

Publications

Susanta Nanda

susanta@cs.sunysb.edu

URL: <http://www.ecsl.cs.sunysb.edu/~susanta/>

[1] *General Dynamic Information Flow Tracking for Secure and Repairable Distributed Applications*

Abstract

The ability to accurately correlate events of distributed applications is essential for building secure, manageable and repairable distributed applications. Previously researchers took a statistical learning approach to the problem of distributed system event correlation. The research project presented in this paper takes a program transformation approach to enable automated and accurate correlation of distributed system events. More specifically, this paper describes the design, implementation, and evaluation of a generalized dynamic information flow tracking compiler (called GIFT) that, given a specification of application-specific metadata and their propagation rules, automatically inserts code into a distributed application that propagates these metadata along the execution paths according to the specified rules. Using GIFT, we have developed three applications: (1) A dynamic checking compiler for web applications that can effectively protect a web site from SQL-injection attacks and from being a relay of script injection attacks, (2) an automated attack signature generation system for internet servers that are vulnerable to buffer overflow attacks, and (3) a repairable three-tier internet service that can accurately identify the damage scope of a detected attack and repair the compromised service state with minimal collateral damage. Through these applications, we show how accurate correlation of distributed system events is indeed a key enabling building block and how GIFT facilitates the introduction of such event correlation capability. Despite the extensive instrumentation that GIFT introduces for event correlation, its run-time performance penalty remains below 30% for a set of web applications we have tested.

Full Text: N/A

[2] *VDRS: An Inexpensive Approach to Server Disaster Recovery*

Abstract

Traditional disaster recovery solutions involve specialized data replication hardware and software, dedicated wide-area circuits, and duplicate infrastructure at primary and backup locations, and introduce considerable cost and complexity. As a result, such solutions are limited primarily to large enterprises and institutions. For smaller, more cost-conscious environments such as moderate volume e-commerce sites, we propose VDRS, a virtualized disaster recovery system that provides end-to-end failover for Linux-based server applications. VDRS is completely implemented in software and leverages Xen-based system virtualization to mask hardware dependencies. VDRS provides data replication by extending existing operating system drivers, availability monitoring through network- and application-level probing, and application recovery through remote virtual machine management.

We describe the details of the VDRS implementation, along with the results of our evaluation of its performance, data loss, and recovery time using several workloads. The performance results predict a maximal throughput loss of about 7% compared to running the application in Xen alone. This loss is due to the data replication overheads that our solution incurs on the I/O path. Our experiments show an upper bound for data loss at about 12 MB based on measurements for an application virtual machine that writes continuously and whose backup site is separated by a high-latency WAN connection. Measurements of data loss with the TPC-W application benchmark indicate much smaller data loss (on the order of 200-300 KB). Remote recovery times for such a guest VM are on the order of 30s which is about half of a typical machine boot time.

Full Text: N/A

[3] *Foreign Code Detection on the Windows/X86 Platform*

Abstract

As new attacks against Windows-based machines emerge almost on a daily basis, there is an increasing need to “lock down” individual users’ desktop machines in corporate computing environments. One particular way to lock down a user computer is to guarantee that only authorized binary programs are allowed to run on that computer. A major advantage of this approach is that binaries downloaded without the user’s knowledge, such as spyware, adware, or code entering through buffer overflow attacks, can never run on computers that are locked down this way. This paper presents the design, implementation and evaluation of FOOD, a foreign code detection system specifically for the Windows/X86 platform, where foreign code is defined as any binary programs that do not go through an authorized installation procedure. FOOD verifies the legitimacy of binary images involved in process creation and library loading to ensure that only authorized binaries are used in these operations. In addition, FOOD checks the target address of every indirect branch instruction in Windows binaries to prevent illegitimate control transfers to either dynamically injected mobile code or pre-existing library functions that are potentially damaging. Combined together, these techniques strictly prevent the execution of any foreign code. Experiments with a fully working FOOD prototype show that it can indeed stop all spyware and buffer overflow attacks we tested, and its worst-case run-time performance overhead associated with foreign code detection is less than 35%.

Full Text: <http://www.ecsl.cs.sunysb.edu/tr/TR203.pdf>

[4] *A Feather-weight Virtual Machine for Windows Applications*

Abstract

Many fault-tolerant and intrusion-tolerant systems require the ability to execute unsafe programs in a realistic environment without leaving permanent damages. Virtual machine technology meets this requirement perfectly because it provides an execution environment that is both realistic and isolated. In this paper, we introduce an OS-level virtual machine architecture for Windows applications called *Feather-weight Virtual Machine* (FVM), under which virtual machines share as many resources of the host machine as possible while still isolated from one another and from the host machine. The key technique behind FVM is *namespace virtualization*, which isolates virtual machines by renaming resources at the OS system call interface. Through a copy-on-write scheme, FVM allows multiple virtual machines to physically share resources but logically isolate their resources from each other. A main technical challenge in FVM is how to achieve strong

isolation among different virtual machines and the host machine, due to numerous namespaces and inter-process communication mechanisms on Windows. Experimental results demonstrate that FVM is more flexible and scalable, requires less system resource, and incurs lower start-up and run-time performance overhead than existing hardware-level virtual machine technologies, and thus makes a compelling building block for security and fault-tolerant applications.

