Linux Security: Exploring Open Source Security for a Linux Server Environment

Jin Xiong
Robert J. Brenneman
Desh Sharma
Duane Beyer
William Jay Huie
Raj Atchutuni
Sireesha Rudraraju

Table of Contents

PART 1

Introduction

The need for security in today's networked world is overwhelming. With e-commerce, e-mail, information distribution and all the benefits of the explosive growth of the Internet, come destructive attacks, identity thefts, access violations, and worms that propagate across networks to infect thousands to millions of computers. Consider the string of denial of services attacks that struck high profile Web sites such as eBay® and amazon.com® at the beginning of 2000 [3]. They can happen anywhere.

However, there are processes to apply and products to use to help manage vulnerabilities and threats to businesses. The continual process of prevention, detection and reaction can successfully thwart many attacks; iterative testing and updating ensure the latest security fixes are applied; and integrated security products provide the depth that can help prevent different kinds of attacks without compromising too much usability.

Linux® is now available on increasingly more platforms and the use of Linux has grown to hosting services and enterprise applications - now there are editions from different vendors developed for that purpose, that come with capabilities that match or surpass those of proprietary operating systems. With the on demand initiative and growing emphasis on using Linux for business, securing a Linux environment becomes extremely crucial.

The good news is with the open source solutions driven by the growing popularity of Linux, there are increasingly more reliable open source security products designed to protect the Linux environment. Open source security products range from firewalls, to intrusion detection systems, to e-mail encryption and communication protocols.

To take advantage of these open source security products, a study was conducted over a period of 3 months at the IBM Linux Test Integration Center (LTIC). The goal for the security study was to deploy and compare various open source security tools that were available for free in the industry, and provide solution recommendations based on experiences during the study. The study involved the installation, usage, and testing of entirely open source security tools on an already established Linux middleware test environment. The LTIC team broke the security process down into 3 phases, with experimentation, iterative testing and protection done for each phase. As a result, piecemeal open source security tools were combined to form an end-to-end security solution that addressed key areas of a common security policy.

This paper is broken down into two parts. Part 1 provides a high level overview with brief descriptions of the security study. This part also provides solution recommendations based on the LTIC team's experiences during the study. Part 2 acts as a reference for those who want to see details of product implementation and various illustrative examples of the open source security products.

Open Source Security Products Used

During the basic security study the LTIC team experimented with the following open source security tools (not necessarily in this order):

Network Intrusion Detection Systems:

Network intrusion detection systems (NIDS) monitor network traffic and send alerts if necessary. Some NIDS can also be used to help prevent malicious packets from passing into sensitive areas of the network.

- *Snort™ – Snort is the most popular open source NIDS that has a set of rules to detect suspicious traffic.*
- *Hogwash – Hogwash is a relatively new NIDS that acts as an inline packet scrubber, which means it can filter malicious traffic as well as being a monitor.*

Web Intrusion Detection Systems:

Web intrusion detection systems detect intrusions toward a Web server.

- *ModSecurity – ModSecurity is an Apache plugin that processes HTTP requests for suspicious requests before they are seen by the Web server. ModSecurity can alert and help prevent known suspicious traffic.*

Host Intrusion Detection Systems:

Host intrusion detection systems detect attempted attacks on the host.

- *PortSentry – PortSentry detects scans by monitoring local ports and prevents access from detected sources by utilizing various configurable forms of blocking techniques.*
- *Port Scan Attack Detector (PSAD) – PSAD detects scans by parsing local Linux firewall logs and blocks scanning sources by setting local firewall rules.*

Firewall:

Firewalls are designed to keep intruders out and allow authorized traffic through.

- *iptables/netfilter – iptables/netfilter is the standard Linux software firewall. A user can configure a set of iptables rules to log, deny, or permit packets.*

Auditing and Logging:
Logging keeps kernel and application messages for auditing or monitoring purposes. The LTIC team used the standard Linux system logger, syslog, for centralized logging, and the following auxiliary programs to generate analyzable reports:

- *Lire™ - Lire generates reports for a variety of services in multiple formats.*
- *Swatch – Swatch parses logs and can actively take action based on configurable options.*

User Authentication:
User authentication refers to the secure identification of users in order to provide access.

- *Pluggable Authentication Modules (PAM) – PAM is a suite of shared libraries that enable the local system administrator to choose how applications authenticate users.*
- *OpenSSH – OpenSSH is a free implementation of the Secure Shell (SSH) protocol. It provides cryptographic authentication and secure transmission for users to access remote machines.*
- *MIT and Heimdal Kerberos V5 – MIT and Heimdal Kerberos are free implementations of the authentication protocol that provides centralized user authentication in a network of hosts.*

Remote Scan:
Remote scanners scan systems and their services for open ports, potential vulnerabilities and security holes.

- *nmap – nmap is a command line multi-use scanning tool with capability to scan remote machines for open ports.*
- *Nessus – Nessus is an automated remote scanner that determines what services are running, attacks these services are exposed to, as well as recommendations on how to reduce security risks.*

System Hardening:
System hardening is a series of steps taken to make a system and its services more secure.

- *Bastille Linux – Bastille Linux is a GUI tool that hardens the system and provides security information to user.*

Intended Audience

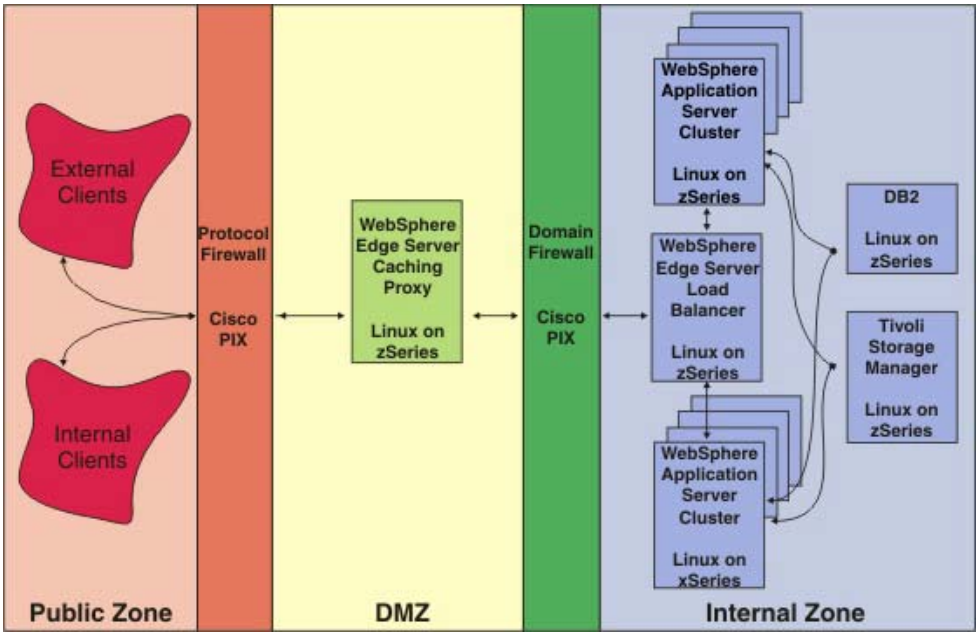
This study's flexible end-to-end open source solution can be referenced for those that are interested in open source security alternatives or options. It can also be helpful for those that are not able to afford the expense of a formal security solution, yet must have a security policy in place. In Part 2 one can find descriptions of the purpose of each tool, what kind of attacks each tool addresses, as well as detailed implementation of each open source security tool.

The paper assumes intermediate Linux knowledge and basic understanding of security issues such as basic security threats and risk assessment. It does not explain how to perform certain processes such as risk assessment and cost analysis. The paper only provides recommendations for best practices and processes as well as detailed description of the security study to be used as a reference.

Environment Description

In order to experiment with different security tools, the LTIC team needed an environment to deploy them on that was realistic and practical. Conveniently, there was an already established Linux middleware test environment available to use as a reference production environment so that the LTIC team could focus on experimenting with the open source tools and security processes. Figure 1 below provides a bird's eye view of the test environment and the rest of this section provides an overview of each component and a description of the workflow.

Figure 1: Existing Logical Environment Diagram



Environment Configuration

IBM's eServer Integrated Platform for e-business on IBM @server® zSeries® provides the capability to quickly and easily order, install, and deploy a robust starter infrastructure for e-business applications. The LTIC team took the basic Internet Topology of the Integrated Platform for e-business V2R1 solution and expanded it to create a high availability configuration. They kept the hardware protocol and domain firewalls from the official design, but clustered the WebSphere® Application Servers (WAS) so that each cluster resided on a different platform thus creating a more fail proof environment. If the IBM @server xSeries® cluster is down they could failover to the enterprise zSeries hardware. Each WAS cluster was managed by a WAS Network Deployment Manager, which handled workload distribution among its cluster members. The Java™ Message Service (JMS) server resided on the same image as WAS Network Deployment Manager, and handled message flow among cluster members. In Figure 1, each cluster represents one WAS Network Deployment Manager, one JMS server, and three WAS cluster members.

For more information on Integrated Platform for e-business, please refer to www-1.**ibm.com**/servers/eserver/zseries/os/linux/integrated/.

WebSphere Application Server and WebSphere Application Server Network Deployment

IBM WebSphere Application Server (WAS) has a set of application services including capabilities for transaction management, Web services, security (although WAS security was not covered in this open source security study), performance, availability, connectivity, and scalability. WebSphere Application Server Network Deployment (WAS ND) offers load balancing, message queuing, and management services to WAS. For the security study, WAS served the Trade3 application.

For more information on these two products, please refer to the WAS library at www-306.**ibm.com**/software/webservers/appserv/was/library/.

IBM WebSphere Application Server Edge Components

- *Caching Proxy reduces bandwidth use and improves a Web site's speed and reliability by providing a point-of-presence node for one or more back-end content servers. It can cache and serve static content and content dynamically generated by WAS. The proxy server intercepts data requests from a client, retrieves the requested information from the servers, and delivers that content back to the client.*
- *Load Balancer creates edge-of-network systems that direct network traffic flow, reducing congestion and balancing the load on various other services and systems.*

For more information on WebSphere Application Server Edge components, please refer to the Edge components InfoCenter at www-306.**ibm.com**/software/webservers/appserv/doc/v50/ec/infocenter/index.html.

WebSphere Workload Simulator

WebSphere Workload Simulator is an automated test tool that simulates multiple Internet browser users for the purpose of testing Web applications and Web servers. It supports HTTP GET (both page elements and pages), HTTP POST, Cookies (specification 0 & 1), Capture and Playback through a Socks server, and Secure Socket Layer (SSL). The WebSphere Workload Simulator can be used to simulate Trade3 transactions. In LTIC's environment, the simulator's controller ran on Microsoft® Windows® NT/2000 and its agents on Linux on Intel®.

For more information on WebSphere Workload Simulator, refer to www-306.**ibm.com**/software/awdtools/studioworkloadsimulator/about/.

Trade3

Generally available WAS/DB2® workload - includes a web-based stock trading application which exploits Java database connectivity (JDBC™), including session-based servlets and Enterprise JavaBeans (EJB). The Trade3 application is installed on WAS, while the workload is generated by the WebSphere Workload Simulator.

For more information on Trade3, please refer to
www-306.**ibm.com**/software/webservers/appserv/benchmark3.html.

Workflow

From the public zone, one of the WebSphere Workload Simulator engines generates a Trade3 request in the form of HTTP or HTTPS to the target URL. This Trade3 request gets sent across the hardware protocol firewall to the Caching Proxy in the Demilitarized Zone (DMZ), which redirects the transaction through the hardware domain firewall to the Load Balancer. In the internal zone, the Load Balancer then takes the transaction and directs it to one of the WAS cells and the Network Deployment Manager then transfers it to the specific server to service this transaction. Upon receiving the request the server retrieves needed information from the database backend, and the information gets bubbled back to the client machine.

Technology Survey

This section provides an overview of the technologies and processes used during the open source security study. The Getting Started section describes the planning steps to take before implementing security measures. The Security Infrastructure section provides an overview of the complete end-to-end security solution the LTIC team eventually assembled. Follow on sections provide overviews of the products and connections added during each phase of the security study in order to achieve the end-to-end solution described in the Security Infrastructure section. For details of each phase please refer to Part 2.

Getting Started

It is imperative to have an open minded approach to security, and to accept the fact that it will never end. Threats to security are constantly changing and evolving requiring constant updates and diligent security policies. Security policies need to be flexible to adapt to an environment of changing/expanding threats. Hence it is important to develop security standards and processes which are audited and reviewed on a regular basis. This section briefly goes over the kinds of processes that should be started in the very beginning to help keep systems secure.

Risk assessment

Each system is different and one set of system goals is different from another. It doesn't make sense to apply the same security policy to every system – one size doesn't fit all. Assess the risks that the system is exposed to and secure only those risks. Risk assessment can be accomplished by running remote security scanners like nmap or Nessus or by hiring a security professional. Most attacks can be stopped by continuously taking a few measures to reduce the risk to an acceptable level. Do not implement unnecessary security mechanisms and strive for simplicity. Each security product or tool should support a defined goal. Read this paper and select the applicable tools based on recommendations and experience descriptions.

Have a robust backup policy in place

It is absolutely a necessity to have a consistent and reliable backup policy in place. Systems must be ready to recover, quickly, when the unthinkable happens. Hard drives wear out, crackers or malicious employees temper with the machines, even system administrators can make mistakes that become irrevocable. Having a robust backup policy can provide peace of mind and facilitate recovery before it is too late and money is lost. Consistent backing up is not just good system administration habit but a secure habit to integrate into any administrative routine.

A good practice is to backup everything (potential targets for attack are ubiquitous; a malicious user can place a corrupt "ls" program that will secretly change file permissions when executed). The backup environment should be completely isolated from the production environment, and backups should be run regularly to save the latest updates. To be really thorough, protected and possibly paranoid; keep a couple levels of backups.

Get the latest software

It is important to use the latest security products because they often contain patches with the latest security fixes. In an already established environment, it is a good practice to upgrade software often, especially if it contains security fixes. Updating goes for Linux distributions as well, Red Hat Enterprise Linux Advanced Server editions have the up2date program which automatically pulls the latest upgrades (such as kernel security fixes). SUSE LINUX's Yast online update is a similar tool for getting the latest upgrades. Incorporate software and operating system updates as part of the security process.

Break the security process into stages

It is a daunting task to try to implement all the different pieces of security at once. The LTIC team decided to break down the process into 3 phases with each phase progressing towards producing a more security-rich environment. Each phase is described in detail in Part 2. Briefly, the security study was broken down into the following three phases:

Phase I: Have the basic infrastructure in place, have all servers and security tools installed and running. Servers should be connected but not necessarily all security tools. Obtain initial list of security checks.

Phase II: Have basic security services are running.

Phase III: Add to security study with enhancements and optimizations. Begin iterative process of continual updates, scans, and fixes.

Have an incident response policy in place

Develop and document an incident response policy that includes reporting guidelines before any security incidents occur. If servers have been compromised, execute the organization's incident response policy immediately. The incident response policy should include contact information of computer security incident response teams (CSIRT) such as The Computer Emergency Response Team (CERT®), and this contact information is best kept offline in case of a network attack.

The CERT home page is www.cert.org. For incident reporting guidelines, see www.cert.org/tech_tips/incident_reporting.html.

The CERT Coordination Center (CERT/CC) incident reporting form is available at the secure Web site https://irf.cc.cert.org.

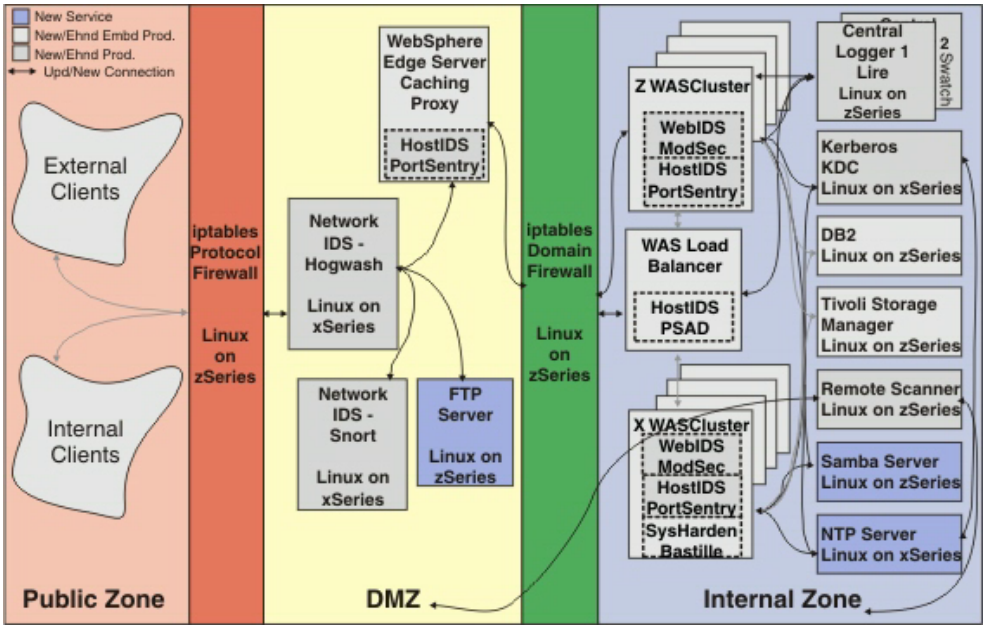
"The Forum of Incident Response and Security Teams (FIRST™) is a consortium of CSIRTs that serve more specialized constituencies [1]". Their home page is www.first.org. Their member list, with applicable constituencies, is available at www.first.org/team-info.

The government has agencies that act as CSIRTs as well, contact them to report activities that fall within their jurisdiction. The National Infrastructure Protection Center (NIPC) home page is www.nipc.gov [1].

Security Infrastructure

In designing the security infrastructure the LTIC team kept in mind the processes talked about above as well as recommended usage of the security products. They attempted to integrate an end-to-end security infrastructure into a realistic production environment. By comparing the topology depicted in Figure 2 below to the original environment in Figure 1, one can easily see that the basic infrastructure is still there. On top of that the LTIC team integrated security provisions such as Kerberos, Hogwash and Snort.

Figure 2: Complete Security Infrastructure Integrated with Existing Environment



The team added the File Transfer Protocol (FTP) and SAMBA servers into the environment to mimic a realistic production environment. The FTP server was placed in the DMZ because its purpose was to serve public FTP requests, but should still be protected to some degree. The Samba server was placed in the internal zone because its purpose was to provide file and printer sharing services to internal systems. The Network Time Protocol (NTP) server was added in the internal zone to provide time synchronization for centralized user authentication with Kerberos.

All other servers, except the Caching Proxy, were located in the internal zone. The internal zone is the most protected zone because it is the most critical. For reasons of usability and practicality, the team chose not to add local firewalls to each of the servers during this study, except for the Kerberos Key Distribution Center (KDC). However, local firewalls could be added for future iterations while accounting for the increase in management it would bring.

The team replaced the two hardware firewalls with Linux iptables/netfilter for firewalling and routing. Hogwash was placed as an inline packet scrubber between the protocol firewall and the DMZ. As an inline packet scrubber it acted as a bridge that sniffed and "scrubbed" packets as they came through. Host IDS was added to the Caching Proxy and Load Balancer, as those two servers had to be extra secure because they were bottlenecks and if they were brought down then even if the Web servers were up the whole system was inoperable. Security is only as strong as its weakest link.

