# Global Information Assurance Certification Paper

# Implementing Router ACLs As Part of A Layered Security Strategy
# A Step By Step Case Study

Practical Assignment (Version 1.4, Option 2) for the GSEC Certification
By J.T. Hsieh
October 10, 2002

## TABLE OF CONTENTS

## Introduction

This paper details the step-by-step process of implementing Access Control Lists (ACLs) during a company-wide project to implement global ERP and CRM systems. ACLs were included in the project (Project Overhaul) as part of a strategy to build layered security, or defense-in-depth, into the new system infrastructure. As the GSEC course material pointed out, effective security must be part of the system design rather than an afterthought.

To provide proper context, our Before Snapshots will describe the company's security culture, network and systems environment, risk assessment, as well as the layered security strategy for Project Overhaul. These Before Snapshots will help our readers better understand some of the decisions we made to balance the many conflicting priorities of Project Overhaul while implementing ACLs.

Next, our During Snapshots will provide a step-by-step discussion of our ACL implementation, including our approach, tools, procedures, and sample ACL statements.

Finally, our After Snapshot will discuss our successes and our opportunities for future improvement. Through our discussions, we hope to provide a realistic picture of the battles that must be fought, negotiations that must be won, and compromises that must be made when attempting to improve the overall security posture of a Fortune 500 company.

## Before Snapshot - Company Security Culture

The Security team reports to an IT Manager who has three more layers of management above him before reaching the CIO. From an organizational standpoint, we are a far cry from the generally recommended structure where a CSO or CISO has responsibility for security across the enterprise and reports directly to the CEO or at least the CIO. Senior Management's lack of interest in security is clearly manifested in such an organizational structure.

Other characteristics of the company's security culture include:

- Security is viewed as an IT function, one that is not critical to the company.
- Security policies are narrowly focused, out of date, and almost universally ignored.
- Information assets are not classified and labeled according to sensitivity. The company as a whole does not have a good understanding of the value of its information assets or the threats to such assets.
- Project teams are always under tremendous pressure to deliver systems on aggressive time schedules. Project Overhaul was no exception; its schedule was so aggressive that project teams started building the infrastructure even before all the requirements were identified. This caused many complications as the project progressed.

- System developers are not required to include the Security team in initial system designs. Security is often brought in at the end after the infrastructure has already been built.
- The CEO established a company-wide goal to get "the gunk out," meaning cut costs aggressively. Budgets and resources were tight everywhere; however, security was especially impacted due to the company's view that security is a cost with questionable returns on investment.

All of the above factors contributed to an environment where security is often placed last on the list priorities.

**Before Snapshot – Network and System Environment:**
Although the company deployed strong Internet-facing controls (firewalls, DMZ, proxy servers, and secured portal gateway), the internal network was traditionally viewed as a trusted network with few internal controls. Prior to Project Overhaul, the network was essentially one flat network consisting of Token Ring LANs interconnected by an ATM backbone. Users were not separated from servers, and the network was not fully switched. Also, there were no ACLs on internal routers, no system monitoring, and no intrusion detection systems of any kind.

In other words, the company was relying on a "hard perimeter, soft center" approach to network security. The internal network, however, also included branch office networks from all over the world. Although some policies have been developed to secure the perimeter, they were not well communicated or enforced. There were many known instances of external traffic traversing the company's internal network. Such unauthorized traffic provides strong indication of the existence of backdoors (modems, wireless access points, remote users with broadband connections but no personal firewalls, etc.) into the company's internal network.

The result was a network where anyone with access, including unauthorized users coming in through unknown backdoors, can sniff for information such as usernames and passwords. Malicious users can also spoof legitimate IP addresses, hijack sessions, or perpetrate man in the middle attacks. Since there was no system monitoring or intrusion detection, any compromise would likely go unnoticed unless it caused a major disruption.

Concurrent with the launch of Project Overhaul, the company decided to migrate its internal network to Ethernet. The new Ethernet network inherited the Token Ring network's flat landscape with user workstations and servers still residing on a few big virtual LANs (VLANs). Although much of the new network was now switched, the network retained many of the vulnerabilities of the old Token Ring network. After the migration, the network was also plagued with severe broadcast storms that often slowed the network down to a crawl. Each time a server or workstation came up on the network, it would send ARP and other

types of broadcasts throughout its large VLAN. Each broadcast message must be processed by every host attached to the VLAN. This problem was further exacerbated by newly installed, malfunctioning failover NICs that repeatedly switched back and forth between the clustered NICs, sending out broadcasts each time a switch occurred.

The risk of compromise is further increased by the lack of consistent configuration management for hosts and network devices. The company does not have mandatory processes for maintaining standard system configurations. Systems are built differently each time and are often not patched to eliminate known vulnerabilities. Unix and Windows hosts as well as Cisco routers were configured with vulnerable services.

**Before Snapshot - Risk Assessment And Layer Security**

During the design phase of Project Delta, the Security team performed a risk assessment and concluded that the existing system infrastructure cannot provide adequate security and reliability for a mission critical system. The result of the assessment is presented below:

| Risk Element | Assessment |
|---|---|
| **Asset Value** | **Mission Critical**<br>(Scale: Mission Critical, Important, Useful, Non-essential)<br><br>Single point of failure with potential for seriously disrupting the company's business processes on a global scale. |
| **System Vulnerabilities** | **High**<br>(Scale: High, Medium, Low)<br><br>There are no standard processes to ensure hosts are properly configured. The vulnerabilities of improperly configured hosts and network devices are well documented and publicized. Automated hacker tools and malicious codes to exploit known vulnerabilities are readily available and require little technical expertise to use. |
| **Probability of Threats** | **High**<br>(Scale: High, Medium, Low)<br><br>The company's increasingly open network (accessible to brokers, field offices, sales offices, plants, business partners, contractors, remote users, etc.) means increased threat that a legitimate but unethical user can exploit vulnerabilities in The company's systems. The company's internal network is also vulnerable to unauthorized access. |

| Risk Element | Assessment |
|---|---|
| **Overall Assessment** | **High Risk to Mission Critical Assets**<br>Such a high risk exposure requires strong controls to minimize the possibility of major losses. |

The above assessment is based on the concept that risk only exists when real threats can exploit actual vulnerabilities to adversely impact valued assets. We purposely kept the assessment result simple for ease of communication.

