Purdue University
Purdue e-Pubs

Computer Science Technical Reports | Department of Computer Science

2003

Protection of Application Service Hosting Platforms: An Operating System Perspective

Xuxian Jiang

Dongyan Xu
Purdue University, dxu@cs.purdue.edu

Report Number:
03-010

Jiang, Xuxian and Xu, Dongyan, "Protection of Application Service Hosting Platforms: An Operating System Perspective" (2003). *Computer Science Technical Reports.* Paper 1559.
http://docs.lib.purdue.edu/cstech/1559

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

PROTECTION OF APPLICATION SERVICE HOSTING PLATFORMS: AN OPERATING SYSTEM PERSPECTIVE

Xuxian Jiang
Dongyan Xu

Computer Science Department
Purdue University
West Lafayette, IN 47907

CSD TR #03-010
March 2003

Protection of Application Service Hosting Platforms: an Operating System Perspective
(*position paper*)

Xuxian Jiang, Dongyan Xu
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA
{jiangx, dxu}@cs.purdue.edu

Abstract

The Application Service Hosting Platform (ASHoP), as a realization of the utility computing *vision, has recently received tremendous attention from both industry and academia. An ASHoP provides a shared and high performance platform to host multiple Application Services (ASes). The ASes are* outsourced *by Application Service Providers (ASPs) to save their own IT resources. Furthermore, ASHoP resources are allocated to the ASes in an on-demand fashion, so that resource supply always follows the time-varying service load.*

In this paper, we argue that the protection of ASHoPs *poses new challenges. Different from a dedicated server platform which is analogous to a private house, an ASHoP is like an apartment building, involving the 'host' - the ASHoP infrastructure and the 'tenants' - the ASes. As a result, an ASHoP has inherent requirement of openness, sharing, and mutual isolation: it must provide protection and isolation between the host and the tenants, as well as between different tenants. Unfortunately, traditional OS architecture and mechanisms are not adequate to meet these requirements.*

We advocate new OS architecture and mechanisms for ASHoP protection, based on the virtual OS *technology. Our experience shows that virtual OS achieves better protection of ASHoP infrastructure, as well as better isolation between the ASes hosted. Furthermore, we present novel protection mechanisms we have implemented: (1)* resource isolation *between ASes, (2)* virtual networking and firewalling *between ASes in a physical ASHoP server, and (3)* untamperable and privacy-conserving AS blackboxing *for the logging of activities inside each AS. Analogous to the blackbox on an aircraft, the software blackbox in each AS is* untamperable*; and it continues to log even after this AS has been compromised. Moreover, for the privacy of the AS, log data in the AS blackbox are* not *viewable to the 'landlord' (namely ASHoP owner) without authorization.*

1 Introduction

An Application Service Hosting Platform (ASHoP) is a set of high performance servers connected by high speed LANs or switches. The ASHoP creates a shared platform for the hosting of multiple Application Services (ASes), which are *outsourced* by their corresponding Application Service Providers (ASPs). Examples of AS include on-line event catering (such as a conference), on-line shopping, and e-laboratories. In an ASHoP, ASes are created on-demand at the requests of their ASPs, and ASHoP resources are dynamically allocated to the ASes according to their time-varying service loads. Therefore, ASHoP reflects the vision of *utility computing*: computational resources are supplied on-demand, and turned off when no longer needed. ASHoPs have recently drawn tremendous attention from both industry (such as Oceano [5] of IBM and Utility Data Center [4] of HP) and academia (such as Denali [31], SHoP [28], and SODA [19]).

However, current research in ASHoP mainly focuses on resource and service quality issues. Little efforts have been devoted to the critical problem of OS architecture and mechanisms for *ASHoP security and protection*. In this paper, we show that ASHoP protection poses new research challenges. The new challenges are due to an ASHoP's inherent requirement of *openness*, *sharing* and *mutual isolation*. Unlike a dedicated server platform which is analogous to a private house, an ASHoP is like an apartment building, involving both the 'host' - the ASHoP infrastructure and the 'tenants' - the ASes. From a tenant's point of view, the ASHoP should be a safe place to 'live' with privacy and isolation from other tenants. From the landlord's point of view, the tenants should not do any damage to the 'property' and should not bother other tenants. In this paper, we consider the following requirements for ASHoP protection:

- *ASHoP infrastructure safety* The ASHoP servers should be protected from malicious attacks from the

outside. If one ASHoP server is attacked, *all* ASes residing in it will be affected.

