Global Information Assurance Certification Paper

GIAC Enterprises Security:
A Low Cost Solution

By: Greg Leisner

February 15, 2003

Firewalls, Perimeter Protection, and VPNs
GCFW Practical Assignment
Version 1.8

Introduction

This is the Practical Assignment for GIAC's Certified Firewall Analyst (GCFW) certification. While trying to meet all the requirements of the assignment, this paper, like most GIAC assignments that are submitted, has an organizing theme. The theme of this paper is "minimize total cost of ownership". The reason for this theme is the current cost pressure most organizations find themselves under.

The methods used to minimize total cost of ownership are:

1) Use software with no purchase cost.
2) Minimize the tools used to both reduce training costs and increase proficiency.
3) Architect a system that reduces false positives as much as possible thus reducing wasted staff time.

Of course, false negatives are the real danger. But to sustain any policy, it must have management support. And a firewall architecture that is complex and generates many false positives could collapse under cost cutting that reduces available licenses and staffing below design requirements.

Thus, this solution drives down the required person hours to maintain the system. It also minimizes costs to scale up the security system by only using software that has no purchase cost.

With this aggressive cost posture, the system ought to perform as designed, even under extreme funding restrictions combined with a growing workload.

1.0 Security Architecture

1.1 Overview

GIAC Enterprises, Inc (GIAC) is an e-business that engages in the online sales of bulk fortune cookie sayings. The company employs 45 people in production activities and 40 in other, support, activities. The inherent value added to a Business-to-Business distribution system, like GIAC operates, is directly proportional to the difficulty in delivering the product to market. However, sending text files across the Internet is a free activity (at the margin) and the capital investment required is minimal. GIAC is operating in an environment where the barriers to entry are extremely low and competition ought to drive profits to near zero. Yet GIAC dominates the industry and has been, by far, the most profitable firm in the business according to Wall Street industry analysts.

Automation of the business activity is a fundamental requirement to compete. The success of GIAC is based on that, it's supplier and partner relationships, and it's mass customization of the fortune cookie distribution process. The relationships are handled by the firms founding partners and are outside the area of this study. The ordering and production process, however, significantly impact the security architecture.

Rather than offer a standard set of bulk cookie fortunes to the marketplace, GIAC builds each set of delivered cookie fortunes based on the customers' requirements. The ordering parameters are:

1) Number of fortunes required
2) Minimum characters per fortune
3) Maximum characters per fortune
4) Language
5) Topic (humor, love, career, lottery #'s, etc)
6) Gender of reader
7) Ethnic sensitivities to include or exclude
8) Serialized poems/stories
9) Allow/disallow repeats with past orders

A scandal recently hit GIAC and has negatively impacted business. Somehow, interspaced with the fortunes delivered to a customer, were messages of an offensive nature. Because the fortunes are delivered in bulk, the customers never closely read the fortunes, thus allowing the offensive messages to be discovered by consumers; it made the papers and lawsuits have been filed. After investigating the event, it is clear that the offensive messages did not come from GIAC's fortune data bank. However, it is not clear if the offensive messages were placed in the customer's order:

1) by an insider,

2) during warehousing at GIAC,
3) during transit over the Internet, or
4) after delivery to the customer.

This security review is managements' response to the incident. Because of the damage done by the publicity surrounding the incident, management wants to go public with as much of the security steps taken as possible as a confidence building PR measure.

Due to the (hopefully temporary) downturn in business, some support staff have already been let go. There is no money for added security staff and existing personnel must absorb all new tasks. A previously planned shift to a Linux only network has been accelerated. The security personnel let go were the Windows and Mac support staff.

Also, management is accelerating plans for expanding the firm into three new, but related, lines of business:

1) Personalized horoscopes (cookie fortunes for one)
2) A regionalized horoscope for newspapers (cookie fortunes for the masses)
3) An e-Almanac (cookie fortunes on a time line)

These new lines of business will be expected to use the existing infrastructure of the cookie fortune database, distribution facilities and computing resources. There will be more production employees if the new products take-off, but management is determined that support staff will not expand as the promotional expenses are expected to be quite large and margins narrow in the beginning.

From the security architecture point of view, outside of management's enthusiastic backing for making serious changes for security reasons, there is no real empowerment. Financial constraints mean no big pot of money for people, software or hardware. Due to the PR requirements, there can be little or no expectation of 'security through obscurity'. Lastly, the drive into new markets means that the solution has to be highly scaleable. The scalability issue is quite severe since the new businesses, by their nature, will have vastly more orders per revenue dollar than the existing bulk business.

1.2 Rational for the architecture

1) Low up front cost.
   This requires reusing as much hardware, software and software skills as possible. Minimal special purpose hardware and software reduces the purchase, training and replacement costs.
2) Low maintenance cost.
   There will not be many personnel hours available for monitoring the network.
3) Transparency.

The PR effort to overcome the previous security breach will include a media walk through of the changes made. Management wants positive Computer World, Wall Street Journal, etc stories to counter the existing bad stories. This means that any attacker could have as detailed understanding of GIAC's architecture as an insider could.

4) Scalability.
   If the new business takes off, it will swamp the transactional volume of the old business by a wide margin. Thus the new architecture must have lots of spare capacity. On the hardware side, it must be modular and expandable. On the Computer Operations side, the existing personnel (just enough to do the job now) ought to be sufficient for the new endeavors; personnel additions are discouraged, but not impossible if justified.

The fundamental design decision was to leverage security skills. A limited staff cannot be expected to be efficient and effective if required to use many platforms and tools from many vendors. Thus the security staff was going to use a small number of tools and learn to use them well. Since the existing firewall consisted of Linux and NetFilter, and the staff was experienced and comfortable with both, it was decided to make them the foundation of the new security architecture.

Every server and workstation on the internal network will be built from the same set of binaries. Some hosts will have more software, and there will be different configurations due to use and subnet location, but the platform software will be from a unitary and minimalist set. This will decrease the patch management effort, decrease the patch compatibility testing time and ease remote and scripted management. It does increase risk because of the lack of diversity. Management believes that there is net benefit in the trade off selected.

1.3 Universal Software Platform architecture

GIAC has standardized on Redhat Linux 8.0. A single copy of Redhat Linux 8.0 Personal Edition ($39.95) suffices for the entire organization. There are no license issues to manage. The corporate base platform (common to all hosts) is listed in Appendix H. Specific hosts have additional packages added as required by their intended function within the organization. All hardware is x86 compatible.

Redhat 8.0 shipped with kernel 2.4.18-14. The most current update as of the submission of this paper is 2.4.18-24 dated 2-3-2003. The Iptables (NetFilter) version that shipped with 8.0 was 1.2.6a-2 and that binary remains the current version.

1.4 Network Access Communities

Following are GIAC's access groups and their access requirements and

restrictions.

1) Customers
Customers for GIAC's bulk cookie fortunes initially make contact either through GIAC's web site, mass mailings, the outside sales force or the inside sales force. Once the customer decides to do business with GIAC, a customer account is setup in GIAC's accounting system. The information can be filled out by the customer on the web site or by GIAC sales personnel on behalf of the customer. Included with the information are the assignment of an account ID and password and the provision of GIAC's public key. Thereafter, the customer can order on the web site, from GIAC's traveling sales force during a site visit or via phone to GIAC's customer service personnel. In any case, after the order is scheduled, an acknowledgement is emailed (digitally signed) to the customer. When the order is completed, an order availability notice is emailed (digitally signed) to the customer. For many orders, the acknowledgement and availability notice are combined. Then the customer signs on to the ftp site and downloads the file from an order specific URL.

User account information for the Web and FTP servers is retrieved from an LDAP server on the internal network.

A task on the externally visible ftp server monitors the download and sends notification to billing; the same task schedules file deletion for 24 hours later. If files are not downloaded within a prescribed time (as dictated by the customer in their account setup), follow up with the customer is made.

Security Summary:

a) SSL access to Web site to order
b) Email acknowledgements and billing
c) FTP download from GIAC by customer
d) CRON scheduled follow up on FTP server
e) GIAC employees can act as proxy for customers
f) PKI required for email

2) Suppliers
Relations with cookie fortune suppliers are, in many ways, a mirror image of relations with customers. Suppliers contact GIAC through GIAC's web site (a bid/auction system), email, telephone or site visit. Purchasing gives them a vendor account in GIAC's accounting system. Purchasing emails digitally signed contracts to suppliers when they are selected for specific work. The cookie fortunes flow from the supplier to GIAC, requiring: GIAC's personnel to make contact with supplier's Internet sites, the supplier's personnel to connect to GIAC's FTP server or GIAC's personnel to authenticate email from suppliers. If the supplier's Internet site is used, authentication is handled by the method implemented by the supplier. If GIAC'S ftp server is used, authentication is by a supplier account on GIAC's network. If email is used, authentication is by

exchanged public keys.

Security summary:

a) Suppliers may use GIAC's Web bid/auction system
b) GIAC phone's, emails or enters orders on supplier's Web site
c) Supplier phone's or emails acknowledgements
d) GIAC employees receive the product either via email, access the supplier's FTP site, or the supplier uploads the product onto GIAC's FTP site
e) PKI required for email authentication
f)

3) Partners

Partners are trusted vendors. For many of their transactions, they use the same mechanisms as suppliers. However, they are also given user accounts with access to GIAC's internal network and have limited access to GIAC computing resources.

Security summary:

a) Includes all supplier items
b) Require accounts on GIAC network
c) Require VPN access to GIAC internal network
d) Require partner's public key for VPN access

4) GIAC employees inside the firewall

Previously, all GIAC employees were on the same subnet. Access to computing resources was managed through a mix of filesystem permissions and multiple logins on servers.

The major division of employee groups is between production and administration employees. These two groups have many mutually exclusive resources as well as some shared resources. Within each major group, there are many subdivisions with a single employee enrolled in potentially many of those subdivisions.

GIAC employees access the Internet for both business and personnel use. Under no circumstances, however, is pornography, pirated software or hate literature tolerated. All employees have email accounts and can send and receive email internally and externally. All employees can access web sites, web services and download files from the Internet. Only selected employees are allowed to upload files to sites on the Internet, and only for business purposes. Only selected employees (computer operations and production personnel) are allowed access to servers on GIAC's service network. Only selected computer operations personnel are allowed to sign in to the various routers, servers and firewalls via secure shell accounts.

Security summary:

a) A major split exists between production employees and administrative employees
b) All employees have internal and external Web access
c) All employees have internal and external email access
d) All employees can download from external hosts via FTP
e) Selected employees can upload to external hosts via FTP
f) Access to infrastructure hosts is via SSH

5) GIAC employees outside the firewall

This category includes all the mobile workforce as well as employees who work part time from home. The mobile workforce needs to connect to the administrative and product systems to inquire about customers and order status. They also need to pickup email and voice mail messages. The home workers fall into two overlapping groups. First are the employees scheduled to work from home. During their scheduled work hours, they may be continuously connected to the internal network for up to 9 hours or so. They may also have intermittent connections. The second group is those who are traveling or want to 'check in' on some activity or process on off hours. All employees accessing GIAC must do so using company supplied laptops. GIAC only allows external users to connect through the Internet gateway (no modem access).

Security summary:

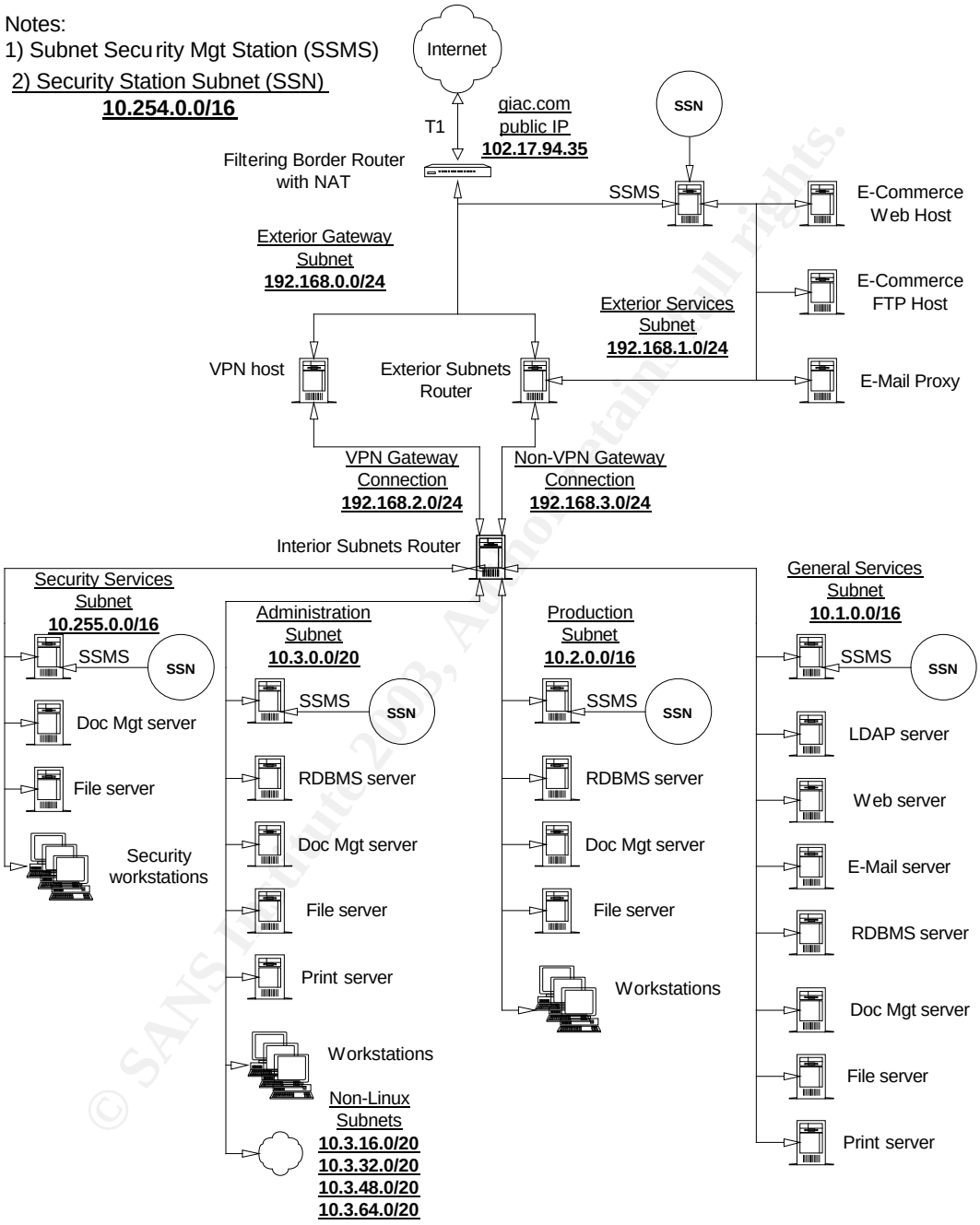
a) VPN access to GIAC internal network through company laptops
b) Time of day of access isn't firm
c) Activities allowed are the same as if the employee was locally attached

6) Other hosts on the internal network

a) Printers and copiers are all directly connected to the network. Maintenance of these devices is outsourced. Staff and management believe the risks are low enough to leave these devices unmonitored.
b) A few fax machines exist in house, but none are connected to either a host or the network. No action will be taken for these units.
c) Vendor supplied hosts (i.e. FedEx, Pitney Bowles, etc) are all standalone and, like the fax machines, no action will be taken.
d) Legacy hosts. There are a few applications that run on Mac's (Marketing), Windows (Accounting), OS2 (maintenance) and even DOS (payroll). These hosts will be fire walled by a Linux box either individually or, where possible, on a subnet of like machines. (The current users are urged to find a Linux based substitute.) The Linux firewall for these hosts will only allow through the minimal set of services (ping, dhcp, DNS, ntp, http, ftp and email), and in no case will these hosts be allowed access outside of the interior network (this excludes the exterior services network as well as the Internet). Users requiring Internet or exterior service network access will have two hosts, the second one being a Linux host. (It is hoped that

this discomfort will accelerate the move to a Linux solution.)

