Project Part 3 – Final Project Report
Christopher Zimmer
04/21/2009

The goal of this project was to implement security methods in cyber physical systems. This work is a continuation of on going research using static timing analysis and attributes isolated to real-time systems to create a secure environment. Security in cyber-physical systems can not be addressed in the same way it is done in general purpose systems. Cyber-physical systems often lack sophisticated operating systems and powerful general purpose processors. Predictability in this realm is more important than raw processing power and thus we have this trade off. Securing our cyber-physical systems is extremely important. Attacks on unsecured systems are more frequent now making it an appropriate time to investigate how to secure these systems.

The primary aim of this work was to implement a periodic scheduler based security check. In this check the scheduler would periodically wake up, and attain the last PC value of the last running thread. It would then use the PC value to isolate the worst case execution time that this PC could occur in. To handle this required making several modifications to an existing real-time simulation framework consisting of the Simple Scalar Toolset, and a Static Timing Analysis Toolset.

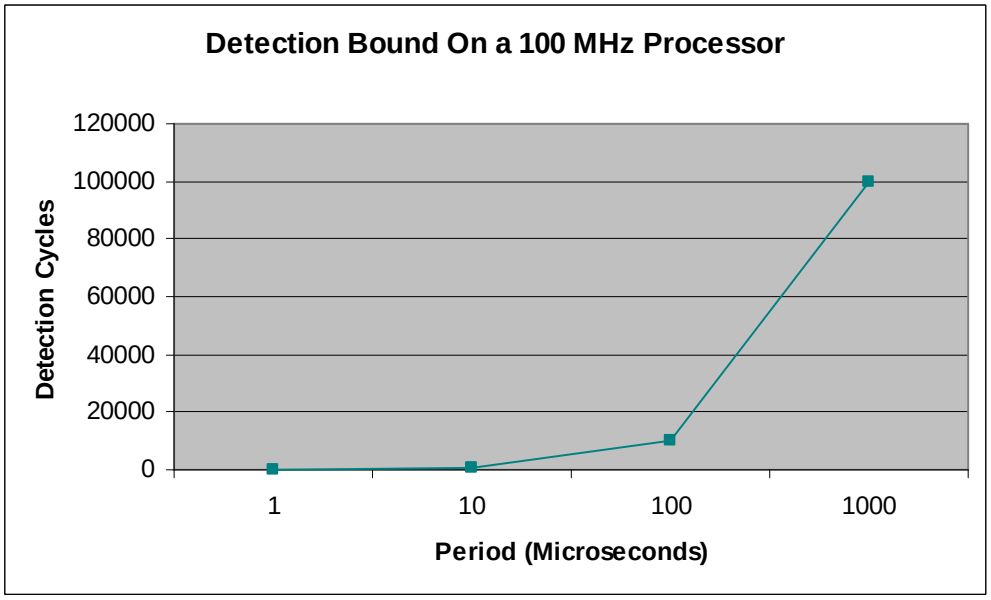
Week 1
In the first part of this work it was necessary to perform background reading to determine if any work from general purpose could be used to support this proposal. There doesn't appear to be any work in general purpose that deals with strict timing requirements to facilitate security mechanisms, but there is work in securing the stack, using a variety of methods to protect against code injection. These include stack shield, address space layout randomization, and compiler based canary values.

Week 2
The Simulator environment I am working in has been modified with a new system call that allows the scheduler to set up a periodic timer interrupt. This is a programmable interrupt made from the scheduler. My current scheduler is currently being interrupted every 2 us in simulation time. The frequency of the scheduler is clocked at 1 Ghz so that's roughly equivalent to every 2000 cycles. I feel that interrupts every 2000 cycles will be a good starting point in evaluation to determine cost and trade-off regarding overhead of this project vs detecting timing anomalies. After further investigation I reduced the frequency of the simulator to 100 Mhz and set the interrupt to occur every 20000 cycles which is 200 us.

Week 3
I have modified the scheduler and the simulator to support new system calls that enable the scheduler to determine the last running PC of the last task in a single processor environment. A test real-time task set has been configured using two simple clab benchmarks CNT and SRT. The tasks have been analyzed in both a regular timing analyzer tool set and a parametric timing analyzer tool set.

