Time Stamps: a neglected security issue

By Andrea Pasquinucci – ISSA Member, Italy Chapter

Establishing a timeline of ICT events requires that each event has a well-defined time stamp. Unfortunately time stamps are not always generated in a format and in such a way that unequivocably identifies exactly the moment in time when an event has happened. This article presents an overview of how time stamps should be generated and in which format they should be recorded.

This article is the outcome of personal frustration in having to deal daily with time stamps which are not precise and unequivocable.

Time stamps are crucial in managing ICT incidents, forensics, and in general any audit, but also in debugging and overall ICT security. Every time one needs to understand what has happened in an ICT system, one has to establish a time line of events; the first characteristics of an event in such case is to have a time stamp associated with it.

So how is it that more often than not, in critical investigations time stamps fall short of establishing unassailable event time lines?

For example, it is not easy to establish a time line of events using logs loaded in a database when the original time stamps in the logs omitted the year, which was then added at the moment of loading into the DB. It has happened that looking at events of December 31, 2008, some records had a correct time stamp in December 2008 but others in December 2009 (it was possible to distinguish them only because they were in the future).

In other cases it has not been possible at all to establish a time line of events because the clocks of the machines involved were not aligned, making it impossible to compare events happening on different hosts. In this situation a network forensics analysis was not possible, and no evidence could be admitted in court.

What makes this situation especially incredulous it that there are international standards which describe and mandate how time stamps should be generated and recorded, yet even in new applications these standards are not followed. It seem very few people in the ICT security community fully appreciate the importance of getting the time right – until the moment they really need to have it, which is too late.

Indeed it is a scary, yet enlightening, exercise to imagine the possible consequences when time stamps are:

- **Imprecise**: for example, there is the date but not the time of the event
- **Incomplete**: for example, there are the time, the year, and the month of the event but not the day

- **Wrong**: for example, if the clock which has generated them is not aligned to a reference clock.

This article tries to make the case for why security practitioners must, both when developing applications and when managing them, to guarantee that the time stamps generated are standards-based, precise, and appropriate. The starting point of this article will be the review of traditional or historical time stamps, describing what is wrong with them, in order to understand why some effort is still needed in getting the time right.

Syslog time stamps

The prime place for time stamps is the log file, and the most common protocol and format for logging is the, by now ancient, syslog protocol.#1 If you look at a few syslog logs, you will see that each record in a file starts with the time stamp of the event, which is very good. But if you look at these time stamps, you will find something like the following examples:

```
Nov 21 16:36:27 localhost ...
Mar 26 18:14:58 localhost ...
```

where *localhost* is the name of the machine and not part of the time stamp. Now there are some obvious major problems with these time stamps:

1. The year is missing

2. The month is not in number notation but is an abbreviation in the current language (locale).

The omission of the year introduces a lot of problems, from the transition of December 31 to January 1 to the impossibility of re-loading old log files in analysis tools or databases and getting the year right.

The second point is less trivial than what it seems at first sight: in which language do you think is the second time stamp? English would be the general answer, but instead it was Italian. Consider now the following time stamp (generated by a generic date command):

```
Mar Mar 24 2009
```

Besides determining which language it is, the major question is understanding which is the month and which is the day of the week. (The correct answer is that the first is the name of the day of the week and the second the name of the month, again in Italian – but you knew that.)

Even generating only numerical time stamps, one can get an entry like the following:

```
02/03 18:14:58
```

which in some countries is interpreted as the second of March, while in others the third of February.

