Trustworthy Computing at APL

Susan C. Lee

ABSTRACT

As dependence on cyber-enabled, networked systems grows, so does the need to make them more trustworthy, especially in the high-stakes adversarial environment of Johns Hopkins University Applied Physics Laboratory (APL) sponsors. This article describes APL's research over nearly two decades to make computing systems more trustworthy—ensuring that they will do what we expect them to do, and nothing else. The basic elements of a trustworthy computing environment—high-assurance systems providing a reference monitor and a separation mechanism—appeared in the literature in the 1970s. When APL began its work, however, few mechanisms to create these environments existed, and even fewer systems incorporated them. True to its legacy, APL did not stop at research into potential mechanisms for trustworthy computing; it also led the way by demonstrating how they could be realized in practical, working systems. This article details APL's development of reference monitors, including an early reference monitor that today is able to detect the activity of a very stealthy threat in executing code. It describes APL's innovative use of existing COTS hardware to provide separation and APL's role in the creation of new hardware in COTS computers specifically intended to provide a high-assurance separation mechanism. Finally, it describes APL's continuing efforts to make high-assurance system development tools both available and usable by APL's engineers, including the application of formal methods, a very potent but nascent software assurance technique, to find life-threatening flaws in real-life critical systems. In the 21st century, APL may create the defining innovations in trustworthy computing that will make the Internet—the defining innovation of the 20th century—safe to use.

INTRODUCTION

The Challenge of Trustworthy Computing

Unprecedented innovation and investment in the last quarter of the 20th century transformed computing from isolated instances of number crunching into an invisible fabric woven of ubiquitous devices and connections that has achieved the status of a dimension beyond physical space and time. Our lifestyle, our critical infra-

Note: This article includes content from the issue "Trustworthy Computing at APL," *Johns Hopkins APL Tech. Dig.* **32**(2), 459–554 (2013).

structure, and even our national security are embedded in cyberspace as much as they are in the more familiar dimensions. As a human-made domain not bound by the laws of physics and too complex for its human creators to fully understand, cyberspace challenges our ability to control and secure it. Fundamentally, we lack the technical capability for building and understanding systems of such complexity. Security researchers, Johns Hopkins University Applied Physics Laboratory (APL) researchers among them, seek to discover the scientific foundations and engineering disciplines of trustworthy computing, allowing us to build systems in cyberspace with no more concern about their reliability and security than we have for brick-and-mortar structures.

Creating Trust

As it relates to computer security, trust can be described as blind faith; we expect a computer system to operate correctly, although we have no proof that it is doing so. If a trusted system fails, its behavior does not meet our expectation, *but we often cannot detect when a system misbehaves*. Most cyber systems used today are considered trusted systems in this sense. It is important to note that a system can be trusted and untrustworthy at the same time. A trustworthy system requires four things: (*i*) a complete definition of correct operation, (*ii*) a monitor to collect evidence of correct operation, (*iii*) a means to ensure that the monitor cannot be corrupted by the system, and (*iv*) guarantees that the monitor is correct and incorruptible.

The challenge of attaining these pieces of evidence can be explained by a thought experiment. Imagine that the operation of system A is monitored and evaluated by system B. If system B reports no incorrect operation by A, can we assume it is safe to trust system A? The answer is yes, but only if we assume that system B is trustworthy. To assess whether system B is operating correctly, we require another system (system C) to monitor and evaluate the operation of system B. If system C does not report incorrect operation of system B, we can trust that system A is operating correctly—but only if we trust system C. Unavoidably, every trustworthy computing system will include at least one *trusted* system; we depend on this system's correct behavior even though we cannot verify it. In logic, this is referred to as an axiom. In the domain of trustworthy computing, this situation is called the root of trust. From the root of trust, we can create a chain of trust, which in our thought experiment is system C→system B→system A.

In our thought experiment, we described system B as verifying the correct operation of system A. More precisely, system B is verifying system A's adherence to a security policy—that is, the definition for correctness. (Note that correctness pertains to operations or behavior, not results; in other words, the program itself may not give the correct answer, but if it faithfully executes the instructions we gave it, the system is operating correctly.) Ideally, a security policy would encompass all the behavior we want and would preclude all behavior we do not. As systems approach the size and complexity of most operating systems and applications, a comprehensive statement of correct operation is outside the realm of today's technology. A significant challenge for trustworthy computing is defining a security policy that captures those aspects of correct operation that indicate that the system is free from malicious manipulation.

For system B to accurately assess the operation of system A, system A must be unable to corrupt the operation of system B. If system A has the means to influence system B, then it is possible that system A can *cause* system B to be unable to discover system A's misbehavior. This is precisely the defect in many current security products: they can be affected by, and sometimes even rely on, the very systems they are observing, allowing the products to be circumvented or subverted. A mechanism for separation, as the concept is called in the field of trustworthy computing, prevents the *monitored* system from corrupting the *monitoring* system. Creating this separation is vital to maintaining the chain of trust.

A flaw of logic in or implementation of the root of trust and the separation mechanism may allow a system to violate the security policy, breaking the chain of trust. By definition, these trusted elements provide no evidence of correctness while they operate. For this reason, we need to verify their correct operation before deployment, through testing or analysis. Many tools can be used to improve confidence that a given system is free of flaw, but the size and complexity of most functional systems overwhelm today's capability to provide *a priori* guarantees. In a true trustworthy system, the root of trust and separation mechanisms must be provably correct. This is the final and most fundamental challenge of trustworthy computing.

A SHORT HISTORY OF TRUSTWORTHY COMPUTING

As early as the 1970s, a Defense Science Board report offered a discouraging view of computer security.[1] Recommendations from that report and other DoD reports described most of the basic concepts for trustworthy computing,[2,3] such as a reference monitor mediating computer program behavior, separation for the reference monitor, and high assurance for correct operation of the monitor and separation mechanism. Over the next quarter-century, a number of government-funded efforts produced requirements for, designs of, and, in a few cases, prototypes of secure operating systems and trustworthy computers.[4,5] But the era of custom trustworthy computing platforms ended in the 1980s with creation of the commodity computer market (i.e., the personal computer); government spending on computers was dwarfed by that of the commercial market, which had far lower expectations for security.

The government reacted to the COTS computing era in the 1980s with the issuance of the "Rainbow Series"[6] of specifications for industry, outlining the requirements for building COTS computers qualified for sensitive government applications. Some industry providers attempted to meet the Rainbow standards, but, eventually, applying these standards proved to be very difficult and time consuming. The evolution of computer and software technology outpaced the ability of providers to produce certified, state-of-the-art computing capability. Ultimately, industry lost interest in supplying certified computers, and the government wanted to leverage inexpensive, state-of-the-art commercial computers. The government's focus shifted to adding security to COTS products.

By the start of the 21st century, hacking had evolved from a teenage amusement into the cyberattack, a serious vehicle for disruption and crime. Computer and software manufacturers began considering, and then providing, commodity products including mechanisms that could form the foundation for a trustworthy computing environment.[7,8] Although the evolution is far from complete, today's computers offer a significant number of options for creating trust. APL has contributed to and taken advantage of some of these in its pursuit of trustworthy computing.

DESIGNING TRUSTWORTHY SYSTEMS AT APL

Research into trustworthy computing began very early in the history of cyber operations at APL. Initially, prompted by tasking from the National Security Agency (NSA), APL made significant advances in defining security policy and monitoring systems. Recognizing that separation mechanisms needed to be rooted in hardware, APL began exploring novel ways to use and improve existing COTS hardware-supported separation for trustworthy computing.

NetTop Failed Operation Recovery and Correction Element

In 2000, NSA asked APL to address the challenge of providing fault-aware capabilities for NetTop,[9] a single system with multiple levels of security. NSA was experimenting with using virtual machines to allow a user to access both unclassified and classified networks from a single physical host, rather than needing a separate physical host for each network; in addition, NSA hoped to create in-line encryptors in software to replace standalone hardware-only implementations.

