SVGrid: A Secure Virtual Environment for Untrusted Grid Applications

Xin Zhao Kevin Borders Atul Prakash

University of Michigan, 1301 Beal Ave, Ann Arbor, MI, 48109, USA

{zhaoxin,kborders,aprakash}@eecs.umich.edu

ABSTRACT

Most grid security researches focus on user authentication and secure communication, the protection of grid computers is left to the underlying operating system. Unfortunately, most OS level protection mechanisms can be turned off after an attacker manages to exploit a vulnerability to gain privileged access.

This paper proposes SVGrid, a Secure Virtual Grid computing environment, to protect grid computers' filesystem and network from malicious grid applications. SVGrid works by isolating grid applications in one or more grid virtual machines whose filesystem and network service are moved into a dedicated monitor virtual machine. All file and network access requests are then forced to go through the monitor virtual machine, where security policies can be enforced. The resource compartment guarantees that appropriate security policy enforcement cannot be bypassed or disabled, even if a grid virtual machine is compromised. We tested SVGrid against attacks on grid virtual machine using rootkit and internet worm, SVGrid was able to prevent both of them from maliciously accessing filesystem and network. We also evaluated the performance of SVGrid system and only found that performance cost was reasonable considering the security benefits of SVGrid.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Network operating systems

General Terms

Secure Grid System

Keywords

virtual machine, grid computing

1. INTRODUCTION

MGC '05, November 28- December 2, 2005 Grenoble,France

1.1 Motivation

Grid is designed to enable the sharing of geographically distributed "autonomous" resources. However, sensitive resources could be abused by malicious grid applications. For example, By modifying sensitive files such as system binaries or configurations, an adversary can tamper with sensitive data, install backdoors for system re-entry, or inject virus to disrupt the normal running of the grid computer. By abusing network, a malicious user can use the victim grid computer as a midway to attack other hosts [14]. Many resource owners are therefore reluctant to share their resources to run grid applications for other grid users.

Most grid computing infrastructures rely on the operating system to mediate accesses to sensitive resources. However, intruders can often exploit vulnerabilities in system design and implementation to compromise the operating systems and turn off the OS protection on sensitive resources [16]. Because modern operating systems are usually complex, it is hard to completely eliminate system vulnerabilities and prevent intrusions. In this paper, we are exploring a method that can protect sensitive resources even in a compromised environment. Currently, we focus on the protection of two kinds of sensitive resources: file system and network system.

1.2 Potential Solutions

To prevent abuse of sensitive files, one can deploy sandboxing systems like *chroot* [12], *Janus* [10] and *GridBox* [7] to prevent untrusted applications from seeing the whole system resources. Mandatory access control systems like SELinux [11] and GridPortal [6] also help protect sensitive resources by associating immutable attributes with subjects and objects and performing authorization checks based upon those attributes. However, all these approaches suffer from the same problem: they can be turned off if intruders compromise the operating system and gain system privileges [1]. The file protection they provide is thus less effective in a compromised environment.

Network firewalls such as iptables [13] helps prevent malicious network accesses. However, if the firewall is deployed on a grid computer, it can be disabled by a successful intruder; if the firewall is deployed on a stand-alone device like a router, it is less effective in protecting computers that are directly interconnected and behind the firewall.

Virtual machine monitors such as VMWare [15] and Xen [3] can be extended to isolate grid applications into guest virtual machines so that nothing a malicious application does on its virtual machine can change or damage the real computer. However, files in a virtual machine are still subject

to malicious accesses. Therefore, a malicious grid application can still disrupt subsequent applications that will run on the same virtual machine by modifying sensitive files. In addition, different classes of grid users or applications may have different access permissions. Existing virtual machine systems do not dynamically change security policy according to the grid user/applications running in them, rendering the security policy enforcement less flexible.

1.3 Overview of SVGrid

Aiming at protecting sensitive resources even if a grid user already subverts the system, this paper proposes ***SVGrid***, a **S**ecure **V**irtual **Grid** running environment based on Xen virtual machine (VM) architecture [3]. SVGrid is developed to provide an non-bypassable access control mechanism to regulate the accesses to sensitive system files and networks. Currently, SVGrid is designed for Linux systems, but the same idea can be applied to other operating systems.

