SOFTICE: Facilitating both Adoption of Linux Undergraduate Operating Systems Laboratories and Students' Immersion in Kernel Code

Alessio Gaspar, Sarah Langevin, Joe Stanaback, Clark Godwin
University of South Florida, 3334 Winter Lake Road, 33803 Lakeland, FL, USA
[alessio | Sarah | joe | clark]@softice.lakeland.usf.edu http://softice.lakeland.usf.edu/

Abstract

This paper discusses how Linux clustering and virtual machine technologies can improve undergraduate students' hands-on experience in operating systems laboratories. Like similar projects, SOFTICE[1] relies on User Mode Linux (UML) to provide students with privileged access to a Linux system without creating security breaches on the hosting network. We extend such approaches in two aspects. First, we propose to facilitate adoption of Linux-based laboratories by using a load-balancing cluster made of recycled classroom PCs to remotely serve access to virtual machines. Secondly, we propose a new approach for students to interact with the kernel code.

Keywords: Undergraduate Operating System Course, Linux Kernel Programming, User Mode Linux, Warewulf Clustering.

1. Introduction

Problem Statement

Laboratories' hands-on experience can significantly improve an undergraduate student's understanding of operating systems. Despite their complexity, "real" operating systems help clarifying lectures by exposing the motivations and roots of the concepts, algorithms and tradeoff being discussed. Doing so can help students "make sense" of facts rather than merely memorizing them (thus introducing deductive and active learning dynamics). Linux kernel source code's availability, ubiquity in server-side applications and its documentation through publications [9][10][11][13] and pedagogical material [7][8][12][24] makes it a natural choice in such an endeavor.

A review of current Linux-based laboratories reveals a strong focus on having students modify the kernel directly (e.g. [12]). This can make updates awkward as a new kernel release may introduce new concepts and algorithms which are not covered in the current laboratories. Such material rapidly becomes obsolete and, just as rapidly, discouraging to update.

This review also reveals that facilitating the adoption of such laboratories is a critical issue too often left unaddressed. These laboratories require students to be granted privileged (if not physical) access to computers. This imposes additional constraints on the classrooms' setting; isolation from the University network, difficulty to use the workstations for other courses, frequent re-installations… Because of this, virtual machines, such as the one provided by the User Mode Linux (UML) project [4][5][6] are becoming more common as a means to enable students to safely and conveniently tinker with kernel internals. Security concerns are therefore addressed and re-installation processes considerably simplified. However, one potential problem still remains; we have been assuming all along the availability of Linux workstations in the classrooms. Let's consider an instructor is interested in a Linux-based laboratory at an institution using exclusively Windows workstations. Chances are that the classroom setup will be assigned to the instructor or that new Linux-qualified personnel will be hired. In both cases, the adoption of such laboratories will depend on the complexity, the initial set-up, and day-to-day administration of the system.

Related Projects

SOFTICE separates system administration issues from pedagogical ones, acknowledging the importance of the former to the success of the latter. We also believe there are alternatives to having students make modifications directly to the kernel code. We propose to depart from and extend current approaches along 3 different axes:

(1) Facilitating adoption by using an easy to administrate cluster of recycled classroom PCs to serve access to UML virtual hosts to any kind of workstation on or off campus.

(2) Facilitating replication by disseminating on the SOFTICE wiki best practices, technical documentation as well as software packages.

(3) Developing new laboratories using a novel approach to kernel-level programming which emphasizes progressive immersion into the kernel code and ease of update.

Paper Organization

The following sections will begin with a discussion of the various choices and pitfalls related to setting up the cluster and virtual machines infrastructure (e.g. cluster management, load balancing and virtual machine technologies). We'll then set the objectives for the OS Laboratories we want to develop, taking into consideration existing work and possibilities offered by our particular setting. We will also introduce a new pedagogical approach to OS laboratories and present early results being developed in our department.

[1] SOFTICE stands for Scalable, Open-source, Fully Transparent and Inexpensive Clustering for Education

2. SETTING UP THE CLUSTER

In order to facilitate adoption of a Linux-based solution by any kind of institution, regardless of their current OS choices, we propose to use a load-balancing cluster made of recycled classroom PCs. The advantage is that our implementation results in having only one machine to administrate regardless of the number of cluster nodes and of the amount of students' workstations accessing it. This section discusses the technological choices we made concerning both the clustering and virtual machine technologies.