Systems in the Company network segment were initially given SSH access to the internal zone through the firewall. As this creates yet another attack vector, the team only used it during the initial setup portion of the testing. Once the team tightened security the only terminal access to the internal segment was from other systems on the internal segment.

As depicted in Figure 2, the LTIC team took advantage of their mixed environment by deploying the various security tools on mixed platforms and distributions to further mimic a realistic production environment and increase coverage for this study. The following sections in Part 1, Phase I Security Infrastructure Setup, Phase II Security Connections, and Phase III Security Optimizations, provide overviews of components and connections implemented in each phase in order to assemble the complete environment depicted in Figure 2.

Phase I Security Infrastructure Setup

For Phase I, the goals were to have the basic infrastructure in place, with all servers and security tools installed and running, but not necessarily connected. For instance, in this phase the central loggers were set up but other hosts were not configured to log to them yet. As for the basic infrastructure, at the end of this phase, the two Linux routers were to be setup, and all systems should be on appropriate segments and pointing to the correct router. This provided a base environment to run the first security scan on.

Before the LTIC team went into the security study they backed up all WebSphere Application Servers and all WAS Network Deployment Managers entirely with Tivoli® Storage Manager onto a tape drive. Since this backup activity was not specifically part of this study, they did not attempt to use open source tools. Backing up is nonetheless very important as noted in

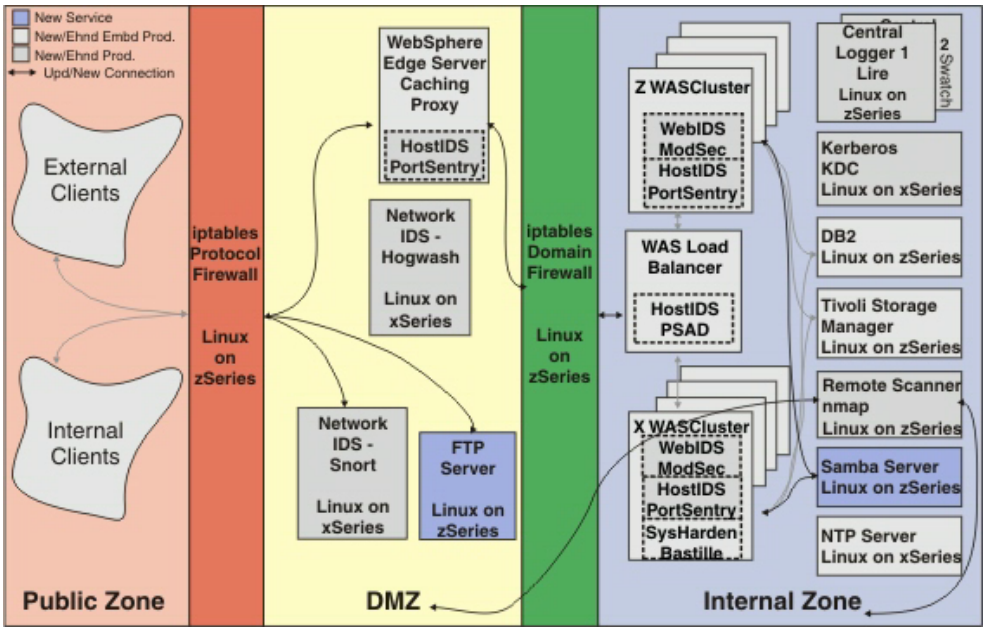
Getting Started, and administrators should have a backup policy along side their security and incident response policies.

In this phase, the LTIC team installed various Linux systems on zSeries as z/VM® guests and various Linux systems on xSeries machines, then installed the standalone security products and services on top where applicable. Host IDS and Web IDS were installed on top of already existing machines. There were a few security products that were used during this phase, but mostly only installations occurred during this time. Bastille Linux was installed and ran on xSeries only. At the time of the study the team discovered a compilation problem trying to run Bastille on SUSE LINUX Enterprise Edition 8 SP3, this problem has since been fixed in a patch for Enterprise Edition 8 SP3 as well as in SUSE LINUX Enterprise Edition 9. Table 1 and Figure 3 below indicate which connections, servers and products were added during this phase. Part 2 describes activities and findings of this phase in detail.

Table 1: Open Source Security Tools and Services Added in Phase I

Open Source Sec. Tools & Misc. Services	Linux Distribution	Platform	Action
Bastille Linux	Red Hat Enterprise Linux AS Release 2.1	xSeries	Installed and ran
Lire	Red Hat Enterprise Linux Release 3	zSeries	Installed
Swatch	SUSE LINUX Enterprise Edition 8	zSeries	Installed
Central Log Servers	SUSE LINUX Enterprise Edition 8	zSeries	Installed and configured
Snort	SUSE LINUX Enterprise Edition 8.1	xSeries	Installed and ran as packet logger
Hogwash	SUSE LINUX Enterprise Edition 8.1	xSeries	Installed
Kerberos KDC	Red Hat 9	xSeries	Installed and configured kerberos KDC and Kerberos admin server
PortSentry	Existing SUSE LINUX Enterprise Edition 8 and Red Hat AS 2.1 systems	xSeries and zSeries	Installed and ran in regular mode on the Caching Proxy and two WASs
PSAD	SUSE LINUX Enterprise Edition 8	zSeries	Installed and ran on the Load Balancer
ModSecurity	Existing SUSE LINUX Enterprise Edition 8 and Red Hat AS 2.1 systems	xSeries and zSeries	Installed and ran on the Web servers
Routers/Firewalls	SUSE LINUX Enterprise Edition 8	zSeries	Setup correct routing
Remote Scan with nmap	SUSE LINUX Enterprise Edition 8	zSeries	Scanned for open ports and removed unnecessary services
FTP Server	SUSE LINUX Enterprise Edition 8	zSeries	Installed and ran
SAMBA Server	SUSE LINUX Enterprise Edition 8	zSeries	Installed and ran

Figure 3: Phase I Added Tools and Connection



Phase II Security Connections

In Phase I, the team installed all their services and security products and had most of them running. They scanned their machines for any obvious vulnerability and disabled many insecure unnecessary services from running. In Phase II, the team's main goal was to have basic security services running and connected for as appropriate. This involved:

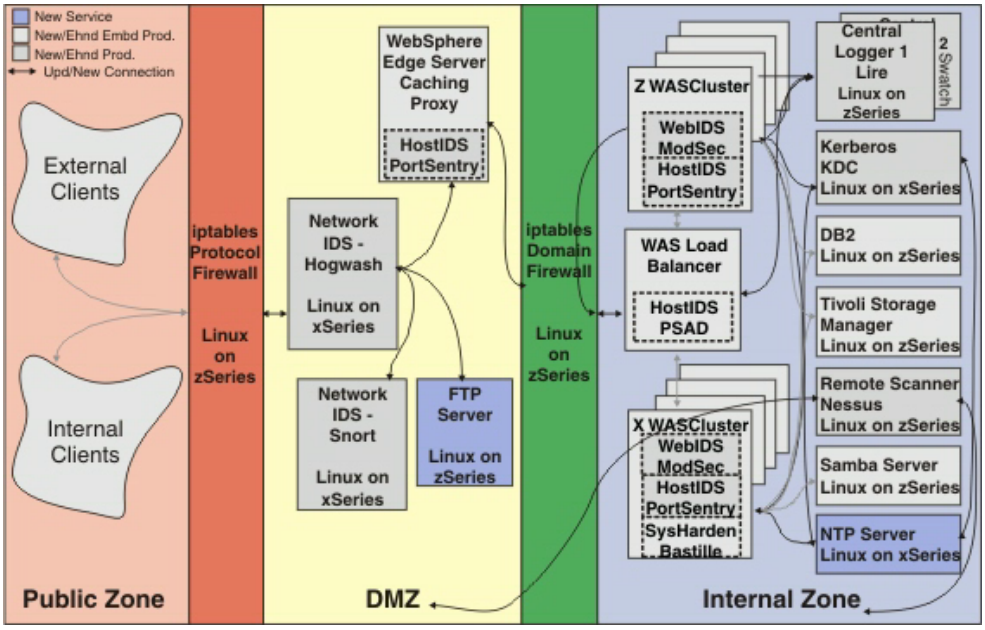
- *Securing/having risk assessment for vulnerabilities found and not closed in Phase I*
- *Interconnecting security tools*
- *Centralizing logging*
- *Having firewalls and auditing in place*
- *Possible rescanning and re-securing to patch up holes*

As Table 2 and Figure 4 below indicate, the LTIC team completed all the connections in this phase in addition to adding the Network Time Protocol (NTP) server. They removed the direct connection from the protocol firewall to the Caching Proxy and FTP server, and replaced this connection with Hogwash so that it now acted as an inline scrubber that not only detected malicious packets but prevented them as well. Additionally, the team connected other machines to remotely log to the central log servers. They also configured server machines to use the Kerberos Key Distribution Center (KDC) for user authentication. Minor optimizations in other areas were added as well. For details of this phase please refer to Part 2.

Table 2: Open Source Security Tools and Services Added or Updated in Phase II

Open Source Sec. Tools & Misc. Services	Linux Distribution	Platform	Action
Central Log Servers	SUSE LINUX Enterprise Edition 8	zSeries	Configured other machines to transmit their logs to central log servers
Snort	SUSE LINUX Enterprise Edition 8.1	xSeries	Ran as Network IDS
Hogwash	SUSE LINUX Enterprise Edition 8.1	xSeries	Configured and ran as inline packet scrubber
Kerberos KDC	Red Hat 9	xSeries	Configured other machines to use PAM-Kerberos for system-wide centralized user authentication
PortSentry	Existing SUSE LINUX Enterprise Edition 8 and Red Hat AS 2.1 systems	xSeries and zSeries	Ran in stealth mode
PSAD	SUSE LINUX Enterprise Edition 8	zSeries	Explored other PSAD options and configurations
ModSecurity	Various	Various	Investigated performace issue
Firewalls	SUSE LINUX Enterprise Edition 8	zSeries	Setup iptables rules on both firewalls
Remote Scan with Nessus	SUSE LINUX Enterprise Edition 8	zSeries	Installed Nessus and scanned/tested internal zone machines. Fixed applicable vulnerabilities.
NTP Server	Fedora Core Release 1	xSeries	Configured local NTP Server and configured all Kerberos machines to sync to it

Figure 4: Phase II Added or Enhanced Tools and Connections



Phase III Security Optimizations

The LTIC team was almost ready to go into production and have the security process down for this iterative and continuous journey to maintain an acceptable security level. Their goals for this phase were to:

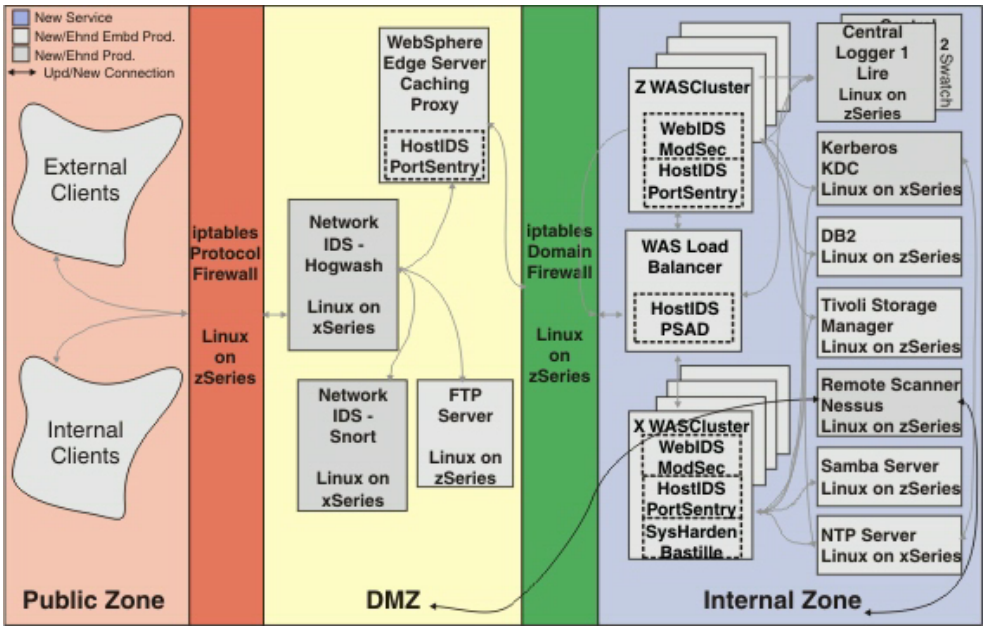
- *Enhance and experiment as time allowed*
- *Rescan for vulnerabilities*
- *Produce and document security process to maintain acceptable security level*

In Phase III the LTIC team did not go into every security area as they did in Phases I and II. Because of time constraints they were only able to go into a few areas for enhancements and/or further study. For details of this phase please refer to Part 2.

Table 3: Open Source Security Tools and Services Added or Updated in Phase III

Open Source Sec. Tools & Misc. Services	Linux Distribution	Platform	Action
Lire	SUSE LINUX Enterprise Edition 8	zSeries	Generated log reports
Swatch	SUSE LINUX Enterprise Edition 8	zSeries	Generated log reports
Hogwash	SUSE LINUX Enterprise Edition 8.1	xSeries	Studied two Hogwash examples that used rules to filter out malicious packets
Kerberos KDC	Red Hat 9	xSeries	Studied another Kerberos configuration used for centralized user authentication. Investigated Heimdal Kerberos
Remote Scan with Nessus	SUSE LINUX Enterprise Edition 8	zSeries	Scanned from public zone and patched applicable holes

Figure 5: Phase III Added or Enhanced Tools and Connections

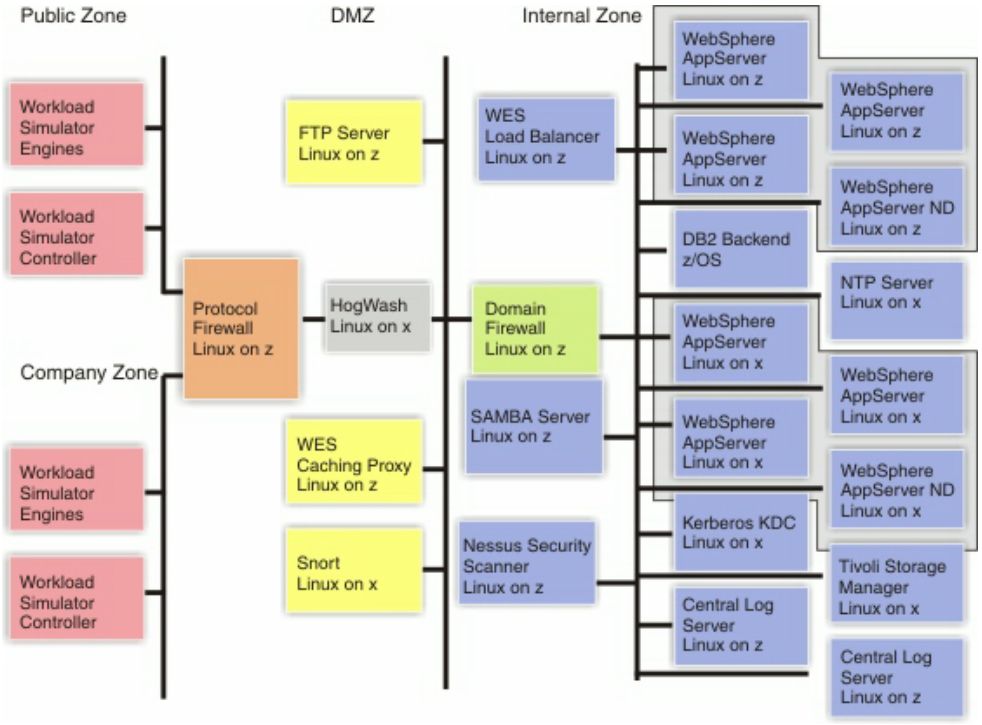


Summary and Recommendations

Summary

The goal for the study was to deploy and compare various open source security provisions that are designed to protect a networked environment. As depicted in the network layout below, the LTIC team devised an integrated security solution that employed multiple readily scalable open source security products to provide an end-to-end security strategy for their middleware environment.

Figure 6: Complete Security and Middleware Setup



Solution Recommendations

Security Processes and Practices

This section reiterates some of the practices that were found to be critical in order to maintain a robust level of security and confidence:

- *Develop a thorough backup policy if one does not exist. Backing up is the most important piece of the security process because systems have to be always prepared for an attack.*
- *Maintain the latest security fixes for applications and operating systems. Do this routinely and diligently. Other updates are good too but additional or improved functionalities are also where potential loopholes and vulnerabilities reside. Take into consideration whether a new feature is worth the effort to revalidate security.*
- *Develop an incident response policy. In case systems are compromised, one needs be ready to respond and take action. Responding to an attack may involve other site administrators or even law enforcement.*

- *Break the security process into stages. Don't attempt to do it all at once. Use the iterative process of defense, detection, and prevention. Follow the security plan developed.*
- *Remember to install only the minimal services that the system actually needs in order to function. Remove all other unnecessary services and connections because they are potential vulnerabilities that attackers can exploit.*
- *Actively monitor system activities. Even though certain IDSs provide prevention capabilities, they are usually only as good as the administrator monitoring them. During the 22-hour eBay outage of 1999, their IDS set off alarms constantly, but their staff was too busy to respond on time [2], and the prolonged outage resulted in financial loss as well as loss of customer confidence. Another problem that IDSs face is distinguishing valid data from malicious data. False alarms can be annoying and blocking valid traffic is even worse, both can be especially difficult to avoid with the sophistication of today's attack tools. Hence, it is a good practice to always monitor IDSs carefully and pay close attention to the alerts they are configured to send. Otherwise they are futile and potentially dangerous. In addition, be sure to keep up to date with the most recent versions and advancements of IDSs.*
- *Rescan all systems routinely for new vulnerabilities with the latest scanners (for Nessus, get the latest plugins). Rescan after new or upgraded software is installed. Patch new vulnerabilities found by rescans and verify them in order to maintain an acceptable security level.*

Comparison of Security Tools

For certain aspects of security, more than one tool was used in order to compare and contrast them to determine their relative strengths and weaknesses. Additionally, security is something best done in depth. Providing diversity in the tools allowed the LTIC team to layer their defenses in many directions. None of these tools are exactly the same, and there may be situations where one fits better than another. Aside from the observations the team made throughout each phase (see Part 2), here are some comparisons to consider when making decisions about which tool to use in a given situation:

1. Snort vs. Hogwash (Network Intrusion Detection Systems):

   Snort has been around longer than Hogwash, and has better documentation and more available rule sets. Snort also has undergone a longer development and test period. So if one is looking for a more mature and better supported tool – go with Snort. Use oinkmaster for automatic Snort rule updates (for more details refer to Part 2). Hogwash, on the other hand, is relatively new so there aren't as much documentation and support. Hogwash seems to have a more fledging development team so they could take more time to respond to issues.

Hogwash has the ability to run as an inline scrubber to detect and prevent. The team realized that the environment would probably be more robust if Hogwash was placed before the protocol firewall. Hogwash didn't have an IP address thus it couldn't be targeted for remote attacks like the protocol firewall.

Thus, Hogwash can go unnoticed on the network as it does its job. Snort does not have this ability to transparently bridge two interfaces and scrub packets along the way.

Snort can run in multiple modes: sniffer, packet logger and NIDS – providing great flexibility for one tool.

