Chapter 2

SECURING CURRENT AND FUTURE PROCESS CONTROL SYSTEMS

Robert Cunningham, Steven Cheung, Martin Fong, Ulf Lindqvist, David Nicol, Ronald Pawlowski, Eric Robinson, William Sanders, Sankalp Singh, Alfonso Valdes, Bradley Woodworth and Michael Zhivich

Abstract Process control systems (PCSs) are instrumental to the safe, reliable and efficient operation of many critical infrastructure components. However, PCSs increasingly employ commodity information technology (IT) elements and are being connected to the Internet. As a result, they have inherited IT cyber risks, threats and attacks that could affect the safe and reliable operation of infrastructure components, adversely affecting human safety and the economy.

This paper focuses on the problem of securing current and future PCSs, and describes tools that automate the task. For current systems, we advocate specifying a policy that restricts control network access and verifying its implementation. We further advocate monitoring the control network to ensure policy implementation and verify that network use matches the design specifications. For future process control networks, we advocate hosting critical PCS software on platforms that tolerate malicious activity and protect PCS processes, and testing software with specialized tools to ensure that certain classes of vulnerabilities are absent prior to shipping.

Keywords: Process control systems, access control, intrusion detection, secure platforms, vulnerability testing

1. Introduction

Process control systems (PCSs) are used in a variety of critical infrastructures, including chemical plants, electrical power generation, transmission and distribution systems, water distribution networks, and waste water treatment plants [3]. Until recently, PCSs were isolated, purpose-built systems that used specialized hardware and proprietary protocols; and they communicated via radio and/or direct serial modem connections without regard to security. However, current PCSs are increasingly adopting standard computer platforms and networking protocols, often encapsulating legacy protocols in Internet protocols (e.g., Modbus encapsulated in TCP running on Windows, Unix or real-time embedded platforms using networks with conventional switches and routers) [4]. The change is driven by the improved functionality and lower cost offered by these technologies and the demand for data to travel over existing corporate networks to provide information to engineers, suppliers, business managers and maintenance personnel [2]. Enterprise IT systems based on conventional hardware, software and networking technologies are vulnerable to myriad attacks. In some cases, PCSs are built using commodity components that are known to have vulnerabilities (e.g., WinCE [16] and QNX [17]) and, unless the vulnerabilities are eliminated or mitigated, the long life of PCS components means that these vulnerabilities will persist in industrial control environments [10].

A recent report [10] predicted that within ten years, "control systems ... will be designed, installed, operated and maintained to survive an intentional cyber assault with no loss of critical function." Achieving this goal will be difficult, but concrete steps must be taken now to improve the security of current and future control systems.

This paper focuses on the problem of securing current and future PCSs, and describes tools that automate the task. For current systems, we advocate adopting techniques from enterprise IT: developing a security policy, limiting access to and from selected hosts, verifying the security policy implementation, and monitoring network use to ensure the policy is met. In particular, we describe two tools. The first tool (APT) verifies that firewall configurations match the specified security policy. The second tool (EMERALD) ensures that network traffic matches policy.

For future control systems, we advocate using secure platforms and automated testing for vulnerabilities in PCS applications. We describe two tools. The first tool (SHARP) monitors applications and subdivides privilege use. The second (DEADBOLT) automates testing for buffer overflows in applications software.

Security in enterprise systems places a higher value on confidentiality and integrity than availability. Frequent patching and system reboots are standard practice. Also, there is typically a fairly low suspicion threshold before administrators take a system offline for forensic examination and remediation. In control systems, on the other hand, availability is the primary security goal. As a result, security components that sacrifice availability will be adopted more cautiously in industrial control environments. We therefore advocate intrusion detection systems rather than intrusion prevention systems for process control networks, and implementation-time testing to prevent certain classes of vulnerabilities instead of employing *post hoc* techniques that sacrifice availability for integrity (e.g., stack canaries [7]). Moreover, it is important to ensure that the defensive mechanisms used in industrial control systems and networks do not themselves become attack vectors.

2. Securing Current Systems

This section describes two strategies for securing current PCSs. The first involves the verification of security policy implementations in PCSs (using APT). The second strategy involves model-based intrusion detection (using EMERALD), which is very effective in control systems because of their regular topologies and connectivity patterns.

2.1 Verifying Access Policy Implementations

The management of the defense of a PCS against cyber attacks is driven by a set of objectives; specifically, what activities the system should or should not allow. For a control network built using current communications technologies such as Ethernet and TCP/IP, the objectives might involve accepting or rejecting traffic at the network layer; accepting or rejecting protocol sessions at the transport layer; and ensuring that application layer resources are used securely. System-wide security objectives are ultimately implemented and enforced by mechanisms that form the first line of defense against an adversary by restricting host and user access to devices, services and files. These access control mechanisms include, but are not limited to:

- Router-based dedicated firewalls (e.g., Cisco PIX series).
- Host-based firewalls, which could be based in software (e.g., iptables [20] in Linux, in-built firewalls in Windows XP, and various products from Symantec and McAfee for Windows) or hardware (e.g., 3Com Embedded Firewall NICs [1]).
- Operating-system-based mechanisms (e.g., discretionary access control in Linux and Windows, and mandatory access control in SELinux [19] or similar functionality provided for Windows systems by the Cisco Security Agent [6]).
- Middleware-based mechanisms (e.g., Java Security Manager) that provide for the specification and enforcement of fine granularity access control policies for Java programs.

