Software Process Control For Secure Program Execution

Computer Measurement Laboratory, Inc.
info@cmlab.biz

The lack of basic principles of engineering design and discipline has led to some very serious problems in the computer security arena. At the heart of modern a nuclear power plant are software systems that provide real time control. It is interesting to note that every aspect of the power plant that is under control of the control system is fully instrumented and the operation of the monitored system is totally under management of the control system. However, the *software* that controls the entire system is essentially open loop and uncontrolled. It would be unthinkable to build a complex hardware system without embedding monitors for process control throughout the whole architecture. Ironically, the backbones of modern control systems are almost entirely dependent on a software system that is running out of control

The lack of monitoring of complex software systems opens the door for problems in both security and reliability. From a security perspective, the software may be compromised and made to perform the bidding of a hostile agent. From a reliability perspective, the software may begin to execute software components that are not thoroughly tested. The end result of either of these two scenarios is the same. The software system will be compromised. What is worse, because the processes that constitute the software system are not being monitored, the integrity of the hardware component is most certainly compromised as well.

The foundation of our current work is the design and development of software control systems. These systems are built to provide engineering integrity for executing software systems. It is our philosophy that software systems must be subject to the same rigorous scrutiny during their operation as would any mission critical or safety critical hardware system.

A most pressing problem looms in the immediate future. In an information warfare scenario, the enemy will deploy an attack that has never been seen before. This attack will completely debilitate both defensive and offensive software capabilities. Unfortunately, the state space of possible attacks is at once large and unknown for a complex software system. If a system is monitored for nominal operation, then it is possible to recognize instantly any abnormal activity. Once the abnormality is recognized, it is then possible to restore the system to a known certified state. Our company's major innovation is a comprehensive approach for tolerating security violations in mission/safety critical software systems by closing the software feedback control loop. To the best of our knowledge, this approach is the first technology that uniformly applies the real-time measurement paradigm to the reliability and security of mission critical software.

In our control theoretic approach to software process monitoring, we are interested in measuring and monitoring the activity of the software system. We have designed a security coprocessor that monitors the running software system and measure its activity while it is running. The central issue here is that once we understand the notion that a program exhibits activity that can be measured, we can leverage this measurement information to assert control on the program execution. We can certify certain activities as nominal and reject activities that are outside of

this range. We can do this in real time because we are monitoring the activity of the system in real time.

In our approach to software process monitoring, we are interested in the activity of the software system. We have designed and developed a security coprocessor that monitors the running software system and measures its activity while it is running. In this context the data may be seen as stimuli for the program that exhibits some activity in response to each datum. Data in the normal range of user activity will induce normal activity on the software. Data outside of this range will induce different or unusual activity on the system that may be readily observed. In this approach the focus shifts from trying to model and understand the data space to which a program may be subjected to the activity of the program in response to the data input.

The principal of software process control should be a major component of computer security. It is also a totally neglected component of computer security. It has long been accepted that data control in the form of encryption is a necessity to preserve the integrity of information flowing from one agency to another. Controlling access to system resources has also shown great value for imposing a security regime.

Access control has been used over time as a means of attaining some modicum of security. In the middle ages, castles were constructed to limit the access of marauding bands of itinerant soldiers to the populace of a region. These castles were effective if and only if they were sufficiently strong. This made them a nightmare for the occupants. The castles were cold, drafty and very restrictive in terms of the movement of their inhabitants. With the advent of the trebuchet and the cannon, even these imposing and uncomfortable structures became obsolete. Access control, the, is a deterrent but not a solution.

Another cornerstone of computer security is encryption. This technology has been with society almost since the inception of the written word. It remains an extremely valuable security tool.

The missing piece of the computer security paradigm is that of process control. It is simply not possible to build systems that are free from vulnerabilities. Engineers have long understood that it is simply not possible to build a defect free bridge or building. Mistakes and human error are a fact of the construction process. Thus, defect free software should never be an objective of software development. Normal users of a system do not exploit vulnerabilities. Only the deliberate misuse of systems will exploit vulnerabilities. This misuse can be detected and acted on immediately, if the systems are being monitored in real time. The problem is not the fact that there are vulnerabilities in the software. The problem is that the software is not monitored and the vulnerabilities open the door for exploits.

