Reconstructing web pages from browser cache

E. Schaap
edwin.schaap@os3.nl

I. Hoogendoorn
iwan.hoogendoorn@os3.nl

University of Amsterdam

July 5, 2013

University of Amsterdam

Nederlands Forensisch Instituut
Ministerie van Veiligheid en Justitie

This research examines the possibilities to reconstruct a web page from normalized browser cache. Where other tools are using pre-processing mechanisms, this research shows that post-processing leads to more complete results without tampering with the evidence. Apart from studying the way browsers initially store the caching information and how this data is normalized, it is also investigated if the information collected is sufficient to reconstruct a web page. The outcome of the prove of concept using this post-processing approach leads to the expected results. The value of a reconstruction is outlined with an analysis on the shortcomings and impact on this process. It will show that the trustworthiness of such a reconstruction is debatable.

Acknowledgements

We would like to express our gratitude to all who supported this project. First of all, we would like to thank our supervisor, Ruud van Baar MSc. for his support and feedback. Second, we would like to thank the Netherlands Forensic Institute for providing us the facilities needed to conduct our research.

Contents

1. Introduction

Very often when a suspect commits a certain crime, good preparation is done for perfection reasons and to stay out of the hands of the civil authorities. This preparation is often done with the use of information that can be found on the Internet. This can also be the case for people that witnessed certain evidence on the Internet. This data can be one of many important elements to build a case against a suspect in the conviction process.

Browser forensics and in particular the web cache aspect, can provide additional information or proof to build or break a case. Due to the continuous growing amount of requests to investigate what suspects exactly looked at during their "criminal research" the Netherlands Forensic Institute (NFI) has developed a module that is part of their Traces library that collects web browser cache files from various browser vendors and normalize this data for easier processing in a later stage. Applications like XML Information Retrieval Approach to digital Forensics (XIRAF) [2] and Hansken use the Traces library to get their inputs from.

1.1. Browser caching

Browser caching is a mechanism where files retrieved from websites that are visited are stored (temporary) on a specific location on a hard disk. The main purpose of this is that (parts of) web pages that were visited earlier can be loaded much faster when this web page is visited again in a later stage. The browser usually compares the data of the web page that is remotely available on the Internet with the one that is hold locally in the caching folder. When this web page has not changed, the cache will be used or parts of the cache and otherwise the web page is downloaded again, displayed and possibly cached.

When the web browser is closed the web cache remains stored on that specific location of the hard disk. Different options like the amount of cache that can be stored, the cache retention or delete the web pages that are cached are available in most browsers.

There are two type of caches available, online and offline caching. The high level difference is that with offline caching the web page developer determines what elements of the requested web page are cached by defining these elements in a predefined file, also called a manifest. With online caching the browser determines what is cached and what is left out.

1.1.1. Browser caching and forensics

In the field of forensics, the web browser is an important element. A huge number of criminal and civil cases may depend on the information related to the Internet activity of a user in order to prove something or to use this as extra evidence in the bigger picture.

By analyzing the history, or browser cache information it can give good insight of the behavior of a suspect. The information found can provide clues that can lead to conviction or acquittal of this suspect.

The browsers that are currently used the most (figure 1) are Google Chrome, Microsoft Internet Explorer, Mozilla Firefox, and Apple Safari. All these browsers are using their own method of storing web caching information which makes it harder to analyze this.

1.2. Research Question & Approach

In this research, we state our research problem as follows:

In what ways can one visually reconstruct websites with information retrieved from normalized browser caches that can be used for computer forensic examiners to build a case?

More specifically, we focus on the following problems:

- In what way is web browser cache data stored using different web browsers?
- What methods are there to reconstruct websites from web browser cache data?
- How reliable is a webpage that is reconstructed from web browser cache data?

1.2.1. Scope

The project is scoped to research the possibility to reconstruct websites with the use of web browser caching. A selection will be made of three different browsers.

The NFI currently has a package that they use to retrieve and normalize the caching data for several browsers, and we will investigate if this done in a proper way, or if we can contribute to this in what ever way. The reconstruction of the web pages will be done with the use of the package created by the NFI.

1.3. Related work

Previous research shows that extracting the Uniform Resource Locator (URL)s from the cache[13] and matching against the cached web files is possible[20]. It is possible to create a time-line of web resources which have been visited[15]. In some cases this can be achieved using a handful of proprietary software. Tools like NetAnalysis[19] and CacheBack[3] are capable of performing simple visual reconstruction but the inner workings are unknown.

1.3.1. Forensic web browser analysis - existing tools

There are a few software packages available that have the functionality to examine web cache data. These tools and their capabilities are explained below.

NirSoft NirSoft provides a collection of small freeware utilities, all of them developed by Nir Sofer.

- *MozillaCacheView*[18]
- *ChromeCacheView*[4]
- *SafariCacheView*[25]

The utilities only provide a list with the files that can be found in web browser caches and in some cases it is possible to export single files to the appropriate formats.

Tim Johnson *Cache View*[27] is a viewer for the Netscape Navigator, Mozilla Firefox, Opera, and Internet Explorer web caches. This tool also provides a list of the cached items so the files can be viewed separately or with the Fully Qualified Domain Name (FQDN) online.

