BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2018
http://excel.fit.vutbr.cz

Automatic Web Page Reconstruction

Viliam Serečun*

Abstract
Many legal institutions require a burden of proof regarding web content. This paper deals with a problem connected to web reconstruction automation and web archiving. The main goal is to provide an open source solution which will satisfy legal institutions with their requirements. This work presents two main products. The first is a framework which is a basic building block for developing web scraping and web archiving applications. The second product is a web application prototype. This prototype shows the framework utilization and the solution for institutions requirements. The output from the application is MAFF archive file which comprises a reconstructed web page, web page screenshot, and meta information table. This table shows information about collected data, server information such as IP addresses and ports of a device where is the original web page located, and time stamp.

Keywords: web reconstruction — web scraping — web indexing — Lemmiwinks — web page — MultiFIST

Supplementary Material: Lemmiwinks GitHub — MultiFIST GitHub

*xserec00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

With a new form of crime which is evolving on the Internet such as a drug market or people trafficking, legal institutions demand to collect and archive published information by cybercriminals for a burden of proof. However, there is not an open source solution to offer this functionality.

In this work, I introduce web reconstruction approaches. There are two main problems related. The first is a selection of web archive format. There is not a standard archive format and only a few of them are fully supported by web browsers. The second problem is an HTML document structure and complexity related to modern web pages, especially with dynamic content.

Most of the current tools and solutions give partial functionality related to automation of web reconstruction and web scraping. Commercial solutions are mostly focused on web scraping such as Dexi.io [1] or Scrapinghub [2], and either open source solution [3] provides similar approach. There are also browser extensions [4, 5] for web archiving, but it is not possible to use them for automation.

However, any of these solutions provide a comprehensive platform for web reconstruction and web scraping. I developed a framework and web application, where I tried to connect archiving, scraping and indexing of web page content. The Lemmiwinks framework provides a set of tools to archive content and extract information from it. Upon this framework, I developed a web application having an ability to schedule scraping and archive tasks. This application also serves as a search platform where a user can look for stored information and even visualize the web page snapshot.

My current solution provides many possibilities to develop automation tool to archive web content and extract information from it. The framework supports accessing websites with and without JavaScript rendering of web content. The currently supported archive format is Mozilla Archive Format. The web application MultiFIST (Multi-Functional Index Scraping Tool) is a demonstration of the possible future product. The goal of accessing archived website was to create feeling that behaviour of an accessed archived web page is the same as the original one stored on the

Internet.

2. Web Reconstruction

During the accessing the web page there are two approaches to reconstruct the web page content:

- with JavaScript execution when the content is reconstructed with dynamic elements already rendered. A drawback of this solution requires more CPU power and it is slow, but it provides better results on dynamic web content.
- without JavaScript execution. The web page is not rendered before archiving. This fact can cause a problem with dynamic content rendering. This approach is extremely fast because it requires just HTTP GET requests when the received content is stored directly into an archive.

2.1 Mozilla Archive Format

MAFF file format was designed to store multiple web pages into a single archive. The archive is an ordinary zip file. The design of this format allows storing even dynamic content like videos or audio files. It also allows storing meta-data of saved resources [6].

The MAFF specification defines four conformance levels. Elementary, basic, normal and extended levels define different requirements for reading and writing implementation of the standard. The main differences among these levels are specifications of storing and accessing meta-data stored in archives [7].

Figure 1 shows the directory structure of MAFF archive satisfying basic level [7].

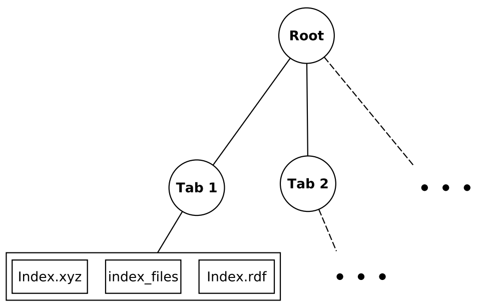


Figure 1. MAFF archive structure comprises a root directory. The root directory has 1 to N sub-directories. Each sub-directory represents one tab. There is an index file, *index.rdf* (optionally), and *index_files* directory inside the sub-directory. The ”xyz” extension of index file depends on the file content type. If the index file consists of external resources, these resources will be stored in the *index_files* directory. The collected meta information from resources is in the *index.rdf* file [7].