The rules imposed by these distributed and layered mechanisms are complex with hard-to-anticipate interactions; thus, the true security posture of a PCS relative to a global policy is difficult to discern. Therefore, it is not surprising that misconfigurations of these mechanisms are a major source of security vulnerabilities. In fact, a recent study suggests that most firewalls suffer from misconfigurations [31].

In an industrial environment it is important to isolate the process control network to ensure proper functioning and system security. Conflicts in the implementation of policy objectives due to the configuration of myriad access control elements can lead to access being denied or traffic being dropped in situations where the opposite would be preferred. Improper configurations can

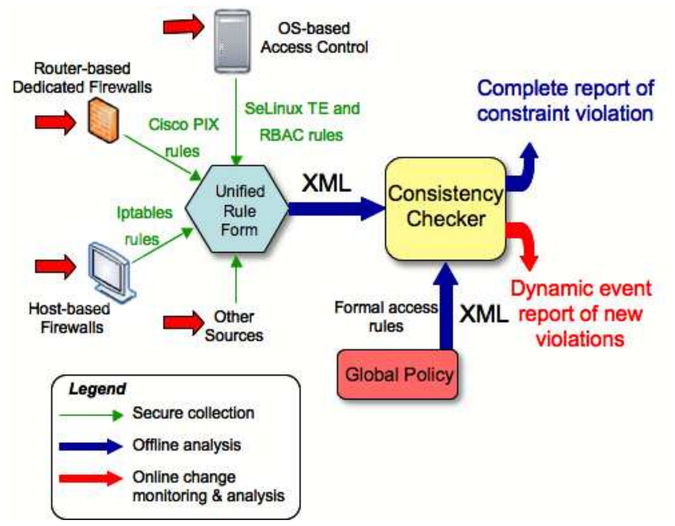

Figure 1. Operational overview of APT.

also lead to access being granted or traffic being accepted in contexts where this is not intended. This type of vulnerability may arise because a network administrator tried to "fix" a conflict that denied desired or convenient access, and by doing so created a hole. It can also arise more subtly from interactions between routing and firewall functions, or routing and operating system functions, among other interactions. Therefore, it is important for security administrators to ensure that high-level specifications of system access constraints are reflected in the configurations of access control mechanisms that are distributed throughout the network, and that changes to the configurations adhere to the global security objectives. In addition to discovering (and possibly quantifying) the deviations between configurations and policy objectives, an accurate and precise diagnosis of the root causes of the deviations would be of great utility.

We have developed the Access Policy Tool (APT) to address the needs described above. APT analyzes the security policy implementation for conformance with the global security policy specification (Figure 1). It captures configuration information from a variety of sources typically found in control networks and conducts a comprehensive offline analysis as well as dynamic online analysis of compliance to ensure that all access control elements work in harmony. The tool includes a graphical front-end to increase usability and provide ease of information management. It complements and supplements other tools and technologies used to secure PCSs. It can ensure the proper configuration of COTS and proprietary components by verification against a specification of intended functionality. In the online mode, APT provides an immediate analysis of configuration changes, and generates alerts for an in-

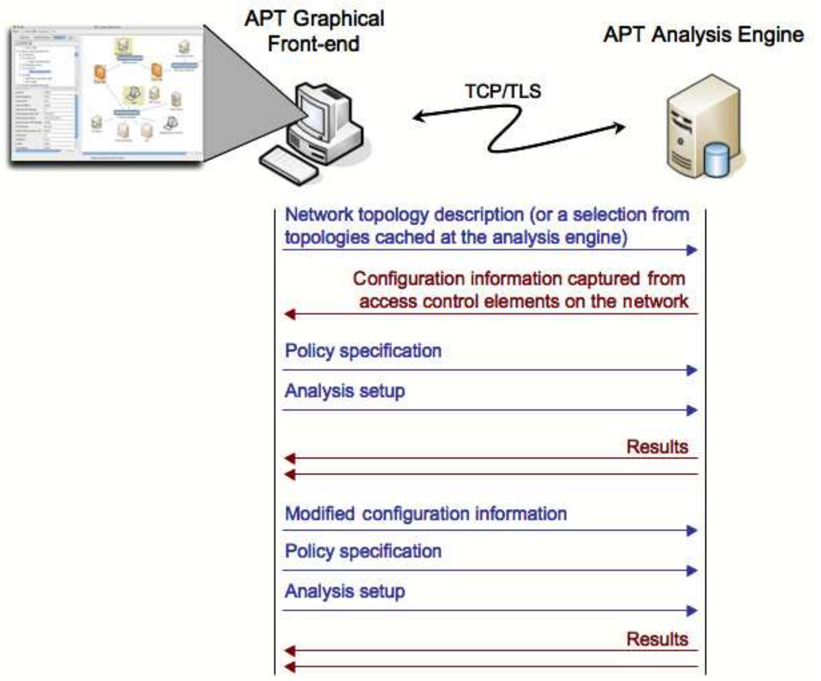
APT Graphical Front-end
APT Analysis Engine
TCP/TLS
Network topology description (or a selection from topologies cached at the analysis engine)
Configuration information captured from access control elements on the network
Policy specification
Analysis setup
Results
Modified configuration information
Policy specification
Analysis setup
Results

Figure 2. APT architecture.

trusion detection or anomaly detection system if any policy holes are detected during system operation.