- *Confinement of ASes* The ASHoP should prevent any damage by its tenants - the ASes. It is desirable that the activities of each AS are strictly confined within their allocated space in the ASHoP.

- *Isolation between ASes* It is equally important that the ASHoP should provide isolation *between* the tenants: Each AS should run as if it is in a dedicated environment. Between ASes sharing the same ASHoP server, isolation is desirable with respect to (1) *administration*: an ASP should have administrator privilege, but *only* within its own AS; (2) *fault and attack*: a crash or security breach of one AS should not affect other ASes; and (3) *resources*: each AS should be guaranteed the 'slice' of ASHoP server allocated to it, and it should not be able to launch a local DoS attack upon other ASes.

- *Secure communication between ASes* Sometimes neighbors do talk to each other: an AS may need to communicate with another AS to form a composite and value-added *service chain*. Therefore, the ASHoP should enable secure inter-AS communication.

- *Untamperable and privacy-conserving logging* The activities inside each AS should be logged for auditing or forensic purpose. In each AS, the logging module should be untamperable by an intruder and should continue to log even after the AS is compromised. On the other hand, the log data may contain sensitive information, such as the customer information in an on-line shopping service. The ASHoP should assure that the log data are *not* viewable by the landlord (i.e. ASHoP owner), except during post-attack forensic analysis with authorization.

Unfortunately, the traditional *single-level* OS architecture, in which one underlying *host OS* supports *all* ASes running on top of it, is not adequate to meet the above requirements (to be discussed in Section 2).

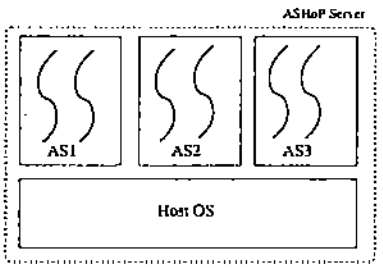
In this paper, we present a new *two-level* ASHoP architecture, based on the *virtual OS* technology. In the two-level architecture, each AS runs on top of a virtual *guest OS*; while the guest OS runs on top of the *host OS*. Our experience shows that virtual OS achieves better protection of ASHoP infrastructure, as well as better isolation between the ASes hosted.

Although the virtual OS technology is not brand new [8, 13, 31], we have designed and implemented a number of novel mechanisms for ASHoP protection. Most notably, we have implemented (1) *resource isolation* (CPU, bandwidth, and memory) between ASes, (2) *virtual networking and firewalling* between ASes inside the same ASHoP server, and (3) *untamperable and privacy-conserving AS blackboxing* for the logging of activities inside each AS. Analogous to the blackbox on an aircraft, the software blackbox in each AS is *untamperable*, and logs information even after this AS has been compromised. Moreover, for the privacy of the AS, log data in the AS blackbox are *not* viewable to the landlord without authorization.

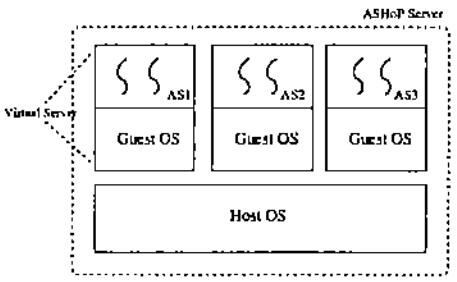
The rest of the paper is organized as follows: Section 2 compares the single-level and two-level ASHoP architectures. Section 3 presents an overview of our two-level ASHoP architecture called SODA. Section 4 describes the novel ASHoP protection mechanisms in SODA. Section 5 compares our work with related work. Finally, Section 6 concludes this paper and outlines future work.

2 Comparison of ASHoP Architectures

The two different ASHoP architectures are shown in Figure 1: Figure 1(a) shows the traditional *single-level* ASHoP architecture; while Figure 1(b) shows the *two-level* architecture based on the 'guest OS/host OS' structure.