1.5 Network Graphic



1.6 Filtering Border Router w/NAT

The Filtering Router that straddles GIAC's Internet connection is a generic Linux host. In the future, should a generic host prove unable to scale to the required volume of new business, a special purpose routing host, running Linux 2.4, can be substituted. (The NetFilter script is not expected to require any change.) The purpose of the router is:

1) leverage GIAC's single public IP address for use by multiple hosts on the internal private address network by running Network Address Translation (NAT).
2) protect the private network from:
    a) denial Of Service (DOS) attacks.
    b) simple probes.
3) prevent address spoofing from either side.
4) segment the Internet between trusted and untrusted domains.

Because the Border Router runs Network Address Translation (NAT), it functions, by default, as the primary firewall between GIAC and the Internet. The NetFilter software will:

1) reassemble fragments before routing them.
2) Keep track of state for all connections (required for NAT)
3) Deliver all inbound non-NAT'd packets (to GIAC from the Internet) to the Border Router's own TCP/IP stack where they must be filtered.

This covers the bulk of the work of a firewall located on the perimeter. It is common, for performance reasons, to separate the static filtering done by a Border Router from the stateful inspection performed by the primary firewall. However, since the Border Router's NetFilter is NATing packets, it must be stateful. The security staff feels that combining the firewall and router in one host makes sense because of the nearly complete overlap in functionality.

The added filtering needed on the router to provide all the functionality of a firewall is relatively minor. The staff feels that the clarity of having a single host perform all the filtering outweighs any minor performance concerns. Adding more hardware capability to the generic hardware platform can enhance performance or, if required, request funds for a special purpose router running Linux 2.4.

Linux based routers tend to be much less expensive to purchase than the industry standard Cisco products. For example, a comprehensive router line is available from ImageStream:

(http://www.imagestream-is.com/index_1280.html)

The NetFilter scripts are portable to these special purpose Linux routers, thus providing a seamless and relatively inexpensive upgrade path if one is required.

1.7 Exterior Subnets Router

The Exterior Firewall Subnets Router protects the Exterior Services Subnet from the Untrusted Internet, and it also protects the Interior subnets from both the Untrusted Internet and the Exterior Services Subnet. The Exterior Subnets Router Firewall performs the following activities:

1) logs all TCP and most UDP connections.
2) blocks unauthorized traffic.
3) prevents address spoofing.
4) reassembles fragmented packets.

1.8 VPN

The decision to reduce costs by only holding one public IP address and to deploy one standardized platform causes the following problems for the architecture of VPN's.

1) A Network-to-Network VPN requires the connected organizations to coordinate their use of private IP addresses. If it were just two parties that had to negotiate, that hurdle would seem to be low. However, because of the "transitivity of Trust", it actually includes the IP address space of all the networks that each party is attached (or becomes attached) to plus all the networks that those networks become attached, etc. The coordination effort could spiral towards GIAC essentially setting up it's own IANA for it's VPN network.
2) The lack of platform diversity at GIAC means that not only will the effort of GIAC technical personnel scale well, so will the effort of any attacker. GIAC tightly controls the software available to users on its network in order to minimize the toolset available to an attacker on the inside. By placing many uncontrolled platforms and users "on the inside" (via a VPN), GIAC greatly increases the risk of a compromise.

The decision by GIAC was to employ the simplest and safest solution. The router will tunnel port 22 (SSH) to a host connected to the interior firewall. Users will login to the VPN host for a secure session (the user's host and GIAC's VPN concentrator will mutually authenticate themselves as well). This setup preserves the single IP address and uses NAT. By requiring all user work to be performed on the VPN concentrator, no "hole" is opened in the Exterior Firewall. There is no trust relationship with a host outside of GIAC's network (external hosts are acting as nothing more than dumb terminals). Attacker toolkits are avoided because the external users are restricted to a platform (the VPN concentrator) that is under GIAC control.

1.9 Subnet Security Management Stations (SSMS) and the Security Station Subnet (SSN)

Each subnet has attached a host (SSMS) that performs the following tasks:

1) Dynamic Host Configuration Protocol (DHCP) server (if used on that subnet)
2) Domain Name Service (DNS) server
3) Network Time Protocol (NTP) server
4) remote syslog server
5) snort IDS agent (manually activated)

The SSMS hosts are tied together with a network that is private to them. This network's purpose is to:

1) insure Quality of Service (QOS) for remote syslog traffic by preventing it from crossing any router, thus reducing both total network load and the possibility of loss of the UDP traffic.
2) insure that a compromised router cannot subvert remote logging.
3) allow access to all SSMS hosts from the Security Subnet without allowing traffic analysis of the communication from any compromised non-SSMS host.
4) isolate DNS and NTP communication between SSMS hosts from attack by a compromised non-SSMS host.

1.10 Network addressing

GIAC has one fixed public IP address (GIAC.com, 102.17.94.135). assigned to the external interface of the Border Router. All other network taps use either the 192.168.0.0/16 or the 10.0.0.0/24 private address spaces.

The 192.168.0.0 addresses cover all network connections between the Internet and the Interior Subnets Router. This network is both small and static in its hardware composition. Thus, these addresses are static and assigned in each host's configuration scripts. The ARP mappings are also entered in scripts and are marked as permanent. Thus no spoofing of hosts ought to occur (eliminating man in the middle attacks), and analysis of syslogs (both interactive and scripted) can be tailored to the specific host.

The 10.0.0.0 network addresses cover everything inside the Interior Subnets Router. The interior network is divided into 5 subnets:

1) General services 10.1.0.0/16 (gs.giac.com)
2) Production 10.2.0.0/16 (prod.giac.com)
3) Administration 10.3.0.0/20 (admin.giac.com)

Administration itself is sub netted due to the presence of legacy Windows and Mac hosts. A dual hosted Linux box acting as a firewall controls each subnet. The function of the Linux box is to provide NetFilter syslog, DHCP, DNS, and NTP services for the subnet:

   a) General Administration 10.3.0.0/20 (ga.admin.giac.com)
   b) Accounting (10 Windows hosts) 10.3.16.0/20 (acc.admin.giac.com)
   c) Marketing (5 Mac hosts) 10.3.32.0/20 (mrk.admin.giac.com)
   d) Maintenance (1 OS2 host) 10.3.48.0/20 (main.admin.giac.com)
   e) President's office (3 Windows hosts) 10.3.64.0/20 (boss.admin.giac.com)

4) Security 10.255.0.0/16 (sec.giac.com)
5) Security Station Subnet 10.254.0.0/16 (ssn.giac.com)

1.11 Security Domains

General information regarding user accounts and security domains:

1) All user accounts at GIAC are maintained on an LDAP server on the interior network.
2) The following conceptual security domains exist:
   a) The Internet-untrusted (IU)
   b) The Internet-trusted (IT)
   c) GAIC external services subnet (ExSub)
   d) GAIC internal subnet (InSub)
   e) SSN
3) The following protocols services are allowed between security domains
   a) IU ← → IT: no direct connection
   b) IU ← → ExSub: SSH, HTTP, FTP, SMTP
   c) IU → InSub: established on outbound connection
   d) IT ← → ExSub: SSH
   e) IT ← → InSub: no direct connection
   f) ExSub ← → InSub: SSH, HTTP, FTP, SMTP
   g) InSub → IU:
      1) SMTP- proxy via ExSub
      2) SSH, HTTP, FTP, DNS, NTP
      3) All others- not allowed
   h) SSN ← → all others: proxy through an SSMS via SSH logon

1.12 Interior Routers

All traffic to GIAC's interior subnets goes through its Interior Subnets Router (ISR). The purpose of the ISR is to:

1) isolate security, production and administrative data.
2) record inter-subnet traffic to validate the partioning logic of the network.
3) archive administrative isolation of untrusted Internet traffic from trusted

Internet traffic by routing the VPN gateway past the exterior firewall.
4) prevent and report address spoofing

All Linux hosts (servers and workstations) will run NetFilter. Besides protecting the individual host, probes and illicit traffic will be logged.

1.13 Non-VPN Remote Access

There is no remote access other than through the VPN gateway.

1.14 Intrusion Detection Systems

As stated in 1.10 above, NetFilter protects all hosts. The logs are recorded locally and sent to the local subnet's SSMS where analysis takes place.

All Linux hosts have /home directory trees on separate disk partitions. These home partitions will be mounted with the options nodev, noexec, nosuid, and nouser. While not a complete solution (noexec can be worked around), it helps a lot. All programs, libraries, scripts, document macros, and other executable content will be outside of /home and users will not have write permission to those directories. (For a user to get executable content onto those drives, Computer Operations must vet the executable and then add it to the master filesystem template and installation routines.) Where write permission is granted, the file will not be marked as executable.

The point of isolating executables from user maintainable filesystems is to assist in making Tripwire effective by eliminating false positives. Tripwire will monitor the (largely static) filesystems where executables reside. The filesystems that users can write to will not be allowed to have executable content and thus will not be monitored by Tripwire.

There are other requirements for securing Linux hosts (lots of them), but only their effects concern us here. The other significant item is the limitation of unfettered root access to Computer Operations staff only; no end user will have root access except through sudo and those cases that require suid and are in the master filesystem template. This helps insure that the filesystem permissions, and their Tripwire monitoring system remain unadulterated. Thus, for network security purposes, the expected network traffic is very tightly constrained by design. Combining the presence of NetFilter and Tripwire on every host with the limited allowed traffic, the normal IDS function is implemented through analysis of NetFilter and Tripwire entries on the Remote Log Server. This poor man's IDS has the advantage of both securing the host and monitoring traffic between hosts, even when both hosts are internal to the local network.

2.0 Security Policy and Tutorial

2.1 Overview

Since the rulesets are executable scripts, they are in appendices for readability, rather than being included here in the commentary. Each script contains comments concerning its organization, references and rule rational. The tutorial for implementing the Border Router policy is contained below.

2.2 Rulesets

All the scripts have default policies of DROP for the INPUT, OUTPUT and FORWARD chains. Each chain (built-in and custom) is separately defined in it's own block in the scripts. Custom user chains only LOG packets and take negative action (DROP, REJECT). Thus all the built-in chains explicitly contain all ACCEPT targets within a short span of each script. This increases the ease of auditing for the critical ACCEPT rules.

The Rulesets are all organize as follows:

1) Declare macros
2) Initialize
3) Create user chains
4) Mangle table
5) NAT table
6) Filter table-INPUT chain
7) Filter table-OUTPUT chain
8) Filter table-FORWARD chain

This organization is universal, even if a particular host doesn't implement a particular section. For example, the VPN has no mangle, NAT or FORWARD chain entries, yet the sections are there for consistency.

Each rule has a reference in its comment, if the rule is derived or copied from some source document. Generally, the script macros and initialization closely follow the example scripts in Oskar Andreasson's "Iptables Tutorial 1.1.11" found at:

http://www.netfilter.org/documentation/tutorials/blueflux/

The actual ruleset is generally this author's. If specific sources were used in creating individual rules, they are referenced at the point of use:

1) snort rules: ("; sid:625") (http://www.snort.org/dl/rules/)
2) rfc's: ("; rfc:3330") (http://www.rfc-editor.org/)
3) SANS online self study course: ("SANS course materials-

fw_26_netfilter_gaunt.pdf, page 210")
(http://www.sans.org/onlinetraining/track2.php)

The security policy (NetFilter script) is provided for:

1) The NetFilter script template (Appendix A)
2) The Border Router (Appendix B)
3) The Exterior Subnets Router (Appendix C)
4) The VPN (Appendix D)
5) The Exterior Gateway's SSMS (Appendix E)
6) A typical user workstation (Appendix F)

Notes on the uses of the targets LOG, DROP and REJECT:

1) LOG: This target is used as follows:
   a) --log-level crit is used only on infrastructure hosts (routers, VPN, SSMS, servers; not on user machines) when a situation may signal either an insider attack on that host, or a compromise of that host.
   b) Except for some obvious scanning and attacks from within its ISP, GIAC doesn't bother logging DROP'd or REJECT'd external traffic. The reason is that there simply isn't the time to do anything about them, and they both take resources to LOG and clutter the other, meaningful entries in the syslog files.
   c) Internal host's NEW connections are always logged except for traffic to SSMS hosts. The reasons for not doing so are that a udp syslog packet would trigger a new syslog entry, which could cause a new udp syslog packet, … in a never-ending loop. Because of timers in the state module, this shouldn't actually happen, but there would still be added volume and redundancy in the LOGing that GIAC wants to avoid. This opens up port 514 on SSMS host to dos attacks. Because of this hole, trying to prevent dos attacks via DNS (port 53), NTP (port 123) and DHCP (port 67) on SSMS hosts is pointless. Thus the LOGing can also be turned off on those ports. (The SSN network allows access to the SSMS hosts even when they are under dos attack, so the security staff can still investigate the situation and capture information for remediation of the situation.)
   d) --log-prefix strings have the following patterns:
      1. "<host> invalid <builtin chain>"
      2. "<host> NEW <builtin chain>"
      3. "<host> excessive pings"
      4. "<router> spoofing"
      5. "SSN NEW <builtin chain>"
      6. "tcp-chain: <scan/attack>"
      7. "ISP: <scan/attack>"
   e) --log-tcp-sequence is only LOG'd for external NEW connections. GIAC's tcp sequences are known to be random, but if outsider initiated

connections have anomalies, this will capture those sequences.

