6.XX Clock Issues

6.XX.1 Description of application vulnerability

All processors and operating systems maintain multiple representations of time internal to the system. In a typical system there are the following notions of time, and potentially identifiable clocks:

- CPU time
- Process/task/thread execution time
- Calendar clock time, local and/or GMT
- Elapsed time - i.e. time since system inception in seconds, or in fixed portions thereof
- Network time

These times have different representations, different scaling, and different semantics. For example, a time-of-day clock must account for leap years, leap seconds and standard/daylight saving times. A CPU or processor clock is a monotonic clock must maintain time used by a task / thread / process in a granularity appropriate to CPU speed - possibly sub-nanosecond. A real time clock is a monotonic clock that manages and represents time to a granularity and representation needed to correctly manage the algorithms of the system.

Some of these clocks are manifested in programming languages. For example, most languages have tome of day clock lookup, while real time languages often include monotonic clocks for various purposes. Alternatively, some languages provide library services to access, and manipulate time bases and to schedule activity based upon one of the time bases.

Time Conversion

When multiple time bases are supported, there are mechanisms to convert from one time format to another to support calculations done. Conversion errors, rounding errors or cumulative errors can develop:

- If the conversion is not done from the most precise time formats to less precise time formats,
- If conversions are done from one format to another and then back for comparison, or
- If iterative calculations are done using less than the most precise time base possible.

This can lead to missed deadlines or wrong calculations that depended on accurate time representation and can result in catastrophic loss of the application or the parent system. A classic example of this is the common (wrong) paradigm to use the calendar clock to derive values to be programmed into the monotonic clock.

Synchronicity

When code is written for an application, the developer usually assumes that there is a common time base for all portions of the application that are in communication with each other. When the system is spread over multiple processors, it the time base used by each processor will either drift from each other, or the time delay in communicating between these partitions will cause apparent drift.

Time Roll-over

Because each clock has a fixed internal representation of time which is updated periodically by some amount, eventually, if the system is long-enough lived, the time representation will completely fill the storage and will roll-over and return to zero, or the initial time. This can also happen if the time base is external, such as the global positioning satellite time base. Code that relies upon the time-base constantly increasing will fail if/when a rollover occurs, leading to failure of the computational system and possible catastrophic loss of the parent system, unless the application is programmed to account for this rollover.

Most systems create a real-time time base such that the system will never roll over within the expected operational time of the system. Modifications to the system, however, such as speeding up the clock that feeds the time base or dramatically increasing the expected operational lifetime of the system can make such errors happen, with potential catastrophic loss of the system and any systems that depend upon it.

6.XX.3 Mechanism of failure

The time of day clock is adjusted internally to jump or to be set backwards when going to or leaving summer time, inserting leap seconds, switching time zones or correcting time to synchronize the clock with a time base or another clock. Using the wrong clock, especially the ToD clock, to schedule events can result in jitter in the system, events being scheduled early, or the event being late. The mischeduling of events can have real world applications up to and including catastrophic loss of the parent system.

Converting from one time-base to another time-base can result in loss of precision, rounding errors, and conversion errors which can lead to complete jitter in the application behavior or complete failure of the application

6.XX.4 Applicable language characteristics

The vulnerability is intended to be applicable to languages with the following characteristics:

Languages that support a model of time.

6.XX.5 Avoiding the vulnerability or mitigating its effect

Software developers can avoid the vulnerability or mitigate its effects in the following ways:

- Always convert time from the most precise and stable time base to less precise time bases.
- Avoid conversions from calendar clocks or network clocks to real time clocks.

- Avoid using the time of day clock to schedule events, unless the event is demonstrably connect with real world time of day, such as setting an alarm for 7 am.

- Avoid resetting or reprogramming the real-time clock or execution timers, unless the complete application is being reset. Allow some variability or error margin in the reading of time and the scheduling of time based on the read.
- Use only clocks that have known synchronization properties.
- Protect any code that uses real-time time bases with any potential of roll-over from going from a large value to a zero or a negative valuer. This is done by assuming that a rollover can occur and if it is expected that always `T1<T2`, but is found that `T1` is nearing `Time_Base'Last`, then `T2<<T1` will be accepted.

6.XX.6 Implications for standardization

In future standardization activities, the following items should be considered:

6.YY Resource Consumption Measurement

6.YY.1 Description of application vulnerability

All applications consume resources as they execute, in particular Time. Each thread, event, interrupt and OS service consume CPU time that may be separately measurable by the system.