(a) Single-level architecture (host OS)



(b) Two-level architecture (guest OS/host OS)

Figure 1. Two different ASHoP architectures (only showing one ASHoP server)

In both architectures, multiple ASes are hosted in one ASHoP server. In the single-level architecture, all ASes run directly on top of the host OS. In the two-level architecture, each AS runs within a *virtual server*, which is physically a

'slice' of the ASHoP server. Inside the virtual server, the AS software runs on top of a virtual guest OS.

We argue that the traditional single-level architecture is not adequate for ASHoP protection, in the following aspects:

- *Administration isolation*: It is desirable that each ASP has full administrator privilege only *within* the corresponding AS, so that the ASP can perform AS-specific management tasks such as data/software upgrade. However, if the administrator privileges of all ASPs are at the same (host OS) level, access control will become complicated and may lead to security holes.

- *Installation isolation*: Different ASes may require the same library, but of *different* versions, or their service daemons may require the *same* port binding. In the single-level architecture, such conflicts are difficult to resolve and can potentially lead to local DoS attacks (by port exhaustion, for example) between ASes. On the other hand, the two-level architecture naturally eliminate such conflicts.

- *Fault/attack isolation*: If all ASes run at the same host OS level, any fault or security breach in one AS will affect the host OS and therefore other ASes. For example, *ghttpd* [22] is a light-weight web server run by the root. However, one known attack upon *ghttpd* is: a malicious packet is sent as an HTTP request, causing buffer overflow to bind a shell on a certain port. Then the attacker can remotely log in using the port, and run a remote shell! On the other hand, in the two-level architecture, since the root that runs *ghttpd* is the root of the *guest OS*, not the host OS, the attack will *not* affect the host OS as well as other ASes.

- *Crash recovery and forensics*: In the single-level architecture, to recover from an attack/crash of an AS, the entire ASHoP server will have to be rebooted. As a result, other ASes in the same ASHoP server will be affected. On the other hand, in the two-level architecture, the recovery of one AS has *no* impact on the normal operations of other ASes: the ASHoP administrator can simply restart the virtual server; and the image of the attacked virtual server is dumped to a file and sent to an off-line site for forensic analysis.

However, the advantages of the two-level ASHoP architecture do come with a cost. Due to the guest OS/host OS structure, the performance slow-down is inevitable. In other words, to achieve the same performance or service quality as in the single-level architecture, the two-level architecture requires more CPU capacity. With the rapid advances in high performance server systems, such a price is expected to be more affordable in the near future.

3 SODA: Our Two-Level ASHoP Architecture

We are developing a two-level ASHoP architecture called SODA[1]. Currently, SODA supports Linux as the host OS of ASHoP servers. For the guest OS running in each virtual server, we leverage an open-source virtual OS project called UML [13], or User-Mode Linux. Unlike other virtual machine techniques such as VMWare [3], a UML runs directly in the unmodified *user space* of the host OS; and processes within a UML will be executed in the virtual server exactly the same way as they would be executed in a native Linux machine. A special thread is created to intercept the system calls made by all process threads of the UML, and redirect them into the host OS kernel. The following features are enabled by leveraging the 'UML (guest OS)/Linux (host OS)' structure:

- The host OS has a separate *kernel space* from the UMLs, therefore preventing any harm done by the individual UMLs.

- An IP address is assigned to each virtual server, so that it can perform internetworking like a real server.

- A virtual server can be frozen/restarted without affecting other virtual servers: the images of both the UML and the AS on top of it can be copied to a file, and be conveniently backed up and restarted. Such feature enables easy fault/attack recovery and forensic analysis.

However, neither the original Linux (as host OS) nor UML (as guest OS) is powerful enough to support ASHoP security. We have extended both of them by implementing a number of novel protection mechanisms, as described in the next section.

4 ASHoP Protection Mechanisms in SODA

In this section, we present three ASHoP protection mechanisms in SODA: *resource isolation*, *virtual switching and firewalling*, and *untamperable and privacy-conserving AS blackboxing (logging)*. All these mechanisms aim at meeting the ASHoP protection requirements listed in Section 1.

Although far from being a *complete* suite of ASHoP security solutions, these mechanisms can be seen as the basis for the implementation of more complicated mechanisms and policies.

[1] SODA stands for Service-On-Demand Architecture. Details about the non-security aspects of SODA can be found in [19].