Instead, if all time stamps would be written in the following ISO-8601#2 format (ISO-8601):

```
YYYY-MM-DDThh:mm:ss.nnn[nnn[nnn[nnn]]]
```

T denotes the letter *T* (which can be upper- or lower-case) or substituted by a space; the square parenthesis denote optional terms – in practice it means that you can have 3, 6, 9, or 12 digits of precision). For example:

- `2009-01-01 00:00:00.123`
- `2009-12-31T23:59:59.123456789`

There can be hardly any doubt about the interpretation of these time stamps. In the next section it will be explained why it is required that milliseconds be present in a time stamp, whereas microseconds, nanoseconds, picoseconds, and beyond are optional.#3

If having dashes, spaces, and colons in the time stamp is a problem, then I suggest (this is not in any standard, but I have found it a very useful convention) adopting the following convention for the time stamps:

```
YYYYMMDD_hhmmss_nnn[nnn[nnn[nnn]]]
```

The previous examples can be rewritten as follows:

- `20090101_000000_123`
- `20091231_235959_123456789`

This is very useful format if, for example, you need to embed a time stamp in a file name.

Networks, time synchronization, and NTP

As far as comparing time stamps generated by the same clock, which in practice is ever hardly the case, one could think that the previous time stamp format would solve all problems.

But in my experience the general situation is to have time stamps generated by applications running on different computers, which obviously have different clocks. Immediately a fundamental question arises: How is it possible to set and keep synchronized the clocks of different machines?

The Network Time Protocol (NTP)#4 is a protocol which has been designed precisely for this application and is supported practically by all operating systems. NTP synchronizes the system's clock with a trusted reference time source, either local or one of the freely available international standard clocks. The theoretical precision of NTP is of 232 pico-seconds,#5 but in practice on standard hardware and OSs, the best one can get is milli-second synchronization. The typical NTP implementation is usually able to tell you how many milliseconds the clock of your computer differs from the reference time source – this is usually called the *clock offset*.

So you could believe that if you synchronize all your computer clocks with NTP and keep time stamps to the second, you would be safe. Unfortunately it is not so. As an example, consider two computers each with its clock synchronized to the same time source *but* the second clock is 20 milliseconds behind the first clock. This can happen due to the practical milli-second precision of NTP. Now the first clock registers an event at a certain instant in time and gives it a time stamp with time 10:10:10.001. Ten milliseconds later a second event is registered by the second clock, which gives it a time stamp with time 10:10:09.991. If one just reads the time stamps one, the conclusion would be that the second event has happened before the first, even limiting to seconds precision! (Notice that if both events would have been registered by the first clock, the second event would have had time 10:10:10.011.)

The imprecision interval

This example demonstrates the need to record time stamps to a precision related to the clock offset, typically millisecond precision, and that one should try to resolve the ambiguities due to the different clock offsets. Since it is practically impossible to keep track of the offsets of every clock,

including the reference clocks, the atomic clocks, etc., one can resolve to a practical approach which leaves room for some indeterminacy but makes things easy to manage. To do that one can introduce the concept of *imprecision interval*#6 as follows:

> The imprecision interval is the interval of time such that if two time stamps made by different clocks differ for less than this value, it is not possible to establish their order in time unless both clock offsets are exactly known.

A simple way to give an estimate for the imprecision interval is the following:

1. Synchronize all clocks to the same reference clock
2. Estimate the imprecision interval as twice the largest difference in time (clock offset) between any clock and the reference clock.

So, if the largest clock-offset on all servers is 29 milliseconds, the imprecision interval would be 58 milliseconds. With more than one reference clock, to estimate the imprecision interval one needs to keep in consideration also the clock offsets of the reference clocks among themselves.

Adopting this practical approach, it is not possible to decide the time order of two time stamps generated by different clocks and which differ by less than the imprecision interval. As a further consequence, this principle implies that all time stamps must be recorded to a precision at least as that of the imprecision interval. Since in most computer systems the imprecision interval ranges between 20 and 100 milliseconds, the precision usually required in time stamps is at least milliseconds.

Geography and time zones

What is possibly the most confusing part of managing time stamps is the *time zone*. We all know very well that we have to change the time of our watches when we travel to other countries. This is quite obvious due to the earth being spherical, rotating on its axis and with respect to the sun, and to the human requirement that 12:00:00 should mark approximately the time at which the sun is highest in the sky independently of where man is on the face of earth.