APT has two components: a graphical front-end written in Java Swing, and the analysis engine written in C++ (Figure 2). The two components can run on separate machines in a network, communicating with each other securely via TCP/TLS. The analysis engine needs to automatically and securely capture the configuration information of various network devices, and hence it should be deployed at a strategic location in the network. The separation of the two components allows for such a deployment, while enabling systems administrators to use the front-end as a management console run from their workstations. Using the front-end, the analysis engine can be configured with basic information about network topology and parameters for secure access to the various constituent devices. Thus, APT can obtain a snapshot of the overall configuration in a secure manner.

The access control mechanisms that APT considers for analysis include, but are not limited to: router-based dedicated firewalls (e.g., Cisco's PIX firewalls); host-based firewalls implemented in software (e.g., iptables in Linux) or hardware (e.g., 3Com's Embedded Firewall NICs); OS-based mechanisms (e.g., NSA's SELinux); and middleware-based mechanisms (e.g., Java Security Manager). The configuration information obtained is translated to an XML schema designed to handle a variety of types of information. Network interconnectivity and data flow between the policy enforcement rules are represented internally using a specialized data structure called a multi-layered rule graph. Each node in the rule graph represents a possible access decision (e.g., a rule in a Cisco PIX firewall that might match incoming traffic), with the paths representing possible sequences of access decisions. The tool can enumerate the possible attributes of the traffic and user classes that traverse a path in the rule graph (i.e., attributes of the traffic that can undergo the sequence of access decisions represented by the path); these attributes are then checked for potential violations against a specification of global access policy.

The tool performs an exhaustive analysis of the rule graph data structure. Possible violations of the specified access policy are enumerated and highlighted in the front-end's graphical display. The algorithms for exhaustive analysis can handle fairly large systems if the analysis is limited to firewall rule sets. However, a PCS might include other access control mechanisms such as SELinux's role-based access control and type enforcement (when other security tools are used in the PCS) or Java Security Manager (when embedded devices are used). The presence of these mechanisms in addition to firewall rule sets produces a massive number of possible interactions between mechanisms that can make exhaustive analysis impossible, even for relatively small PCSs.

To provide scalability in such cases, APT offers a statistical analysis capability that quantitatively characterizes the conformance of the policy implementation to the specification of the global access policy. This is accomplished using a statistical technique known as importance sampling backed by appropriate mathematical constructs for variance reduction [12, 26]. Thus, a fairly accurate estimation of the security posture can be obtained with a limited (and hence, rapid) exploration of the rule graph data structure. The statistical analysis produces a (likely incomplete) sample set of policy violations and quantitative estimates of the remaining violations. The latter includes the total number of violations, average number of rules (or other access decisions) involved in a violation, and the probability that there are no violations given that none were discovered after the analysis was performed for a specified amount of time.

Efficient data structures and algorithms are used to represent and manipulate the traffic attribute sets. These include multi-dimensional interval trees for representing the network traffic component of the attribute sets and custom data structures for efficiently representing the discrete components of attribute sets (e.g., security contexts and object permissions). Intelligent caching of the results of sub-path analysis helps minimize repeated computations.

We have used APT to analyze a variety of PCS configurations, including those for the Sandia testbed (Figure 3), which represents a typical process control network used in the oil and gas sector. The testbed has two dedicated Cisco PIX firewalls with about 15 rules each, which we augmented with host-based firewalls (iptables) for five (of 17) hosts. All the hosts run stock versions of Windows or Linux. The global access constraints were defined using the tool's graphical front-end and were set to emphasize the tightly controlled isolation

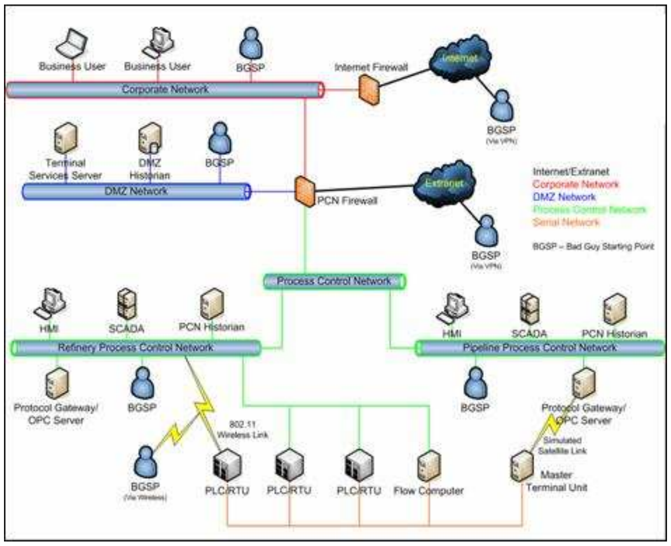

Figure 3. Representative testbed (courtesy Sandia National Laboratories).

of the process control network from the corporate network. In particular, hosts in the process control network could access each other, but only the historian could be accessed from outside the control network. Furthermore, the historian could be accessed by (and only by) a user belonging to the "sysadmin" class via the DMZ historian and by a "manager" class user from a host in the corporate network.