The RDF file usually include the following information [7]:

- The file name of the main document.
- The original URL the page was saved from.
- The date and time of the save operation.
- The title of the page, if present.
- The character set to use when parsing files that are part of the page.

2.2 Web Page Structure

Each web page is constructed from various elements. Some of those elements can refer to external resources. These resources can be images, videos, scripts etcetera. The elements referencing to external resources can cause the recursion. For example, linked CSS file can refer to another CSS file [8, 9].

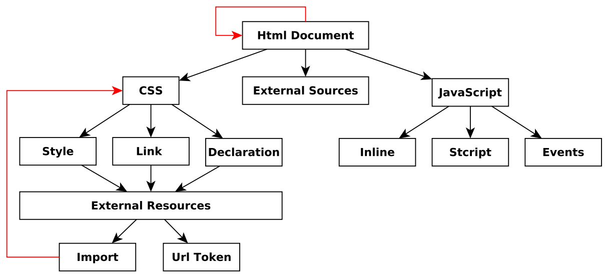


Figure 2. Web Page document structure. Red arrows represent recursion where the resources can refer to other resources.

Figure 2 presents possible parts of web page document which can cause recursion. During the website content reconstruction, it is necessary to take these aspects into consideration. Ignoring recursive resources can cause insufficient results in course of archive rendering.

2.3 Web Page elements

The following subsection shows some examples of web page elements to create clearer vision how elements can refer to external resources and even how they can create a recursive reference as was discussed in the previous subsection. Figure 3 shows mentioned elements.

External resources can be referenced using URL addresses. URL address path can be in absolute or relative format. Relative paths are resolved using base URL address path. Some elements can hold a special reference type called data URI. This means that resource data are encoded as BASE 64 string. This string is placed in data URI scheme [10].

(a) HTML elements

```
<iframe src="url"/>
<img src="data:image/jpeg;base64,"/>
<script src="url"></script>
```

(b) CSS elements

```
@include "veverka.css";
@include url("/veverka.css");
background: url(veverka.gif);
```

Figure 3. HTML and CSS elements referencing to external resources [8, 9].

3. Framework

The Lemmiwinks framework provides basic building platform for developing applications focused on web archiving and scraping. This framework is developed in Python 3 and compared to other open source solution [11, 3] provides asynchronous HTTP clients using the asyncio standard library [12]. The advantage of this solution is that the framework also provides HTTP client developed under Selenium library [13] to render dynamic content. It is possible to divide it into four major modules.

- HTTP client
- Parser
- Archive
- Extractor

3.1 HTTP Client

HTTP client module implements two types of clients. The first is asynchronous client using asyncio and aiohttp library[12, 14]. The second client uses Selenium library. Each of Selenium instance is running asynchronously using threads. This approach allows faster and more efficient websites processing. The Selenium client is managed by Selenium pool. The Selenium pool functionality is described in Figure 4.

A Selenium library wraps real browsers. The instances in the framework are created by a factory. This factory can currently work with Firefox, Chrome, and PhantomJS browsers types. The last type is browser without a user interface, therefore it saves some resources.

3.2 Parser

In the parser module, I implemented two types of parsers. The CSS parser, which searches for import and URL tokens. The parsers allow to extract external references and update them. The similar functionality provides HTML parser. This parser can also extract all textual data from web content to help Extractor module with data extraction.

