Retroactively estimating system clock skew from stored web browser cookies

Wicher Minnaard (wicher.minnaard@os3.nl)

March 19, 2013

A person with one watch knows what time it is;
a person with two watches is never sure.

Abstract

Cookies as stored by web browsers can provide a record of local system clock skew. Deriving this clock skew is fraught with difficulties. By taking probability into account and by applying statistics derived from empirical data, it becomes feasible to estimate skew in spite of many uncertainties and pitfalls inherent in the mechanisms involved.

This research project report presents the considerations that need to be taken into account when deriving skew from web browser cookies. It is accompanied by open source software that implements the algorithms proposed.

Contents

1 Introduction

This research project report and associated software[1] are the result of a one-month effort to create and automate a forensic method for retroactively determining a system clock skew record of some end-user system, by using web cookies found on that same system and comparing them with a collection of cookies retrieved from public web servers.

1.1 Rationale

The reliability of time stamps plays an important role in forensic investigations. Time stamps are used to determine the relative ordering of associated events and to correlate these events with other events that are external to the system. Knowing if (and by how much, and when) the clock that was used in setting time stamps was skewed with respect to other clocks can be critical.

The importance of knowing such a skew deserves to be illustrated:

> John is a suspect in a fraud case – supposedly, he has tampered with the electronic cash register (PC software) in the grocery shop where he is employed. John claims that he did no such thing and that some other person working the next shift must have been responsible.
> A forensic investigation of security camera footage shows John leaving the store premises at 13:00. The investigation also shows that the fraudulent records were timestamped 13:03.
>
> The question is obvious: *what was the skew of the PC's clock* (which has set the record timestamps) *with respect to the clock of the security camera* (which has set the video footage timestamps) ?

CERT advises first responders to register system clock skew when collecting evidence. [5, section 3.9.1.2] Unfortunately, this will only provide the forensic examiner with knowledge of the skew at evidence collection time. To extrapolate that skew to a point in time months or years prior to capture is a far stretch.

Using his experience and implicit notions of what he considers 'likely', a forensic examiner might arrive at a conviction of the relative order of events. Such personal convictions might suffice for an internal investigation – e.g. to find out in which way some system was breached – but they do not hold up in criminal court, where it may come to the point that some relative order of events has to be proven beyond reasonable doubt. Boyd and Forster note the attention time information receives in court:

> Date and time evidence is a fundamental part of many forensic computing examinations. Forensic examiners know that lawyers are often drawn to dates and times because they represent a concrete link between the real world and the less easily understood world of computer

[1] Available at `http://nontrivialpursuit.org/cookieskewer/cookieskewer.tar.gz`. This software is meant for research purposes and is not designed with production purposes in mind.

> evidence. Experienced examiners know that date and time evidence is not simple and contains many potential pitfalls. Usually the more knowledge and experience that an examiner possesses the less they are willing to commit to a particular time or date. They will always try and look at the fuller picture seeking corroboration or verification of their findings. [2]

What is needed is a historical record of offsets against some reference clock. Ideally, system logs contain such information[2] – see listing 1 on the following page for an example of such a log. Lacking an accurate record, forensic investigations dealing with time information often turn to other, less perfect traces of time information set by clocks external to the system.

As it happens, many systems are not run in isolation. Network protocols may carry time information as part of a mechanism employed to maintain state or caches. A prime example of such a protocol is HTTP. It is pervasive due to the widespread use of the world wide web. The storage mechanisms that HTTP clients employ to maintain state and caches make traces of external time prevalent on many end-user systems, ranging from laptops to smartphones.

1.2 Research goals

The research goals that follow from the rationale are straightforward:

1. Determine to what extent the mechanisms that are involved in the HTTP state mechanism allow the web client cookie store to be used for estimating skew. What are the caveats and how can they be dealt with?
2. Devise an automated method that, given a web cookie collection and a database of cookie characteristics, arrives at an estimate of clock skew. Implement a prototype.
3. Evaluate the method.

The structure of the remainder of this report is as follows:

- ▷ Sections 2 and 3 address goal #1.
  Surveying and classifying empirically gathered data has taken up a major part of the effort spent on this project.
- ▷ Sections 4 and 5 address goal #2.
  Coming up with an algorithm and building software has taken up another major part of the effort.
- ▷ Section 6 addresses goal #1 and #3.
  Arguably, this is where the most important question – *"can it be done?"* – will need to be answered. The short answer is *"yes"* – initial results are promising. Many questions addressing the limits of the approach remain unanswered, though not unmentioned.

[2] Those logs could be tampered with, so one would still like to corroborate them with other sources of time information.

Listing 1: Example of an NTP log from a GNU/Linux system running OpenNTPD. It contains a fairly fine-grained historical record of time skews, and mentions the external time sources it has synced up with.

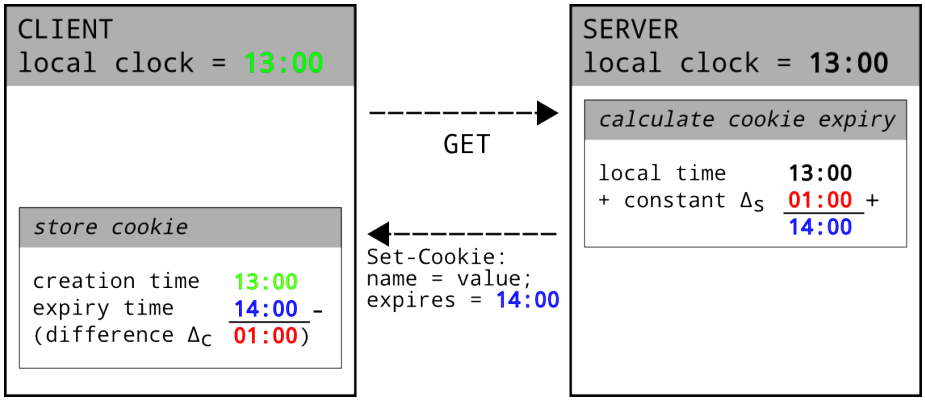
```
Jan 31 10:43:29 [ntpd] peer 96.226.123.84 now valid
Jan 31 10:51:23 [ntpd] adjusting local clock by 0.597745s
Jan 31 11:00:32 [ntpd] adjusting local clock by 0.377247s
Jan 31 11:00:32 [ntpd] clock is now unsynced
Jan 31 11:04:48 [ntpd] adjusting local clock by 0.214659s
Feb 05 11:14:26 [ntpd] adjusting local clock by 2.033003s
Feb 05 11:18:49 [ntpd] adjusting local clock by 1.945286s
Feb 05 11:23:04 [ntpd] adjusting local clock by 1.823792s
Feb 05 11:27:17 [ntpd] adjusting local clock by 1.675674s
Feb 05 11:31:37 [ntpd] adjusting local clock by 1.589458s
Feb 05 11:35:59 [ntpd] adjusting local clock by 1.457986s
Feb 05 11:40:17 [ntpd] adjusting local clock by 1.295471s
Feb 05 11:44:36 [ntpd] adjusting local clock by 1.158376s
Feb 05 11:48:48 [ntpd] adjusting local clock by 1.034453s
Feb 05 11:53:13 [ntpd] adjusting local clock by 0.912603s
Feb 05 11:57:26 [ntpd] adjusting local clock by 0.801233s
Feb 05 12:01:40 [ntpd] adjusting local clock by 0.671071s
Feb 05 12:05:54 [ntpd] adjusting local clock by 0.527679s
Feb 05 12:10:14 [ntpd] adjusting local clock by 0.446749s
Feb 05 12:14:36 [ntpd] adjusting local clock by 0.286010s
Feb 05 12:34:23 [ntpd] skew change 60.748 exceeds limit
Feb 05 12:34:23 [ntpd] clock is now synced
```

2 Method for deriving skew from web cookies

2.1 External time carried in cookie expiry information

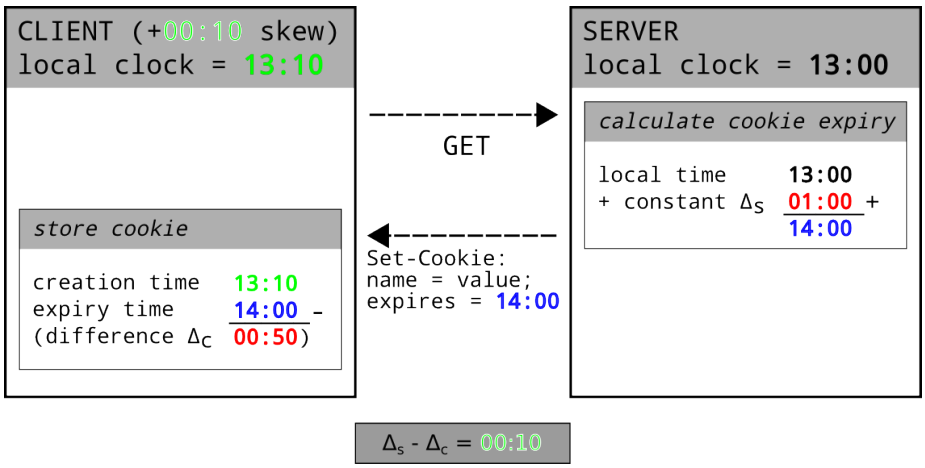
A web browser cookie can carry time information that reflects the state of a web server's clock at the time of the request that led to the creation of the cookie on the client side. The following schema (Figure 2.1) illustrates how information on the state of the server clock can end up on the client.

