Time skew analysis using web cookies

Björgvin Ragnarsson
University of Amsterdam
System and Network Engineering RP1

February 18, 2013

Abstract

The accuracy of timestamps has important implications in digital forensics. This paper presents two algorithms for determining the system clock skew by using web cookies in a web browser cache. The first algorithm lists possible time skews in a ranked order while the second algorithm finds different time intervals when the computer's system clock had a certain alteration. The algorithms were implemented and tested on modified web browser caches which simulated the system clock being altered or misconfigured at the time the web browser was used. The first algorithm is able to correctly identify the actual time skew as the one with the highest rank. The second algorithm can identify intervals of time skews but suffers from false positive results.

Contents

1 Introduction

The goal of computer forensics is to analyze digital material and exhibit facts derived from it, for instance, reasoning about when certain computer activity took place. Timestamps, such as modified, accessed & created (MAC) times in filesystem metadata are commonly used to anchor down the time of events, however, their accuracy cannot be trusted without further examination. According to Schatz, Mohay, and Clark [1], the factors affecting timekeeping accuracy are:

- The *system clock implementation* uses unstable crystal oscillators. Region specific timezone changes further complicate the implementation of the system clock.
- Incorrect *clock configuration* such as timezone settings.
- The computer clock can be *deliberately modified* to generate incorrect timestamps.
- Computers using NTP and SNTP *synchronisation protocols* without cryptographic authentication are open to protocol-based attacks.
- *Misinterpreting* timestamps, such as assuming that the timezone offset on the machine under investigation applies, when the timestamp was generated elsewhere.
- Software *bugs* can affect timekeeping accuracy.

The limitations of the system clock for accurate timekeeping were recognized by Weil [2] who proposed dynamic timestamp analysis. Weil infers the time skew of a particular computer from web pages in a browser cache. Some of web pages have dynamically generated timestamps set by the server and the time skew is the difference between the creation time of the cached web page and the server generated timestamp inside it. Weil concludes that "when multiple sources of time are present, the reliability of approximate actual system time and date increases. In general, more independent sources of verification increase the reliability of the data because the external systems also utilize their individual system time and date". This approach has notable drawbacks. Firstly, the manual work is time consuming and prone to errors. The approach requires an investigator to go through a web cache and distinguish between static web sites with constant timestamps in them and dynamic sites where the timestamp is generated upon request of the page. Secondly, there is no measurement of the reliability of the approximate actual system time and date.

Schatz et al. describe an automated method of finding time skew by correlating web browser cache records to logs from web proxies operated by ISPs. While the algorithm performed well, it is of no use when proxy logs are not available.

1.1 Problem statement

Previous work can be improved by creating automated way to find the time skew without relying on digital traces outside of the computer under investigation. The problem statement of this paper is to develop an automated method for

finding the time skew of a computer by comparing its system clock to external sources of time from browser caches. The original idea was to arrive at a measure of confidence for a certain system's time skew, by using Bayesian analysis on web cookies and a database of prior probabilities. Bayesian analysis was not applicable for the dataset we decided to work with as discussed in Section 3.1 so other methods were used. These methods were put under test using software developed in Python and the results evaluated.

The paper is structured as follows. Section 2 looks at what web cookies are made up of, how they are transported from servers to web browsers and how browsers store them. Finally, the section introduces a simple method for finding a system's time skew from a cached cookie and discusses its complications. Section 3 presents analysis on a corpus of HTTP header responses for the purpose of finding which cookie expiry times are likely to be used by servers. An algorithm for ranking possible time skews on a machine is presented and tested in Section 4. This algorithm takes into account previously discussed complications and makes use of the expiry times from Section 3. A second algorithm for finding multiple time skews is presented in Section 5 and put under test. Finally, conclusions are drawn in Section 6.