In summary, for a more mature and tested NIDS with better documentation and support, then Snort is the way to go. However, if filtering is also desired and more resources are available to work on NIDS, then Hogwash is the right choice.

2. PortSentry vs. PSAD (Host Intrusion Detection Systems):

   PortSentry binds to ports or monitors them stealthily. Thus it takes a more active approach to monitoring port activities. On the other hand, PSAD depends on the robustness of iptables rule set to drop unwanted packets. If one doesn't want to implement local firewalls, then PortSentry is the way to go. If local firewalls are already in place, PSAD is the better choice. PortSentry can complicate administration of any firewall because the firewall administrator has to make sure not to block packets to ports PortSentry is configured to monitor.

   PortSentry has three auto-blocking options: iptables, TCP wrappers, and re-routing unwanted traffic to bogus networks. PSAD can only block via iptables/netfilter. Therefore if one needs more flexibility in blocking mechanisms PortSentry is the right choice.

   PortSentry does not consider the source port used in potentially malicious packets whereas PSAD provides information about both source and destination ports of the packet. Thus, PSAD is a better tool for investigative and auditing purposes.

   PortSentry does not have a count of scans whereas PSAD keeps track of the scans by maintaining a counter. This is again good for investigative and auditing purposes.

   PortSentry cannot detect any probes that utilize the ICMP protocol whereas PSAD can detect ICMP scans as long as iptables is configured to drop ICMP packets. Detecting ICMP scans is an advantage that PSAD has over PortSentry – it provides better detection coverage.

   In summary, PortSentry provides more auto-blocking techniques for those looking for flexibility in their security configurations while PSAD can only be as effective as the local firewall. However, if a robust local firewall is already in place, then PSAD offers more auditing and investigative features and has a wider coverage, and thus is a better choice.

3. Lire vs. Swatch (Auditing Tools):

   Lire provides support for individual services such as FTP and Web servers, as well as syslog; whereas Swatch supports only syslog and keywords specified in the configuration file. If one wants to audit only a specific service, then Lire makes this very easy to do. If one wants to audit for certain types of messages that come from multiple services, then Swatch is more flexible.

   Lire can generate reports in many more formats than Swatch such as HTML and PDF– which may be important for certain processes.

   Lire is more user friendly than Swatch and is faster to learn and to deploy. Swatch is useless without customization to suit the environment and user's needs. For beginners and those seeking faster deployment, Lire would be the right choice.

   Swatch can run actively, watching logs as messages arrive whereas Lire only does reporting and analysis after messages have been logged – although this can be remedied by setting up Lire cron jobs.

   In summary, Lire suits beginners and general users. Those seeking ease of use, variety in services audited, and variety in report formats will also find Lire a good match. Swatch is better for advanced users who want to customize their auditing process and don't need extra the features that Lire offers.

4. Standard local PAM vs. centralized Kerberos (User Authentication):

   The major difference between PAM and Kerberos authentication is that PAM can only be used for local authentication, whereas Kerberos can centralize authentication and make user administration easier in large environments with numerous machines.

   PAM comes already configured with most Linux distributions, whereas Kerberos requires manual configuration, some times manual installation, and extensive infrastructure and planning to support. It makes sense to use Kerberos for larger environments because while Kerberos takes more work to setup, it may cost less in the long run.

   Kerberos is more complex than PAM so PAM is best for small environments with a few machines. For example, Kerberos has an administration utility, kadmin, as well as kpasswd and various other utilities that take time to learn, while PAM has a uniform infrastructure that can be used for applications system-wide – all it takes is to learn how to use PAM for one application.

   PAM is the same between SUSE LINUX and Red Hat; whereas MIT Kerberos comes with Red Hat and Heimdal Kerberos comes with SUSE LINUX and they are not fully compatible in all areas (see Part 2 for details). This disparity definitely dampens the convenience of Kerberos in a heterogeneous environment where both Red Hat and SUSE LINUX are used. An administrator should take the difference between MIT Kerberos and Heimdal Kerberos into consideration as well because the differences will become cumbersome to manage in time. In a heterogeneous environment it is best to use one implementation of Kerberos; if the one chosen isn't the default on some systems then manually installing it will suffice.

With Kerberos one has to worry about time synchronization and security lockdown of the Kerberos Key Distribution Center (KDC), with PAM that's not an issue.

In summary, an administrator would find Kerberos easier to use when tens or hundreds of machines are in the administrative domain. While planning and implementation of Kerberos is far more complex than PAM, the total cost of ownership for large environments can be lower. On the other hand, home users and small environments comprised of a few machines would find PAM a better and simpler match.

Miscellaneous Recommendations

1. The LTIC team used VSWITCH for both Linux on z/VM routers/firewalls in order save on hardware resources. However, they ran into a problem where some packets were being duplicated on the network (see Part 2 Phase I A note on using VSWITCH for Linux routers for details). The recommendation at the time of the study is that when using a Linux router on the mainframe, using a guest LAN is the better option. The router virtual machine would have a virtual NIC on the guest LAN, and real sub channels on the OSA. This would eliminate duplicate packets being bounced on the network. Workarounds as well as fixes are also explained in detail in Part 2 Phase I.
2. Take advantage of system logger's ability to centrally log messages and use auxiliary programs like Lire and Swatch to monitor all system activities instead of using host IDSs to monitor logs on individual machines. By using centralized logs one can get a global look at all systems and still receive alerts on malicious activity on individual machines without having to locally maintain separate host logs and reports.

PART 2

Technology Details

This section provides all the implementation details of each product and examples that were used for each phase in the open source security study. Details and examples can be used as a reference for those who want to use the open source security tools that the LTIC team used.

Phase I Security Infrastructure Setup

System Hardening

The defensive process of system hardening involves many steps. The first step is to cut down unnecessary and insecure services. Then apply various techniques to help increase the security level of the services that must run, and finally secure access controls for the root user. Hardening tools can also educate and advise users as to why certain services are insecure and what they can do to help secure them later on.

The LTIC team used Bastille Linux for System Hardening. On Red Hat Web servers, the team installed it manually, and on SUSE LINUX the team attempted to use the Bastille Linux packaged with the distribution. For Red Hat they downloaded the Bastille-2.1.1-1.0.i386 and perl-Tk-800.022-11.i386 rpm's from the Bastille Web site: www.bastille-linux.org/, and installed them using the rpm –Uvh. One can also use CPAN to install perl-Tk/perl-Curses. Consult the Web site's tables for the right Perl version for each distribution. For SUSE LINUX Enterprise Edition 8 SP3 on zSeries the team attempted to use the Bastille Linux (version 1.3.0) that came with the distribution but found that even though it installed successfully, it had various problems during the execution of the BastilleBackEnd script. Details of this problem can be found in bugzilla #5695: https://bugzilla.linux.ibm.com/. After the study ended and during the writing of this paper the bug was fixed in a patch for SUSE LINUX Enterprise Edition 8 SP3 and Enterprise Edition 9. The rpm that has the Bastille compilation fix is bastille-2.1.1-32.3.s390.rpm.

On Red Hat the system hardening process was successful. One has to run Bastille on a system that have X Window System or can export or forward the display. After installation, the team ran bastille as superuser and answered its questions based on their experiences and understanding of system suggestions relative to the environment. Not all questions and answers apply to every environment and one should use Bastille at one's discretion by providing appropriate answers to applicable questions. Below is a list of questions Bastille asked the LTIC team and their responses to those questions:

1. Would you like to set more restrictive permissions on the administration utilities? **Yes.**
2. Set User ID root permissions. You may change it later with `chmod u+s filename`.
   - Would you like to disable SUID status for `mount/umount`? **Yes.**
   - Would you like to disable SUID status for `ping`? **Yes.**
   - Would you like to disable SUID status for `at`? **Yes.**
   - Would you like to disable the r-tools? **Yes.**
   - Would you like to disable SUID status for `usernetctl`? **Yes.**
   - Would you like to disable SUID for `traceroute`? **Yes.**
   - For x-wrapper? **Yes.**
3. Disable clear-text r-protocols that use IP-based authentication? **Yes.**
4. Enforce password aging? **No.**
5. Restrict the use of cron to admin accounts? **Yes.**
6. Set default umask? **Yes.** What mask would you like to set for users on the system? 027.
7. Should disallow root login on tty's 1-6? **No.**
8. Would you like to password protect the GRUB prompt? **Yes.** Then typed in a password.
9. Would you like to disable CRTL-ALT-DELETE rebooting? **No.**
10. Would you like to password protect single-user mode? **Yes.** Then chose to use root password.
11. Would you like to set a default –deny on TCP wrappers and xinetd? **No.**
12. Should Bastille ensure the telnet service does not run on this system? **Yes.**
13. Ensure that FTP doesn't run? **Yes.** Unless this is a FTP server, then No.
14. Would you like to display "authorized use" messages at login time? **Yes.** Refer to http://ciac.llnl.gov/ciac/bulletins/j-043.shtml on creating login banners.
15. Who is responsible for granting authorization to use this machine? ***Admin Name.***
16. Would you like to put limits on system resource usage? **No.** If you want to change this later, `edit /etc/security/limits.conf`, and set number of `core files = 0`, `number of processes = 150/user`. If you restrict the resources available in this manner, you can effectively cripple some DoS attacks.
17. Restrict console access to a small group of user accounts? **No.**
18. Additional logging? **Yes.**
19. Do you have a remote logging host? **Yes.** Then typed in the IP address of one of the central log servers (see the Auditing section).
20. Misc Daemons:
    - Disable apmd? **Yes.**

- Disable pcmcia? **Yes.**
- Disable gpm? **No.**
- Stop sendmail from running in daemon mode? **Yes.**
- sendmail via cron? **No.**
- Disable VRFY and EXPN sendmail cmds? **Yes.**
- Bind Web server to listen only to localhost? **No.**
- Bind Web server to a particular interface? **No.**
- Deactivate the following of symbolic links? **Yes.**
- Deactivate server-side includes? **Yes.**
- Disable CGI scripts, for now? **Yes.**
- Disable indexes? **No.**

21. TMPDIR:

    - Install TMPDIR/TMP scripts? **No.**

    - Run packet filtering script? **No** -> This skips the PSAD configuration which the LTIC team had already manually installed on some systems. See the Host Network Intrusion sections for more details on PSAD.

22. Save configuration and run this configuration! Bastille logs are in `/var/log/Bastille/action-log and /var/log/Bastille/error-log`.

One can revert back to old settings with the `bastille -r` command. Reverting back is a nice option to have if a mistake was made during configuration.

Network IDS

Network intrusion detection systems (NIDS) monitor packets on the network wire and attempt to discover if a cracker is attempting to penetrate or exploit a system (or cause a denial of service attack). A typical example of a NIDS is a system that watches for a large number of TCP connection requests to many different ports on any machine within its jurisdiction, thus discovering if someone is attempting a TCP port scan. An agent-based NIDS runs on the target machine and watch its own traffic (usually integrated with the network stack and services themselves), and a sniffer based NIDS runs on an independent machine strategically positioned to watch all network traffic. NIDS have rule sets that specify which packets will get logged where and how, and what actions to take for different types of packets. As a brief example one can set rules for Snort, an open source IDS, to alert the administrator via e-mail that a possible denial of service attack is underway.

The two open source NIDS that the LTIC team studied were Snort and Hogwash. Running NIDS can degrade network performance, so in their environment they decided to install them on dedicated machines. NIDS on dedicated machines also provided single administration points for keeping NIDS rules. In Phase I the LTIC team performed the necessary steps in installing and running these tools. They put both Snort and Hogwash in the DMZ so that

they could monitor all traffic from the external world that passed through the protocol firewall to enable the traffic was logistically legitimate traffic – the NIDS didn't have to see any garbage that the protocol firewall could filter out.

A note on running in promiscuous mode:
Promiscuous mode allows a network device to intercept and read each network packet that arrives in its entirety on a network segment. Agent based NIDS do not need to run in promiscuous mode because they only monitor local traffic. However, sniffer based NIDS like Snort must run in promiscuous mode in order to capture traffic on the whole segment. The LTIC team hosted their NIDS on xSeries (Intel) machines. Linux restricts the use of promiscuous mode to the superuser, so during network interface setup always run as root. Red Hat and SUSE LINUX distribute kernels with support for the packet socket protocol enabled. For those distributions that don't, one should rebuild the kernel with the option `CONFIG_PACKET=y (or CONFIG_PACKET=m` to build a kernel module).

Then, one needs to run the following command to setup promiscuous mode on a network interface:

`ifconfig` *`interface`* `promisc, i.e. ifconfig eth0 promisc`

To turn off promiscuous mode, run:

`ifconfig` *`interface`* `-promisc`

Getting started with Snort
Snort is an open source NIDS that can run in three modes: sniffer, packet logger, and full fledged Network Intrusion Detection mode. As a sniffer it taps everything on the wire, as a packet logger it saves the packet information to disk (it still collects every packet it sees), and as a NIDS it will log those packets specified by the rule sets, and more importantly act on the rules.

For Phase I the LTIC team ran Snort in packet logger mode. In Phase II they examined Snort as a NIDS along with its rule sets and configurations.

Snort is included in SUSE LINUX distributions but not Red Hat. For manual installation, download the source tarball from www.snort.org and install it according to instructions on their documentation Web page. The LTIC team used Snort 1.8.7 that came with SUSE LINUX Enterprise Edition 8.1 for Intel which they minimally installed on an xSeries machine. One of the differences between the Snort that comes with SUSE LINUX distributions and the downloaded one is in the location of its configuration files and rule sets. In SUSE LINUX, they are located under `/etc/snort`, and in manual installation they are

located under `/usr/local/share/rules`, unless otherwise specified during manual installation.

To add Snort as a service that starts on boot time, in SUSE LINUX run `chkconfig snort on`, in manual installations create a script in `/etc/init.d/` and then use `chkconfig`.

For Phase I the LTIC team ran Snort in packet logger mode:

```
snort -devl /var/log/snort
```

The above options logged all the headers (`v` option), all application data (`d`), all link layer data (e) and logged packets (l) to the `/var/log/snort/src_ipaddress` directory.

To test Snort running in packet logger mode, they generated external Trade3 workloads and observed on the Snort machine that in `/var/log/snort/`, log directories were formed corresponding to the source IP addresses of the traffic. Within these directories were files where packet contents were stored. File names were denoted by protocol:`src_port-desc_port # (i.e. TCP:59478-80)`.

Note: the man page for Snort advises against the usage of `-v` in NIDS mode because it is really slow and thus inadvertently drops packets. For more information on snort options, see man page for snort or the snort user guide on www.snort.org.

Getting started with Hogwash
Hogwash is a NIDS/packet scrubber. Being a scrubber means that Hogwash can detect attacks on a network, and has the option to filter them out. As an inline scrubber Hogwash simply takes packets from one interface and sends them out on the other, similar to a bridge, but only after Hogwash has had a chance to filter them through its rule sets. Hogwash uses Snort's detection engine (rule sets) but also has the available option to drop packets. If one wants Hogwash to run as a NIDS/packet scrubber all the time, the LTIC team recommends using Hogwash on dedicated high bandwidth interfaces and machines with lots of computing power.

The LTIC team set up the Hogwash machine in stealth mode and installed Hogwash. They did not attempt to run Hogwash during the first phase because the Linux firewalls weren't setup at the time. In Phase II, they setup Hogwash to run in inline scrubber mode and examined the Hogwash configuration file and rule sets, and put it into production. In Phase III they studied two Hogwash examples.

To run Hogwash at system boot time, use the same technique described in the Getting started with Snort section above.

The LTIC team set up the Hogwash machine differently than Snort by putting it in stealth mode. What stealth mode means is that the Hogwash machine is running on unconfigured interfaces (no IP address), thus would not be a direct target for network attacks.

Installing Hogwash:
Unlike Snort, there aren't as much good Hogwash documentation available. What follows is a more specific description of the team's installation procedure for Hogwash H2 V2.1:

1. Download hogwash tar file (devel-0.5-latest.tgz) from http://hogwash.sourceforge.net/download.html into a directory
2. Unpack the tarball `tar -xzvf devel-0.5-latest.tgz`
3. Compile the source: `cd distro/devel-0.5/devel-0.5/` and execute `./configure script` and then run `make`.
4. Copy the executable "hogwash" command to a directory in root's $PATH such as `/sbin`, `/usr/sbin/`, or `/usr/local/sbin` and optionally strip by running `strip -s hogwash`. Stripping the binary executable removes all the debugging symbols that an attacker could use to gain information about other systems in the environment and reduces the size of the binary to help enhance performance.
5. Copy the desired rules to use with hogwash to rules' target directory (when Hogwash is executed one specifies where the main `.rules` file is, in that file one can specify other `.rules` files, so it's best to keep all the `.rule` files in one place, preferably with the configuration file).

Web IDS

Web Intrusion Detection Systems (Web IDS) detects attacks toward a Web server. There are two basic types of Web IDSs, some are similar to Host IDSs in that they analyze the access logs generated by the Web servers and detect Web server attacks (Tivoli Risk Manager). Others, like the one used for this study (ModSecurity for Apache), come in the form of a plugin that pre-processes HTTP requests and detects and prevents incoming Web application attacks.

ModSecurity
ModSecurity is an open source Web intrusion detection and prevention engine for Web applications. It operates embedded in the Web server and supports both branches of the Apache Web server. ModSecurity scans incoming HTTP requests for patterns that match one of a list of possible attacks, and denies or logs the request if it matches anything in the rule list. The list is derived from the current list of Snort rules which are available on the Web (www.snort.org, or see the Snort sections of this paper), and can be modified as needed. For example, one common attack would be to generate a HTTP request that contains 10,000

repeated '/' characters in the URL. This could crash a Web server if it tries to process such a request. A ModSecurity rule could be written to detect and prevent any request with more than 5 '/' characters in the URL if there will never be more than 5 in a valid request, and thus would prevent the request from being processed by Apache.

In the LTIC environment, the team installed the ModSecurity plugin on all their Web servers. This provided a last line of defense against HTTP type attacks targeted at these Web servers even if they were from the internal network segment. If an attacker managed to get an invalid request through the Hogwash filter via a combination of attacks, or request obfuscation, or some other technique, then ModSecurity could have one last chance to prevent a feral HTTP request from doing damage. ModSecurity also serves to provide some level of protection from compromised machines, or an inside attack. If, for example, an attacker managed to get the Caching Proxy to generate a dangerous HTTP request, ModSecurity could prevent it from being processed.

In Phase I the team installed ModSecurity. In Phase II they investigated some issues with this tool.

Installation and execution:
If using multiple distributions, the LTIC team suggests tackling this one distribution at a time. Below are the steps that were followed in installing ModSecurity on one distribution.

Do steps 1-5 on one of the Web servers, then steps 6 - 8 on the rest:

1. Download mod_security from www.modsecurity.org/download/index.html.
2. Download mod_security reference manual: www.modsecurity.org/documentation/index.html
3. Unpack code and use `/$webserverhome/bin/apxs` to compile it as a dynamic shared object (DSO). Detailed information for this part can be found under the Installation section of the reference manual. Note: The apxs script may have to be edited to reflect the correct location of the Apache libraries and Perl interpreter. Otherwise apxs may not work out of the box.
4. Edit `/$webserverhome/conf/httpd.conf` to add the ModSecurity settings and filter list:
   ```
   LoadModule security_module    libexec/mod_security.so

   AddModule mod_security.c
   ```
5. Restart apache (now it will run with the mod_security settings).
6. Copy `mod_security.so` from `/$webserverhome/libexec/` to the other Web servers' `/$webserverhome/libexec/` directory.
7. Edit `httpd.conf` on the other Web servers to add the mod_security settings and filter list.