Figure 2.1: External time time incorporation (without skew). A client sends a request. In the response, the server sets a cookie. To determine the expiry of the cookie, the server adds a constant Δ_s (of, in this case, one hour) to the current time, which it derives from its local clock. The client stores the cookie with the expiry timestamp received from the server, and timestamps the cookie with a creation time that is derived from its local clock. The difference between these two timestamps on the client side is Δ_c. Note that Δ_s and Δ_c are identical.



If a clock skew is introduced in the client system, the mechanism results in artefacts from which that same skew can be derived ex post. See Figure 2.2 on the following page.

Figure 2.2: External time time incorporation (with skew). A client sends a request. In the response, the server sets a cookie. To determine the expiry of the cookie, the server adds a constant Δ_s (of, in this case, one hour) to the current time, which it derives from its local clock. The client stores the cookie with the expiry timestamp received from the server, and timestamps the cookie with a creation time that is derived from its local clock. The difference between these two timestamps on the client side is Δ_c. Note that Δ_s minus Δ_c yields the skew.



2.2 Exploiting expiry information

Given a cookie C from the browser's cookie storage, we can retrieve Δ_c simply by subtracting the creation time from the expiry time – both these timestamps are metadata stored with the cookie. And if we also know the Δ_s that was used by the server in the creation of C, we can calculate a skew S : $S = \Delta_s$-Δ_c .

How then to determine Δ_s? If we know the HTTP reply that lead to the creation of C, and the time that response was made, we can subtract that time from the expiry in the reply – this gives us Δ_s. But we will first need to match the cookie to the HTTP reply. To do so, we need to define an indentity for a cookie.

2.2.1 Identity of a cookie

Which attributes of a cookie can we use to match it to some recorded HTTP reply?

The authoritative definition of how web browsers and web servers should handle cookies is drafted by the IETF.[3] The most current definition is in RFC6265 - *HTTP State Management Mechanism*. [1] That definition is relatively recent – the mechanism has been defined in two earlier incarnations as RFCs 2965 and 2109.

[3]Internet Engineering Task Force; http://www.ietf.org/.

As these RFCs have been compromises between prescription on the one hand, and description of the status quo of web browsers and web servers on the other, they are not guaranteed to describe the behaviour of any particular web browser found in the wild. [8, pp. 60-62]

When a cookie is generated on the web server, it is sent to the client in the `Set-Cookie` HTTP reply header. Apart from a payload – a name-value pair – this header value contains metadata that indicate to the web browser the *circumstances* under which it is to send the payload on following requests.

A typical `Set-Cookie` header has the following appearance:

```
Set-Cookie: foo=bar; path=/; expires=Fri, 21-Dec-12 00:00:00 GMT; domain=.baz.com
```

`foo=bar` are the name and value for the cookie; this is what the web server wishes to receive from the browser on subsequent requests. The rest of the fields are optional. For some of them, a default value is assumed, for others, a value is derived by the browser based on the request URI. The saved metadata fields will be matched against the request URI and system clock on subsequent requests.

Using section 5.3 "Storage Model" of RFC6265 as a reference, we can infer a definition for the identity of a cookie that serves our purpose of matching stored cookies to recorded HTTP replies. Two necessities aid us in this:

- ▷ When a browser is steered towards a web site it needs to decide if it has any cookies in store that need to be sent along with the request.
- ▷ Conversely, when a server sends a browser a cookie, the browser will need to decide whether it is going to store a new cookie or update one of its stored cookies.

When sending a request, the browser will match the request URI to its stored cookies based on their path and domain attributes.[4] Therefore, path and domain are logical components of the identity.

When receiving a reply, the browser will again match the path and domain. In addition, it will try to match the name. If a match is found it will update the value of an existing cookie with that name, instead of creating a new one. Clearly, the name is part of the identity.

To match a stored cookie to the server response from which it originates, we will use the combination of name, path and domain:
$id(C) = \{name, path, domain\}$. From hereon, we refer to such identities as **CID**s.

This identity definition aligns well with collision directive 1 of step 11 in the RFC storage model, which notes:

> Let old-cookie be the existing cookie with the same name, domain, and path as the newly created cookie. (Notice that this algorithm maintains the invariant that there is at most one such cookie.) [1]

[4] The standards are unclear on whether protocol security (identified by the `secure` attribute) and exclusive access (`httponly` attribute) are part of the identity of a cookie. More specifically – if a browser receives two cookies, with the same name, domain and path, one of which is carrying the `httponly` attribute – will it store two cookies or just one? [7]

2.3 Caveats

We have shown a mechanism that allows us to derive a skew S. However, this mechanism relies on circumstances that may not always be met. To make matters worse, it is not always possible to tell whether they have been met.

2.3.1 Cookie creation time is misleading

RFC6265 [1], section 5.3 dictates how cookie updates should be handled. On the topic of creation times it notes that if a cookie is updated, the creation time is to be preserved.

The result of this mechanism is that Δ_c can only be determined if the modification time of the cookie is known – for if a cookie is modified, its creation time becomes useless and Δ_c should be calculated from the modification time instead. To serve an example: Suppose, at $t = 0$, a web client receives a cookie that expires at $e = 100$. The creation time will be set at $c = 0$, and its expiration at $e = 100$. At $t = 50$, the web client visits the server once more, and the server sends the client a new version of the cookie with an updated value (and expiration at $e = 150$). The client will update its cookie with the new value and a new expiration, but the creation time will be preserved. When calulcating Δ_c using the creation time, and having established that $\Delta_s = 100$, it will seem like the client's clock was skewed – while it was not.

2.3.2 Expiration metadata is not always derived from an external clock

max-age

Instead of `expires` – an absolute time, the server can use the `max-age` attribute to specify an offset to the current time. The client will add this offset to its current time. When this mechanism is used, the cookie does not store time that is derived from the server's clock – and there is no way to distinguish between the two ways the expiry could have been set just from looking at the cookie. Servers can send both an `expires` and a `max-age` attribute in a `set-cookie` header, in which case the latter takes precedence. Further complicating the matter is that not all web browsers treat these attributes the same way. Until recently, Internet Explorer did not honor the `max-age` attribute at all. [7]

javascript

Cookies can be read and set by javascript scripts. With the Mozilla Firefox browser, any time one wishes to set or modify a cookie with javascript expiration information needs to be added or the cookie will not get stored.[5] As Javascript executes on the client machine any expiration information will most probably not originate on the server.[6] The `httponly` cookie attribute protects cookies from being read or updated by javascript scripts. As such, a cookie with this attribute

[5] Tested with Mozilla Firefox 18.0.

[6] Technically it could, though a programmer would have to go out of his way to make this work.

is guaranteed to not have undergone modification by javascript. Any other cookie could have had its expiration information set using the local clock.

static expiries

Nothing dictates that a web server should calculate expiry by adding some constant to its current time. As we will see, it just happens to be the common case, but there are many exceptions.