User Mode Linux

First and foremost, UML is an open source technology. This guarantees its affordability, its availability, and its flexibility for other institutions willing to replicate, adapt, or extend the clustering infrastructure we are developing in this project. While our experience with the integration of UML in a clustering environment proved challenging, UML is a relatively mature technology which has already been integrated in the 2.6.x Linux kernel source tree. It is also being used by many web hosting and (virtual) server collocation companies [5].

Technically, UML is a port of the Linux kernel to a new architecture. Unlike most ports, this one isn't about making the kernel work on some new hardware. Instead, the objective is to be able to run a Linux kernel as a simple process on top of another Linux Kernel. For this reason, UML is often referred to as a port of the Linux Kernel onto itself which makes it more efficient and elegant than hardware-emulation methods. Simply by recompiling the standard Linux kernel source tree for this new um architecture, we can generate an executable which constitutes the "guest" kernel that students will execute in order to create a new instance of a virtual machine. To complete the virtual system, regular files can be used as storage devices, formatted as a hard drive, and booted from by the UML executable after they have been populated with any Linux distribution.

For all these reasons, UML has already been successfully used in operating system laboratories**.** Several of which confirmed that this added emulation layer was indeed enabling the setting of a secure, sandboxed environment for the students without noticeable loss in term of flexibility [7][8]. The number of already existing UML-based labs has motivated us to go one step further by revisiting with an original pedagogy. The lab material generally used with UML comforted us in the feeling that, independently of our proposed pedagogical approach, other institutions could benefit from shared experience. Setting up a clustering infrastructure facilitates the adoption of a Linux-based solution regardless of the previous OS choices, budget, or flexibility in classroom scheduling. This aspect of the problem is addressed by the choice of an appropriate clustering technology.

Choosing a clustering technology

Another feature that distinguishes SOFTICE from similarly intended projects is that, in order to provide a secure remote access to virtual Linux hosts, we are going to centralize, in a cluster, all necessary computing resources and load-balance their access from a single point of entry. By doing so, students can use simple tools such as X-win32 (windows x-server) and SSH clients to access their accounts and, from there, recompile and test a kernel or run a couple of virtual hosts to study networking or system administration related topics. The cluster will be interconnected by its own local area network using a switch and connected to the internet only through the "master node". The latter will also be providing necessary services such as NFS, intrusion detection, and will load-balance SSH access to the cluster nodes (generally, cheap PCs with large amounts of RAM).

To fulfill these objectives, we reviewed several clustering technologies starting with the Linux Virtual Server (LVS) project. LVS started as a kernel patch (IPVS) which is now mainstream in the 2.6.x kernel series. Its role can be described as implementing a kernel-space level 3 router, which boils down to redirecting network traffic on the basis of application layer information. As of today, LVS is a de facto standard for implementing web farms or high availability services.

LVS implements three load balancing techniques based respectively on Network Address Translation (NAT), IP tunneling (TUN), or Direct Routing (DR). The choice of a specific technique impacts on the cluster topology, scalability, and efficiency. We chose to sacrifice part of the scalability in order to increase security by using NAT. With this approach, the incoming network traffic is redirected to the nodes in a load-balanced fashion while the nodes' responses are redirected in the opposite direction by the master-node. The latter is therefore responsible for rewriting packets' headers in order to maintain the masquerading of the internal nodes. This is why this approach is considered less scalable. As the number of cluster nodes grows, the master node can become a bottleneck if its computing power doesn't match the induced workload. On the other hand, this also limits the exposure of the cluster's nodes to the internet which considerably facilitates the implementation of security measures and the system administration tasks. In such a setting, the cluster nodes don't need a routable IP address and can all be "hidden" inside the cluster thus making intrusions less likely.

The cluster we are using has been designed with a somewhat oversized master-node capacity (dual xeon) to avoid any bottle neck due to the number of nodes. It is also used as a standalone server allowing early testing of our lab material in the classrooms. In practice, we expect that a recent PC (2-3Ghz, mono core) with two network adapters and a good amount of memory (1-2Gb) will be more than sufficient once the load balancing is in place. After deciding on LVS, we investigated ways to further adapt the idea of load balancing to our educational purposes and explored SSI clustering technologies.