GSEC's Day Two material describes risk as a function of threat and vulnerabilities. Although it could be argued that threat inherently includes the idea of asset value, we decided to separately discuss asset value to emphasize the mission critical nature of Project Overhaul's systems.

The Security Team was successful in getting approval for a strategy to improve the overall security of Project Overhaul's system infrastructure. The strategy called for the implementation of the following security layers into the infrastructure:

- **Data Security Layer** – Data will be classified and labeled. Owners and custodians will be assigned. Security measures will be commensurate with the data classification.
- **User Awareness and Security Policies** – Information security policies will be updated and expanded as required. Company-wide user awareness programs will communicate approved policies to all employees and, when appropriate, to business partners.
- **Physical Security** – Servers, desktops, and laptops will be secured with appropriate measures.
- **Network Security** – Project Overhaul servers will be located in their own private VLANs protected by ACLs. Both NIDS and HIDS will monitor network for unauthorized activities.
- **Application Security** – User access will be limited to what is authorized.
- **Operating System Security** – Configuration standards will be developed and applied to all servers and network devices. Symantec's Enterprise Security Manager (ESM) and other monitoring tools will be deployed to ensure servers and network devices comply with configuration standards.

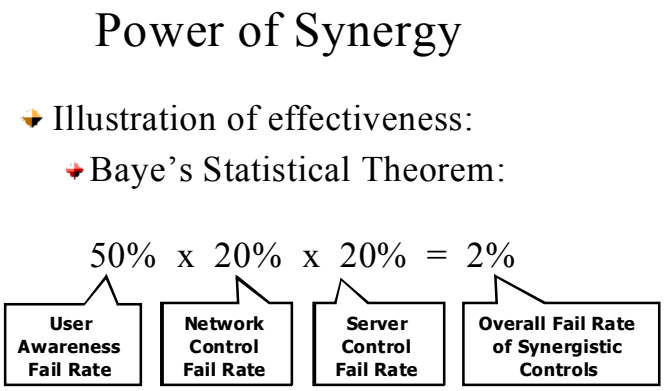
The above security layers were designed to work together synergistically, each layer compensating for weaknesses in other layers. Consequently, although each layer may not be completely effective by itself, all the layers work together to improve overall effectiveness.

The benefit of this defense-in-depth approach is well presented by Peter Tippett in his Information Security Magazine article on "synergistic security." [i] Tippett points out that "any single control that's 99 percent effective would cripple" a business' productivity. He suggests that a better way is to layer together more

user-friendly controls, which may be less effective by themselves, but just as or even more effective as a whole.

To illustrate this, Tippett uses the statistical theory called Baye's Theorem, "which describes a 'new' probability (control effectiveness) given a 'prior' probability... If one control is 80 percent effective, then it fails one out of five times. Two controls, each 80 percent effective, together will fail one out of 25 times. Three 80 percent effective controls, operating together, will fail one out of 125 times. In other words, they will succeed with a likelihood of 99.2 percent."

We developed the diagram below to help senior management appreciate the power of layered, synergistic security:




Although we were happy to win approval for our layered security strategy, we knew implementation would not be easy. The company culture still required that we obtain consensus from each of the project teams affected by the strategy, including Unix sysadmins, Windows administrators, Network engineers, and all the application teams.

This security-by-consensus approach turned out to be extremely difficult to practice. We had to repeatedly educate and negotiate, going back to the drawing board again and again to come up with the controls for each of the security layers in our strategy.

Due to conflicting priorities, resources constraints, and extremely aggressive deadlines, this security-by-consensus approach made it impossible to fully implement our layer security strategy. We had to settle for the highest impact controls that could be practically implemented under the high-pressure environment.

**During Snapshot – Understanding ACL Types**

The first step in our ACL implementation was to understand the type of ACLs available to us. Day 2 of the GSEC material mentioned three basic types of ACLs: Standard, Extended, and Reflective. Other types of ACLs are also available for our Cisco Catalyst 6509 Switches (6509 Switches). These switches run on hybrid software: Catalyst OS Version 6.3(5) on the Supervisor Engine and IOS Version 12.1(3a)E4 on the Multilayer Switch Feature Card (MSFC). The MSFC provides the 6509 Switches with router functions.

As described in Chapter 23 of Cisco's "Catalyst 6500 Series IOS Software Configuration Guide," [ii] the other ACL types not mentioned in the GSEC material and applicable to our discussion are:

**MAC Address-Based Traffic Blocking**

The 6509 Switches can be configured to block all traffic to and from a MAC address in a specified VLAN.

**VLAN Access Control Lists (VACLs)**

The above referenced IOS Configuration Guide explains that "VACLs can provide access control for all packets that are bridged within a VLAN or that are routed into or out of a VLAN or, with releases 12.1(13)E or later, a WAN interface. Unlike regular Cisco IOS standard or extended ACLs that are configured on router interfaces only and are applied on routed packets only, VACLs apply to all packets and can be applied to any VLAN or WAN interface. VACLS are processed in hardware. VACLs use Cisco IOS ACLs. VACLs ignore any Cisco IOS ACL fields that are not supported in hardware.

