Time Issues in Programs
Vulnerabilities for Programming Languages or Systems

Stephen Michell,
Maurya Software Inc, Canada
stephen.michell@maurya.on.ca

Abstract

ISO IEC JTC 1 SC 22 WG 23 Vulnerabilities Working Group is documenting new vulnerabilities in preparation for the release of TR 24772 edition 3. An identified area is in the accounting and management of time in normal systems and realtime systems. Various vulnerabilities related to time are documented, as well as avoidance and mitigations of the issues raised.

1. Introduction

The Programming Language Vulnerabilities Working Group (ISO IEC JTC 1 SC 22 WG 23) has been identifying and documenting weaknesses in programming languages since 2006, and documenting its findings in ISO IEC TR 24772 "Guidance to avoiding programming language vulnerabilities"[1]. The first edition of TR 24772 was published in 2010 and described a set of vulnerabilities that were described in a language-neutral way. It also included a set of vulnerabilities that are more related to the environment in which an application may reside, such as related to filing systems, OS traits, and communication protocols. At the time of publication, WG 23 acknowledged that there were a number of weaknesses not yet documented, such as concurrency issues, and that the document required a description of how typical languages deal with the vulnerabilities identified.

A major part of the first edition, and all following editions, is sub-subclause 6 of each vulnerability description which gives developers specific guidance on ways to avoid the vulnerability described.

WG 23 has relied on the Common Weakness Enumeration (CWE)[8], the JSF C++ coding guidelines[7], MITRE C[4] and MITRE C++[5] guidelines, and Guidance in the use of Ada for High Integrity Systems[6] for base material for its document.

The second edition was published in 2012 and added language-specific annexes that show how each language documented exhibits, avoids or mitigates the vulnerabilities identified. Annexes for Ada, C, PHP, Python, Spark, and Ruby were added. Edition two also added some vulnerabilities associated with concurrency, but the format of the document was relatively unchanged.

WG 23 is now working on edition three of TR 24772[1]. TR 24772 is being subdivided into multiple documents, called "Parts", with the original "main" document being TR 24772-1, The Ada specific annex becoming TR 24772-2, and so on. New language-specific parts will be added for Fortran, COBOL, and likely C++, C# and Java. Wording of the "avoidance" subclause was tightened up to be more directive and usable in coding standards. New vulnerabilities are being added to document vulnerabilities associated with object-oriented programming, such as deep-copy and polymorphism. The floating-point vulnerability was enhanced with the help of floating point experts. Also, the avoidance mechanisms from all the "avoidance" subclauses are aggregated to provide a summary of guidance, becoming effectively the top-N avoidance rules.

In analysing missing vulnerabilities from Edition 2, WG 23 decided that a set of vulnerabilities for "Time" are required for Edition three. To that end, the issues identified so far are presented in this paper, with a request that they be discussed and confirmed as vulnerabilities, and mitigations discussed. In addition, any missing time-related vulnerabilities will be noted and returned to WG 23 for documentation. Liberal use was made of [3] which enumerated a number of concurrency and time vulnerabilities.

2. Time Usage Based Vulnerabilities

2.1. External Visibility of Usage parameters vulnerability

This vulnerability is characterized by the measurement of external parameters of an application, and using those measurements to make determinations that allow information about algorithms used or information about the data being processed. A typical example is that a smart card is inserted into a card reader and exchanges encrypted messages with a remote system. A listener is placed into the reader where it can capture the encrypted message as well as measure the time taken to encrypt/decrypt messages and/or the power drawn to do the processing.

Many low power devices must use the smallest possible encryption keys in order that they can encrypt data in a limited time. If an attacker can determine how much time was taken for the encryption and knows the message size, then the algorithm can be determined and brute-force decryptions become possible. The same attack using the power drawn by the victim is possible.

Mitigations include:

- write the sensitive algorithm so that, no matter what algorithm it uses, it uses them same amount of time, consumes the same amount of space and consumes the same amount of power.

3. Differing Time Bases 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 process or processor clock must maintain time used by a task / thread / process in a granularity appropriate to CPU speed - possibly sub-nanosecond. A real time clock must manage and represent time to a granularity and representation needed to correctly manage the algorithms of the system.

There is a requirement in every system to convert time from one 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.

Mitigations:

- Always convert time from the most precise and stable time base to less precise time bases.
- Never convert from calendar clocks or network clocks to real time clocks.

4. Clock rollover vulnerability

All computer systems, by their nature, have a fixed internal representation of time. The most basic representation is usually that time is stored in a word (16, 32 or 64 bits) of fixed length. The clock is updated periodically by incrementing the timer word by one or more for each clock tick, such as every nanosecond, or every microsecond. 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.

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.

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.

Mitigation:

Always protect any code that uses real-time time bases from potential rollover. 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.

5. Virtualization

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. When we discuss virtualization, we include the common notions, such as VMWare™, but we 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.

See also section 7.2 for related issues in non-virtual systems.

