from where an owner and its address has been extracted;
- Addresses – contains scraped crypto addresses and their metadata.

4. The Platform Monitoring

This chapter focuses on monitoring of the entire platform and describes how critical this step is in software development.

Monitoring of the final product is one of the essential steps in software development. Without proper real-time behaviour analysis and alerting system, it is nearly impossible to maintain and operate complex systems [10].

This work uses Graylog tool for overseeing the entire platform. All modules stream logs into this tool. Graylog uses Elasticsearch DB to store the logs and MongoDB for managing configuration files. The tool is capable of displaying metrics generated from logs, creating alerts, dashboards, investigating log streams

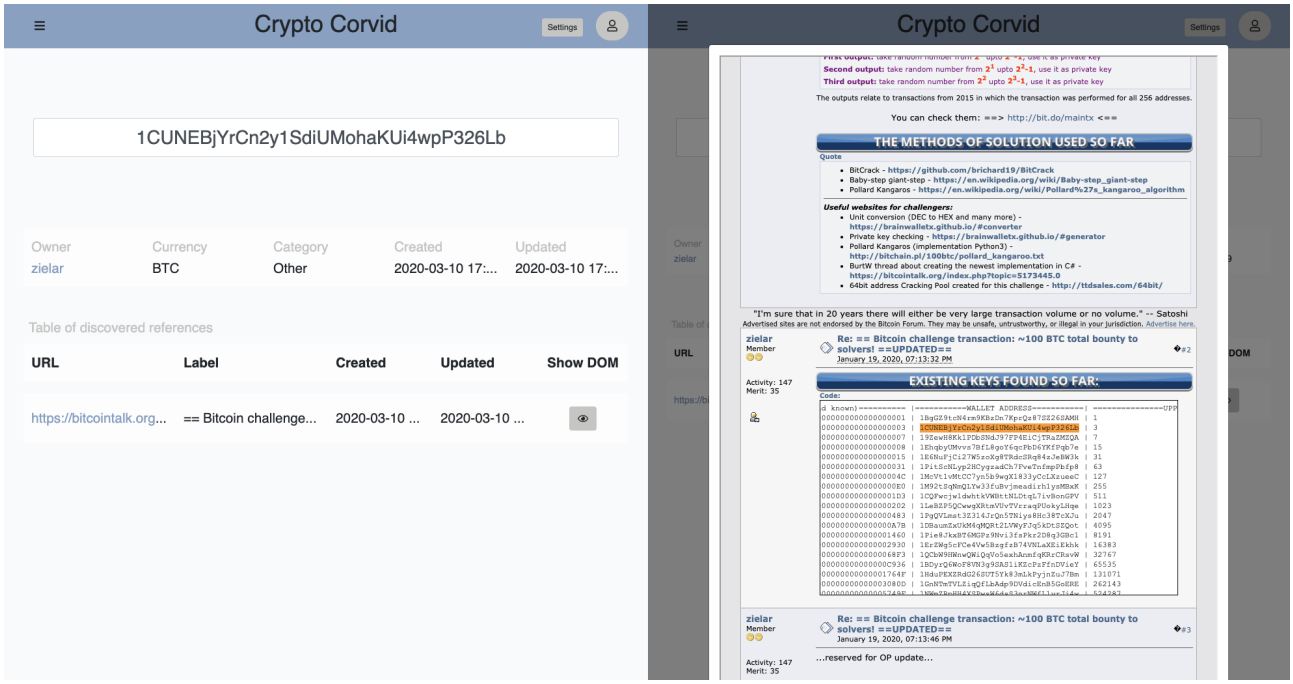

Figure 4. On the left screenshot, there are metadata as a result of a cryptocurrency address lookup. The right side shows a copy of a DOM associated with the searching address. The DOM copy is displayed after clicking on the ”Show DOM” button.

and has many other features that are useful for monitoring such complex systems. Figure 3 shows an example of a graph with metrics from scraping bitcointalk.com. The Kafka module produces the following metrics:

- stored – number of records stored into PostgreSQL database;
- produced – number of messages streamed into Kafka output topic;
- consumer – number of messages consumed from Kafka input topic;
- warning – number of connection warnings, in this case, it is when the scraper hits rate-limits of a particular website;
- other less exciting metrics.

5. Searching In The Results

This chapter introduces a web-based application that is capable of searching in the scraped metadata and scheduling repetitive scraping jobs.

The described platform is capable of generating a massive amount of cryptocurrency metadata. Only an intuitive search engine with a friendly user interface can maximize the data usage though. To reach the maximal potential of the scraped data, the author also designed and implemented a simple web-based application in PHP and Vue.js as a part of the work.

The web search engine has three major use-cases where a user can search for the following information and receive these properties:

- a cryptocurrency address;
  - category – Exchange, Mining pool, Person, Scam, etc.;
  - currency – BTC, LTC, ETH and others;
  - owner – an identified internet entity;
  - references – what web pages contain the address;
  - timestamps – when was the address scraped for the first time and when was updated at the last time.
- a source of a scraped data;
  - URL – web address of the source;
  - type – Web forum, Social network, Abuse report tool etc.;
  - addresses – which addresses were scraped from the source.
- an owner or wallet.
  - category – Exchange, Person etc.;
  - sources – which websites contain a mention of the owner;
  - addresses – list of addresses assigned to the wallet.