4.1 Resource Isolation between Virtual Servers

Resource isolation not only provides performance guarantee to the AS running in each virtual server, but also prevents an ill-behaving or malicious AS from launching local DoS attacks upon other ASes in the same ASHoP server. Currently, our SODA implementation supports CPU, network bandwidth, and memory isolation.

- *CPU capacity isolation* is achieved by implementing a coarse-grain CPU proportional sharing scheduler in the Linux host OS. The scheduler enforces the CPU share allocated to each virtual server. The CPU share of a virtual server is decided when the corresponding AS is admitted to the ASHoP. Within one virtual server, all processes bear the same user (AS) id. The host OS CPU scheduler then enforces CPU proportional sharing among all processes based on their user ids.

- *Network bandwidth isolation* is similarly achieved by implementing a traffic shaper inside the Linux host OS. The traffic shaper enforces the outbound bandwidth share allocated to each virtual server. Recall that each virtual server has its own IP address. The traffic shaper achieves bandwidth isolation between virtual servers based on the IP addresses of outgoing packets generated by these virtual servers.

- *Memory isolation*: Memory is critical to the performance of virtual servers (and therefore that of ASes). SODA simply leverages the *memory usage limit* feature of UML: the maximum amount of memory available to a virtual server (both AS and guest OS) can be specified as a parameter when UML, the guest OS, is started.

Note that resource isolation only prevents DoS attacks *between* ASes. To prevent intra-AS DoS attacks launched by the clients of an AS, other methods (such as client puzzles [6]) still need to be installed as part of the AS software.

4.2 Virtual Switching and Firewalling

SODA uses virtual switching to connect the virtual servers to the outside world as well as between themselves. In the meantime, it creates a strong protection for the ASHoP server: the ASHoP server itself will have *no* IP address. Therefore, the host OS is totally 'invisible' and therefore un-attackable from the outside.

To realize the seemingly conflicting goals of virtual server networking and ASHoP server in-visibility, we implement a software switch module inside the host OS (Figure 2). This solution is inspired by a real layer-two switch connecting physical NICs, yet the switch itself does not have an IP address. Similarly, we create a software switch in the host OS, connecting multiple virtual NICs of the virtual servers and one physical NIC of the ASHoP server.

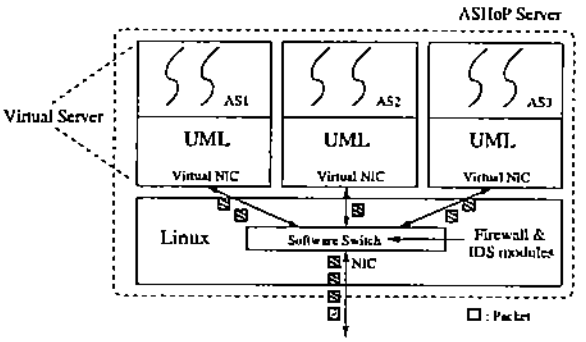


Figure 2. Virtual switching and firewalling: software switch inside the host OS (Linux)

The software switch forwards packets to/from the virtual servers. Furthermore, we enhance the software switch to support firewalling and intrusion detection *between the virtual servers*. The physical firewall and intrusion detection system (IDS) can protect the ASHoP server from attacks from *outside*, but they *cannot* handle attacks from one virtual server against another virtual server *inside* the same ASHoP server. Fortunately, our software switch provides an ideal venue to perform inter-virtual-server firewalling and intrusion detection: the firewall and IDS modules are plugged in on the packet path between the virtual servers. Currently, the firewall and IDS code in SODA is based on the widely adopted *netfilter/iptables/snort* suite. However, the software switch can easily accommodate more complicated firewalling rules (such as *reverse* firewalling) and intrusion detection methods (such as NATE [27]).

With virtual switching and firewalling, the ASes can communicate and collaborate in a secure fashion, making it possible to create composite and value-added application services. On the other hand, it is also possible to enforce communication isolation via the virtual firewall - for example, to prevent two competing ASes (selling the same products) from attacking each other.

4.3 AS Blackboxing

In ASHoP, the logging of AS activities is essential to auditing and forensic analysis. However, we face the following dilemma: If the guest OS performs logging, the log may be tampered with or even erased by an intruder, who tends to do so first thing after breaking in. On the other hand, if the host OS performs logging, two problems will arise: (1) the privacy of the tenants (ASes) is violated, because the landlord can view the AS log data and (2) it

is difficult or even impossible to log activities that happen *inside* the virtual server.