"You can configure VACLs for IP, IPX, and MAC-Layer traffic. VACLs applied to WAN interfaces support only IP traffic.

"When you configure a VACL and apply it to a VLAN, all packets entering the VLAN are checked against this VACL. If you apply a VACL to the VLAN and an ACL to a routed interface in the VLAN, a packet coming in to the VLAN is first checked against the VACL and, if permitted, is then checked against the input ACL before it is handled by the routed interface. When the packet is routed to another VLAN, it is first checked against the output ACL applied to the routed interface and, if permitted, the VACL configured for the destination VLAN is applied. If a VACL is configured for a packet type and a packet of that type does not match the VACL, the default action is deny."

VACLs and CBACs (see below) cannot be applied to the same interface.

**Context-Based Access Control (CBAC)**

According to the "Cisco IOS Security Configuration Guide,"[iii] "CBAC intelligently filters TCP and UDP packets based on application-layer protocol session information. You can configure CBAC to permit specified TCP and UDP traffic through a firewall only when the connection is initiated from within the network you want to protect. CBAC can inspect traffic for sessions that originate from either side of the firewall, and CBAC can be used for intranet, extranet, and Internet perimeters of your network.

"Without CBAC, traffic filtering is limited to access list implementations that examine packets at the network layer, or at most, the transport layer. However, CBAC examines not only network layer and transport layer information but also examines the application-layer protocol information (such as FTP connection information) to learn about the state of the session. This allows support of protocols that involve multiple channels created as a result of negotiations in the control channel. Most of the multimedia protocols as well as some other protocols (such as FTP, RPC, and SQL*Net) involve multiple channels."

CBAC provide stateful inspection, but is only available as part of the Cisco IOS Firewall option. Each CBAC statement also uses up a small amount of the router's memory. Cisco recommends judicious use of CBAC to avoid exhausting the router's memory resources.

**During Snapshot – Matching Features to Needs**

To help the reader understand our choice of ACL types, we will next describe the system landscape for Project Overhaul and explain our objectives for ACLs.

The network hardware used by Project Overhaul consists of five 6509 Switches:

- Two core switches with MSFCs to provide load balancing, failover-enabled routing and switching.
- Three switches without MSFCs to provide the required switching among the numerous servers. These switches do not provide routing functions.

The infrastructure design for the ERP system includes the following elements:

- Unix servers running Sun Solaris 8
- Each server is attached to three VLANs:
  - Client - for client (user) traffic,
  - Data - for system to system (interface) traffic, and
  - Backup - for backup traffic.
- Three separate system environments:
  - Development (Dev),
  - Quality Assurance (QA), and
  - Production (Prod)

The three environments share one Backup VLAN, so there is a total of seven VLANs for the ERP system.

The infrastructure for the CRM system is similar and includes:

- Unix servers running Solaris 8
- Windows servers running Windows NT
- Each server is also attached to three VLANs:
  - Public – for all traffic from outside the CRM VLANs except backup
  - Private – non-routed VLAN dedicated for server-to-server traffic between the CRM servers on the same VLAN.
  - Backup – for backup traffic
- Three separate system environments:
  - Development (Dev),
  - Quality Assurance (QA), and
  - Production (Prod)

There are a total of nine VLANs for the CRM system.

Although we would like to see ACLs implemented through out the company's network, the scope of our ACL work for Project Overhaul was realistically limited to the protection of the above 16 VLANs. We later eliminated from scope the CRM Dev VLANs because the environment was so dynamic and the costs outweighed the benefits.

The Backup Team also decided that the Unix-based backup system would have an interface to the ERP Backup VLAN to improve performance. This effectively made the backup system into a backdoor around whatever ACLs we implement. To better control access to this backdoor, we required the Backup Team to implement RSA's SecurID on the backup system. SecurID requires users to use one-time passwords provided by a token. We will also monitor the system's configuration via ESM for changes, including the addition of users or services.

Our main objective for the ACLs is to filter the traffic going into the VLANs within our scope; however, we also decided to filter the traffic coming from these VLANs to prevent backdoors and other means of circumventing network controls. We did not view these dedicated VLANs as "trusted."

After carefully considering our requirements, we decided on implementing extended ACLs for the following reasons:

1. We needed to filter on protocol and ports, so standard ACLs and MAC address-based filtering were ruled out.

2. We did not need to filter all packets, only routed packets, so we decided against VACLs. Deploying VACLs would mean extra work since we would have to configure them in all five 6509 switches. With extended ACLs, we

only needed to concern ourselves with the two core switches with routing functions.

3. Since we wanted to filter both inbound and outbound traffic, we also decided against reflective ACLs. According to Cisco's IOS Security Configuration Guide, reflective ACLs work best when applied to only one direction.

4. CBACs were attractive especially for multi-port protocols such as RPC and for UDP packets; however, we did not have any budget for the implementation of the IOS Firewall option. There were also some concerns over the processing overhead imposed by CBACs.