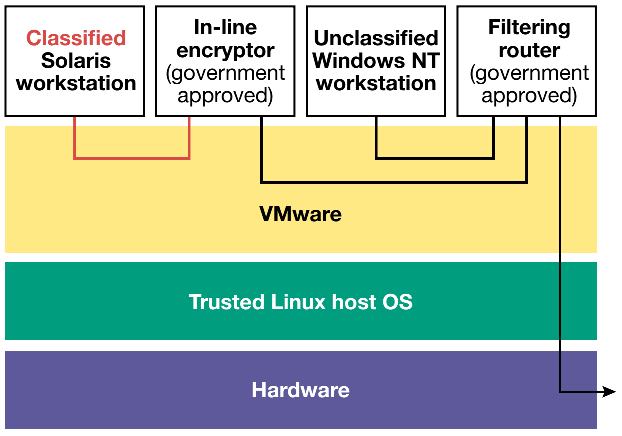

Classified Solaris workstation
In-line encryptor (government approved)
Unclassified Windows NT workstation
Filtering router (government approved)
VMware
Trusted Linux host OS
Hardware

Figure 1. NetTop concept with in-line encryptor.

NetTop was a software system consisting of the NSA Security-Enhanced Linux (SELinux)[10] operating system and VMware, a commercial hypervisor, executing on a standard Intel x86 computer. VMware allowed the simultaneous execution of multiple virtual machines (VMs), while SELinux provided the mandatory access controls needed to ensure that no data flowed between VMs running at different classification levels (see Fig. 1 for a configuration that included an in-line encryptor). Any system enforcing the separation of unclassified and classified networks had to be a trustworthy system. Responding to NSA's need, APL invented one of the first systems for trustworthy computing on a commodity computer, the NetTop Failed Operation Recovery and Correction Element (N-FORCE).[11,12]

The N-FORCE Design

N-FORCE was a software–hardware hybrid that comprised a security policy, an assessment system, and a separation mechanism. N-FORCE had a software component called the N-FORCE Daemon and a hardware component called the N-FORCE Box (see Fig. 2). The Daemon was responsible for periodically examining code and data structures on the system and detecting any errors; the Box, a custom standalone hardware component, was responsible for ensuring that the Daemon was executing when expected.

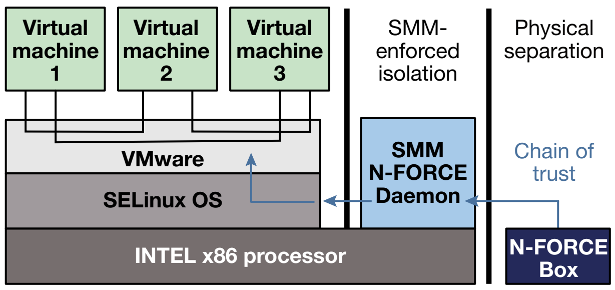


Figure 2. NetTop trustworthy operation with N-FORCE.

To provide separation, the N-FORCE Daemon was executed in a special processor mode, called System Management Mode (SMM), that was available on the commercial x86 processor. The N-FORCE code was loaded into an area of memory called System Management Random Access Memory (SMRAM) at boot time through use of an Option Read-Only Memory (ROM). Option ROMs allow an incorruptible set of instructions to be executed before any potentially corrupted operating system executes. As soon as the N-FORCE code is loaded, write access to SMRAM is disabled, protecting the code from modification by the assessment target (SELinux, VMware, and the VMs). When SMM is triggered, normal execution on the system stops and control is passed to N-FORCE Daemon code, allowing N-FORCE to freely examine the state of the system. Note that not all implementations of the SMM included the memory write-protect feature that was critical to N-FORCE separation.

While the Daemon code was protected from alteration by its location in SMRAM, its execution could be prevented by malicious code running as part of the assessment target. To ensure complete separation of N-FORCE from its assessment target, the N-FORCE Box was designed to check that the Daemon was executing when expected. As a hardware component independent of the x86 platform, the N-FORCE Box formed the root of trust on which the NetTop chain of trust was built. Although very simple in design and construction, and essential to NetTop's trustworthy operation, the N-FORCE Box was never implemented, partly due to reluctance to add custom hardware to a COTS design.

Designing an appropriate security policy to make NetTop trustworthy, especially for the in-line encryption application, was another pioneering task. Over time, NSA had evolved the fail-safe design analysis (FSDA) methodology for assuring hardware encryptors. FSDA involved creating a comprehensive fault tree for the design and determining that no fault resulted in the failure to encrypt the classified data that passed through it. For NetTop, the SELinux/VMware software combination controlled the movement of the data stream; a fault could possibly allow data to flow from the classified side straight to the unclassified network without passing through the encryption VM. Applying FSDA directly to software as large and complex as SELinux and VMware would not be possible. APL analyzed FSDA and defined an equivalent method for software. Application of this modified methodology to NetTop identified the critical code that must have integrity and execute in a predefined manner to demonstrate correct operation.

During execution of the NetTop system, N-FORCE periodically checked the state of the machine, verifying the critical components and timing. To assess the integrity of code and data belonging to critical objects in the system, a cryptographic hash (called the golden hash) was obtained from a reference system prior to NetTop's execution. By hashing a contiguous block of code and static data structures during execution and comparing it to the golden hashes, NetTop confirmed that the critical code and data had not been altered. Checks on the timing between invocations of critical code segments confirmed that they were executing as expected.

The Significance of N-FORCE

N-FORCE was considerably ahead of its time. Even years later, guarantees for the integrity of critical software in commercial systems were limited to comparisons to golden hashes at boot time only, offering no guarantees for the software as it executes. Exploits accomplished while the software was running had free rein until the system was rebooted. Even though it was confined to monitoring structures that were expected to be static during execution, N-FORCE's ability to do so periodically, while the critical software was running, represented a pioneering capability. Even today, a few commercial capabilities provide dynamic monitoring, but none of them have the breadth of the N-FORCE fault coverage.

The Linux Kernel Integrity Measurer

Development for Secure Virtual Platform

The Linux Kernel Integrity Measurer (LKIM) project was a direct follow-on to N-FORCE, funded by NSA as part of the Secure Virtual Platform (SVP) project in 2005. LKIM was developed as a measurement service—that is, it created a representation of the critical structures in memory but relied on a separate attestation service to decide whether the representation was correct. Figure 3a shows a use case for granting access to a resource based on the results of a measurement. LKIM depended on other developments under SVP to provide separation.

LKIM greatly improved the N-FORCE monitoring capability by expanding it beyond hashes of the static portions of the executing code to encompass the dynamic changes that occur as Linux runs. LKIM uses a precise specification of possible behavior extracted from the Linux code before it is executed. A program's code governs how the program structures in memory evolve as it executes. By examining memory periodically during execution, LKIM can detect when its state is not consistent with the code that should be executing; these deviations always indicate that something is amiss, most often the presence of malware. Figure 3b is a graphical representation of the structures LKIM checks, illustrating the complexity of the code and the assessment. Although LKIM is named for its first application, measuring the Linux kernel, the approach is general and has been implemented for other operating systems.