APT discovered more than 80 policy violations in under 10 seconds. Its post-analysis evaluation identified two rules, one in each of the two Cisco firewalls, as the root cause of the problems. The tool generated alert messages indicating the identified rules and suggestions for modification. The indicated rules were then modified or deleted using APT's user interface to quickly produce a configuration that did not result in any violations. Our experimental results indicate that the tool's exhaustive analysis functionality is very useful for analyzing network layer concerns, allowing operators to gain confidence in the implementation and configuration of their control networks.

2.2 Monitoring Process Control Systems

Intrusion detection and prevention are still relatively new to control system applications. Early implementations are typically signature-based applications of enterprise security technologies. We believe that model-based detection, which includes specification-based detection and learning-based anomaly detection, would be very effective in process control networks because they have more regular topologies and connectivity patterns than enterprise networks.

Monitoring a process control network—or any computer network—provides an important orthogonal defense even in networks with well-designed architectures and segmentation. It is essential to confirm continuously that the systems are working properly and that the defenses have not been bypassed.

Firewall configurations are also an issue. It is often the case that a firewall is misconfigured initially, or it may be reconfigured to permit additional connections, or some configurations may be changed erroneously. Indeed, change management is a major concern for control system operators. Monitoring also provides visibility into attempts to breach firewalls. Furthermore, monitoring helps detect attacks that bypass firewall defenses such as those that come in over maintenance channels or exploit PCS component vulnerabilities.

The SRI EMERALD component suite provides a lightweight, cost-effective solution to PCS monitoring and situational awareness. It features the following components detecting attacks, aggregating related alerts, and correlating them in incident reports according to the system mission:

- Stateful protocol analysis of FTP, SMTP and HTTP, with a custom knowledge base and the high-performance P-BEST reasoning engine [15].

- A Bayesian sensor for the probabilistic detection of misuse [29].

- A service discovery and monitoring component coupled to the Bayesian sensor discovers new TCP services (new services in a stable system may be suspicious) and monitors the status of learned services [29].

- A version of the popular Snort IDS [24] with signature set tuned to complement the above, as well as to provide a unique model-based detection capability [5].

- A high-performance, near-sensor alert aggregator combines alerts from heterogeneous sensors based on probabilistic sensor fusion [30].

- Mission-aware correlation (MCORR) that correlates alerts from EMERALD components and other sensors (via an API), and prioritizes security incidents according to user-specified asset/mission criticality [23].

The knowledge base incorporates the Digital Bond SCADA rule set [8] as well as methods developed for Modbus service discovery and model-based detection [5]. Our protocol-based models use the specification of the Modbus protocol, and detect unacceptable Modbus messages by considering, for example, deviations in single-field and dependent-field values and ranges. The model-based approach also detects deviations from expected communication patterns and roles in the control network. This approach is based on the hypothesis that specification-based—or more generally model-based—detection, which is difficult in enterprise networks due to the cost and complexity of encoding accurate models, can be effective in control networks because of their relatively static topology, small number of simple protocols and regular communication patterns. Moreover, model-based detection can detect new (zero day) attacks because it does not depend on attack signatures.

EMERALD requires an appliance configuration, where the monitoring interface is passive and connected to a span port on a switch or router. The reporting interface is ideally on a private network dedicated to security functions. As such, the appliance itself is invisible on the monitored network. Depending on performance requirements, the recommended deployment is to house the detection components on multiple platforms and MCORR on another, with the detection platforms reporting to the MCORR platform. If network traffic is light, all the components can be installed on one platform.

Users of this or any other integrated PCS security monitoring system benefit from the ability to detect a variety of attacks and suspicious events. The system alerts users to probes crossing network boundaries (e.g., DMZ and control system probes from the Internet by way of the corporate network or control system probes from a compromised node on the DMZ), known exploits against commodity platforms and operating systems, appearance of new Modbus function codes in a stable system, and unexpected communication patterns.

The performance of the system was validated via experiments on Sandia's PCS testbed (Figure 3). Also, SNL developed a multi-step attack scenario in which an adversary first compromised a system on the corporate network. From there, the attacker gained access to the DMZ historian server and subsequently accessed a historian in the process control network. The attacker then performed reconnaissance before attacking Modbus servers and other hosts in the process control network. During an attack run, `tcpdump` traces were collected, which were later used to validate the sensors.

The experimental results provide evidence that the model-based intrusion detection approach is effective for monitoring process control networks, and that it complements the signature-based approach. Different sensors detected different aspects of the multi-step attack scenario. Specifically, Snort and EMERALD's Bayesan sensor detected network scans. The signature-based rules developed by Digital Bond detected events involving an unauthorized host sending read and write attempts to a Modbus server. The system also generated Modbus server/service discovery messages during a Modbus attack. The protocol-level rules detected invalid Modbus requests (e.g., Modbus requests containing unsupported function codes). Finally, the communication pattern rules generated alerts for attack steps that violated the expected communication patterns.