**During Snapshot – ACL Application Considerations**

Extended ACLs provide filtering on the network layer (IP addresses) and on the application layer (ports). The syntax for extended ACLs can be broken down to the following nine elements (based on the diagram "Anatomy of An Access List" from Peter Morrissey's article on "Demystifying Cisco Access Control Lists." [iv] ):

| Element 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| access-list 100 | permit | tcp | 10.8.0.0 | 0.0.0.127 | 10.20.20.0 | 0.0.0.255 | eq | 23 |
| access-list 120 | deny | icmp | host 192.168.33.45 | | 192.168.50.0 | 0.0.0.255 | | echo-reply |
| ip access-list extended v10in | permit | ip | any | | any | | | |

| Element No. | ACL Element | Required Optional | Description and Remarks |
|---|---|---|---|
| 1 | access-list (100-199)<br><br>ip access-list extended <name> | Required | Extended ACLs can be named or numbered.<br><br>Numbered: Every extended access list has a number from 100 to 199, which identifies the list in two places: 1) when building the ACL, each statement belonging to the ACL must be labeled with the same number; 2) when applying the ACL to a router interface, it is referenced by its ACL number.<br><br>Named: a name can be used instead of numbers using the alternative command syntax. The name must only use alphanumeric characters. |
| 2 | deny<br>permit<br>remark | Required | A permit or deny rule must be applied to every statement in the ACL.<br><br>Remark is ignored by the router and is used for clarifying remarks. |

| Element No. | ACL Element | Required Optional | Description and Remarks |
|---|---|---|---|
| 3 | ICMP<br>IP<br>OSPF<br>TCP<br>UDP | Required | A protocol must be specified. Beside those listed here, other protocols can also be specified; we only listed the ones pertinent to our discussion here. |
| 4 | xxx.xxx.xxx.xxx.<br>any<br>host xxx.xxx.xxx.xxx | Required | Source IP address. Every statement must specify a source IP address. **Any** will match any address; while **host** denotes a single IP address. |
| 5 | xxx.xxx.xxx.xxx | Optional if **any** or **host** specified above | Source wildcard mask. The wildcard mask is written as the inverse of the source IP address subnet mask. For example,<br>• a source ip address range with a 255.255.255.0 subnet mask would require a wildcard mask of 0.0.0.255<br>• 255.255.255.128 would need 0.0.0.127<br>• 255.255.255.192 would need 0.0.0.63<br>• 255.255.255.224 would need 0.0.0.31<br>• 255.255.248.0 would need 0.0.7.255<br><br>If **host** was specified in element 4, there is an implicit wildcard mask of 0.0.0.0, and it is not necessary to explicitly specify the mask. |
| 6 | xxx.xxx.xxx.xxx.<br>any<br>host xxx.xxx.xxx.xxx | Required | Destination IP address. Every statement must have a destination IP address. Syntax is the same as for the source IP address (element 4). |
| 7 | xxx.xxx.xxx.xxx | Optional if **any** or **host** specified above | Destination wildcard mask. Syntax is the same as for the source wildcard mask (element 5). |
| 8 | eq<br>gt<br>lt<br>neq<br>range <n> <n> | optional | Not applicable if IP, ICMP, or OSPF was specified in element 3.<br><br>For TCP and UDP, the following functions are available:<br>eq = equal<br>gt = greater than<br>lt = less than<br>neq = not equal<br>range = all ports from the first specified number to the second, inclusive. For example, **range 21 23** means ports 21, 22, 23. |
| 9 | <0-65535><br><named protocols> | optional | If filtering on TCP or UDP ports is desired, specify the port numbers here. Alternatively, well known ports can be specified by the application name (e.g., telnet, ftp, ftp-data, http, etc.) For **range**, specify the range as two numbers separate by a space.<br><br>For ICMP, specify the type either by number or by protocol name (e.g., echo, echo-reply). |

Cisco IOS also allows port specifications (elements 8 and 9) to precede destination address specifications (elements 6 and 7). Thus,

access-list 100 permit tcp any any eq ftp-data

can also be written as:

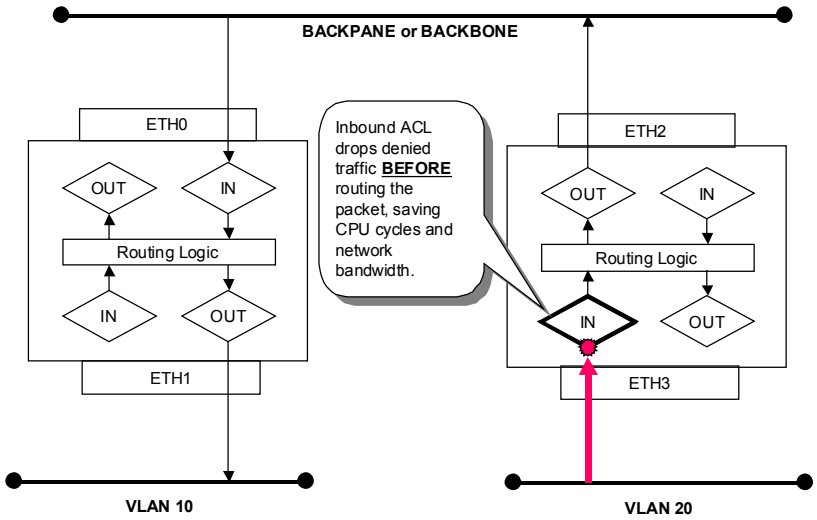
access-list 100 permit tcp any eq ftp-data any

Some have argued that the latter syntax results in faster processing by the router because unauthorized packets will be dropped faster; however, we have not found strong evidence to prove this assertion. We did discover that the latter syntax is preferable when applied to ftp-data. This will be discussed in detail in the next section on capturing traffic requirements.