SVGrid runs multiple ***Grid Virtual Machines (GVM)***. Each GVM is dedicated to run one grid application at one time. Grid applications can run in their GVM as if they run directly on a standard physical host. No modification on grid applications is needed. The filesystems and network services of all GVMs are moved to a dedicated ***monitor virtual machine***. When a grid application in GVM needs to access files or networks, its request is forced to go through the monitor virtual machine to verify compliance of this request with security policies. With this setting, SVGrid presents the following properties in running untrusted grid applications:

- **Transparency**. While SVGrid moves the filesystem and network service of GVM to the monitor virtual machine, those resources can be transparently accessed by grid applications without modification on application source code. The details will be described in Section 3.

- **Complete mediation**. As the filesystem and network support of a GVM is moved to the monitor virtual machine, all accesses to these resources must go through the monitor virtual machine and are subject to security checking. Therefore, the access control cannot be bypassed.

- **Attack resistance**. The monitor virtual machine is responsible for security policy enforcement. It is securely isolated from GVMs by the Xen virtual machine monitor. Therefore, grid applications in GVMs cannot access or disable the policy enforcer running in the monitor virtual machine, no matter whether they can compromise their own GVMs or not.

We implemented SVGrid and tested it against attacks on GVM using several available malicious softwares. SVFS was able to prevent prevent all of them from modifying sensitive files and accessing unauthorized network hosts. We also compared the application performance on SVGrid to that on a native Linux system and found that performance penalty brought by SVGrid is 8.3% for the experiment workload. We believe that performance cost was reasonable, considering the security benefits of SVGrid.

The remainder of this paper is organized as follows. Section 2 gives background and terminology related to virtual machines. Section 3 presents the design of the system. Section 4 presents the evaluation on effectiveness and performance of SVGrid. Finally, Section 5 presents conclusions and directions for future work.

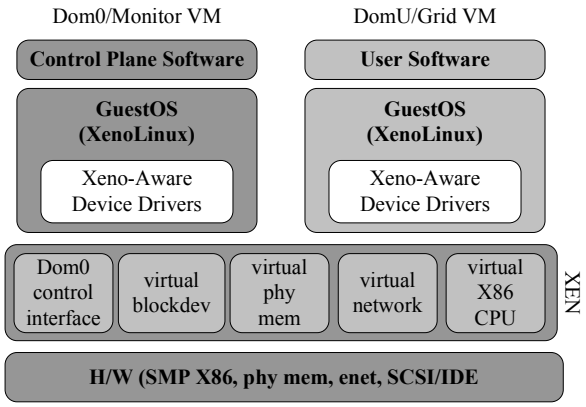

Figure 1: Xen Virtual Machine Architecture

2. OVERVIEW OF VM TECHNOLOGY

SVGrid is built upon the Xen virtual machine infrastructure. Xen is a virtual machine monitor for x86 that supports execution of multiple guest operating systems with resource isolation and performance close to bare machine [3]. As shown in Figure 1, Xen is a thin software layer that runs directly on a physical machine's hardware. It provides each virtual machine with a set of virtual interfaces that resemble direct interfaces to the underlying hardware. Applications on a virtual machine can run without modification as if they were on a dedicated physical machine. Furthermore, Xen provides secure partitioning between virtual machines (known as `domains` in Xen terminology). Guest VMs run as if they are physically separated machines. They can only see and access their own virtual resources, preventing a malicious VM from directly manipulating other virtual machine's resources. There are two kinds of virtual machines in a Xen server: `Domain0` and `DomainU`. Domain0 is a required part of any Xen-based server. It is a privileged virtual machine that can communication with Xen via specified interfaces and control the running of other unprivileged virtual machines, DomainU.

The bootup procedure of guest operating systems is different to that provided by the underlying hardware. Domain0 is created and booted by Xen itself. For all subsequent guest virtual machines (known as DomainU), the analogue of the boot-loader is the `domain builder`, user-space software running in Domain0. The domain builder is responsible for loading a DomainU's OS kernel image and setting up this DomainU's running environment, such as filesystem and network interface. SVGrid uses Domain0 as the monitor virtual machine. Therefore, the monitor virtual machine knows the valid MAC and IP address of all other virtual machines' network interfaces. This allows the monitor virtual machine to check if network packets sent by a GVM contain spoofed MAC or IP address. This capability is used by SVGrid to enforce network access control as described in Section 3.3. Futher details about Xen can be found in [3] and Xen official website: *http://www.cl.cam.ac.uk/Research/SRG/netos/xen/*.