2) DROP is used for scans/attacks and trash from the Internet; REJECT (with --reject-with tcp-reset for tcp) is used for indent connections from the Internet and all internal problem connections. This maximizes internal host performance.

Lastly, the 192.168 networks are small and have few address changes. Thus to avoid arp poisoning and host spoofing on these static networks, the arp cache and hostnames/IP's are fixed. Complete listings are in Appendix G.

2.3 Security Tutorial for the Border Router

2.3.1 Platform preparation

If this is an existing production machine, skip steps 1 and 2 immediately below. Otherwise, if the Border Router is a vendor-supplied platform that runs Linux (2.4 required), follow that vendors recommendations for securing the platform; then skip step 1 immediately below.

To secure a "generic host" (i.e. general purpose computer), the following steps are required:

1) Install GIAC's base platform. (Appendix H)
2) Physically secure the host and secure the boot process. (Appendix I)
3) Install/uninstall all additional packages as appropriate. Only packages required to boot, manage the host from the SSMS and perform the required functions ought to be installed.
4) Log the host/packages into the Security database.

2.3.2 Create the initial NetFilter script

1) If an existing script can be used, just export it from the CVS archive. Don't reinvent the wheel; it's expensive.
2) If an existing script can be easily modified, export it from the CVS archive. Do not check it out! To avoid unintended and potentially dangerous collateral effects on other scripts, each script must stand on it's own.
3) If many modifications are required, or no similar script exists, start from scratch. The script's logical integrity and ease of audit far outweigh minor savings in script generation. Export the NetFilter template script.
4) Only if this is maintenance to an existing script, do a checkout from the CVS archive.

2.3.3 Script rules construction

1) Do not modify, reorder or delete any of the script's template lines. The template lines all begin with 6 or more '#' symbols. Also, all your

comments must use only 2 or 4 '#'s to distinguish them from the template lines.
2) The template is supposed to enforce a chain-sequenced organization. Do not place rules for a chain outside of that chain's section.
3) Organize the rules in some reasonable fashion. If possible, use the same organizational logic of existing scripts.
4) Place references for a rule in a comment immediately preceding the rule.
5) If reasonable, place high volume rules before low volume rules. However, if the logic for a chain becomes too messy, simplify by sacrificing performance.
6) Do not place extraneous matching logic in rules. Each match in a rule MUST be required for the rule to work correctly. Placing extraneous matches in a rule misleads a reviewer about the structure and logic of a chain. For example, assume only tcp, udp and icmp are valid in this chain, and all the tcp and icmp rules come first and DROP rules for the balance of those protocols have been declared. Then recognize that with a comment and a "–p ! udp –j DROP" rule thus both DROPing all other protocols and excluding any further "-p udp" matches. This helps a reviewer follow the chain's logic by systematically eliminating partitions of the universe of packets.

2.3.4 Script testing

Scripts need to be tested. The tester cannot be the script's author. Each series of tests is crafted for the particular script, but includes some common steps.

1) Review the script for completeness of coverage, clarity, correctness and performance.
2) Run the series of automated scanning tools currently specified by Company policy. (GIAC's current automated testing tools are listed in Appendix J.)
3) Verify that the traffic expected to be ACCEPT'd is actually passed through.
4) The other test partition, traffic that ought not to be ACCEPT'd is not, cannot be definitively verified via test packets (it's complexity is usually too great.) However, if any particular situation appears questionable, it ought to be exercised with test packets.

2.3.5 Script acceptance

At this point, the script appears to be good. But the last stage is use in production and final archiving.

1) For the first week in production, the local network SSMS must run packet capture on the host running the new or modified script. And both the script's author and the script's tester must review those dumps independently for problems. If any problems occur, go back to 2.3.2 or

2.3.3; otherwise execute step 2 or 3 as appropriate.

2) If this is a new script:
   a) Import the script to the CVS archive.
   b) Log the host/script into the Security database.
3) If this is an existing script:
   a) Checkin the script to the CVS archive.
   b) If this is a new host, log the host/script into the security database.
   c) Export the script from the Security database and update all hosts registered in the security database as using this script.

3.0 Verify the Firewall Policy

3.1 Overview

The ruleset for the Border Router (Appendix B) can be tested like any program. The ideal method for testing a program's functionality is to create a testset that exercises all logic paths in the code. In the case of the Border Router's NetFilter script, that testset could be done created simply by following the logic tree of the tables and chains. Then a predicted result for each rule (LOG's, REJECT packets, evidence of successful connections, tcpdump of forwarded packets, etc) would be checked against the syslog entries and tcpdump listings. Nessus (http://www.nessus.org/) is the ideal tool for this selective packet passing. The use of Nessus would also allow throughput tests to be run on the Border Router to examine performance impact of various ruleset choices and orderings.

But, as is usual here at GIAC, the ideal is taking a backseat to the doable. The actual test will involve nmap and ping. The tcpdump of incoming, forwarded, and returned packets along with the resulting log entries from NetFilter will be examined for correct behavior, along with the output of nmap and ping themselves. This is actually a necessary test to perform to insure that there are not missing, malformed or mal-ordered rules.

3.2 The Plan

Because GIAC is using a general-purpose computer running GIAC's self-defined platform as a Border Router, the correctness (but not the performance) of the NetFilter script can be fully and completely examined on a test bed network. The test bed host will have the CVS archive of post install configuration changes from the production Border Router applied. There will be no production hosts required, and no impact on daily operations.

The test bed consists of two hosts, the Border Router (BR) host and the nmap & ping (testing) host. The BR host is outfitted with two NIC's, but the testing host has just one, aliasing that single NIC as all the required internal non-BR hosts. The two machines will be connected with a hub.

The BR host, besides it's syslog entries from NetFilter, will run tcpdump on an interface. The testing host will just keep its nmap and ping output.

After starting the NetFilter script on the BR host, the following was will be run:

```
tcpdump –n –i eth1 –w /root/tcpdump_eth1.txt
```

The initial tests to be run from testing host address 102.17.94.36 are:

1) nmap -e eth0:6 -S 102.17.94.36 -sS -P0 -F -v -n -oN nmap_extest.txt

102.17.94.35
2) ping –c 30 102.17.94.35nmap –sS –P0 –oN /root/nmap_1.txt 102.17.94.35
3) nmap -e eth0:6 -S 102.17.94.36 -sF -P0 –p22 -v -n -oN nmap_extest_sA.txt 102.17.94.35
4) nmap -e eth0:6 -S 102.17.94.36 -sF -P0 –p55555 -v -n -oN nmap_extest_sA.txt 102.17.94.35
5) nmap -e eth0:6 -S 102.17.94.36 -sX -P0 –p22 -v -n -oN nmap_extest_sA.txt 102.17.94.35
6) nmap -e eth0:6 -S 102.17.94.36 -sX -P0 –p55555 -v -n -oN nmap_extest_sA.txt 102.17.94.35nmap –sT –P0 –F –oN /root/nmap_2.txt 102.17.94.35
7) nmap -e eth0:6 -S 102.17.94.36 -sN -P0 -p55555 -v -n -oN nmap_extest_sF.txt 102.17.94.35
8) nmap -e eth0:6 -S 102.17.94.36 -sN -P0 –p22 -v -n -oN nmap_extest_sA.txt 102.17.94.35
9) nmap -e eth0:6 -S 102.17.94.36 -sN -P0 –p55555 -v -n -oN nmap_extest_sA.txt 102.17.94.35
10) nmap -e eth0:6 -S 102.17.94.36 -sA -P0 –p22 -v -n -oN nmap_extest_sA.txt 102.17.94.35
11) nmap -e eth0:6 -S 102.17.94.36 -sA -P0 –p55555 -v -n -oN nmap_extest_sA.txt 102.17.94.35nmap –sX –P0 –F –oN /root/nmap_3.txt 102.17.94.35
12) ping -c 30 -I eth0:5 192.168.0.2
13) http to 102.17.94.35

Then packets originating from the interior (IP's 192.168.0.3 and 10.3.0.5):

14) ping –c 30 102.17.94.36
15) http to 102.17.94.36

The results of the previous tests will determine whatever follow-on action to take. The initial series of tests is expected to take 3 hours to run and an additional 2 hours to review the results. After the first complete analysis (including any NetFilter corrections required), future test results can be compared to the original results via analysis scripts that strip out timestamps and compare the remainder of the text.

3.3 The Tests

A limitation of the test bed was highlighted while constructing the tests. (It actually should have been obvious from the start.) By having the testing machine alias one NIC for both sides of the firewall, the TCP/IP stack behavior ought to prevent the packets from hitting the wire by looping the packets internally. For this reason, no tests from the interior network to the exterior network were attempted. (Because of NAT, connections from the exterior are OK because

they are always pointed at the NIC on the BR host.) While initial connection packets should clear the firewall if they were directed to an address not assigned to the testing host's NIC, there would be no return traffic to validate the connection.

The same problem exists in reverse. However, while deferring the validation of NAT connection handling (the http connection), the scanning by nmap and pings of same network hosts can continue.

Thus the tests can only involve the following:

1) scans of the firewall from the exterior
2) initial syn packets from the exterior to DMZ hosts
3) pings of the firewall (DNAT'd and unDNAT'd ports)
4) interior packets directed to the firewall's interior interface

In terms of the proposed tests from 3.2 above, only #'s 13, 14 and 15 are being dropped in this past iteration of the test.

In retrospect, the testing host needs two NICs and instead of a single common hub, either two null modem cables or two hubs are required.

3.4 The Setup

To setup a new environment, new or changed entries can be 1) entered in configuration files, entered via commandline tools or entered via GUI tools. The results are all the same, the choice made is up to personnel taste. (As part of the test, all three methods were used in this test. The /etc/hosts file was edited to added the host names needed for the tests; the arp command was used to create permanent arp entries; and the KDE NIC configuration tool was used to set IP address.) The results of this work are seen in the output of the arp command listing the arp cache. First on the BR hosts:

```
Address                  HWtype  HWaddress           Flags Mask   Iface
delta.egs.giac.com       ether   00:40:63:C1:9D:D3   CM           eth1
theta.ess.giac.com       ether   00:40:63:C1:9D:D3   CM           eth1
eta.ess.giac.com         ether   00:40:63:C1:9D:D3   CM           eth1
gamma.egs.giac.com       ether   00:40:63:C1:9D:D3   CM           eth1
zeta.ess.giac.com        ether   00:40:63:C1:9D:D3   CM           eth1
alpha.egs.giac.com       ether   00:40:63:C1:9D:D3   CM           eth1
somebody.nowhere.com     ether   00:40:63:C1:9D:D3   CM           eth0
Entries: 7      Skipped: 0      Found: 7
```

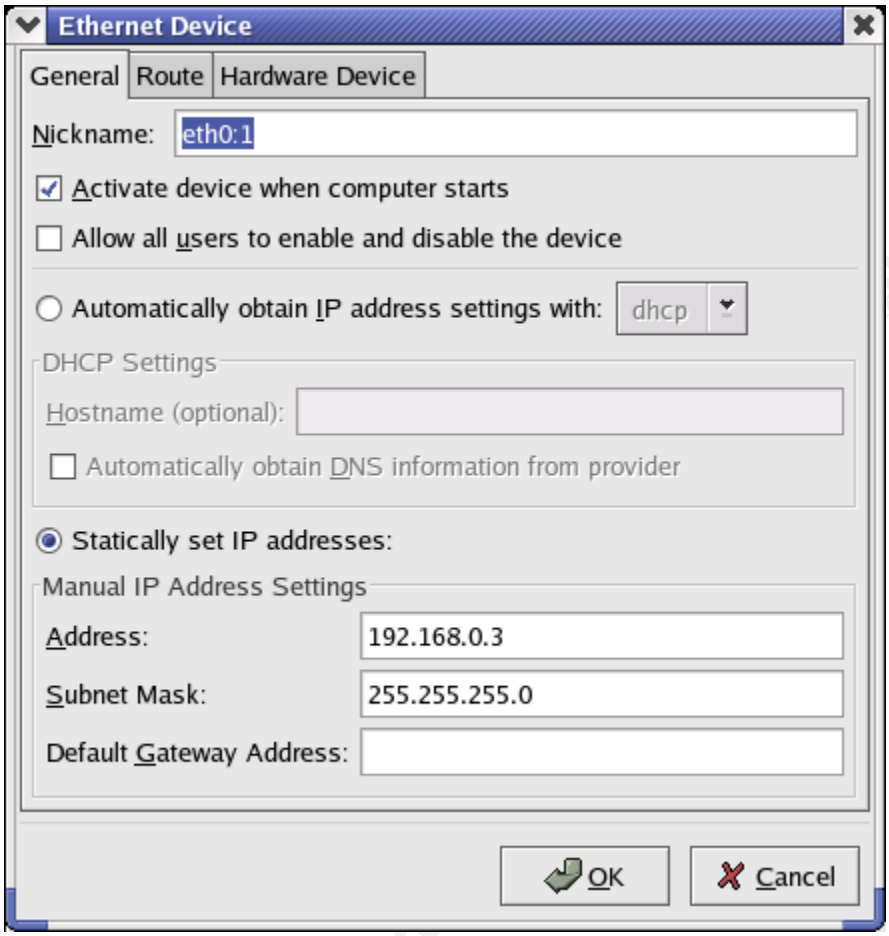
and also on the testing host:

```
Address                  HWtype  HWaddress           Flags Mask   Iface
```