Digital Detective *NetAnalysis*[19] is software for the extraction and analysis of data from Internet browsers. It has a host of features to help with forensic examination such as the ability to import history and cache data from Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Apple Safari and Opera browsers. NetAnalysis has the ability to rebuild web cached pages for offline review. According to the authors of NetAnalysis, this is the first forensic software designed for this purpose[1]. It also has the ability to utilize Structured Query Language (SQL) queries to quickly identify relevant evidence and a number of other tools to assist in the review and decoding process.

This tool uses the pre-processing method that will be explained in section 3 in order to reconstruct web pages.

Siquest

- *CacheBack v3*[3]
- *Internet Examiner v3*[7]

CacheBack v3 is the predecessor of *Internet Examiner v3. Internet Examiner* also claims that it is the first forensic product to provide investigators with the ability to rebuild cached web pages for all five top leading browsers[2].

Internet Examiner also uses the pre-processing method [3] that will be explained in section 3 in order to reconstruct web pages.

Foxton Software

- *FoxAnalysis Plus*[22]
- *ChromeAnalysis Plus*[21]

[1]`http://kb.digital-detective.co.uk/display/NetAnalysis1/Home`
[2]`http://www.siquest.com/productdetailPage.php?pID=4&mid=253af8zzyy`
[3]`http://youtu.be/9lw-C4z2wVg`

Both Analysis Plus software packages have the ability to reconstruct web pages from browser cache data.

Foxton Software also uses the pre-processing method [4] that will be explained in section 3 in order to reconstruct web pages.

All of these tools mentioned above are "closed" tools and don't provide any details on how the data is (pre) processed and reconstructed. With the forensic aspect in mind it is important that we know what happens under the hood.

1.4. Research paper outline

Section 1 [*Introduction*] provides a general description of web browser caching, the research question and the scope of this research, followed by **Section 2** [*Theory*] where we go deeper into the theory of browser caching for Google Chrome, Mozilla's Firefox and Apple's Safari and how this relates to forensics. In **Section 3** [*Methods*] the method is described how reconstruction from web caching is done and how certain meta data can be important or less important. **Section 4** [*Results*] will provide and explain the results from the used methods. **Section 5** [*Analysis*] will provide some analysis about the outcome and go deeper on the technical (in)compatibilities of the used methods. This paper will conclude in **Section 6** [*Conclusion*] with an overview of this research and conclusion based on all previous sections. **Section 7** [*Further research*] will outline additional research topics that were out of scope for this research paper.

2. Theory

To give a better understanding on how the normalized data is used within this research it is important to understand where this data comes from. This section will describe how the different browsers store their data and how it is processed in the normalization step.

Figure 1 shows that Google Chrome worldwide has gained popularity (the past year) over other browsers like Microsoft Internet Explorer, Mozilla Firefox and Apple's Safari.

2.1. Firefox

Mozilla Firefox[8] is an open source and free web browser that is developed for OS X, Linux and Windows. There are also versions available for mobile operating systems like Android.

Firefox is developed by the Mozilla Foundation and its subsidiary, the Mozilla Corporation. It uses the Gecko Layout Engine[6] in order to render the requested web pages that supports the current web standards.

[4] `http://forensic-software.co.uk/Screenshots/FoxAnalysisReconReport.html`

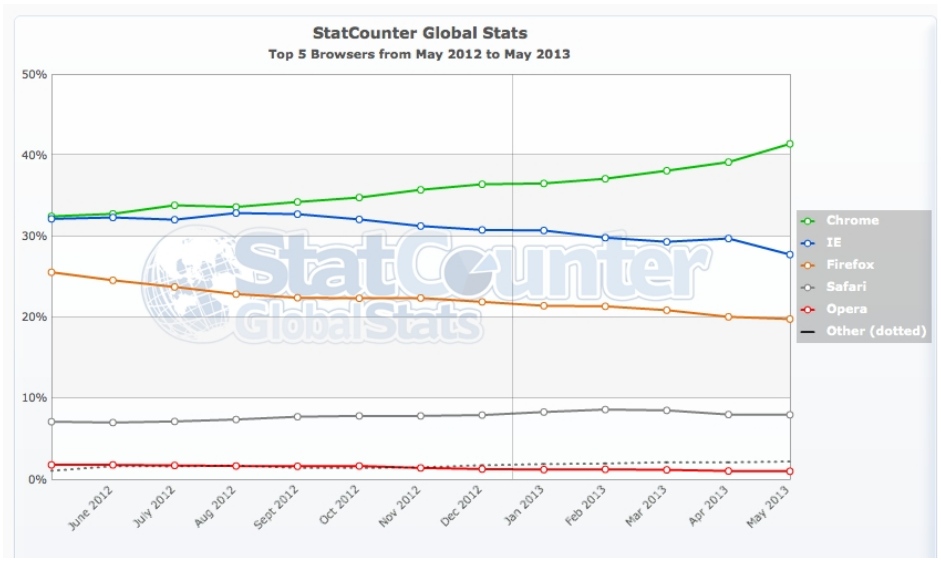

Figure 1: Browser popularity - Worldwide (source: StatCounter).

2.1.1. Cache file structure

Firefox online cache consists of *metadata* (information about the cache entries itself) and *normal data* (the actual cache items).

Firefox cache **metadata** consists of the following pieces of basic information:

- The complete request string (the unique entry to identify a cached item).
- Time & date of cache item first request.
- Time & date of the last time the cache item was requested.
- Time & date when the cache item will expire.
- The fetch count (the number of times the cache item has been requested).
- The complete server response header to the request.