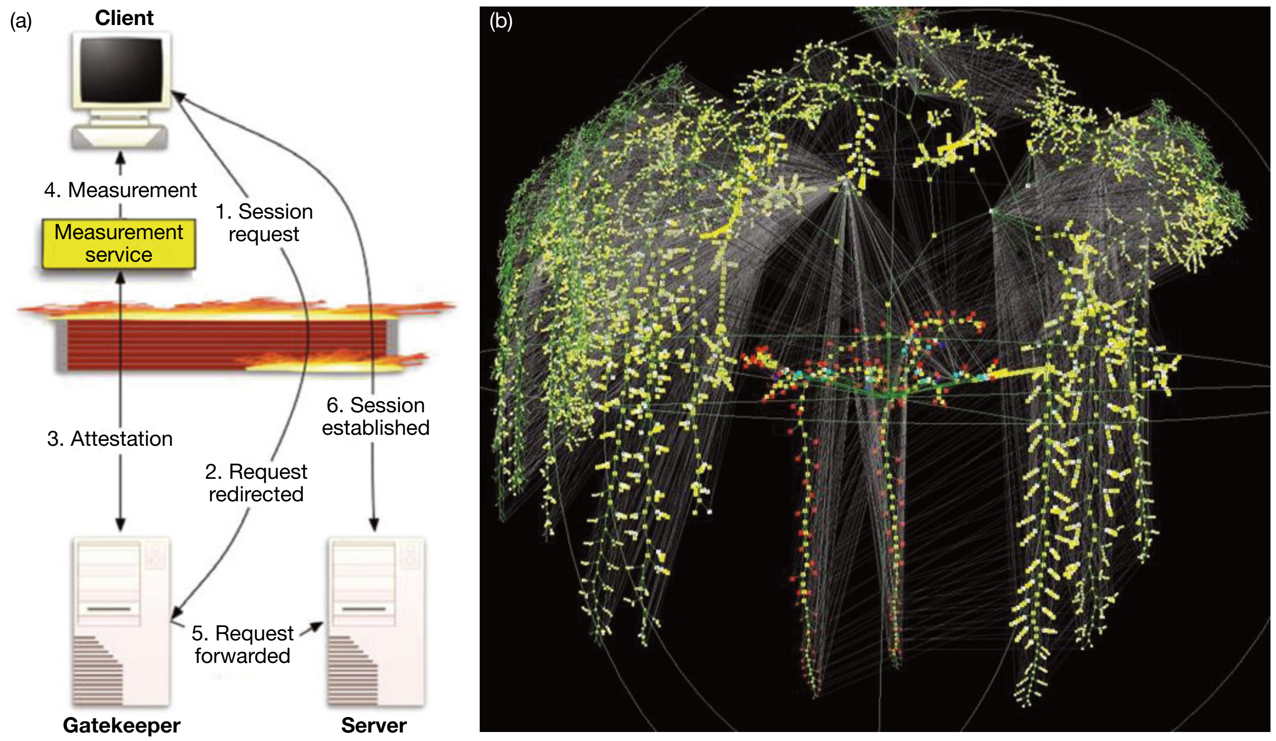


Figure 3. (a) Resource access attestation. (b) LKIM measurement of the Linux kernel visualized as a graph.

The Significance of LKIM

Other malware detection technologies work on signatures, heuristics about how malware behaves, or deviations from previously observed "good" behavior. Products using signatures fail to detect new (so-called zero-day) malware. Because both heuristics and observation-based models are imprecise descriptions of behavior, the tolerance (threshold) for deviations must be high, lowering their probability of detecting malware in favor of avoiding a large number of false alarms. In contrast, the LKIM security policy is based on the logic of the desired code; it will reliably reveal the presence of malware, even previously unknown malware, with a very low rate of false alarm. The LKIM measurement was able to detect deeply buried malware characteristic of the "tier 5 and 6"[13] actors (nation-states) with high confidence.

LKIM Deployment

In 2010, LKIM was deployed on 50+ Linux servers on APL's network. LKIM was incorporated into the standardized server provisioning process. Attestation was orchestrated by a centralized APL-developed attestation service called Maat. To date, the attestation service has not been incorporated into any APL security process.

APL developed Tactical LKIM for use on Navy platforms. For tactical use, a new, faster method of capturing memory snapshots was needed to avoid interfering with the near-real-time operation of tactical systems. In addition, measurements take place on request, rather than periodically, to give the system operators a feeling of greater control and surety. To date, LKIM with the information assurance subsystems has been installed on several naval vessels. In the 2017–2019 time frame, integration with additional information assurance, tactical control, sonar, and imaging servers is planned.

Record and Replay

Ideally, a monitor assessing correct operation would execute continuously, alongside the target, pinpointing the exact moment that the target deviated from correctness, thus minimizing the damage. One way to perform continuous monitoring and dynamic analysis without incurring unacceptable performance impact is to use a technique known as record and replay (RnR). RnR records a program's behavior at speed with minimum overhead; then high-overhead analysis is performed offline, during the target's idle periods, or in another processor, such as another core in the same machine or in another machine in a cloud. This technique was first applied to debugging, where a developer records a trace of a program with a bug and then replays that trace to determine the cause of the bug.[14] More recently, application of RnR to intrusion analysis, detection, and prevention has been suggested.[15–17] APL suspected that many existing and new techniques could leverage RnR to support trustworthy computing, so the Lab created its own RnR system to use in devising and experimenting with novel dynamic analysis applications.

RnR Design

Most computer operations are deterministic. Given an initial state, the final state of a machine is predictable after execution of any uninterrupted sequence of instructions. In a typical program, hundreds of thousands of sequential instructions can be replayed from an initial state. Not all execution is deterministic, however, since unpredictable events occur, such as keyboard input, network traffic, and other external interrupts. These events cause the flow of execution to jump from one set of sequential instructions to another. Because events occur nondeterministically, the complete sequence of instructions is not deterministic either.

To accurately reproduce the execution of a program, a recorder first captures the initial state of the target (e.g., the CPU registers, virtual memory, virtual hard disk, and other devices that are part of the virtual machine) and then records the time of all nondeterministic events that occur, along with any associated data (e.g., the content of a network packet). To replay, a player restores the initial state and executes the original program. The player recreates the nondeterministic events at the appropriate points during the execution (see Fig. 4).

APL implemented an RnR prototype in a VM environment using KVM/QEMU, one of many virtual machine systems. Performance measurements indicated that neither recording overhead nor storage requirements precluded the use of RnR as a monitor in a trustworthy computing application.[18]

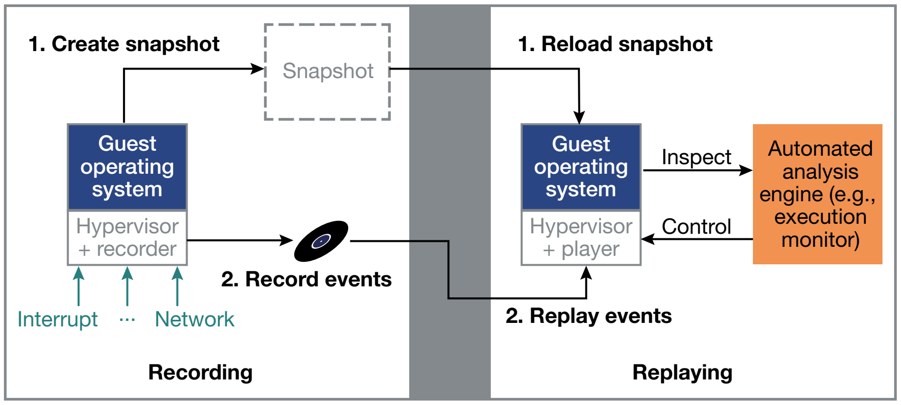


Figure 4. Basic RnR operation.

The Significance of RnR

N-FORCE and LKIM examine periodic snapshots of memory during program execution. While superior to assessment only at boot time, this approach reveals only that something unwanted occurred between snapshots. It cannot determine what occurred or pinpoint when it occurred to a granularity finer than the snapshot interval. An RnR-based continuous monitoring system captures the moment of exploitation, reveals how the exploitation worked, and records exactly what the malware did. In theory, this capability leads to systems that "inoculate" against detected exploits, repair damage, and continue execution without a glitch in real time.

RnR Applications

In practice, RnR has been most frequently used as a forensics tool. Today's malware employs a repertoire of techniques to make ordinary forensic analysis very difficult. APL developed an RnR-based tool called REnigma that was deployed on APL's network in FY2016 and is in daily use. Suspected phishing e-mails and suspected malicious websites are "detonated" and recorded on REnigma. The replay reveals exactly what the malware did, enabling cleanup and future prevention. Another RnR-based forensics tool, called Control-Flow Integrity Monitoring (CFIM), is specifically used to detect telltale signs of a return-oriented programming attack launched from a malicious PDF file. DoD Acquisition, Technology, and Logistics funded development of CFIM and, in FY2016, began funding its transition to operational use.

RnR has also been incorporated into a tool called General Analysis Toolkit Using Record and Replay (GATOR), funded by the Department of Homeland Security Science and Technology Directorate. This tool enables software developers to find, analyze, and fix critical software bugs before their software is deployed. In this context, RnR provides some of the software assurance that is also needed for a trustworthy computing system (see the Assurance of Correct Functionality section).

The next goal is to use RnR as a host-based intrusion detection monitor—similar to Aftersight.[19] As the host executes, the recording of activity is immediately sent to the separate process that analyzes the execution in the background. Analysis could be as extensive as desired through parallelization, sending the recording to multiple analysis systems, with each performing a different technique simultaneously. Continued research might lead to a system that automatically analyzes and recovers from an attack.