2 Anatomy of web cookies

HTTP is a stateless communication protocol in its essence. Web cookies were introduced by Netscape Communications as a mechanism to have the client keep track of stateful information between HTTP requests. Such stateful information can be a list of products put in a virtual shopping cart by the user of a web browser. The cookies need to be available to the server when the customer has made up his mind and places his orders. When the user sends a request to put a product in the shopping cart, the server includes a cookie in the HTTP response. The cookie is a key-value pair with information about which product was selected and for how long the information should be store in the browser cache until it is discarded. The following is an example of how an HTTP response with a cookie looks like.

```
HTTP/1.0 200 OK
Date: Fri, 21 Sep 2012 05:51:31 GMT
Status: 200 OK
Set-Cookie: pId=17; expires=Fri, 28-Sep-12 05:51:31 GMT; domain=example.com
```

This cookie is supposed to be stored in the browser cache for a week before it is discarded. The browser stores the information from the cookie along with a creation timestamp which the browser gets from the system clock. An alternative to the `expires` attribute is `Max-Age`. An HTTP header field which sets a cookie with `Max-Age` could be

```
Set-Cookie: pId=17; Max-Age=604800; domain=example.com
```

where 604800 is the number of seconds in 7 days. This method of setting the cookie expiry was included in the original RFC 2109 specification for cookies [3] while it is the recommended way in the currently proposed update, RFC

6265 [4]. Browsers are moving to adopt this standard but if older versions of browsers are to be supported, servers have to use the `expires` attribute. RFC 6265 allows both attributes to be used at the same time but `Max-Age` should take precedence. Further look into how common the usage of the `expires` and `Max-Age` attributes, can be found in Section 3.2.

2.1 The cookie database in Firefox

Browsers implement the cookie store differently but they contain essentially the same information from platform to platform. We chose to use the cookie database from Firefox for this project because it is easy to work with. Firefox stores cookies in the user's profile directory in a file called `cookies.sqlite`. The file format is sqlite3, a single file relational database and is easy to interface with using Python libraries and the SQL query language. The following is an example of values stored in a Firefox cookie database.

```
id:           9768
baseDomain:   os3.nl
name:         squirrelmail_language
value:        deleted
host:         webmail.os3.nl
path:         /
expiry:       1361966744
lastAccessed: 1359374744772316
creationTime: 1346668747276127
isSecure:     1
isHttpOnly:   1
```

The creation time is stored in nanoseconds while the expiry is in seconds.

2.2 Clock skew from cookies

The clock skew of a system can be found by comparing the server generated expiry date to the client generated creation date from a cookie stored in a browser cache. One might be tempted to assume that a cookie with a local creation timestamp of 12:44 on March 7th, but a server defined expiry time of 12:53 on March 14th, was created on a machine with a 9 minute clock skew. However, this method has its complications:

- The assumed expiry time for the cookie as intended by the HTTP server is grounded on a baseless assumption. The example above assumes a week of expiry time but there is nothing stopping an administrator from setting the expiry to a week and nine minutes.
- It is unknown whether the browser got the expiry information from the `expires` attribute or if it is derived from its own system clock because of interpretation of the `Max-Age` attribute.
- Cookies can be modified after the browser caches them. Many websites use Javascript to create cookies or modify them. Javascript generated

cookies will have an expiry coming from from the system clock.

- The HTTP server sending the cookie could also have a clock skew. A study by Buchholz and Tjaden [5] showed that 74% of web servers were within 10 seconds of reference time (UTC) for the 6 month period of the study.

The rest of the research focused these issues and solutions to them.

3 Finding popular cookie expiry times

To find popular HTTP cookie expiry times we used the Shodan HQ HTTP Header Survey [6]. The survey is a collection of HTTP responses from the 10.000 most popular sites on the web according to Alexa Internet. Each website was requested 14 times, each time with a different user agent to imitate the behavior of different browsers. The responses were recorded and published as tabular data, `top10k_ua.csv.gz`.