In order to conduct this research, sample cache data is created

A quick look in the caching folder of Firefox points out that there are three type of files stored that reassemble the overall caching data. There is a cache map file, three cache block files and the separate cache data files.

The cache map file (_CACHE_MAP_)[5] will be the main file in order to reconstruct web pages from Firefox Cache data(figure 2). The structure of this map file consist of a header for the file itself which is followed by allocated space, called buckets, that contains information about the mapping towards the cached data. The _CACHE_MAP_ file consists of a total of 32 buckets and within each bucket there is room for 256 records (total of 8192 records).

[5]`http://code.google.com/p/firefox-cache-forensics/wiki/FfFormat`

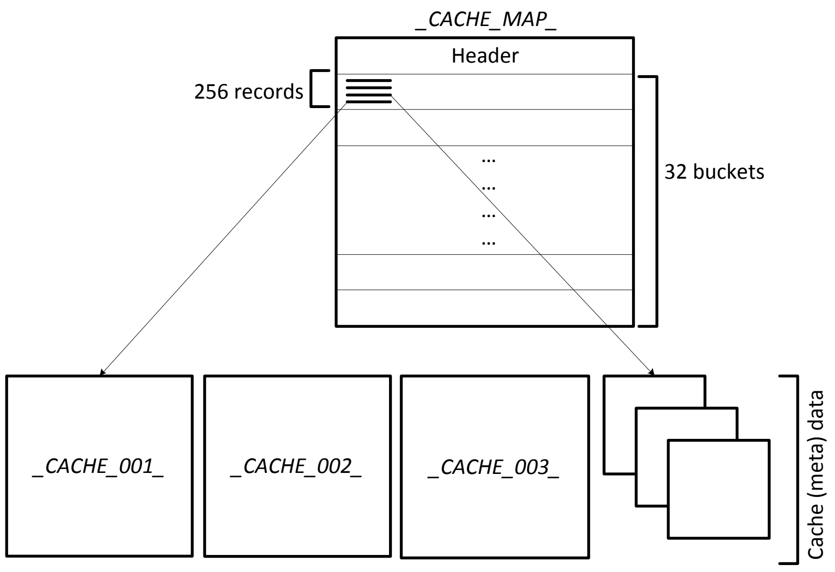

Figure 2: Firefox file cache structure.

Each record contains information for one single instance of data that is cached. One single record consists of four 32-bit integers.

1. A Hash Number.
2. An Eviction Rank.
3. The Data Location.
4. The Metadata Location.

Firefox has two methods how it saves caching data, it either saves the information inside a Cache Block File or it creates a separate file when the data cannot fit into a bucket. The hash number is used to name the separate file if the cache data is stored in that separate file. The eviction rank determines how long cache information is stored and the data location field will be used to reconstruct the cache data. The data location field points the actual data located somewhere in the cache. This also applies to metadata.

According to Keith J. Jones and Rohyt Belani [14] the determination whether the caching data is stored inside a cache block or separate files is by taking the metadata "location" field and 1) "bitwise AND" this with 0x30000000, 2) left shift the result 28 bits. When this is done the result will be a number from zero to three. When this is a zero the cache metadata is saved in a separate file and when this one, two or three this data is embedded in one of the cache block files.

When caching files are stored in blocks this is done into the files:

- **_CACHE_001_** - used to store small metadata and data entries in 512-byte blocks.
- **_CACHE_002_** - used to store medium-sized metadata and data entries in 1024-byte blocks.
- **_CACHE_003_** - used to store large metadata and data entries in 4096-byte blocks.

2.2. Chrome

Chrome[5] is a freeware web browser developed by Google Inc., which uses the Webkit[23] layout engine.

2.2.1. Cache file structure

Google Chrome uses at least five files to manage the web caches. One is the index file and a minimum of four data files which are named *data_n* where *n* is the number of the file which starts at zero (figure 3). If any of these files are missing or are corrupted for any reason these files will be recreated. The index file contains a hash table that is used to locate entries of the cached files. The data files contains information about Hyper Text Transfer Protocol (HTTP) headers and data about a given request. These files are also called block files because the file format is optimized to store information on fixed-size blocks. A block file can store blocks up to 256 bytes of data and the data can be stored/spanned across one to four of these block.

When the size of the cached data is bigger than 16 KB the web cache data will no longer be stored inside a standard block file but the data will be stored in a separate file without special headers and is just the plain data we are saving/caching. The name is usually f_xx where xx is a hexadecimal number that identifies the file.

2.3. Safari

The Safari web browser[24] is developed by Apple and is included as the internet browser for OS X and iOS operating systems. For Apple's mobile devices that has iOS as the operating system Safari is the native browser and it became native for OS X starting at version 10.3 (Panther).

2.3.1. Cache file structure

Safari versions 3 to 6 uses an SQLite database which is called "Cache.db". Versions prior stores the cache in files with the extensions ".cache". The SQLite Manager[17], a tool to read sqlite databases, showes that the cache.db database exists of three relevant tables(figure 4).