A common paradigm in managing applications is to monitor such resource usage by thread and take action to cease the calculation for that thread, such as abort, raise exception, lower priority or suspending the thread. If the calculation cannot be completed in time or within the resource constraints imposed upon it, then the application may fail.

The consumption of CPU resources (execution time) can be affected by changes in the CPU itself: for example, CPU's may slow down to manage heat, resulting in more execution time to achieve a result. Similarly, cache misses due to the way a program is organized and executed, due to multiprocessor effects, can increase the execution time needed to complete a calculation.

6.YY.2 Cross references

6.YY.3 Mechanism of failure

Many applications measure resource consumption to detect failures of portions of portions of the algorithm and to make decisions about alternative actions. For example, excessive consumption of CPU may indicate that a thread is executing erroneously; or that other needed threads may not be able to execute due to excessive resource consumption.

Other factors, such a CPU speed changes and cache misses can cause a thread to consume significantly more CPU resources than expected to perform the same calculations.

A thread executing more resources than planned can result in missed deadlines for itself, or can take resources needed by other threads, causing incorrect processing or missed deadlines for other threads. Missed deadlines are catastrophic for hard real-time systems, and cover the range of causing wrong results through to complete failure of the application.

6.YY.4 Applicable language characteristics

6.YY.5 Avoiding the vulnerability or mitigating its effect

Software developers can avoid the vulnerability or mitigate its effects in the following ways:

- Verify or test the application on systems that are executing in the slowest system configuration
- Where cache misses provide a significant potential hindrance, execute the application with cache disabled

6.ZZ Missed events or Deadlines

6.ZZ.1 Description of application vulnerability

Many real time systems are characterized by collections of jobs waiting for a start-time for a time-based iteration, or an event for sporadic activities. A common mistake in programming such systems is to base the start time of the next iteration upon either a non-monotonic or a non-real time clock, or to base it upon an offset from the start time or completion time of the last iteration. In the first case, conversion errors and possible drift of the real time clock can cause the next iteration to be wrongly programmed. In the second case, higher priority work may have delayed the actual start or completion of the task in an individual iteration, resulting again in time drift.

With enough drift, an iterative task will begin missing its deadlines, and will either produce the wrong results, or will fail completely, resulting in arbitrary failures up to catastrophic loss of the enclosing system.

Many systems have moved to a virtualization approach to fielding systems. Sometimes the virtual system is only an OS change, such as running Windows and Linux on the same hardware. Sometimes the virtual system is hardware and software. Sometimes hardware is dedicated, such as 2 cores from an 8 core system, while in others the virtual system under consideration only executes when needed. The discussion of virtualization includes the common notions, such as VMWare™, Hypervisor™, but also include systems as diverse as satisfying ARINC 653[ARINC 653], which uses a time-based partition approach to schedule mixed criticality systems on a single CPU.

In any case, when a system is virtual, its connection with the real world (i.e. hardware and virtualizer) clocks is indirect. Clocks for the virtualized system are updated when the system resumes, and time may “jump” or may advance much faster than normal until the clocks are synchronized with the real world. This can result in processes being mis-synchronized or missing deadlines if time jumps or progresses too quickly for the task to get its work completed.

If an attacker is aware that an application is virtualized, and can determine what other virtualized applications share the same resource, they may be able to generate load for the other virtualized applications so that the one in question can not retain enough resources to function correctly.

6.ZZ.2 Cross references

6.ZZ.3 Mechanism of failure

Any change in the progression of time can result in a disconnect between the spacing of the delivery of time events to the application, and can make jobs within the application run past their deadlines (as viewed by the timing events).

Deadline overrun is a serious flaw in the application, and usually results in failure of portions of the application up to catastrophic failure of the application and may result in loss of the parent system.

When a system is virtualized, an attacker can use influence over other applications to consume resources needed by the critical system that could trigger such systems.

Programming mistakes, such as failure to use monotonic clocks to schedule iterations, or incorrectly programming the next iteration calculations (such as setting the next wake time based on the the start of the current wake time vs a fixed offset from the previous scheduled start time) result in drift or jitter which may result in missed real world inputs or loss of synchronization with external systems.

6.ZZ.4 Applicable language characteristics

6.ZZ.5 Avoiding the vulnerability or mitigating its effect

Software developers can avoid the vulnerability or mitigate its effects in the following ways:

- Always set the next (absolute) start time for the iteration from the the start time of the previous programmed iteration.
- Only use the real-time clock in scheduling tasks or events.
- Create management jobs that can monitor and detect