In response to the need for a software process control system, we have developed the CML Hardware-enabled Monitoring of Software, HMS, system under the aegis of a contract from the Office of the Secretary of Defense. The primary objective of the HMS project is to create the infrastructure for an autonomic kernel protection system and then productize this infrastructure. There are essentially two types of attacks that can be made on an operating software system: those that have been deployed against existing systems and those that have never been seen before. Current approaches to the kernel defense rely on the knowledge of the attack signature.

That is, there are recognizable symptoms of the effect of the attack on the kernel. Clearly, this approach renders most systems vulnerable to the very type of attack that is most likely in an information warfare scenario.

We have implemented the HMS architecture on a PCI express card for use in Linux based PCs. In its security role, the PCI HMS system is used as a software process controller. It measures the continuing activity of the software running on a host machine and it uses the telemetry that it gathers from this process to compare against a model of standard or certified software activity. At the point that a software system begins to deviate from its standard model of operation, the PCI HMS system may be empowered to alter the execution of the offending process.

Our approach is a dual of the traditional security paradigm. We are able to model the normal activity of a system and detect departures from that normal model. This methodology permits each implementation of a software system to be calibrated for operation in a particular environment. Once a departure from the nominal model of software activity has been detected, the offending activity may be arrested before it has compromised the system. The problem that we have solved is the mathematical representation of normal system activity. The state space of normal activity for large number of mission critical systems is very small and well defined. On the other hand, the state space of abnormal activity is unbounded. It can never really be known.

There are four fundamental components to the HMS system that we have developed to implement this approach. These are the Logic Analyzer, the Analytical Engine, the Policy Engine and the Adaptive Engine. The Logic Analyzer consists of the set of monitor operations that will be performed by the adjunct CPU running on a separate bus architecture. The Analytical Engine will provide a dynamic analysis of the currently executing process to insure that it is executing within certified limits. The Calibration Engine will generate the nominal execution certificate for a process that is to be monitored subsequently by the HMS system. The Policy Engine will manage the rule sets for determining the actions that are to be taken when a process is found to be executing outside the bounds of its nominal operation. The Adaptive Engine will act to alter the execution environment when a process is found to be out of bounds. The action may include, for example, suspending the executing process, termination of an executing process, or modifying an executing process to eliminate suspected malware.

Ultimately each task or process that is to be monitored by the system will carry its own operational certificate. In this guise, as each process is loaded and readied for execution by the operating system, the certificate for its operation will be transferred to the Analytical Engine. When it detects off-nominal activity, the Analytical Engine will return control to the Linux kernel via an interrupt. The corresponding interrupt service will be vectored to the Policy Engine that will, in turn, decide what action is to be taken by the operating system.

In its simplest form, the HMS system is comprised of two central processing systems that are linked with a control interface and also with a shared memory component. All of the processes in the user space run under the aegis of the main CPU. On this side of the system is the Linux kernel and all of the user processes. The analytical engine resides entirely on the second CPU system. There is, however, no operating system on the analytical engine side. There is an element of shared memory that is visible to both of the CPUs. The two systems communicate by

passing information through the shared memory element. There is also a control interface between the two CPU cores.

This technology will enhance and harden the kernel software and other executing software systems from malicious alteration and assaults from the communication network. In that the monitoring functionality is implemented out of band in a hardware co-processor it is simply not possible for an external program activity to access the monitoring system. In that the monitoring function is an external activity, the knowledge of its activity and design will not provide comfort to an enemy.

It is our security thesis that software vulnerabilities are not the fundamental issue. It is very expensive and impractical to try to build a vulnerability free system. No civil engineer would ever expect to build a perfect bridge. An engineer would, instead, expect to build a bridge within practical tolerances and design in compensation for environmental variation. The problem in software, then, is not the vulnerabilities. The problem is that the vulnerabilities get expressed because no one is watching and controlling. We choose to trust no software. We monitor and control everything in the software execution environment.