To manage the different time zones, from the first of January 1972 all time zones are defined with respect to the Coordinated Universal Time (UTC) time scale, which in practice substitutes what formerly was Greenwich Mean Time (GMT).

It is important to understand how a time zone is defined. A time zone, or a geographical time zone just to avoid any confusion, is defined by a name like Europe/Rome, America/New_York or CET (for Central European Time). Each geographical time zone specifies the time difference rules from UTC of the local clocks in that geographical zone in hours and minutes.#7 A typical definition for a geographical time zone is the following (this example is for the Europe/Rome time zone):

- `2009-03-29 00:59:59 UTC = 2009-03-29 01:59:59 Europe/Rome isdst=0 utcoff=+01:00`
- `2009-03-29 01:00:00 UTC = 2009-03-29 03:00:00 Europe/Rome isdst=1 utcoff=+02:00`
- `2009-10-25 00:59:59 UTC = 2009-10-25 02:59:59 Europe/Rome isdst=1 utcoff=+02:00`

- `2009-10-25 01:00:00 UTC = 2009-10-25 02:00:00 Europe/Rome isdst=0 utcoff=+01:00`

The explanation of these rules is as follows: each rule starts with a time-stamp in UTC and after the equal sign there is its value in the Europe/Rome time-zone local clock. In other words, to the left of the equal sign there is the datetime in Greenwich and to the right the same datetime but in Rome. Notice that on March 29 at 01:00:00 UTC, the time difference between the UTC and Europe/Rome timezone changes because the Europe/Rome zone goes to (summer) daylight saving time. This is why the Boolean indicator *isdst* (Is Daylight Saving Time) is set to be true, and the UTC offset changed accordingly.

Notice also that given the UTC time, one adds the UTC-offset to get the local time, vice-versa to go from a time stamp in a local time zone to the UTC time, one subtracts the UTC offset from the time stamp.

So in the previous example, between 2009-03-29T01:00:00 UTC and 2009-10-25T00:59:59 UTC the Europe/Rome geographical time zone is two hours ahead of UTC, whereas before and after this time interval the Europe/Rome geographical time zone is one hour ahead of UTC.

Going from UTC to a local time is always well defined, whereas the vice-versa it is not when using geographical time zones. Indeed if one would find the time stamp 2009-10-25 02:30:00 in the geographical time zone Europe/Rome without knowing if it was in daylight saving time or not, one would not know which of the two UTC time stamps it would correspond to. Notice that there is no ambiguity if the UTC offset is specified in a time stamp, and for this reason the UTC offset is often called also a numerical time zone.

From the previous example it follows that using geographical time zones in time stamps in local time can give ambiguous results whereas time stamps in UTC and time stamps with UTC offsets (or numerical time zones) are never ambiguous.

Since time in a local geographical time zone can jump if, for example, the zone goes to daylight saving time, and there is not a one-to-one relation with the UTC time, one must set the hardware clock of all computers to UTC. All OSs and applications must internally work with time stamps in UTC with a precision at least that of the imprecision interval of NTP (obviously NTP works in UTC).

(Note that most Unix OSs manage internally time stamps as the number of seconds from 1970-01-01 00:00:00 UTC, to which is added a precision in micro- or picoseconds. This is an equivalent and convenient way of managing internally in OS and applications time stamps in UTC, but it has the unfortunate consequence that on 32-bit machines, time will wrap on 2038-01-19 03:14:08 UTC.)