3. Securing Future Systems

This section describes two strategies for securing PCSs of the future. The first, involving SHARP, provides security for PCS platforms. The second, involving DEADBOLT, is intended assist vendors in eliminating buffer overflows in PCS software.

3.1 Securing Control System Platforms

PCSs often run on commodity computers that are susceptible to attack by malicious insiders and outsiders. None of the common operating systems were

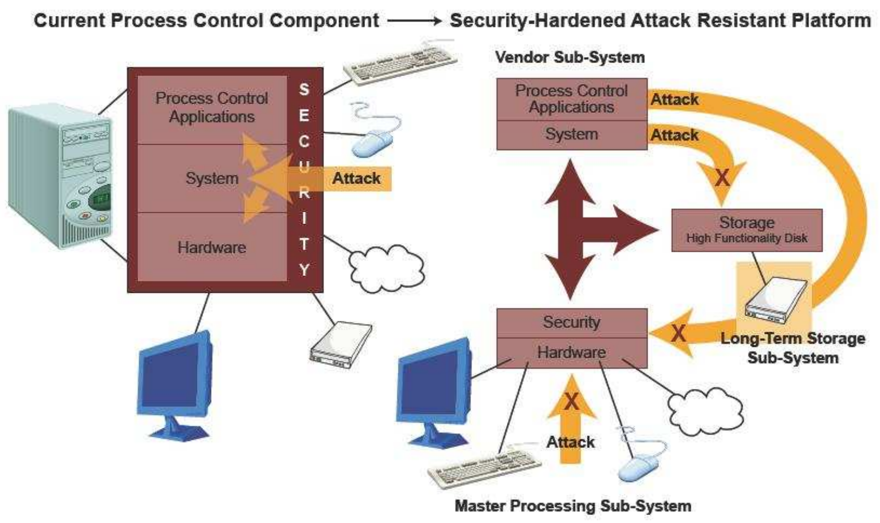
Current Process Control Component → Security-Hardened Attack Resistant Platform
Process Control Applications
System
Hardware
SECURITY
Attack
Vendor Sub-System
Process Control Applications
System
Attack
Attack
X
Storage
High Functionality Disk
Security
Hardware
X
Long-Term Storage Sub-System
X
Attack
Master Processing Sub-System

Figure 4. Vulnerable PCS system (left); SHARP platform (right).

designed with security as a primary requirement [28]. Experience has shown that securing such operating systems requires high levels of skill and is a never ending process involving configuration changes, software patches and new layers of security software. The result is uncertain and fragile security coupled with higher costs and reduced productivity. Furthermore, security is commonly achieved by limiting network connections and restricting applications to a minimal audited set of services. These security measures result in limited functionality that can inhibit or compromise the primary function of a PCS. The Security-Hardened Attack-Resistant Platform (SHARP) concept was created to address this issue: it provides increased security PCSs without limiting the functionality of PCS software.

SHARP is a hardware and software architecture designed to reliably detect and respond to unauthorized physical and/or network access by malicious users and software. By design, SHARP employs publicly available or commercial computing platforms that use proven security methodologies. For example, SHARP uses minimized hardened operating systems [21] to reduce the number of attack vectors and separation of duty via different roles to mitigate risk. Additionally, it uses an independent security supervisor that runs on separate hardware [27]. This supervisor constantly assesses the security state of the PCS, monitors network traffic and data access, and constrains user activities. SHARP also uses system partitioning to better secure the PCS. It places high-value, harder-to-secure systems behind a high-performance security system that provides better protection and monitoring capabilities (Figure 4).

SHARP partitions a PCS system into three parts: vendor subsystem (VSS), long-term storage subsystem (LSS) and master processing subsystem (MPS).

VSSs are high-value legacy master terminal units, human machine interfaces and other systems that are needed for operations. As the left-hand side of Figure 4 illustrates, an adversary who gains access to a classical PCS can obtain control of the entire system. However, as shown in the right-hand side of Figure 4, an adversary who successfully attacks a VSS will have difficulty accessing the other partitions used by the PCS application. All VSS inputs and outputs are proxied and monitored by the MPS (discussed below). However, the VSS also provides some security services, including authentication, authorization and integrity checking as dictated by the underlying software and associated policies. These measures are independent of the security services provided by the MPS.

The LSS partitions and secures all SHARP data that requires persistence. It runs separately from other system components and is secured by permitting access only through special protocols. Also, it can provide encrypted storage of persistent data to reduce insider threats.

The MPS is the core of SHARP, serving as an intelligent high security data traffic controller. It runs without user interaction to mitigate vulnerabilities and protect against internal and external threats. The MPS also provides validation of itself and other system components; this enhances attack detection. It boots only from read-only media so that any detected coercion is remedied by a restart from a known good state. Furthermore, it initializes cryptographic functions and communications to combat insider threats.

The MPS uses three major technologies to provide security services:

- **File Monitor:** This detector monitors all proxied data I/O between a VSS and LSS. It looks for policy violations and responds by interrupting or reversing the activity, alerting operations or implementing other appropriate responses according to policy.