3.1 Computing expiry times

We came up with a method of subtracting the cookie `expires` attribute from the `Date` header field to find for how long the web server intended the browser to store the cookie. This assumes that the `expires` timestamp is the same as the `Date` timestamp, with an added delta. This assumption turned out to be true most cases but required filtering out HTTP responses where this relationship between the dates did not exist. The filtering was done by calculating a delta for the 14 responses from each server and calculate their standard deviation. Some variance was allowed as time passes between the generation of the `expires` attribute and the `Date` header field and that may result in an extra second added to the delta even though the creation of the HTTP header as a whole takes only a few milliseconds. After filtering, 232 deltas were identified as the only expiry times used by the 10.000 servers. This number is an upper limit and rare deltas might not be correct because of misinterpretation of timestamp relationships or delay between their generation. The distribution of the deltas calculated from all cookies in the HTTP Header Survey can be seen in figure 1.

At first glance it might be look as the distribution of the deltas from the HTTP Survey could be used to make predictions about the expiry times of cookies in browser caches and used as a prior probability in Bayesian calculations. When figure 1 is compared to figure 2 which shows expiry deltas from a Firefox cookie database, it is obvious that this is not the case. The reason is that as time passes, cookies with a lower expiry date are discarded by web browsers while cookies with an expiry further in the future are accumulated. Another reason for the difference in the distribution could be that the web servers sending the cookies in the Firefox cache behave differently than the top 10.000 sites on the web.

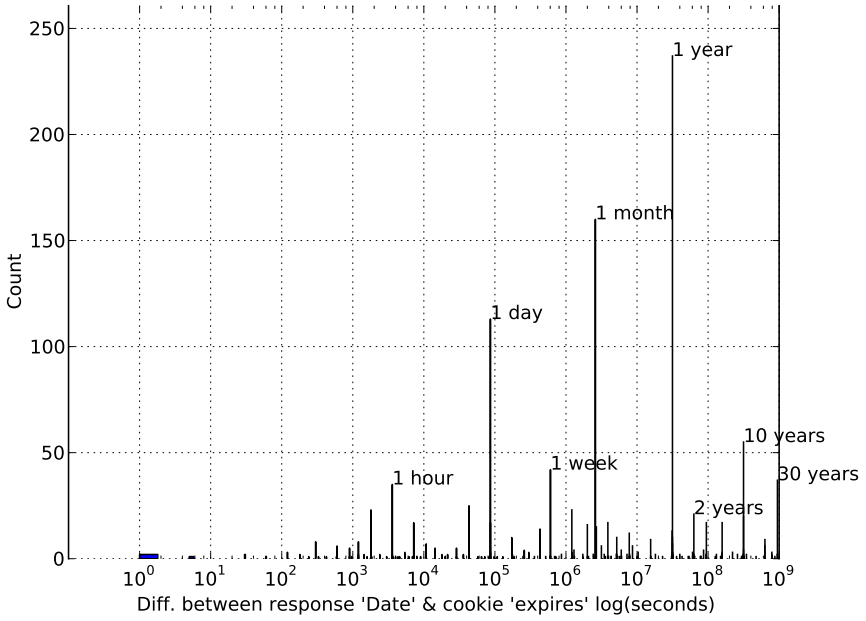

Figure 1: Histogram of expiry deltas of cookies from the HTTP Header Survey. Logarithmic axis is used to better display the wide distribution of deltas.

3.2 Other facts derived from the HTTP Survey

Table 1 shows some statistics derived from the HTTP Header Survey. The ratio of cookies with a `Max-Age` attribute is 2.8% out of cookies using either method for setting expiry. This percentage is likely be higher in future surveys as the `Max-Age` attribute is the recommended way to set the expiry as of the proposed standard, RFC 6265 [4]. The `HttpOnly` attribute is seen in 11.7% of cookies. `HttpOnly` is a security feature to mitigate theft of sensitive cookie values using cross-site scripting. Browsers which interpret the `HttpOnly` attribute correctly do not allow Javascript to modify cookies using it.

4 Ranking possible skews

The first algorithm developed identifies possible time skews on a computer and ranks them in an order of confidence. The algorithm makes use of expiry times of web cookies from a web browser cache as an external time source and the creation time as the local system clock. Therefore, it only able to determine the time skew of a computer for the period of time found within the creation time of the matching web cookies. That period is skewed and the corrected period can be found by subtracting the assumed skew from it.