In SODA, we use a novel strategy to solve the above problem: the log data are *generated* by the guest OS kernel[2], but *stored* in the host OS. The logging module inside the guest OS kernel, called *AS blackbox*, is capable of taking snapshots of AS execution, as well as collecting system-wide (within the virtual server) log data such as those from *syslogd* and *klogd*, and verbatim record of user console (local and remote) input. The log data are immediately *pushed down* to the host OS for storage. Recall that the host OS is un-attackable from both inside and outside, and therefore an ideal place to store the log data.

Furthermore, to conserve the privacy of the ASes, the AS blackbox will *encrypt* the log data before pushing them down to the host OS. For an AS, the key to encrypt the log data is *generated and compiled* into the UML kernel by a trusted authority called the *Trusted UML Factory*, before the AS is created in the ASHoP server. As shown in Figure 3, the Trusted UML Factory obtains the image of the AS from the ASP, and builds a customized UML kernel with the key. The images of the AS and UML will then be downloaded to the ASHoP server. The key is nowhere to be obtained in the ASHoP server. Instead, it will be kept by the Trusted UML Factory and by the ASP. In the event of a crash or an attack, the key will be used to decrypt the log data for forensic analysis.

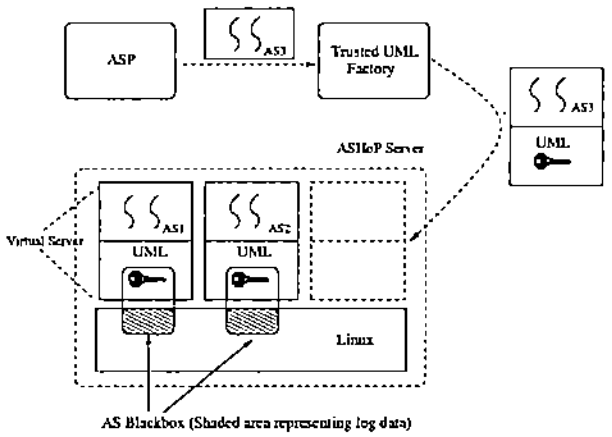


Figure 3. AS blackboxing: AS blackbox 'manufactured' by Trusted UML Factory; log data generated and encrypted in guest OS (UML) kernel and stored in host OS (Linux)

One of the main challenges in the AS blackbox implementation is to 'hijack' the log information originally logged by *syslogd*, which in the danger of being killed by an intruder. The challenge is the syslog *library hijacking* in glibc library: *glibc* is so essential in the guest OS (UML) that once *glibc* library is upgraded or replaced, both the AS and the UML will be affected. Also the dynamic loader (usually the file /lib/ld-linux.so.2) should reflect the change in *glibc*, and be consistent with *glibc* at all times[3]. Finally, for the development environment, the *gcc* compiler and all related header files should reflect the new *glibc* library to ensure consistency. In our implementation, this major *UML rebuild* is performed by the Trusted UML Factory: The Trusted UML Factory uses the cross-compiler and toolchain (currently targeting x86 platforms) to generate a much modified UML kernel (with the AS blackbox key inside), and packages the AS into the *ext2* file system, which will be mounted by the UML as the root file system.

5 Related Work

Internet hosting has been moving from content hosting to application service (AS) hosting. The former involves the hosting of digital contents (such as photos and audio/video files) outsourced by a content provider. Examples of content hosting platform include *Yahoo! Photos* and Akamai [1]. On the other hand, AS hosting involves not just contents, but also highly application-specific software (such as genome matching and e-Commerce). Therefore, it requires a higher degree of isolation between ASes in resource, administration, and fault/attack recovery. Examples of ASHoP platform include Oceano [5] of IBM and Utility Data Center [4] of HP.

Recently, virtual OS has received significant attention in the OS community. Representative projects include Denali [31], UML [13], UMLinux [12], SODA [19], and Xen [8]. They all support the creation of virtual servers based on the guest OS/host OS architecture. However, the protection and security mechanisms in SODA have *not* been reported in the other projects.