8. Restart apache on the rest of the Web servers.

In `httpd.conf`, the rules look like this:

```
<IfModule mod_security.c>

# Turn the filtering engine On or Off

SecFilterEngine On

#Rejects any request with 3 /'s in a row. Note that #each / is escaped
#with a \ to prevent any regular expression expansion.

SecFilter "\/\/\/"

</IfModule>
```

The default rules that come with ModSecurity are rather skimpy. There is a more robust rule set derived from Snort rules here: www.modsecurity.org/documentation/converted-snort-rules.html. The converted Snort rules are quite long, ~840 rules. It is quite a list to manage, and it did not parse correctly. Some characters such as '.' and '?' have to be escaped with a leading '\' character.

Host IDS

A Host Intrusion Detection System (Host IDS) can offer protection for the host in many ways. Some will look at the local system logs for evidence of malicious activity and monitor key system files for evidence of tampering. Others monitor suspicious application activity in real time. There are many different kinds of Host IDSs out there and many of them are hybrid systems: a combination of host-based and network-based IDSs. If a cracker is trying to do a port scan or trying to attack the system by changing the system files, host IDS tools which are specifically configured to detect such intrusions will log the messages and send alerts if necessary.

The open source security tools the LTIC team chose to investigate are PortSentry and PSAD. Both of these tools detect port scans on a host system. They chose them since most network attacks to a host are usually preceded by a scan generated from an automated tool such as nmap. For this study, PortSentry was installed and configured on the Caching Proxy and two Web servers, and PSAD was installed and configured on the load balancer.

For Phase I, the team installed and ran PortSentry and PSAD with basic configurations. In Phase II, they further investigated runtime and configuration options for PortSentry and PSAD.

PortSentry
PortSentry is one of the Sentry tools which provide host-wide security services for the *NIX (UNIX, Linux, FreeBSD, etc) platform. PortSentry detects scans on a host and implements responses to them in one of three ways specified in its configuration file: 1) re-route detected traffic to bogus IP address or network, 2) add firewall rules to block scanning IP, and 3) add TCP wrapper rule to deny scanning IP from using services protected by TCP wrappers. Additionally, it can log to the system logger (for information about the system logger refer to the Auditing and Logging sections).

PortSentry provides the auto-blocking option, to automatically block suspicious IP's upon detection using one of the three methods talked about above, and it provides extensive support for stealth scan detection as the LTIC team found out after the first scan (see Phase I's Remote Security Scan section).

Installation and execution:
In Phase I the LTIC team was just trying to find ports vulnerable to attack so they did not run PortSentry in auto-blocking mode. Below are the steps that they followed in order to install and run PortSentry:

1. Download and unpack the source code from packages.debian.org/unstable/net/portsentry.html. (LTIC team downloaded portsentry-1.2)
2. Follow instructions given in the `README.install` file that came with the package. Accept the default for everything, except for the following:

   In `portsentry.conf`, change the value of BLOCK_UDP and BLOCK_TCP to "0". They are set to "1" by default. A value of "1" means when ever some one is trying to scan the ports which the PortSentry is set to monitor, block those ports; "0" means do not block and only log.

   In `portsentry.conf`, comment out KILL_HOSTS_DENY="ALL: $TARGET$". This option writes the IP address of the target which is trying to do a port scan to the `hosts.deny` file for TCP wrappers to handle.
3. Compile source code:

   ```
   make linux
   ```

   Note: Compiling it using make generic provides only TCP and UDP modes, and the four stealth modes are not provided.
4. Install PortSentry:

   ```
   make install
   ```

   This installs PortSentry in the default `/usr/local/psionic/portsentry` directory.
5. Run PortSentry in basic port-bound TCP mode:

```
./portsentry -tcp
```

and in basic port-bound UDP mode:

```
./portsentry -udp
```

PortSentry startup messages are in the log `/var/log/messages`, one should see all the ports customized for PortSentry to listen to in `portsentry.conf`; if a port is in use PortSentry will give a warning that it couldn't bind to it and will continue on until all the other ports are bound. For example, if sshd port is in use PortSentry gives this alert:

```
Nov 14 14:23:19 litxwas02 portsentry[14511]: adminalert: ERROR:
could not bind TCP socket: 22. Attempting to continue
```

PSAD
The Port Scan Attack Detector (PSAD) is a tool consisting of lightweight system daemons written in Perl and in C that are designed to work in conjunction with the Linux firewalling code (iptables in the 2.4.x kernels, and ipchains in the 2.2.x kernels) to detect port scans and other suspicious traffic. Because PSAD analyzes information generated by iptables/ipchains log messages, its effectiveness is only as good as the logging rules included in the firewall rule set.

PSAD daemons (as explained in the PSAD man pages):
The psad daemon configures syslog to write all kern.info messages from `/var/log/messages` to a named pipe `/var/lib/psad/psadfifo` and then reads all messages out of the pipe that are matched by a regular expression designed to catch any packets that have been logged by the firewall.

The kmsgsd daemon reads all messages that have been written to the `/var/lib/psad/psadfifo` named pipe and writes any message that matches a particular regular expression (or string) to `/var/log/psad/fwdata`.

`psadwatchd` is a software watchdog that will restart any of the other two daemons should a daemon die for any reason. More information on psad can be found in the man page that comes with the installation.

Firewall configurations:
Usually the best way to setup the firewall is with a default "deny and log" or "drop and log" rule at the end of the INPUT chain, and then insert rules above this to allow valid traffic through.

The LTIC team installed and ran PSAD on the Load Balancer, thus they designed the local firewall to allow incoming packets through to the Web server ports, as well as ports for the

Load Balancer itself. The team used DROP for everything else, and logged all the dropped packets. iptables logged dropped packets to `/var/log/messages`, which were processed by PSAD daemons to determine what level of scan had been initiated against the machine.

Installation and execution:
The LTIC team followed these steps to install and execute PSAD.

1. Download and extract the latest source code for PSAD from www.cipherdyne.org/psad/download/. (At the time of the study, it was psad-1.2.4)
2. Edit psad.conf to suit desired environment and setup. Make sure that all paths to binaries are correct. The LTIC team left all the defaults except for the following:

   Change the value of EMAIL_ADDRESSES field to which ever email-id PSAD is to send alert messages:

   ```
   EMAIL_ADDRESSES             administrator@logmachine.com;
   ```

   Change the value of FW_MSG_SEARCH to whatever string the firewall rules prefix the dropped packet with (this corresponds to the `-j LOG --log-prefix="DROP"` part of iptables):

   ```
   FW_MSG_SEARCH               DROP;
   ```

3. Run the `./install.pl` script that comes with the package. This should result in a functional installation of PSAD on the system with the configuration set up in step 2. The PSAD daemons are installed in `/usr/sbin` by default, and an init script is also installed in `/etc/init.d`.

   Note: One can install a new version of PSAD over an existing one; just run `install.pl` again. The installation script will preserve any old configuration parameters when installing the new versions of `psad`, `psadwatchd`, and `kmsgsd`. If one doesn't need or want any old configurations to be preserved, just execute `./install.pl -n`. PSAD can be completely removed from the system by executing `./install.pl --uninstall`.
4. Run `/etc/init.d/psad` to start `psad`, `kmsgsd`, and `psadwatchd`.

A brief example:
The LTIC team set up the local firewall to drop all ICMP packets, so that when they tried to ping the Load Balancer the following messages showed up in `/var/log/psad/fwdata`:

```
Nov 24 09:18:44 litlb01 kernel: DROPIN=eth0 OUT= MAC= SRC=192.168.70.8
DST=192.1
68.71.97 LEN=84 TOS=0x00 PREC=0x00 TTL=62 ID=24936 DF PROTO=ICMP TYPE=8
CODE=0 I
D=28005 SEQ=1
```

```
Nov 24 09:18:45 litlb01 kernel: DROPIN=eth0 OUT= MAC= SRC=192.168.70.8
DST=192.1
68.71.97 LEN=84 TOS=0x00 PREC=0x00 TTL=62 ID=24944 DF PROTO=ICMP TYPE=8
CODE=0 I
D=28005 SEQ=2
Nov 24 09:18:46 litlb01 kernel: DROPIN=eth0 OUT= MAC= SRC=192.168.70.8
DST=192.1
68.71.97 LEN=84 TOS=0x00 PREC=0x00 TTL=62 ID=24945 DF PROTO=ICMP TYPE=8
CODE=0 I
D=28005 SEQ=3
```

Running PSAD status caught the packets sent:

```
litlb01:/etc # psad -S
 .. psadwatchd (pid: 19942)  %CPU: 0.0  %MEM: 0.1
    Running since: Mon Nov 24 09:50:01 2003

 .. kmsgsd (pid: 19940)  %CPU: 0.0  %MEM: 0.1
    Running since: Mon Nov 24 09:50:01 2003

 .. psad (pid: 19938)  %CPU: 0.0  %MEM: 2.7
    Running since: Mon Nov 24 09:50:01 2003
    Command line arguments: [none specified]
    Alert e-mail address(es): sireesha@us.ibm.com

    Global packet counters:
        tcp:  0
        udp:  0
        icmp: 3

    [No scans detected]
```

Firewall

Having a firewall is essential in any environment to keep intruders out and allow authorized traffic through. It acts as a boundary between the private network and the vast public. The LTIC team had two firewalls, one protecting their entire environment from the public and another to protect the internal production part from the semipublic part/DMZ. The first boundary was referred to as the protocol firewall and the second as the domain firewall. For visual firewall placements refer to Figure 2. In Linux, netfilter/iptables is the standard for constructing firewalls, and is included with most distributions. The LTIC team replaced the Cisco® PIX™ firewalls and replaced them with netfilter/iptables.

For Phase I, the team installed the firewalls and set up the correct routing. In Phase II they setup iptables firewall rules for both firewalls and investigated how they help keep intruders out and allow legitimate traffic in.

Router setup:
The LTIC team performed the following steps to setup firewalls to do the proper routing in this phase.

Note: One must be superuser in order to setup routing and firewall in Linux.

1. Install Linux minimally for firewall usage (no X Window System, no extraneous services). The distribution this study used for both firewalls is SUSE LINUX Enterprise Edition 8 SP3.
2. In order to use the firewall as a router the following needs to be configured:

   1) Set up Linux to act as a router:

   Since the LTIC team was running Linux as a guest on z/VM 4.4 using a VSWITCH (see A note on using VSWITCH for Linux routers below for cautionary comments) they needed to do the following to get IP Forwarding to work:

   a) In the VSWITCH definition, set the VSWITCH to be a PRIROUTER (Primary Router) in z/VM. One can query this with the QUERY VSWITCH command and see if it is NON-ROUTER or PRIROUTER.

   b) In `/etc/chandev.conf`, add `primary_router` after the port name for both interfaces. Like this:

   ```
   noauto;qeth0,0x0200,0x0201,0x0202;add_parms,0x10,0x0200, \ 0x02
   02,portname:PRV71,primary_router
   ```

   2) Make sure IP_FORWARDING is turned on (place it in a network startup script):

   ```
   echo 1 > /proc/sys/net/ipv4/ip_forward
   ```

   Or edit `/etc/sysconfig/network` so that FORWARD_IPV4 is set to true.

   3) Build the routing tables:

   a) Add a default route statement to the domain firewall that points to the protocol firewall:

   ```
   route add default gw protocol-firewall-DMZside-IP
   ```

   b) On the protocol firewall add a route back to the internal LAN thru the DMZ interface on the protocol firewall:

   ```
   route add -net internal-lan-network/netmask gw \
   domain-firewall-DMZside-IP
   ```

   c) On every DMZ node, add the following route statement for correct routing:

```
route add default gw protocol-firewall-DMZside-IP
```

d) On every internal node, add the following route statement for correct routing:

```
route add default gw domain-firewall-internalSide-IP
```

A note on using VSWITCH for Linux routers:
As described earlier in Router setup, the team used the same VSWITCH for both Linux routers/firewalls. However, since not all Linux systems that the two Linux routers were routing to were using VSWITCH; duplicate packets were being generated for systems that were not native to the VSWITCH. The problem was that using PRIROUTER results in inbound packets with a destination IP address not registered by any member of the VSWTICH. This is normal if one of the members is acting as a virtual router for another network. But the VSWITCH doesn't know which guest is the router (because all packets are sent to the same real MAC address), so it gives a copy to all members. In most cases, only one of those guests knows what to do with the packet. The others go "huh?" and bounce it back to their default router. If that's the router that originated the packet in the first place, the packet could ping-pong until TTL expires. And, of course, there isn't just one game of ping-pong, there are 'n-1' games (if n hosts are connected to the VSWITCH). It's a geometric progression.

The team investigated this issue at the time of the study and discovered that the recommendation from z/VM developers is that when using a Linux router on the mainframe, using a guest LAN is the better option. The router virtual machine would have a virtual NIC on the guest LAN, and real sub channels on the OSA. This could eliminate duplicate packets being bounced on the network. If using VSWITCH for a Linux router is still desired, then one can avoid this problem by not having the VSWITCH as a PRIROUTER, but instead have the Linux router ProxyARP the guests behind it on a guest LAN. This would not be a good solution for lots of guests, but for a handful, it tends to help eliminate some of the problems that arise using PRIROUTER. z/VM Development is also currently working on various ways of addressing the above issue; such as issuing warning messages, providing recommendations, and delivering code for MAC-based communications.

Auditing and Logging

About the system logger

Linux provides a system logger daemon (`syslogd`) which can log system messages locally and remotely. Logs of system activities used as part of an organization's auditing process provide information about the state of the system. Additionally, these logs provide useful information for detecting suspicious activity and can be easily parsed into readable formats with auxiliary programs like Lire and Swatch for further investigation.

The `syslogd` daemon collects messages from programs and the kernel. These messages are tagged with a facility that identifies the broad category of the source such as `mail`, `kern` (for kernel messages), or `authpriv` (for security and authorization messages). Additionally, a priority specifies the severity of each message. The lowest priorities are `debug`, `info`, and `notice` and the highest priority is `emerg`. The complete set of facilities and priorities are described in the man pages for `syslog.conf` (system logger's configuration file) and `syslogd` [1].

Messages can be directed to different log files, based on their facility and priority; this is controlled by the configuration file `/etc/syslog.conf`. The system logger conveniently records a timestamp and the machine name for each message [1].

For example, in `syslog.conf`, one can have all emerg messages print directly to the console:

```
*.emerg                                 /dev/console
```

For local logging, specify in `/etc/syslog.conf` the desired types and priorities of messages as well as locations to log messages to:

```
news.crit                          -/var/log/news/news.crit
news.err                           -/var/log/news/news.err
news.notice                        -/var/log/news/news.notice
```

For remote logging, configure the central log servers with the `-r` option in `/etc/sysconfig/syslog`:

```
SYSLOGD_OPTIONS=".... -r -m 0..."     #Red Hat
SYSLOGD_PARAMS="...-r -m 0..." #SUSE LINUX
```

Note the `-m 0` option. This tells the system logger to log immediately. If one wants to log in intervals then one would set this to a number greater than 0 to specify the interval in minutes. The default time is approximately 20 minutes and the remote logging port is UDP 514.

On all other machines that are sending messages to the central log servers, edit `/etc/syslog.conf`:

```
*.*                                     @loghost1
*.*                                     @loghost2
```

Parameters `loghost1` and `loghost2` specify the hostnames or IP addresses of the central log servers. If there is more than one central log server configuration may be specified like above. Having two central log servers provides many flexible usage options. For example, each central log server can be used for one type of message or priority. Another option is to specify both log servers to log the same messages so that they serve as backups for one another.

The benefit of remote logging is that it provides a central auditing point that allows for scalability and another level of security for local log files in case attackers try to delete local logs in an attempt to hide traces of hacking activity. However, when using the system logger for remote logging, by default there is no encryption or authentication of the messages sent to the central logger, and this is certainly a risk that should be taken into consideration when designing the logging infrastructure. Secure Socket Layer (SSL) is a good protection protocol to use in this case.

Even though the system logger is really powerful, one should also be careful about using facility and priority to sort system messages simply because they are subjective and may be misleading. However, they are useful for restricting different file permissions and performing different rotation treatment for different log files generated based on the types of messages each receives [1].

What the LTIC team did
The team planned to have two central log servers in the environment and all other machines (all servers and all services) remotely send their logs to both central log servers. That way, each log server also serves as a backup for the other in case one log server goes down. To be really paranoid and more secure, delete all the log files on the local machines and only rely on the central log servers for information about every machine. However, in that case one should also install local firewalls to help protect the central log servers. For long term and continuous auditing, one should also rotate system logs routinely to avoid maxing out disk capacity, and remember to follow the backup policy and perform routine backups of logs.

In the LTIC environment, the two central log servers were placed in the internal segment along with the Web servers. This places them under the protection of two layers of firewalls and in the trusted zone.

The team used auxiliary programs Lire and Swatch, to scan the log files collected on the central log servers, and to generate clean looking reports that help locate suspicious activities on the machines.

For Phase I, the team performed the necessary activities to get the security products installed. Since the team was not going to use Lire and Swatch as real time log analyzers, they did not run Lire and Swatch right after installation. The LTIC team configured the system logger to receive logs from remote machines and ran `syslogd` on both central log servers to bring the system loggers up. In Phase II, the team configured the rest of their machines to log to the two central log servers. Lire and Swatch's abilities to generate reports were studied in Phase III.

Getting started with Lire
Lire is an open source reporting and analysis tool developed and maintained by the Log Report Foundation. Currently Lire can generate reports for a variety of e-mail, Domain Name Server (DNS), Web server, firewall, system logger and even printer log files. The list of Lire-supported services is growing every day. Reports can be generated in a variety of formats from HTML to PDF.

Lire can run as an online responder client, as a cron driven system, or as a command line driven system. In this study, Lire was used as a command line driven system to generate reports on an ad hoc basis. However, if one wants to use Lire as an automatic auditing tool continuously, then having a cron driven setup that reads and processes log files after they're rotated is a better choice. When a cron job is used to generate Lire reports, make sure there is an administrator that reviews the reports periodically.

For Phase I, the LTIC team minimally installed Red Hat Enterprise Linux 3, downloaded the source tarball for Lire from www.logreport.org/lire/dl.php, and followed instructions on that Web page to install Lire.

Of the prerequisites, the team had to install the XML::Parser, DBD::SQLite, and Digest::MD5 modules, before installing Lire. They then built Lire version 1.4 from the directory which it was unpacked:

```
[root@litlog02 lire-1.4]# ./configure --prefix=/lire
[root@litlog02 lire-1.4]# ./make
[root@litlog02 lire-1.4]# ./make install
```

Lire provides RPM's for certain distributions and should be installed that way if the distribution is supported. For a list of Lire RPM's refer to the Lire website at `http://www.logreport.org`.

For more information, including full list of supported services and report formats, and installation options, refer to the Lire documentation: `http://logreport.org/doc/lire/`.

Getting started with Swatch
Swatch is another analyzing tool similar to Lire, except Swatch can actively scan log entries as they are written, and inform users what is happening and take actions such as e-mailing the administrator. Swatch matches the logs to regular expressions specified in its configuration file and generates reports and takes actions based on any matches it finds.