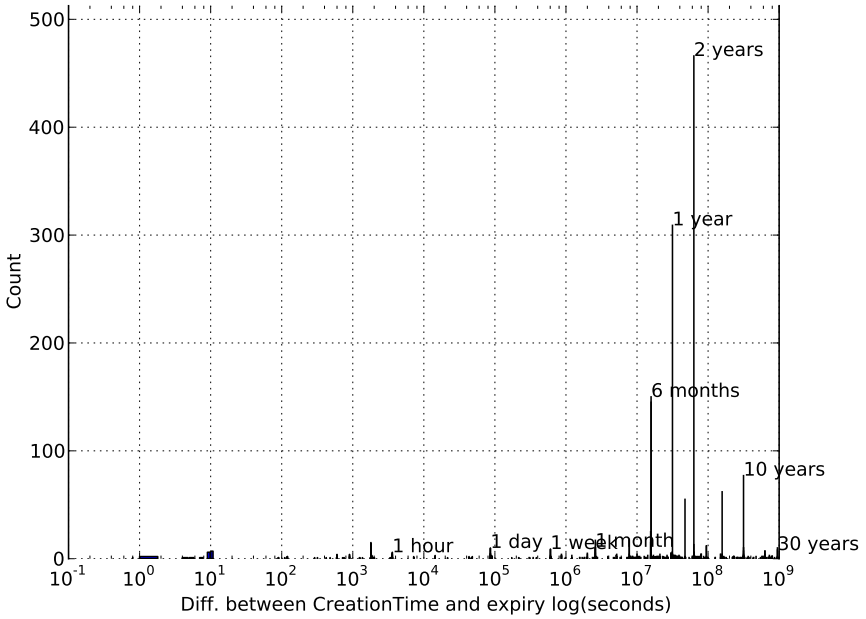

Figure 2: Histogram of expiry deltas of from a Firefox cookie database after 5 months of daily usage. The computer running the browser was regularly synchronized to a timeserver.

4.1 Assumptions

While designing the algorithm, assumptions about which time skews are commonly found in browser caches were avoided. The general assumption made about browsers was that they acts according to current RFC standards [7, 4]. This includes assuming they correctly interpret the `Max-Age` and `HttpOnly` cookie attributes.

Assumptions had to be made about the cookies themselves from the cache. Certain proportion of cookies are assumed to have the `Max-Age` attribute set and therefore have an unusable the expiry time for time skew calculations. It was mentioned in Section 3.2 that 2.8% of cookies in the HTTP Header Survey had the `Max-Age` attribute set and the same fraction is assumed to be seen in browser caches. Modification of cookies at the client side is also an issue. For cookies which do not have the `HttpOnly` attribute set, it is assumed that the chance that they have been created or had the expiry modified by Javascript code is 20%. Because of time constraints of the project, this probability could not be investigated further. Both the `Max-Age` frequency and the probability of Javascript modifications are set with parameters in the algorithm implementation.

The algorithm which is described in Section 4.2 ranks a certain client time skew higher as more cookie expiry times "agree" on it. It is important that servers with wrong time skews do not aid to incorrect ranking. The assumption is that

Survey date	2012/09/22
Web sites requested	10.000
Number of User agents used	14
Responses recorded	132.182
Cookies in responses	59.453
Cookies with both Max-Age and expires	481
Cookies with only Max-Age	355
Cookies with only expires	28.764
Cookies with the HttpOnly attribute set	6.937

Table 1: Statistics on the HTTP Header Survey

most web servers have their clock set correctly and that the rest of them have a varying skew. This is according to the previously discussed findings by Buchholz and Tjaden.

4.2 The first algorithm

The following pseudo code shows the workings of the algorithm. It runs in two loops. The first loop finds possible skews for each cookie in the browser cache and the second aggregates the confidence ranking from each unique possible

skew.