Single System Image (SSI) clustering is a technology that aims at presenting to processes the illusion that a cluster of workstations is nothing but a single system with a shared memory address space, shared devices, and in which processes can be seamlessly migrated from one node to the other in order to balance the workload. One of the many challenges involved in a SSI implementation is to keep this illusion coherent even when processes are using IPC mechanics not meant to be used over a network. Besides its obvious advantages in terms of process-level load balancing, SSI clustering also provides a unified technology for many types of clustering needs. In our case, it also enables us to manage an arbitrary number of nodes by simply administrating a unique master-node. This master-node will be responsible for automatically providing any booting cluster node with a kernel, an initial ram disk, and a root file system image through PXE. This makes system administration much more scalable than if we had to install and maintain each node individually. It also makes it easier for an institution to adopt this infrastructure with a minimal system administration overload.

Mosix and open Mosix were the first SSI clustering projects to catch our attention. After preliminary tests on recycled hardware, we realized that these kernel patches follow a very constraining release policy whereby only a specific kernel version is fully maintained. We then searched for alternatives and discovered OpenSSI. The OpenSSI project integrates features similar to (open) Mosix and combines them with the network traffic load balancing capabilities of LVS. We implemented our first full cluster prototype using 2.6.10 kernel's open SSI release. By using the flexibility of the process migration combined with LVS network traffic load balancing capabilities and the single system administration, we were convinced that this is the best infrastructure to support our laboratories.

Unfortunately, as we started the labs development with UML, we realized that having students work on a recent (2.6.x) kernel version on top of the open SSI 2.6.10 kernel was going to prove problematic. As of the time of these experiments (spring – summer 2005), UML had serious problems operating on some kernel versions. This included, naturally, the only version supported by OpenSSI.

While featuring similar capabilities, (open) Mosix and open SSI both showed integration problems with other kernel-based technologies not yet integrated in the mainstream development tree. The first solution we considered was to edit by hand the patches in order to make their modifications co-exist harmoniously. This work involves gaining an in-depth knowledge of the chosen SSI technology and maintaining the patch for every new release of UML or OpenSSI. At the time of these experiences the development of UML was very active in response to many bugs related to its integration to the 2.6.x kernel series. As a result, we decided that this would represent too much of a time investment in the context of this particular project. Besides, early experiments that consisted in benchmarking a kernel compilation over the SSI cluster, revealed, as we expected, that the gain wasn't justifying all these efforts. SSI technologies are based on the fact that processes can be migrated from one node to the other on the basis of the cluster's workload distribution. In many cases, applications can be seamlessly migrated to an idle node as soon as the one they are running on becomes overloaded. In the case of a kernel compilation task, this might not be the case. Building a kernel is identical to building a UML machine, which we expect to be a relatively common task for OS students. Such a task spawns numerous short-lived processes which are each responsible to compile a couple of C files at a time. The duration of these processes is, in fact, so short that they don't last long enough for the SSI load balancing algorithms to migrate them. Therefore, if a student logs in to the master node, all of his/her compilations processes will most likely remain on this node. If many students compile simultaneously their kernel(s), the master node will be overloaded while the cluster node will sit idle and nobody will actually benefit from the SSI capabilities. This observation led us to realize that process-level load balancing wasn't sufficient for our specific application and that we would need to combine it with SSH connections load balancing (e.g. IPVS). The idea is that, as a student logs in to the cluster, he or she will be redirected to an arbitrary cluster node in a round robin fashion. From then on, even if the process-level load balancing is not effective, we at least already managed a rough load balancing by spreading incoming connections over the whole cluster. At that point, we had to revisit our infrastructure and find a new way to obtain the features which our previous experimentations deemed interesting:

1. network load balancing
2. process migration
3. easy cluster management

LVS could easily provide us with the first feature since it's already integrated in stock 2.6.x kernels. Process migration would be the hardest to obtain in so far that it requires having a SSI-enabling kernel patch on the host kernel that is compatible with the UML patch on the guest kernel. Unfortunately, no such patch has been successfully incorporated into any of the mainstream kernels as of this writing. On the other hand, this feature, while undeniably desirable, also proved to be the less critical to the success of one of the most common OS students' activities. Easy cluster management is necessary to maintain a reasonable system administration workload. Requiring interested institutions to commit personnel to administrate the cluster node individually would be a critically discouraging argument against adopting SOFTICE. The Warewulf project provided us with a perfect solution which didn't necessitate any kernel modification.