The *cfurl_cache_response* contains metadata about cache records. This includes the version of the table structure and storage policy. The *request_key* column is used to contain the URL of the cached item and the *time_stamp* column is used to store the

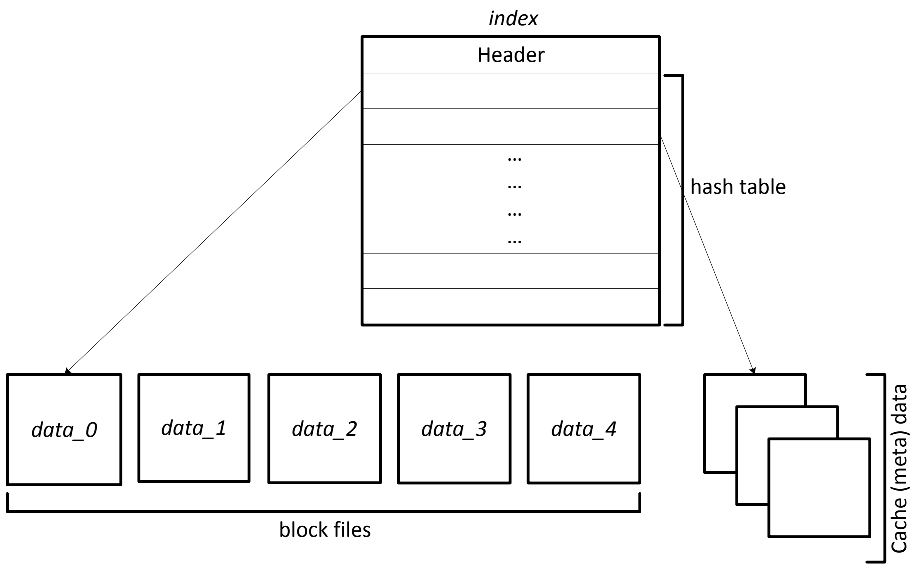

Figure 3: Chrome file cache structure.

time (in UTC) of the item that was cached. The records of *cfurl_cache_blob_data* hold information related to the HTTP request and response of a web page. This information consist of HTTP headers. The table *cfurl_cache_receiver_data* is used to store the cached item itself (jpgs, gifs, pngs, html etc.) as a Binary Large OBject (BLOB). The *entry_ID* column is the primary key in these tables which allows relating the data in one table to data stored in another table.

2.4. Firefox, Chrome and Safari web cache comparison

Now that it is clear how these three browsers store their cache data, it is now possible to compare them. Table 1 shows the differences in types of data stored.

2.5. Traces

The library of the NFI processes different types of cache data from different browsers. Section 2.4 shows the different data types of the browsers. Traces normalize this data into a generic model. This model, a so called CacheEntry, consists of the following data:

- Unique identification.
- URL request string.
- Time/Date (first request).

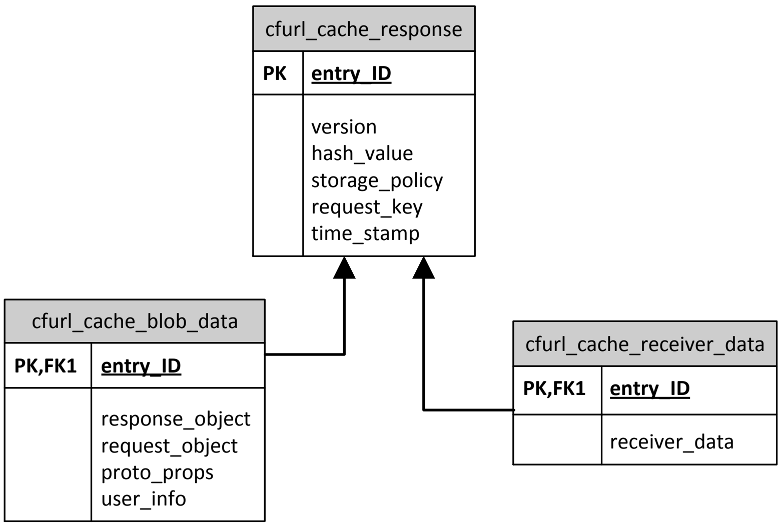

Figure 4: Safari Cache.db model.

- Server response body.

3. Methods

In order to be able to generate a visual representation of a website, multiple approaches are possible. In this section we will evaluate two possible solutions and describe why one is chosen and how the application is implemented. The starting point is the normalized data gathered by Traces. This data must be used in some way to visually present it. As browsers are particularly good at doing this job it is common sense to see if it is possible to use such a browser or the rendering engine inside it.

Since this application is a Proof of Concept (PoC) for a specific task within XIRAF and Hansken, there are some constraints in place. First, the system itself is built in Java and therefore the PoC is also created in Java. Another important factor is the usability of the system. XIRAF and Hansken are built with the user experience in mind and tries to minimize the dependencies on the user side. Because of this concept, these applications are built as online services and can be used with only a browser. This constraint forces the PoC to also use this kind of user interface and must output web specific content to a browser on a uncontrolled environment. Everything between the source data, the normalized cache, and the output can be a custom implementation.

3.1. Pre- or Post-processing

Letting a browser render a webpage means that the browser must access multiple resources like style sheets or images. To make sure these resources originate from the given

	Chrome	**Firefox**	**Safari**
Unique identification	✓	✓	✓
Eviction	✓	✓	✗
URL request string	✓	✓	✓
Time/Date (first request)	✓	✓	✓
Time/Date (last request)	✗	✓	✗
Time/Date (expire)	✗	✓	✗
Fetch count	✗	✓	✗
Client request headers	✗	✗	✓
Server response headers	✓	✓	✓
Server response body	✓	✓	✓

Table 1: Firefox, Chrome and Safari web cache comparison table.

cache there are two options to manage this: alter the communication before or after the browser will processes the data.