It is noteworthy and interesting that the user-level mechanism for system call interposition in [18] is very similar to the way the UML [13] guest OS is implemented. They both exploit the *ptrace* mechanism provided by a UNIX-style OS (including Linux) for the surveillance of other processes. The purpose of [18] is intrusion detection and confinement, also similar to that of the UML.

Server resource isolation has been extensively studied in the traditional single-level OS architecture [7, 23, 24, 26]. On the other hand, SODA is the *first* virtual-OS-based architecture that implements resource isolation between virtual servers.

The *honeypot* is "a computer system that is specifically designed to capture all activity ... of a criminal who has

[2]To make the guest OS kernel un-damageable by its processes, we leverage the 'skas' mode of UML: the UML kernel is in a separate address space from it processes, making the UML kernel totally invisible to its processes.

[3]We had an experience in which we used the newer version of /lib/libc.so.6 without updating the corresponding /lib/ld-linux.so.2. The entire UML became unusable, since most commands depended on both *glibc* and dynamic loader.

gained unauthorized access to the system" [11]. Recently, virtual OS (including UML) has also been applied to the deployment of honeypot [2], in order to achieve better attack confinement and log data capture. However, due to its different purpose from SODA, UML-based honeypot does *not* support resource isolation and virtual firewalling *between* honeypots. As to logging, UML-based honeypot supports untamperable logging of attacker's keystrokes. However, it is *not* able to capture the log data originally captured by *syslogd*. As discussed in Section 4.3, this is a highly challenging task in our implementation of AS blackboxing.

Another project that addresses untamperable logging is ReVirt [14]. ReVirt is based on the UMLinux [12] guest OS. One key weakness of ReVirt is that its logging module is completely *outside* the guest OS. As a result, the log data of ReVirt are much less detailed than those captured by SODA's AS blackbox.

A paradigm similar to the ASHoP is the execution of mobile code on a foreign host platform. It also involves a two-way security relationship between the mobile code and the host: we need protection of the latter against the former [16, 29] *and vice versa* [16, 21]. However, the ASHoP paradigm is different in that the host-tenant relationship in an ASHoP usually lasts longer, and that the inter-tenant relationship should also be considered.

Finally, our work in OS support for ASHoP protection complements the works in programming language support for software safety [15, 17, 25, 30]. The language-level solutions are critical to the checking and enforcement of safety properties of the AS software, guest OS, and host OS, in order to create a highly secure ASHoP.

6 Conclusions and Future Work

We have shown that the protection of Application Service Hosting Platforms (ASHoPs) poses new research challenges. Due to ASHoP's inherent requirement of openness, sharing, and mutual isolation, novel OS architecture and mechanisms are needed to provide protection and isolation between the host (ASHoP infrastructure) and the tenants (ASes), as well as between different tenants. We argue that the traditional single-level ASHoP architecture is *not* adequate for such protection, and that the two-level architecture based on virtual OS technology appears to be a promising candidate.

The development of SODA, our two-level ASHoP architecture, has so far been very encouraging. We have successfully implemented a number of novel security mechanisms, which can be used as the basis for more complicated mechanisms and policies. Meanwhile, there are many problems waiting to be solved in ASHoP protection and security. Examples include: resource access authorization [9, 32] in ASHoPs, privacy policy model [20] for ASHoP, replay of attacks based on log data in AS blackboxes, and risk management [10] in ASHoP - including an ASP's risk in selecting an ASHoP, and the ASHoP owner's risk in accepting an AS. We expect that studies on these problems - in the context of two-level ASHoP architecture, will yield new research results and opportunities.

References

[1] Akamai. *http://www.akamai.com.*

[2] Know Your Enemy: Learning with User-Mode Linux. *Honeynet Project White Paper, http://project.honeynet.org/papers/uml.*

[3] VMWare. *http://www.vmware.com.*

[4] HP Utility Data Center. *HP Technical White Paper*, 2001.

[5] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, and M. Kalantar. Oceano: SLA based Management of a Computing Utility. *IFIP/IEEE Intl. Symp. on Integrated Network Management*, May 2001.

[6] T. Aura, P. Nikander, and J. Leiwo. DOS-resistant authentication with client puzzles. *Security Protocols Workshop 2000, LNCS*, 2133, 2000.

[7] G. Banga, P. Druschel, and J. Mogul. Resource Containers: a New Facility for Resource Management in Server Systems. *USENIX Symposium on Operating Systems Design and Implementation (OSDI'99)*, 1999.