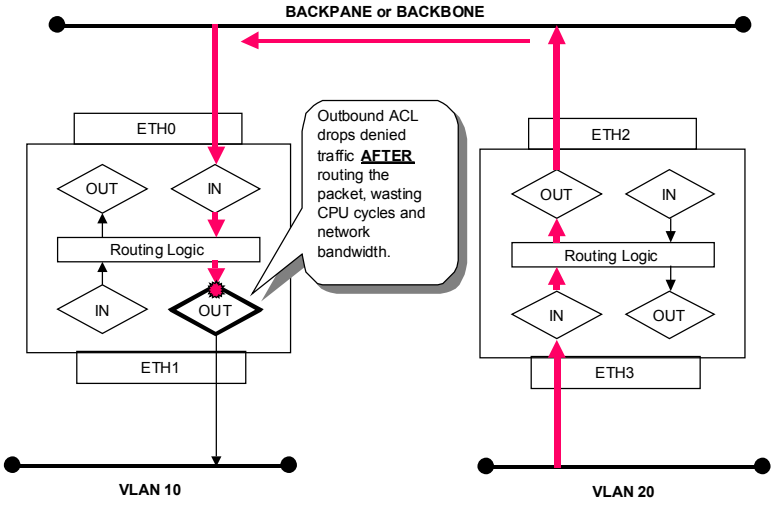
Once we understood the functionality of extended ACLs, we had to decide where to apply them. From the standpoint of efficiency and resource utilization, the best way to apply extended ACLs is to filter the inbound interfaces. As explained by the GSEC material, with inbound ACLs, the router filters packets ***before*** the routing logic, saving network bandwidth and router processing cycles. Outbound ACLs filter packets ***after*** the routing logic, wasting the network bandwidth and processing cycles required to deliver the packet to the outbound interface before it is dropped.

To illustrate this difference, I expanded the diagram from the GSEC Material into the following two diagrams:

**INBOUND ACLs**



**OUTBOUND ACLs**




To provide the control desired, ACLs would need to be applied to ALL inbound interfaces on ALL routers connected to the company network. Although this is a desirable goal, there just was no practical way to accomplish such a feat in the time frame allotted and with the resources available. Consequently, we had to settle for applying ACLs to both the inbound and the outbound interfaces for the 10 VLANs remaining in our scope. Twenty ACLs were needed as shown in the table below:

| No. | VLAN | Interface | Environment |
|---|---|---|---|
| 1 | VLAN 10 | Inbound | ERP Dev – Client |
| 2 | VLAN 11 | Inbound | ERP Dev – Data |
| 3 | VLAN 20 | Inbound | ERP QA – Client |
| 4 | VLAN 21 | Inbound | ERP QA – Data |
| 5 | VLAN 30 | Inbound | ERP Prd – Client |
| 6 | VLAN 31 | Inbound | ERP Prd – Data |
| 7 | VLAN 40 | Inbound | CRM QA – Pub |
| 8 | VLAN 41 | Inbound | CRM QA – Bkp |
| 9 | VLAN 50 | Inbound | CRM Prd – Pub |
| 10 | VLAN 51 | Inbound | CRM Prd – Bkp |
| 11 | VLAN 10 | Outbound | ERP Dev – Client |
| 12 | VLAN 11 | Outbound | ERP Dev – Data |
| 13 | VLAN 20 | Outbound | ERP QA – Client |
| 14 | VLAN 21 | Outbound | ERP QA – Data |
| 15 | VLAN 30 | Outbound | ERP Prd – Client |
| 16 | VLAN 31 | Outbound | ERP Prd – Data |
| 17 | VLAN 40 | Outbound | CRM QA – Pub |
| 18 | VLAN 41 | Outbound | CRM QA – Bkp |
| 19 | VLAN 50 | Outbound | CRM Prd – Pub |
| 20 | VLAN 51 | Outbound | CRM Prd – Bkp |

**During Snapshot – Capturing Traffic Requirements:**
We had settled on an approach for applying ACLs. Now, we needed to determine what traffic to permit and what traffic to block. Getting a good set of traffic requirements to build our ACLs turned out to be the greatest challenge of this project. This was due to the following factors:

- The application teams and the system administrators could not provide us with a clear set of requirements for their servers and applications. From our first initial network traces, we quickly saw that the requirements provided us were woefully incomplete.

- We repeatedly discovered that the system landscape was not being built according to design. By the time we facilitated the correction that could be made, we were starting QA integration testing. Unfortunately, the misconfigured ERP Dev environment had become so critical to the progress of the project, it could not be touched until some indefinite future time after Project Overhaul's Go-Live.

- System and application requirements changed continuously, even during QA. In attempts to make up for lost time, many temporary configuration changes were made to shortcut QA testing. This situation made it impossible to lock down the traffic requirements for our ACLs.

To further add to the challenge, we were told that the burn rate for Project Overhaul was millions of dollar per MONTH, so we were told our most important priority is to avoid causing any major disruptions as we build and apply our ACLs. The vice-president of IT, in charge of Project Overhaul's infrastructure, personally made it clear to us that any major disruptions means we will do without ACLs at least until after Project Overhaul's Go-Live.

We understood that to mean we will do without ACLs indefinitely since there are two more phases of Project Overhaul scheduled for the next few years. This would mean the company's mission critical systems would not be adequately protected for a prolonged period. This was not an acceptable situation, and the Security team decided that we would implement ACLs even if only partially effective for the following reasons:

- Partial protection is still better than no protection, especially in a layered security approach. Holes in our ACL controls can be mitigated by other security measures such as configuration monitoring and IDS.
- We needed to get the company acclimated to ACLs. Much of the resistance to our efforts is due to the lack of experience with ACLs.
- As the company gets more comfortable with ACLs and as the infrastructure stabilizes, we can gradually tighten the control to improve effectiveness.

- We want ACLs to be part of the development environment for Phase 2 and Phase 3 to force development teams to work out ACL requirement as they progressed.

To meeting the challenge of building ACLs without clear and stable requirements, we adopted the following implementing strategy:

1. We placed sniffers in Project Overhaul VLANs and used the traces to get a better understanding of the network traffic requirements.
2. Requirements were documented and ACLs developed to match the traffic identified from our traces.
3. "Permit-any" statements were temporarily placed at the bottom of each ACL so we can identify any missed traffic requirements without causing disruptions.
4. As the environments stabilized and as we slowly ferreted out the required ports and address ranges, we eliminated the permit-any statements one by one to the extent practical.

