Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

A. Malhotra
Boston University
M. Hoffmann
Open Netlabs
W. Toorop
NLnet Labs

October 30, 2017

On Implementing Time
draft-aanchal-time-implementation-guidance-00

Abstract

This document describes the properties of different types of time values available on digital systems and provides guidance on choices of these time values to the implementors of applications that use time in some form to provide the basic functionality and security guarantees.

1. Introduction

The basic functionality and security guarantees claimed by many applications running on digital systems locally or in the Internet hinge on some notion of time. These applications have to choose one of the many types of time values available on the system, each of which has its own specific properties. However, currently these applications seem to be oblivious to the implications of choosing one or the other time value for implementation. This behaviour can be attributed to: a) the lack of clear understanding of the distinct properties of these time values, b) trade-offs of using one or the other for an application, and c) availability and compatibilty of these time values on different operating systems.

In this document we describe the properties of various available time values on modern operating systems, discuss the trade-offs of using one over the other, and provide guidance to help implementors make an informed choice with some real-life examples.

2. Keeping Time: Different Clocks

Because time is relative to an observer, there cannot be a universally agreed upon time. At best we can achieve an approximation by updating our own observed time with a common reference time shared with other observers.

As this reference time is what we naively assume clocks on a wall are showing, we shall call it the "wall time." For most applications, it is based on the Universal Coordinated Time (UTC), an international standard time determined by averaging the output of several high-precision time-keeping devices. However, as UTC is following Earth's solar time, it occasionally needs to be adjusted through leap seconds.

An individual computer system's preception of time differs from this idealized wall time. Staying close to it requires some effort that comes with its own set of drawbacks. Systems therefore provide access to different types of clocks with different properties. Unfortunately, there is no standard terminology and definitions for these types. For the purpose of this document, we therefore define three different kinds of clocks that a system may or may not provide.

2.1. Raw Time

At its most fundamental, a system has its own perception of time; its unmodified, "raw time." This time is typically measured by counting cycles of an oscillator. Its quality therefore relies on the stability of this oscillator.

As it is a purely subjective time, no general meaning can be attached to any specific value. Only the amount of time passed can be determined by comparing two values.

Because raw time is unaltered, it is continuous and strictly monotonically increasing. Its value will always grow at a steady pace, never decrease, never make unexpected jumps, or stip. Such a time is sometimes called a "monotonic time."

2.2. Adjusted Raw Time

Even if highly accurate oscillators are used, raw time passes at a slightly different rate than wall time. This difference is called clock drift. It depends not only on the quality of the time source but also on environmental factors such as temperature.

When this drift is componsated by comparing the passage of raw time to some external time source that is considered to be closer to wall time, the result is "adjusted raw time." This adjustment doesn't happen sporadically but rather, the rate of advance of time is slowed down or sped up slightly until it approaches the reference time again. As a result, adjusted raw time is still monotonic. Like raw time, adjusted raw time is subjective with no specific meaning attached to its values.

The most frequently used method of acquiring an external time source is through network timing protocols such as NTP [RFC5905]. As a result, adjusted raw time is susceptible to vulnerabilites of these protocols which may be exploited to maliciously manipulate this time.

2.3. Real Time

With adjusted raw time, a system already has access to a time that passes at a rate very similar to wall time. By adjusting the time value so that it represents the time passed since an epoch, a well-defined point of wall time such as seconds since midnight January 1st, 1970 on Unix systems, time values themselves gather meaning. The result is "real time."

While it is often assumed that real time is set to match wall time, this doesn't need to be the case. A system's operator is free to

change the value of real time at any time, likewise, system services such as a local NTP client may decide to do so.

As a consequence, real time is not monotonic. Not only may it jump forward, its value may even decrease.

2.4. Differences from Wall Time

These three clock types differ from wall time in three aspects:

- Both raw time and adjusted raw time can only represent differences in time by comparing two clock values. Only real time provides absolute time values that can be compared to wall time values.