Warewulf

To quote the Warewulf project's wiki at http://warewulf-cluster.org/: *"Warewulf is a Linux cluster solution that is scalable, flexible and easy to use."* [edited out] *"Warewulf is the first of its kind which elegantly solves many of the problems associated with administration and scalability."* [edited out] *"Warewulf facilitates the process of installing a cluster and long term administration. It does this by changing the administration paradigm to make all of the slave node file systems manageable from one point, and automate the distribution of the node file system during node boot. It allows a central administration model for all slave nodes and includes the tools needed to build configuration files, monitor, and control the nodes. It is totally customizable and can be adapted to just about any type of cluster. From the software administration perspective it does not make much difference if you are running 2 nodes or 500 nodes. The procedure is still the same, which is why Warewulf is scalable from the admins perspective. Also, because it uses a standard chroot'able file system for every node, it is extremely configurable and lends itself to custom environments very easily."*

By combining LVS and the Warewulf toolkit, we obtain two out of three of the features we wanted in our cluster. Warewulf allows for a single master-node to serve a root file system, kernel, and initial ram disk to any cluster node at boot time. It therefore implements the centralized cluster management scheme which SSI systems benefit from. This toolkit, because it is not based on kernel modifications, also turned out to integrate perfectly with both LVS and UML. As of today, our cluster is based on these technologies to provide us with a development environment for the OS labs we will be discussing below.

3. OS LABS: A NEW APPROACH

While educational Operating Systems (Minix, Nachos, Topsy, GeekOS…) enable students to comprehend a complete OS in one semester, they are often bound to implement the simplest algorithms for each component. An alternative approach is to cover fewer components but in a much more realistic way. For this approach, the Linux kernel is increasingly being used to provide students with a "real world" experience on operating systems internals. This is the approach we propose to develop further. Our first step has been to explore the nature of existing laboratories [14][11][13][10][8].The next section will focus on the lessons we learned and how we intend to use them to shape a somewhat different approach

Toward a different OS labs pedagogy

Introducing students to the Linux kernel internals in one semester can be a daring challenge depending on the average programming skills level and the motivation. Regardless of these factors, the Linux kernel source base is sufficiently large nowadays to pose the "where do I start first" problem to anyone not fluent in kernel programming. We will not revisit the arguments of the "real " vs. "educational" OS controversy and explore possibilities to introduce such a large code base to the students while avoiding the most common pitfalls that experience taught us:

First, students can be overwhelmed by the sheer quantity of code. As a result, they focus only on the micro changes required of them and absorb a minimal amount of information from the overall kernel.

Secondly, directing students to make modifications to existing code and guiding them each step of the way can cause them not to develop their "start from scratch" programming skills.

Our response to these concerns is based on the Loadable Kernel Modules (LKM) technology and the examples left by famous Root-kits such as Knark.

Loadable kernel modules are software components featuring initialization and clean-up functions respectively executed when the module is started (loaded into the kernel) and unloaded. These components are compiled against a kernel source tree and dynamically linked to a running kernel thus allowing expansion of a running kernel's capabilities without complete recompilation. This approach is most commonly known for the dynamic loading of device drivers. For our purposes, one of the interesting features of loadable kernel modules is that modules can be of any size and complexity. In an operating systems laboratory, the complexity shouldn't be in writing the code in the module itself but rather in the way the module interacts with kernel's data structures and routines. With loadable kernel modules, students can manipulate well targeted kernel elements while still feeling in control of a relatively small code of their own. Also, the code written in a module is definitively kernel code and therefore exposes the students to the same "comfort level" that kernel programmers benefit from (no stdin, no system calls wrappers…). In pedagogical terms, the focus of the lecture can be easily reinforced by a lab briefing on specific, kernel elements that students' modules will interact with.

We can therefore introduce kernel data structures, system calls and kernel routines progressively and let students understand them from the perspective of "how do I manipulate these structures". This approach has already been explored in Linux kernel books [11] which don't try to provide an exhaustive code commentary or reference but more of a functional description of the kernel addressed and who might want to work with it. So far, we haven't specified how the students' loadable kernel modules are to interface with the kernel; root-kits have been the inspiration behind this interface and the design of the operating systems laboratories we propose.
Root-kits have been using loadable kernel modules [18][19][20] to allow a successful intruder to preserve a way to easily regain root access on a machine he previously broke into. Common tasks achieved by such root-kits include hiding from common system tools processes, directories and files, escalating privileges of any arbitrary processes fulfilling a trigger condition and so on so forth. During the course of the semester, students will work on re-implementing in their own modules which will affect the kernel in some way. Class lectures will cover the kernel internals and related data structures which the students will be exploiting throughout the term. This allows for a naturally paced introduction to the kernel with an interesting twist toward computer security related issues that will expand topics coverage in the OS lecture while further motivating students. Also, since the kernel source tree is introduced in small intervals over the duration of the term, the students will have the ability to discover kernel internals through a series of well-delimited forays designed to familiarize them with the kernel code.