Separation

Assured separation between the assessed and assessing systems is crucial to achieving trustworthy computing. APL's use of SMM for separation on N-FORCE was innovative for its time and still holds potential for certain applications. The separation guarantees provided by SMM alone, however, are insufficient without some additional external assurance that the assessment functions performed by code using SMM are being executed. The failure to implement the N-FORCE Box despite its low cost and profile illustrated the hurdles that any noncommercial solution would likely encounter. Concurrent with work on N-FORCE, APL began to explore commercial separation guarantees.

Independent Research and Development

Rockwell Collins claimed that its AAMP7 is a unique processor that is "a separation kernel in hardware." Although not used in commodity computers, it was produced commercially. APL explored the separation claims made for the AAMP7 and examined how it might be applied in tactical systems to enhance their trustworthiness.[19]

APL also embarked on a project, called Trusted Ring, to take better advantage of the Intel memory ring system to provide separation.[20] Intel processors divide memory into four rings, 0, 1, 2, and 3, that have different privilege levels and access to code running in other rings. At that time, the full operating system executed in the most privileged ring, 0. All other code executed in the least privileged ring, 3. APL demonstrated the capability to transparently "lift" an executing operating system from its very privileged location in ring 0 to a less privileged location in ring 1. This freed up ring 0 to provide hardware-enforced separation for security functions, like a monitor for trustworthy computing. Today, commercial operating systems also use the ring system to achieve separation for critical kernel functions.

The Trusted Platform Module

Near the start of this century, major manufacturers, such as Intel and Microsoft, formed a consortium call the Trusted Computing Group (TCG). Driven by e-business security concerns, the TCG created specifications for separation support that can be implemented by processor manufacturers and used by software developers who create sensitive applications. Their foundational specification is the Trusted Platform Module (TPM), which establishes a hardware-enforced root of trust for software in main memory.

APL has contributed significantly to the specification of the TPM through participation in TCG work groups. While at IBM, APL's Dr. David Challener worked on the design of the IBM PC embedded security subsystem and the first TPM chip. APL continued to support his contributions to the TCG after he joined the Lab in 2009. He serves on the board of directors, chairs the TCG Software Stack Work Group, and participates in the Technical Committee, the Virtualization Work Group, and the Storage Work Group. He currently co-chairs the TPM Working Group. Dr. Challener was instrumental to creating the most extensive modification to the TPM specification (version 1.1 to version 2.0) while at APL.[21]

In addition to the main TPM work group, the TCG has work groups for mobile,[22] embedded,[23] and virtualized platforms.[24] Mobile devices pose a challenge for the TCG, since most have space, power, and cost constraints that hinder the adoption of a discrete TPM chip; however, recent mobile technologies, such as the ARM TrustZone technology, have hardware mechanisms to partition the device into secure and nonsecure resources.[25] An APL researcher, Kathleen McGill, cochairs the TCG Mobile Platform Working Group, whose goal is to specify a mobile adaptation of the TPM and a reference architecture.[26]

The Significance of the TPM

Hardware resources to provide guarantees for trustworthy computing are foundational; having these resources on COTS computers opens a whole world of possibilities for designing trustworthy systems. PC manufacturers shipping TPM-enabled PCs include Dell, Lenovo, HP, Toshiba, and Fujitsu. Microsoft has announced that all systems submitting to the Windows Certification Program after January 1, 2015, will be required to include a TPM meeting the TPM 2.0 specification.[27] TPMs have been used in a wide variety of applications, including

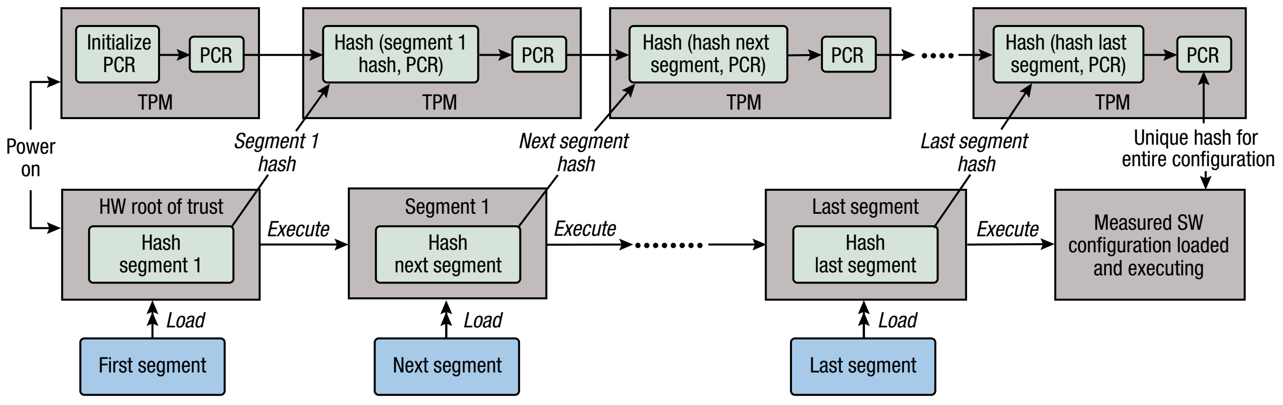


Figure 5. Measured Boot chain of trust.

secure military platforms,[28] secure industrial control systems,[29] and secure electronic voting systems.[30]

The most common use case for the TPM is called Measured Boot. As each part of a software configuration (e.g., OS, applications) is read from the disc, the executable is hashed and sent to the TPM. The TPM concatenates the hash with the value in a Platform Control Register (PCR) and rewrites the result into the PCR ("extends the PCR"). When the Measured Boot completes, the PCR on the TPM contains a cryptographically unique representation of the software on the system that can be used for attestation (see Fig. 5). Even though Measured Boot can provide evidence of only the initial state of the platform, it provides a foundation to build a more comprehensive chain of trust.

ASSURANCE OF CORRECT FUNCTIONALITY

The tenets and technology of trustworthy computing all depend on trusting some system that is not itself subject to dynamic assessment of correct operation. A human could be a root of trust—for example, a system administrator might be trusted to keep a password secret. In that case, a background check on the person may provide assurance. However, in the case of software and hardware roots of trust, we need some way of proving that they will operate correctly without observation of their actual behavior. Analysis and testing are the tools used to provide this assurance. APL has been working to improve the effectiveness and usability of software assurance technology for the past decade.

Formal Methods

Human constructions in the physical world are constrained by the laws of physics. Human constructions in cyberspace are not. Although the hardware portions of a computing system are subject to laws of electromagnetics and the electrical properties of materials, the operation of computing hardware is driven by the arrangement of these materials—an arrangement that in today's chip manufacturing world is controlled by software. There is little loss of generality in the claim that proving the correctness of a cyber system is equivalent to proving the correctness of software, an entity that is not subject to the laws of physics.

Despite its independence from physical constraints, the operation of software is not without limits. In cyberspace, logic takes the place of physics. No matter how unexpected or odd the behavior of software seems, we can be sure it is obeying the laws of logic. While few programmers think they are composing elaborate theorems that can, in principle, be proven to describe some specified behavior, this is exactly what programs are. Formal methods (FM) are the collection of logical constructs and tools that allow us to prove the theorems we write as programs.

FM fall into one of two categories: model checking or deductive verification. Model checkers perform a brute-force exploration of the state space of the system model, proving that it will never enter some user-defined undesirable state or that it will probably reach some user-defined desirable state. Deductive verification proves properties about a program by first describing the valid input states of the program as a logical predicate—that is, a precise, logical statement about the system's state, which can be true or false at any given moment in the system's evolution. Then, by applying inference rules corresponding to each instruction in the program, the starting predicate is transformed into a similar predicate describing the program's final state. This process is similar to traditional mathematical proofs, such as the two-column proofs familiar to many people from high school geometry.

Today, most commonly used programming languages are not amenable to applying logic to prove properties about the programs; further, the programs are far too large and complex. To address these issues, FM are applied to small, but critical, portions of a program that can be cast in a form amenable to transformation through the application of valid rules of logic. Proving correctness for critical operations can greatly increase confidence in the entire program. For example, in one of the first applications of FM at APL, researchers sought to prove that the specification for a field-programmable gate array implementation of the Scalable Configurable Instrument Processor (SCIP) correctly handles stack operation in all cases. SCIP is designed to execute programs written in a specific programming language that performs the majority of logical and arithmetical operations using the stack; thus, correctly handling stack manipulation is essential to correct execution.[31] Since the SCIP was designed for use in satellite-borne scientific instruments, unrecognized flaws could lead to the loss of scientific data obtained at great cost.