- On the other hand, raw time and adjusted raw time are always monotonic whereas real time may experience sudden changes in value in either direction.

- Only adjusted raw time and real time are subject to external adjustments so that time passes at approximately the same rate as wall time. Raw time will over time drift away due to inevitable imperfections of the clock.

3. Expressing Time

Protocols or applications can express time in one of the two forms, depending on whether global agreement over the point in time is necessary.

3.1. Time Stamps

A "time stamp" expresses an absolute point in time. In order to reference the same point across multiple systems, it needs to be stated in wall time.

Time stamps are often used to express the validity of objects with a limited lifetime that are shared over the network. For instance, PKIX certificates [RFC5280] carry two time stamps expressing their earliest and latest validity.

In order to validate a time stamp, a system needs access to a clock that is reasonably close to wall time.

3.2. Time Spans

In contrast, a "time span" expresses a desired length of time. Examples of time spans are timeout values used in protocols to

determine packet loss or Time to Live (TTL) values that govern the lifetime of a local copy of an object.

While no access to wall time is necessary for correctly dealing with time spans, using a clock whose time passes at a different rate than wall time will result in different interpretations of time spans by different systems. However, in a network environment, the uncertainty introduced by differing transmission times is likely larger than that introduced by clock drift.

4. Current Implementations and Their Flaws

Currently, some software takes a common approach towards time stamps and time spans. Time stamps are registered with their wall time value, and time spans are registered with two time stamp values marking the start and the end of the span. Conversion of a time span into those time stamp markers is regularly based on real time.

Note that the start of a time span will be the current (real) time in case of a TTL. So, in case something needs to be cached for a certain time, the start time stamp is irrelevant and it is registered together with only the (real) expiration time.

Programmers might have had different reasons to base those markings on real time, for example:

1. A point in time is intuitively thought of as a wall clock time stamp. Time stamps from outside the software, which the software has to manage are already in wall clock time. The POSIX function to get the current (real) time which is regularly used for this, is gettimeofday(), which comes accross as something providing near wall clock time and which can be used for this purpose.

2. Managing time stamps and time span similarly, prevents code complexity.

   For example, many software is organized around I/O event notification mechanisms like the POSIX select() and poll() system C API functions. These functions wait for a given time span for file descriptors to become ready to perform I/O. The given time span is determined by substracting the current real time value from smallest registered time stamp. When file descriptors are ready, the non-blocking I/O is performed, otherwise the given time span has passed and the action associated with the smallest registered time stamp needs to be performed.

   For this programming pattern, a sorted list of time stamps has to be maintained by the software. To avoid coding complexity,

programmers might prefer a single list for both actual wall clock time stamps and those generated from real time to mark the end of a time span.

Using real time as a basis for the time stamps marking the start and end of a time span is bad because of the following reasons.

1. It can be set or overwritten manually,

2. It is subject to adjustments by timing protocols which on one hand is important to make sure that this time is in sync with the rest of the world but on the other hand makes it dependent on the correctness and security of timing protocols.

Recent attacks [SECNTP], [MCBG] show how timing protocols like NTP can be leveraged to shift real time on systems.

Time stamps are always based on wall time, so the best one can do is to use real time while dealing with them. However, this limitation does not hold for the time spans. Managing time spans may be implemented in alternative ways which may prove to be more secure and robust.

An obvious question to ask is: Why do we need inception and expiration time stamps in the first place to define the validity period of cryptographic objects? Why can't we just use time spans like TTL values instead? The reason is straightforward.

The authority determining and setting the validity period on the object can be different from the operator delivering the object. For example the TTL value on DNS resource records indicates to caching DNS resolvers how long to cache those records. These are an operational matter and are thus left to the operators of the DNS zone.

The content of the resource records are however determined by the signer of the records. When she is not also the zone operator, she has no way to determine when the records will be queried for, and thus has to depend on cryptographically signed wall clock based time stamps to limit the validity.

Note however that DNSSEC signatures do contain the original TTL of a resource record set, restricting the maximum TTL value with which the operator may deliver the resource records.