Week 4

During this week I created an extension to the proposal that would extend a current security method and improve upon it using the periodic timer interrupt. In the previous work for this area I implemented a Scheduler based timer interrupt that used calls made from the running job at predetermined timed locations to validate that the job was running within bounds. One of the drawbacks of this method was that if an attack occurred and never returned back to the running job, this system would only detect it at the occurrence at a deadline miss. For real-time systems this isn't desirable. So as an extension to that using this work I implemented a similar approach using periodic timer interrupts that could detect missed checkpoints at the next interrupt thus hopefully providing the system with more time to handle the attack than a deadline miss. In the figure below I show the trade off in detection cycles for scaling the period up on a 100 Mhz Processor.



Most of the benchmarks that I've utilized in embedded real-time systems generally seem to run between 1 Million and 100 Million cycles. So for this particular set it seems to make sense to have a larger period and still get a large number of checkpoints occurring in the application. Of course the observation made above cannot possible encompass all real-time jobs and thus deciding the period is ultimately a decision based on the real-time task set, or setting a different period for each job when the scheduler puts it on the processor. The figure below shows the time after an attack occurred that this method was able to detect it and put an end to it. In this figure all of the attacks never return back to the running application.

Task	Hijack Location	Time Attack Forced to End
FFT	FFT Method Return	1660 Cycles
LMS	LMS Method Return	869 Cycles
CNT	Scheduler Check 2	1690 Cycles

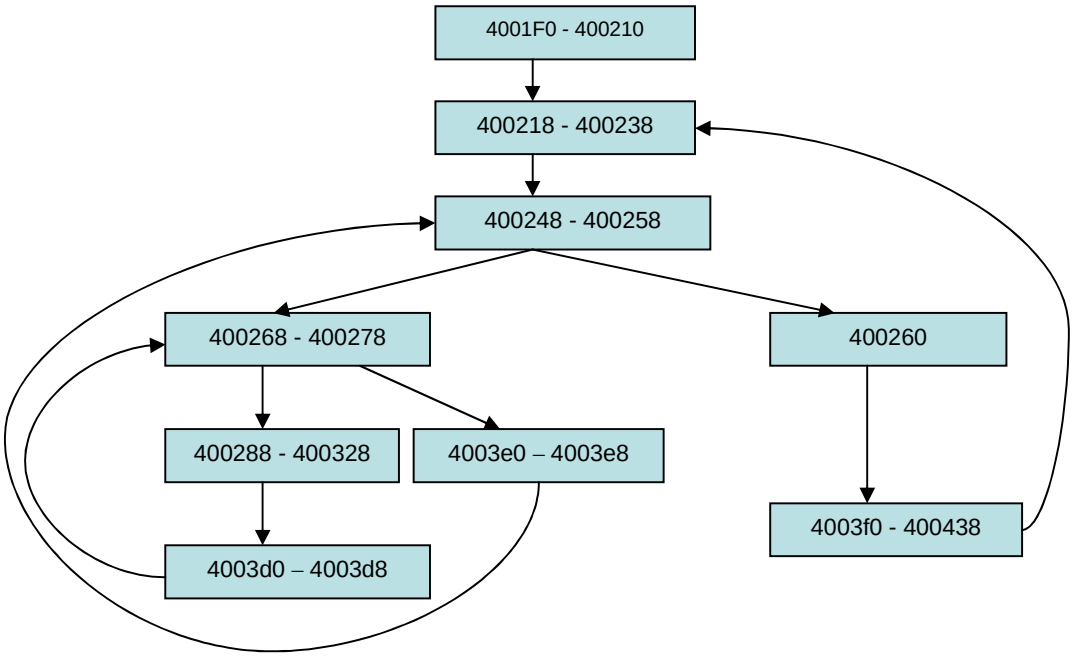
Utilizing 20 us Periodic Scheduler Timer

Week 5

During week 5, I worked on developing the information to provide to the scheduler to determine pc look ups and costs. The Timing Analyzer tool is great for determining overall WCET values and WCET values for functions and loops but it is not well suited for analyzing down to the block level or instruction level. To achieve this; several of the values had to be derived from the verbose timing analyzer by hand. To facilitate this work for the class I decided upon using a partition approach instead of trying to feed every possible PC and it's WCET into the scheduler. To do this I worked on the object dump of each of the tasks used to test this work, CNT, FFT and SRT. Below is a representation of the block contents for the CNT benchmark. These ranges were computed and then compared against the static analysis data to determine the WCET values for PC's occurring within the ranges.