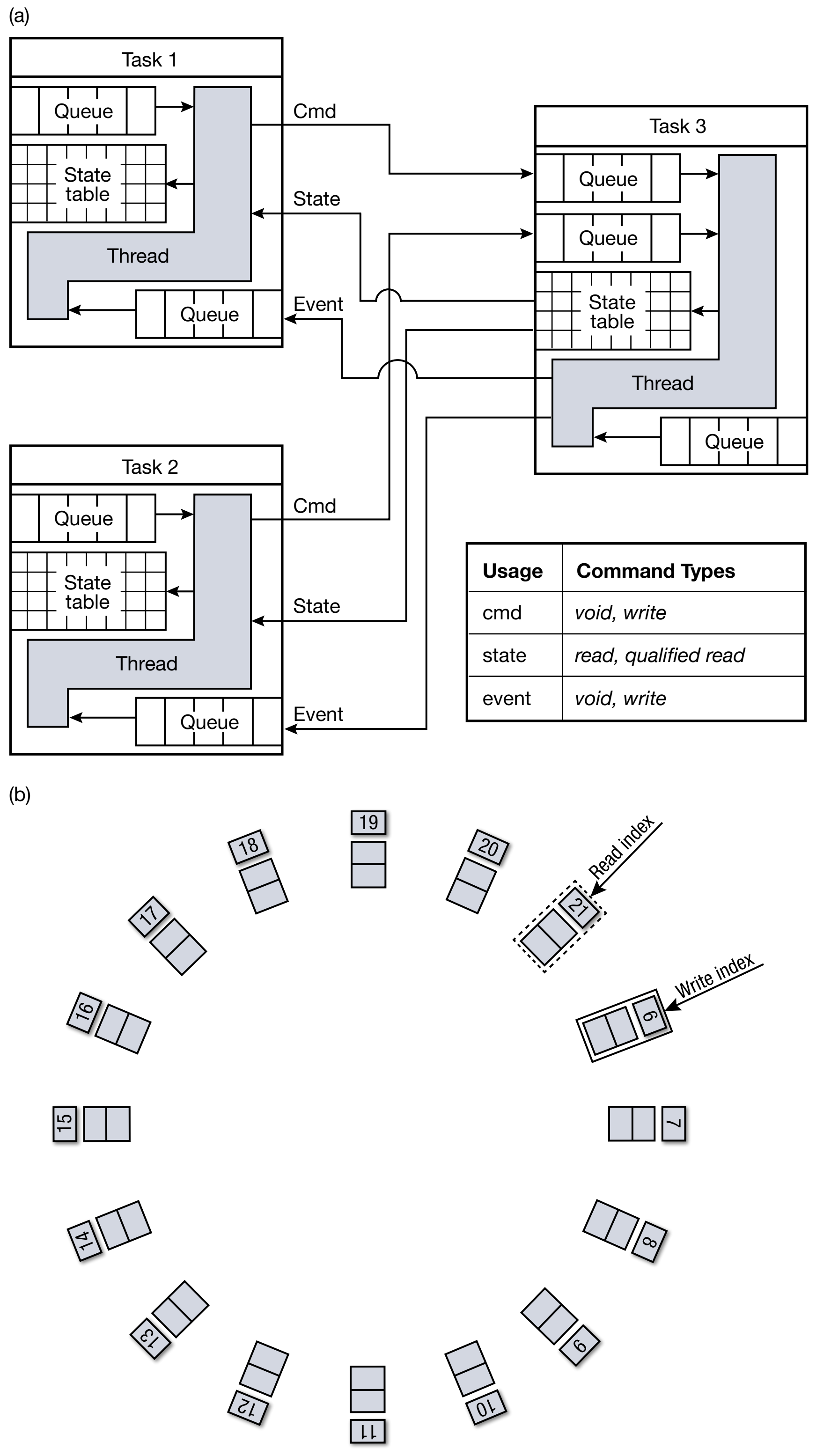
In addition to scaling limitations, FM can suffer from limitations in expressiveness. Expressiveness is the ability of a particular FM to express, or describe, certain aspects of a program's operation. Until recently, existing FM did not express the properties of concurrency (common in real-time programs) or physical–cyber interfaces well. Critical systems such as weapon systems and critical infrastructure control systems employ software that has real-time requirements and interfaces with physical systems. To verify these systems, FM must include some means of expressing the temporal and physical relationship between two (or more) systems operating simultaneously and independently. The rules of logic must be capable of proving that some correct relationship will hold for all inputs, for all times. In principle, no amount of testing can establish the validity of that claim, and, in practice, even large amounts of testing have been found wanting. APL has been leading efforts to move academic research on these expressiveness issues to practice on real-life critical systems.

Real-Time Guarantees

APL's first application of FM to a real-time system used a formal logic called History for Local Rely/Guarantee (HLRG)[32] to verify a key operation in a software framework called the Surgical Assistant Workstation (SAW). (SAW was created by the National Science Foundation Engineering Research Center for Computer Integrated Surgical Systems and Technology, or CISST ERC, at the Johns Hopkins University, in partnership with Intuitive Surgical, Inc., developer of the da Vinci surgical robot.) HLRG uses logical statements, or predicates, to describe the behavior of concurrent software without referencing its specific implementation. Within HLRG, predicates are not confined to the current state of the system but refer to a vector of system states, called a trace, that represents the history of the system's evolution over time.[33,34] Using these predicates, one can assert something like "In the past, address 100 contained some value, but at some subsequent point in time, it contained value 23 at the same time that address 200 contained the original value" and use logical operators to prove it true or false.

SAW is a concurrent system, where different threads of execution can interleave in unexpected patterns, producing unanticipated and sometimes undesirable behavior. From the outset, SAW was developed using a well-defined process and set of tools. In particular, SAW used the CppUnit and PyUnit testing frameworks to implement an automated nightly test suite consisting of more than 1100 tests. Unfortunately, for systems like SAW, testing cannot adequately cover the state space of a real-time system, and the exact conditions of tests are unrepeatable.

APL analyzed a core algorithm that mediates exchange of state information (such as the position and



Usage	Command Types
cmd	*void, write*
state	*read, qualified read*
event	*void, write*

Figure 6. (a) Data structures used in communicating state among concurrent threads. (b) The state table circular buffer with read and write pointers. (Reprinted from Ref. 35.)

velocity of the robot's joints in space) among concurrent threads that use it. The communication mechanism that the SAW uses to share state data among concurrent threads employs no locks or blocking to prevent simultaneous update and use, which would result in the use of an internally inconsistent state vector. Ensuring that all parts of the surgical robot use a consistent picture of the state is essential to its correct operation.

The goal of the formal analysis APL performed was to ensure that the SAW algorithm indeed guarantees that no thread *ever* uses a state vector that is in the process of being rewritten, for all time (see Fig. 6). After casting the effect of the SAW algorithm as HLRG predicates, APL attempted to prove that those logical statements were true. The first proof failed, finding that the SAW algorithm might allow threads to read corrupted state information. APL had found a subtle bug, not by informally examining the system or by testing it, but by carefully modeling it, writing a lemma, and attempting a formal proof. Months of extensive nightly testing had failed to uncover this problem. A corrected version of the algorithm was proven, making this critical element of the SAW trustworthy.[36,37]

Cyber-Physical Guarantees

Hybrid cyber-physical systems also present challenges to FM. A cyber system that operates at discrete time steps must correctly follow and control a physical system that operates continuously. The physical system obeys the laws of physics, and its behavior can often be described by a set of equations. A cyber system can only sense the system state and apply controls to change it at discrete intervals. Engineers have long written code to model a hybrid system and then run the model with particular inputs to see how the system behaves; at best, this approach affords only the equivalent of evaluating individual test cases. It cannot guarantee the exploration of all important corner cases, nor can it always eliminate unexpected behavior.

Proof that the control algorithm predictably influences the physical system requires hybrid logics that are tailored to include both models of discrete programs and the continuous equations that govern the analog components. Powerful FM available today can reason about cyber-physical systems, in the context of both model checking[38] and deductive verification.[39] With them, we can create an accurate model of cyber-physical system components and, subject to the limitations of the model, make guarantees about the system's behavior under all possible input conditions.