Pre-processing The first option is to alter the resources before it is given to the browser. This is done by finding all referenced resources in the response data and changes these to the local version which represent the cache. For instance:

http://google.com/logo.png ⇒ http://google.com**.local**/logo.png

The rewritten domain points to the local version itself. With this method, resources can be addressed to the application which in turn can deliver the requested resources. This method has some advantages but also disadvantages which are listed below.

- Advantages:
  1. Requires no configuration of the rendering browser.
  2. Can even run in the browser of the user enabling interaction.

- Disadvantages:
  1. Tampering the evidence.
  2. Hard to parse all resource identifiers, especially if JavaScript[12] is used.
  3. Non-parsed resource identifiers are circumventing the application.

The advantage of not needing to configure the rendering browser makes is great to use all kinds of rendering engines, from standalone browsers to embedded engines. The advantage of an embedded engine is that it can directly be programmed into the application, for instance a web view in Java, without external dependencies. By loading the cached website inside the users own browser, it gives the user direct access to it. The user is able to not only view the website but also to interact with it. The downside with this is that the user needs a supported browser and this adds a dependency to the system itself.

The biggest disadvantage is that the cache data (evidence) is altered. This means that the parsed data is not in its original state. Another downside can lie in the fact that suspicious websites may contain malicious code which can cause great risk for the users system. Parsing of resource identifiers is rather hard and difficult to locate them all. The majority can be parsed with a relatively small effort, but in order to get all resource identifiers it takes a lot of effort to manage this and even then it cannot be guaranteed to parse them all.

Post-processing To process the resource identifiers after the rendering browser has accessed the webpage, can be done by capturing all requests from the browser and inspect these in order to retrieve the right cached resource. This can be done by using a proxy server. A proxy server is a specific server which intercepts the HTTP requests and forwards them to the corresponding address. Reasons to use such a system can be to anonymize the origin of the request, speed up access by caching resources, scan content for malware and many others.

To be able to use a proxy server, it does not need to forward a request to the destination requested but instead look it up in the cache and serve the resource if it exists. In order to direct all requests to the proxy the rendering browser must be configured to use the proxy server which is setup for this purpose.

- Advantages:
  1. All resource identifiers are captured by the proxy.

- Disadvantages:
  1. Requires proxy configuration of rendering browser.
  2. SSL traffic is hard to deal with.

Since all requests are directed to the proxy server, all resource identifiers are captured and thus can be processed by the application. Even identifiers used by complex JavaScript are captured with this technique.

This design adds a dependency to the rendering browser which requires an extra configuration for the proxy server. Because of this, it is not feasible to use the browser of the user to present this. Because the original domains are still used for the resource identifiers, the SSL connection cannot be presented with another certificate which enables the decryption of the content without breaking the certificate-chain.

3.2. Proof of Concept design

The PoC is divided in different components as can be seen in figure 5. As described in the introduction of this section, the input and output are predefined. The input is the browser cache provided and the output is the response which can be viewed by the user in a browser.

The application flow starts with a request from the user, by using a browser, to request a preview of a specific URL. The frontend of the application processes this request and

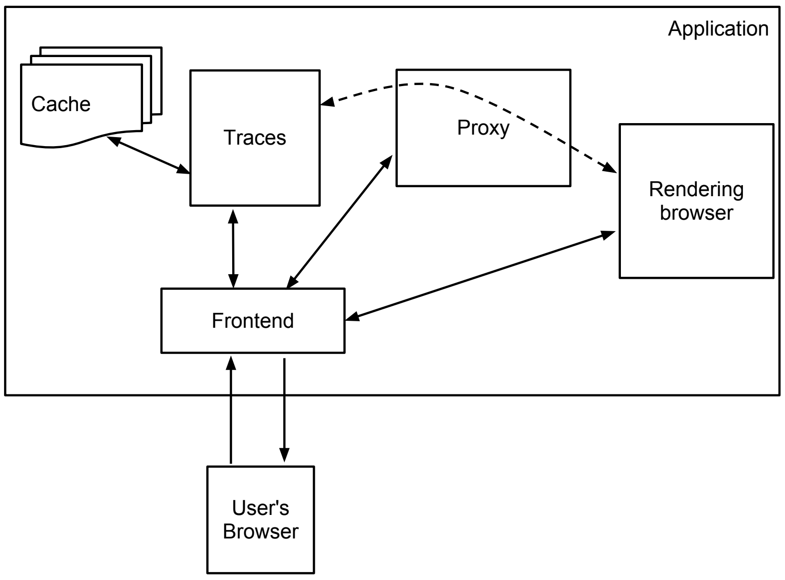

Figure 5: Application design.

needs the cache to see if the requested URL exists in the cache data. The cache data itself must be processed by Traces to be normalized in order to treat different caches the same. After a cache entry is found, the application creates a local proxy server which is coupled to the cache. The application then starts a rendering browser which uses the proxy server to load the resources. After the browser has loaded all resources it provides the application with a screenshot of the web page. The application itself passes this screenshot to the user who can now view how the web page looks based on the available cache data. This is the basic flow of viewing a cached entry. Figure 6 shows the flow from the requested cached entry till the screenshot is taken.

Frontend The frontend is the part which handles the HTTP communication with the user's browser. It receives a request, processes it and returns the response. It is a centralized component in the application design and has a connection with most of the other components.

Cache This component represents the original cache data which is retrieved from a target object like a pc or mobile phone. This is the input for Traces.