5. Alternative Approaches

For time spans, where we only need the rate of passage of time to be close enough to the rest of the world, one should not use the real time to establish the start and end time for the reasons mentioned above. The other two types of time are raw time and adjusted raw time. The important aspect of these monotonic time sources is not their current value but the guarantee that the time source is strictly linearly increasing and thus useful for calculating the difference in time between two samplings. But each comes with its own caveats.

> Raw time is not subject to any adjustments by timing protocols, i.e., it is not adjusted for the error introduced by clock drift. This could have two repercussions. First, this makes correctness of raw time independent from the errors or security vulnerabilities of the timing protocols. Second, its correctness depends on the clock drift which further depends on various factors such as quality of the oscillator, work load, or ambient temperature on the system and may vary.

> Adjusted raw time, on the other hand, is subject to adjustments by timing protocols. While it therefore compensates for the errors introduced by the drift of the local clock, this time can be incorrect as it is vulnerable to accuracy and security vulnerabilities of the underlying timing protocol.

The choice of time value to be used is application-specific. For instance in applications that can tolerate a certain amount of clock drift [CLOCKDRIFT], implementers can use raw time. However, if that is an issue then one has no choice but to fall back to adjusted raw time.

POSIX defines a system C API function which may provide raw time: clock_gettime(), when used with a clock_id of CLOCK_MONOTONIC (when supported by the system). POSIX does not make a distinction between raw time and adjusted raw time in the definition of this function. Beware that with some systems, CLOCK_MONOTONIC deliveres adjusted raw time and that CLOCK_MONOTONIC_RAW needs to be used as clock_id to get unadjusted raw time. Non-POSIX systems may provide different APIs

Software employing the pattern organized around I/O event notification mechanisms, as described in Section 4, should maintain two sorted lists of two different types of time stamps:

1. One to register events based on time stamps expressed in wall clock time

2. One to register the start and end of time spans in (adjusted) raw time

To determine the timeout value for a call to select() or poll(), the program needs to get the current time in both real time and in (adjusted) raw time. The current real time is substracted from the lowest value of the time stamps expressed in wall time list. The current (adjusted) raw time from the lowest value of the time stamps expressed in (adjusted) raw time list. The lowest of the values should be used as the timeout value for select() or poll() and determines which action should be performed when te function times out.

Alternatively a single list of (adjusted) raw time could be used for both time stamps and time spans. In that case time stamps expressed in wall clock time should be converted into (adjusted) raw time, by first converting it into a time span by substracting real time from it, and then adding the current time in (adjested) raw time.

6. Acknowledgements

We are thankful to Sharon Goldberg and Benno Overreinder for useful discussions.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

Time is a fundamental component for the security guarantees claimed by various applications. Therefore, any implementor concerned with security should be concerned with how these time values are implemented. This document discusses the security considerations with respect to implementing time values in applications in various sections.

9. Informative References

[CLOCKDRIFT]
Marouani, H. and M. Dagenais, "Internal clock drift estimation in computer clusters", 2008, <http://downloads.hindawi.com/journals/jcnc/2008/583162.pdf>.

[MCBG] Malhotra, A., Cohen, I., Brakke, E., and S. Goldberg, "Attacking the Network Time Protocol", 2015, <https://eprint.iacr.org/2015/1020>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[SECNTP] Malhotra, A., Gundy, M., Varia, M., Kennedy, H., Gardner, J., and S. Goldberg, "The Security of NTP's Datagram Protocol", 2016, <http://eprint.iacr.org/2016/1006>.

Authors' Addresses

Aanchal Malhotra
Boston University
111 Cummington Mall
Boston 02215
USA

Email: aanchal4@bu.edu

Martin Hoffmann
Open Netlabs
Science Park 400
Amsterdam 1098 XH
Netherlands

Email: martin@opennetlabs.com

Willem Toorop
NLnet Labs
Science Park 400
Amsterdam 1098 XH
Netherlands

Email: willem@nlnetlabs.nl