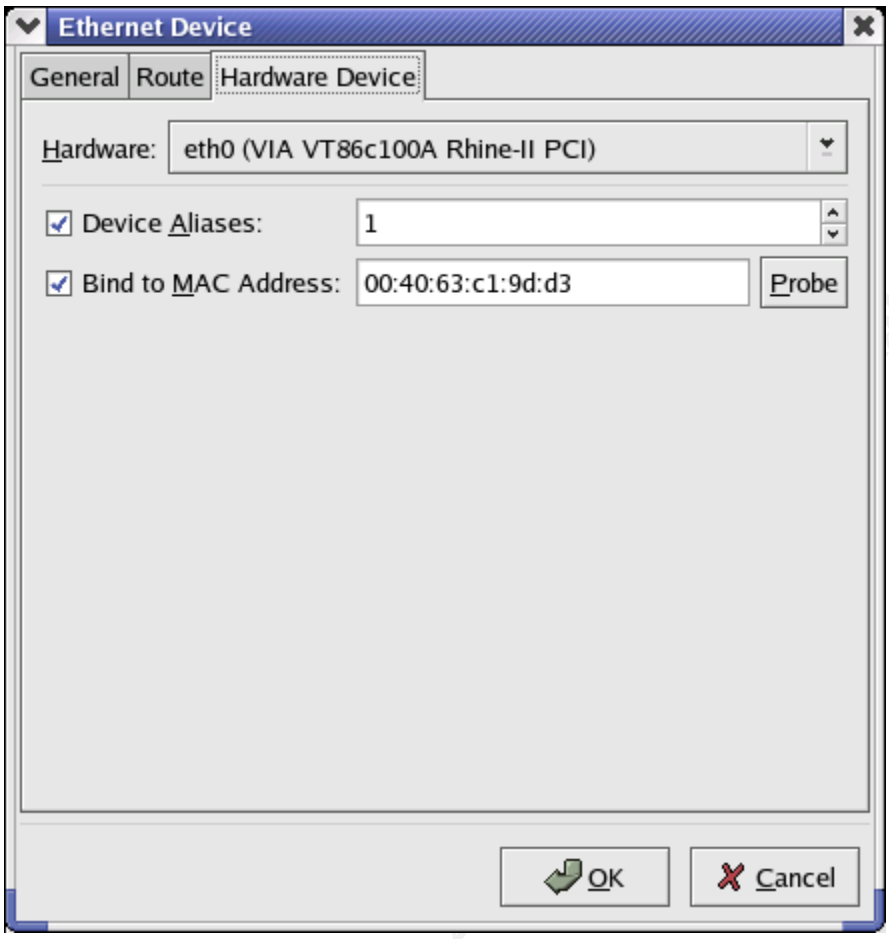
```
beta                ether   00:03:6D:13:37:FD   CM          eth0
GIACcom             ether   00:03:6D:14:29:ED   CM          eth0
Entries: 2      Skipped: 0     Found: 2
```

However, the GUI NIC maintenance tool in the KDE environment provides an easy way to create and manage two separate IP profiles for hosts. The 'Common' profile is the one used by the hosts in their normal functioning and a new profile 'Fwtest' was created for this exercise. The most interesting work was done on the many-aliased testing host.

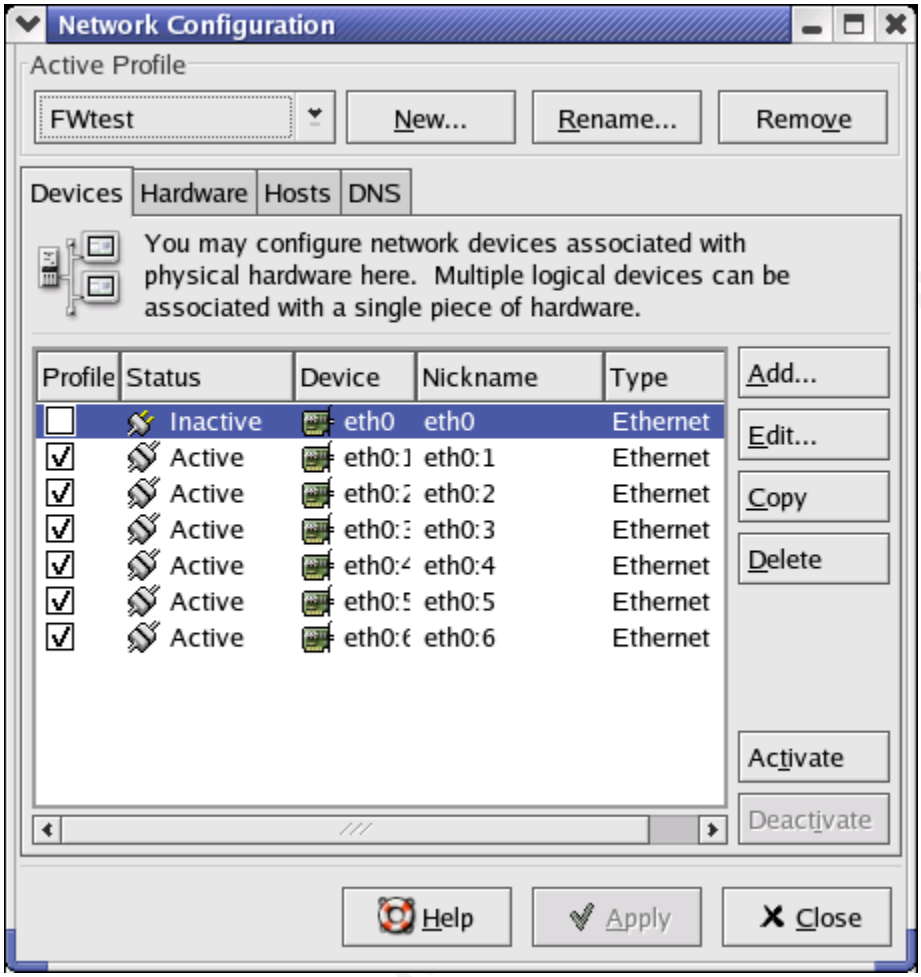
To add an alias, two simple screens need to be filled out. First, the name and IP address are specified:



then the alias is attached to a NIC interface:



After all these aliases were setup for the testing host, the KDE tool displayed:



The same information is displayed in the familiar ifconfig output:

```
eth0     Link encap:Ethernet  HWaddr 00:40:63:C1:9D:D3
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:286 errors:0 dropped:0 overruns:0 frame:0
         TX packets:67 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0
         RX bytes:18792 (18.3 Kb)  TX bytes:4314 (4.2 Kb)

eth0:1   Link encap:Ethernet  HWaddr 00:40:63:C1:9D:D3
         inet addr:192.168.0.3  Bcast:192.168.0.255  Mask:255.255.255.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:2   Link encap:Ethernet  HWaddr 00:40:63:C1:9D:D3
         inet addr:192.168.1.2  Bcast:192.168.1.255  Mask:255.255.255.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

```
eth0:3   Link encap:Ethernet  HWaddr 00:40:63:C1:9D:D3
         inet addr:192.168.1.3  Bcast:192.168.1.255  Mask:255.255.255.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:4   Link encap:Ethernet  HWaddr 00:40:63:C1:9D:D3
         inet addr:192.168.1.4  Bcast:192.168.1.255  Mask:255.255.255.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:5   Link encap:Ethernet  HWaddr 00:40:63:C1:9D:D3
         inet addr:10.3.0.5  Bcast:10.3.15.255  Mask:255.255.240.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:6   Link encap:Ethernet  HWaddr 00:40:63:C1:9D:D3
         inet addr:102.17.94.36  Bcast:102.17.255.255  Mask:255.255.0.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

lo       Link encap:Local Loopback
         inet addr:127.0.0.1  Mask:255.0.0.0
         UP LOOPBACK RUNNING  MTU:16436  Metric:1
         RX packets:72 errors:0 dropped:0 overruns:0 frame:0
         TX packets:72 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0
         RX bytes:4682 (4.5 Kb)  TX bytes:4682 (4.5 Kb)
```

3.5 The Audit Results

The tcpdump capture file contained 4,840 lines, almost all of which were of the nature of:

```
15:36:57.869323 somebody.nowhere.com.63222 > 102.17.94.35.2241: S [tcp
sum ok] 2513326235:2513326235(0) win 4096 (ttl 51, id 19967, len 40)
```

The above record, and thousands of others like it, is not matched in the NetFilter log file since the NetFilter script was not started with the –test option. (That option would have had NetFilter LOG all otherwise silently DROP'd packets.) Later selective testing may require such logging, but the simple scanning and pinging here doesn't. The tcpdump capture file has had all those lines expunged for readability.

Comments were added to the tcpdump capture file to label the entries by the test they cover. Also, the nmap and ping output captured on the test host in included in the comments. Only the NetFilter output is missing. It was kept together on it's own for integrity purposes.

Appendix K contains the annotated tcpdump capture file and Appendix L has the complete and unadulterated NetFilter log.

While the detail is in the above two Appendixes, in summary the following can be stated.

1) All nmap scans verified that the NetFilter script performed correctly. Only the ports covered by DNAT rules show as passing through the NetFilter chains.
2) The ping tests showed the NetFilter rule on the ping LOG rules were correct. However, the ACCEPT rules had the same limit match and thus performed opposite of what they ought to have. They ignored the packets they should have answered and answered the packets they should have ignored. As noted in the commentary, simply dropping the '!' in the ACCEPT rules will fix the problem.
3) Besides the expected log records, the NetFilter log shows some entries that were DROP'd in the firewall's OUTPUT chain. These packets were not part of the test, but were the result of the modifications of the router's networking parameters for the test. The modifications incorrectly left intact the pre-existing DNS server entries.

Lastly, regrettably, the continuation of testing has been indefinitely postponed. As the analyst was examining the results of the first round of tests, other personnel tore down the testbed platform in order to return the units to their previous uses. (OK, my wife demanded access to the Internet and her email and the tests were stopped.) At a resource strapped enterprise such as GIAC, dedicated testing equipment is always in short supply as is the time to do the tests. Because of the problems in the testing layout (see 3.3 above) as well as the error uncovered in the ping tests, this firewall script is being temporarily withheld from production.

4.0 Design Under Fire

4.1 Overview

The Practicum selected for this part is by Robert Alley, and it can be found at:

http://www.giac.org/practical/GCFW/Robert_Alley_GCFW.pdf

[The network diagram contained in that paper is printed on the next page.]

Two months ago, Hortis Hacker was eating out at a Chinese restaurant. The dessert included with his Chicken Lo Mein Loo was a fortune cookie. Hortis's cookie had this fortune:

"You will be very lucky in the next lotto drawing."

Thrilled, Hortis took all his savings, borrowed from family and friends, and maxed out his credit cards (all 12) by taking cash advances. He then used all this cash to purchase PowerBall tickets as the next drawing was for a prized of over $300 million. Surely this was what the cookie fortune referred to!

Needless to say, somebody else won. Hortis was ruined! And Hortis vowed revenged. After some research, Hortis discovered that the fortune came from a company named GIAC.

Hortis prepared for his attack. He searched for all the information he could find out about GIAC and scanned their network. He discovered that an ISA Server protected GIAC. He discovered much more about GIAC and was able to draw the network diagram found on the following page. Then Hortis began to plan his attack…

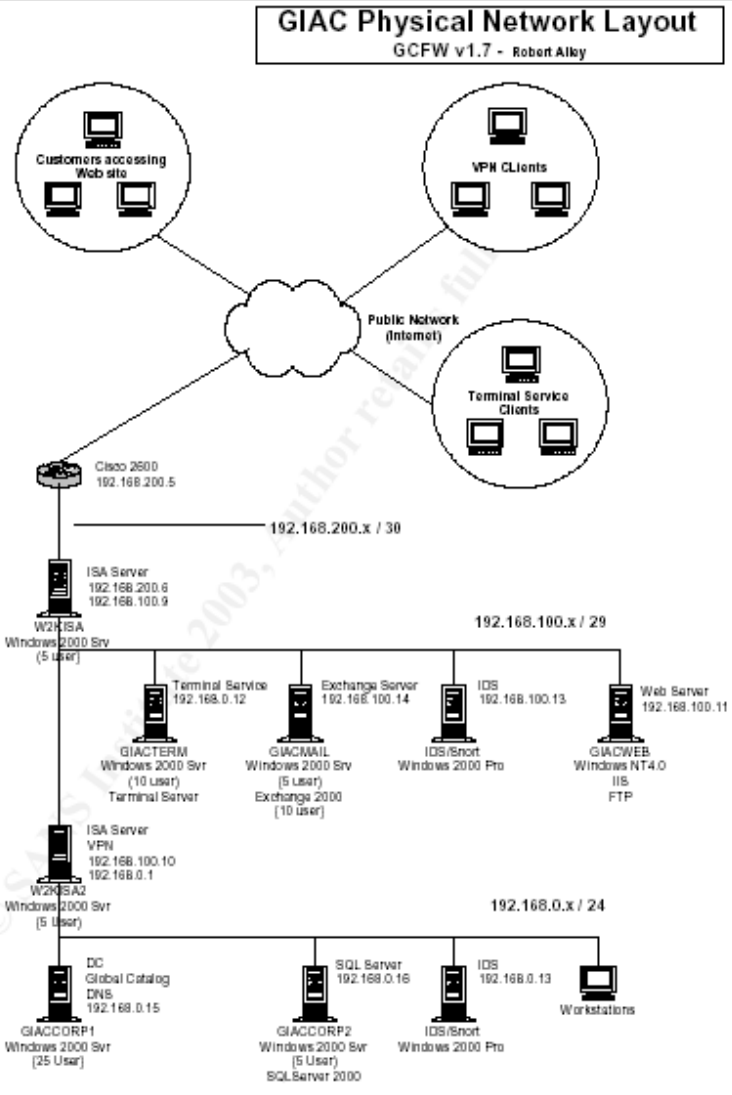
GCFW Practical Assignment
A Recipe for Good Fortune

Version 1.7	GIAC Enterprises

Assignment 1:

Security Architecture

(12)



4.2 Attacking the firewall

4.2.1 Research

When searching for vulnerabilities for an ISA Standard Edition firewalls, the attacker knows both what firewall is employed and the underlying Operating System. From the system requirements for the ISA Server page at:

> http://www.microsoft.com/isaserver/evaluation/sysreqs/default.asp

Hortis read the following:

> "Operating System: Microsoft Windows® 2000 Server or Windows 2000 Advanced Server with Service Pack 1 or later*, or Windows 2000 Datacenter Server operating system."

This provides a number of avenues of attack. They break down roughly into the following categories:

1) ISA Server vulnerabilities
2) W2k Server vulnerabilities
3) Vulnerabilities arising from misapplied patches
4) Mis-configuration vulnerabilities in the ISA Server
5) Mis-configuration vulnerabilities in the W2k Server

If an administrator runs a network of only one kind of host OS, that administrator has the ability to scale their skill set for the particular platform. While Windows OS vulnerabilities outnumber ISA Server vulnerabilities, thus offering many more opportunities, the relative rarity of the ISA server on this administrator's network could mean that the ISA Server software might not be as competently maintained as the underlying OS.

Because any unsuccessful attack on a network can be expected to alert it's administrator to the attack, just as in hunting, your first shot is your best shot. Therefore, the sequence of assaults will be ISA Server vulnerabilities first and W2k Server vulnerabilities second.

The first step Hortis took was to check for known vulnerabilities at the vendor's site (www.microsoft.com) and in the SANS (www.sans.org) vulnerability listings. The latest vulnerability related to the Gopher protocol; Microsoft's advisory was MS02-027 (http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-027.asp). This gave a brief description, and included a Common Vulnerabilities and Exposures key of CAN-2002-037. This lead to the CAN document (http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0371) which contained a URL that eventually led to a somewhat detailed

explanation of how to trigger the vulnerability: (http://www.solutions.fi/index.cgi/news_2002_06_12?lang=eng).

4.2.2 Designing the attack

According to the information, Hortis needs exploit code to take advantage of the buffer overrun, a server to deliver up the malicious buffer overrun code, and a user to run the request to Hortis's server through the ISA proxy Web server. Then, if the ISA server is not patched, the attack can take effect.

The web sites Hortis found explained how to trigger the attack, but they didn't contain any actual exploit code. Thus Hortis has to either write his own from scratch, or search the seamy underside of the Internet for a prewritten exploit.

Setting up a server for the attack is simple. Hortis doesn't actually need a Gopher server, just a program listening for the connection attempt. This program opens a server socket and returns the exploit code when a client connects.

Lastly, Hortis has to have a user inside of GIAC make the connection to his gopher server. This will cause the exploit to run in the proxy on the ISA server. The simplest way to do this is to send users email with the URL contained within it. The social engineering involved is both simple and familiar. Common approaches are to:

1) offer free, sexually explicit content via the URL.
2) claim that a prize of some sort is available via the URL.
3) offer humorous or otherwise interesting content via the URL.

Hortis decides to offer a chance to enter a contest to win a laptop computer. His email is simply a copy of a well know company's email to him on the very subject. Hortis substitutes his URL for the URL of the real contest's web site. To protect the innocent, only the original and substituted tags are included here; and the original tag has a fake URL.

Original hypertext link:

<TD align=middle width=450 colSpan=2> To enter <A href="http://www.click-for-contest.com/contest/contest.htm">Click

here</A></TD></TR></TBODY></TABLE></TD></TR></TBODY></TABLE>
</BODY></HTML>

Hortis's version:

<TD align=middle width=450 colSpan=2> To enter <A href="gopher://192.168.17.45:7000/0">Click

here</A></TD></TR></TBODY></TABLE></TD></TR></TBODY></TABLE>
</BODY></HTML>

In order to get email accounts for users at GIAC, Hortis finds out that GIAC conveniently offers Microsoft Outlook Web Access and allows guests to peruse their entire address book to find email addresses for all the employees. Since the listings also state the employees' department, Hortis is able to avoid the security, network and system administration staff, targeting more likely candidates in production and administration.

Now, Hortis is already in lots of trouble and doesn't want more, so he needs to make sure that the attack can't be traced back to him. There are many ways to strip the email of headers; Hortis finds an anonymous remailer that does this. The hard part is finding a host to act as his gopher server. This server will have to remain available for hours or days, waiting for the email victim to connect. And if the victim can locate this server, so can the police. All the information will be in the email that launches the attack.

Since Hortis expects the police to find the gopher server, this host must be Hortis's first victim. How Hortis gains control of this host is another story that, unfortunately, we don't have time for here.

4.2.3 The Results

According to the hot fix patch for the ISA proxy server

(http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=2581B8C5-E709-4914-91BC-CFA13D031BC8) :

> "When a malicious Gopher request is received, the Proxy Server 2.0 Web Proxy may send back an invalid response, generate an access violation error message, and stop providing services."

Hortis can actually test his exploit in advance since Microsoft offers a 120-day trial version of the ISA Server, downloadable from the Microsoft web site:

http://www.microsoft.com/isaserver/evaluation/trial/default.asp

If Hortis has an exploit that, as stated in the Microsoft document, can just crash the ISA server, then, he only needs to know if it succeeded or failed. In order to know if his attack is successful, Hortis has to simply have his fake gopher server send a follow up packet to close the connection (gopher uses the tcp protocol) to the targeted client inside GIAC's network. If the client responds (sends an ack packet in response to Hortis's fin packet), the attack failed (meaning the

vulnerability was patched.) If there is no response, then Hortis has successfully taken down the ISA Proxy Server on the GIAC firewall.

If Hortis found or constructed an exploit that takes over the ISA server, then his attack program must verify the zombie status of the ISA server through some communication with the exploit code running on the ISA server.

In both cases, the attack program Hortis uses will send a specially worded message to some predesignated, high traffic site that Hortis can monitor without drawing suspicion to himself. Hortis has chosen a popular Bulletin Board System (BBS). The attack program will post a prewritten, innocuous sounding message. The subject line will signal success or failure. Only Hortis will understand the posting's real meaning. Then the attack program erases itself, helping to break the trail back to Hortis.

4.2.4 Defensive measures

This attack on the firewall is merely a specific case of the general case of an attack on a host. See the list of defensive measures under Targeting a Particular Host

4.3 Denial of Service Attack

Definition:

A denial of service (DOS) attack consumes all available instances of a resource, preventing other hosts or processes from being able to carry on normal activity that requires the subject resource.

4.3.1 The Design

Hortis has 50 zombie hosts with broadband connections via cable modem or DSL. These hosts typically have a maximum upstream bandwidth of 128 kilobits per second (kbps) (see http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci213915,00.html). Since Hortis can harness 50 of these zombies, he has (50 x 128kbps =) 6.4 million bits per second (Mbps) of available peak bandwidth for his attack. If Hortis's peak transmission bandwidth exceeds GIAC's peak receive bandwidth, then Hortis can perform a Distributed DOS (DDOS) attack on GIAC by filling GIAC's connection to the Internet. Since GIAC is an online company, this is equivalent to shutting down GIAC's business.

From the earlier reconnaissance on GIAC's network, Hortis knows that the gateway router is a Cisco 2600. According to the Cisco spec sheet:

http://www.cisco.com/en/US/products/hw/routers/ps259/products_data_sh

eet09186a00800a912b.html

the 2600 can handle a maximum of 8 T1/E1 connections. Since the T1 is 1.55Mbps and the E1 is 2Mbps, the maximum bandwidth of a wire into GIAC is an E1 at 2Mbps. Since Hortis found only one line between GIAC and it's ISP, that means Hortis cam fill the bandwidth, preventing anybody else from establishing, or for practical purposes, using a connection to GIAC.

GIAC, by itself, cannot prevent an attack on its bandwidth. GIAC must determine where the attack is coming from and gain the cooperation of administrators of the routers between itself and the attacking hosts. GIAC will probably contact its ISP (by phone) and ask for the flood of traffic to be identified and dropped at the ISP's outer perimeter. By varying the signature of the attack (type of packets, spoofing addresses), each of Hortis's zombies can continue to cause trouble via a different vector until the owner or ISP of the zombie finally shuts it down. This can take time. If Hortis doesn't need all of his zombies to mount a successful attack, then the extra ones can be held in reserve to reinforce and renew the attack as the original hosts get shutdown.

The response to the DDOS attack requires human intervention. Thus the timing of the attack (i.e. off hours in GIAC's time zone) can mean a slower response to the attack. But the attack has to occur when GIAC's customers, vendors and partners are trying to connect to the GIAC site. Intelligence about GIAC's business would help to target the best window for an attack. Ideally, the attack should be started after hours in GIAC's time zone and at the start of normal hours in the time zone of its major customers, vendors and partners. Also, the International Dateline can assist in the planning if much of GIAC's business originates in Asia while GIAC resides in North America. Then an attack can start on Sunday in North America (very, very much off hours) while it's a normal Monday workday in Asia. If GIAC's business originates in Europe, a different timing is called for. Holidays in the various countries can also impact the effectiveness of the attack. Hortis needs to study GIAC's business in order to cause the most damage.

If GIAC's connection to the Internet had exceeded Hortis's maximum attack bandwidth, then Hortis would have to target a host (or hosts) on GIAC's network and overwhelm it (them) in some manner. This could be as simple as the jolt2 attack that floods a Windows host with unassembled packet fragments (see SANS GCFW "IP Behavior III" page 4-30) or the generic and venerable tcp SYN flood (http://www.cert.org/advisories/CA-1996-21.html). To maximize the effectiveness of this attack, Hortis needs to know more about how GIAC's network is organized.

For example, if more than one service is carried on a single host (as the perimeter network's host GIACWEB does ftp as well as http), a DOS attack on that particular host takes down two or more services. And which services are

most critical to GIAC depends upon how GIAC conducts it's business. Thus visits to GIAC's site can tell Hortis which services GIAC most depends upon.

But due to the narrow nature of the pipe into GIAC, Hortis chooses to launch a simple DDOS attack, and flood the pipe with a mix of traffic (icmp, http, ftp, dns, smtp, etc) from 16 zombies. The zombies will (if their connection allows it) spoof their IP address, thus requiring more effort in tracking down the attackers and shutting them off at the source.

This attack across many protocols and services means that GIAC's ISP can't just shutdown one protocol or service. The more time consuming task of tracing (the hopefully spoofed) traffic will be required. Of course, the ISP could just drop all traffic inbound to GIAC, but that makes the DOS attack succeed! As the attack has been timed for off hours, detection and remediation resources at both GIAC and its ISP should be minimal.

These 16 hosts should be the minimum required to flood the GIAC connection to the Internet. The other 34 zombies will be held in reserve. They will randomly try to make an http or ftp connection to GIAC every 10 to 30 seconds. If the connection succeeds, that zombie will join the attack. Thus, as GIAC's ISP succeeds in shutting off the attackers one by one, the reserve zombie hosts will take up the slack. This will both lengthen the attack, and withhold information on how numerous the zombie army is.

4.3.2 Defensive Measures

There really are no defenses against a DOS attack. There are things that can be done to mitigate the effects, but they are usually expensive.

For example, GIAC could have had a higher bandwidth connection to the Internet. Hortis only had 6.4Mbps of attacking bandwidth. But more bandwidth costs money and it can be overcome by more zombie attackers.

GIAC could have arranged for more remediation resources to be called upon in case of an attack. If the ISP doesn't offer those resources and GIAC doesn't want to hire the extra staff itself, then a security service can be hired. Again, all of these steps cost more money.

GIAC needs to have a security plan in place that describes, in detail, what to do in case of an attack. Included in those steps ought to be both a call to law enforcement and steps for the capture of forensic evidence. This will not stop the current attack, but it is a necessary step to take that, over the long term, raises the cost to the attacker. Maybe, someday, the frequency of attacks will decline because of vigorous and successful law enforcement.

4.4 Targeting a Particular Host

4.4.1 Selection of the Victim

The victim selected is the one noted above; the dual ftp and http server on the perimeter network. It is selected both because it supports two protocols and because it is the lone NT host in a W2k network.

Because it supports both ftp and http, a vulnerability in either protocol can compromise the host and automatically compromise the other service. It's a two for one deal.

Also, since this is the only NT host in an otherwise W2k environment, the chances are greater for mis-management of the host by its administrator. The administrator has to concentrate on his/her most prevalent platform first; more bang for the buck. That increases the chances of short changing the minor platform by either skipping some maintenance or doing it in a hurried manner and possibly doing it wrong.

4.4.2 How to Design an Attack

Because the host provides services to clients across the Internet, the perimeter defenses at GIAC will allow traffic relating to those services through. The attacker doesn't have to defeat the router or the firewall, both will cooperate if the traffic looks legitimate. Thus we need to research the five protocols:

1) http
2) ftp
3) ip
4) tcp
5) imp

for vulnerabilities that may exist in either IIS or NT. Just as with the attack on the firewall, we will search for these vulnerabilities at:

1) the vendor's web site
2) security web sites (SANS, CERT, Bugtraq)
3) hacker sites

From Microsoft's s site, all the NT 4.0 service packs are listed at:

http://support.microsoft.com/default.aspx?scid=/support/servicepacks/WinNT/4.0/default.asp

Starting with the latest package (SP6a) at:

http://support.microsoft.com/default.aspx?scid=/support/servicepacks/Win

NT/4.0/SP6a.asp

the list of security fixes is listed. In SP6a, there are hundreds of patches, including 58 for the basic OS, 21 for IIS and 31 for security. If the service pack has not been applied and no individual hot fix for the vulnerability has been applied either, then the host may fall to an attack through this vector. Each vulnerability must be read to determine if a compromise is possible. Simply crashing the system, or having it misbehave isn't enough.

From the Bugtraq site:

> http://online.securityfocus.com/archive/

It is prudent to search for any vulnerability discovered after the release of SP6a (11/30/1999). A quick search of the archives for "NT 4.0" provided 186 posting after the SP6a date to pursue, while "MS IIS" showed 67 items.

And there are numerous hacking sites, all of which this author avoids.

4.4.3 Defensive measures

The number one defensive measure is to apply all service packs, security updates, hot fixes, etc. The systems administrator must keep current on patching vulnerabilities. The attacker (as noted above) will look at vendor updates, searching for a vulnerability to exploit. If the intended victim doesn't bother to apply the patches that the attacker is researching, then the game is certainly lost. And as obvious a step as this is, the recent Slammer event showed that most organizations don't do it. Even Microsoft had not applied its own patch to its own internal hosts six months after releasing the patch:

> http://zdnet.com.com/2100-1105-982305.html

The second defensive step to take is to minimize the effects of a successful attack. In the example above, two different, but vital services were hosted on the same machine. A successful attack against either delivered the second service for free.

A third defensive measure is to reduce the workload by making administration as scalable as possible. In the example network above, the target host was the lone NT 4.0 host. The administrator thus had two platforms (W2k and NT 4.0) to administer on the vulnerable perimeter services net. Diversity is often promoted as a defense against attack. However, it also provides both more potential vulnerabilities for the administrator to fix and more potential vulnerabilities for attacker to test.

The network appears to be lacking a common defensive measure; there isn't

mention of a remote logging feature. If (when?) a host is compromised, the host-based logfiles (if any) can be sanitized by the attacker. The compromise may never be detected until the attacker decides to tip his/her hand.

Also, as mentioned above in the DOS defensive measures, a security plan detailing what to do (including contacting law enforcement and collecting forensic evidence) is a useful precaution.

Lastly, the network administrator has to have an active defense as well as a passive defense. On the perimeter network was an IDS system. If (when?) a host becomes compromised, intrusion detection systems (both network and host based) are the last line of defense that allows the administrator to defeat the attacker even after the attacker has experienced initial success. Without IDS, the defenses are just a hard shell that delay but don't defeat the attacker. Without IDS, a single mistake by the defense forfeits the entire game. No network can win that kind of a game and no administrator should want to fight on those terms.

5.0 References

Andreasson, Oskar. “Iptables Tutorial 1.1.11” Version 3.14159-1.00a-pretest-20010004-ojmw. 21-March-2002. URL http://www.netfilter.org/documentation/tutorials/blueflux/

Bartz, Manfred. NetFilter Log Format”. URL http://logi.cc/linux/netfilter-log-format.php3

Brockmeier, Joe. “Iptables connection tracking”. 2-October-2002. URL http://www.sns.ias.edu/~jns/security/iptables/iptables_conntrack.html

Computer System Technology, National Institute of Standards and Technology, U.S. Department of Commerce. “NIST Special Publication 800-7; Security in Open Systems”. July-1994. URL http://csrc.nist.gov/publications/nistpubs/800-7/

The Internet Society. “Special-Use IPv4 Addresses”. Request for Comments: 3330. September-2002. URL ftp://ftp.rfc-editor.org/in-notes/rfc3330.txt

Roesch, Martin. “Snort 1.9.0 Ruleset”. 11-September-2002. URL http://www.snort.org/

Russell, Rusty. “Linux 2.4 NAT HOWTO”, Revision 1.18. 14-January-2002. URL http://www.netfilter.org/documentation/HOWTO//NAT-HOWTO.txt

Russell, Rusty. “Linux 2.4 Packet Filtering HOWTO”, Revision 1.26. 24-January-2002. URL http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.txt

Stevens, Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Longman, 1994.

Appendix A.