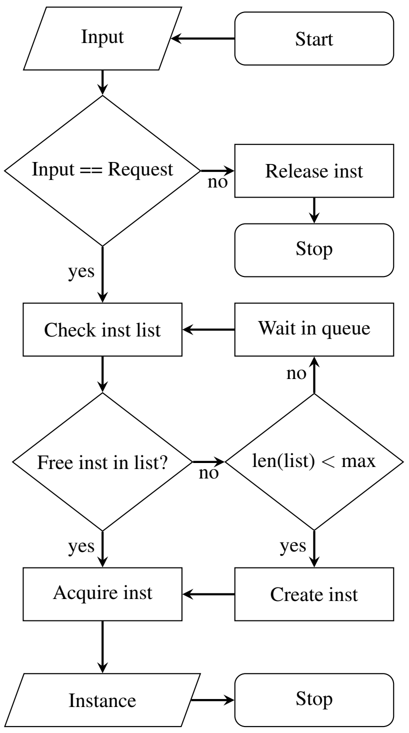


Figure 4. Selenium Pool functionality described by the flowchart. The pool manages instances of Selenium clients. The input can be a request or a Selenium instance. A running task uses the pool to acquire a Selenium client or to release an instance. If a requested instance does not exist and Selenium pool does not manage a specified instance limit, the pool will create a new client instance. Otherwise, a task has to wait in a queue until some instance is released.

3.3 Archive

This module implements two archiving modes. Modes reflect web reconstruction types described in previous section. Currently supported archive format is MAFF. The archive module is designed to support multiple tab archiving. This is achieved using general interface implementation. A programmer can implement custom archiving process using this interface. Archiving processes are then added to a list. This list is processed and all data are stored into an archive file.

Depending on archiving mode JavaScript resources have to be handled properly. If the content is rendered during archiving process, scripts will have to be deleted. It avoids element double rendering.

3.4 Extractor

The last module used for data extraction is the smallest one. Currently supported data extraction type is by regular expressions. I implemented a generic interface which allows other developers to implement the extractor upon their needs.

4. Web Application

Multi-functional index scrapping tool is demonstration application which allows user administration task scheduling, and web archive storing. It also allows search scarped information. The application is developed using Django framework.

4.1 Scheduler

Task scheduling is handled by APScheduler library [15]. The application supports task creation, task editing, task pausing and resuming. Each task consists of several jobs. An archiving job is executed as a subprocess and is parallelized with other jobs. Each subprocess can handle one web page. A task can process one to five web pages with an unlimited number of scraping rules.

4.2 User Interface Design

The user interface design was meant to be simple and easy to use. After signing in to the application a user is redirected into task list view. The view shows Figure 5a. This part allows a user to manage tasks (edit, delete, pause resume) or create a new task.

Each task has its own detailed view. This view shows URL address list of a targeted web pages and scrapping rules applied during data extraction. It also contains search form where a user can search for extracted information from specified rules. Bellow the searching for is a list of archived web pages. Figure 5c shows a described view.

Figure 5d shows a detail view of task where it is possible to see scrapped data from ”the pirate bay” web page, especially Bitcoin and Litecoin address placed on its content.

The last view in Figure 5b shows a list of scrapping rules. This view is accessed true navigation menu and allows a user to create a new rule, edit existing rule or delete unused rule.

5. Conclusions

This paper described a solution for automating web page reconstruction. The result of this work is Lemmiwinks framework and demo application demonstrating framework possible use. The framework support reconstruction with and without dynamic content rendering. Example of a reconstructed web page using the framework shows Figure 6.

However, the framework has some limitations on web pages using Captcha or other advanced tools against web scraping. This is causing a problem when the requested content is not reconstructed.

The advantage of my solution is flexibility during application development. A developer can use the framework to optimize needs of requirements. The framework also provides a complex platform for web archiving and web scraping, which other solutions [1, 3] do not provide.

The future work will be focused to optimize Lemmiwinks framework especially reconstruction using Selenium library can be written more efficiently. One of the other improvement is to implement captcha resolving and adding a module allowing for signing in to websites which require authentication.

Acknowledgements

I would like to thank my supervisor Mr. Vladimír Veselý for his help during development part of this project and to Mr. Martin Vrábel for suggestions.

References

[1] Available at https://dexi.io/. Online; accessed 8 April 2018.

[2] Available at https://scrapinghub.com/. Online; accessed 8 April 2018.

[3] Scrapy. Available at https://scrapy.org/. Online; accessed 8 April 2018.

[4] Danny Lin. webscrapbook. Available at https://github.com/danny0838/webscrapbook. Online; accessed 8 April 2018.