The Skull-Base Surgery Robot

APL's first application of FM to a cyber-physical system was for the experimental skull-base surgery (SBS) robot developed by the Johns Hopkins University Computer Integrated Surgical Systems and Technology Group. Its purpose was to help physicians avoid doing unnecessary damage to the brain during surgical procedures on the base of a patient's skull. The SBS works with the surgeon; that is, both the robot and the physician simultaneously hold a surgical tool. The robot senses the force the surgeon puts on the tool and allows it to move according to the equation

$$\frac{d\bar{p}}{dt} = G(\bar{f}),$$

where $\bar{f}$ is the force exerted by the physician, and $\bar{p}$ is the position of the tool in space. This continuous differential equation represents the negative feedback control circuit with an admittance control design.

SBS implements a control algorithm designed to prevent the surgical tool from moving outside a preoperatively defined planar boundary, to prevent unnecessary injury to the brain. Most of the time, the surgeon is free to move the tool around the surgical site, without interference from the robot. In this "free zone," G is some constant multiple of $\bar{f}$. As the tool approaches one of the preoperatively defined boundaries, however, it enters a "slow zone," where the robot is intended to attenuate the component of the tool's velocity toward the boundary. This attenuation increases in proportion to the tool's proximity to the boundary until, at the boundary, the component of the velocity normal to the boundary should go to zero, so that the tool does not progress farther in that direction (see Fig. 7). APL performed a formal verification of the control algorithm performance, to ensure that its design will, in all cases, prevent the tool from moving outside the boundary.

A system model is an integral part of formal verification. Sometimes, just the process of developing and refining a model identifies problems in the algorithm design. These can be corrected even before creating a proof. APL modeled the SBS control system in a differential dynamic logic (d$\mathcal{L}$) language, a language that is not executable but can be formally reasoned about. Applying sound inference rules to the model,[41] d$\mathcal{L}$ can rigorously prove properties about hybrid system behavior for all possible inputs. Initially, the model represented the control algorithm for safety and force feedback at a

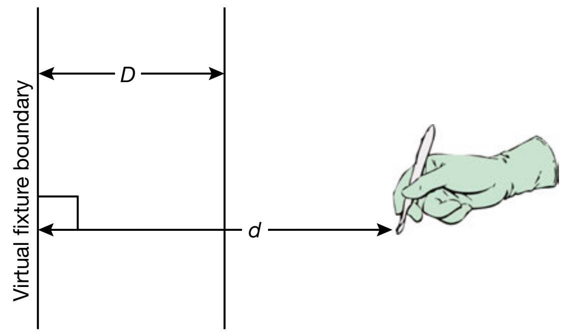


Figure 7. SBS system control regimes. (Reprinted from Ref. 40.)

single boundary; eventually, the model was expanded to control of many boundaries sequentially and produced a combined effect that should ensure safety for a finite collection of boundaries.

APL used KeYmaera, a tool that provides a computer interface for creating models and predicates, and automates some parts of the proof, to perform a formal verification of the SBS control algorithm. APL developed a proof of the safety of the control system algorithm for a single boundary, but counterexamples proved that the algorithm was, in general, unsafe for multiple boundaries. The counterexamples showed specific geometric configurations of multiple boundaries, where the correction the system applied for one boundary could push the tool past another boundary; in addition, for large velocity movements, the process of enforcing one boundary could violate another, sufficiently closely spaced, boundary. Eventually, after corrections were made to the control algorithm, APL was able to prove algorithm correctness; the final proof had 156,024 proof steps.[42]

Next-Generation Airborne Collision Avoidance System

In the 1970s, after a series of midair collisions, the Federal Aviation Administration (FAA) developed an onboard collision avoidance system: the Traffic Alert and Collision Avoidance System (TCAS). Airspace management will evolve significantly over the next decade with the introduction of the next-generation air traffic management system, increased manned air traffic, and the introduction of unmanned aerial vehicles. To meet new requirements for collision avoidance, the FAA decided to develop a new system: the Next-Generation Airborne Collision Avoidance System, known as ACAS X.[43–45]

A typical collision avoidance encounter involves two aircraft: the equipped "ownship" aircraft actively trying to avoid colliding with a nonequipped "intruder" aircraft. Both TCAS and ACAS X avoid such collisions by giving exclusively vertical advisories to the pilot of the ownship, such as "Climb at a rate of 1500 ft/min" or "Do not descend." If both aircraft are equipped with a collision avoidance system, both pilots can receive coordinated advisories. While the design of TCAS was based on geometric considerations on paths taken by aircraft, the design of ACAS X is radically different. ACAS X is designed around a Markov decision process (MDP) estimating the probabilities that two aircraft may climb, descend, or turn in the presence or absence of an advisory.[46] Based on those probabilities, the Markov decision process is optimized to minimize the probability of a collision, while also minimizing extraneous advisories that may distract the pilot. The practical implementation of this algorithm uses a score table, composed of two sub-tables interpolated in succession, one with 698,819 interpolation points and the other with 38,053,125 interpolation points, resulting in a very large number of combinatorial cases.

APL needed to break new ground in FM to provide the FAA with the assurance it sought. First, directly verifying each point in the ACAS X tables is infeasible because of their size. Moreover, ACAS advice cannot always prevent collisions (for example, if two aircraft are flying head-on at the same altitude and are too close to maneuver); thus, the statement describing correct operation cannot simply be "prevents collisions." APL's novel approach first symbolically characterizes safe zones—geometric configurations of the ownship's and the intruder's positions and speeds such that there exists some advisory that, if followed correctly, will not result in a collision. Next, using a hybrid model of the system, combining the differential equations representing continuous motion of the aircraft with the discrete models of advisories, APL formally proved that if the two aircraft are in a safe zone, following the advisory associated with that region will not result in a collision. Last, we exhaustively compared the safe regions to the advisories given by the score table, thus transferring our formal argument of safety to the operational ACAS X.

The results of APL's analysis showed that ~97.5% of the 648,591,384,375 points in the table resulted in a correct advisory (avoided a collision) or represented a case where no advisory would help. The remaining ~2.5% of cases were counterexamples, where following the advisory resulted in a collision that would not have occurred had the pilot remained on the original course (see Fig. 8).

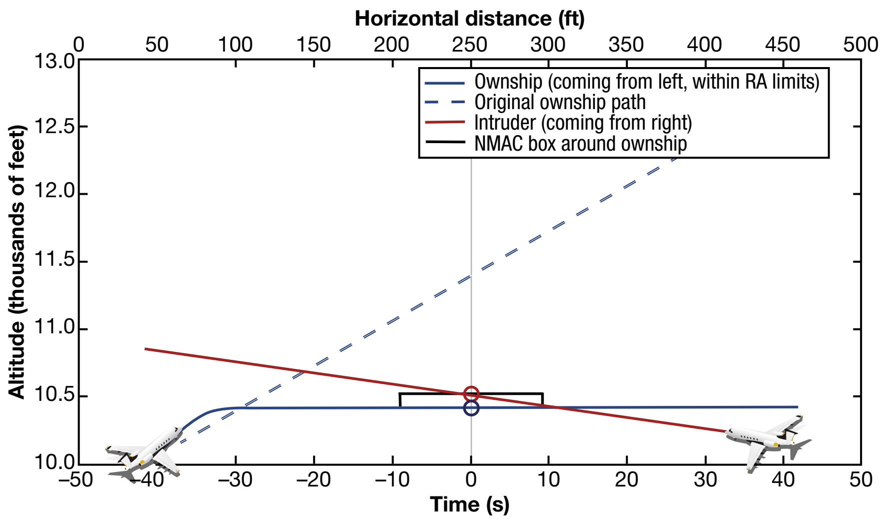


Figure 8. Induced risk counterexample: do not climb advisory puts ownship (blue) on a collision course with intruder (red).

The novel formal verification approach used in this work—modeling, proving, and comparing—is generalizable to other collision avoidance systems or even other systems whose design is based on optimization or machine learning.

The Significance of APL's Work in FM

APL's current work in real-time or cyber-physical guarantees does not specifically address cybersecurity issues; however, similar algorithms have been exploited to create physical damage through cyberattacks. Stuxnet exploited weakness in the protection algorithms of centrifuges to destroy them. Likewise, Idaho National Laboratory has exploited similar weaknesses to destroy a generator like those used in the electric grid.[47] Because APL sponsors are highly concerned with the cyber-physical real-time systems they depend on, APL's pioneering work in applying these new FM to real-world systems positions the Lab to make critical contributions to their future security.