Each of the above steps is discussed in detail below.

**Using Sniffers to Capture Traffic Requirements**

Since we did not have a budget for implementing ACLs, we used free, open source sniffers. We also built six Redhat Linux 7.2 laptops because we found Linux provided many more tools and much better flexibility than Windows 2000 (W2K).

Below is a list of sniffers that we tested and our experience with them:

**Ethereal for Windows** – We had difficulty using the Windows version of Ethereal on our company's standard W2K images. It would cause W2K's IP stack to stop working, requiring that we delete and reinstall the TCP/IP protocol in the Network Connections settings. This may be due to the custom, frozen Registry settings in our company's standard W2K image.

**Ethereal for Linux (Redhat 7.2)** – Ethereal on Linux is a powerful packet analysis tool with many features for analyzing the various layers of a TCP/IP Ethernet packet. The X-Windows GUI is easy to use and read. The downside is Ethereal consumes lots of system resources. A 15-minute trace on a busy 100MB Ethernet network could easily fill up a 100 MB file. Unfortunately, on our IBM T-20 laptop (Pentium III, 900 MHz, 256 MB Ram), Ethereal had difficulty dealing with files bigger than 20MB. It would cause the hard disk to thrash. Read filtering stops working, and printing and scrolling slows to a crawl. Outputting a flat text file is also a cumbersome process and painfully slow for any file larger than 20 MB.

**Tethereal (Linux)** – This is a non-gui version of Ethereal that does not require X-Windows to run. Outputting is similar to TCPdump except the program will translate ports into protocol names based on IANA port listings. This made Tethereal more difficult to use for capturing port numbers and IP addresses.

**TCPDump (Linux)** – This is the sniffer we finally chose for our network traces. We learned how to read the output from the GSEC Day 1 material. With TCPDump and appropriate capture filters, we can run the trace all day, parse the captured data using Linux shell commands such as sort and grep, and quickly isolate the unique sockets (port and address combinations) for identifying traffic requirements.

Analyzing the millions of packet we captured was tedious and arduous. Below are steps we gradually developed to facilitate the analysis. These steps worked even for trace files as large as 10GB. Our descriptions assume knowledge of the following common Linux commands:

| Shell Command | Function |
|---|---|
| \|<br>(shift and \ key) | Pipe. Takes the output of the previous command and sends it as input to the following command |
| ><br>(shift and . key) | Redirect. Takes the output of previous command and writes it to specified file. |
| Cat | Output file listed. |
| Grep | Searches a file for a given pattern. Useful options include:<br><br>-v Inverse match, reject lines matching pattern<br><br>-w Match whole word. Specified pattern must not be preceded or followed by an alphanumeric character.<br><br>-f Pattern specified in the following file. One pattern per line. |
| Sort | Sort a file based on specified criteria. Useful options include:<br><br>-k *n,n* specifies the beginning and ending fields to be used as sorting criteria.<br><br>-u eliminates all duplicate occurrences of the fields specified by -k.<br><br>-o output result to file named. |

**Steps for Trace Analysis:**

Step 1. We first spanned the target VLAN to an accessible switch port on the core switches. This caused all VLAN traffic to route to that switch port. The 6509 Switches only allowed us to span two VLANs per switch.

We configured our Linux laptops with three interfaces: two to sniff on the span ports and one to allow remote access. This made it easy to remotely setup, monitor, and change captures.

Step 2. On our Linux laptops, we used the following options when capturing packets with TCPDump:

-nnttttl -nn specifies that all IP addresses and port numbers remain in numerical format.

-tttt causes TCPDump to output the date and the time.

-l buffers the output to facilitate piping to another command.

-F This option uses a named file as the capture filter.

To eliminate intra-VLAN traffic, we would launch two sessions of TCPDump. The first one used the following filter file (assuming we are sniffing VLAN 20 with IP address 10.8.0.0/26):

v20in.filter:
ip and not net 224.0.0.0/8 and not dst net 10.8.0.0/26

(This filter captures only inbound IP traffic and filters out HSRP packets used for failover operation of the core 6509 Switches.)

The second session used a second filter file:

v20out.filter:
ip and not net 224.0.0.0/8 and not src net 10.8.0.0/26

(This filter captures only outbound IP traffic and filters out HSRP packets.)

Step 3. To further reduce the size of the trace files, we piped the TCPDump output to the following grep command:

grep –vwf *grepack*

*grepack* is a text file with the following lines (text in parentheses are our clarifying remarks and not part of the file):

```
ack (matches the ACK flag)
R  (matches the RESET flag)
P  (matches the PUSH flag)
ok (the last four lines matches packets for established NFS)
nfs
proc
null
```

The above grep command eliminated all packets from established TCP connections. In our experience, this often reduced the size of the trace file by some 80%.

Step 4. We then redirected the grep output to a flat text file and used the tail command to view the output as it is written to the file. So the complete TCPDump command looks like this:

```
tcpdump –nnttttl –F v20in.filter –i eth1 |grep –vwf grepack >v20in.txt & tail –f v20in.txt
tcpdump –nnttttl –F v20out.filter –i eth1 |grep –vwf grepack >v20out.txt & tail –f v20in.txt
```

After entering the commands above, the screen will scroll through the tail records of the trace file as they are written (almost real time – there is a slight delay for the buffer to fill up).