The LTIC team installed Swatch on their second central log server. They first installed the minimal SUSE LINUX Enterprise Edition 8 SP3 operating system, and then downloaded the latest source tarball from `www.cert.org/security-improvement/implementations/i042.01.html`. The release used in this study is 3.0.8. For prerequisites, the team installed the following perl modules from CPAN (Note: you can also have the `Makefile.PL` script that comes with the Swatch package automatically install the prerequisites for you):

- `perl -MCPAN -e 'install Date::Calc'`
- `perl -MCPAN -e 'install Date::Parse'`
- `perl -MCPAN -e 'install File::Tail'`
- `perl -MCPAN -e 'install Time::HiRes'`

The LTIC team then ran the following commands to build Swatch from the directory in which it was unpackaged:

```
litlog01: ~/swatch-3.0.8 # perl Makefile.PL
litlog01: ~/swatch-3.0.8 # make
litlog01: ~/swatch-3.0.8 # make test
litlog01: ~/swatch-3.0.8 # make install
litlog01: ~/swatch-3.0.8 # make realclean
```

The Swatch executable was installed in `/usr/bin`.

User Authentication

User authentication is when a user logs into a system or performs an operation on a system, the user is who he/she claims to be. User identity can be a username, SSH key, or Kerberos credential. The three user authentication methods the LTIC team used were Pluggable Authentication Modules (PAM), Secure Shell (SSH) and Kerberos.

PAM

PAM is a suite of shared libraries that enable the local system administrator to choose how applications authenticate users. Using PAM, one can seamlessly modify the local authentication system without touching the applications themselves. Therefore it's possible to switch to another authentication mechanism without rewriting and recompiling a PAM-aware application. Thus, one central PAM configuration file can be used for multiple applications to produce a system wide authentication scheme. Most recent Linux distributions and all the ones used in this study, already come with PAM authentication configured for most system applications like `su` and `passwd`.

In Phase I, the LTIC team used the default PAM authentication scheme that came with the distributions so that programs such as `su`, `passwd`, and `sshd` all used PAM authentication. The configuration files for these programs and others are located under `/etc/pam.d/`. For more information on PAM settings and files please refer to the Linux-PAM website at http://www.kernel.org/pub/linux/libs/pam/.

SSH

SSH is a protocol that provides strong cryptographic authentication for users to access remote machines. It also provides authentication and encryption for network connections. It's an appropriate technology for many secure networking tasks. OpenSSH, a free implementation of the SSH protocol, is included in most Linux distributions.

The LTIC team strictly used SSH for remote login and the SSH protocol in the form of scp or sftp (both are client programs included with OpenSSH) to copy files between two systems. The team disabled less reliable programs such as `rlogin`, `telnet`, and `ftp`. In Phase I, SSH authentication was handled by the default local PAM authentication configuration. In Phase II, SSH authentication was handled by PAM-Kerberos.

Kerberos V5

Kerberos is another authentication protocol that can provide user authentication. It is designed to provide strong authentication for client/server applications by using secret-key cryptography. Kerberos involves a centralized authentication database maintained on one or more highly-secure hosts acting as Kerberos Key Distribution Centers (KDCs). Principals acting in a Kerberos system (users, hosts, or programs acting on a user's behalf) obtain

credentials called "tickets" from a KDC, for individual services such as remote login, printing, etc. Each host participating in a Kerberos "realm" must be explicitly added to the realm, as must each user.

There are two open source implementations of the Kerberos V5 protocol readily available, MIT and Heimdal. This study covered mainly MIT's implementation of Kerberos V5, and looked into a little of Heimdal's implementation.

For Phase I, the team simply installed and configured the KDC and the Kerberos administration server on a dedicated machine, and made sure that they worked properly and were up and running. The team wanted to make sure all systems were up so as to validate open ports at the end of the phase. In Phase II other systems were configured to take advantage of Kerberos authentication. In Phase III the team looked at another Kerberos configuration and attempted to integrate Hemidal Kerberos with their existing MIT Kerberos infrastructure.

Taxonomy of a Kerberos principal
A principal is made up of three parts. From general applicability to specific they are: the *realm*, *instance*, and *primary*. The realm is a Kerberos administrative domain. Each realm has its own Kerberos key database, and it contains all the information about that domain's users and services. Additionally, a realm can be any group of users and services logically tied together; a section of a computer lab can be under one realm, and a distributed team working on the same project can also be under one realm. The LTIC team used one realm for the security study, but one can configure Kerberos for multiple realms and employ cross-realm authentication if it makes sense logically. An instance is a smaller logical grouping than the realm and it is optional. It can be the hostname of a host that's part of the realm, or it can group users with the same authorities together (such as "admin" for the administrators). The primary is usually a user name or service name. A valid principal can be any of the following:

- *jin/admin@LTIC.POK.IBM.COM*
- *host/litwas01.ltic.pok.ibm.com@LTIC.POK.IBM.COM*
- *jayb@LTIC.POK.IBM.COM*

Installation and configuration procedure for KDC
The LTIC team followed the steps below to install and configure the Kerberos Key Distribution Center (KDC):

1) Install minimal OS (the LTIC team used Red Hat 9), select to install Kerberos packages if applicable.

2) Check to see if Kerberos is installed. Run `rpm -qa` and look for the following packages:

- `krb5-server` (MIT Kerberos)
- `krb5-workstation` (MIT Kerberos)
- `heimdal-`*`version`* (Heimdal Kerberos)

MIT Kerberos comes with Red Hat, and Heimdal Kerberos comes with SUSE LINUX, one can also build the latest release from MIT or Heimdal's website. This study had MIT Kerberos version 1.2.7-10 on RH, and Heimdal version 0.4e-81 on SUSE LINUX Enterprise Edition 8.

3) Add `/usr/kerberos/bin` and `/usr/kerberos/sbin` to search path.

4) Choose a realm name. The LTIC team chose LTIC.POK.IBM.COM because all their hostnames ended in ltic.pok.ibm.com. Realm names must be in all caps.

5) Replace all occurrences of EXAMPLE.COM and example.com with desired realm name (follow respective caps) in the following files in their default locations:

- `/etc/krb5.conf` (Kerberos configuration file)
- `/var/kerberos/krb5kdc/kdc.conf` (Kerberos KDC configuration file)
- `/var/kerberos/krb5kdc/kadm5.acl` (Kerberos admin access control list)

6) In some MIT Kerberos versions, the default `/etc/krb5.conf` is missing this line under the [libdefaults] stanza:

```
default_realm = YOUR-REALM-NAME
```

The above line specifies what the default realm is, and for the common case, this line should be in the configuration file. Otherwise Kerberos commands will not work without explicitly specifying the realm whenever Kerberos commands are run. For the LTIC version of MIT Kerberos, they had to manually add this configuration value.

7) Make sure that hostnames can be resolved properly, otherwise Kerberos wouldn't know where to look for its servers. The LTIC team did not have a DNS server, so they manually added the machine information in `/etc/hosts` so that the KDC and admin server could be properly resolved from `/etc/krb5.conf`.

8) Create the KDC principal database, and choose a master password.

```
~:> kdb5_util create
```

The database master key is then stored in `/var/kerberos/krb5kdc/.k5.realm` [1]. This key allows the KDC to start up unattended (e.g. on a reboot). This convenience is applied at the cost of some security, since the password key can now be stolen if the KDC host is compromised. One may remove this file, but then one has to enter the master password on system startup as well as at various other points.

9) Run `kadmin.local`. This is a bootstrap version of `kadmin` that one first uses to modify the key database directly using root privileges, add `-m` option to supply the master password.

10) Under `kadmin.local` add common Kerberos policies (a policy is a collection of parameters and restrictions on accounts):

```
kadmin.local: addpol users
```

```
kadmin.local: addpol admin
kadmin.local: addpol hosts
```

The above creates three policies for user, administrative, and host credentials, and begins applying them. This is a good idea even if not strictly needed, in case the security administrator wants to start using policies later [1]. For example, one can set policies on admin users that enforce strict password checking and at least monthly password changes. With the default policies above, passwords that are in the dictionary are not allowed.

11) Add a Kerberos principal for the security administrator and another with administrative privileges:

```
kadmin.local: ank -policy users username
Enter password for principal "username@YOUR-REALM-NAME":
Re-enter password for principal "username@YOUR-REALM-NAME":
Principal username@YOUR-REALM-NAME created.
kadmin local: ank -policy admin username/admin
Enter password for principal "username/admin@YOUR-REALM-NAME":
Re-enter password for principal "username/admin@YOUR-REALM-NAME":
Principal username/admin@YOUR-REALM-NAME created.
```

12) Add a host principal for the KDC host:

```
kadmin.local: ank -randkey -policy hosts host/kdc-hostname
kadmin.local: ktadd -k /etc/krb5.keytab host/kdc-hostname
kadmin.local: ktadd -k /var/Kerberos/krb5kdc/kadm5.keytab \
   kadmin/admin kadmin/chagepw
kadmin.local: quit
```

Use the `-randkey` option to generate a random key instead of using a password. When a user authenticates via Kerberos, he/she uses his/her password. A host also has credentials, but cannot supply a password, so a host's secret key is stored in the protected file `/etc/krb5.keytab` [1].

13) Run `kadmind` or `service kadmin start`. This daemon allows one to run the Kerberos Administration command line interface (`kadmin`).

14) Test the kadmin service:

```
~:> kinit          (to obtain Kerberos user credentials)
~:> klist          (to examine user credentials)
```

Monitor its log at the default location `/var/log/kadmind.log`.

15) Test the Kerberos administrative system:

Run kadmin as the user that has administrative privileges:

```
~:>kadmin
Authenticating as principal username/admin@YOUR-REALM-NAME with
password.
Enter password:
```

Under kadmin, run listprincs to see all the principals created. The LTIC team had the following:

```
kadmin: listprincs
K/M@LTIC.POK.IBM.COM
host/litkdc.ltic.pok.ibm.com@LTIC.POK.IBM.COM
jin/admin@LTIC.POK.IBM.COM
jin@LTIC.POK.IBM.COM
kadmin/admin@LTIC.POK.IBM.COM
kadmin/changepw@LTIC.POK.IBM.COM
kadmin/history@LTIC.POK.IBM.COM
krbtgt/LTIC.POK.IBM.COM@LTIC.POK.IBM.COM
```

16) Add `kadmin` and `krb5kdc` using `chkconfig` so they'd start on reboot:

```
~:> chkconfig --level 12345 kadmin on
~:> chkconfig --level 12345 krb5kdc on
```

Another way to test Kerberos is to run the `sserver` and `sclient` programs that come with the installation (specify the port number which the server will listen on, and add the principal `sample/FQDN@REALM` and put this in a key table. More details in `http://www.oreilly.com/catalog/kerberos/chapter/ch05.pdf`). Briefly, here are the steps the LTIC team followed:

1. Run `kadmin`, and add sample/FQDN@REALM to the key database and key table:
   ```
   kadmin:  addpol services
   kadmin:  ank -policy services \
   sample/litkdc.ltic.pok.ibm.com@LTIC.POK.IBM.COM
   kadmin:  ktadd -k /etc/krb5.keytab
   sample/litkdc.ltic.pok.ibm.com@LTIC.POK.IBM.COM
   ```
2. On one terminal session, run

   ```
   sserver -p 13135
   ```

3. On another, run:

   ```
   sclient litkdc.ltic.pok.ibm.com 13135 sample
   ```

4. Successful output on client session:

   ```
   sendauth succeeded, reply is:
   reply len 30, contents:
   You are root@LTIC.POK.IBM.COM
   ```

First Security Scan

For the first scan the LTIC team's intent was simply to gain an understanding of what services were running on all their systems. They performed this scan using the port scanning feature of nmap, an open source multi-use scanning tool. There are many ways nmap can be used, either by scanning entire subnets or by focusing a scan on an individual system. For this study, the team used a very simple Perl script with a list of all system IP addresses. This script forked off two nmap process for each system; one initiated a TCP scan and the other a UDP scan. This strategy allowed the scans to execute concurrently and not affect any other machines unrelated to the security study. In the real world one would probably want to scan entire subnets to make sure that every machine was accounted for and that no unexpected machines were present on the network. The Perl script solution was also intended to provide the flexibility to extend the scanning procedure if necessary, perhaps by comparing results from past runs to determine if any ports had changed availability.

Since the team ultimately wanted to know what services were running on each system, they ran this scan from inside the most trusted zone. This took into account services that may not be available due to firewall filtering but that would still exist and be vulnerable should the firewall be bypassed. Based on this survey the LTIC team could then determine if a service was a necessary exposure or not and whether or not it was externally accessible. Any ports for services that weren't needed had to be disabled. By first reducing the number of services running and then rescanning using a more sophisticated open source scanner, such as Nessus, the team reduced the information to sort through and was able to focus the security processes on services that would actually be running in the "production" environment.

Once all systems were scanned the LTIC team collated the output by system and then listed the ports that were available. Organizing the scan discussion by service enables a team to quickly decide what ports need to be available for the environment. For example, time is saved by declaring SSH to be necessary once rather than reviewing SSH on a per system basis. However, exceptions to this general rule still need to be considered. For example, it is possible that one does not want to have SSH running on specific systems. Sorting by service also allows a team to focus on services which seem anomalous. If only one instance of a service is running on one port it may trigger a specific check where as reviewing by system that single service may get over looked in the monotony.

The LTIC team ran the first nmap scan on all their machines and discovered numerous TCP and UDP ports. After they accounted for their production service ports such as port 80 and various WAS ports, the team sorted out the following interesting and possibly vulnerable ports that the scan exposed:

- *All the ports that PortSentry was listening on (because it was run in non-stealth mode).*
- *portmapper ports (commonly used for NFS):*
  - *111/udp sunrpc*
  - *111/tcp sunrpc*
- *Ports that were unknown to us:*
  - *800 mdbs_daemon*
  - *32787 sometimes-rpc28*
  - *32376 sometimes-rpc15*
- *177/udp xdmcp (X display manager protocol for managing remote displays)*
- *53/udp domain (name server)*
- *6000/tcp X11 (X Window system)*

SSH ports were exposed on every machine because the LTIC team used SSH for remote login for administration purposes.

Removing unnecessary services

The first nmap scan exposed many ports that were for services that were not needed. Since they added to the list of ports that were vulnerable to remote attacks, the team decided to remove these unnecessary services. One can manually stop services, with the `service` *`service`* `stop` option, and then `chkconfig` *`service`* `off` to make sure services don't start on reboot. On Red Hat, run setup, and choose the system services needed to have started at boot time, then deselect everything else. The LTIC team removed all services except for: `anacron, crond, gpm`, `ipchains/iptables, keytable, network, ntpd, rhnsd, sshd, syslog`, and xinetd on the Web servers. After trimming down and rebooting, the team was able to close the portmapper port, and UDP ports 800 and 32787. On SUSE LINUX distributions, one can use either chkconfig or remove the unwanted services' entries from `/etc/rc.d/* and /etc/init.d/rc#.d/*` to prevent them from starting on boot time. Some unnecessary services that are typically enabled by default in many distributions are: `portmapper, sendmail, rpc.mountd, rpc.nfsd, automount, smbd, nmbd, maned, lpd, inetd, telnet, rlogin, rexec, ftp, finger comsat, chargen, echo, and identd`. There are exceptions such as the FTP server machine needing the FTP service, but the rule of thumb is to remove any service that a server should not provide and use.

The LTIC team was able to resolve the exposed PortSentry ports issue in Phase II.

Phase II Security Connections

Network IDS

Snort

In Phase I Snort was run in packet logger mode, and it was tested to see that it was scanning and logging network traffic to disk. Here the LTIC team ran Snort as a NIDS using the default configuration file and rule sets.

The default installation comes with a comprehensive set of rules. However, one should upgrade to the latest rule sets regularly as part of the security process and add specific rules for anything not covered by default. There's a great tool to retrieve the latest rule sets from Snort called `oinkmaster` that can be downloaded from here www.snort.org/dl/contrib/signature_management/oinkmaster/. Start `oinkmaster` like this (don't run this as root):

```
./oinkmaster.pl -o $RULE_PATH 2>&1 | logger -t oinkmaster
```

One can edit the configuration file `snort.conf` and specify the `HOME_NET` value for the desired network, but the default value of `any` will work and is usually a good start. Make sure `RULE_PATH` is correct before running Snort in NIDS mode.

To run Snort in NIDS mode the LTIC team used the following command:

```
snort -de -c /etc/snort/snort.conf -l /var/log/snort
```

This command logs all application and link layer packets of interest to the rule sets specified in the configuration file `/etc/snort/snort.conf`, and logs them to `/var/log/snort/`*`src-ip-address`*.

Here are two examples that demonstrate how Snort as NIDS would use the rules to alert the administrator:

Example #1:
The LTIC had a script that generated /'s:

```
./bigurl 192.168.72.150 80 10000
```

The above command sent `10000 '/'` characters to the Caching Proxy listening on port 80.