- **Network Monitor:** This detector monitor the ingress and egress sides of the MPS network stack. It adaptively responds to network-based denial of service attacks using a decision engine that implements a security policy.

- **Memory Monitor:** This detector monitors process images in memory for unexpected changes. The system can respond to these changes by restarting the process from a CD-ROM image.

The use of these technologies enhances PCS security and, therefore, availability by detecting specific malicious activities and responding to them based on preset policies. For example, if a denial of service attack on a vendor PCS application host is perpetrated by flooding the network interface, the network monitor on the MPS will give priority to known communications and block other (lower priority) packets that are received in high volume.

SHARP is very successful at limiting unauthorized changes to PCS settings, which can occur after a successful attack on a VSS. It continues to protect the PCS even when other measures to control unauthorized access to the VSS have failed; this increases system availability. The physical partitioning of computing resources within SHARP provides these security services with minimal impact to PCS applications. This enhances the availability of existing systems that would otherwise only be achieved by investing in new platforms to support the functional PCS infrastructure.

The risk to plant operations increases when a physical failure is coupled with a simultaneous control system failure [22]. SHARP is designed to increase the availability of the control system in the face of deliberate attacks and unanticipated failures without the additional cost of a major upgrade to the process control infrastructure. Moreover, the additional effort required to attack SHARP would produce a stronger attack signature that could be detected more easily by other network security mechanisms. Our future work will focus on developing a memory monitor for the MPS, and implementing SHARP as a plug-in appliance for existing PCSs.

3.2 Securing Control System Applications

Buffer overflows are among the most common errors that affect the security of open source and commercial software. According to the National Vulnerability Database, buffer overflows constituted 22% of "high-severity" vulnerabilities in 2005 [18]. Techniques to prevent attackers from exploiting these errors can be divided into those that discover the vulnerability before software is deployed (and thus enable the developer to fix the core problem) and others that make the successful exploitation of deployed vulnerabilities more difficult by creating additional barriers at runtime.

Current manual techniques for finding errors in software (e.g., code review by expert programmers) are expensive, slow and error-prone. Automated review by static analysis systems is also prone to very high false negative and false positive rates [33]. Code instrumentation (e.g., fine-grained bounds checking [25] or program shepherding [13]) can detect runtime problems when they occur, but it usually slows execution, sometimes significantly. Faster techniques that catch fewer attacks (e.g., address space layout randomization and using canaries to detect overflows [7]) can be implemented and supported by the operating system; in fact, both of these are used in modern enterprise operating systems (e.g., Microsoft Vista and Linux). All these techniques raise exceptions that translate programming errors into denials of service, which is an acceptable strategy for many enterprise systems and applications. However, such a defensive response is unsuitable for PCSs, where data must be reliably collected and control must be maintained.

A better solution is to discover and fix errors before deployment; this eliminates the vulnerability entirely and prevents denials of service due to foiled attacks. Such an approach is also cheaper: it costs thirteen times less to fix software during development than after deployment [14]. Furthermore, if vulnerabilities can be reliably fixed at implementation time, solutions that require even modest amounts of additional memory (e.g., [7]) or hardware support (i.e., the per-page no-execute bit) would not be required. Even modest gains such

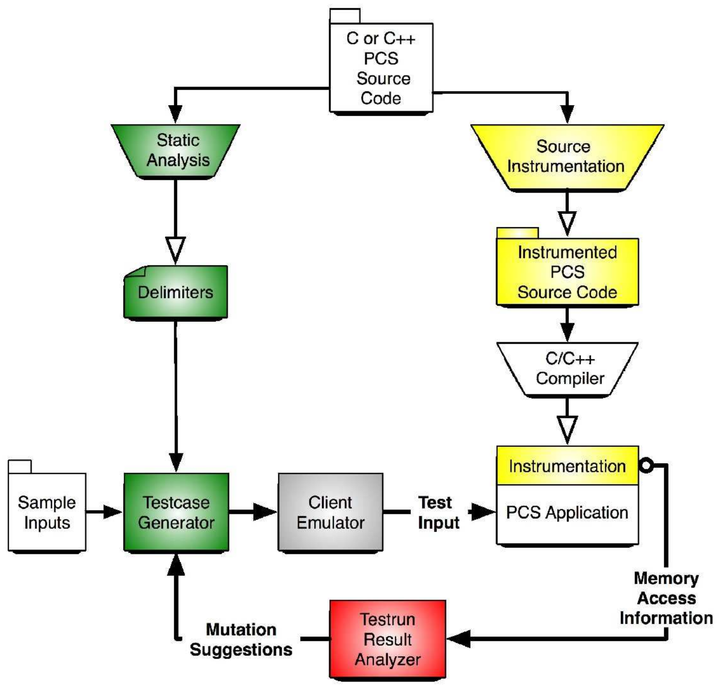

Figure 5. DEADBOLT architecture.

as these are important in embedded systems, where memory is at a premium and hardware support is limited.