Each computer should be configured with a default local geographical time zone which specifies the rules to transform a UTC time stamp to a local time stamp. Optionally a user can ask for a time stamp to be printed in any geographical or numerical time zone. A geographical time zone should be used only to specify how to transform a time stamp from UTC to local time, but should never be used in the printed time stamp. So when printing a time stamp it is always necessary to specify in which time zone the time stamp is, and to do so one should always include also the UTC offset / numerical time zone as follows:#8

- `2009-12-31 20:12:23.123 Z`

- `2009-12-31 20:12:23.123 +00:00`
- `2009-12-31 20:12:23.123 -00:00`
- `2009-12-31 22:12:23.123 +02:00`
- `2009-12-31 14:12:23.123 -06:00`

The first space can be substituted by the letter T (upper or lower case) and the second space is optional. The letter Z and +00:00 denote both UTC, whereas -00:00 denotes UTC but in a system where the local geographical time zone is not defined. The fourth time stamp is in a geographical time zone two hours ahead of UTC, like CET or Europe/Rome in summer time, so that the corresponding time stamp in UTC is obtained by subtracting two hours. Similarly in the last example, the corresponding time stamp in UTC is obtained by adding six hours.

But why should one have UTC offsets/numerical time zones in time stamps? Besides resolving the ambiguities of the transition from/to summer-time, a short practical example should suffice. I was requested to establish the time line of events that happened one summer day on three different hosts using logs files from these hosts where the records had full time stamps but without UTC offsets/numerical time zones. A first analysis of the records in the log files with an automatic tool did not lead to anything. A manual comparison between the log files of the three hosts did not make any sense either until it was understood that time stamps on the three hosts differed by one hour each. The very unlucky situation was that one host was logging with time stamps in UTC, another in CET (+01:00) and the third in CEST (+02:00). If all time stamps generated instead the UTC offset/numerical time zone, the automatic tool would have done all the analysis in a matter of minutes and not the full day it ended up taking.

Summing up

To be as practical as possible, here is a short summary in a few rules of how to correctly manage time stamps:

1. When writing software, time stamps should always be managed internally in UTC and at least with millisecond precision

2. Default printed time stamps should be in full ISO-8601 format with a numerical time zone (i.e. UTC offset) like 2009-12-31T22:12:23.123456+02:00 (or 20091231_221223_123456+0200.filename.txt) at least with millisecond precision

3. The clocks of all machines should be set to UCT and aligned to reference clocks with NTP; on each host should also be configured a local time zone

4. The imprecision interval among all hosts in the network should be evaluated and ICT personnel should be aware that it is not possible to decide the order of events on different hosts whose time stamps differ by less than this value (typically 20 to 100 milliseconds).

About the Author

Andrea Pasquinucci, PhD, CISA, CISSP, is a freelance ICT security consultant. His main activities are strategic and global ICT security projects, governance, compliance, audit, and training. His main

technical fields of expertise are security of networks and operating systems, and cryptography. Andrea can be contacted at posta@ucci.it.

References

1 C. Lonvick, *The BSD syslog protocol*, RFC 3162, August 2001.

2 *Data Elements and interchange formats – Information interchange – Representation of dates and times*, ISO 8601:2000, International Organization for Standardization, December 2000; G. Klyne and C. Newman, *Date and time on the Internet: timestamps*, RFC3339, July 2002.

3 To obtain nano- and pico-second precision you probably need special hardware and software whereas usual COTS and OS allow to reach milli- and micro-second precision.

4 J. Burbank, W. Kasch, J. Martin and D. Mills, *Network Time Protocol (NTP) Draft Version 4*, http://www.eecis.udel.edu/~mills/ntp.html; D. Mills, *Network Time Protocol (NTP) Version 3*, RFC-1305 March 1992; D. Mills, *Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI*, RFC-2030 October 1996.

5 Since January 1, 1972 at 00:00:00, the time and frequency standards of the world have been based on the International Atomic Time (TAI), which is defined and maintained using multiple cesium-beam oscillators to an accuracy better than a microsecond per day. This date is also the reference date for the UTC timescale, see below.

6 This approach is in spirit similar to the one of an experimental physicist who must know the precision of the instruments he uses for his experiments; in other words, an experimental physicist must know and manage the errors (imprecision) of each measurement.

7 Before the advent of UTC, time zones could differ also in seconds, this is not anymore possible since 1972-01-01.

8 Note, newer syslog implementations, for example rsyslog (http://www.rsyslog.com), print time stamps in this ISO-8601 format by default.