The rule `/etc/snort/rules/web-cgi.rules` which was included in `/etc/snort/snort.conf` outputted the following alert:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-CGI
scriptalias access"; flow:to_server,established; uricontent: "///";
reference:cve,CVE-1999-0236; reference:bugtraq,2300;
reference:arachnids,227; classtype:attempted-recon; sid:873;  rev:5;
```

Example #2 (port scan):
If the LTIC team ran nmap `192.168.72.150, the rule /etc/snort/rules/scan.rules`, which was also included in `/etc/snort/snort.conf`, outputted multiple alerts that warned about the scan:

```
alert tcp $EXTERNAL_NET any-> $HOME_NET 8080 (msg:"SCAN Proxy \(8080\)
attempt"; flags:S; classtype:attempted-recon; sid:620; rev:2;
```

Hogwash
In Phase I the LTIC team installed and setup the Hogwash machine to enable it to run in stealth mode, but did not attempt to run Hogwash as an inline scrubber. As Figure 3 shows, nothing was connected to the Hogwash machine in Phase I. In this phase, the team connected the Hogwash machine by setting it up with an inline connection between the internal connection of the protocol firewall and the interface leading to the DMZ. They also used a basic Hogwash configuration and executed it for the first time. In Phase III two Hogwash examples were investigated.

Hogwash configuration file:
For this phase the configuration file that came with the installation was used, with a few updates to make it applicable to the LTIC environment. The Hogwash configuration file `stock.config` is broken down into different sections and the LTIC team modified them when necessary:

- *System section: Specifies information related to the Hogwash machine.*
- *Interface section: Specifies an interface section for each interface Hogwash needs to use.*
- *Routing section: Specifies how Hogwash routes its packets. The default is IDS mode and no routing. For this security study, the LTIC team used SBridge mode. It is the simplest form of routing as it simply passes packets from one interface to the next without trying to intelligently figure out what's going on. One can only bridge two interfaces with SBridge. See the initial configuration file below for an example.*
- *Mangling section: Primarily used for other configurations and not applicable to LTIC's environment.*
- *IP Lists: Specify lists of IP's that will be used as groups anywhere else in the configuration file or rule set. This allows one to supply them once and use their group name to represent*

them everywhere else. For example one can have an IP List of all his/her Web servers, and use the IP List name assigned for use in a rule set:

```
<rule>
ip dst(WebServers, 10.3.3.3)
</rule>
```

- *Actions: Specifies the set of actions to take. One can specify up to 64 different actions, and which set of actions to take in a certain rule set. When a certain rule is matched the action set specified in that rule will be launched.*
- *Modules: These are plugins that perform additional checks.*

For more information on the configuration file visit the Hogwash online documentation `http://hogwash.sourceforge.net/docs/config.html`. Be aware that this doc has spelling and grammar mistakes. For example it spells SBridge wrong (SBrige).

LTIC's initial configuration file (with most comments removed to save space):

```
<system>
Name=Hogwash Sensor
ID=1001
Threads=1
AlertHeader=%ac %m/%d/%y %h:%min:%s %sip:%sp->%dip:%dp
</system>

<interface eth0>
Type=linux_raw
Proto=Ethernet
Role=Normal
</interface>

<interface eth1>
Type=linux_raw
Proto=Ethernet
</interface>

<IPList WebServers>
192.168.72.150
# Caching Proxy IP address
</list>

<IPList DNSServers>
0.0.0.0/0
</list>

<IPList FTPServers>
192.168.72.113
# FTP Server IP address
</list>

<IPList AllServers>
```

```
WebServers
DNSServers
FTPServers
</list>

<action default>
response=alert console
response=alert file(hogwash.alert)
#response=alert socket(www.logger.com:12345)
#response=alert syslog(facility=LOG_AUTH, priority=LOG_INFO,
options=LOG_NDELAY|LOG_PID)
#response=alert
mysql(user=hogwash,dbase=hogwash5,host=localhost,port=3306,pass=passwor
d,logpackets=1)
response=dump packet(packet.log)
response=drop
</action>

<routing>
SBridge(eth0, eth1)
</routing>
```

To ensure Hogwash was installed and configured correctly the LTIC team ran Hogwash with the default rule set that came with the installation to check all the traffic passing through the interface:

`./hogwash -c stock.config -r stock.rules -l /hog` (where `/hog` is the directory for Hogwash logs to be stored).

The `stock.rules` file can invoke other `.rules` files. If they all reside in the same directory then the following lines would suffice:

```
<include nikto.rules>
<include x11.rules>
<include nimda.rules>
<include general.rules>
```

The rules used in this security study came with the Hogwash installation. They were based on Snort rules, but the team did not find any Hogwash documentation that described how that conversion would be done. However, one can edit the rules to his/her liking following the other Hogwash rules as examples.

In Phase III LTIC explored two examples of Hogwash rules.

The LTIC team installed and ran ModSecurity in Phase I on all the Web servers. Because ModSecurity is an Apache plugin it is running as long as the Apache Web server is up. For Phase II, the team saw no need to connect this tool to any other machine or tool since it only protected the Web server on the machines it was installed on.

However, the team did notice some performance degradation with ModSecurity in this phase. While the Web server was running with ModSecurity, it required a significant amount of CPU time to process every HTTP request against the ~840 converted Snort rules that it had been using since Phase I. The transaction rate with ModSecurity and all of its ~840 rules enabled was about 1/20th of what it was without ModSecurity. For this performance reason the rules used by ModSecurity must be carefully chosen. For instance, scanning for IIS attacks against an Apache Web server would not be a wise use of CPU cycles. Trim the ModSecurity rules to examine the minimal set of exploits that the target Apache system might be vulnerable to.

Host IDS

PortSentry

Based on the first scan, the LTIC team found that nmap detected all the ports that PortSentry was listening on. Below is a list of the open ports detected by nmap on the host that was running PortSentry in basic port-bound UDP mode. Remember that the host wasn't supplying all of these services; the ports were open because PortSentry was listening for connections on them:

```
Port        State       Service
1/udp       open        tcpmux
7/udp       open        echo
9/udp       open        discard
69/udp      open        tftp
111/udp     open        sunrpc
123/udp     open        ntp
161/udp     open        snmp
162/udp     open        snmptrap
177/udp     open        xdmcp
513/udp     open        who
523/udp     open        ibm-db2
635/udp     open        mount
640/udp     open        pcnfs
641/udp     open        unknown
700/udp     open        unknown
31335/udp   open        Trinoo_Register
31337/udp   open        BackOrifice
32770/udp   open        sometimes-rpc4
32771/udp   open        sometimes-rpc6
32772/udp   open        sometimes-rpc8
```

```
32773/udp  open        sometimes-rpc10
32774/udp  open        sometimes-rpc12
54321/udp  open        bo2k
```

In order for remote scans to not detect that PortSentry was running, the team needed to run PortSentry in stealth mode. PortSentry has two stealth methods. The first method uses a pre-defined list of ports to monitor and if someone attempts to connect to them it activates an alert. The other method watches a range of ports except for those that have services bound to or ones that the administrator manually excluded. The second approach is the most sensitive but is also more CPU intensive, more prone to false alarms, and more likely to cut off legitimate traffic. The second method is also known as advance stealth mode. False alarms are more likely to occur for advanced UDP stealth mode because PortSentry makes no distinction between broadcast and direct traffic and often ends up blocking legitimate broadcast traffic.

To enable stealth mode using the first method run PortSentry with options `–stcp` or `–sudp`. To run PortSentry in advanced stealth mode, use command line options `–atcp` or `–audp` and configuration file options ADVANCED_PORTS_TCP and ADVANCED_PORTS_UDP to specify the range of ports to watch, and ADVANCED_EXCLUDE_TCP and ADVANCED_EXCLUDE_UDP options to specify ports to exclude.

The LTIC team used the regular stealth mode to run PortSentry during this phase:

```
[root@litxwas02 portsentry]# ./portsentry –stcp
```

The file `/var/log/messages` contained the following messages after the above command was run:

```
Nov 19 15:50:46 litwas02 portsentry[12849]: adminalert: PortSentry
1.2 is starting.

Nov 19 15:50:46 litwas02 portsentry[12850]: adminalert: Going into
stealth listen mode on TCP port: 1

……

Nov 19 15:50:46 litwas02 portsentry[12850]: adminalert: PortSentry
is now active and listening.
```

While PortSentry was running in stealth mode, another nmap scan was executed against the hosts that had PortSentry, and the team was able to verify that nmap couldn't detect the PortSentry ports anymore.

One can have PortSentry start at boot time by creating an init script in `/etc/init.d/` using the standard init script configurations.

PSAD
In Phase I the LTIC team installed and ran PSAD with mostly default configurations. In this phase they experimented with different options and configurations with PSAD.

When restarting PSAD, one can preserve all the command options used to start it originally and clear PSAD of last scan and packet counter history by entering `psad -R`. This is useful for quickly testing out different configuration options in the same configuration file.

When PSAD was run with the default configuration, it sent e-mail alert messages at a danger level of 1. Level 1 specifies that the local firewall configuration has dropped five packets from the same source IP. This prompts PSAD to send an e-mail alert message to the administrator specified by the configuration option EMAIL_ADDRESS. One can see this in `/var/log/messages` and `/var/log/psad/fwdata` as:

```
Jan 28 04:27:12 litlb01 psad: .. scan detected: 192.168.71.17 ->
192.168.71.97  tcp=0 udp=0 icmp=1 dangerlevel: 1

Jan 28 04:27:12 litlb01 psad: .. sending e-mail alert to:
admin-email-address
```

As the administrator one will probably get a lot of false alarms this way. LTIC was running regular legitimate transactions that produced a danger level of 1 and caused the administrator to be alerted. To avoid this behavior, raise the danger level to 2 or higher. The team tested this by raising the danger level to 2. Then they drove regular transactions and separately tried to remote scan (using nmap) the PSAD machine, this time only the IP address for the scanning machine got reported. One can also limit the number of e-mails to receive for a single scanning IP by changing the configuration parameter PSAD_EMAIL_LIMIT (the default is 50).

The team also tried the ENABLE_AUTO_IDS and AUTO_IDS_DANGER_LEVEL configuration options. By specifying Y for ENABLE_AUTO_IDS, PSAD will automatically block an IP that has reached a (configurable) danger level through modification of iptables or TCP wrapper rule sets. Use the AUTO_IDS_DANGER_LEVEL option to specify the danger level that triggers auto-blocking. The team chose to auto-block using iptables be setting the option IPTABLES_BLOCK_METHOD to Y. This option blocks ALL traffic from the source IP if it has reached the danger level. Auto-blocking using iptables was verified after remote scanning using nmap caused PSAD to detect a danger level of 2 and all traffic from the nmap

machine was blocked. Only after a `psad -F` command (which unblocks all auto blocked IP's) did PSAD allow traffic from the nmap machine again:

```
litlb01:~ # psad -F
 .. Removing iptables auto-generated block rules.
 .. Deleting rule 1 for 192.168.71.17, mangle table PREROUTING chain:
        DROP       all  --  192.168.71.17        0.0.0.0/0
 .. Deleting rule 1 for 192.168.71.17, nat table PREROUTING chain:
        DROP       all  --  192.168.71.17        0.0.0.0/0
```

One thing to note is that due to default e-mail send/receive times it took a while to receive the alert that warned a danger level of 2 was reached. It was probably about 5 – 10 minutes before PSAD e-mailed the administrator about the potential danger – but with auto blocking set it was not too late since it immediately blocked once the danger level was reached. However, one should be aware that an attacker can also abuse the auto block option by spoofing the source IP address to be from a particular system in an effort to make it look as though the site is scanning the PSAD machine. This would cause PSAD to consequently block all access from that machine. The administrator has to weigh these options to make the best use of PSAD. For more PSAD options, including how it works with Snort, please refer to the man pages that come with the installation.

Firewall

Setting up the firewalls:
The LTIC team created custom iptables rules to allow legitimate traffic in and out, and by default drop all other traffic. The team then created a script, `/etc/fw.rules`, to hold all the iptables rules they customized for their environment. A corresponding script, `/etc/fw.clear` was created to remove all of the rules and open up the firewall for testing purposes. In addition to iptables rules, a number of Linux kernel switches were set to help protect the environment. Having all these rules in one place allowed the team to consistently follow a security policy that was also portable to another machine.

To help reduce the number and complexity of the rules, the iptables rule: `iptables -A INPUT -m state -- state RELATED,ESTABLISHED -j ACCEPT` was used to allow all established and related traffic to a rule to immediately pass through the firewall. This rule takes advantage of iptables' statefulness feature by permitting incoming packets only if they are part of established ongoing connections. An example of this synergy would be to setup a rule allowing IP traffic from any source IP to access Web server xyz at port 80. With the `iptables -A INPUT -m state -- state RELATED,ESTABLISHED -j ACCEPT` global rule set there is no need to have a second rule to allow IP traffic from Web server xyz to the IP that established the connection. In the LTIC firewalls, this one rule cut the number of rules needed in half.

Protocol firewall setup:
Below is a complete listing of the `/etc/fw.rules` file from LTIC's protocol firewall.

```
## Protocol Firewall rules for LITPFW01 ##
#
# Turn off ICMP redirects
for x in /proc/sys/net/ipv4/conf/*/accept_redirects
do
   echo 0 > $x
done
for x in /proc/sys/net/ipv4/conf/*/send_redirects
do
   echo 0 > $x
done
#
#  Turn on ICMP broadcast rejection
#
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
#
# Turn on Source address validation
#
```

```
for x in /proc/sys/net/ipv4/conf/*/rp_filter
do
   echo 1 > $x
   done
#
#  Turn on syn_cookie protection
#
echo 1 > /proc/sys/net/ipv4/tcp_syncookies
#
#    Flush all the current rules
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables –F

# Set a default policy to drop all forwarded packets
iptables -P FORWARD DROP


####   FORWARD   FORWARD    FORWARD   #########
#set the forward rules
# allow internal LAN to access Web (Test rule only – Removed later)
iptables -A FORWARD -p tcp --dport 80 -s 192.168.71.0/24 -j ACCEPT
# allow the public LANs to access caching proxy
iptables -A FORWARD -p tcp --dport 80 -d 192.168.72.150 -s
192.168.73.0/24 -j ACCEPT
iptables -A FORWARD -p tcp --dport 80 -d 192.168.72.150 -s
192.168.75.0/24 -j ACCEPT

# allow the internal clients LAN to SSH to anything on internal zone
iptables -A FORWARD -p tcp --dport 22 -s 192.168.75.0/24 -j ACCEPT

# allow public LANs to ftp to litftp01
iptables -A FORWARD -p tcp -dport ftp -d 192.168.72.113 -s
192.168.75.0/24 -j ACCEPT
iptables -A FORWARD -p tcp -dport ftp -d 192.168.72.113 -s
192.168.73.0/24 -j ACCEPT

#######
# allow the NTP Server to get out
iptables -A FORWARD -p udp --dport 123 -s 192.168.71.29 -j ACCEPT
iptables -A FORWARD -p tcp --dport 123 -s 192.168.71.29 -j ACCEPT
iptables -A FORWARD -p upd --dport 873 -s 192.168.71.29 -j ACCEPT
iptables -A FORWARD -p tcp --dport 873 -s 192.168.71.29 -j ACCEPT

# allow established and related communication - both ways.
iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
```

```
#allow access to the IBM name servers
iptables -A FORWARD -p udp --dport 53 -d 9.0.3.1 -j ACCEPT
iptables -A FORWARD -p udp --dport 53 -d 9.0.2.1 -j ACCEPT
iptables -A FORWARD -p tcp --dport 53 -d 9.0.3.1 -j ACCEPT
iptables -A FORWARD -p tcp --dport 53 -d 9.0.2.1 -j ACCEPT

# LOG everything that is to get dropped
iptables -A FORWARD -j LOG --log-prefix "DROPPING " --log-level alert

#######  INPUT     INPUT      INPUT    #######
## Prevent anything from getting into the firewall by adding default
DROP rule
iptables -P INPUT DROP

# allow SSH on the firewall
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --sport 22 -j ACCEPT

# allow established and related communication - both ways.
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

# LOG everything that is dropped
iptables -A INPUT -j LOG --log-prefix "DROPPING " --log-level alert

#list the rules
iptables -L
```

Domain firewall setup:

```
#   Domain Firewall rules for LITDFW01
#
# Before starting with the iptables - lets tighten up Linux
#
# Turn off ICMP redirects
for x in /proc/sys/net/ipv4/conf/*/accept_redirects
do
   echo 0 > $x
done

for x in /proc/sys/net/ipv4/conf/*/send_redirects
do
   echo 0 > $x
done
#
#  Turn on ICMP broadcast rejection
#
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

```
#
# Turn on Source address validation
#
for x in /proc/sys/net/ipv4/conf/*/rp_filter
do
   echo 1 > $x
done
#
#  Turn on syn_cookies
#
echo 1 > /proc/sys/net/ipv4/tcp_syncookies

#  Now for the IP Tables
#
#    Flush all the current rules
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -F

# Set a default policy to drop all forwarded packets
iptables -P FORWARD DROP

#### FORWARD   FORWARD    FORWARD #########
#set the forward rules

# allow internal LAN to access Web  (Test rule only - Removed later)
iptables -A FORWARD -p tcp --dport 80 -s 192.168.71.0/24 -j ACCEPT

# allow the internal clients LAN to SSH to anything on internal zone
iptables -A FORWARD -p tcp --dport 22 -s 192.168.75.0/24 -j ACCEPT

# allow the Caching Proxy to access port 80 on the Network Dispatcher
# z Series cluster
iptables -A FORWARD -p tcp --dport 80 -s 192.168.72.150 -d
192.168.71.96 -j ACCEPT
# x Series cluster
iptables -A FORWARD -p tcp --dport 80 -s 192.168.72.150 -d
192.168.71.98 -j ACCEPT

# allow connection to the central log servers' port 514 from DMZ
machines
iptables -A FORWARD -p udp --dport 514 -s 192.168.72.113 -j ACCEPT
iptables -A FORWARD -p udp --dport 514 -s 192.168.72.108 -j ACCEPT
iptables -A FORWARD -p udp --dport 514 -s 192.168.72.22 -j ACCEPT

# allow centralized Kerberos user authentication from DMZ machines
```

```
iptables -A FORWARD -p udp --dport 88 -s 192.168.72.113 -j ACCEPT
iptables -A FORWARD -p udp --dport 88 -s 192.168.72.108 -j ACCEPT
iptables -A FORWARD -p udp --dport 88 -s 192.168.72.22 -j ACCEPT

iptables -A FORWARD -p tcp --dport 749 -s 192.168.72.113 -j ACCEPT
iptables -A FORWARD -p tcp --dport 749 -s 192.168.72.108 -j ACCEPT
iptables -A FORWARD -p tcp --dport 749 -s 192.168.72.22 -j ACCEPT

#allow the NTP Server to get out
iptables -A FORWARD -p udp --dport 123 -s 192.168.71.29 -j ACCEPT
iptables -A FORWARD -p tcp --dport 123 -s 192.168.71.29 -j ACCEPT

#allow rsync to get out to the internet
iptables -A FORWARD -p upd --dport 873 -s 192.168.71.31 -j ACCEPT
iptables -A FORWARD -p tcp --dport 873 -s 192.168.71.31 -j ACCEPT

#allow access to the IBM name servers
iptables -A FORWARD -p udp --dport 53 -d 9.0.3.1 -j ACCEPT
iptables -A FORWARD -p udp --dport 53 -d 9.0.2.1 -j ACCEPT
iptables -A FORWARD -p tcp --dport 53 -d 9.0.3.1 -j ACCEPT
iptables -A FORWARD -p tcp --dport 53 -d 9.0.2.1 -j ACCEPT

# allow established and related communication - both ways.
iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT

# LOG everything that is to get dropped
iptables -A FORWARD -j LOG --log-prefix "DROPPING " --log-level alert

#######  INPUT     INPUT      INPUT    #######

## Prevent anything from getting into the firewall by adding default
policy of # DROP
iptables -P INPUT DROP

# allow SSH on the firewall
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --sport 22 -j ACCEPT

# allow established and related communication - both ways.
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

# Log everything that's been dropped.
iptables -A INPUT -j LOG --log-prefix "DROPPING " --log-level alert

#list the rules
iptables -L
```

DROP rules:
Note that the default rule (specified with the `-P` option) for both FORWARD and INPUT tables is to DROP packets. One could use REJECT as well but that would send notification back to the sender that the firewall has rejected the packet. Depending on the environment, one may not want the sender who may be an attacker to know that the target exists. A REJECT rule can lead to a firewall's unwitting involvement in a DoS attack: an attacker could spoof the source address to be that of the targeted victim and send numerous requests to a firewall that uses REJECT; the firewall would in turn unintentionally flood the victim with rejections. Specifying the rule using DROP, on the other hand, would not send anything back, thus the sender could not know if there is a firewall or if the machine even exists. In a production environment DROP is a better default rule to avoid potential abuse and unnecessary security exposure.

When building firewall rules it is good practice to start with the default rule, usually DROP, and then set ALLOW rules to enable legitimate traffic flow.

Save and load firewall at boot time:
Once the firewall is configured appropriately, save it with `iptables-save > filename`, and restore it with iptables-restore filename (for both Red Hat and SUSE LINUX). In Red Hat, one can have the firewalls startup at boot time by running the command `iptables-save > /etc/sysconfig/iptables`. To enable iptables at boot time simply run `chkconfig iptables on`. Then the firewall rules are restored from `/etc/sysconfig/iptables` by the existing `/etc/init.d/iptables` script. SUSE LINUX takes a different approach and builds rules from variables set in `/etc/sysconfig/SUSEfirewall2`. The firewall is enabled with the SUSEfirewall2 script, which you can configure with chkconfig to start at boot time. For more information on SUSE LINUXfirewall2 read `/etc/sysconfig/SUSEfirewall2`.

Linux switches:
Turning off ICMP redirects with switches `accept_redirects` and `send_redirects` denies the "use of a different 'shorter' route for packets that could include the cracker's box or a third party for a denial of service attack [5]".

To disable ICMP echo broadcast so that denial of service attacks that abuse the router's broadcast functionality don't occur, turn on the `icmp_echo_ignore_broadcasts` switch.