To accomplish these objectives, we have developed DEADBOLT, a developer environment for automated buffer overflow testing (Figure 5). DEADBOLT uses source code instrumentation and adaptive testcase synthesis to discover buffer overflows automatically. Code instrumentation in the form of a bounds-checking runtime library enables detection of overflows as small as one byte beyond an allocated buffer [25]. However, fine-grained buffer overflow detection alone is insufficient to identify buffer overflows—a way to trigger the overflow is also required. This task is performed by adaptive testcase synthesis that mutates sample valid inputs into ones that cause buffer overflows.

Code instrumentation (shown in the right half of Figure 5) is implemented as a source-to-source translator that parses the original C++ code and inserts calls to a library that monitors memory accesses at runtime. The library keeps track of all allocated memory objects (regardless of whether they were explicitly allocated by the programmer) in a splay tree that enables determination of a "referent object" (memory object referenced by a pointer) whenever a pointer operation is performed. Thus, all memory operations using a pointer can be checked against the size of "referent object" with the violations being reported as errors. In addition, the bounds-checking library keeps track of memory access statistics for each memory object, including the maximum and minimum number of accesses over its lifetime. The resulting instrumented source code is compiled using a standard or platform-specific C/C++ compiler and linked to the bounds-checking library. This process produces an executable that not only detects when an overflow has occurred, but also provides valuable information about memory accesses during normal program operation.

Adaptive testcase synthesis (shown in the left half and bottom of Figure 5) provides an automated method for discovering buffer overflows in instrumented executables. Testcases are generated by mutating sample valid inputs, and information produced by the runtime bounds-checking library during program execution directs further testcase generation to overflow buffers that are considered potentially vulnerable. Adaptive testcase synthesis improves on existing automated testing approaches (e.g., random testing) by automatically generating testcases from sample inputs without a grammar, adapting mutations based on the results of previous program executions, and providing detailed information about overflow locations in source code. A prototype implementation of DEADBOLT was able to discover five buffer overflows in a C implementation of Jabber instant messaging server using a simple input mutation strategy and only a handful of testcases [32].

Achieving fine-grained buffer overflow detection and providing detailed information about memory access patterns needed for adaptive testcase synthesis comes at a cost. Adding instrumentation that monitors memory accesses results in potentially significant performance slowdowns when running instrumented executables; this is because all pointer operations are converted to calls to the bounds-checking library, and additional calls are inserted to keep track of memory object creation and deletion. This additional code also increases the memory footprint of the instrumented executable, which may become too large to run on an embedded device with limited memory. In such cases, testing can be conducted using an emulator on the developer's desktop that has sufficient memory and computing power. We believe that these tradeoffs are acceptable, as the application performance and memory footprint are only affected during testing—once errors are found and fixed, the application may be compiled and shipped without any instrumentation.

We anticipated that the relatively small scale of PCS firmware and software would facilitate the reliable detection of vulnerabilities at implementation time. After discussions with several vendors, we discovered that many PCSs are implemented using the entire C++ language, not the simplified Embedded C++ [11]. Therefore, our solution developed for software written in a limited form of C++ will not protect a significant portion of PCS software.

To extend DEADBOLT to support applications that use advanced C++ features such as templates and exception handling, we are employing a C++ front-end from Edison Design Group [9] to transform C++ constructs to C intermediate language. This requires simpler instrumentation while retaining information about the original source code that is necessary for accurate and meaningful error messages. We are also focusing on input mutation strategies and test framework modifications that will enable DEADBOLT to provide effective buffer overflow discovery for PCS applications. Once modifications and further testing are complete, we hope to make the tool available to PCS vendors to assist them in eliminating buffer overflows when developing applications software.

4. Conclusions

Unlike their enterprise network counterparts, industrial control networks emphasize availability over confidentiality and integrity. This requires techniques and tools designed for enterprise networks to be adapted for use in industrial control environments. Alternatively, new techniques and tools must be developed specifically for control environments. Our strategy for securing current and future process control systems and networks is to leverage existing enterprise security solutions as well as research results from the broader discipline of information assurance. This strategy has led the development of an innovative suite of tools: APT, a tool for verifying access policy implementations; EMERALD, a model-based intrusion detection system; SHARP, a tool that mitigates risk by partitioning security services on tightly controlled hardware platforms; and DEADBOLT, a tool for eliminating buffer overflows during software development. The security requirements for industrial control environments differ from those for enterprise networks. Nevertheless, effective security solutions can be developed for process control networks by considering the special security requirements of industrial control environments, and carefully selecting, adapting and integrating appropriate enterprise security techniques and tools.

Acknowledgements

This work was partially supported under Award 2003-TK-TX-0003 from the Science and Technology Directorate of the U.S. Department of Homeland Security, and by Air Force Contract FA8721-05-C-0002.

References

[1] 3Com Corporation, 3Com embedded firewall solution (www.3com.com /other/pdfs/products/en_US/400741.pdf), 2006.

[2] T. Aubuchon, I. Susanto and B. Peterson, Oil and gas industry partnership with government to improve cyber security, presented at the *SPE International Oil and Gas Conference*, 2006.

[3] S. Boyer, *SCADA: Supervisory Control and Data Acquisition*, Instrumentation, Systems and Automation Society, Research Triangle Park, North Carolina, 2004.