Traces Traces is the NFI library (section 1) which provides digital traces based on browser cache data. It converts the original cache data into normalized objects which can be used by the frontend or proxy server.

Proxy The proxy server itself is a short-lived entity within the system. It only lives as long as needed to let the browser load all resources and generate a screenshot. In principle it works by creating a proxy server object which lives in a separate thread. This proxy server is given the specific cache in which the requested URL is located. After initializing a socket it listens on a specific port for requests. When it receives a request it spawns another thread to handle this request. The new thread processes the contents and retrieves the URL requested. If it has found a URL within the request it will search the given cache to see if a match exists. After a match is found, it retrieves the associated content. This content is then used as a response to the requests after which it closes the communication channel and the thread is ended. When the screenshot is taken, the proxy server is no longer needed. The application will shut down the proxy server and the socket is closed.

Rendering browser A screenshot from a website is taken with a rendering browser. In this case the Firefox browser will perform this task. The browser is controlled by the Selenium WebDriver[28] which is a collection of language specific bindings to drive a browser. To be more precise, the Java version for these bindings is used by the application. The application creates a new WebDriver object by defining capabilities such as proxy settings which point to the special created proxy server. It then launches the browser with a temporal profile and point it to the requested URL. The browser then performs a request for the specified URL to the proxy server and gets the associated content returned. Because the browser can use this content to display a website, it is able to retrieve all related resources from this single response. Since most websites are using more than just text, it will probably perform more requests to the proxy server for resources like images, style sheets, JavaScript, etc. All these requests are handled by the proxy server and are matched against the internet cache in order to return the corresponding content. After all the resources are loaded, the browser is ready to take a screenshot. The application invokes this order and gets a screenshot from the WebDriver object. At this moment, the browser has done its job and is closed. The screenshot itself is then ready to be presented to the user.

3.2.1. Technical details

The components itself are a combination of existing closed and open source packages as well as custom code to combine all together and implementing the workflow. The NFI packages are closed source. The browser control components are based on the Selenium framework[26], which has adapters in many different programming languages including Java. For the rendering browser Firefox is chosen because of its maturity, stability and it is cross-platform.

The proxy is a custom written component created for the purpose of this research. It is written in Java[11] and supports multithreading so the application itself is not blocked while running a proxy. It uses the provided browser cache directly as source to feed incoming requests.

The application itself is built on the Play Framework[10] which is a web application framework written in Scala[1] and Java. It follows the Model-View-Controller (MVC)

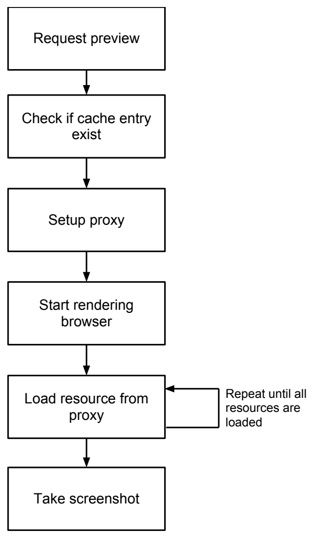

Figure 6: Application flow starting with a request for a cached entry till the screenshot is taken.

pattern[16] and both Scala and Java can be used as application language. The application acts as a container which ties all components together.

4. Results

To be able to determine if the application design works as expected some tests are performed. This is done by selecting a set of websites and creating a cache for this set. The websites selected are ranging from very basic to complex with a lot of elements. The websites and their description can be found in table 2. These websites are visited for different browsers. After creating a cache, it is fed to the application which processes it according to the description in section 3.

Website	**Description**	**Complexity**
`blackle.com`	Search engine	Simple, few images
`google.com`	Search engine	Medium, few image, advanced JavaScript
`blog.i-1.nl`	Weblog	Medium, high amount of static resources
`tweakers.net`	News website	High, high amount of resources
`nu.nl`	News website	High, high amount of resources, ads and JavaScript

Table 2: Websites used for testing the application performance.

At the same time the cache is created, there were also screenshots taken from these websites. This is done so it is possible to reference back to the original presentation of the webpage.

4.1. Performance of simple websites

The search engines and weblog websites are rather simple. When testing these, the websites got rendered without any errors and all resources requested were found in the cache. Comparing the reference screenshot for the Google website with the output of the application shows that little differences could be found (figure 7). Because of the nature of these webpages, simple resources are used like images and Cascading Style Sheets (CSS) files. JavaScript is used to load some resources like these. Comparing this result to the output of a tool as NetAnalysis, this is also added to figure 7 and it clearly shows that NetAnalysis does not manage to handle all resources. It misses the website logo and the position of elements is incorrect.

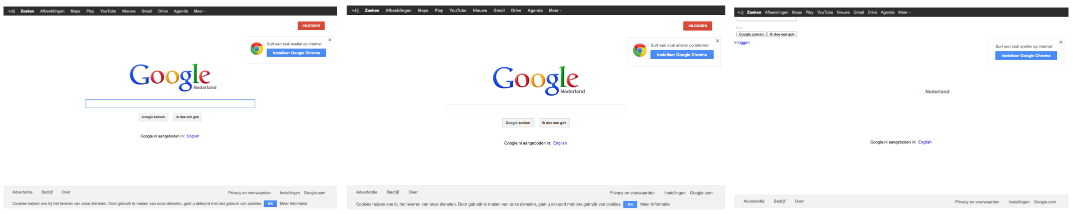

Figure 7: Comparison between the original website (left), the output of the Proof of Concept (center) and the output of NetAnalysis (right).