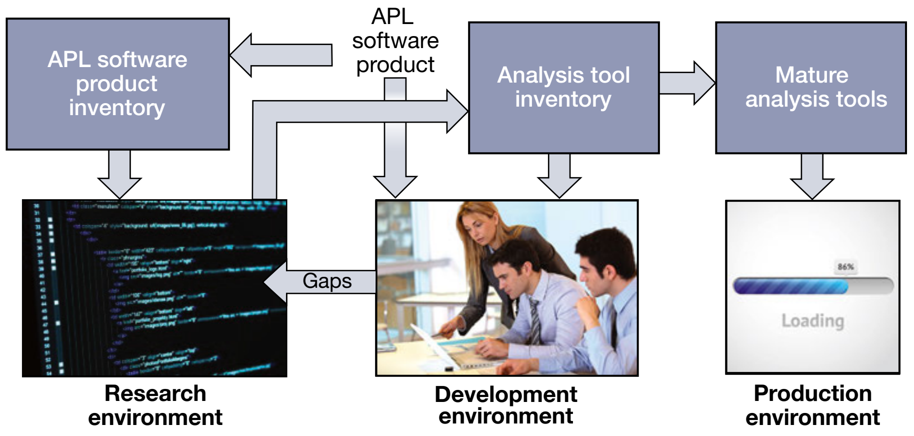


Figure 9. SARA Laboratory concept of operations. (Reprinted from Ref. 48.)

The Founding of SARA

Concept

Because APL delivers software, hardware, and algorithms to its sponsors—either on their own or embedded in systems—it must do its utmost to make sure that they are free from errors, especially those that an adversary could maliciously exploit. To enhance APL's ability to do so, the Asymmetric Operations Sector established the Software Analysis Research and Applications (SARA) Laboratory in late FY2012. The SARA concept of operations is shown in Fig. 9. SARA is envisioned as a place where software, hardware, and algorithm developers across the Laboratory can bring requirements, designs, algorithms, or code for analysis. With this analysis, developers gain insights that improve their products, and Asymmetric Operations Sector researchers learn the capabilities and limitations of commercial and research software analysis tools and advance the state of the art in software analysis. When a tool in the SARA development environment proves to be useful and user-friendly, it can be moved to the production environment, from which the tool and its documentation can be downloaded for use throughout APL.

SARA Lab Operations

Since its inception, the SARA Lab has accumulated a robust inventory of commercial and open-source static analysis tools, implementing a variety of capabilities for analysis, including but not limited to FM. To expand the utility of FM tools, the SARA team has mechanized much of the HLRG logic for real-time guarantees to allow less-experienced users to take advantage of this powerful analysis tool.

The SARA team has worked with development teams across the Laboratory to analyze software as part of both independent research and analysis efforts and sponsored tasks. In addition, the SARA team has experimented with visualization techniques and other improvements (e.g., better descriptions, fusion of results from multiple tools, in-line code views) to create more meaningful presentation of the analysis results.[31] Analysis tools identify a large number of defects, many of which are repetitious or trivial in nature. For example, one SARA analysis resulted in ~10,000 defects, of which ~5,000 were repaired. Closer inspection revealed that most of those defects were the result of just a few errors, and fixing those errors reduced the number of defects by one half. The other ~5,000 defects were the result of a few more errors that were not deemed serious enough to repair. Better automated filtering of analysis results to the relatively few that are important will enhance their effectiveness.

In FY2015 and FY2016, the Laboratory developed a self-service software assurance portal to provide user-friendly baseline analysis capabilities for APL-wide software development efforts. The Self-Service Portal has a front end that allows developers to submit code for analysis and a back end for integrating analysis tools into the portal. By the end of FY2016, users will be able to take advantage of four static analysis tools from the portal. Portal instrumentation will give insight into the impact of specific tools and alerts.

The Significance of SARA

The analysis of LKIM, the integrity checker discussed previously, is a striking example of the SARA Lab's impact. Despite literally years of intense scrutiny and testing of its codebase, SARA analysis revealed a serious flaw that could be exploited to compromise the security of the entire network. This example clearly demonstrates the need for applying assurance techniques to trustworthy computing mechanisms. By creating better tools, and making them readily available to developers, SARA will greatly enhance the trustworthiness of computing systems.

Originally, SARA analysis was regarded with some skepticism by those who thought it was superfluous to testing. As SARA analysts compiled a significant record of important successes, program managers began funding SARA analysis as well as testing for their programs. Today, SARA Lab resources are stretched to meet demand. Rather than simply advising sponsors to use assurance technology for cybersecurity or performing assurance research only when requested by sponsors, APL is actively applying its own cybersecurity principles. This not only assures the best result possible today for Lab sponsors, but it also gives APL a deeper insight into the technology that will lead to important innovations for the future.

LOOKING FORWARD

Work on trustworthy computing is just beginning. Many research challenges remain, not the least of which is recognition of both the need for and the immense potential of trustworthy computing. For example, monitoring software execution for adherence to a rigorous specification of correct operation, rather than for signs of specific misbehavior, is the only means of escaping the menace of the zero-day exploit. Today, extracting the specification of correct behavior from code (as needed for LKIM, for example) requires a considerable degree of manual analysis. Yet research into automating this process takes a backseat to creating high-profile techniques for defeating today's threats even though these techniques quickly become obsolete. Trustworthy computing requires a long-term focus and commitment.

There are positive signs. The groundbreaking work on ACAS X was the first sponsor-funded work at APL using FM after a decade of independently funded research. The work was enabled by a farsighted APL program manager who presented the possibility to an equally farsighted sponsor. Today, APL is assessing the safety and security of the Unmanned System Common Control System (UxS CCS). Because of the complexity and criticality of the software, APL is applying formal model verification to the UxS CCS design. A few years ago, this would not even have been considered. The increasing demand for SARA services is another harbinger of progress, at least on the assurance front.

Today, APL has a history and record of achievement in trustworthy computing of which it can be justly proud. More importantly, APL has the opportunity to build on this foundation to achieve its new centennial vision of creating defining innovations to ensure U.S. preeminence in the 21st century.

REFERENCES

[1]Ware, W. H., *Security Controls for Computer Systems: Report of Defense Science Board Task Force on Computer Security*, RAND Report R-609-1, RAND Corporation, Santa Monica, CA (1970).

[2]Anderson, J. P., *Computer Security Technology Planning Study*, Deputy for Command and Management Systems HQ Electronic Systems Division (AFSC), Bedford, MA (1972).

[3]Bell, D. E., and La Padula, L. J., *Secure Computer Systems: Mathematical Foundations*, Deputy for Command and Management Systems, Bedford, MA (1973).

[4]Sami Saydjari, O., "LOCK: An Historical Perspective," in *Proc. 18th Annual Computer Security Applications Conf.*, Las Vegas, NV, pp. 96–108 (2002).

[5]Neumann, P. G., and Feiertag, R. J., "PSOS Revisited," in *Proc. 19th Annual Computer Security Applications Conf.*, Las Vegas, NV, pp. 208–216 (2003).

[6]*Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, U.S. Department of Defense, Washington, DC (Dec 1985).

[7]Grawrock, D., *Intel Safer Computing Initiative Building Blocks for Trusted Computing*, Intel Press, Santa Clara, CA (2006).

[8]IBM News Room, "Compaq, Hewlett Packard, IBM, Intel, and Microsoft Announce Open Alliance to Build Trust and Security into PCs for e-Business," 11 Oct 1999.

[9]Meushaw, R., and Simard, D., "NetTop: Commercial Technology in High Assurance Applications," *Tech Trend Notes* **9**(4), 1–8 (Fall 2000).

[10]Loscocco, P., and Smalley, S., "Meeting Critical Security Objectives with Security-Enhanced Linux," in *Proc. 2001 Ottawa Linux Symp.*, Ottawa, pp. 1–11 (2001).

[11]Heine, D., and Kouskoulas, Y., *NFORCE Conceptual Design Document: Functional Decomposition and Fault Tree Enumeration*, JHU/APL, Laurel, MD (2003).

[12]Heine, D., and Kouskoulas, Y., *N-Force Daemon Prototype Technical Description*, Technical Report VS-03-21, JHU/APL, Laurel, MD (July 2003).