Figure 4 shows screenshots of the implemented web-based search engine. The engine displays results from search by a bitcoin address. The result discovers hypothetical user `mOtex2` that is classified

as an abuser, and also there are displayed several activities, associated with the user. Every activity is associated with a DOM copy from the time of scraping so a user of this application can see the proof from where was the information taken. The DOM copies are provided by archival capabilities of integrated project [5].

6. Evaluating The Platform

This chapter describes several ways of how the platform is evaluated and possible testing improvements that might be implemented in the future.

When the infrastructure part of the platform is executed, it takes around one minute to get the data streaming module, with two monitoring tools and databases, prepared for operation. After that, the scraping/crawling modules can be triggered.

Every crawler stores an URL metadata into a database and streams them into Kafka topic simultaneously (note, this can be simplified by using Confluent Kafka Connectors[9]). This is the first touchpoint where we can verify the behaviour. First of all, we know the numbers from the monitoring 3, but we can also perform SQL query in the database as well as in the Kafka, so all the numbers have to match for a specific time–topic–crawler combination. After performing this, we can be sure the data are not lost during the process. The similar evaluation can be done for all the web-scrapers as they also stream/store data into two destinations, and all the scrapers are monitored.

The second touchpoint is verifying whether the scraped data are actually associated with the correct webpage. Currently, this is manual work and can be surely automated. We can use the web application, built on top of the platform, to search for a specific crypto address and click on the link, associated with the search result 4. The address has to be present on the page.

Due to the fact that some websites were scraped by other projects at FIT BUT in previous years, we can also compare the older data with currently scraped results. This touchpoint is just a theory, and no such a cross-project comparison has been performed yet.

The very interesting finding would be to compare how much time it takes when the web scraping is performed sequentially and with the use of Kafka parallelism. Currently, this analysis is also not possible because the platform is missing a connection with the smart proxy that is using an IP address pool. It means, the scrapers work in parallel, but they have to wait in order to not to hit rate-limiting.

[9] https://www.confluent.io/connectors

This paper does not conduct any results in terms of the platform performance. These results are associated with the performance of the Apache Kafka covered by the study *Kafka versus rabbitmq* [8].

7. Conclusions

The goal of this work was to implement a platform for collecting cryptocurrency addresses and web application for managing this platform. The platform meant to be highly modular with monitoring for each module and suppose to utilize scalability features. The core of this platform should parse interesting web pages containing cryptocurrencies metadata and store the data into storage with a unified database scheme. The data should be extracted from publicly available sources according to the definition of OSINT.

The solution had to be platform-independent. The platform independency supposes to be achieved by Docker containerization and using loosely coupled modules communicating through the Apache Kafka streaming platform.

The author successfully designed and implemented the platform with the aim of modularity and easy addition of new webpage parsers. The whole platform composes from multiple Docker containers. The parsing core of this platform consists of several tasks that are scheduled from a web application and all of them are monitored through Graylog service. The platform uses PostgreSQL as a persistent data storage.

The processing pipeline outputs metadata about cryptocurrency addresses that are used to searching for activities that happened outside of blockchain world. For the searching purposes, there was implemented web-based application. The gained information can lead to crypto criminal activities detection. With a combination of integrated project *Automated Web Analysis and Archivation*, the created system is able to provide legal evidence of scraped data validity.

This work solves most of the web parsing issues and enables seamless extensibility of scraping modules that can be implemented during networking courses here at BUT FIT or as part of Bachelor's or Master's thesis. Currently, all the scrapers cannot reach the maximal speed due to rate-limiting because of the missing proxy server. The proxy will be implemented later.

The core functionality is meant to be published as open-source software. The author plans to continue with further development and to connect all parts of project [5] with this platform.

Acknowledgements

I would like to thank my supervisor Mr. Ing. Vladimír Veselý, Ph.D. for his professional guidance and Ing. Tomáš Kocman for help with the integration of his master's thesis.

References

[1] Dante Disparte. Crypto crime is taking a violent turn. online, Apr 2020. https://www.forbes.com/sites/dantedisparte/2019/01/28/crypto-crime-is-taking-a-violent-turn.

[2] Wilhelm Hasselbring. Microservices for scalability. Keynote Talk Abstract, 3 2016.

[3] Megan Mary Jane. How to bypass anti-scraping techniques in web scraping. online, Apr 2020. https://bigdata-madesimple.com/how-to-bypass-anti-scraping-techniques-in-web-scraping/.

[4] Pierre de Wulf. A guide to web scraping without getting blocked in 2020. online, Apr 2020. https://www.scrapingbee.com/blog/web-scraping-without-getting-blocked.

[5] Tomáš Kocman. Automated web analysis and archivation, 2019.

[6] FriendsOfPHP. Goutte, a simple php web scraper. online, Apr 2020. https://github.com/FriendsOfPHP/Goutte.

[7] Hendrik Swanepoel. A super quick comparison between kafka and message queues. online, Apr 2020. https://hackernoon.com/a-super-quick-comparison-between-kafka-and-message-queues-e69742d855a8.

[8] Kyumars Sheykh Esmaili Philippe Dobbelaere. Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper. *ACM International Conference on Distributed and Event-based Systems*, (11):227–238, 2017.

[9] Apache.org. Kafka introduction. online, Apr 2020. https://kafka.apache.org/intro.

[10] Gregor Scheithauer Matthias Winkler, Jorge Cardoso. Challenges of business service monitoring in the internet of services. *International Conference on Information Integration and Web-based Applications Services*, (10):613–616, 2008.