2.3.3 Time and network-related problems

server time

When we are depending on the server's local clock we either need to determine its skew with regard to universal time (UTC), or we will just have to assume it is running in sync with UTC. How safe an assumption is that? In 2006-2007, Bucholz and Tjaden have conducted a large scale measurement of popular web server's skews that sheds some light on this issue:

> Approximately 74% (6040 out of 8149) of all hosts that responded were within $\pm$10 s of our reference time. The other 26% (2109 out of 8149) that responded were more than 10 s out of synchronization – some by seconds, some by minutes, some by hours, some by days, some by weeks, some by months, and some by years. About two thirds (5510 of 8149) of the hosts that responded were tightly synchronized to within $\pm$2 s of our reference time, and a little more than half (4213 of 8149) showed a time difference of 0 s as measured by web-time. We were a bit surprised to see only 50% of the hosts with no time difference from our machine. Before run- ning the experiment we had expected that this number would be much higher as these are popular Internet web servers and we had expected the vast majority to be tightly synchronized. [3]

If one is to rely on server time, it is definately desirable to pre-emptively measure server skew.

network lag

The instant the server sets the expiry is not the same instant the client processes the cookie and sets the creation time. The latter will naturally take place later. This means that even if the server's and the client's clock are running in perfect sync a positive value for S will be derived, since $\Delta_s < \Delta_c$. This apparent skew may amount to seconds if there was serious network congestion at the time of cookie transfer. Congested links are no exception at the consumer end of the spectrum, especially when it concerns mobile environments.

timestamping resolution

When time is expressed in second granularity, as is the case with the data at hand, a difference of mere milliseconds may easily amount to a recorded measurement of one second. Consider the following scenario. Again, a server and a client are running in perfect sync with regard to their clocks. The server dispatches a cookie at $T = 1.999s$. When setting the expiry information, it uses the current time – but measured in seconds, which results in $T = 1s$.[7] Network lag is minimal so the client receives the cookie at $T = 2.001s$. When setting the cookie creation timestamp, it will use $T = 2s$. So even though the transfer time was two milliseconds, this phenomenon will make it seem like the client was skewed by one second.

[7] Whether rounding or flooring is used to downsample doesn't matter: one way or the other, there is a boundary – either at .0 or at .5 .

3 Creating a database of server deltas

To perform our skew calculations we will need to relate a cookie C to the server delta Δ_s which was used in its creation. To relate C to server headers, we can use its identity – $\{name, path, domain\}$ – all of which can be derived from `Set-Cookie` headers and the request URI that led to the reply that the header is part of.

To calculate Δ_s, we will need to know what the server clock's time was at the moment the `Set-Cookie` header was generated. Fortunately RFC2616 dictates that a server must represent the date and time at which the reply was generated in the `Date` header. [4]

Therefore, it appears that all we need to create a database of server deltas is a corpus of HTTP replies.

3.1 Shodan: A web server reply corpus

The "HTTP Header Survey" published by Shodan Research is just such a corpus. [6] It contains the reply headers that the Alexa[8] top ten thousand websites generated in response to requests made with 14 different user-agent request headers. The request URI is not recorded in full, but the fully qualified domain name is, and the responses imply that a request is made to the root (/) of each web server.

Below is an example of an entry in the dataset:

```
36,apple.com,Mozilla/2.0 (compatible; Ask Jeeves),"HTTP/1.0 302 Object Moved
Location: http://www.apple.com/
Content-Type: text/html
Cache-Control: private
Connection: close

"
```

The dataset is in CSV format. Since HTTP replies include linebreaks, there are linebreaks within the records. Each record contains four fields: site-id (the rank in the top-10,000), fully qualified domain name, user agent and HTTP reply, respectively. The particular reply listed above is meant to instruct the user-agent to redirect to a different URI.

For this research project, we use the September 22nd, 2012 issue of the dataset.[9]

3.2 Representativeness and shortcomings

- ▷ The dataset does not include the time of request dispatch, nor the time the reply was received. This makes it infeasible to check whether the web server was in sync with UTC or to control for forementioned network lag effects. Not being able to correct for server skew will likely introduce errors in some data derived from the dataset.

[8] Alexa (`http://www.alexa.com`) are a web analytics company. They publish a freely downloadable report of what they deem to be the top one million websites.

[9] Shodan Research has given permission to redistribute this issue of the dataset as part of the `cookieskewer` software distribution.

- ▷ The top-10,000 web sites are just the sites that people steer their browsers towards. Those web sites might well host large amounts of information on content distribution networks, or they might embed content from advertising networks – all of which could employ cookies for some reason. If a user were to visit each of the 10,000 web sites that user will likely end up with many cookies that cannot be related to the dataset.

- ▷ It is not a longitudinal survey. A longitudinal data set would permit determining how often a server delta changes, deriving a measure of flux that would serve to indicate how trustworthy a certain Δ_s is. For now, we will just have to assume that if some CID leads to a certain Δ_s, that this particular delta has always been used.
  However, due to the fact that 14 requests were made to each server – one request for each user agent – we still have multiple samples, which will help us to find out whether a server is actually using a constant Δ_s to calculate expiry.

- ▷ The top-10,000 web sites may be particular in their architecture because they have to cope with large amounts of visitors. Statistics derived from the dataset might be skewed towards high performance architectures. For instance, setting a constant expiry or using max-age is arguably less resource intensive than polling the system clock. When dealing with millions of requests this can make a difference.

3.3 Sanitizing input data

The input data originates from an unregulated domain. Unregulated, for RFCs are not enforced. It should come as no suprise then that there is some variance in adherence to RFCs, and this poses challenges.

Point in case: Opinionsphere.com sets a cookie `Expires` attribute with the following date: `Fri, 21-Sept-2012 12:56:19 GMT`. The question is not if one can make sense of this date. A human can, but that by itself does not warrant inclusion of the CID in the database – for by doing so, one would be implicitly assuming that *any browser can and will interpret this information*. But as it happens, this date does not conform to RFC specifications.[10]

In this project a pragmatic approach to admission issues is taken. If data does not fit the formats defined in the RFCs, it is simply discarded. The net effect of this measure is that some samples from the Shodan dataset will not be represented in the database. Since the utility of the database does not depend specifically on modeling each and every one of the 10,000 web sites of the Shodan survey, this choice can be defended.

However, when parsing input data caution has been taken by quantifying the effects of each decision. The assumption is that if a significant part of the dataset shows non-RFC properties, user agents will most probably accept those properties.

[10] The locus of the problem is '`Sept`'. Most probably the month of September was intended, which should be abbreviated as '`Sep`'.

One site may be misconfigured, but if many sites deviate from the standards in a specific way, then that behaviour is quite likely accepted at the other end. Being too strict would discard significant parts of the dataset for no apparent benefit: we have already noted that the RFCs are not a proper model of web browser behaviour.

3.4 Applications of the database

We process the headers from the Shodan dataset into a semi-normalized relational form.[11] What facts can be learned by querying the database? And which derived data sets will prove useful? What follows is an overview of derived datasets and statistics that will be used when processing a browser cookie store.

MCL

"**M**ax-age **C**ookie **L**ist" – From the server headers, it is straightforward to gather CIDs of cookies which could derive their expiration through the max-age mechanism. Any cookie that has an identity which is in this list is unfit to be used for estimating skew. There are 95 such identities on this list.

BDL

"**B**ad **D**ate **L**ist" – A list of expiries that appear to be constants. Expiries can be static: the same date will be set in every cookie. Often this is a date far into the future, such as, for instance, `2038-01-19 03:14:07`[12] or `9999-12-31 23:59:59`. Dates in the past (such as `1970-01-01 00:00:00`) occur as well – setting an expiration date in the past is the only way for a server to get a client to delete a cookie.[13]

By comparing multiple samples from the same server, constants can be detected.[14]

VCL

"**V**ariable expire-delta **C**ookie **L**ist" – Some servers act erraticaly in that delta nor expiry are constant. The resulting Δ_s cannot be used for estimating skew. Variance can occur for a couple of reasons.

If the variance of Δ_s across multiple samples is small, then that could be due to requests straddling the second boundary: as noted before, the instant the `Date` header is set is not necessarily the instance that the `Expires`-attribute of a cookie is determined. This is a natural consequence of the 1-second resolution of the timestamps.

Larger variance of Δ_s can arise if the `Date` header is created on a different

[11] In the `cookieskewer` software distribution, this is accomplished by executing 'make zodan.db'.

[12] This is exactly 2^{31}seconds after January 1st, 1970 and the largest time representable by a 32 bit (signed) integer in the structure returned by the gettimeofday() system call (POSIX.1-2001).

[13] The natural consequence of this is that these past dates will not be found in browser cookie expiration metadata.