[5] Paolo Amadini Christopher Ottley. Mozilla Archive Format, with MHT and Faithful Save. Available at https://addons.mozilla.org/en-US/firefox/addon/mozilla-archive-format/. Online; accessed 8 April 2018.

[6] Paolo Amadini Christopher Ottley. Features of the MAFF file format. Available at http://maf.mozdev.org/maff-file-format.html. Online; accessed 8 April 2018.

[7] Paolo Amadini Christopher Ottley. The MAFF specification. Available at http://maf.mozdev.org/maff-specification.html. Online; accessed 8 April 2018.

[8] Tab Atkins and Simon Sapin. CSS Syntax Module Level 3. Available at https://drafts.

csswg.org/css-syntax-3/. Online, accesed 8 April 2018.

[9] Mozilla. HTML elements reference. Available at https://developer.mozilla.org/en-US/docs/Web/HTML/Element. Online; accessed 8 April 2018.

[10] Larry Masinter. RFC 2397: The "data" URL scheme,". *IETF, August*, 1998.

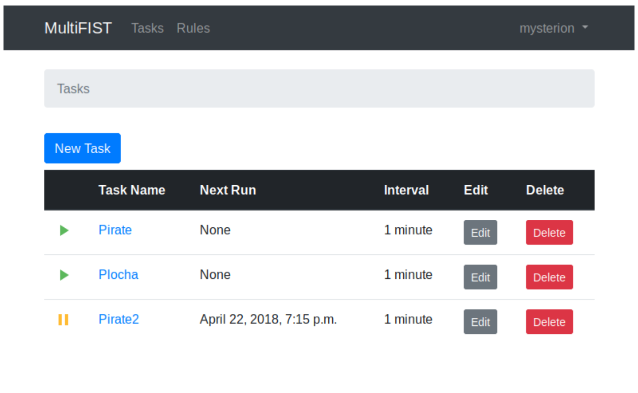
[11] Mikhail Korobov. Splash - A javascript rendering service. Available at https://github.com/scrapinghub/splash. Online; accessed 8 April 2018.

[12] Python Software Foundation. The Python Standard Library. Available at https://docs.python.org/3/library/index.html. Online; accessed 8 April 2018.

[13] Selenium. Available at https://www.seleniumhq.org/. Online; accessed 8 April 2018.

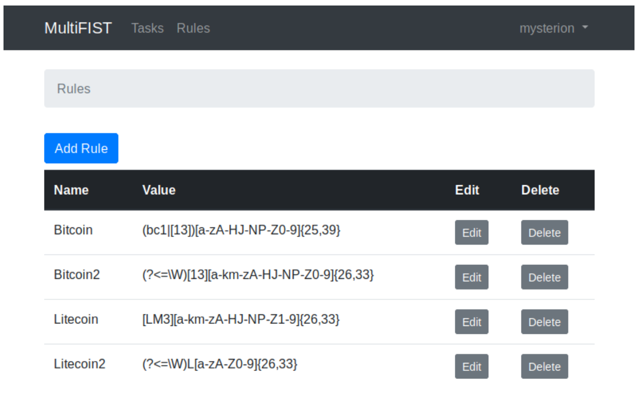
[14] Aiohttp contributors. Welcome to AIOHTTP. Available at https://aiohttp.readthedocs.io/en/stable/. Online; accessed 8 April 2018.

[15] Alex Grönholm. Advanced Python Scheduler. Available at https://apscheduler.readthedocs.io/en/latest/. Online; accessed 8 April 2018.