```
########
######## Template NetFilter bash script
########

########
######## macros for common values
########

###### executable

###### interfaces

###### local subnet hosts
######
###### except for SSMS host,
###### at least a LOCAL_SSMS
###### must be defined here

########
######## Initialization
########

###### Load modules

###### Set chain policies
######
###### This is GIAC's default security posture;
###### DROP it.  This allows the ruleset to grow
###### by simply allowing in whatever new traffic
###### that is required.  Since the new rules
###### can be added directly before the DROP
###### rule in the specific chain, there is
###### no great security concern about emergency
###### maintenance to the ruleset.  Any
###### performance concerns can be addressed
###### by a thoughtful review of the entire
###### ruleset at a later time.
######
###### must contain these lines only
###### $IPTABLES -P INPUT DROP
###### $IPTABLES -P OUTPUT DROP
###### $IPTABLES -P FORWARD DROP
######
###### $IPTABLES -t nat -P PREROUTING DROP
###### $IPTABLES -t nat -P POSTROUTING DROP
###### $IPTABLES -t nat -P OUTPUT DROP
######
###### $IPTABLES -t mangle -P PREROUTING ACCEPT
###### $IPTABLES -t mangle -P OUTPUT ACCEPT

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP
```

```
$IPTABLES -t nat -P PREROUTING DROP
$IPTABLES -t nat -P POSTROUTING DROP
$IPTABLES -t nat -P OUTPUT DROP

$IPTABLES -t mangle -P PREROUTING ACCEPT
$IPTABLES -t mangle -P OUTPUT ACCEPT

###### Enable/Disable (1/0) IP Forwarding
######
###### echo "0" > /proc/sys/net/ipv4/ip_forward
echo "0" > /proc/sys/net/ipv4/ip_forward


########
######## Create User chains
########

###### site-policy-chain


########
######## Mangle
########

########
######## NAT
########

###### DNAT: NATing new connections from the exterior interface

###### SNAT: NATing new connections from the interior interface

########
######## Filtering
########

###### INPUT chain ######


###### OUTPUT chain ######

#### FORWARD chain ######
```

Appendix B.

```
#!/bin/bash

########
######## Template NetFilter bash script
########

#### Border Router
#### by Greg Leisner

## For testing purposes, a limited number of
## conditionally executed LOG rules will
## log otherwise silently dropped packets.
##
## This logging is activated by invoking this
## script with a single argument of '--test'.

########
######## macros for common values
########

###### executable

IPTABLES=/sbin/iptables

###### interfaces

IF_Exterior="eth0"
IF_Interior="eth1"
IP_Exterior="102.17.94.35"
IP_Exterior_Mask="102.17.0.0/16"
IP_Interior="192.168.0.2"

## exterior services subnet
ES_Server_HTTP="192.168.1.2"
ES_Server_SMTP="192.168.1.3"
ES_Server_FTP="192.168.1.4"

###### local subnet hosts
######
###### except for SSMS host,
###### at least a LOCAL_SSMS
###### must be defined here
LOCAL_SSMS="192.168.0.1"
EG_Server_SSH="192.168.0.3"

## interior subnets
IP_ADMIN_SUBNET="10.3.0.0/16"
IP_PROD_SUBNET="10.2.0.0/16"
IP_SEC_SUBNET="10.255.0.0/16"

## Internet based services
ISP_DNS_PRIMARY="102.17.194.1"
ISP_DNS_SECONDARY="102.17.35.40"
ISP_SMTP_PRIMARY="mailhost1.mytown.myisp.net"
```

```
ISP_SMTP_SECONDARY="mailhost2.mytown.myisp.net"
NPT_SERVER_1="ntp1.cs.wisc.edu"
NPT_SERVER_2="ntp3.sf-bay.org"
NPT_SERVER_3="ntp-1.mcs.anl.gov"



########
######## Initialization
########

###### Load modules

/sbin/depmod -a -q

/sbin/modprobe -q ip_tables
/sbin/modprobe -q ip_conntrack
/sbin/modprobe -q iptable_filter
/sbin/modprobe -q iptable_mangle
/sbin/modprobe -q iptable_nat
/sbin/modprobe -q ipt_LOG
/sbin/modprobe -q ipt_limit
/sbin/modprobe -q ipt_state
/sbin/modprobe -q ipt_REJECT
/sbin/modprobe -q ip_conntrack_ftp
/sbin/modprobe -q ip_nat_ftp
/sbin/modprobe -q ipt_length
/sbin/modprobe -q ipt_ttl

###### Set chain policies
######
###### This is GIAC's default security posture;
###### DROP it.  This allows the ruleset to grow
###### by simply allowing in whatever new traffic
###### that is required.  Since the new rules
###### can usually be added directly before the
###### final DROP rule in the specific chain, there
###### is no great security concern about emergency
###### maintenance to the ruleset.  Any
###### performance concerns can be addressed
###### by a thoughtful review of the entire
###### ruleset at a later time.
######
###### must contain these lines _only_
###### /sbin/iptables -P INPUT DROP
###### /sbin/iptables -P OUTPUT DROP
###### /sbin/iptables -P FORWARD DROP
######
###### /sbin/iptables -t nat -P PREROUTING ACCEPT
###### /sbin/iptables -t nat -P POSTROUTING ACCEPT
###### /sbin/iptables -t nat -P OUTPUT ACCEPT
######
###### /sbin/iptables -t mangle -P PREROUTING ACCEPT
###### /sbin/iptables -t mangle -P OUTPUT ACCEPT

/sbin/iptables -P INPUT DROP
```

```
/sbin/iptables -P OUTPUT DROP
/sbin/iptables -P FORWARD DROP

/sbin/iptables -t nat -P PREROUTING ACCEPT
/sbin/iptables -t nat -P POSTROUTING ACCEPT
/sbin/iptables -t nat -P OUTPUT ACCEPT

/sbin/iptables -t mangle -P PREROUTING ACCEPT
/sbin/iptables -t mangle -P OUTPUT ACCEPT

###### Enable/Disable (1/0) IP Forwarding
######
###### echo "0" > /proc/sys/net/ipv4/ip_forward
echo "1" > /proc/sys/net/ipv4/ip_forward


########
######## Create User chains
########

###### site-policy-chain

## There is no overriding site policy chain at this time.

#### tcp-chain

## This chain checks the tcp header; it will
## only LOG and DROP bad packets.  The main
## chain will have to ACCEPT the packet.
##
## If the default logging isn't desired, the
## host will have to do tcp checks in the
## main chains.
##
## The order is unimportant except that the
## syn/fin must be last because we want to
## catch all other combinations not enumerated
## earlier in the chain.

/sbin/iptables -N tcp-chain
## data in syn packet; sid: 526
/sbin/iptables -A tcp-chain -p tcp --tcp-flags SYN SYN -m length ! --length 40:40 -j LOG --log-
prefix "tcp_chain: syn packet data "
/sbin/iptables -A tcp-chain -p tcp --tcp-flags SYN SYN -m length ! --length 40:40 -j REJECT --
reject-with tcp-reset
## null scan; sid: 623
/sbin/iptables -A tcp-chain -p tcp --tcp-flags ALL NONE -j LOG --log-prefix "tcp_chain: null scan "
/sbin/iptables -A tcp-chain -p tcp --tcp-flags ALL NONE -j REJECT --reject-with tcp-reset
## xmas scan; sid: 625
/sbin/iptables -A tcp-chain -p tcp --tcp-flags ALL ALL -j LOG --log-prefix "tcp_chain: xmas scan "
/sbin/iptables -A tcp-chain -p tcp --tcp-flags ALL ALL -j REJECT --reject-with tcp-reset
## cybercop scan; sid: 627
/sbin/iptables -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,URG -j LOG --log-prefix "tcp_chain:
cybercop scan "
/sbin/iptables -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,URG -j REJECT --reject-with tcp-reset
## nmap fingerprint scan; sid: 629
```

```
/sbin/iptables -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j LOG --log-prefix
"tcp_chain: nmap scan "
/sbin/iptables -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j REJECT --reject-with tcp-
reset
## syn/fin scan; sid: 624, 630
/sbin/iptables -A tcp-chain -p tcp --tcp-flags SYN,FIN SYN,FIN -j LOG --log-prefix "tcp_chain:
syn/fin scan "
/sbin/iptables -A tcp-chain -p tcp --tcp-flags SYN,FIN SYN,FIN -j REJECT --reject-with tcp-reset

#### udp-chain

## There is no udp-chain because the only check is on
## ports and they are validated in the NAT process.

#### icmp-chain

## There is no icmp-chain because the only checks
## are on:
##
##     1) connection-state and they are validated
##        in the NAT process.
##     2) ping rules in the Input and Forward chains.

########
######## Mangle
########

## There are no mangle rules at this time.

########
######## NAT
########

## nat rules are not logged, this should occur if
## and when the connected is ACCEPTed.

###### DNAT: NATing new connections from the exterior interface

## in order of expected volume:HTTP, SMTP, FTP, SSH
/sbin/iptables -t nat -A PREROUTING -d $IP_Exterior -p tcp --dport 80 -j DNAT --to
$ES_Server_HTTP
/sbin/iptables -t nat -A PREROUTING -d $IP_Exterior -p tcp --dport 25 -j DNAT --to
$ES_Server_SMTP
/sbin/iptables -t nat -A PREROUTING -d $IP_Exterior -p tcp --dport 21 -j DNAT --to
$ES_Server_FTP
/sbin/iptables -t nat -A PREROUTING -d $IP_Exterior -p tcp --dport 22 -j DNAT --to
$EG_Server_SSH


###### SNAT: NATing new connections from the interior interface

## no protocol or address filtering here
/sbin/iptables -t nat -A POSTROUTING -o $IF_EXTERIOR -j SNAT --to-source $IP_Exterior


########
```

```
######## Filtering
########

###### INPUT chain ######

## The first rules applied are the overriding,
## always enforced, site policy.
##
## Then, the order is sensitive to volume,
## both by connection-state and within
## each connection-state.
##
##     1) established connections
##     2) new connections from the Exterior Interface
##     3) new connections from the Interior Interface

## Since we have validated the IP connection,
## pass the existing connections.
/sbin/iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

## Next, allow Unix socket stuff.
/sbin/iptables -A INPUT -i lo -j ACCEPT

## The exterior interface is first because this is where most of
## the traffic will be coming from.

## IF_Exterior ##

## The Border router is different from all other filtering rule sets
## because on the exterior interface we will generally drop all unwanted
## packets without logging them.  The reason for this is that GIAC
## isn't responsible for remediation of any issues outside of it's
## network.  To do so would consume scarce internal resources (people
## time.)  An exception is made for some packets indicating a compromise
## on GIAC's ISP's network. (Neighborhood Watch)

## After NATing the packets in prerouting, anything left directed to
## our external IP address is probably trash.  We respond to pings
## from our ISP's address space, but that's all.