[4] E. Byres, J. Carter, A. Elramly and D. Hoffman, Worlds in collision: Ethernet on the plant floor, *Proceedings of the ISA Emerging Technologies Conference*, 2002.

[5] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner and A. Valdes, Using model-based intrusion detection for SCADA networks, presented at the *SCADA Security Scientific Syposium*, 2007.

[6] Cisco Systems, Cisco security agent (www.cisco.com/en/US/products/sw/secursw/ps5057/index.html), 2006.

[7] C. Cowan, C. Pu, D. Maier, H. Hinton, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle and Q. Zhang, StackGuard: Automatic adaptive detection and prevention of buffer overflow attacks, *Proceedings of the Seventh USENIX Security Symposium*, pp. 63–78, 1998.

[8] Digital Bond, SCADA IDS signatures (digitalbond.com/index.php/category/scada-ids), 2005.

[9] Edison Design Group, C++ front end (www.edg.com/index.php?location=c_frontend), 2006.

[10] J. Eisenhauer, P. Donnelly, M. Ellis and M. O'Brien, Roadmap to Secure Control Systems in the Energy Sector, Energetics, Columbia, Maryland, 2006.

[11] Embedded C++ Technical Committee, The embedded C++ specification (www.caravan.net/ec2plus/spec.html), 2006.

[12] P. Heidelberger, Fast simulation of rare events in queueing and reliability models, *ACM Transactions on Modeling and Computer Simulations*, vol. 5(1), pp. 43–85, 1995.

[13] V. Kiriansky, D. Bruening and S. Amarasinghe, Secure execution via program shepherding, *Proceedings of the Eleventh USENIX Security Symposium*, pp. 191–206, 2002.

[14] R. Lindner, Software development at a Baldridge winner: IBM Rochester, presented at the *Total Quality Management for Software Conference*, 1991.

[15] U. Lindqvist and P. Porras, Detecting computer and network misuse through the production-based expert system toolset (P-BEST), *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 146–161, 1999.

[16] National Institute of Standards and Technology, CVE-2004-0775: Buffer overflow in WIDCOMM Bluetooth Connectivity Software (nvd.nist.gov/nvd.cfm?cvename=CVE-2004-0775), 2005.

[17] National Institute of Standards and Technology, CVE-2004-1390: Multiple buffer overflows in the PPPoE daemon (nvd.nist.gov/nvd.cfm?cvename=CVE-2004-1390), 2005.

[18] National Institute of Standards and Technology, National Vulnerability Database Version 2.0 (nvd.nist.gov), 2007.

[19] National Security Agency, Security-enhanced Linux (www.nsa.gov/selinux/index.cfm).

[20] netfilter.org, The netfilter.org iptables project (www.netfilter.org/projects/iptables/index.html).

[21] P. Neumann and R. Feiertag, PSOS revisited, *Proceedings of the Nineteenth Annual Computer Security Applications Conference*, pp. 208–216, 2003.

[22] C. Piller, Hackers target energy industry, *Los Angeles Times*, July 8, 2002.

[23] P. Porras, M. Fong and A. Valdes, A mission-impact-based approach to INFOSEC alarm correlation, in *Recent Advances in Intrusion Detection (LNCS 2516)*, A. Wespi, G. Vigna and L. Deri (Eds.), Springer, Berlin-Heilderberg, pp. 95–114, 2002.

[24] M. Roesch, Snort: Lightweight intrusion detection for networks, presented at the *Thirteenth USENIX Systems Administration Conference*, 1999.

[25] O. Ruwase and M. Lam, A practical dynamic buffer overflow detector, *Proceedings of the Network and Distributed System Security Symposium*, pp. 159–169, 2004.

[26] S. Singh, J. Lyons and D. Nicol, Fast model-based penetration testing, *Proceedings of the 2004 Winter Simulation Conference*, pp. 309–317, 2004.

[27] S. Smith, *Trusted Computing Platforms: Design and Applications*, Springer, New York, 2005.

[28] K. Stouffer, J. Falco and K. Kent, Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security – Initial Public Draft, National Institute of Standards and Technology, Gaithersburg, Maryland, 2006.

[29] A. Valdes and K. Skinner, Adaptive model-based monitoring for cyber attack detection, in *Recent Advances in Intrusion Detection (LNCS 1907)*, H. Debar, L. Me and S. Wu (Eds.), Springer, Berlin-Heilderberg, pp. 80–92, 2000.

[30] A. Valdes and K. Skinner, Probabilistic alert correlation, in *Recent Advances in Intrusion Detection (LNCS 2212)*, W. Lee, L. Me and A. Wespi (Eds.), Springer, Berlin-Heidelberg, pp. 54-68, 2001.

[31] A. Wool, A quantitative study of firewall configuration errors, *IEEE Computer*, vol. 37(6), pp. 62–67, 2004.

[32] M. Zhivich, Detecting Buffer Overflows Using Testcase Synthesis and Code Instrumentation, M.S. Thesis, Department of Electrical Engineering and Computer Sciences, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2005.

[33] M. Zitser, R. Lippmann and T. Leek, Testing static analysis tools using exploitable buffer overflows from open-source code, *Proceedings of the International Symposium on the Foundations of Software Engineering*, pp. 97–106, 2004.