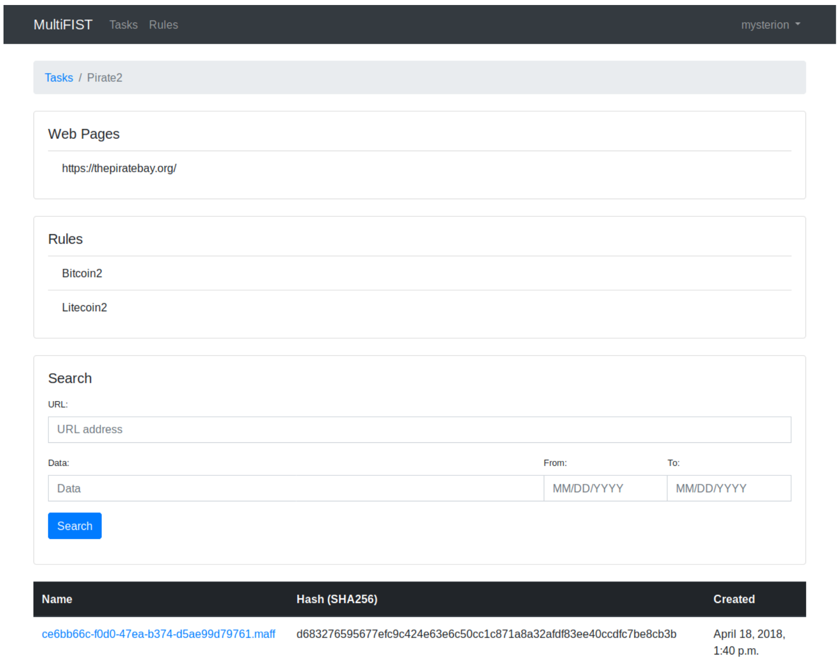
	Task Name	Next Run	Interval	Edit	Delete
▶	Pirate	None	1 minute	Edit	Delete
▶	Plocha	None	1 minute	Edit	Delete
‖	Pirate2	April 22, 2018, 7:15 p.m.	1 minute	Edit	Delete

(a) Task list view



Name	Value	Edit	Delete
Bitcoin	(bc1|[13])[a-zA-HJ-NP-Z0-9]{25,39}	Edit	Delete
Bitcoin2	(?<=\W)[13][a-km-zA-HJ-NP-Z0-9]{26,33}	Edit	Delete
Litecoin	[LM3][a-km-zA-HJ-NP-Z1-9]{26,33}	Edit	Delete
Litecoin2	(?<=\W)L[a-zA-Z0-9]{26,33}	Edit	Delete

(b) Rule list view



Web Pages

https://thepiratebay.org/

Rules

Bitcoin2

Litecoin2

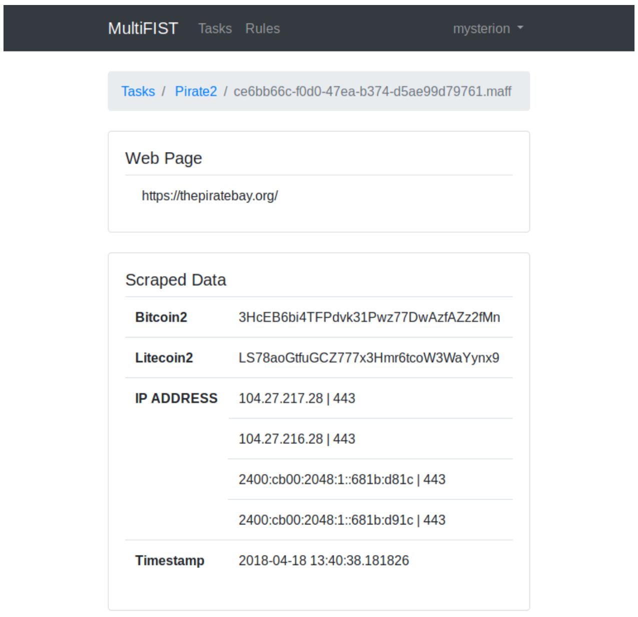
Search

URL: URL address

Data: Data From: MM/DD/YYYY To: MM/DD/YYYY

Name	Hash (SHA256)	Created
ce6bb66c-f0d0-47ea-b374-d5ae99d79761.maff	d683276595677efc9c424e63e6c50cc1c871a8a32afdf83ee40ccdfc7be8cb3b	April 18, 2018, 1:40 p.m.