4.2. Performance of complex websites

More complex websites often contain more advanced resources. This can be observed when the output of the `nu.nl` news website is compared with the reference screenshots (figure 8. We can directly see that most of the advertisements are not shown when the cache is used to preview the website. The data types of these advertisements are often based on Adobe Flash[9] or complex and dynamic JavaScript.

To compare the how complete this result is, the output of NetAnalysis is also add in figure 8. It clearly shows that this tool misses a lot of resources which are not included in the reconstructed web page. Notice that some of the headlines do not match between the left picture and the middle and right pictures. This is due to the fact the website is a highly dynamic page and new headlines appeared in the time the reference screenshot was taken and the sample data was created.

5. Analysis

Now that the mechanism used in this report is proven to be working, it is important to analyze the value of the outcome as described in section 4. To do this, this section will explain two aspects of the cache data and the preview generated from it and the reliability.

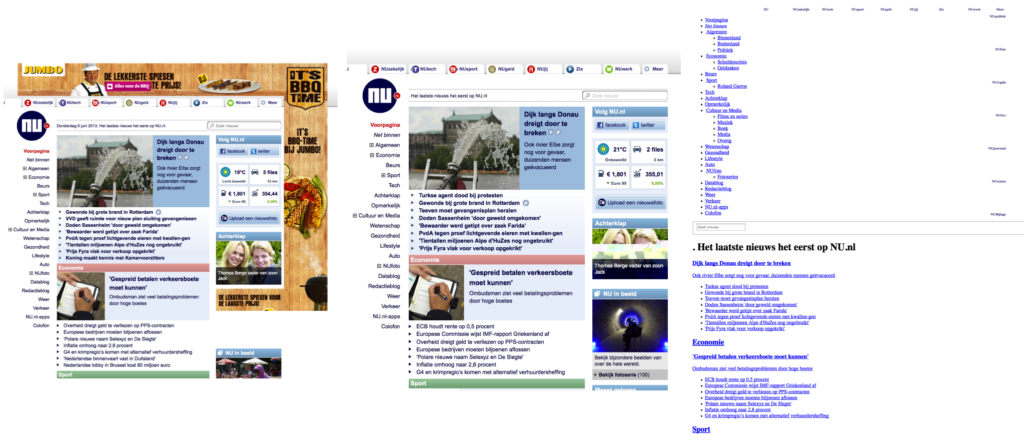

Figure 8: Comparison between the original website (left), the output of the Proof of Concept (center) and the output of NetAnalysis (right).

5.1. Dynamic resources

Dynamic resources have the property that they can change over time or based on other data like geolocation, session and other properties. As an example an advertisement banner can be taken. This banner can have different content every time a web page is displayed. If such a banner is implemented in a language which is depended on plugin software like Adobe Flash, they are not always using the browser cache. As figure 8 in section 4 shows, the advertisement does not exist in the cache data and therefore cannot be retrieved.

Another possible situation which can occur is when a certain image is stored in cache, but that this image is changed after a specified time. The reliability of the preview degrades as the following scenario will show:

1. Browser S displays website W1 on time A.
2. Website W1 contains resource R.
3. Browser S displays website W2 on time B.
4. Website W2 contains resource R, but the content of R has changed.

The above scenario can be visualized as depicted in figure 9 This scenario can represent two different web pages which use the same advertisement image which changes every time it is loaded. Because the browser determines which items are cached and which are not two things can happen in this scenario. The browser may decide to overwrite R with the new version. As a result the previous version of R is removed. When rebuilding W1 and W2, the preview will show both web pages with the same resource R. This means that the preview W1 is not reliable since it browser S originally showed W1 with the old version of R.

The other case is that the browser may decide not to retrieve R from the Internet but instead load it from the cache. This leads to the cache record of R not being updated

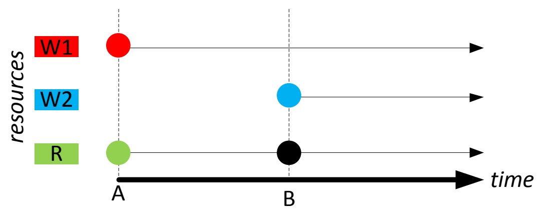

Figure 9: A scenario of two websites using the same resource.

and remain available. When rebuilding W1 and W2, they will both use the same old version of R in the rendered preview. Since browser S used the old version of R for W2, this is not a problem as it represents the web page as it was displayed at time B.

5.2. Runtime dependencies

Information that is displayed on a website can be dynamic. This is often handled by JavaScript code. The information can be dependent on properties that are available during the rendering of the page. Some properties that can be used are time, screen size or random values. These properties are dependent on the moment the preview is created or the machine used. The next scenario shows how time can be a runtime dependency.

1. Browser S visits website W.
2. Website W contains a dynamic time T.
3. Time T is taken from the local system time.

When the dynamic time T is displayed on the web page, this means that this can be different (after the web cache reconstruction) then the time that browser S actually displayed on the website W. The website shown in figure 8 contains such a dynamic element. Just to the right of the logo, a JavaScript code displays the date and time based on the local system time.

6. Conclusion

This research project aimed in providing better understanding of web page reconstruction based on browser cache. Different browsers maintain their cache data in different ways. Although browsers differ in the level in which data types are stored, normalized data contains the same elements which represent the cache data (section 2). Apart from showing that it is possible to use this data to reconstruct a web page, the value of such a reconstruction is also evaluated.