[14] In the `cookieskewer` software distribution, these can be generated through `make BDL.txt`. It should be noted that the method used in `make BDL.txt` – naively – does not take into account the sample count of the CIDs. Therefore, the expiration for a CID with one single sample will end up on the BDL while it may not be a constant expiry. A manual selection of bad dates is included in the distribution as `BDL.curated`.

server than the `Set-Cookie` header. One way to spread server load is to host a pool of dynamic content servers (handling cookies) behind one or more reverse proxies (that handle caching and set the `Date` header). If these servers are not running with their clocks in sync, or if one timestamp is cached while others aren't, this gives rise to variance.[15]
Lastly, servers might use a constant Δ_s for calculation of the expiration attribute but cache the result, only updating it periodically.[16]
Complicating the matter is that the samples are taken using different user agent headers in the request. If variance is observed, this may not actually mean the cookie is useless – servers might deliberately send different expiry information to a mobile browser user agent, for instance.
These are the criteria by which the CIDs that make up the VCL are selected:

1. 10 or more samples need exist for this CID.
   (This discards 12% of the identities)
2. The largest Δ_s is larger than the smallest by three or more seconds.

These criteria result in a VCL that contains 368 of the 1819 CIDs that have their expiration set through an 'Expires'-attribute.

DSM

"**D**elta **S**erver **M**ap" – A mapping of CIDs to their canonical Δ_s. Given the variance described earlier, how do we define inclusion?

1. 10 or more samples need exist for this CID.
2. The largest Δ_s is no more than two seconds larger than the smallest.
3. At this point, there can be at most three distinct Δ_s in the samples for this CID. There should be one dominant value; one which is more common than the two others. This dominant value will be the canonical Δ_s for this CID.

These criteria result in a DSM that contains 1192 identities. Note that there are 259 CIDs that are neither in the VCL nor in the DSM. They are not designed to cover all the identities that can be found in the dataset; the dataset does not allow to classify all server behaviour as either usable or unusable for calculating skews.

Distribution of Δ_s

The values in the DSM follow a non-random distribution. See figure 3.1 on page 27. Figure 3.2 on the following page shows the distribution is highly uneven; close to 50% of the Δ_s are covered by just five values: 1 year, 1

[15] An example of variance that could have such an explanation can be found in the headers for *chatvibes.com* .
[16] *www.net.cn* seems to be doing this.

month, 1 day, 1 week or 10 years. The 50 most-occurring values account for 90% of the distribution.

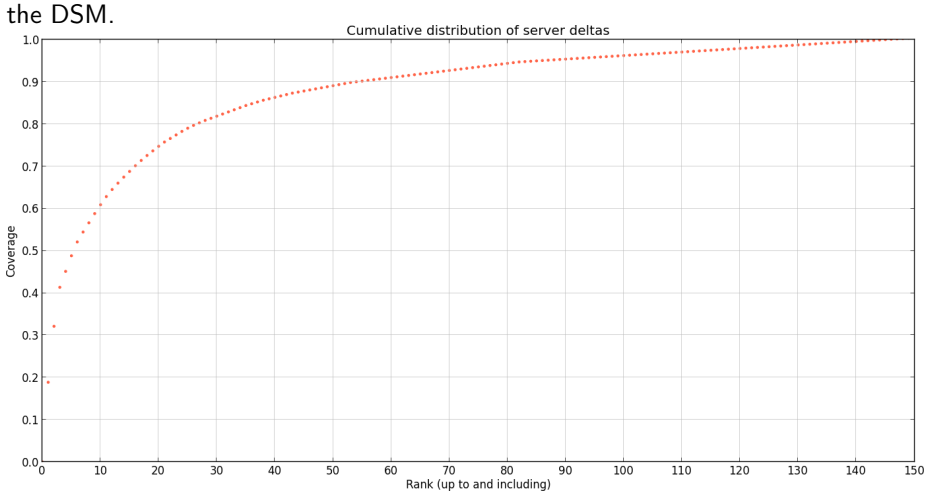
Incidence of the httponly attribute

The httponly attribute is present for close to 12% of the cookie identities.[17] Would it have been much more – half of them, perhaps – then it would become a viable approach to simply disqualify for analysis any cookie that does not have this attribute, taking away one of the uncertainties noted in section 2.3.2 on page 9.

Ratio of usable cookies

For an unrecognized cookie that has an expiry that is not on the BDL, what is the propability that its expiration timestamp is set through the mechanism we depend on? The information on the sizes of the above derived datasets allow us calculate this; it is the amount of useable cookies expressed as a fraction of the total: $\frac{1192}{95+368+1192} \approx 0.7$. The `max-age` attribute is not prevalent – it is fortunate that this RFC-recommended attribute does not seem to have gained much traction (yet).

Figure 3.2: This shows the cumulative distribution for Δ_s. Taken together, less than 10 values account for the majority of Δ_s found in the samples that make up the DSM.



[17] This is reproducable by running `make httponlyfraction.magic` from the `cookieskewer` software.

4 The skew estimation algorithm

4.1 Description of inputs

The algorithm operates on a browser cookie store. For each cookie, it needs the following attributes:

- ▷ **name, path, domain**
  Used to compose a CID.
- ▷ **creation timestamp and expiration timestamp**
  Used to calculate Δ_c.
- ▷ **httponly**
  Used for determining cookie trustworthiness.

From the database, the following derivatives are used:

- ▷ **MCL**
  Any cookie which is known to have derived its expiration from a max-age attribute, should be discarded.
- ▷ **BDL**
  Any cookie of which its expiration timestamp is on the list of bad dates, should be discarded.
- ▷ **VCL**
  Any cookie which is known to originate from a server that behaves erratically with respect to Δ_s, should be discarded.
- ▷ **DSM**
  Used to match retrieve a reliable Δ_s associated with a CID.
- ▷ **ratio of usable cookies**
  Used for determining cookie trustworthiness.
- ▷ **distribution of Δ_s**
  Used for forming (ranked) hypotheses for cookies that are not in the DSM.

4.2 Dealing with imprecision

Variance of Δ_s and Δ_c introduced by jitter (through network latency and sampling rate artefacts) poses a challenge. For instance, when analyzing a web browser cookie store, all cookies from the store could be associated with a Δ_s (in the terminology of section 3.4 on page 14, all CIDs would be in the DSM). Supposing the system has never been skewed, an analysis would have to show zero skew.

Variance will disturb this outcome. Even if variance for both Δ_s and Δ_c is only ±1, resulting skew estimates will be in the domain $[-2, 2]$ – distortion is cumulative. A method to curb this effect is needed. The mechanism would collapse the domain to a single value, so that, in the example given, all cookies would predict a zero

second skew. Such a method would trade precision for stronger support of found skew hypotheses. This becomes especially important when dealing with multiple skews.

The algorithm that provides this mechanism is implemented such that it collapses counts within a folding range in order of dominance. That is to say, given the following series of values:

$[98, 99, 100, 100, 101, 102, 102, 102, 103]$

and their respective counts;

$\{[98:1], [99:1], [100:2], [101:1], [102:3], [103:1]\}$

with a folding range of $[-1, 1]$, a new and collapsed counting will result:

$\{[98:1], [100:3], [102:4]\}$.

Note that the 'contested' value of 101, which fell within the folding range of both 100 and 102 has been taken up by the latter, which was the most dominant initial value. This algorithm will be applied during different stages in the analysis: first, when creating an a priori probability distribution of Δ_s, second, when forming skew hypotheses. In the remainder of this report, whenever the term 'folded' or 'folding' comes up, this algorithm is what is referred to.

4.3 Forming hypotheses – A voting system

For each cookie, a skew S : $S = \Delta_s$-Δ_c will be calculated. For cookie identities that are in the DSM, Δ_s can be determined by a lookup and these cookies will each predict a single skew S, or rather, a single vote for some skew.

However, the dataset might not cover all CIDs found in browser cookies. To solve this, a cookie can predict multiple skews – each of the Δ_s found in the distribution of Δ_s will be used to calculate a skew, forming a *skew set*. In this way, a cookie casts votes for many skews, and if this is done for all cookies then the amout of skew estimates will equal the cartesian product $\Delta_s \times \Delta_c$. When viewed in isolation, this seems useless – when we *know* that only one Δ_s was involved in the creation of the cookie, and certainly not all of them, why apply all of them?