(c) Task detail view



Web Page

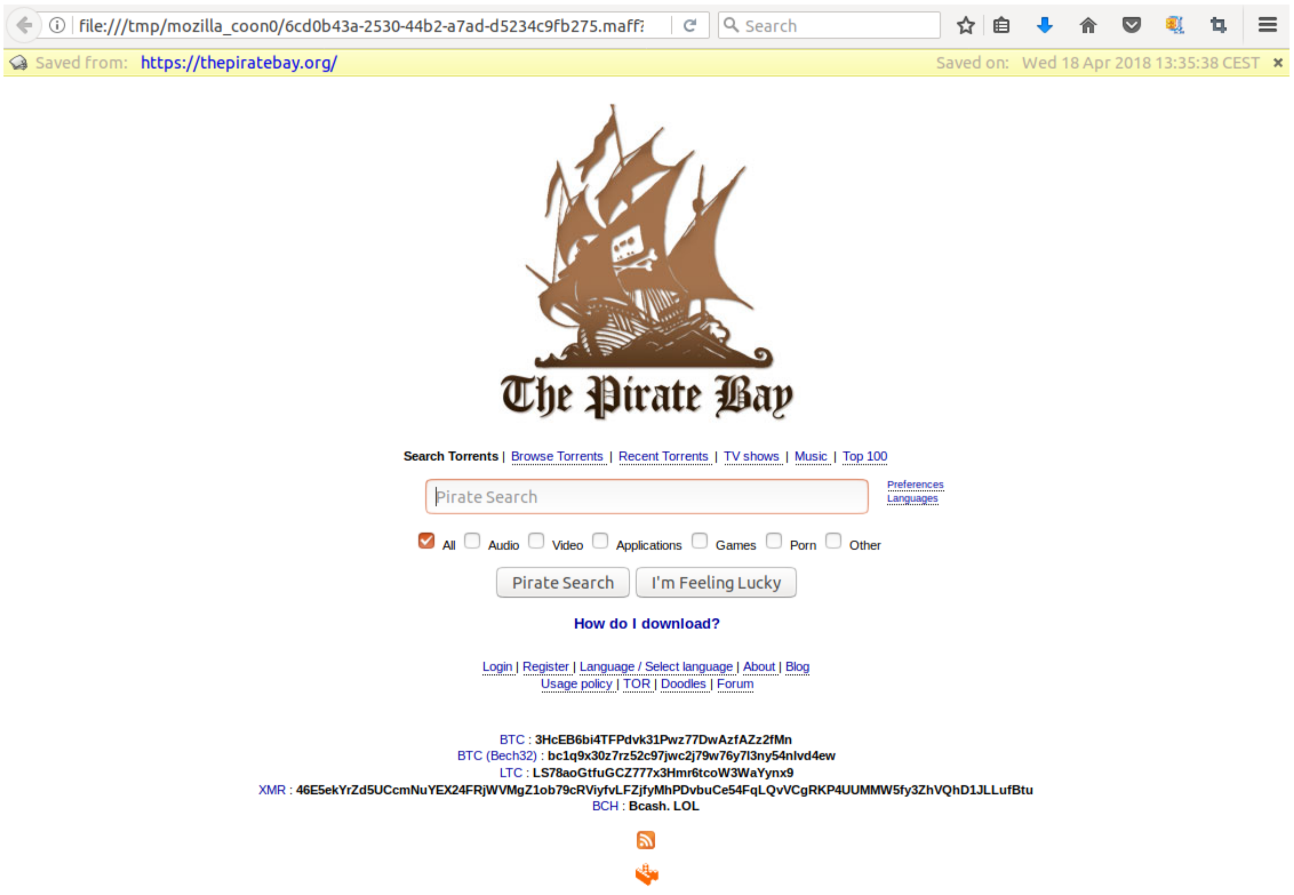
https://thepiratebay.org/

Scraped Data

Bitcoin2	3HcEB6bi4TFPdvk31Pwz77DwAzfAZz2fMn
Litecoin2	LS78aoGtfuGCZ777x3Hmr6tcoW3WaYynx9
IP ADDRESS	104.27.217.28 | 443
	104.27.216.28 | 443
	2400:cb00:2048:1::681b:d81c | 443
	2400:cb00:2048:1::681b:d91c | 443
Timestamp	2018-04-18 13:40:38.181826