This research paper shows that there are two ways to reconstruct a web page: pre- and post-processing (section 3). Although there are some tools which are able to reconstruct web pages from cache, none of these use the post-processing mechanism. By creating a PoC it is shown that the post-processing mechanism also works. Comparing the results (section 4) between this research and existing tools shows that the post-processing mechanism handles every resource request whereas pre-processing manages this only partly.

To give better understanding about the reconstructed website, an analysis (section 5) shows that the outcome of the reconstruction process may not fully reflect the original presentation. Understanding of the resources which are used within this process is therefore important.

With regards to the constraints of Traces which is responsible of normalization of the cache data, it is possible to create a application design that supports the reconstruction of websites from browser cache data. The PoC demonstrates the viability of this process. The proposed design can be incorporated into existing forensic application frameworks to facilitate this feature to the end user.

7. Further research

Due to the wide scope of browser cache forensics, further research can be done on different aspects to improve usability.

Normalization of browser caches This research paper uses the normalized data that is produced by Traces. The web browser cache data is extracted and normalized in a specific way and only certain parts are collected from the raw cache data. Further research can be done on investigating the way how this data can be enriched with additional types of data like the ones described in section 2.

Devices types With the continuous growth of the usage of smart phones and tablets another field of research opens itself up for mobile web browsers. Since this research paper only focuses on traditional desktop computer and thus desktop browsers, previews are also based on this. Further research may look into the possibilities how page reconstruction can be improved to reflect the way web pages are displayed in mobile browsers.

Dynamic elements As pointed out in section 5 certain parts of web pages can be dynamic that can lead to arguments if a web page that is reconstructed from cache data is really presented exactly the same way as the browser displayed it. The behavior of these dynamic elements regarding to browser caching can provide additional insight in the web page reconstruction process. Additional research on recreating the original environment may improve the reliability of the outcome. One can think of using the original browser environment or change system time when rendering a web page.

References

[1] Scala a general purpose programming language. http://www.scala-lang.org/.

[2] R.A.F. Bhoedjang, A.R. van Ballegooij, H.M.A. van Beek, J.C. van Schie, F.W. Dillema, R.B. van Baar, F.A. Ouwendijk, and M. Streppel. Engineering an online computer forensic service. *Digital Investigation*, 9(2):96 – 108, 2012.

[3] CacheBack. version 3.7.25. http://www.siquest.com/productdetailPage.php?pID=1&mid=2e207d7e57, 2012.

[4] ChomreCacheView. version 1.46. http://www.nirsoft.net/utils/chrome_cache_view.html, 2012.

[5] Chrome. version 27.0.1453.116. http://google.com/chrome, 2013.

[6] Gecko Layout Engine. https://developer.mozilla.org/en-US/docs/Mozilla/Gecko.

[7] Internet Examiner. version 3.9.0. http://www.siquest.com/productdetailPage.php?pID=4&mid=253af8zzyy, 2013.

[8] Firefox. version 21. http://mozilla.org/firefox/, 2013.

[9] Adobe Flash. http://adobe.com/flash.

[10] Play Framework. http://www.playframework.com/.

[11] Java. http://java.com.

[12] an interpreted computer programming language JavaScript.

[13] Keith J Jones. Forensic analysis of internet explorer activity files, 2003.

[14] Keith J. Jones and Rohyt Belani. Web browser forensics. http://www.symantec.com/connect/articles/web-browser-forensics-part-2, 2005.

[15] Michael D Kelly and Sean J Geoghegan. First forensic internet replay sequencing tool. In *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*, pages 270–273. IEEE, 2009.

[16] A. Leff and J.T. Rayfield. Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International*, pages 118–127, 2001.

[17] SQLite Manager. https://code.google.com/p/sqlite-manager/.

[18] MozillaCacheView. version 1.57. http://www.nirsoft.net/utils/mozilla_cache_viewer.html, 2012.

[19] NetAnalysis. version 1.56. http://www.digital-detective.co.uk/netanalysis.asp, 2013.

[20] Junghoon Oh, Seungbong Lee, and Sangjin Lee. Advanced evidence collection and analysis of web browser activity. *digital investigation*, 8:S62–S70, 2011.

[21] ChromeAnalysis Plus. version 1.2.0. http://forensic-software.co.uk/ChromeAnalysis.aspx, 2013.

[22] FoxAnalysis Plus. version 1.2.1. http://forensic-software.co.uk/foxanalysis.aspx, 2013.

[23] WebKit Open Source Project. http://www.webkit.org/.

[24] Safari. version 6.0.5. http://apple.com/safari, 2013.

[25] SafariCacheView. version 1.11. http://www.nirsoft.net/utils/safari_cache_view.html, 2012.

[26] Selenium. version 2.0.0. http://docs.seleniumhq.org/, 2013.

[27] Cache View. version 1.11. http://www.progsoc.org/~timj/cv/, 2005.

[28] Selenium WebDriver. version 2.33. http://docs.seleniumhq.org/, 2013.

A. Acronyms and abbreviations

BLOB	Binary Large OBject
CSS	Cascading Style Sheets
FQDN	Fully Qualified Domain Name
HTTP	Hyper Text Transfer Protocol
MVC	Model-View-Controller
NFI	Netherlands Forensic Institute
PoC	Proof of Concept
SQL	Structured Query Language
URL	Uniform Resource Locator
XIRAF	XML Information Retrieval Approach to digital Forensics