## ping traffic
## first DROP anything not from the ISP's address space
if [ "$1" == "--test" ]; then
        /sbin/iptables -A INPUT -s ! $IP_Exterior_Mask -p icmp --icmp-type 8 -j LOG --log-prefix
"testing: INPUT DROP "
fi
/sbin/iptables -A INPUT -i $IF_Exterior -s ! $IP_Exterior_Mask -p icmp --icmp-type 8 -j DROP
## the remaining ping traffic is logged and DROP'd if excessive
/sbin/iptables -A INPUT -i $IF_Exterior -m limit ! --limit 20/hour -p icmp --icmp-type 8 -j LOG --log-
prefix "ISP: excessive pings "
/sbin/iptables -A INPUT -i $IF_Exterior -m limit ! --limit 20/hour -p icmp --icmp-type 8 -j DROP
## This ping we will respond to
/sbin/iptables -A INPUT -i $IF_Exterior -p icmp --icmp-type 8 -j ACCEPT

## Bad traffic & scans; just grab enough to alert the ISP to the problem.
## Only grab packets that can't be false positives.
```

```
## There is more that we could do, but we just keep it simple.

## reserved IP addresses; rfc:3330
/sbin/iptables -A INPUT -i $IF_Exterior -m limit --limit 1/hour -s 10.0.0.0/8 -j LOG --log-prefix "ISP:
bad address 1 "
/sbin/iptables -A INPUT -i $IF_Exterior -m limit --limit 1/hour -s 127.0.0.0/8 -j LOG --log-prefix
"ISP: bad address 2 "
/sbin/iptables -A INPUT -i $IF_Exterior -m limit --limit 1/hour -s 169.254.0.0/16 -j LOG --log-prefix
"ISP: bad address 3 "
/sbin/iptables -A INPUT -i $IF_Exterior -m limit --limit 1/hour -s 172.16.0.0/16 -j LOG --log-prefix
"ISP: bad address 4 "
/sbin/iptables -A INPUT -i $IF_Exterior -m limit --limit 1/hour -s 192.0.2.0/24 -j LOG --log-prefix
"ISP: bad address 5 "
/sbin/iptables -A INPUT -i $IF_Exterior -m limit --limit 1/hour -s 192.168.0.0/16 -j LOG --log-prefix
"ISP: bad address 6 "
/sbin/iptables -A INPUT -i $IF_Exterior -m limit --limit 1/hour -s 192.18.0.0/15 -j LOG --log-prefix
"ISP: bad address 7 "
/sbin/iptables -A INPUT -i $IF_Exterior -m limit --limit 1/hour -s 240.0.0.0/4 -j LOG --log-prefix
"ISP: bad address 8 "
## data in syn packet; sid: 526
/sbin/iptables -A INPUT -i $IF_Exterior -p tcp --tcp-flags SYN SYN -m length ! --length 40:40 -m
limit --limit 1/hour -j LOG --log-prefix "ISP: data in syn packet"
## null scan; sid: 623
/sbin/iptables -A INPUT -i $IF_Exterior -p tcp --tcp-flags ALL NONE -m limit --limit 1/hour -j LOG --
log-prefix "ISP: null scan"
## xmas scan; sid: 625
/sbin/iptables -A INPUT -i $IF_Exterior -p tcp --tcp-flags ALL ALL -m limit --limit 1/hour -j LOG --
log-prefix "ISP: xmas scan"
## cybercop scan; sid: 627
/sbin/iptables -A INPUT -i $IF_Exterior -p tcp --tcp-flags ALL SYN,FIN,URG -m limit --limit 1/hour -
j LOG --log-prefix "ISP: cybercop scan"
## nmap fingerprint scan; sid: 629
/sbin/iptables -A INPUT -i $IF_Exterior -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -m limit --limit
1/hour -j LOG --log-prefix "ISP: nmap scan"
## syn/fin scan; sid: 624, 630
/sbin/iptables -A INPUT -i $IF_Exterior -p tcp --tcp-flags SYN,FIN SYN,FIN -m limit --limit 1/hour -j
LOG --log-prefix "ISP: syn/fin scan"

## The rest from the Exterior is all trash
if [ "$1" == "--test" ]; then
        /sbin/iptables -A INPUT -i $IF_Exterior -j LOG --log-prefix "testing: INPUT DROP "
fi
/sbin/iptables -A INPUT -i $IF_Exterior -j DROP

## IF_Interior ##
##
## The interface must be the interior, so -i $IF_Interior is implicit in all the following

## Our major activity for the INPUT chain is control from the
## Interior Network.  We only accept SSH connects from our
## local subnet's SSMS or the Security subnet; the rest get
## logged and dropped.

## log and pass the new connections
/sbin/iptables -A INPUT -s $LOCAL_SSMS -p tcp --dport 22 -m state --state NEW -j LOG --log-
prefix "Bdr Router NEW INPUT "
```

```
/sbin/iptables -A INPUT -s $LOCAL_SSMS -p tcp --dport 22 -m state --state NEW -j ACCEPT
/sbin/iptables -A INPUT -s $IP_SEC_SUBNET -p tcp --dport 22 -m state --state NEW -j LOG --
log-prefix "Bdr Router NEW INPUT "
/sbin/iptables -A INPUT -s $IP_SEC_SUBNET -p tcp --dport 22 -m state --state NEW -j ACCEPT

## The rest should be invalid and could signal an attack
/sbin/iptables -A INPUT -j LOG --log-prefix "Bdr Router invalid INPUT "
/sbin/iptables -A INPUT -j DROP


###### OUTPUT chain ######

## The first rules applied are the overriding,
## always enforced, site policy.
##
## Then, the order is simply by volume.  There
## are few rules because there is little traffic.

## site policy
#/sbin/iptables -j site-policy-chain
/sbin/iptables -A OUTPUT -o lo -j ACCEPT

## Top priority is the syslog dump.
/sbin/iptables -A OUTPUT -o $IF_Interior -d $LOCAL_SSMS -p udp --dport 514 -j ACCEPT

## check all tcp traffic
/sbin/iptables -A OUTPUT -p tcp -j tcp-chain

## pass the existing connections
/sbin/iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

## allow out valid DNS traffic
/sbin/iptables -A OUTPUT -o $IF_Interior -d $LOCAL_SSMS -p udp --dport 53 -m state --state
NEW -j ACCEPT

## allow out valid NTP traffic
/sbin/iptables -A OUTPUT -o $IF_Interior -d $LOCAL_SSMS -p udp --dport 123 -m state --state
NEW -j ACCEPT

## The Border Router is passive; syslog, DNS & NTP above are it's only
## internally initiated IP traffic.  The rest should be invalid
## and could signal a compromise.
/sbin/iptables -A OUTPUT -j LOG --log-prefix "Bdr Router invalid OUTPUT "
/sbin/iptables -A OUTPUT -p tcp -j REJECT --reject-with tcp-reset
/sbin/iptables -A OUTPUT -j REJECT


#### FORWARD chain ######

## Here is where the real work gets done.
##
## The first rules applied are the overriding,
## always enforced, site policy.
##
## Then, the order is sensitive to volume,
## both by connection-state and within
```

```
## each connection-state.
##
##     1) established connections
##     2) inbound new connections
##     3) outbound new connections

## first, rules for both directions

## check all tcp traffic
/sbin/iptables -A FORWARD -p tcp -j tcp-chain

## Since we have validated the IP connection,
## pass the existing connections.
/sbin/iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

## IF_Exterior ##

## From the Internet
##
## Because we use NAT, traffic inbound from the Internet
## either has matched our DNAT targets, or it is part of
## an existing connection.
##
## Since all hosts filter interior traffic, any checks made
## here at the Border Router of Internet traffic would
## duplicate those checks.  However, since all interior
## hosts log their invalid packets, the Border Router will
## filter all syn packets related to DNAT targets and
## silently discard the bad ones.  This is done instead
## of using the tcp-chain because GIAC will not waste
## time trying to remediate the situation.
##

## tcp ##

## NEW packet, no syn; see Iptables Tutorial App B
if [ "$1" == "--test" ]; then
        /sbin/iptables -A FORWARD -i $IF_Exterior -p tcp ! --tcp-flags SYN SYN -m state --state
NEW -j LOG --log-prefix "testing: FORWARD DROP "
fi
/sbin/iptables -A FORWARD -i $IF_Exterior -p tcp ! --tcp-flags SYN SYN -m state --state NEW -j
DROP

## IDENT, quickly, gracefully kill all INDENT requests;
##     SANS course materials-fw_26_netfilter_gaunt.pdf, page 210
/sbin/iptables -A FORWARD -i $IF_Exterior -p tcp --dport 113 -j REJECT --reject-with tcp-reset

## data in syn packet; sid: 526
if [ "$1" == "--test" ]; then
        /sbin/iptables -A FORWARD -i $IF_Exterior -p tcp -m length ! --length 40:40 -j LOG --log-
prefix "testing: FORWARD DROP "
fi
/sbin/iptables -A FORWARD -i $IF_Exterior -p tcp -m length ! --length 40:40 -j DROP

## additional flags; sid:624, 625, 1228, 629, 630, 627
## because of the first rule in this section, these packets
```

```
## all have the SYN flag set
if [ "$1" == "--test" ]; then
        /sbin/iptables -A FORWARD -i $IF_Exterior -p tcp --tcp-flags FIN FIN -j LOG --log-prefix
"testing: FORWARD DROP "
        /sbin/iptables -A FORWARD -i $IF_Exterior -p tcp --tcp-flags RST RST -j LOG --log-prefix
"testing: FORWARD DROP "
        /sbin/iptables -A FORWARD -i $IF_Exterior -p tcp --tcp-flags URG URG -j LOG --log-
prefix "testing: FORWARD DROP "
fi
/sbin/iptables -A FORWARD -i $IF_Exterior -p tcp --tcp-flags FIN FIN -j DROP
/sbin/iptables -A FORWARD -i $IF_Exterior -p tcp --tcp-flags RST RST -j DROP
/sbin/iptables -A FORWARD -i $IF_Exterior -p tcp --tcp-flags URG URG -j DROP

## OK, looks good to accept new connections
/sbin/iptables -A FORWARD -i $IF_Exterior -p tcp -m state --state NEW -j LOG --log-prefix "NEW
Incoming to GIAC " --log-tcp-sequence
/sbin/iptables -A FORWARD -i $IF_Exterior -p tcp -m state --state NEW -j ACCEPT

## udp ##

# none allowed

## icmp ##

# none allowed

## lastly, out with the trash
if [ "$1" == "--test" ]; then
        /sbin/iptables -A FORWARD -i $IF_Exterior -j LOG --log-prefix "testing: FORWARD
DROP "
fi
/sbin/iptables -A FORWARD -i $IF_Exterior -j DROP

## IF_Interior ##

## To the Internet
##
## The interface must be the interior, so
## -i $IF_Interior is implicit in all the following.

## All dns is handled through the local SSMS.
## Since we are a single external IP, there are no
## zone transfers and udp should do but because of
## volume, we do not record NEW connections.
/sbin/iptables -A FORWARD -s $LOCAL_SSMS -d $ISP_DNS_PRIMARY -p udp --dport 53 -m
state --state NEW -j ACCEPT
/sbin/iptables -A FORWARD -s $LOCAL_SSMS -d $ISP_DNS_SECONDARY -p udp --dport 53 -
m state --state NEW -j ACCEPT

## We allow out HTTP from any address on the sec, admin and production subnets
## but because of volume, we do not record NEW connections
/sbin/iptables -A FORWARD -s $IP_ADMIN_SUBNET -p tcp --dport 80 --tcp-flags SYN SYN -m
state --state NEW -j LOG --log-prefix "NEW Out going from GIAC "
/sbin/iptables -A FORWARD -s $IP_ADMIN_SUBNET -p tcp --dport 80 --tcp-flags SYN SYN -m
state --state NEW -j ACCEPT
```

```
/sbin/iptables -A FORWARD -s $IP_PROD_SUBNET -p tcp --dport 80 --tcp-flags SYN SYN -m
state --state NEW -j LOG --log-prefix "NEW Out going from GIAC "
/sbin/iptables -A FORWARD -s $IP_PROD_SUBNET -p tcp --dport 80 --tcp-flags SYN SYN -m
state --state NEW -j ACCEPT
/sbin/iptables -A FORWARD -s $IP_SEC_SUBNET -p tcp --dport 80 --tcp-flags SYN SYN -m
state --state NEW -j LOG --log-prefix "NEW Out going from GIAC "
/sbin/iptables -A FORWARD -s $IP_SEC_SUBNET -p tcp --dport 80 --tcp-flags SYN SYN -m
state --state NEW -j ACCEPT

## allow out valid NTP traffic
/sbin/iptables -A FORWARD -s $LOCAL_SSMS -d $NPT_SERVER_1 -p udp --dport 123 -m state
--state NEW -j ACCEPT
/sbin/iptables -A FORWARD -s $LOCAL_SSMS -d $NPT_SERVER_2 -p udp --dport 123 -m state
--state NEW -j ACCEPT
/sbin/iptables -A FORWARD -s $LOCAL_SSMS -d $NPT_SERVER_3 -p udp --dport 123 -m state
--state NEW -j ACCEPT

## We allow out FTP from any address on the sec, admin and production subnets
## but we log it
/sbin/iptables -A FORWARD -s $IP_ADMIN_SUBNET -p tcp --dport 21 --tcp-flags SYN SYN -m
state --state NEW -j LOG --log-prefix "NEW Out going from GIAC "
/sbin/iptables -A FORWARD -s $IP_ADMIN_SUBNET -p tcp --dport 21 --tcp-flags SYN SYN -m
state --state NEW -j ACCEPT
/sbin/iptables -A FORWARD -s $IP_PROD_SUBNET -p tcp --dport 21 --tcp-flags SYN SYN -m
state --state NEW -j LOG --log-prefix "NEW Out going from GIAC "
/sbin/iptables -A FORWARD -s $IP_PROD_SUBNET -p tcp --dport 21 --tcp-flags SYN SYN -m
state --state NEW -j ACCEPT
/sbin/iptables -A FORWARD -s $IP_SEC_SUBNET -p tcp --dport 21 --tcp-flags SYN SYN -m
state --state NEW -j LOG --log-prefix "NEW Out going from GIAC "
/sbin/iptables -A FORWARD -s $IP_SEC_SUBNET -p tcp --dport 21 --tcp-flags SYN SYN -m
state --state NEW -j ACCEPT