It is when tallying the votes and thereby combining all skew sets of all cookies that an interesting effect makes its appearance: convergence on the true skew. Of the many skew estimates of one particular cookie, only very few will overlap with those of one particular other cookie. Even fewer will be shared between three cookies, andsoforth. The support for the true skew will be large (its upper bound is the size of the cookie collection) whereas there will be little shared support for stray estimates.

The gravity of the effect depends on the size of the browser cookie collection – obviously, the effect is not present when a collection consists of only a single cookie. Second, it depends on the fidelity of the browser cookies; ideally their expiration timestamps would reflect the state of the server's clock. Third, it depends on the representativeness of the distribution of Δ_s used.

4.4 Ranking hypotheses – Weighted voting

Hypotheses are not created equal. There is uncertainty associated with many of the cookies that support a hypothesis. Besides uncertainty with respect to a cookie's fidelity, there is uncertainty for many of the skews in a cookies' skew set – at most one of the Δ_s used to create the skew set was the true Δ_s.

It is possible to calculate the 'trust' for each hypothesis using each individual cookie's reliability estimate and the prior probabilities of Δ_s applied in creating its skew set. Hypotheses supported solely by cookies that are recognized from the DSD will have relatively higher trust than hypotheses that are not. Cookies that have an `httponly` attibute set will increase aggregate trust for a hypothesis more than cookies that have not.
A cookie's trustworthiness is quantified as follows:

1. Let A be the probability that the cookie's expiration has been affected through javascript interactions.
   If the cookie has the `httponly` attribute set, assign $A = 1$.
   If not, assign $A = (1 - j)$, with j being a runtime parameter that represents an investigators estimation of the general a priori probability that a cookie's expiration has been affected through javascript interactions.[18]

2. Let B be the probability that the cookie is usable with respect to behaviour which takes place on the server. In other words, the probability that the cookie's expiration was set through the mechanism explained in section 2 on page 6.
   If the cookie's CID is in the DSM, assign $B = 1$.
   If not, assign to B the general ratio of usable cookies described in section 3.4 on page 14 – this can be determined at runtime.

3. Assuming A and B are independent, a cookies' trustworthiness CT is their product: $CT = A \times B$.

For each skew in a cookie's skew set, the probability of it being the correct skew can be quantified as follows:

1. Let X be the probability that the skew is the result of applying the correct Δ_s. If the cookie's CID can be found in the DSM, assign $X = 1$.
   If not, assign to X the relative frequency of the particular Δ_s, which can be derived from the distribution of Δ_s**.**

2. Assuming CT and X are independent, the trustworthiness of a specific skew ST from a cookie's skew set is their product: $ST = CT \times X$.

The aggregate trust for a skew estimate can be derived from summing up the ST, for that skew, of each of the cookies that support the skew estimate. The upper bound of that number is the size of the cookie collection — the product of probabilities A, B and X is invariantly ≤ 1.

[18] Empirical data on javascript cookie interactions is sorely missing.

5 Implementation

5.1 The program interface

Listing 2 on the next page shows the command line interface of the `skewy.py` program. As inputs it needs at the very least the database derived from the Shodan data set, and a browser cookie store.

At the moment there is one single browser cookie store adapter, handling cookies stored by the Mozilla Firefox (versions 3.0 and up) browser.

Some of the options are non-self-explanatory:

fold
: (line 20) This option governs the behaviour of the algorithm that adjusts for variance. Using 1 or 2 as a parameter is a good choice if the objective is to correct for the non-meaninful variance induced by the sampling resolution of 1 second. As the folding is applied twice, the resulting imprecision in skew estimates is twice this number. In other words, when using `--fold 2` a skew estimate of 10 actually means 10 ± 4.

min_ds
: (line 23) This option modifies the Δ_s distribution, restricting it to larger values. This is meant to adjust for the fact that short-lived cookies are short-lived: cookies that have an expiration one minute into the future are likely to be garbage collected by the browser after a short while.[19] Therefore, the probability of finding short-lived cookies is low. This option allows to investigate this effect.

topsd
: (line 23) This option also modifies the Δ_s distribution, restricting it to more common values. As can be gathered from the cumulative distribution of Δ_s (figure 3.2 on page 16), extending the distribution to encompass uncommon values offers diminishing returns – but at the same time, it will likely increase noise. This option allows to investigate this effect.

interactive
: (line 33) Activation of this option presents an interactive Python shell that exposes all the internals of the program, making it easy for researchers to investigate the properties of the data and the algorithms.

[19] This is completely dependent on the particulars of the cookie storage mechanism of specific browsers.

Listing 2: Command line parameters for `skewy.py` from the `cookieskewer` software distribution.

```
% ./skewy.py --help

usage: skewy.py [-h] -z ZODAN -c COOKIEDB [-b BADDATES] [-f NUMBER]
                [--min_ds NUMBER{s,m,h,d,w,y}] [-v] [-d {csv,ascii}]
                [-l NUMBER] [-i] [-t NUMBER] [-j FRACTION]

Estimate time skew for collection of web browser cookies. Outputs results in a
table.

optional arguments:
  -h, --help            show this help message and exit
  -z ZODAN, --zodan ZODAN
                        Path to Zodan database
  -c COOKIEDB, --cookiedb COOKIEDB
                        Path to Firefox cookies.sqlite database. If path ends
                        with '.pickle', it will be parsed as a previously
                        pickled cookie collection.
  -b BADDATES, --baddates BADDATES
                        Path to Bad Date List
  -f NUMBER, --fold NUMBER
                        Fold values into the nearest and most numerous one if
                        distance is less than NUMBER.
  --min_ds NUMBER{s,m,h,d,w,y}
                        Don't consider Δs smaller than NUMBER units. Units are
                        seconds, minutes, hours, weeks or years.
  -v, --verbose         Be chatty (on stderr)
  -d {csv,ascii}, --dump {csv,ascii}
                        Dump table of found skews to stdout in CSV or pretty-
                        printed ASCII.
  -l NUMBER, --limit NUMBER
                        Limit table output to top-NUMBER skew estimates,
                        ordered by trust.
  -i, --interactive     Drop to interactive shell
  -t NUMBER, --topsd NUMBER
                        Only use top NUMBER Δs
  -j FRACTION, --jsmod FRACTION
                        Use FRACTION as a general (written-by-JS/written-by-
                        js+clean cookies) ratio assumption
```

5.2 An example run

5.2.1 A web browser cookie store

For an example run of the program we will need a collection of web browser cookies. Such a collection is included in the `cookieskewer` software distribution as `xorry.pickle`. It contains cookies gathered accumulated over the course of two years using the Mozilla Firefox browser on a Linux workstation (named xorry) that has had NTP synchronisation configured. Ideally, one would want to know the exact skew record of this machine, but such is unavailable. Given its configuration, in the experiments that follow, its skew is assumed to be zero. From hereon, the corpus

will be referred to as 'xorry'.

In light of the problem with cookie creation time identified in section 2.3.1 on page 9, the choice of using a corpus gathered by the Firefox web browser is a surprising one as this browser does not record modification timestamps when updating cookies. The choice was made for pragmatic reasons; the SQLite format used by Firefox to store cookies is well suited to exploration. Tests showed that even under this adverse condition – having to rely on less trustworthy Δ_s calculated from creation timestamps – the algorithm was able to correctly derive skews.

The xorry corpus has been anonymized to the extent that it has not impacted the algorithm; only the CIDs that were in the DSM, MCL or VCL have been preserved (but the value of the name-value pair has been anonymized).

5.2.2 Program output

Let us go over the lines of output of the program as displayed in listing 3.