Using the above method, we found that we could keep the TCPDump sessions going for days without filling up our hard disk. Below is a sample of the output for the inbound trace:

```
09/24/2002 21:40:14.281815 10.8.0.10.60762 > 10.8.10.10.32773:  udp 96 (DF)
09/26/2002 17:33:13.608003 10.8.0.10.630 > 10.8.10.10.2049: S 105513541:105513541(0) win 24820 <nop,nop,sackOK,mss 1460> (DF)
```

Notice the TCP packet only has the SYN flag. The date and the time fields are useful for chronological analysis after combining the two trace files (inbound and outbound).

Step 5. Once we finished capturing our trace files, we used the following sort command to combine the inbound and outbound trace files while simultaneously eliminating duplicate combinations of addresses and ports (sockets):

```
sort –k 3,6 –u –o v20-uniq.txt *.txt
```

The result of the above command is the file v20-uniq.txt, which will only contain unique combinations of fields 3 through 6 of the trace file. From the above sample, the reader can see fields 3 through 5 records the socket combination while field 6 records the protocol.

Step 6. Initially, we reviewed the entire file of unique sockets to get a feel for the traffic patterns. We found importing the text file into Microsoft Excel facilitated our review.

Once we started understanding the traffic patterns, we captured the requirements in an Excel spreadsheet like the one below:

| Applied Interface | Source Address | Dest Address | udp port | tcp port | Services |
|---|---|---|---|---|---|
| v101-in | any | any | | established | all TCP |
| v101-in | any | 224.0.0.0/8 | 1985 | | hsrp |
| v101-in | any | 224.0.0.0/8 | | | Ospf |
| v101-in | v101-10.8.10.0/24 | v33-10.8.33.0/24 | | 3300 | RFC |
| v101-in | v101-10.8.10.0/24 | v201 - 10.8.0.0/25 | any | any | all |
| v101-in | v101-10.8.10.0/24 | v111-10.8.20.0/25 | any | any | all |
| v101-in | v101-10.8.10.0/24 | svrdsd (10.8.64.179) | | 2049 | nfsd(cots) |
| v101-in | v101-10.8.10.0/24 | svrdsd (10.8.64.179) | 111 | | nfs-rpc |
| v101-in | v101-10.8.10.0/24 | svrdsd (10.8.64.179) | >32000 | | nfs-rpc |
| v101-in | v101-10.8.10.0/24 | any | | 21-23 | ftp-ssh-telnet |
| v101-in | v101-10.8.10.0/24 | any | 22 | | ssh |

Step 7. As we captured the traffic requirements, we also started compiling a file for use in filtering out packets that match already known requirements. We will call this file greplist. For each requirement captured and verified, we would attempt to develop an entry for it in greplist. The table below shows a sample entry in greplist for the fourth requirement shown in the spreadsheet above.

The first three lines do not need entries because packets matching these lines were already filtered out during capture.

| Greplist Entry | Remarks |
|---|---|
| 10.8.10[.].*[.].*>.*10.8.33[.].*[.]3300: | Grep interprets a period "." as a wild card for any one character. The [ ] causes grep to interpret the period literally. |

To use the greplist, we modified our use of the sort command from Step 5 as follows:

```
sort –k 3,6 –u *.txt |grep –vwf greplist >v20-done.txt
```

With the above command we not only eliminated duplicate sockets, we also filtered out packets matching known requirements. The resulting file v20-done.txt should contain mostly packets that require further investigation.

**Trace Analysis Results**

Most application traffic patterns were straight forward, boiling down to the identification of IP addresses and destination ports. However, four situations required special consideration. These are: 1) FTP, 2) NFS, 3) other applications that requires a wide range of dynamic ports, and 4) secure use of telnet and FTP.

First, let us consider the situation with FTP. There are two modes of FTP: active and passive. An excellent explanation of the traffic patterns for these two modes can be found in Jay Ribak's article "Active FTP vs. Passive FTP, A Definitive Explanation." [v]

In the active mode, the following traffic pattern occurs:

1. A client launches a connection to the server's port 21. The source port on the client is a randomly chosen, unused port above 1024 (dynamic port). For this discussion, let's say port 2000.

2. The server will respond from port 21 to the client's dynamic port (2000) to establish the connection. Authentication takes place via this connection.

3. When the authenticated user starts a file transfer (get, put, mget, mput, etc.), the server will initiate a new connection from its port 20 targeting the next port number above the dynamic port used to establish the initial connection with port 21. In this case, the target port would be 2001; however, it could be any port greater than 1024 depending on which dynamic port was randomly chosen in step 1.

The above traffic pattern would require ACLs to permit all TCP ports above 1024 to be opened to the server. In the case of Project Overhaul's systems, the application teams require the ability to FTP from the ERP and CRM servers to a wide range of IP addresses outside of their VLANs. Without some sort of stateful inspection, we would have to open all TCP ports above 1024 to the same wide range of IP addresses to enable FTP transfers. This practically renders the outbound ACLs useless.

The traffic pattern for passive FTP is not much better:

1. Again, a client launches a connection from a dynamic port (2000 in this case) to the server's port 21.

2. As in active mode, the server will respond from port 21 to the client's dynamic port (2000) to establish the connection.

3. When the authenticated user starts a file transfer, the ***client*** this time will initiate a new connection from one port higher (2001) than the dynamic port used in step 1. The target port for this new connection is another randomly selected dynamic port greater than 1024.

The passive mode traffic pattern requires ACLs to permit all TCP ports above 1024 to be opened the ***client***. For Project Overhaul, this practically renders the inbound ACLs useless.