Input: Web cookies from a browser cache.
Input: List of bad expiry dates. These dates are assumed to be constants.
Input: Probability of a cookie being generated by Javascript. 20% is used an assumption.
Input: Ratio of cookies with the `Max-Age` attribute set. 2.8% according to the HTTP Header Survey.
Input: List of possible expires-deltas found from the HTTP Header Survey.
Input: List of cookies and their expires-deltas as known from the HTTP Header Survey.
Output: List of possible skew-rank pairs.
for *each cookie in the browser cache* **do**
  if *cookie expiry is not in list of bad expiry dates* **then**
    if *cookie has the HttpOnly attribute set* **then**
      probability of the cookie expiry being modified by Javascript is 0%
    else
      probability of the cookie expiry being modified by Javascript is 20%
    end
    if *cookie is known from the HTTP Header Survey* **then**
      if *does the server set this cookie with the* `Max-Age` *attribute?* **then**
        probability that the cookie can be used because of the `Max-Age` attribute is 0%
      else
        probability that the cookie can be used because of the `Max-Age` attribute is 100%
      end
      calculate one possible skew by subtracting the expiry delta of the cookie in the browser cache from the expiry delta from the cookie in the HTTP Header Survey.
    else
      probability that the cookie can be used because of the `Max-Age` attribute is 2.8%
      for *each delta from the list of possible expires-deltas* **do**
        calculate a possible skew by subtracting the expiry delta of the cookie in the browser cache from the expiry delta from the cookie in the HTTP Header Survey.
      end
      all skews calculated in the previous loop are considered possible
    end
    probability that the cookie is usable for identifying a clock skew is the multiple the probabilities of it not being modified by Javascript and not having an expiry derived from the `Max-Age` attribute
end
for *each skew previously found to be possible* **do**
  set the skew rank as the sum of usable-cookie probabilities from all cookies agreeing on this being a possible skew
  divide rank by the total sum of cookies for normalization
end

Algorithm 1: Ranking possible clock skews

The top ranking skew-rank pairs are displayed for evaluation.

4.3 Implementation and testing

The algorithm was implemented in Python and tested on a database of 2899 cookies from Firefox which had been in use for five months. The machine was running Ubuntu 12.04 and running NTP with default settings to synchronize time. Two tests were performed. One on the unmodified cookie database and another where the creation time of the all cookies were set 83 seconds back in time, simulating a clock skew on the machine.

4.3.1 Test results from an unmodified cookie database

```
$ make showskews0s
./skewy.py -z ../shodan/zodan.db -c cookies.sqlite \
                -bdl ../shodan/BDL.curated -p -j 0.2 -m 0.028
skew,rank,cookiecount,cookieratio
0,0.37,1326,0.47
63072000,0.32,1185,0.42
86400,0.27,990,0.35
31536000,0.25,911,0.32
-864000,0.23,828,0.29
-31449600,0.22,808,0.28
94608000,0.22,803,0.28
31622400,0.22,793,0.28
-31535999,0.22,789,0.28
283824000,0.21,781,0.27
...
```

4.3.2 Test results on a modified cookie database

```
$ make showskews83s
./skewy.py -z ../shodan/zodan.db -c 83sback.sqlite \
                -bdl ../shodan/BDL.curated -p -j 0.2 -m 0.028
skew,rank,cookiecount,cookieratio
-83,0.31,1104,0.39
63071917,0.26,936,0.33
86317,0.22,780,0.27
31535917,0.20,719,0.25
-31449683,0.19,677,0.24
-864083,0.18,665,0.23
283823917,0.18,657,0.23
94607917,0.18,651,0.23
-31536082,0.18,644,0.23
15767917,0.17,639,0.22
...
```

4.3.3 Discussion

The skew with the highest ranking is the correct one in both cases. The second highest rated skew in both cases is an addition of two years. The creation timestamps range from September 2012 to January 2013 which would require the actual date when the cookies were created to be between September 2010 and January 2011. This time skew is inconsistent with other known facts, such as the computer was installed in the year 2012. Another highly ranked skew assumes the clock was set one year in the past and is easily dismissed as time travel is uncommon. By making use of known facts, incorrect skews can be removed which widens the gap between the hightest rated skew and the second one.