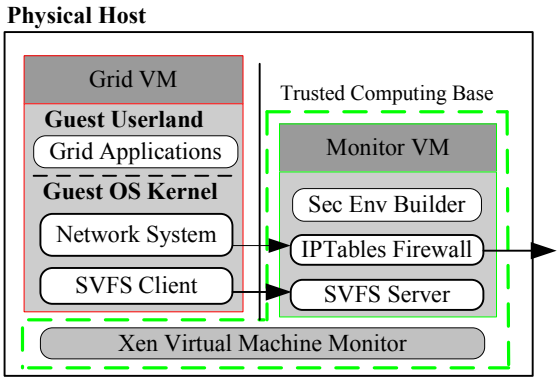

Figure 2: SVGrid Architecture

```
# Sample Customized Policies.  Shows the options
user {abc}              # the policy is customized for grid user "abc"
#Network Description
IP {
  a.b.c.d/netmask       # abc is allowed to access host/subnet a.b.c.d/netmask
}
bind {
  80                                # app can bind to port 80,
  443                               # and port 443
}

#File Access Description
open_rw {                           # app can read these files
  /path/to/file1
  /path/to/file2
  /path/to/dir/*
}
open_a  { }                         # app can append to these files
unlink  { }                         # app can unlink these files
```

Figure 3: A Sample Customized Policy

In this paper, Xen and Domain0 (and thus the monitor virtual machine) are regarded as the trust computing base (TCB) for two reasons. First, Xen is a thin software layer that is primarily responsible for CPU and memory virtualization. It consists of approximately 60K lines of code. Similarly, Domain0 runs a reduced Linux operating system dedicated to providing device virtualization and domain management services only. Compared to a full operating system that typically has several million lines of code, Xen and Domain0 are much simpler and more likely to be implemented correctly [8]. Second, Xen and Domain0 only exposes a very limited interface, which prevents the outside world from directly accessing most of its functionality. For Xen layer, it can only be accessed via predefined hypercalls. Domain0 primarily provides virtual device I/O and domain management services. The virtual device I/O service is delivered to a DomainU via a virtual communication channel established between Domain0 and this DomainU. The communication channel is essentially a memory region shared by Domain0 and DomainU. It can be established only by Domain0 at the bootup stage of the DomainU. Xen protects the communication channel from being accessed by a third party, preventing source spoofing and data tampering over this channel. Currently, Domain0 allows an administrator to use *ssh* or console to log into Domain0 to manage DomainUs. However, there is no technical difficulty to completely disable network access to Domain0 or develop a dedicated daemon for controlling domains remotely. Because the daemon is developed for a single purpose and relatively simple, it is reasonable to believe that this domain management interface can be implemented correctly.

3. DESIGN OF SVGRID

SVGrid is designed to prevent malicious access to system files and networks. As Figure 2 shows, an SVGrid system consists of two major components: the `grid virtual machine` and the `monitor virtual machine`. Grid virtual machine (GVM) is a virtual machine dedicated to run grid applications. An SVGrid system may have one or more grid virtual machines. Each GVM runs one grid application for one grid user at a time. The monitor virtual machine, on the other hand, is responsible for monitoring the network and filesystem activities of a GVM. It identifies the grid user/applicatioin by the GVM ID and enforces appropriate security policy according to the grid user. Currently, the monitor virtual machine is Domain0, the only privileged virtual machine in a Xen server.

To effectively monitor the GVM's filesystem activities and enforce security policies, SVGrid moves the GVM's filesystem to the monitor virtual machine. These files are exported back to GVM with a secure virtual file system called SVFS, which is developed by us. When grid applications need to access files that are stored in the monitor virtual machine, they have to go through the SVFS layer, which happens transparently. Thus, all file accesses from GVM must be checked against security policies by the SVFS server running in the monitor virtual machine. The details of SVFS is described in Section 3.2.

In a Xen environment, all GVM's network communication are forced to go through Domain0 (and thus the monitor virtual machine) [3]. The monitor virtual machine is therefore vantage point to block malicious network traffic. This network protection comes from two aspects. First, as described in Section 2, the monitor virtual machine is able to check the source IP/MAC address of packets originated from a GVM against a trusted database, preventing a malicious grid user from sending packets with fake source IP/MAC address. Second, SVGrid uses `iptables` [13] network firewall to enforce network access control policy and prevent adversaries in GVM from sending malicious packets to attack other network hosts. We will describe the network protection in detail in Section 3.3.