Fortunately, our tests show that ACL statements written using the name ftp-data seem to provide stateful inspection in the 6509 Switch. However, the syntax of the ACL statement must be as follows:

```
access-list 100 permit tcp any eq ftp-data any
```

Ftp-data must be specified before the destination address. In our experience, the above statement permits both active and passive FTP data transfers. If the statement is written in the alternative syntax as shown below, both active and passive FTP fails.

```
access-list 100 permit tcp any any eq ftp-data
```

Besides FTP, NFS on Sun Solaris is another application that uses multiple ports to establish connections. From our network traces, we found that NFS requires Sun remote procedure calls (RPC). This means the NFS client needs access to the following ports on the NFS server: 2049 tcp, 111 UDP, and a static UDP port somewhere higher than 32000. The last port is configured by the sysadmin. Unfortunately, we found that the port chosen is not consistently configured across the ERP or CRM servers. Consequently, our ACLs had to permit all UDP ports greater than 32000.

The NFS server, on the other hand, requires access to the all UDP ports greater than 1024 on the client. This is because when the client initiates a RPC call, it randomly chooses a dynamic UDP port as the source port.

Other applications also require a wide range of ports to be open to them. These include DNS, our third party event monitoring system, and trusted hosts used by the Unix team to perform maintenance tasks. The servers that support these applications become backdoors around our ACLs.

To better control access to these backdoors around our ACLs, we will require SecurID to be implemented on these applications servers. We also require SecurID authentication for all telnet and FTP sessions to any of the ERP or CRM Unix servers.

**Using Permit-Any Statements**

Once we felt confident we had captured most of the traffic requirements, we built our ACLs and prepared them for implementation. To ensure that we do not cause any disruptions, we placed a number of permit-any statements at the bottom of each ACL. These permit-any statements look like this:

```
access-list 129 permit ip 192.168.94.0 0.0.0.255 any
access-list 129 permit ip 10.8.10.0 0.0.0.127 any
access-list 129 permit ip 10.8.20.0 0.0.0.127 any
access-list 129 permit ip 10.8.0.0 0.0.0.127 any
access-list 129 permit ip 192.168.33.128 0.0.0.63 any
access-list 129 permit ip 192.168.45.0 0.0.0.63 any
access-list 129 permit ip 192.168.45.64 0.0.0.63 any
access-list 129 permit ip 192.168.36.0 0.0.0.127 any
access-list 129 permit ip 192.168.41.64 0.0.0.63 any
access-list 129 permit ip 192.168.64.0 0.0.7.255 any
access-list 129 permit ip any any log
```

Each of the statements, except the last one, represented a discrete segment of our internal network. The last one represents the world outside of our internal network. The statements are listed in order of threat, with the bottom statement representing the highest threat. As shown above, the last statement is being logged to see what hits it. In the 6509 Switches, approximately every 10th packet that hits a logged statement will be written to system log. The packets that hit the last statement represent either missed traffic requirements or unauthorized traffic.

Once a missing traffic requirement is identified, we add a statement to the ACL to match it. This prevents the same packets to hit the bottom statement again. We continue this process until no more missing requirements are logged. The bottom statement is then removed and the next statement is logged.

We used this process to finalize the ACLs for all network segments outside of our data center. We were very happy to report that we did not cause any major disruptions as we tightened the ACLs. The remaining permit-any statements are only for our server VLANs. As I write this, Project Overhaul's infrastructure is still undergoing change. Go-Live is 54 days away. We will wait for the environment to stabilize before we attempt to remove any more permit-any statements. Meanwhile, we feel we have at least provided ACL protection against our greatest threats.

**After Snapshot – Step In the Right Direction**

Although we did not fully achieve the objectives of our layered security strategy, we consider our ACL work to be a success for the following reasons:

1. We proved to the company that we could implement ACLs and ACLs do indeed bring benefits. The network is more stable. End-user access to the ERP and CRM servers has been limited to only what is required. Developers and system administrator have a much clearer understanding of the traffic patterns for their systems. In addition, we showed we could quickly fix the few minor issues that did arise.

2. ACLs are fulfilling their required functions as part of a layered security strategy. ACLs are mitigating weaknesses in the other layers, while the other layers cover weaknesses in the ACLs. We believe we are achieving the synergistic effects described by Baye's Theorem mentioned above.

3. Our education and negotiation efforts have raised the overall security consciousness of our IT organization. We expect our Phase 2 and 3 work will require much less time spent on education, negotiations, and network traces to capture traffic requirements.

4. The ACL team has also earned the respect of the other infrastructure teams. We are being consulted more often and earlier in the project cycle. Our experience with network monitoring has proven to be useful in helping other teams troubleshoot issues. We feel we have established a good working relationship that can be leveraged to continue improving the overall security posture of the company.

The compromises we were forced to make, not only with the ACLs but also with the other layers of our security strategy, highlighted the fact that we still have a long way to go. Our experience motivates us to work even harder on educating upper management on the importance of security and the role they should play to move us away from our very ineffective security-by-consensus culture.

[i] Tippett, Peter. "Defense-In-Breadth—How To Reduce Risk Using 'Synergistic Security.'" www.infosecuritymag.com. February 2002.
URL: http://www.infosecuritymag.com/2002/feb/columns_executive.shtml

[ii] Cisco Systems. "Catalyst 6500 Series IOS Software Configuration Guide." www.cisco.com. October 3, 2002.
URL: http://www.cisco.com/univercd/cc/td/doc/product/lan/cat6000/12_1e/swconfig/secure.pdf

[iii] Cisco Systems. "Cisco IOS Security Configuration Guide." www.cisco.com. September 15, 2002.
URL: http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121cgcr/secur_c/scprt3/scdcbac.pdf

[iv] Morrissey, Peter. "Demystifying Cisco Access Control Lists." www.networkcomputing.com. April 15, 1998.
URL: http://www.networkcomputing.com/907/907ws1.html

[v] Ribak, Jay. "Active FTP vs. Passive FTP, a Definitive Explanation." www.slacksite.com. August 10, 2001.
URL: http://slacksite.com/other/ftp.html