4. LABORATORIES OVERVIEW

This section discusses how the above principles can be applied to develop an OS lab series. As of today, the first laboratories of the series have been developed. The whole set should be available at the project's wiki site by the end of spring 2006. We decided to use Knark [19][21] as a model of LKM-based root-kit for the development of our laboratories.

[Lab-1] System Calls Interception

This first laboratory is meant to let students work inside a UML machine and test some provided code for intercepting the fork system call. Results are shown by running simple bash external commands and observing the shell forking a child process to execute the command. Students can then, by analogy, develop another LKM aimed at intercepting the open system call in a slightly different way. Once these preliminaries are acquired, passing parameters to a module at load time is considered and, a loadable kernel module capable of banning the use of fork (then open) to a specific PID passed at load time is implemented. To conclude, a more generic system call interception module is written that receives load time parameters specifying which system call number to block for which process.

[Lab-2] Hiding a process

The focus is here on process-related kernel data structures and the /dev/kmem kernel memory device. The technology is based on Knark sources and enables students to implement a module capable of hiding information related to a given process from system commands such as ps, top… This activity complements the former by developing it toward a topic that seems to always interest students; security. Furthermore, this module constitutes the very first step of a root-kit like module that students will be building from scratch.

[Lab-3] File System Stacks

This laboratory is an intermediate step used to introduce file systems kernel data structures and use the approach described by Erez Zadock [22][23] to not overwhelm students with the kernel source code for their first forays into this topic. The objective is also to prepare students to interact with the proc pseudo file system which is examined in the next laboratory.

[Lab-4] Communication through ProcFS

This lab focuses on the /proc pseudo file system and teaches students how to register file entries for their modules so that information can be exchanged with them after they are loaded into the kernel. The structure of Lab #01 is used again; first, the LKM intercepting the fork system call is extended to be able to prevent a list of processes from calling fork. The PIDs are visible through /proc/ban-fork.out and new PIDs can be added by writing to /proc/ban-fork.in. The open system call interception module is then modified in a similar manner thus allowing a list of files to be specified by their absolute paths. The module will ensure that these files won't be opened by any process. This part is left to the students as an exercise so that they can repeat the drill by analogy with the source used as example. The loadable kernel module hiding processes is then enhanced using this same mechanics thus enabling to specify at run time the processes to hide / reveal. With a minimal syntax, students can very easily start controlling their loadable kernel module form the command line, or use it within their own user space programs.

[Lab-5] Hiding Files & Directories

The next natural step in stealth is to conceal the existence of files and directories. This lab is the analog of the precedent and starts by introducing the students to the file systems-related kernel data structures. Once again, Knark is guiding the resulting implementations of this laboratory and the object is to provide students with knowledge on one of the many available file systems. In our case, we will focus on the Ext2FS and Ext3FS family. Such project illustrates perfectly how User Mode Linux facilitates experimentations not only on the kernel internals but also on storage devices. It is safe to assume that some students might severely damage an existing file system while investigating the corresponding kernel code. With UML, a simple file will be formatted then mounted from the virtual machine. In case of a crash, restoring the image, or even the whole FS for that matter, is as simple as a copy command.

5. Discussion & Future work

This paper discussed recent advances in the SOFTICE project which ambitions are to provide educational institutions with system administration and pedagogical resources to bring an innovative approach to hands-on activities for CS and IT courses such as undergraduate operating systems courses. An open-source load balancing cluster infrastructure has been evaluated for this purpose and the experience we gained from its implementation herein discussed. To illustrate the new possibilities offered by this first step, a novel approach to undergraduate operating systems laboratories has been presented and its early implementation discussed. The project's wiki site will provide further technical and pedagogical resources (http://softice.lklnd.usf.edu).