Mitigations:

- If a critical application is virtualized, take steps to guarantee that processors, memory and time resources are locked to the application and not shared with other virtual services.
- Do not virtualize critical applications.

6. Synchronicity Vulnerability

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 is likely that the time base used by each processor will either drift. As time refinement uses smaller granularity, even processors that are only centimeters apart may not have exactly the same time base. In such cases, tests for equality of time could lead to application failures. Another variation is that a request to fetch a time may be routed to another processor where the time base is maintained, resulting in incorrect values being read.

Mitigations:

- 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.

7. Real Time scheduling vulnerabilities

When the application is a real time application, then the correct execution of the application depends not only on the correctness of the calculations done by the application, but also on the timeliness of the calculations. In order to function correctly, such real time applications are constrained the execution time of critical code segments, by the execution load of code that executes at higher priority such as interrupts, or by overheads from the runtime kernel or garbage collection.

A characteristic of real time applications is that they are embedded into larger systems, and are used to control important aspects of the enclosing systems. The failure of a real time application can lead to catastrophic outcomes for the enclosing system. The design of such real time programs must therefore take into account the importance of correct execution of the application. Static verification of the correctness and timeliness of the application is a hallmark of such applications.

7.1. Missed Deadline

Many real time systems are characterized by time-locked loops, scheduled by a hard real time timer. Simple periodic activities and pieces of more compute-intensive work are allocated to specific loop iterations to balance the load of all cycles. In such a system if an individual iteration exceeds its time bound and overflows into the next iteration's start time, then the deadline has been exceeded and the application cannot recover. It may have checkpoints and may restart, or the failure could be catastrophic, resulting in damage to the parent system up to and including loss of the parent system.

Mitigations:

- Program in a more flexible – priority/task –based way
- Improve the analysis to detect potential deadline overruns.

7.2. Scheduling the next iteration

Many real time systems are characterized by collections of jobs (tasks in Ada) 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.

This vulnerability has some characteristics of section 5, Virtualization, in that clock jumps or regressions can happen if the time of day clock is used and is reset to match GPS time, network time or time from a different processor. In such scenarios, some clocks can progress slower than expected or may even regress due to synchronization issues.

Mitigations:

- 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.

7.3. Reading or setting a real time clock, interrupts and events?

Real time systems are characterized by tasks interacting with interrupt hardware, runtime events, and the various clocks in the system, either reading, writing or being scheduled by them. Errors can develop if either the actual calls to manipulate these system-level services are not protected against concurrent access, or if the most precise bases for calculating the next set of services is not used. This can result in jitter in the system, or in missed notifications, which can result in catastrophic loss of the parent system.

Mitigations:

- Ensure that all access to system-level service is protected from concurrent access.
- Ensure that the task is in the correct state to receive system-level notifications so that it can act upon them in a timely manner.

7.4. Time accounting

Computer systems provide mechanisms to determine how much time a given process / task consumes, either in terms of a passage of time or a time usage mechanism. Some systems provide a time budget which aborts the portion of the application or the application itself if the budget is exceeded. Tasks or applications are often permitted to have the budget reset as needed.

There are a few challenges with the time budget approach to application management. One challenge is that system-level work, such as interrupt handling or garbage collection is often charged to the task / application that is currently scheduled, even though the work is not part of the task / application. Another is that the time accounting may have been created for a single-cpu system and the presence of multiple cores or virtualization changes the way that accounting is done.

Mitigations:

- Validate all interactions with the time accounting subsystems to ensure that assumptions made hold true.

8. Conclusions and future work

SC 22/WG 23 is still identifying vulnerabilities that need documentation. Vulnerabilities that have resulted in known attacks or known failures are the highest priority. In addition to the vulnerabilities themselves, WG 23 requires that we identify real-world mitigations so that such vulnerabilities can be eliminated or neutralized.

Bibliography

[1] ISO IEC TR 15942:2012, "*Information Technology – Programming Languages – Guidance on Avoiding Programming Language Vulnerabilities*", International Standards Organization, Geneva, Switzerland, 2012

[2] ISO IEC 8652:2012. "*Programming Languages and their Environments – Programming Language Ada*". International Standards Organization, Geneva, Switzerland, 2012.

[3] Burns, Alan, and Wellings, Andy. "*Programming Language Vulnerabilities – Lets not forget concurrency*", International Real Time Ada Workshop 14, ACM Ada Letters, New York, NY, 2012.

[4] Motor Industry Software Reliability Association. "*Guidelines for the Use of the C Languag(s in Vehicle Based Software*", 2012 (third edition).

[5] Motor Industry Software Reliability Association."*Guidelines for the Use of the C++ Language in Vehicle Based Software"*, 2008.

[6] ISO IEC TR 14592, "Guidance for the use of Ada in high integrity systems", ISO, Geneva Switzerland, 2001.

[7] Stroustrop, B. "Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program" , available from http://www.stroustrup.com/JSF-AV-rules.pdf

[8] Common Weakness Enumeration, MITRE Corp, cwe.mitre.org