[13]Defense Science Board Task Force, *Resilient Military Systems and the Advanced Cyber Threat*, Defense Science Board Report, Washington, DC (2013).

[14]LeBlanc, T. J., and Mellor-Crummey, J. M., "Debugging Parallel Programs with Instant Replay," *IEEE Trans. Comput.* **36**(4), 471–482 (1987).

[15]Chow, J., Garfinkel, T., and Chen, P. M., "Decoupling Dynamic Program Analysis from Execution in Virtual Environments," in *Proc. USENIX Technical Conf.*, Boston, MA, pp. 1–14 (2008).

[16]Dunlap, G. W., King, S. T., Cinar, S., Basrai, M. A., and Chen, P. M., "Revirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay," in *Proc. 5th Symp. on Operating Systems Design and Implementation*, Boston, MA, pp. 1–14 (2002).

[17]Elbadawi, K., and Al-Shaer, E., "Timevm: A Framework for Online Intrusion Mitigation and Fast Recovery Using Multi-Time-Lag Traffic Replay," in *Proc. 4th International Symp. on Information, Computer, and Communications Security*, New York, pp. 135–145 (2009).

[18]Grizzard, J. B., and Gardner, R. W., "Analysis of Virtual Machine Record and Replay for Trustworthy Computing," *Johns Hopkins APL Tech. Dig.* **32**(2), 528–535 (2013).

[19]DiRossi, M., *Enhancing Platform and Application Security*, IR&D Completed Project Report, JHU/APL, Laurel, MD (2007).

[20]DiRossi, M., "Towards a High Assurance Secure Computing Platform," in *Proc. 10th IEEE High Assurance Systems Engineering Symp.*, Dallas, TX, pp. 381–382 (2007).

[21]Osborn, J. D., and Challener, D. C., "Trusted Platform Module Evolution," *Johns Hopkins APL Tech. Dig.* **32**(2), 536–543 (2013).

[22]Trusted Computing Group, "Mobile," https://trustedcomputinggroup.org/work-groups/mobile/ (accessed 16 Feb 2017).

[23]Trusted Computing Group, "Embedded Systems Working Group," https://trustedcomputinggroup.org//work-groups/embedded-systems (accessed 16 Feb 2017).

[24]Trusted Computing Group, "Virtualized Platform," https://trustedcomputinggroup.org/work-groups/virtualized-platform/ (accessed 16 Feb 2017).

[25]ARM, Inc., "ARM TrustZone," http://www.arm.com/products/security-on-arm/trustzone (accessed 16 Feb 2017).

[26]McGill, K. N., "Trusted Mobile Devices: Requirements for a Mobile Trusted Platform Module," *Johns Hopkins APL Tech. Dig.* **32**(2), 544–554 (2013).

[27]Microsoft Corporation, Windows Certification Program: Hardware Certification Program, https://msdn.microsoft.com/en-us/library/windows/hardware/jj125187.aspx (accessed 16 Feb 2017).

[28]National Security Agency, *HAP Technology Overview: Trusted Computing Technologies Used in the High Assurance Platform*, National Security Agency/Central Security Service (2011).

[29]User TCGadmin, "Byres Security Demonstrates Industrial Control System (SCADA)," *YouTube*, https://www.youtube.com/watch?v=4UJis7Ud89I (16 July 2009).

[30]Fink, R. A., Sherman, A. T., and Carback, R., "TPM Meets DRE: Reducing the Trust Base for Electronic Voting Using Trusted Platform Modules," *IEEE Trans. Inf. Forensic. Secur.* **4**(4), 628–637 (2009).

[31]Pendergrass, J. A., "Verification of Stack Manipulation in the Scalable Configurable Instrument Processor," *Johns Hopkins APL Tech. Dig.* **32**(2), 465–475 (2013).

[32]Fu, M., Li, Y., Feng, X., Shao, Z., and Zhang, Y., "Reasoning About Optimistic Concurrency Using a Program Logic for History," *Lecture Notes in Computer Science*, Vol. 6269, P. Gastin and F. Laroussinie (eds.), Springer, Berlin/Heidelberg, pp. 388–402 (2010).

[33]Ishtiaq, S. S., and O'Hearn, P. W., "BI as an Assertion Language for Mutable Data Structures," in *Proc. 28th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, New York, pp. 14–26 (2001).

[34]Reynolds, J. C., "Separation Logic: A Logic for Shared Mutable Data Structures," in *Proc. 17th Annual IEEE Symp. on Logic in Computer Science*, Copenhagen, Denmark, p. 55 (2002).

[35]Kouskoulas, Y., and Kazanzides, P., "Applying Mathematical Logic to Create Zero-Defect Software," *Johns Hopkins APL Tech. Dig.* **32**(2), 476–489 (2013).

[36]Kazanzides, P., Kouskoulas, Y., Deguet, A., and Shao, Z., "Proving the Correctness of Concurrent Robot Software," in *Proc. IEEE International Conf. on Robotics and Automation*, St. Paul, MN, pp. 4718–4723 (2012).

[37]Kouskoulas, Y., Fu, M., Shao, Z., and Kazanzides, P., *Certifying the Concurrent State Table Implementation in a Surgical Robotic System (Extended Version)*, Yale University Technical Report, http://flint.cs.yale.edu/flint/publications/statevec-tr.pdf (2011).

[38]Fränzle, M., and Herde, C., "HySAT: An Efficient Proof Engine for Bounded Model Checking of Hybrid Systems," *Form. Method. Sys. Des.* **30**(2), 179–198 (2007).

[39]Platzer, A., "Differential-Algebraic Dynamic Logic for Differential-Algebraic Programs," *J. Log. Comput.* **20**(1), 309–352 (2010).

[40]Kouskoulas, Y., Platzer, A., and Kazanzides, P., "Formal Methods for Robotic System Control Software," *Johns Hopkins APL Tech. Dig.* **32**(2), 490–498 (2013).

[41]Platzer, A., "Differential Dynamic Logic for Hybrid Systems," *J. Autom. Reasoning* **41**(2), 143–189 (2008).

[42]Kouskoulas, Y., Renshaw, D., Platzer, A., and Kazanzides, P., "Certifying the Safe Design of a Virtual Fixture Control Algorithm for a Surgical Robot," in *Proc. 16th International Conf. on Hybrid Systems: Computation and Control*, pp. 263–272 (2013).

[43]Federal Aviation Administration, *Introduction to TCAS II*, Version 7.1, U.S. Department of Transportation, Washington, DC (2011).

[44]Holland, J. E., Kochenderfer, M. J., and Olson, W. A., "Optimizing the Next Generation Collision Avoidance System for Safe, Suitable, and Acceptable Operational Performance," *Air Traffic Cont. Quart.* **21**(3), 275–297 (2014).

[45]Kochenderfer, M. J., Holland, J. E., and Chryssanthacopoulos, J. P., "Next Generation Airborne Collision Avoidance System," *Lincoln Lab. J.* **19**(1), 17–33 (2012).

[46]Kochenderfer, M. J., and Chryssanthacopoulos, J. P., *Robust Airborne Collision Avoidance Through Dynamic Programming*, Technical Report ATC-371, MIT Lincoln Laboratory, Lexington, MA (Jan 2010).

[47]Salmon, D., Zeller, M., Guzman, A., Mynam, V., and Donolo, M., "Mitigating the Aurora Vulnerability with Existing Technology," in *Proc. 6th Georgia Tech Protective Relaying Conf.*, Atlanta, GA, pp. 1–7 (2010).

[48]Pendergrass, J. A., Lee, S. C., and McDonell, C. D., "Theory and Practice of Mechanized Software Analysis," *Johns Hopkins APL Tech. Dig.* **32**(2), 499–508 (2013).

Susan C. Lee, National Security Analysis Department, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Susan Lee is an analyst in the National Security Analysis Department. She received a B.A. in physics at Duke University and an M.S. in computer science and an M.S. in technical management, both at the Whiting School of Engineering at Johns Hopkins University. During her long career at APL, she has contributed to building satellites performing scientific missions, data acquisition and analysis systems to measure the magnetic signatures of SSBNs, and implantable biomedical devices. As chief scientist of the Asymmetric Operations Sector, where the work in this article was performed, she was instrumental in creating, shaping, and maintaining the research program on trustworthy computing. Her e-mail address is sue.lee@jhuapl.edu.