3.1 Secure Environment Setting

SVGrid allows the system administrator to specify a global access control policy on the filesystem and network system used with grid applications. This system policy applies to all grid users. By default, each known grid user have a home directory. The user can create, delete, or modify any files in her home directory. The base system files provided by SVGrid host are set to read-only. This kind of files include operating system files, utilities and Grid supporting systems such as Globus [9] and MPICH [2]. Global policies helps expedite policy configuration for most grid applications.

In some cases, global policies maybe too restrictive for certain grid applications. If those applications are trusted (for

example, they are developed by trusted companies or people), SVGrid also allows system administrators to customize security policies for specified grid applications to meet their requirements. A sample customized policy is shown in Figure 3. The customized policy will be applied along with the system policy. If it conflicts with the global policy, the customized policy overrides the global policy. This allows flexible policy and helps resolve the difficulty when the global policy is too restrictive.

Upon receiving a new grid task, SVGrid runs a `secure environment builder` in the monitor virtual machine to create a new grid virtual machine and setup the running environment to run the incoming grid application. The setup procedure includes two major steps:

1. Exporting appropriate filesystem to the new GVM. The monitor virtual machine maintains a base filesystem to be shared by all grid virtual machines, which includes operating system files, commonly used utilities, and the grid supporting systems such as Globus [9] and MPICH [2]. The secure environment builder always exports this base filesystem to the new GVM. Besides, the secure environment builder also exports the grid user's home directory, which is usually used to hold application data and results.

2. Loading security policies into the SVFS server and network firewall. By default, the sensitive files on base filesystem such as system executable binaries, scripts, configurations and grid supporting utilities are set to be read-only as grid applications seldom need to modify them. Grid users are allowed to create, delete, or modify any files in their home directory. Besides the system security policy, SVGrid also allows system administrators to customize security policies for difference grid users to meet their special requirements. The secure environment builder translates the system policy and the user specific policy into SVFS and firewall rules, which are fed into SVFS server and iptable firewall to mediate resource access requests from the monitored GVM.

After the environment is set up, the secure environment builder starts the grid application in its GVM.

3.2 SVFS Filesystem

SVFS is a single filesystem that stores files for grid virtual machines and allows applications in a GVM transparently access those files as if those files reside in the local filesystem in GVM.

A SVFS filesystem consists of a `SVFS client` and a `SVFS server`. The SVFS client side runs at the kernel level of the guest operating system in GVM. It registers the SVFS file system and hooks itself to the virtual file system (VFS) layer [5]. Upon receiving a request to files on the SVFS file system, the VFS layer determines that the requested file object belongs to SVFS and calls the corresponding SVFS function registered by the SVFS client. The SVFS client then packs the file request parameters and forwards the request to the SVFS server, which runs in the monitor virtual machine, to get service. After receiving response from the server side, SVFS client unpacks the result and sends it back to the VFS layer, giving the requesting application an illusion that the file access is served locally.

The SVFS server runs at the kernel level of the guest operating system in the monitor virtual machine. It established a virtual communication channel with the SVFS client at the bootup stage of a GVM. As described in Section 2, the VM-based communication channel is hard to spoof and eavesdrop, SVFS client and server can use this channel to exchange data without encryption and reliably identify each other to enforce access control policies. Upon receiving a file request from a SVFS client, SVFS server first identifies the source of the request, then verifies compliance of this file access request with security policies, and finally process the request if it is determined to be valid.

With SVFS filesystem support, all grid virtual machines can share a single base filesystem while having their own home directories. This greatly saves the storage space required to run multiple virtual machines. In addition, SVFS allows a system administrator to specify what files can be seen by grid applications, like a sandboxing system. By specifying sensitive files to be read-only to GVM, one can also run less secure grid applications without worrying about their modifications on sensitive files. Because the real files reside at the SVFS server side, all file accesses of a grid application in GVM must go through SVFS server to get checked. The policy enforcement therefore cannot be subverted or disabled by a malicious grid application.

3.2.1 Virtual Remote Procedure Call

Because all GVM files are moved to a separated virtual machine, all file accesses must be done via cross domain communication. Efficient data exchange across domain boundary is very important to the viability of SVFS. Remote procedure call (RPC) is a convenient way to exchange data over network [4]. However, for communication between two virtual machines on a single physical host, it has two shortcomings. First, it involves expensive operations, such as XDR encoding/decoding, packet packing/unpacking and connections establishing, which are unnecessary for SVFS system as the client and server side resides on the same physical host. Second, standard RPC relies on network address to identify the communication party. Unfortunately, IP address is easy to spoof, allowing hostile parties to inject malicious RPC messages.