To accomplish this required splitting the blocks into multiple types within the scheduler. In sequential non-loop code the given WCET is sufficient. However within loops the WCET must be calculated depending on the cost of each loop. Also depending on the nesting level the parent's number of iterations must also be calculated and be compared with a per loop cost for the parent. This is part of the annotated information that must be fed into the scheduler to support orthogonal checks.

The last portion of this approach that required modification was loading the structure of the loops into the scheduler. In order to ascertain the number of loop iterations the location of the loop control variable must be known to the scheduler, and further for this implementation of the experiment I required that it be stored in a register value. This worked well for benchmarks such as CNT that contained only rectangular loops and automatically used registers to store the counter variable over the entire lifetimes of both the inner and outer loop. Unfortunately this did not work for FFT a benchmark that contains 4 non-rectangular loops. To facilitate this FFT was analyzed and it was determined that there was available register space that could be used as an internal counter for these loops; unfortunately it just wasn't being used. So at the assembly level it was necessary to add counters for each of the loops that had a stride of 1. This information was loaded into the scheduler to allow it to track the progression of FFT's loops.



Week 6

During week six I continued working on porting the tasks to the scheduler and hand analyzing them for input into the scheduler. I then ran the real-time task set and took random samples to get a better understanding of the timing threshold of this approach.

The table below gives a few measured points from each of the tasks running in the set. As can be seen some of the distances between actual timing and worst case timing are quite far off. A major contributing factor to this is that in FFT and SRT the primary loops are non-rectangular and our tool set tends to over estimate these types of loops. Another contributing factor to the overage is that the polynomial functionality of the timing analyzer was broken so the cost of a single iteration was hand derived and for non-rectangular loops can lead to some inaccuracy.

Benchmark	Simulator Cycle	WCET	Distance	PC
FFT	20000	327145	307145	0x00400380
FFT	40000	389655	349655	0x00400530
FFT	260000	143109	116891	0x004004d8
FFT	340000	1387523	1047523	0x00400810
CNT	24860	87488	62628	0x00400300
CNT	44860	111219	66359	0x00400300
CNT	64860	135039	70179	0x004002d0
CNT	144860	230370	85510	0x004002d0
SRT	122752	377323	254571	0x004005b0
SRT	142752	395989	253237	0x00400558
SRT	162752	414655	251903	0x00400500
SRT	182752	433321	250569	0x004004a8

Future Work and Outstanding Problems

One of the primary complaints of this project is the difficulty in pairing the object code and PC values with WCET. This approach needs to be automated in order to avoid human mistakes.

Another issue that needs to be taken care of is enabling the Timing Analyzer to provide more detailed information about the worst case timing of ranges of code. The way this is currently derived is by analyzing the output of the timing analyzer in verbose mode and matching blocks of code to the output. This tedious and prone to error and this method could be improved through modifications to the timing analyzer.

An on going problem in this implementation is the occurrence of some irregular timing anomalies not caused by attacks. Generally at the end of loops once the counter has been reset occasionally the system will invoke an interrupt to a PC that was contained in the loop. In this particular issue the last iteration value is greater than the current iteration value and being that it is an outermost loop it's infeasible. My inclination is that this is occurring due to an incorrect branch prediction. This issue will need more investigation in order to solve.

Conclusion

During the course of this project two new methodologies for applying security in cyber-physical systems using Timing Data have been created. The first one is a periodic extension of method 2 that enables periodic timing checks to occur in synchronization with instrumented checks throughout the application. This method has tight WCET binding but is structured in such a way that analysis from an attacker may enable this to be beat. The second contribution is a completely orthogonal scheduler based approach that uses PC timing values to validate that the task is running within the constraints. In the early analysis of this approach this work is feasible but the wide timing bounds are cause for further analysis. Future work will need to be performed to determine if there are ways that the data can be structured and organized to reduce inaccuracies due to wide timing margins and human error.