The source address validation switch `rp_filter` (from the protocol and domain firewall rules above) is a Reverse Path Filter that ingress filters outgoing packets to only allow packets within the router network to pass through and egress filters incoming packets that claim to come from within the router. It compares the source IP addresses of packets to the

`IP/netmask` of the interface it is filtering on. This prohibits packets with forged source IP's that's not in the prefix range from going out and coming in.

Another popular attack is the TCP SYN flood attack. The attacker floods the victim with SYN requests but does not respond to the corresponding SYN ACK – hence tying up resources at the victim to the point of denial of service. Luckily, from Linux kernel 2.2 on, one can build the kernel with the CONFIG_SYN_COOKIES option, and by turning it on with the switch `tcp_syncookies`, avoid the TCP SYN flood attack. More recent kernels come with TCP SYN defense by default.

All of the above Linux switches have little or no overhead and provide increased protection on a Linux firewall.

Auditing and Logging

The LTIC team ran the system loggers for remote logging in Phase I. They also installed the two auxiliary programs, Lire and Swatch, one per central log server. In this phase the team configured the rest of the machines in the environment to log to both system loggers.

On the central loggers:
As a reminder, in Phase I syslogd was run on both central log servers with the default `/etc/syslog.conf` file, and the following runtime parameters in `/etc/sysconfig/syslog` file:

```
SYSLOGD_OPTIONS="….. -r -m 0……"    #Red Hat
SYSLOGD_PARAMS="….. -r -m 0……."    #SUSE LINUX
```

On machines logging to the central loggers:
All systems except the log servers themselves, were to transmit their logs to both central log servers in addition to keeping a local copy. To accomplish this, the team inserted the following in each machines' `/etc/syslog.conf`:

```
# enable the following to keep all messages in one file
*.*                             -/var/log/allmessages
# log everything to central loggers as well
*.*                             @192.168.71.110  # log server 1
*.*                             @192.168.71.109  # log server 2
```

Then the team ran syslogd on each machine logging to central log servers.

Configuring the system logger as above will log all kernel messages and by default any programs that use syslog. However, not all of the security products considered in this study

log to syslog by default. In fact, some programs don't have the option to send messages to syslog (Hogwash and ModSecurity). Below is a list of how the LTIC team decided to capture logs generated from the open source security tools:

Snort
In the LTIC environment, Snort was a Network IDS (see the Network IDS sections in Part 2) on a dedicated machine. NIDS is the type of system designed for having someone there regularly to check its logs, so it doesn't have to send everything it logs (which is all traffic that matches its massive rule sets) to syslog. Snort offers the option `-s` (specified as command option) that will send only the alerts to syslog (`/var/log/messages`). The default facilities for the syslog alerting mechanism are LOG_AUTHPRIV and LOG_ALERT. Since the LTIC team setup `/etc/syslog.conf` to transmit to the central loggers Snort alerts would show up there as well.

One can tailor Snort's alert logging by using the Snort output module `alert_syslog`. Modules are loaded at runtime by specifying the output keyword in the rules file, i.e.:

```
output alert_syslog: LOG_AUTH LOG_ALERT
```

For more alert syslog options, refer to section 2.11 of the Snort Manual: http://www.snort.org/docs/snort_manual.pdf.

Hogwash
One downside to running Hogwash in stealth mode is that it doesn't have an IP address and can't communicate with other machines on the network. This makes the machine security-rich but can make administration complex. Thus the LTIC Hogwash logs resided locally on the disk in the logging directory specified when running Hogwash (in this study it was `/hog`).

If not running Hogwash in stealth mode then it's able to communicate with central log servers. But Hogwash doesn't have a syslog option. The solution the LTIC team improvised for situations like this was to setup a cron job that will copy (with the `logger` command) the latest messages from the Hogwash logs (found using `diff`) to `/var/log/messages`.

ModSecurity
ModSecurity logs to the same place as other Apache access and error logs. In this study case it was `/opt/IBMHttpServer/logs`. ModSecurity doesn't provide the option to log to syslog so one would use the same cron setup talked about for Hogwash.

PortSentry
PortSentry by default automatically logs incidents via syslog. One can specify the syslog facility and level for PortSentry to use in `portsentry_config.h` before compiling and installing the tool but the defaults are usually fine. See the PortSentry sections for examples of its logs.

PSAD
PSAD by default automatically uses syslog. It retrieves and logs information from and to `/var/log/messages`. See this paper's PSAD sections for information on configuring its logs

iptables
To have iptables log to syslog one has to specify the `-j LOG` option in the rule definition. LTIC firewalls were setup to log dropped packets. An example is:

```
iptables -A FORWARD -j LOG --log-prefix "DROPPING " --log-level alert
```

By adding the above rule at the end of the FORWARD table, after all the ACCEPTS, it logged packets that were dropped from the FORWARD table at the alert level.

In `/var/log/messages`, these dropped packets showed up like this:

```
Jan 28 03:33:16 litlb01 kernel: DROPING=eth0 OUT= MAC=
SRC=192.168.71.17 DST=192.168.71.97 LEN=64 TOS=0x00 PREC=0x00 TTL=255
ID=389 PROTO=ICMP TYPE=0 CODE=0 ID=57005 SEQ=0
```

PAM, SSH, and Kerberos
By default PAM automatically logs via syslog. Because the LTIC team used Kerberos and SSH with PAM modules, related messages were automatically logged to `/var/log/messages` as well. PAM messages looked like this:

```
Jan 26 10:19:46 litxwas01 sshd(pam_unix)[21115]: authentication
failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=192.168.71.112
 user=jin
Jan 26 10:19:47 litxwas01 sshd[21115]: pam_krb5: authentication
succeeds for `jin'
Jan 26 10:19:47 litxwas01 sshd(pam_unix)[21115]: session opened for
user jin by (uid=0)
Jan 26 10:33:56 litxwas01 su(pam_unix)[21368]: session opened for user
root by jin(uid=501)
```

By default, the KDC logs separately to `/var/log/kadmind.log` and `/var/log/krb5kdc.log`. They contain logging information about kadmin and KDC activities, respectively. One may change kadmind and krb5kdc to log to `/var/log/messages` in `/etc/krb5.conf`, by changing the values of the following configuration options:

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log
```

However, this is not recommended because these logs may contain sensitive information that only the KDC administrator should see. Therefore the LTIC team decided these logs should be kept separately and securely on the local system.

In Phase III the team studied Lire and Swatch's abilities to generate informative and presentable reports from the log files on the central log servers.

User Authentication

Kerberos V5

In Phase I, the LTIC team set up the KDC and kadmin servers. For Phase II they explored some basic Kerberos operations and one of two Kerberos configurations for user authentication. The team learned in this phase and next, that Kerberos is very flexible as an authentication mechanism.

Before going into Kerberos authentication configurations, the team had to perform some basic Kerberos operations. The few sub-sections that follow describe the preparations and basic operations needed in order for other machines in the environment to use the central KDC for user authentication. All the machines that used the central KDC for authentication are referred to as Kerberos realm hosts in this document.

1. If not using DNS, add KDC machine's hostname to the `/etc/hosts` file of the Kerberos realm host.
2. On the Kerberos realm host, make sure `krb5-workstation` and `krb5-server` are installed. If using Heimdal, make sure `heimdal` is installed.
3. Transfer `/etc/krb5.conf` from the KDC to the realm host.
4. Include Kerberos binaries' path in current path.

   Note: the path for Kerberos executables is different for MIT and Heimdal implementations. Make sure to use the right one.

5. Add local userIDs matching the primaries of the Kerberos principals. For example, if the Kerberos principal is *username@YOUR-REALM-NAME*, then add a local userID *username*. Local passwords are optional since Kerberos will be used for user authentication.

Time synchronization:
Time synchronization is an integral part of a security infrastructure to correlate logs and communication. In order for Kerberos clients to communicate with the KDC and kadmin servers, their times must be kept synchronized (Kerberos allows a maximum of 5 minutes difference). The LTIC team used the NTP daemon that came with Linux distributions. Then they configured all the machines to synchronize their time with a local central NTP server. For more information on NTP configuration, refer to the NTP man pages.

Adding a host:

1. Make sure kadmind is running on the KDC machine.

2. Run `kadmin` on the new host as a Kerberos administrator. The kadmin utility will then ask for *userID/admin@YOUR-REALM-NAME's* password. For example, if during the Kerberos KDC and kadmin setup root/admin was created as the only admin ID, and the user is logged on as root on the new host, then running `./kadmin` will result in a prompt asking for *root/admin@YOUR-REALM-NAME's* password. If the user is logged in with a userID other than root, then the user should run `kadmin -p root/admin`. Either way Kerberos will prompt the user to enter root/admin's password, because it considers root and root/admin as two different users.

Note: One does not have to specify the full principal name when running kadmin, because kadmin will look in `/etc/krb5.conf` for the realm name and automatically attach it when needed.

3. Now the user should be under the kadmin shell. Use the following commands to add the new host with a random key and the policy "hosts":

```
kadmin: ank -randkey -policy hosts host/newhost.ltic.pok.ibm.com
kadmin: ktadd -k /etc/krb5.keytab host/newhost.ltic.pok.ibm.com
kadmin: quit
```

This will add the new host to the local `/etc/krb5.keytab` which will be used every time the user uses Kerberos on this machine to identify him or herself.

Adding a principal:

1. Do this from any realm host, or from the KDC (which is itself a realm host).
2. Enter kadmin, and input the following commands:

```
kadmin: ank –policy users user1
Enter password for principal "user1@YOUR-REALM-NAME":
Re-enter password for principal "user1@YOUR-REALM-NAME":
Principal user1@YOUR-REALM-NAME created.
kadmin: quit
```

3. Inform the person of the temporary password which should be changed immediately with `kpasswd`.

PAM settings:
In order to start using Kerberos as part of the system wide user authentication mechanism, one can use either Kerberized applications such as SSH or Telnet. It is also possible to configure PAM settings for system wide Kerberos authentication. The LTIC team went with the latter option, since a homogeneous authentication mechanism was desirable, and applications already used PAM modules (the default setting of most distributions). It was also easier than compiling each application with Kerberos and/or modifying multiple configuration files.

Red Hat makes doing this easier since all its PAM modules already use a system-wide module for authentication by default. SUSE LINUX requires editing individual modules of each application to use Kerberos for authentication or manually creating a system-wide module and then editing all other modules to use that one.

For system-wide authentication, using Kerberos on Red Hat with PAM, the team followed these steps:

1. Make sure pam-krb5 is installed.
2. On Red Hat, run authconfig as root and select Kerberos authentication.
3. Changed the following in `/etc/pam.d/system-auth` from:

   ```
   session     optional      /lib/security/pam_krb5.so
   ```

   to:

   ```
   session     required      /lib/security/pam_krb5.so
   ```

4. Test the new authentication by ssh'ing into the new Kerberos client machine just configured, using an ID that was setup in Kerberos and added locally.

Note: If one strictly requires Kerberos for authentication as above, then the configuration runs the risk of relying on the stability and availability of the KDC and the network for all user authentications across the realm. If the security administrator leaves `session` as `optional`, then that person is choosing to use local PAM authentication *or* Kerberos

authentication. Providing the password for either one would grant users access. However, this option takes away some of the security provided by strictly using Kerberos.

Kerberos Configurations:
Technologies like Kerberos are usually used in an environment where the machines are considered as commodity computing resources and it doesn't matter what machine users authenticated with, they always have the same authorities on each machine. To achieve universal configuration the LTIC team added principals `root@LTIC.POK.IBM.COM` and `root/admin@LTIC.POK.IBM.COM` in Kerberos. Then they followed the basic operations noted above to configure Kerberos realm hosts to use Kerberos for user authentication. Then users could log into any machine in the LTIC realm as the root user with root privileges locally.

This configuration was what the LTIC team worked with during this phase. This configuration allowed every administrator root access to all machines, whether they were responsible for them or not. It is fine if everyone is to be trusted with the same privileges on every single machine, but the team wanted something more secure so another configuration was explored in Phase III.

To make the universal configuration more secure, at least make sure that root does not have administrative access to kadmin. This can be achieved by not creating a `root/admin@`*`YOUR-REALM-NAME`* principal. Instead, create an admin principal only for the KDC administrator. This way the KDC administrator is the only person who can access kadmin. Another option is to create a `root/admin@`*`YOUR-REALM-NAME`* principal but give it a password different from `root@`*`YOUR-REALM-NAME`*`'s` password, then make sure only the KDC admin knows root/admin's password. By separating the KDC administrator from everyone else, one can provide protection against unauthorized kadmin access.

The desired configuration the LTIC team wanted to have was to grant root access of machines only to those authorized. For example, only the Web administrator should have root access to the Web servers, and this person should not have equal access to the firewalls. The LTIC team explored this configuration further in Phase III and also looked at Heimdal's compatibility with MIT Kerberos.

Remote Scan

For this phase the LTIC team used Nessus to remotely scan all systems from the internal zone. Nessus is one of the best automated open source network security scanners available to help determine if services are vulnerable to different attacks. Nessus keeps an updated list of attack plugins to check against the latest attacks and generates easily readable reports and charts for analysis. In essence, running security scans using tools such as Nessus is a crucial process to have as part of one's security measures.

Nessus setup

The team installed Nessus version 1.2.3-54 on a SUSE LINUX Enterprise Edition 8 SP3 Linux on zSeries image using the RPM that came with the distribution. Check the distribution as it might already contain a prepackaged version of Nessus. If not, one can download Nessus from www.nessus.org and follow the installation steps described in the Nessus guide here: http://www.sun.com/bigadmin/features/articles/nessus_scanner.html. Since the LTIC security environment had four segments (see Figure 3), the team set up a network interface for each segment using guest LAN technology (a feature of z/VM). They set up one guest LAN interface to be on the internal subnet, another on the DMZ subnet, a third on the company subnet (internal clients in Figure 3), and the last on the public subnet (external clients in Figure 3), all on the same Linux on zSeries image that Nessus itself was installed on. This allowed the same machine to run scans from any subnet. For this scan, only the internal zone interface was configured up so that the scan would originate only from that segment, thus bypassing the two firewalls. This enable the scan to cover the broadest possible security exposures.

Nessus runs in a client server configuration, one must have the server on *NIX operating system whereas the client can reside anywhere from OS X® to Windows to Linux. Since the team used a dedicated Linux on zSeries image for scanning purposes only they put both the Nessus server and client there. The server does the actual scanning and auditing whereas the client provides a user friendly GUI for scan configurations and collecting results. This configuration allows for continuous autonomous scanning.

If running Nessus client and server on separate machines, then it's a good idea to generate SSL certificates for encrypting communication between the two. Nessus provides a script `nessus-mkcert` that will help do exactly this. The LTIC team followed the instructions from the guide above, and created a SSL certificate even though their Nessus client and server resided on the same machine. By following the instructions, one will end up creating a CA as well as a server certificate and server key which is signed by the CA key. (Note: The Nessus guide above did a manual installation so their Nessus paths were different from the paths of the Nessus that came with SUSE LINUX. For example, in SUSE LINUX Nessus,

`nessusd.conf` is in `/etc/nessus/`, whereas the guide's `nessusd.conf` is in `/usr/local/etc/nessus/`)

With the program `nessus-adduser`, the team created one Nessus user that had the ability to scan all four networks. For larger networks one may have a couple of administrators that wish to use one Nessus server. In that case one should consider creating a few of Nessus users, each with authority to scan a certain part of the network, and assign these to the administrators for the network or system that they are responsible for. The `nessus-adduser` program will ask for the IP addresses to accept or deny for each user created. A corresponding `nessus-rmuser` is provided to remove users from the Nessus database.

To run the Nessus server execute `/usr/local/bin/nessusd &`. One can setup an init script to run the Nessus server at boot time.

To run the Nessus client, the LTIC team logged into the Nessus machine using SSH with the X forwarding option (-X) from a machine that supports X Window System. Then they started the client by entering `nessus` which brought up the GUI configuration interface.

The team followed the instructions in the Nessus guide given by the URL above to setup their first Nessus scan. The guide described a Nessus configuration where the client and the server ran on the same machine. The LTIC team also had this setup so most of the guide's instructions applied to LTIC as well. To setup the first scan, the team entered the following on the Nessus client GUI (answers indicated in *italics*):

1. Nessus host: *localhost*; Port: *1241*; Login: *username created with nessus-adduser*; Password: *that user's password specified during nessus-adduser*.
2. SSL Setup (first time login only): *Display and remember the server certificate, do not care about the CA.* Note: If `nessus-mkcert` was used then a CA should have been created which means it's also acceptable to pick the second or third option as well – especially if the client and server are on different machines.
3. Plugin selection: *accepted the default.*
4. Prefs: *accepted the default.*
5. Scan options: *accepted the default.*
6. Target selection: *The team had created a file that contained the entire list of host IP's and loaded that file in the Target box.*
7. User: *left blank.*
8. KB: *accepted the default.* Although for subsequent scans one may want to enable saving knowledge base to only search for new vulnerabilities.

9. *Pressed the "Scan" button.* Depending on the number of hosts and known services and the network bandwidth available, processing time varies. The LTIC team had about 20 hosts and it took 2-3 hours.

Results (and fixes)

The team saved the results in HTML format which contained all the hosts and the hosts' respective vulnerabilities in a very readable format in one file and HTML-plus format which also includes pie charts to graphically display stats such as the most vulnerable host and most prevalent vulnerabilities and services.

The scan found many vulnerabilities and warnings about WebSphere Application Server and Tivoli Storage Manager. However, because the goal of this study was not in security for proprietary products these findings are now listed here (these problems may or may not be prevented by the products' own security offerings. The team saved the results of every scan for interested parties). Interesting results for the open source security study were:

1. Warnings about SSH: "Older versions of the SSH protocol are not completely cryptographically safe so they should not be used." The Nessus recommended solution was to set the option 'Protocol' to '2' in the SSH configuration files, thus configuring the SSH server to not use the less secure version 1 protocol. The team updated all SSH configurations in correspondence to the solution recommendation.
2. Vulnerabilities on SSH: "A buffer overflow exists in the daemon if Andrew File System (AFS) is enabled, or if the options KerberosTgtPassing or AFS TokenPassing are enabled." The Nessus recommended solution was to upgrade to the latest version of OpenSSH. Since the LTIC team was not using both the KerberosTgtPassing nor AFS TokenPassing options and systems didn't have AFS enabled, the team did not upgrade to the latest version.
3. Warning about NTP: "Some versions have been found to be vulnerable to buffer overflows." The Nessus recommended solution was to upgrade if NTP happened to be vulnerable. The team didn't have vulnerable NTP versions.
4. Warning about FTP (on our FTP server): "This FTP service allows anonymous logins. If you do not want to share data with anyone you do not know, then you should deactivate the anonymous account, since it may only cause trouble." The LTIC team consciously setup the FTP server to allow anonymous login only and took the necessary precautions for proper read and write permissions.
5. Warning about xdmcp: "This protocol is used to provide X display connections for X terminals. It is completely insecure since the traffic and passwords are not encrypted." To patch this warning, the LTIC team removed the xdmcp protocol on all of their servers.

   Note that the team did not attempt to secure the application software because they were only investigating open source security tools. If one wants to attempt to be secure one should take a multi-layered approach that helps secure application software as well as the

network. Utilizing the security offerings that come with application software is the best approach to protecting applications because these security offerings are designed specifically for their applications. Remember that security is only as strong as its weakest link.

Phase III Security Optimizations

Network IDS

In earlier phases the LTIC team installed Hogwash on a machine acting as an inline packet scrubber between the protocol firewall and DMZ. They then configured Hogwash to apply to their environment and ran it in stealth mode (no IP address). In Phase III the team investigated two Hogwash rules and how the rules prevent potentially malicious packets from passing in. Both examples below use rules included in the default rule set that's packaged with the Hogwash installation.

Hogwash examples:

1. If one sends ten thousand forward slash characters to the Web servers at port 80, Hogwash will use part of nikto.rules to detect this:

```
<rule>
ip dst(WebServers)
tcp dst(80)
tcp
nocase(////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////)
message=Nikto7
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////
action=default
</rule>
```

   If default action to drop packets is specified in the Hogwash configuration file (see the Hogwash in Phase II) then Hogwash will actively prevent this attack from passing into the network to affect the Web servers.

2. If an attacker wants to scan a system by executing `nmap -p 6000 target-ip` to probe port 6000 on the target machine, Hogwash will detect this using part of `x11.rules`:

```
<rule>
tcp dst(6000-6005)
action=default
message=%sip:%sp->%dip:%dp X11 outgoing
```

```
</rule>
```

Just like the previous example, if the Hogwash action is specified to drop packets matching the above rule then the probing packets will be dropped.

Auditing and Logging

In Phases I and II the LTIC team set up the central log servers and configured all other machines in the environment to transmit their logs to both central log servers. The team also installed log analysis tools Lire and Swatch on the central log servers. In this phase they explored some customization and configuration options for Lire and Swatch and looked at examples of their reports.

Lire

To customize a Lire configuration:

The team followed the steps below to configure Lire.

1. Run `/$LIREPATH/bin/lr_config`
2. Select `a) Create a new file`
3. Enter filename: `lireconfig1.xml`
4. Options:

```
Lire Configurator

1. Operational settings
2. Path settings
3. External programs
4. Docbook XML settings
5. Log management settings
6. Report generation settings
7. Responder settings
8. Cron job settings
9. Edit cron jobs
        0) Save
        a) Abandon changes
```

Initially all values will be unset so that only defaults will be used. To override the default simply set the value by going through the above options. Remember to save the configurations when done setting values. The defaults are located in `/LIREPATH/share/lire/defaults/lire.xml`. The LTIC team used the defaults.

To generate a report:

Use the `lr_log2report` command to generate a report from the command line whenever desired. This command takes the Lire service name of the log file as its first argument. The

various supported services can be found by running `lr_check_service -l`. After specifying which service to generate the report for, enter the path of the desired log file that Lire is to analyze. Finally enter a path to where the report will be written to.

In earlier phases the LTIC team configured other machines to transmit their logs to the two central log servers and installed open source auxiliary audit programs Lire and Swatch on the central log servers. In this phase the team ran Lire on one of these log servers with the wrapper `lr_run` in order to filter the messages according to their preferences:

```
  /LIREPATH/bin/lr_run lr_log2report syslog < /var/log/messages >
~/report.txt
```

The above command generated a report in standard ASCII text. The report was broken down into sections such as Top 10 Hosts, Top 10 Processes, Top 50 Messages, Most Common Event Reports, etc. For example, the Top 10 Processes section looked like this:

```
/LIREPATH/bin/lr_run lr_log2report syslog < /var/log/messages >
~/report.txt
```

The above command generated a report in standard ASCII text. The report was broken down into sections such as Top 10 Hosts, Top 10 Processes, Top 50 Messages, Most Common Event Reports, etc. For example, the Top 10 Processes section looked like this:

```
Process                                                  Messages % Total
-------------------------------------------------------- -------- -------
kernel                                                        343    65.5
syslog                                                         42     8.0
sshd(pam_unix)                                                 34     6.5
sshd                                                           30     5.7
syslogd_1.4.1                                                  18     3.4
-                                                              17     3.2
xinetd                                                         10     1.9
random                                                          4     0.8
atd                                                             4     0.8
dd                                                              4     0.8
-------------------------------------------------------- -------- -------
Total for 524 records                                         524   100.0
```

If syslog recorded some FTP activity then the report reflected that:

```
                        Most Common Event Reports
                      -------------------------

Top 10 Processes

Process                                                  Messages % Total
-------------------------------------------------------- -------- -------
/USR/SBIN/CRON                                                108    59.3
```

```
sshd                                                        37    20.3
vsftpd                                                      36    19.8
syslogd_1.4.1                                                1     0.5
----------------------------------------------------------- -------- -------
Total for 182 records                                      182   100.0


sshd                                                37    20.3      20.3
   Could_not_reverse_map_address_192.168.71.\       11     6.0      29.7
     110.
   Could_not_reverse_map_address_192.168.70.\        7     3.8      18.9
     8.
   Accepted_password_for_root_from_::ffff:19\        1     0.5       2.7
     2.168.70.8_port_40028_ssh2
   Failed_password_for_root_from_::ffff:192.\        1     0.5       2.7
     168.71.110_port_32783_ssh2
   Accepted_password_for_root_from_::ffff:19\        1     0.5       2.7
     2.168.71.110_port_32786_ssh2
vsftpd                                              36    19.8      19.8
   connect_from_127.0.0.1_(127.0.0.1)               24    13.2      66.7
   connect_from_192.168.71.110_(192.168.71.1\        6     3.3      16.7
     10)
   connect_from_192.168.70.8_(192.168.70.8)          5     2.7      13.9
   PAM-listfile:_Refused_user_root_for_servi\        1     0.5       2.8
     ce_vsftpd
syslogd_1.4.1                                        1     0.5       0.5
   restart.                                          1     0.5     100.0
--------------------------------------------- -------- ------- ---------
Total for 182 records                              182   100.0     100.0
```

As an administrator one can clearly see all activities on the network that are logged to the central log server. LTIC had PortSentry running and logging to the central loggers and used Lire to centrally monitor alerts from PortSentry:

```
                    Most Common Event Reports
                      -------------------------

Top 10 Processes

Process                                                    Messages % Total
---------------------------------------------------------- -------- -------
/USR/SBIN/CRON                                                2296    75.7
kernel                                                         302    10.0
portsentry                                                     210     6.9
su                                                              99     3.3
sshd                                                            31     1.0
syslogd_1.4.1                                                   24     0.8
-                                                               22     0.7
init                                                            10     0.3
ntpd                                                             8     0.3
/usr/sbin/cron                                                   8     0.3
---------------------------------------------------------- -------- -------
Total for 3034 records                                        3034   100.0

Top 50 Messages

Message                                                    Messagse % Total
---------------------------------------------------------- -------- -------
```

```
. . . .
attackalert:_Host:_192.168.70.8_is_already_blocked._Ig\      105     3.5
  noring
. . . .
attackalert:_Connect_from_host:_192.168.70.8/192.168.7\        6     0.2
  0.8_to_UDP_port:_54321
attackalert:_Connect_from_host:_192.168.70.8/192.168.7\        6     0.2
  0.8_to_UDP_port:_31337
attackalert:_Connect_from_host:_192.168.70.8/192.168.7\        6     0.2
  0.8_to_UDP_port:_641
eth0:_no_IPv6_routers_present                                  6     0.2
attackalert:_Connect_from_host:_192.168.70.8/192.168.7\        6     0.2
  0.8_to_UDP_port:_32771
Loaded_449_symbols_from_8_modules.                             6     0.2
attackalert:_Connect_from_host:_192.168.70.8/192.168.7\        6     0.2
  0.8_to_UDP_port:_700
. . . .
-------------------------------------------------------- -------- -------
Total for 3034 records                                      3034   100.0
```

If one would like to generate a HTML report with charts, or in any other format, refer to the section "Generating a Report from a Log File" in the Lire user manual: http://download.logreport.org/pub/current/doc/user-manual/index.html.

Swatch

The team ran Swatch to examine `/var/log/messages` using its default configuration by executing the following on the command line:

```
swatch --examine /var/log/messages | more
```

If one wants to run swatch as a continuous monitor, execute this command:

```
swatch –tail-file=/var/log/messages
```

From the output the team noticed that Swatch tried to read a file named `.swatchrc` that resided in the user's home directory. This was the configuration file that hadn't been setup yet. Without this file Swatch used the default configuration of:

```
watchfor = /.*/
echo = random
```

With the above defaults the entire `/var/log/messages` file was displayed with somewhat difficult formatting. The team witnessed colorful but random highlighting or blinking of each line that caused more confusion than aid. Swatch runs much more robustly if one sets up the configuration file to match regular expressions in the log file and take desired actions.

Hence, the LTIC team tried some Swatch configurations that produced meaningful reports:

1. They configured `~/.swatchrc` to look for the words "LOGIN" and "passwd":

```
# Swatch configuration file
#
watchfor   /LOGIN/
echo inverse
watchfor   /passwd/
echo bold
```

   By running swatch `--examine /var/log/messages|more` with this configuration, only messages with the words "LOGIN" or "password" in the log file were captured. Swatch then took the actions specified and displayed the messages with word "LOGIN" in inverse (highlight) and messages with "password" in bold font.

2. The team then configured `~/.swatchrc` to look for modprobe:

```
watchfor   /modprobe/
echo inverse
```

   The output for this displayed this (in highlights that cannot be depicted here):

```
Nov 10 19:10:00 litlog01 modprobe: modprobe: Can't locate module
char-major-10-224
Nov 10 19:15:00 litlog01 modprobe: modprobe: Can't locate module
char-major-10-224
```

   The configuration file `~/.swatchrc` can also be set to e-mail alert the administrator. For more information on the configuration file see the man page for swatch, or visit this guide: http://www.cert.org/security-improvement/implementations/i042.01.html.

User Authentication

Previously the team installed and configured the Kerberos Key Distribution Center. This KDC also served as the Kerberos administrative server. The team then performed the necessary steps in configuring other systems in the environment to use the KDC for user authentication. They also explored the Kerberos authentication configuration for providing one principal with the same authority on every host within the Kerberos realm. In this phase, the team enhanced this universal configuration by providing finer authorization control for principals under one realm.

Kerberos V5

As a reminder, in order for Kerberos to authenticate users in the realm all users and hosts must have principals in the Kerberos Key Distribution Center. The team added principals that allowed equal authorization across all hosts in Phase II of this study. However, they wanted to assign these principals different authorities on different machines within the same

realm. This differed from the original configuration in Phase II in that a user defined in the central repository would not necessarily have the same authorization to all machines defined for the Kerberos realm.

The solution the LTIC team designed built on top of the original configuration and used Kerberized su (`ksu`) to change the user's identity. The original configuration added user principals in the KDC and set up system wide PAM-Kerberos authentication. This approach took it from there and came into play after the user logged into a realm host.

Setting up ksu
The `ksu` tool comes with any standard Kerberos installation. In order to use `ksu`, first make sure the `ksu` binary (`/usr/kerberos/bin/ksu`, or `/usr/lib/heimdal/bin/su`) is owned by root and setuid-root (execute `chomod u+s ksu` as root).

ksu Configuration
The `ksu` tool makes use of the `.k5login` file that resides in the target user's home directory. It contains a list of principals that are allowed the same permissions as the target user. For instance, if a user user1 needs to have the same authorities as user2, then in `/home/user2/.k5login`, `user1@REALM.COM` has to be listed. In order for user1 to have root access, `user1@REALM.COM` must be listed in `/root/.k5login`. This way the security administrator can customize how the same user with one principal can have different authorities on different machines.

Using this approach as the LTIC team did, one shouldn't add `root@REALM.COM` in the KDC so that machines that strictly use PAM-Kerberos for authentication don't allow root to log in directly. If certain users should be allowed root access to a realm host the way to do this is via ksu and having those users' principals listed in root's `.k5login` file as described above.

Below is an example of how user1 can `ksu` right into root without typing in a password because user1's principal is already in `/root/.k5login`:

```
[user1@machine1 user1]$ ksu
Authenticated user1@REALM.COM
Account root: authorization for user1@REALM.COM successful
Changing uid to root (0)
[user1@machine1 user1]#
```

This configuration can be relatively secure against malicious users with root access who want to cause harm to a system as another user. Machines set up to use Kerberos for system wide authentication prohibit a user with root access on a realm host to `ksu` or `su` to another user unless the root principal was listed in the target's `.k5login` file. In the LTIC configuration,

the root principal was removed from the KDC so there was no opportunity to even add the root principal in a user's `.k5login` file. The `su` program cannot be used neither because users were forced to use Kerberos for authentication under LTIC's strict PAM settings.

While this can be fairly secure, like all security solutions it may have its loopholes too. Consider that if user1 had root access to one of the machines in the realm, then with root privileges user1 can do anything he wants, including changing the PAM rules to disallow usage of Kerberos for authentication. Then as root user1 could su to another user and perform malicious operations as that user. The LTIC team thought of one solution to fix this problem: Disallow root access for everyone, including access through `ksu`. Instead specify which programs users are allowed to execute in the target user's `/home/`*`target-userid`*`/.k5users file`. For example, if user1 should have authority to `execute iptables,/root/.k5users` should have:

```
user1@REALM.COM   /sbin/iptables
```

Then, user1 has to enter `ksu -m /sbin/iptables` in order to execute that program as root. One could also setup userIDs with different authorities locally (userIDs such as web-admin or db-admin), and use `.k5login` files to grant authorities to real users.

Note that using `ksu` to become another user does not inherit that user's environment variables. So one must either set them manually or run the target's profile.

Heimdal Kerberos
The LTIC team had limited experience with Heimdal because they ran into a few compatibility issues between Heimdal and MIT Kerberos. Since the Web servers on Linux on zSeries were all on SUSE LINUX Enterprise Edition 8 SP3 distribution, the implementation of Kerberos V5 that came with it was Heimdal 0.4e-81.

The team first discovered that the administration APIs were not compatible between Heimdal and Kerberos when they received segmentation faults trying to add a principal to the MIT KDC via Heimdal's kadmin interface. Support found on the MIT Kerberos mailing list verified that the administration APIs were not yet compatible and claimed that everything else should be compatible. The team proceeded to find that Heimdal was able to successfully read valid `krb5.conf` and `krb5.keytab` files, as long as the team added the host and generated `krb5.keytab` on another host that was running MIT Kerberos and then transferred the `krb5.keytab` file to this host. The team was able to set the sshd-PAM module on SUSE LINUX Enterprise Edition 8 to strictly use Kerberos for authentication, and that worked - no user was able to log in using their local Linux passwords and no user was able to log in directly as root. However, the team was not able to use Heimdal's `su` for

authentication as they did with MIT's ksu, it seemed like Heimdal's `su` was not reading `.k5login`. The team also had problems in running Heimdal's `kpasswd` command. The server would reply "failed reading application request". After posting these problems to the MIT mailing list, the conclusion was that these could be compatibility bugs – but as of this writing there has been no comment from any Heimdal developers.

Remote Scan

Previously the LTIC team had only scanned from the internal zone. This time the team performed their Nessus scan from the public zone. Now that they had all the pieces that they'd wanted together, they wanted to see if an attacker could get access from the public LAN. As the LTIC team hoped the results were not interesting at all because the protocol firewall blocked all scans.

The two security vulnerabilities found were on the Caching Proxy, both were on HTTP/80:

1. The `dll /_vti_bin/_vti_aut/dvwssr.dll` seems to be present.
   "This dll contains a bug which allows anyone with authoring Web permissions on this system to alter the files of other users. In addition to this, this file is subject to a buffer overflow which allows anyone to execute arbitrary commands on the server and/or disable it."
   Nessus recommended solution: `delete /_vti_bin/_vti_aut/dvwssr.dll`
   Risk factor: High
   See also: http://www.wiretrip.net/rfp/p/doc.asp?id=45&iface=1

2. "New Atlanta's ServletExec 4.1 is a servlet Engine for IIS implemented via an ISAPI filter. By making an overly long request for a .jsp file it is possible to crash IIS."
   Nesuss recommended solution: Download patch `#9` from `ftp://ftp.newatlanta.com/public/4_1/patches/`
   Risk factor: High
   See also: http://www.westpoint.ltd.uk/advisories/wp-02-0006.txt

For the first vulnerability, the team could not find `dvwssr.ddl` on the Caching Proxy. For the second, the recommended patch path did not exist. Either the developers moved the patch to a different location and didn't inform Nessus about the change or Nessus didn't update this change yet. Thus no fixes were applied for this scan. Nessus has a very large number of security patches and holes to keep track of. For every product, server, and distribution, there are vulnerabilities, and for every attack and vulnerability there may or may not be available solutions. Cases like non-existent patches can happen, and one's mileage with Nessus can vary.

Security Process

The LTIC team stopped the study after this phase, but in real life the security process must continue in order to maintain an up to date and effective security infrastructure. This process involves having a robust backup policy, routinely updating security tools, distributions and applications, iterative vulnerability scanning with the latest Nessus plugins, and actively monitoring network activity. Re-evaluation of security products is also useful after a period of time because better products may surface and older products become obsolete. By keeping security a continual process one is keeping up with defenses against new or improved attacks and preventing systems from becoming victims of malicious activities.

Glossary

DMZ: Short for demilitarized zone, a computer or small subnet that sits between a trusted internal network, such as a corporate private LAN, and an un-trusted external network, such as the public Internet. Typically, the DMZ contains devices accessible to Internet traffic, such as Web (HTTP) servers, FTP servers, SMTP (e-mail) servers and DNS servers. The term comes from military use, meaning a buffer area between two enemies.

OSA-Express: The IBM Open Systems Adapter-Express adapter family consists of integrated hardware features that are designed to provide direct connection for zSeries and S/390 Parallel Enterprise Servers G5 and G6 to high speed switches, to other high speed servers, and to clients on LANs.

VSWITCH: Virtual Switch is a networking feature of z/VM 4.4 that provides a way to link an external network to guests under z/VM via an OSA Express without the need for a routing function. Simply put, VSWITCH provides a connection to a local LAN segment that does not require a router. Guests attached to the VSWITCH appear to be attached to the LAN that the OSA Express is attached to.

References

1. Barret, D.J., et.al., Linux Security Cookbook, O'Reilly, Sebastopol, CA, June 2003
2. Breslau, A., "Ethical Hacking", IBM Security Investigators Meeting, April 2003
3. Kargl, F., et.al., "Protecting Web Servers from Distributed Denial of Service Attacks," Proceedings of the tenth int'l conference on WWW, pp 514-524, May 2001.
4. Schneier, B., Secrets & Lies: Digital Security in a Networked World, John Wiley & Sons, New York, 2000
5. Toxen, B., Real World Linux Security, Prentice Hall, Upper Saddle River, NY, 2003

Open Source Security Product	References
Snort	http://www.snort.org
Hogwash	http://hogwash.sourceforge.net/docs/overview.html
ModSecurity	http://www.modsecurity.org
PortSentry	http://packages.debian.org/unstable/net/portsentry.html
PSAD	http://www.cipherdyne.org/psad/download/
netfilter/iptables	http://www.netfilter.org/
Lire	http://www.logreport.org/
Swatch	http://www.cert.org/security-improvement/implementations/i042.01.html
Kerberos	MIT: http://Web.mit.edu/kerberos/www/ Heimdal: http://www.pdc.kth.se/heimdal/
PAM	http://www.kernel.org/pub/linux/libs/pam/
Nessus	http://www.nessus.org http://www.sun.com/bigadmin/features/articles/nessus_scanner.html
nmap	http://www.insecure.org/nmap/
Bastille Linux	http://www.bastille-linux.org

IBM Corporation
Marketing Communications, Server Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
06/04

GM13-0636-00