SVFS introduces the `Virtual Remote Procedure Call (VRPC)` mechanism to support efficient data exchange across domain boundary. VRPC shares the same running model and protocol with the standard RPC, but differs in that VRPC does not rely on network to exchange data. Instead, the VRPC client(SVFS client) allocates a memory region and request the VRPC server (SVFS server) to map this memory region into the memory space of the monitor VM for sharing. The shared memory is used as the communication channel between SVFS client and server. The filesystem access requests and responses are be put into the shared memory region and seen immediately by the other party. This shared memory approach greatly reduces the communication overhead. In addition, this VM-based communication channel is hard to spoof and eavesdrop, SVFS client and server can use this channel to exchange data safely.

The design and implementation details of SVFS and VRPC can be found in [17].

3.3 Network Access Control

The network access control in SVGrid is achieved with

the combination of virtual machine technique and `iptables` network firewall.

In a Xen system, network devices is exported to GVM using a split device driver. The monitor virtual machine, which accesses the physical device directly also runs a *backend* driver, serving requests to that device from GVM. Each GVM uses a simple *frontend* driver, to access the backend. Communication between these domains is composed of two parts: First, data is placed onto a shared memory page between the two virtual machines. Second, an event channel between the two domains is used to pass notification that data is outstanding. Virtual network device services are provided by shared memory communication with a backend domain. From the point of view of grid virtual machines, the backend may be viewed as a virtual ethernet switch element with each GVM having one or more virtual network interfaces connected to it. The shared memory is the VM based communication channel between GVM and the monitor virtual machine.

Because all network communication of a GVM must go through the backend, SVGrid deploys the iptables firewall in the monitor virtual machine, where the backend driver runs, to check all network access communication of GVM and enforce network security policies. The network security checking includes two parts:

1. **Validation**. To ensure that domains do not attempt to generate invalid (e.g. spoofed) traffic, the backend driver may validate headers ensuring that source MAC and IP addresses match the interface that they have been sent from. Because the VM based communication channel is established by Xen virtual machine monitor and hard to be spoofed, the backend driver may safely infer the source virtual machine of a packet by identifying the VM based communication channel over which the packet was sent. As described in section 2, the monitor virtual machine is responsible for building other guest virtual machines including GVM. It thus has those guest virtual machines' full information such as the virtual machine identity, the specified MAC and IP addresses. It is therefore able to check that network packets originated from a GVM is not spoofed. Xen support this checking by turning the "anti-spoof" switch in the startup script of guest virtual machines.

2. **Enforcement of Access Control Policy**. To prevent the network device of a victim grid virtual machine from being abused by intruders to mount further attacks against other hosts, SVGrid restricts the IP addresses and ports that a grid virtual machine can access by deploying `iptables` network firewall in the monitor virtual machine to inspect all network traffic of that GVM. Before starting the grid virtual machine, the secure environment builder loads the iptables rules into the iptables policy engine, which will enforce the loaded rules upon all network traffic of this GVM.

4. EVALUATION

4.1 Security Evaluation

We derived file access control policies from default Tripwire policies that are included in the standard Tripwire distribution. In general, we set system directories and files on the base filesystem to be read-only for grid virtual machines. Grid users can modify any files in their home directory. For network access control, we only allows GVM to talk to the source host of the grid application. In the future, SVGrid will collaborate with grid task dispatcher to allow subtasks of a grid task to communicate with each other, which will make SVGrid more practical to run grid applictions.

We evaluate the effectiveness of SVGrid from two aspects. First, we used the `lrk5` rootkit to emulate a malicious grid application to install trojan version of standard system utilities. Examples of sensitive files that lrk5 tried to modify include `tcpd`, `ls`, and `ps`. For this experiment, we assume that a malicious grid user already gains administrative privileges on GVM and run `lrk5` as a root user. We found that SVGrid was able to stop these malicious modifications on sensitive files immediately, which shows the effectiveness of filesystem protection provided by the SVFS system.

To test the effectiveness of network access control provided by SVGrid, we used a modified internet worm, `mworm`, to emulate malicious grid applications to send out malicious packets to compromise other computers in the grid environment. Experimental result shows that SVGrid was able to block those malicious packets by enforce network access control policies in the monitor virtual machine.