Full Text: <http://www.ecsl.cs.sunysb.edu/tr/TR189.pdf>

[5] *BIRD: Binary Interpretation using Runtime Disassembly*

Abstract

The majority of security vulnerabilities published in the literature is due to software bugs. Many researchers have developed program transformation and analysis techniques to automatically detect or eliminate such vulnerabilities. So far, most of them cannot be applied to commercially distributed applications on the Windows/x86 platform, because it is almost impossible to disassemble a binary file with 100% accuracy and coverage on that platform. This paper presents the design, implementation, and evaluation of a binary analysis and instrumentation infrastructure for the Windows/x86 platform called BIRD (Binary Interpretation using Runtime Disassembly), which provides two services to developers of security-enhancing program transformation tools: converting binary code into assembly language instructions for further analysis, and inserting instrumentation code at specific places of a given binary without affecting its execution semantics. Instead of requiring a high fidelity instruction set architectural emulator, BIRD combines static disassembly with an on-demand dynamic disassembly approach to guarantee that each instruction in a binary file is analyzed or transformed before it is executed. It takes 12 student months to develop the first BIRD prototype, which can successfully work for all applications in Microsoft Office suite as well as Internet Explorer and IIS web server, including all DLLs that they use. Moreover, the additional throughput penalty of the BIRD prototype on production server applications such as Apache, IIS, and BIND is uniformly below 4%.

Full Text: <http://www.ecsl.cs.sunysb.edu/tr/TR190.pdf>

[6] *A Survey of Virtualization Technologies*

Abstract

Virtualization is a technology that combines or divides computing resources to present one or many operating environments using methodologies like hardware and software partitioning or aggregation, partial or complete machine simulation, emulation, time-sharing, and others. Virtualization technologies find important applications over a wide range of areas such as server consolidation, secure computing platforms, supporting multiple operating systems, kernel debugging and development, system migration, etc, resulting in widespread usage. Most of them present similar operating environments to the end user; however, they tend to vary widely in their levels of abstraction they operate at and the underlying architecture. This paper surveys a wide range of virtualization technologies, analyzes their architecture and implementation, and proposes a taxonomy to categorize them on the basis of their abstraction levels. The paper identifies the following abstraction levels: instruction-set level, hardware abstraction layer (HAL) level, operating system level, library level and application level virtual machines. It studies examples from each of the categories and provides relative comparisons. It also gives a broader perspective of the virtualization technologies and gives an insight that can be extended to accommodate

future virtualization technologies under this taxonomy. The paper proposes the concept of an extremely lightweight technology, which we call as *Featherweight Virtual Machine* (FVM), that can be used to "try out" untrusted programs in a realistic environment without causing any permanent damage to the system. Finally, it demonstrates FVM's effectiveness by applying it to two applications: secure mobile code execution and automatic clean uninstall of Windows programs.

Full Text: <http://www.ecsl.cs.sunysb.edu/tr/TR179.pdf> [Most cited article]

[7] *Viking: A Multi-Spanning-Tree Ethernet Architecture for Metropolitan Area and Cluster Networks*

Abstract

Simplicity, cost effectiveness, scalability, and the economies of scale make Ethernet a popular choice for local area networks, as well as for storage area networks and increasingly metropolitan-area networks. These applications of Ethernet elevate it from a LAN technology to a ubiquitous networking technology, thus prompting a rethinking of some of its architectural features. One weakness of existing Ethernet architecture is its use of single spanning tree, which, while useful at avoiding routing loops, leads to low link utilization and long failure recovery time. To apply Ethernet to cluster networks and MANs, these problems need to be addressed. In this paper we propose a Multi-Spanning-Tree Ethernet architecture, called *Viking* that improves both aggregate throughput and fault tolerance by exploiting standard virtual LAN technology in a novel way. By supporting multiple spanning trees through VLAN, Viking makes the most of the inherent redundancies in most mesh-like networks and delivers a multi-fold throughput gain over single-spanning-tree Ethernet with the same physical network topology. It also provides much faster failure recovery, reducing the down-time to a sub-second range from that of multiple seconds in single-spanning-tree Ethernet architecture. Finally, based only on standard mechanisms, Viking is readily implementable on commodity Ethernet switches without any firmware modifications.

Full Text: <http://www.ecsl.cs.sunysb.edu/tr/viking.pdf>