Listing 3: An example run of `skewy.py` on the data supplied with the `cookieskewer` distribution. The program output shows a skew hypothesis of zero seconds ranking highest.

```
% ./skewy.py --zodan zodan.db --cookiedb xorry.pickle \
  --baddates BDL.curated --fold 2 --jsmod 0.15 --verbose \
  --dump ascii --limit 10

Client cookies:
2786 initially
2761 after subtracting bad dates
2759 after subtracting known maxage cookies
2756 after subtracting known variable Δs cookies
0 have atime < ctime (red flag)
21 have known Δs
0.72 trust for unknown cookies
+----+-------------+-------+---------+---------+---------------+--------------+
| ID |        Skew | Trust | Support | # Known | Earliest date | Latest date  |
+----+-------------+-------+---------+---------+---------------+--------------+
|  0 |           0 |    71 |    1337 |       9 |    2010-12-07 |   2012-12-19 |
|  1 |   -28943999 |    32 |     425 |       0 |    2010-12-17 |   2012-12-19 |
|  2 |   -31449599 |    22 |     576 |       0 |    2010-12-17 |   2012-12-19 |
|  3 |  -126144000 |    20 |     190 |       0 |    2010-12-15 |   2012-09-03 |
|  4 |   -31535999 |    14 |     737 |       0 |    2010-12-15 |   2012-12-19 |
|  5 |  -155088000 |    14 |     182 |       0 |    2010-12-15 |   2012-09-03 |
|  6 |  -283824000 |    13 |     127 |       0 |    2010-12-22 |   2012-12-19 |
|  7 |    28944001 |    10 |     468 |       0 |    2010-12-17 |   2012-12-19 |
|  8 |  -157593600 |    10 |     303 |       0 |    2010-12-15 |   2012-12-19 |
|  9 |   -30931199 |     9 |     448 |       0 |    2010-12-17 |   2012-12-19 |
+----+-------------+-------+---------+---------+---------------+--------------+
```

6-9 The corpus contains 2786 cookies. Of these, only 30 could be filtered out based on the MCL, VCL and BDL.

10 Firefox keeps a record of when cookies were last accessed – presumably for garbage collection purposes. A record of a cookie that was accessed before

it was created creates a logical inconsistency which can be an indication that the system clock has been rolled back at some time, and this should not go unmentioned.

11 The size of the set-intersection between the 2756 cookie identities in xorry and the 1192 CIDs in the DSM is only 21.

12 The 2730 remaining cookies of which the identities are not in the DSM get assigned this number for probability B (as described in section 4.4 on page 19).

14 The table header refers to an ID (convenient when discussing output), a skew hypothesis (in seconds), the aggregate trust of the hypothesis, the size of the group of cookies that support a hypothesis, the portion of that group that is on the DSM, and the earliest and latest creation date of the cookies within that group. The latter two are not corrected for the associated skew hypothesis.

16 The program correctly estimates the skew at 0 seconds. It is crucial to take notice of the fact that the program has no built-in tendency to assume a skew of 0 seconds. The algorithm does not work by trying to disprove the null hypothesis that the clock was not skewed and the resulting skew estimate of zero seconds is therefore a genuine feature of the input data, and not some preconception.
Interestingly, only 9 of the 21 cookies that are on the DSM are part of the group supporting this skew. This will be discussed in section 6.2.2.

17-25 These lines show the details of 9 other high-ranking hypotheses. They have much less trust than hypothesis #0. None of them are likely to have been correct at any moment during the lifetime of the cookie collection. This is noise, but the values for the hypotheses are peculiar; they will be discussed in section 6.2.1.

6 Discussion: Algorithm performance

While discussing algorithm performance, it is important to keep in mind that the results of the performance tests are dependent on its inputs. Since only one web client cookie corpus is used (which is crippled due to its lack of modification time information) quantitative results, while indicative, hold no absolute value. It is however quite possible to investigate the influence of various parameters.

6.1 Basic performance evaluation

The result of the example run (section 5.2 on page 21) invites to explore the limits of the algorithm. First and foremost, what is the minimum size of a cookie collection for it to allow its skew to be estimated? Second, how does it handle multiple skews?

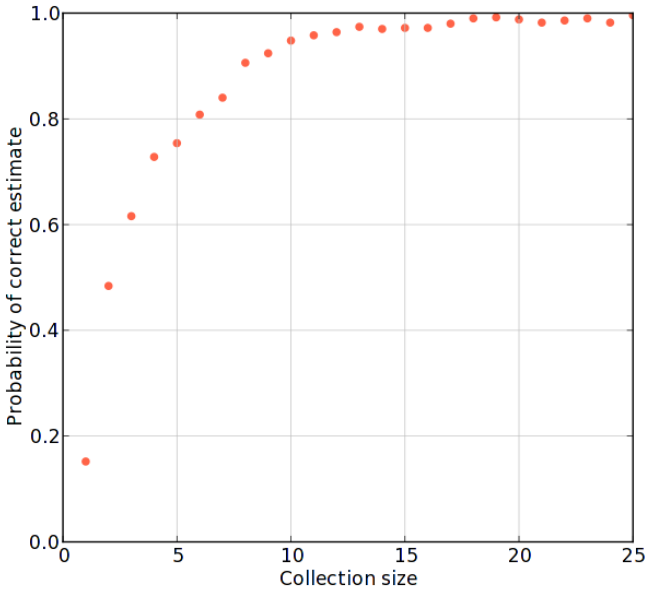
Because the skew integrity of the xorry corpus is unknown (a skew of zero seconds is assumed) test results should be interpreted relative to the baseline performance from the example run.

6.1.1 Minimum collection size

To measure only the performance of the voting algorithm, the DSD is cleared. Random samples of the cookies from the xorry corpus are taken, with sample sizes varying from 1 to 25 cookies. The measure of performance is whether the correct skew estimate (of 0 ± 4 seconds; since `--fold 2` was specified) comes up as the most trusted one.

For each sample size, 500 test runs are done and the fraction of runs for which the correct skew came up as most trusted is recorded. The results are shown in figure 6.1 on the next page. The estimates become reliable for collection sizes upwards of **10** cookies.

Figure 6.1: Correctness of estimate as a function of cookie collection size (folded by 4s). About 10 cookies are needed for a reasonably reliable estimate.



6.1.2 Detecting multiple skews

It is likely that skew will vary over the course of the lifetime of a cookie collection. Possibly, the system clock was skewed for only a short amount of time. This begs the question: how many cookies are needed to distinguish such a secondary skew hypothesis from noise? To answer that question a skew is introduced to some fraction of the cookies of the xorry corpus. The larger the fraction, the more likely it is that the skew will be detected as a secondary skew. The measure of performance is the minimum size of the fraction necessary to let the corresponding estimate rise above the noise and reach the second rank. For the xorry corpus (with cleared DSM and `--fold 2`) this fraction proves to be about **0.35**.[20]

6.2 Exploring complex properties

6.2.1 The origin of bogus skews

In the example run output (listing 3 on page 22, lines 17-25) we have observed peculiar values for the nine lower-ranking skew estimates. Specifically, they are

[20] In `skewy.py` interactive mode, this can be established by introducing an artificial skew of, for instance, 4242 seconds:
from testing import *; data.DSM.clear(); quick_est(skewgen(data.cookies, 4242, 0.35))

peculiar when expressed not in seconds, but in days, weeks and years (365 days). These equivalent values are listed in table 6.1.

Table 6.1: Bogus skew estimates expressed in days, weeks and years, and broken down in their likely components.

skew (in seconds)	equivalent value	Δ_c component	Δ_s component
-28944000	-(1y - 30d)	1y	30d
-31449600	-(1y - 1d)	1y	1d
-126144000	-4y		
-31536000	-1y		
-155088000	-(4y - 30d)	4y	30d
-283824000	-(5y - 30d)	5y	30d
28944000	(1y - 30d)	30d	1y
-157593600	-(5y - 1d)	5y	1d
-30931200	-(1y - 1w)	1y	1w

Since we know these skews are derived as $S = \Delta_s$-Δ_c, we can make a guess towards the values of the components. We know the distribution that Δ_s comes from; it has been discussed at length. We also know that the cumulative distribution for Δ_s (figure 3.2 on page 16) is markedly lopsided. And since prior probabilities of Δ_s are a cornerstone of the skew ranking algorithm, it deserves to be re-mentioned that the four most common values are **1 year**, **1 month**, **1 day** and **1 week**, in that order. These are therefore also likely values for the Δ_s component in table 6.1.

We have not yet investigated the distribution of Δ_c that is being operated on. This distribution is shown in figure 6.2 on the following page. Above this figure, on the same page, figure 3.1 is showing the Δ_s distribution. The histograms are normalized, allowing a direct and revealing comparison between the two distributions.

It is immediately obvious that they are dissimilar – not so much in *where* the peaks occur as in the *size* of the peaks. The center of gravity for the client distribution appears to be towards the right of the 1 year anchor point whereas most of the mass for the server delta distribution is on the left-hand side of it. It has already been mentioned in passing that short-lived cookies are short-lived; they do not accumulate as much as cookies with a longer shelf life. The latter are much more abundant in the Δ_c distribution. The four most common values are **1 year**, **5 years**, **2 years** and **10 years**, in that order.