(d) Web archive detail view

Figure 5. User interface of MultiFIST application



file:///tmp/mozilla_coon0/6cd0b43a-2530-44b2-a7ad-d5234c9fb275.maff

Saved from: https://thepiratebay.org/ Saved on: Wed 18 Apr 2018 13:35:38 CEST

The Pirate Bay

Search Torrents | Browse Torrents | Recent Torrents | TV shows | Music | Top 100

Pirate Search

Preferences
Languages

All Audio Video Applications Games Porn Other

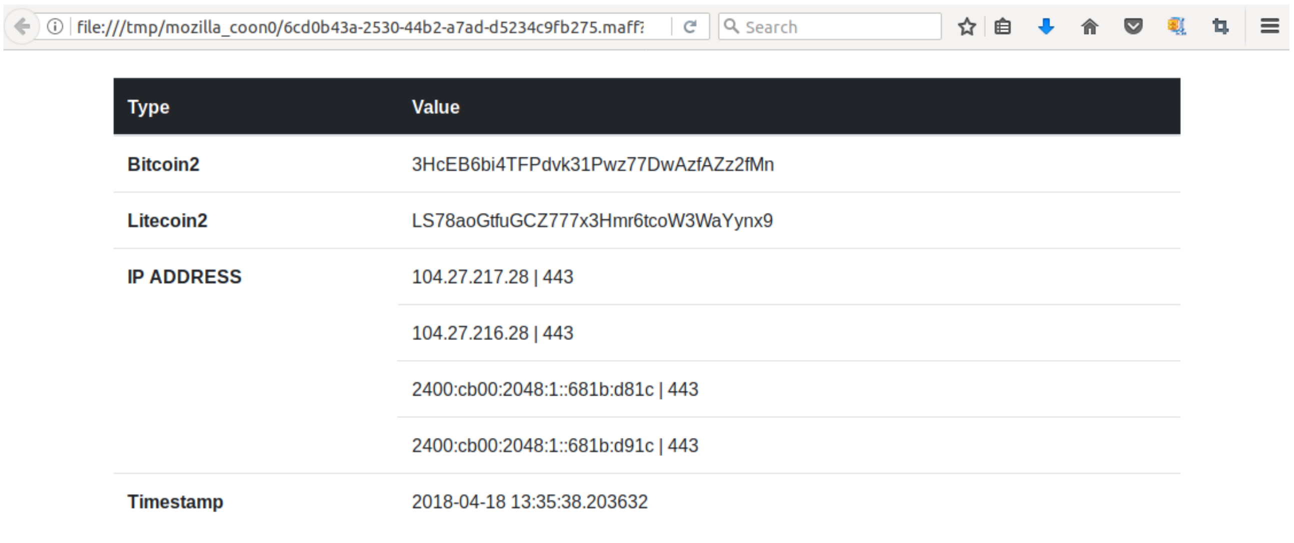
Pirate Search | I'm Feeling Lucky

How do I download?

Login | Register | Language / Select language | About | Blog
Usage policy | TOR | Doodles | Forum

BTC : 3HcEB6bi4TFPdvk31Pwz77DwAzfAZz2fMn
BTC (Bech32) : bc1q9x30z7rz52c97jwc2j79w76y7l3ny54nlvd4ew
LTC : LS78aoGtfuGCZ777x3Hmr6tcoW3WaYynx9
XMR : 46E5ekYrZd5UCcmNuYEX24FRjWVMgZ1ob79cRViyfvLFZjfyMhPDvbuCe54FqLQvVCgRKP4UUMMW5fy3ZhVQhD1JLLufBtu
BCH : Bcash. LOL

(a) Reconstructed web page



file:///tmp/mozilla_coon0/6cd0b43a-2530-44b2-a7ad-d5234c9fb275.maff

Type	Value
Bitcoin2	3HcEB6bi4TFPdvk31Pwz77DwAzfAZz2fMn
Litecoin2	LS78aoGtfuGCZ777x3Hmr6tcoW3WaYynx9
IP ADDRESS	104.27.217.28 | 443
	104.27.216.28 | 443
	2400:cb00:2048:1::681b:d81c | 443
	2400:cb00:2048:1::681b:d91c | 443
Timestamp	2018-04-18 13:35:38.203632

(b) Meta information table

Figure 6. Example of a reconstructed web page with collected information.