5 Finding multiple skews

The algorithm from Section 4.2 is intended to find a single time skew but that might not always reflect reality. An interesting case is a deliberate modification of the system clock which the second algorithm is meant to detect. It uses web cookies as a source of time skew information as the previous algorithm but tries to find all clock skews of the system spanning the period when the cookies were created in the browser cache.

5.1 The second algorithm

The algorithm identifies periods when a certain time skews could have been in place. It does so by finding groups of cookies which share a common possible skew and identifies the period spanning the maximum and minimum creation time of the cookies as a period when the skew could have been in place. The possible skews for each cookie is found using the same steps as in algorithm 1. The number of cookies in a group is a parameter to the algorithm. To limit false positive results, every group of cookies is selected so all of them have a different expiry-creation delta and therefore have more independent matches. If this was not done, a group consisting only of cookies with the same expiry-creation delta - and therefore the same possible time skews - would positively identify a period where all the possible time skews were in place at the same time. The final limitation on groups is that that if there is a cookie with a creation time between the minimum and maximum creation time of cookies in a group, it has to also be in that group. The reason why this is done is because it is more likely that cookies closer in creation time share the same time skew and are therefore put in the same group.

5.1.1 Implementation and testing

The algorithm was implemented using Python and tested with the same cookie database as used in algorithm 1. The database was modified so all cookies created between 2012-11-20 and 2012-11-27 were changed to simulate a system

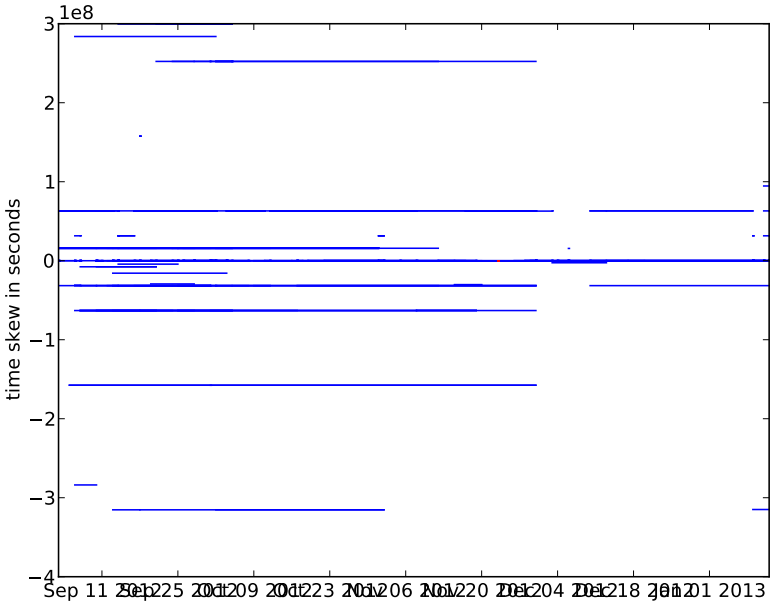

Figure 3: Group size 4, possible skews are calculated from all 232 server-deltas. A lot of false positives are seen.

clock skew of 3 days back in time. Different cookie group sizes were tried but Figures 3 and 4 show testing results with group size of 4. Initial testing showed a lot of false positives and essentially unusable results. After changing the algorithm to use the 6 most common expiry deltas from the HTTP Header Survey to generate possible skews instead of all 232 of them, the false positives were gone.

6 Conclusion

The research proved successful in identifying a single time skew from cookies in a browser cache. This was done without making assumptions about which time skews are more probable than others. Trying to further identify multiple time skews within a certain time period turned out to be challenging and the method used cannot be considered to be reliable.

Further work is needed to investigate the impact of Javascript generated cookies on the methods used. Also, the software needs to modified to support other web browsers than Firefox, such as the popular Internet Explorer. Other improvements to the software could be to add an option to add cookies to the server response corpus which currently only consists of the HTTP Header Survey. This would lower the number of unknown cookies and improve the ranking.