These values coincide with the values for the Δ_c component shown in table 6.1. Armed with an understanding of a major characteristic in the difference between the two distributions, it now becomes clear how the bogus skew estimates might be formed and how they would derive their peculiar values – they are an expression of a mismatch between the distributions. The mismatch leads to the same mistake (of applying an inappropriate Δ_s) being made repeatedly with the same numbers as inputs, simply because certain Δ_c and Δ_s are so abundant.

Figure 3.1: Histogram showing the distribution of Δ_s for samples in the DSM, in the range 10 seconds – 30 years. Size of bins is 120 seconds.

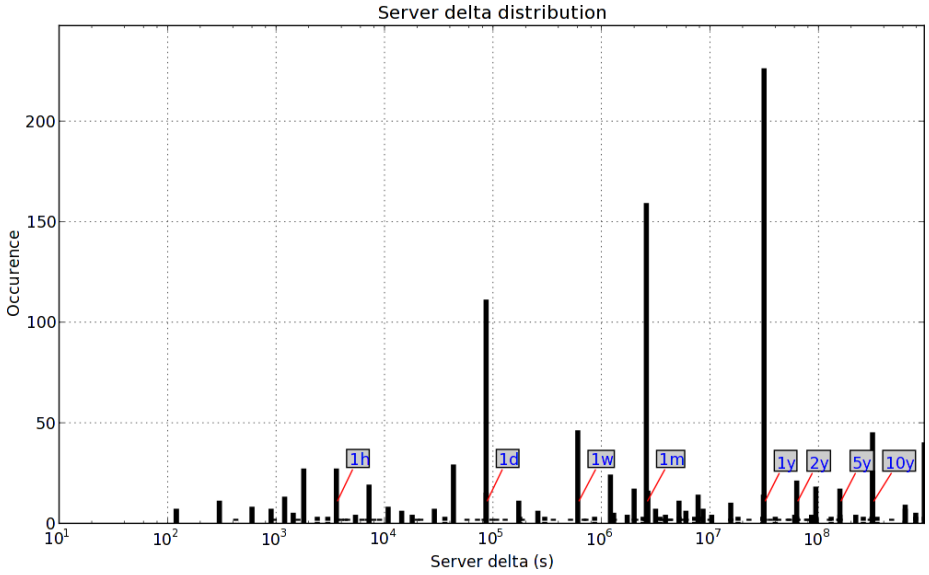
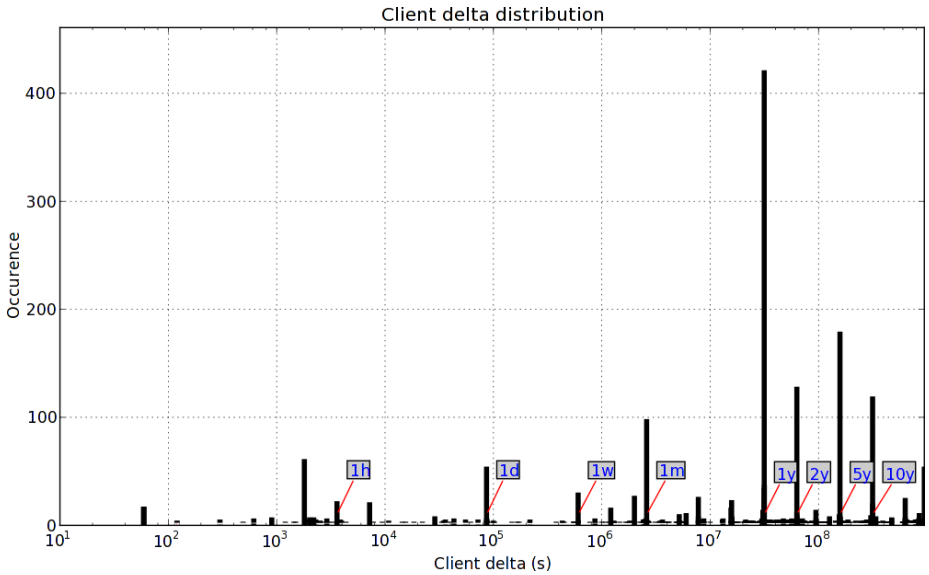


Figure 6.2: Histogram showing the distribution of Δ_c for the xorry web client cookie corpus, in the range 10 seconds – 30 years. Size of bins is 120 seconds.



6.2.2 The consequences of employing an improper Δ_s distribution

Relation of the Δ_s distribution to web browser cookie stores

The skew estimation algorithm relies on a representative distribution of Δ_s. As we have seen in section 6.2.1, the shape of the 'right' distribution depends on the lifetime of the browser cookie store it is applied to – older stores will have accumulated more cookies with far future expirations. But even browser cookie stores of the same age will feature different distributions since their shapes also depend on when the browser was used and on when the garbage collector was last run.

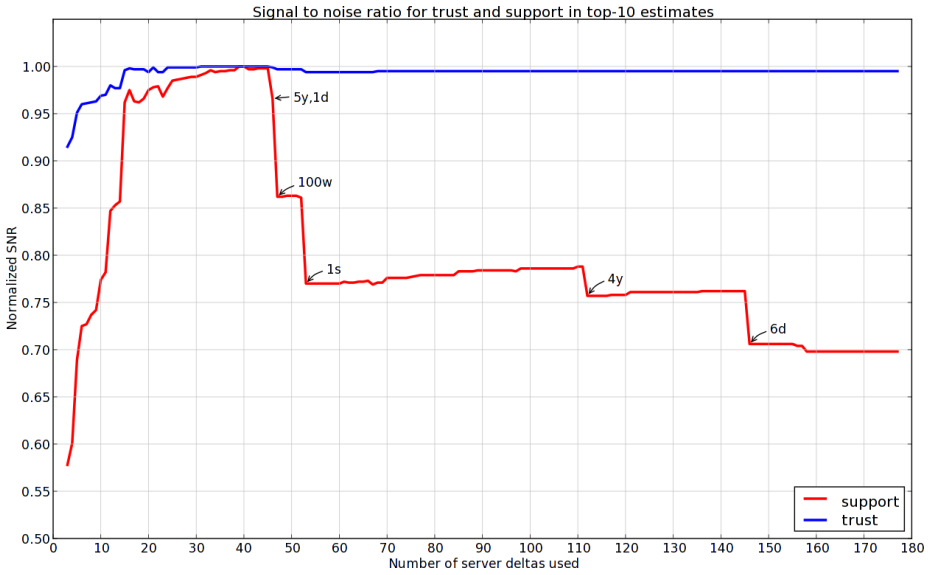
Superficially it seems that the Δ_s should be individually tailored to browser cookie stores under investigation. Doing so will prove to be a challenge, for skew hypotheses rely on the *temporal difference* between corresponding peaks (the horizontal dimension of the histograms in figures 3.1 and 6.2), while the accuracy of the hypothesis ranking relies on the *similarity in relative frequencies* between corresponding peaks (the vertical dimension of the histograms). Unfortunately, finding pairs of corresponding peaks is only possible after correcting for skew: a chicken-and-egg problem.

Completeness of the distribution

Is completeness of the Δ_s distribution a factor of significance in the performance of the algorithm? One way to measure the effect of the distribution size is to gauge the signal to noise ratio (SNR) of the correct hypothesis while increasing the distribution size – starting off with the most prominent Δ_s and progressively adding less prominent Δ_s.[21] The signal to noise ratio is defined as the fraction of the support and trust for the correct skew hypothesis (which for the xorry corpus is assumed to amounts to 0 seconds) of the total support and trust for all hypotheses in the top-10 and normalized to the highest attained value. The experiment is run with the folding parameter set to 2. Figure 6.3 on the next page shows the result for the xorry corpus.

[21] With `skewy.py`, the distribution size can be varied using the `--topsd` parameter.

Figure 6.3: Development of hypothesis trust and support for the xorry client cookie corpus with increasing coverage of the Δ_s distribution distilled from the Shodan dataset. Some steep drops in support SNR are annotated with the associated Δ_s.