[8] P. Barham and et al. Xen 2002. *Technical Report UCAM-CL-TR-553, University of Cambridge*, Jan. 2003.

[9] K. Beznosov, Y. Deng, B. Blakley, C. Burt, and J. Barkley. A Resource Access Decision Service for CORBA-Based Distributed Systems. *15th Annual Computer Security Applications Conference (ACSAC'99)*, 1999.

[10] B. Blakley and G. R. Blakley. All Sail, No Anchor, 1: Cryptography, Risk, and e-Commerce. *5th Australasian Conference on Information Security and Privacy (ACISP 2000)*, 2000.

[11] C. Brenton. Honeynets. *Technical White Paper, Institute for Security Technology Studies, Dartmouth College.*

[12] K. Buchacker and V. Sieh. Framework for Testing the Fault-Tolerance of Systems Including OS and Network Aspects. *IEEE Symposium on High Assurance System Engineering (HASE 2001)*, 2001.

[13] J. Dike. User Mode Linux. *http://user-mode-linux.sourceforge.net.*

[14] G. Dunlap, S. King, S. Cinar, M. Basrai, and P. Chen. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. *USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002)*, 2002.

[15] D. Evans and D. Larochelle. Improving Security Using Extensible Lightweight Static Analysis. *IEEE Software*, Jan. 2002.

[16] S. N. Foley, T. Quillinan, and J. P. Morrison. Secure Component Distribution Using WebCom. *17th International Conference on Information Security*, May 2002.

[17] V. Haldar and M. Franz. Towards Trusted Systems, From The Ground Up. *The 10th ACM SIGOPS European Workshop*, Sept. 2002.

[18] K. Jain and R. Sekar. User-Level Infrastructure for System Call Interposition: A Platform for Intrusion Detection and Confinement. *ISOC Network and Distributed Systems Symposium (NDSS 2000)*, 2000.

[19] X. Jiang and D. Xu. SODA: a Service-On-Demand Architecture for Application Service Hosting Utility Platforms. *IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.

[20] G. Karjoth and M. Schunter. A Privacy Policy Model for Enterprises. *15th IEEE Computer Security Foundations Workshop*, 2002.

[21] O. K. Onbilger, R. Newman, and R. Chow. A Distributed and Compromise-Tolerant Mobile Agent Protection Scheme. *International Conference on Intelligent Agents, Web Technology and Internet Commerce (IAWTIC 2001)*, 2001.

[22] G. Owen. ghttpd. *http://gaztek.sourceforge.net/ghttpd.*

[23] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource Kernels: A Resource-Centric Approach to Real-Time Systems. *SPIE/ACM Conference on Multimedia Computing and Networking (MMCN'98)*, Jan. 1998.

[24] J. Reumann, A. Mehra, K. Shin, and D. Kandlur. Virtual Services: A New Abstraction for Server Consolidation. *USENIX 2000 Annual Technical Conference*, June 2000.

[25] S. Soman, C. Krintz, and G. Vigna. Detecting Malicious Java Code Using Virtual Machine Auditing. *12th USENIX Security Symposium*, 2003.

[26] V. Sundaram, A. Chandra, P. Goyal, and P. Shenoy. Application Performance in the QLinux Multimedia Operating System. *The Eighth ACM Conference on Multimedia*, Nov. 2000.

[27] C. Taylor and J. Alves-Foss. An Empirical Analysis of NATE: Network Analysis of Anomalous Traffic Events. *10th New Security Paradigms Workshop (NSPW 2002)*, Sept. 2002.

[28] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. *USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Dec. 2002.

[29] V. N. Venkatakrishnan and R. Sekar. Empowering Mobile Code Using Expressive Security Policies. *10th New Security Paradigms Workshop (NSPW 2002)*, 2002.

[30] D. Wallach, D. Balfanz, D. Dean, and E. Felten. Extensible Security Architectures for Java. *16th ACM Symposium on Operating Systems Principles (SOSP'97)*, 1997.

[31] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. *USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Dec. 2002.

[32] M. E. Zurko, R. Simon, and T. Sanfilippo. A User-Centered, Modular Authorization Service Built on an RBAC Foundation. *IEEE Symposium on Security and Privacy*, 1999.