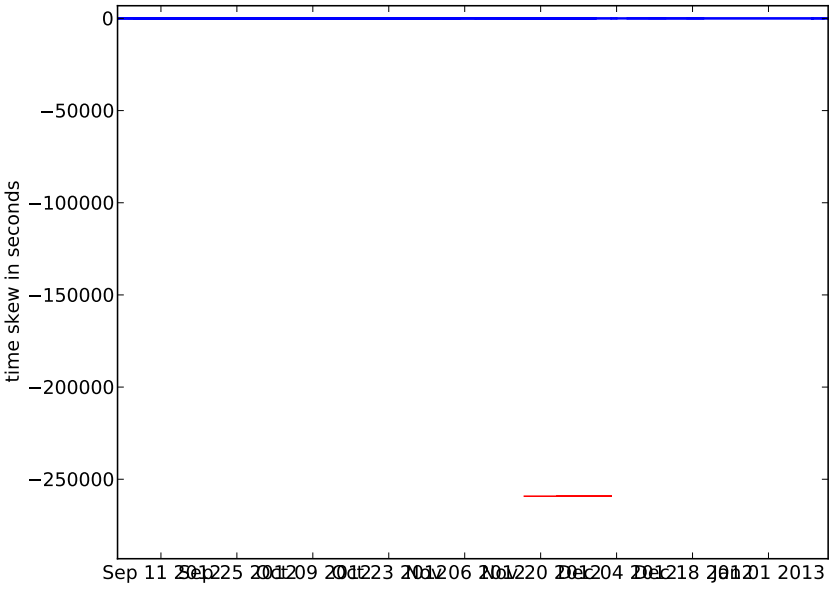

Figure 4: Group size 4, possible skews are calculated from the 6 most frequent server-deltas. The starting point for the period of the time skew is correct but the period is too long.

The viability of these methods for time skew analysis depends on common usage of the `expires` cookie attribue which vast majority of web sites currently use. However, as `Max-Age` is now the preferred method of setting the cookie expiry, the future might not be bright for time skew analysis using web cookies.

7 Acknowledgements

This paper describes research done for the most part in cooperation with Wicher Minnaard. We worked together on development and implementation of the first algorithm but the final version described in this paper does not represent his results from the research.

I would like to thank my supervisor at Netherlands Forensics Institute, Marnix Kaart.

References

[1] B. Schatz, G. Mohay, and A. Clark. "A correlation method for establishing provenance of timestamps in digital evidence". In: *Digital Investigation* 3 (2006), pp. 98–107.

[2] Michael C. Weil. "Dynamic Time & Date Stamp Analysis". In: *International Journal of Digital Evidence* (2002).

[3] D. Kristol and L. Montulli. *HTTP State Management Mechanism*. RFC 2109 (Historic). Obsoleted by RFC 2965. Internet Engineering Task Force, Feb. 1997. URL: http://www.ietf.org/rfc/rfc2109.txt.

[4] A. Barth. *HTTP State Management Mechanism*. RFC 6265 (Proposed Standard). Internet Engineering Task Force, Apr. 2011. URL: http://www.ietf.org/rfc/rfc6265.txt.

[5] Florian Buchholz and Brett Tjaden. "A brief study of time". In: *Digital Investigation* 4, Supplement.0 (2007), pp. 31 –42. ISSN: 1742-2876. DOI: 10.1016/j.diin.2007.06.004. URL: http://www.sciencedirect.com/science/article/pii/S1742287607000394.

[6] *HTTP Header Survey - Analyzing the Top 10,000 Websites' HTTP Headers*. Survey date: 22/09/2012. URL: http://www.shodanhq.com/research/infodisc.

[7] D. Kristol and L. Montulli. *HTTP State Management Mechanism*. RFC 2965 (Historic). Obsoleted by RFC 6265. Internet Engineering Task Force, Oct. 2000. URL: http://www.ietf.org/rfc/rfc2965.txt.