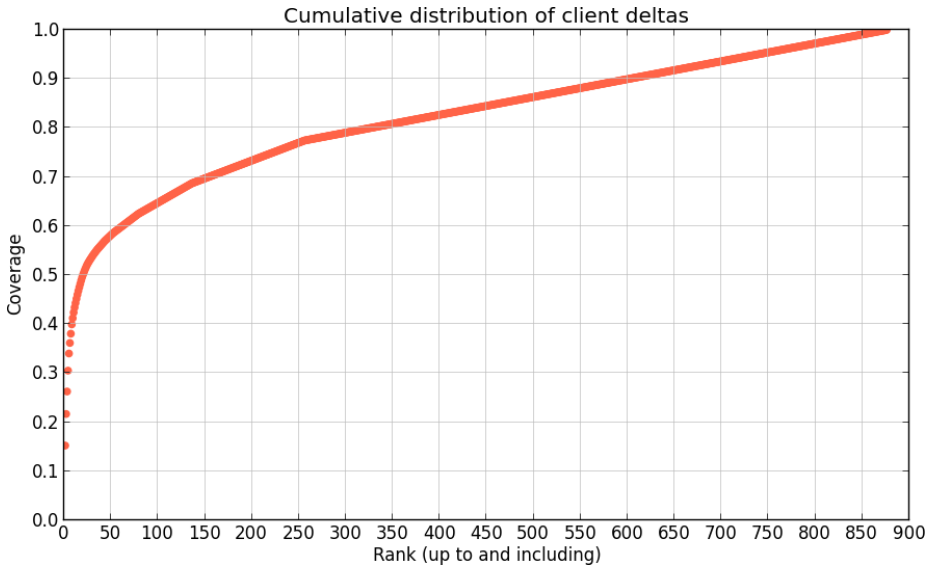
The figure leads to the following observations:

- ▷ The SNR plot experiences a series of sharp drops for hypothesis support SNR after the 46th most prominent value. The Δ_s values up to and including this 46th value cover about 85% of the total number of Δ_s observed, as previously illustrated by the lopsided picture painted in figure 3.2 on page 16.. Some of the 'long tail' Δ_s are inapplicable to the browser cookie store, as indicated by significant drops in support SNR. The one second Δ_s is likely the result of a measuring error as described in section 2.3.3 – a mistaken interpretation of a `Set-Cookie` header meant to expire a previously set cookie. And even if it were not, it is unlikely to find such a short-lived cookie before it is garbage collected. The drop associated with addition of the 4 year Δ_s value may well be an illustration of distribution mismatch: figure 3.1 on page 27 shows a peak at 4 years, whereas the Δ_c histogram (figure 6.2 below it) does not.
- ▷ In contrast to hypothesis support, hypothesis trust SNR is not significantly affected by addition of the annotated values. This is due to the diminishing prior probability of those values; they do not matter enough to make a dent. Moderating hypothesis support by taking prior probability into account shows its worth.
- ▷ It does not seem to pay to include the 'long tail' of the distribution. Maximum trust SNR occurs with about 30 of the most commonly occuring Δ_s, together making up slightly over 80% of the 'mass' (see figure 3.2 on page 16).

Impact of impairment due to absence of cookie modification timestamps

As noted, the Mozilla Firefox browser that produced the xorry corpus does not record modification timestamps. The `skewy.py` program derives Δ_c using creation timestamps instead, which makes them less trustworthy. How untrustworthy depends on how prevalent the act of updating cookies is. Servers can update a cookie stored by the browser in one of two ways: simply by sending an updated value, or by first expiring the old cookie and subsequently setting a new one. In the latter case there is no problem. If many cookies are updated through the former mechanism, this will have the effect of creating many unique values for Δ_c. Therefore, an indication of the gravity of the problem can be derived from a cumulative distribution plot of Δ_c in the xorry corpus, where it will show up as a long tail. Figure 6.4 shows this distribution.

Figure 6.4: The cumulative distribution for Δ_c from the xorry corpus (folded by 120s). All values after the 256th are unique, hence the 'knee' in the transition from values that occur twice to values that are unique at around the 250th value.



Contrasting this distribution with the one for Δ_s shown in figure 3.2, it becomes clear that although some values are much more common than others, the distribution is less lopsided than the one shown in figure 3.2. Whether updated cookies are at the cause of this remains unproven. One way to find out about this would be to contrast these findings with an analysis of a cookie store from a browser that does feature modification timestamps, such as Microsoft Internet Explorer.

Distortion imposed by untrustworthy Δ_c may partly explain the fact that only 9 out of the 21 cookies known from the DSM are found to be supporting the correct skew hypothesis (see listing 3). One of the other cookies is supporting a skew of -8, another is supporting a bogus skew of three years, and the ten remaining cookies

are alone in supporting skews that no other cookie supports. Support for the the three year skew might be a consequence of nonrepresentativeness of the Shodan corpus; the two encounters could be as much as two years apart and the server's configuration may have changed in the meantime. The skew of -8 is insignificant; it may be on the client side, but it might also be on the server side. The isolated skews supported by the other ten cookie identities may be the result of updated cookies.

6.3 Avenues for improvement

During research and development, some ideas for improvement of the algorithm performance had to remain unexplored. What follows is a compact enumeration of these ideas.

6.3.1 Using partial cookie identities

Listing 3 shows that there is a surprisingly small overlap between CIDs in the xorry corpus and CIDs in the Shodan corpus. 26 CIDs are shared. The world wide web is of course much larger than the 10,000 web sites probed for the Shodan corpus. Improving the server reply dataset by probing more sites and creating a longitudional survey will prove to be of value. But there is another improvement possible. Server behaviour is not solely dependent on hostname. It also depends on the software that is run on the web site. Installations of the same software may well have shared properties for cookie expiration.

To name an example: in the Shodan dataset, multiple sites can be observed to be running the phpBB bulletin board software. This software sets a cookie with name 'bb_lastvisit'. All 11 occurences of this cookie name have the same Δ_s of one year. Using probabilities (and a larger dataset), it will be possible to infer a Δ_s, or a probability distribution of Δ_s, for a cookie based solely on its name – a partial CID. On the client side, in the xorry corpus, (coincidentally) another 11 previously unrecognized cookies could then have been associated with this Δ_s, improving accuracy.

6.3.2 Impact of javascript and cookie updates

We have only made guesses towards what the impact that updates of cookies through javascript scripts have on Δ_c reliability (section 2.3.2). The same goes for the importance of having access to the cookie modification time (section 2.3.1). The first could be empirically determined by instrumenting the browser to log Javascript modifications. The second could be empirically determined by comparing creation timestamps and modification timestamps on cookies stored by a browser that keeps both kinds of timestamps, such as Microsoft Internet Explorer. These uncertainties could then be translated into probabilities and be automatically applied.

6.3.3 Finding local skews

Ultimately, the goal is to provide a record of historical skews. Several different skews may have occured over the lifetime of the browser cookie store and the skewy.py program output as shown in listing 3 is only a rough approximation of such a record. In section 6.1.1 we have seen that upwards of 10 cookies are needed to derive a skew based solely on probabilities. The understanding developed so far opens up the possibility of a sliding window algorithm that delivers a description of the development of clock skew over time.

7 Conclusion

This research project has contributed towards an increased understanding of the potential of using web cookies for retroactive skew estimation. The software prototype shows that by taking probability into account, and through the use of a database of background data, it becomes feasible to estimate skew despite the many uncertainties and pitfalls inherent in the mechanisms involved in the HTTP state mechanism. The method is young and requires review and experimental verification before it can be relied upon in forensic investigations.

References

[1] A. Barth. Rfc 6265-http state management mechanism. *Internet Engineering Task Force (IETF)*, pages 2070–1721, 2011. 2.2.1, 2.2.1, 2.3.1

[2] C. Boyd and P. Forster. Time and date issues in forensic computing—a case study. *Digital Investigation*, 1:18–23, 2004. 1.1

[3] F. Buchholz and B. Tjaden. A brief study of time. *Digital Investigation*, 4:31–42, 2007. 2.3.3

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc2616-hypertext transfer protocol – http/1.1. *Internet Engineering Task Force (IETF)*, 1999. 3

[5] R. Nolan, C. O'sullivan, J. Branson, and C. Waits. First responders guide to computer forensics. Technical report, DTIC Document, 2005. 1.1

[6] SHODAN Research. Http header survey. WWW: `http://www.shodanhq.com/research/infodisc/report`, 2012. Retrieved February 6th, 2013. 3.1

[7] M. Zalewski. Http cookies, or how not to design protocols. WWW: `http://lcamtuf.blogspot.com/2010/10/http-cookies-or-how-not-to-design.html`, 2010. Retrieved February 6th, 2013. 4, 2.3.2

[8] M. Zalewski. *The Tangled Web: A Guide to Securing Modern Web Applications*. No Starch Press, 2011. 2.2.1