4.2 Performance Evaluation

we evaluate the performance of SVGrid by comparing application performance on SVGrid to that on a native Linux system. The SVGrid prototype runs on the Xen 2.0.5 virtual machine system. The monitor virtual machine (thus Domain0) runs Linux Kernel 2.4.29-xen0. GVM runs as a DomainU with Linux kernel 2.4.29-xenU. The directories exported to GVM reside on a Ext3 filesystem which is mounted with the "data=journal" option and uses a 50 MB ramdisk as the journal device. All experiments were run on a PC with a Pentium IV 3.0MHZ processor and 512 MB of RAM. GVM is configured to use 128 MB of RAM.

For this experiment, we used a Linux kernel build to emulate a grid task since it has the characteristics of most of the computational grid applications we ever used: it is CPU intensive, but also needs to access some files.

In the experiment, Linux kernel build on SVGrid takes 284.65 seconds in total, while it takes 262.80 seconds in a native Linux box. The performance penalty of SVGrid is 8.3% in this case. This shows that the performance cost of SVGrid is reasonable for most computational grid tasks, considering the security benefit from SVGrid.

5. CONCLUSION

In this paper we proposed SVGrid, a secure virtual grid running environment, for running untrusted grid applications. SVGrid works by isolating grid applications in multiple grid virtual machines and deploying a dedicated monitor virtual machine to mediate the file and network requests from the grid virtual machines. SVGrid moves the grid virtual machine's filesystem and network system into the monitor virtual machine to force all file and network access requests to go through the monitor virtual machine, making the security enforcement on those requests non-bypassable and cannot be disabled. To evaluate the effectiveness of SVGrid, we tested it against a rootkit and a internet worm. SVFS was successfully able to prevent both of them. Also, we evaluated the performance of SVGrid system and found that SVGrid was able to provide enhanced security with the

cost of 8.3% performance penalty.

6. ACKNOWLEDGMENTS

We thank the MGC reviewers for the valuable feedback they provided. This work was partially made possible by NSF ITR award ATM 0325332.

7. REFERENCES

[1] Kernel brk() vulnerability. http://seclists.org/lists/bugtraq/2003/Dec/0064.html.

[2] Olivier Aumage, Guillaume Mercier, and Raymond Namyst. Mpich/madeleine: a true multi-protocol mpi for high performance networks. In *IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 51, Washington, DC, USA, 2001. IEEE Computer Society.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[4] A. D. Birrel and B. J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[5] D. P. Bovet and M. Cassetti. *Understanding the Linux Kernel (Ed. A. Oram)*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2000.

[6] Ali Raza Butt, Sumalatha Adabala, Nirav H. Kapadia, Renato J. Figueiredo, and José A. B. Fortes. Grid-computing portals and security issues. *J. Parallel Distrib. Comput.*, 63(10):1006–1014, 2003.

[7] Evgueni Dodonov, Joelle Quaini Sousa, and Helio Crestana Guardia. Gridbox: securing hosts from malicious and greedy applications. In *Proceedings of the 2nd workshop on Middleware for grid computing*, pages 17–22, New York, NY, USA, 2004. ACM Press.

[8] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. Network and Distributed Systems Security Symposium*, pages 191–206, February 2003.

[9] Globus Project Team. The Globus Project, 2003. http://www.globus.org/.

[10] I. Goldberg, D. Wagner, R. Thomas, and E.Brewer. A Secure Environment for Untrusted Helper Applications. In *Proceedings of the 6th Usenix Security Symposium*, pages 1–13, San Jose, CA, USA, 1996.

[11] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the linux operating system. In *Proc. of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 29–42, 2001.

[12] R. McGrath and Free Software Foundation. Chroot c run command or interactive shell with special root directory. The Linux Manual Pages.

[13] NetFilter/IPTables Project Team. What is netfilter/iptables? *http://www.netfilter.org*.

[14] E. H. Spafford. Crisis and aftermath. *Commun. ACM*, 32(6):678–687, 1989.

[15] J. Sugerman, G. Venkitachalam, and B. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2001. USENIX Association.

[16] Michal Zalewski. Remote vulnerability in ssh daemon crc32 compensation attack detectory, February 2001. http://razor.bindview.com/publish/advisories/adv_ssh1crc.html.

[17] Xin Zhao, Kevin Borders, and Atul Prakash. Svfs: A secure virtual file system. Technical Report CSE-TR-512-05, University of Michigan, September 2005. http://www.eecs.umich.edu/techreports/cse/2005/CSE-TR-512-05.pdf.