We are right now exploring ways to leverage our clustering and virtual machines infrastructure in other courses which benefit from hands-on environments which, just like for operating systems, are problematic to set up and maintain. We want to offer support for replication or development efforts undertaken by the educational community at large. These efforts have resulted so far in experiments with Linux System Administration and undergraduate Networking courses.

We are also starting exploring possibilities to provide automatic installation and management tools to simplify the setup of a similar cluster and the daily system administration tasks. This will help reducing the amount of work involved in "giving a try" at replicating the SOFTICE experiment and thus facilitate its dissemination at large.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation under award number 0410696. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

6. References

[1] SOFTICE project web site and wiki, Alessio Gaspar, http://softice.lklnd.usf.edu/

[2] SOFTICE: Scalable, Open, Fully Transparent and Inexpensive Clustering for Education, Alessio Gaspar, Francois Delobel, William D. Armitage, Arthur Karshmer and Farimah Fleschute, International Conference on Education and Information Systems: Technologies and Applications, EISTA'04, pp. 335-340, July, 2004

[3] SOFTICE: Linux SSI clustering & emulation can facilitate Hands-on Computer Science Education, Alessio Gaspar, Dave Armitage, Farimah Fleshute, Symposium on 21st Century Teaching Technologies, 2005, University of South Florida, March 3 2005, USF, Tampa, FL.

[4] User Mode Linux project web site, Jeff Dike, http://user-mode-linux.sourceforge.net/

[5] User Mode Linux community web site, http://usermodelinux.org/

[6] A user-mode port of the Linux kernel, Dike, Jeff, In Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta, page 63, Atlanta, GA, 2000. Usenix,

[7] A Secured Networked Laboratory for Kernel Programming, J. Mayo and P. Keans, Technical Report TR97-1, Department of Computer Science, College of William and Mary, September 1997.

[8] A Linux-Based Lab for Operating Systems and Network Courses, Linux Journal #41, R. Chapman and W.H. Carlisle, 1997

[9] **Understanding the Linux Kernel 2/e**, Daniel P. Bovet, Marco Cesati, O'Reilly & Associates, 2002, ISBN 0-596-0021309

[10] **Linux Kernel Programming 3/e**, Michael Beck, Harald Bohme, Mirko Dziadzka, Ulrich Kunitz, Robert Magnus, Claus Schroter, Dirk Verworner, Addison Wesley Professional, 2002, ISBN: 0-201-71975-4

[11] **Linux Kernel Development 1/e**, Robert Love, Developers Library, SAMS, 2003

[12] **Kernel Projects for Linux**, Gary Nutt, Addison Wesley, 2001, ISBN 0-201-61243-7

[13] **Linux Device Drivers 3/e**, Alessandro Rubini, Jonathan Corbet, O'Reilly & Associates, 2001

[14] **Operating Systems 3/e**, Gary Nutt, Addison Wesley, 2003, ISBN: 0-201-77344-9

[15] **Operating Systems Concepts 6/e with Java**, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Prentice Hall, 2003, ISBN 0-471-489050

[16] **Modern Operating Systems 2/e**, Andrew Tanenbaum, Prentice Hall, 2001

[17] **Operating Systems, design and implementation 2/e**, A.S. Tannenbaum and A.S. Woodhull, Prentice Hall, 1997, ISBN 0-13-638677-6

[18] Rootkit: Attacker undercover tools, Saliman Manap, Section 2.0: Chronology of Rootkit, http://www.niser.org.my/resources/rootkit.pdf

[19] Alamo: A Linux Forensic Kernel Module, Kelley Spoon, http://www.spoonix.org/software/alamo/ 2001-whitepaper/index.html

[20] Root-kits and integrity, Frédéric Raynal, http://www.security-labs.org/ index.php3?page=454

[21] Knark readme file, online at http://www.ossec.net/rootkits/studies/knark-README.txt

[22] Writing Stackable File Systems, Erez Zadock, 05-2003, **Linux Journal**

[23] FIST: Stackable File System Language and Template, Erez Zadock, http://www.filesystems.org/

[24] Linux in Government: Linux Lab at the University of South Florida Opens Eyes, Tom Adelstein, 12-2004, **Linux Journal**

[25] Warewulf clustering toolkit project web site, Greg M. Kurtzer, http://warewulf-cluster.org/

[26] Open SSI project web site, Bruce Walker, http://openssi.org/

[27] **User Mode Linux**, Jeff Dike, Prentice Hall