## The only outbound SMTP is from the Exterior Services E-Mail proxy to our ISP
/sbin/iptables -A FORWARD -s $ES_Server_SMTP -d $ISP_SMTP_PRIMARY -p tcp --dport 25 --
tcp-flags SYN SYN -m state --state NEW -j LOG --log-prefix "NEW Out going from GIAC "
/sbin/iptables -A FORWARD -s $ES_Server_SMTP -d $ISP_SMTP_PRIMARY -p tcp --dport 25 --
tcp-flags SYN SYN -m state --state NEW -j ACCEPT
/sbin/iptables -A FORWARD -s $ES_Server_SMTP -d $ISP_SMTP_SECONDARY -p tcp --dport
25 --tcp-flags SYN SYN -m state --state NEW -j LOG --log-prefix "NEW Out going from GIAC "
/sbin/iptables -A FORWARD -s $ES_Server_SMTP -d $ISP_SMTP_SECONDARY -p tcp --dport
25 --tcp-flags SYN SYN -m state --state NEW -j ACCEPT

## Pings outbound are a responsibility of the Security staff only
/sbin/iptables -A FORWARD -s $LOCAL_SSMS -p icmp --icmp-type 8 -m state --state NEW -j
LOG --log-prefix "NEW Out going from GIAC "
/sbin/iptables -A FORWARD -s $LOCAL_SSMS -p icmp --icmp-type 8 -m state --state NEW -j
ACCEPT
/sbin/iptables -A FORWARD -s $IP_SEC_SUBNET -p icmp --icmp-type 8 -m state --state NEW -j
LOG --log-prefix "NEW Out going from GIAC "
/sbin/iptables -A FORWARD -s $IP_SEC_SUBNET -p icmp --icmp-type 8 -m state --state NEW -j
ACCEPT

## We don't expect anything else at this point, instead
## of the default policy, we log it and REJECT.  We
## REJECT for performance reasons; these _are_ our hosts.
/sbin/iptables -A FORWARD -j LOG --log-prefix "Bdr Router invalid FORWARD "
```

```
/sbin/iptables -A FORWARD -p tcp -j REJECT --reject-with tcp-reset
/sbin/iptables -A FORWARD -j REJECT
```

Appendix C.

```
#!/bin/sh

########
######## Template NetFilter bash script
########

#### Exterior Subnets Router
#### by Greg Leisner

########
######## macros for common values
########

###### executable

IPTABLES="/sbin/iptables"

###### interfaces

IF_EG="eth0"
IF_ES="eth1"
IF_Interior="eth2"
IP_EG="192.168.0.3"
IP_ES="192.168.1.5"
IP_Interior="192.168.3.2"
IP_EG_MASK="192.168.0.0/24"
IP_ES_MASK="192.168.1.0/24"
IP_Interior_Mask="192.168.3.0/24"
###### local subnet hosts
######
###### except for SSMS host,
###### at least a LOCAL_SSMS
###### must be defined here
LOCAL_SSMS="192.168.0.1"

## interior subnets
IP_ADMIN_SUBNET="10.3.0.0/16"
IP_PROD_SUBNET="10.2.0.0/16"
IP_SEC_SUBNET="10.255.0.0/16"

########
######## Initialization
########

###### Load modules

/sbin/depmod -a

/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe iptable_mangle
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
```

```
/sbin/modprobe ipt_state
/sbin/modprobe ipt_REJECT
/sbin/modprobe ip_conntrack_ftp
/sbin/modprobe ipt_ipv4options
/sbin/modprobe ipt_length
/sbin/modprobe ipt_ttl

###### Set chain policies
######
###### This is GIAC's default security posture;
###### DROP it.  This allows the ruleset to grow
###### by simply allowing in whatever new traffic
###### that is required.  Since the new rules
###### can be added directly before the DROP
###### rule in the specific chain, there is
###### no great security concern about emergency
###### maintenance to the ruleset.  Any
###### performance concerns can be addressed
###### by a thoughtful review of the entire
###### ruleset at a later time.
######
###### must contain these lines only
###### $IPTABLES -P INPUT DROP
###### $IPTABLES -P OUTPUT DROP
###### $IPTABLES -P FORWARD DROP
######
###### $IPTABLES -t nat -P PREROUTING DROP
###### $IPTABLES -t nat -P POSTROUTING DROP
###### $IPTABLES -t nat -P OUTPUT DROP
######
###### $IPTABLES -t mangle -P PREROUTING ACCEPT
###### $IPTABLES -t mangle -P OUTPUT ACCEPT

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

$IPTABLES -t nat -P PREROUTING DROP
$IPTABLES -t nat -P POSTROUTING DROP
$IPTABLES -t nat -P OUTPUT DROP

$IPTABLES -t mangle -P PREROUTING ACCEPT
$IPTABLES -t mangle -P OUTPUT ACCEPT

###### Enable/Disable (1/0) IP Forwarding
######
###### echo "0" > /proc/sys/net/ipv4/ip_forward
echo "1" > /proc/sys/net/ipv4/ip_forward


########
######## Create User chains
########

###### site-policy-chain
```

```
## No nonsense, always applied rules.

$IPTABLES -N site-policy-chain
## IP options shouldn't happen, and if they
## do, it's a "Bad Thing".  Don't even bother
## logging them; GIAC doesn't care what the
## intent was.  If this was part of an attack,
## we'll catch them some other way.  If it
## was innocent, we don't waste time on it.
$IPTABLES -A site-policy-chain -m ipv4options --rr -j DROP
$IPTABLES -A site-policy-chain -m ipv4options --ts -j DROP

#### tcp-chain

## This chain checks the tcp header; it will
## only LOG and DROP bad packets.  The main
## chain will have to ACCEPT the packet.
##
## If the default logging isn't desired, the
## host will have to do tcp checks in the
## main chains.
##
## The order is unimportant except that the
## syn/fin must be last because we want to
## catch all other combinations not enumerated
## earlier in the chain.

$IPTABLES -N tcp-chain
## data in syn packet; sid: 526
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN SYN -j LOG --log-prefix "tcp_chain: syn packet
data "
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN SYN -j DROP
## null scan; sid: 623
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL NONE -j LOG --log-prefix "tcp_chain: null scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL NONE -j DROP
## xmas scan; sid: 625
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL ALL -j LOG --log-prefix "tcp_chain: xmas scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL ALL -j DROP
## cybercop scan; sid: 627
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,URG -j LOG --log-prefix "tcp_chain:
cybercop scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,URG -j DROP
## nmap fingerprint scan; sid: 629
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j LOG --log-prefix "tcp_chain:
nmap scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j DROP
## syn/fin scan; sid: 624, 630
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN,FIN SYN,FIN -j LOG --log-prefix "tcp_chain:
syn/fin scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP

#### udp-chain

#### icmp-chain

########
```

```
######## Mangle
########

########
######## NAT
########

###### DNAT: NATing new connections from the exterior interface

###### SNAT: NATing new connections from the interior interface

########
######## Filtering
########

###### INPUT chain ######

## The first rules applied are the overriding,
## always enforced, site policy.

## site policy
$IPTABLES -j site-policy-chain

## check all tcp traffic
$IPTABLES -A INPUT -p tcp -j tcp-chain

## Since we have validated the IP connection,
## pass the existing connections.
$IPTABLES -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

## LOG and DROP all traffic with the wrong
## interface/IP address combination.
$IPTABLES -A INPUT -i $IF_EG -s ! $IF_EG_MASK -j LOG --log-prefix "EFW spoofing "
$IPTABLES -A INPUT -i $IF_EG -s ! $IF_EG_MASK -j DROP
$IPTABLES -A INPUT -i $IF_ES -s ! $IF_ES_MASK -j LOG --log-prefix "EFW spoofing "
$IPTABLES -A INPUT -i $IF_ES -s ! $IF_ES_MASK -j DROP
$IPTABLES -A INPUT -i $IP_Interior -s ! $IP_Interior_Mask -j LOG --log-prefix "EFW spoofing "
$IPTABLES -A INPUT -i $IP_Interior -s ! $IP_Interior_Mask -j DROP

## ping traffic
## the remaining ping traffic is logged and DROP'd if excessive
$IPTABLES -A INPUT -m limit ! --limit 20/hour -p icmp --icmp-type 8 -j LOG --log-prefix "EFW
excessive pings "
$IPTABLES -A INPUT -m limit ! --limit 20/hour -p icmp --icmp-type 8 -j DROP
## This ping we will respond to
$IPTABLES -A INPUT -p icmp --icmp-type 8 -j ACCEPT

## Our major activity for the INPUT chain is control from the
## Interior Network.  We only accept SSH connects from our
## local subnet's SSMS or the Security subnet.

## log and pass the new connections
$IPTABLES -A INPUT -i ! $IF_Interior -s $LOCAL_SSMS -p tcp --dport 22 -m state --state NEW -j
LOG --log-prefix "EFW NEW INPUT "
$IPTABLES -A INPUT -i ! $IF_Interior -s $LOCAL_SSMS -p tcp --dport 22 -m state --state NEW -j
ACCEPT
```

```
$IPTABLES -A INPUT -i $IF_Interior -s $IP_SEC_SUBNET -p tcp --dport 22 -m state --state
NEW -j LOG --log-prefix "EFW NEW INPUT "
$IPTABLES -A INPUT -i $IF_Interior -s $IP_SEC_SUBNET -p tcp --dport 22 -m state --state
NEW -j ACCEPT

## The rest should be invalid and could signal an attack
$IPTABLES -A INPUT -j LOG --log-prefix "EFW invalid INPUT " --log-level crit
$IPTABLES -A INPUT -j DROP


###### OUTPUT chain ######

## The first rules applied are the overriding,
## always enforced, site policy.
##
## Then, the order is simply by volume.  There
## are few rules because there is little traffic.

## site policy
$IPTABLES -j site-policy-chain

## Top priority is the syslog dump.
$IPTABLES -A OUTPUT -o $IF_Interior -d $LOCAL_SSMS -p udp --dport 514 -j ACCEPT

## check and pass the existing connections
$IPTABLES -A OUTPUT -p tcp -j tcp-chain
$IPTABLES -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

## allow out valid DNS traffic
$IPTABLES -A OUTPUT -o $IF_Interior -d $LOCAL_SSMS -p udp --dport 53 -m state --state
NEW -j ACCEPT

## allow out valid NTP traffic
$IPTABLES -A OUTPUT -o $IF_Interior -d $LOCAL_SSMS -p udp --dport 123 -m state --state
NEW -j ACCEPT

## The Exterior Firewall is passive; syslog & NTP above are it's only
## internally initiated IP traffic.  The rest should be invalid
## and could signal a compromise.
$IPTABLES -A OUTPUT -j LOG --log-prefix "EFW invalid OUTPUT " --log-level crit
$IPTABLES -A OUTPUT -p tcp -j REJECT --reject-with tcp-reset
$IPTABLES -A OUTPUT -j REJECT


#### FORWARD chain ######

## Here is where the real work gets done.
##
## The first rules applied are the overriding,
## always enforced, site policy.
##
## Then, the order is sensitive to volume;
## and a minimal ruleset is

## first, rules for both directions
```

```
## site policy
$IPTABLES -j site-policy-chain

## check all tcp traffic
$IPTABLES -A FORWARD -p tcp -j tcp-chain

## Since we have validated the IP connection,
## pass the existing connections.
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

## LOG and DROP all traffic with the wrong
## interface/IP address combination.
$IPTABLES -A INPUT -i $IF_EG -s ! $IF_EG_MASK -j LOG --log-prefix "EFW spoofing "
$IPTABLES -A INPUT -i $IF_EG -s ! $IF_EG_MASK -j DROP
$IPTABLES -A INPUT -i $IF_ES -s ! $IF_ES_MASK -j LOG --log-prefix "EFW spoofing "
$IPTABLES -A INPUT -i $IF_ES -s ! $IF_ES_MASK -j DROP
$IPTABLES -A INPUT -i $IP_Interior -s ! $IP_Interior_Mask -j LOG --log-prefix "EFW spoofing "
$IPTABLES -A INPUT -i $IP_Interior -s ! $IP_Interior_Mask -j DROP

## Now LOG and ACCEPT the NEW traffic.
## We now can do traffic analysis of traffic
## between our subnets, if need be.
$IPTABLES -A FORWARD -m state --state NEW -j LOG --log-prefix "EFW NEW FORWARD "
$IPTABLES -A FORWARD -m state --state NEW -j ACCEPT

## We don't expect anything else at this point, if
## there is anything, we log it and REJECT.  We
## REJECT for performance reasons; these _are_ our hosts.
$IPTABLES -A FORWARD -j LOG --log-prefix "EFW invalid FORWARD "
$IPTABLES -A FORWARD -p tcp -j REJECT --reject-with tcp-reset
$IPTABLES -A FORWARD -j REJECT
```

Appendix D.

```
#!/bin/sh

########
######## Template NetFilter bash script
########

#### VPN concentrator
#### by Greg Leisner

########
######## macros for common values
########

###### executable

IPTABLES="/sbin/iptables"

###### interfaces

IF_Exterior="eth0"
IF_Interior="eth1"
IP_Exterior="192.168.0.4"
IP_Exterior_Mask="192.168.0.0/24"
IP_Interior="192.168.2.2"

###### local subnet hosts
######
###### except for SSMS host,
###### at least a LOCAL_SSMS
###### must be defined here
LOCAL_SSMS="192.168.0.1"

## interior subnets
IP_10_SUBNET="10.0.0.0/8"
IP_192_168_SUBNET="192.168.0.0/16"
IP_ADMIN_SUBNET="10.3.0.0/16"
IP_GS_SUBNET="10.1.0.0/16"
IP_ES_SUBNET="192.168.1.0/24"
IP_PROD_SUBNET="10.2.0.0/16"
IP_SEC_SUBNET="10.255.0.0/16"

########
######## Initialization
########

###### Load modules

/sbin/depmod -a

/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
```

```
/sbin/modprobe ipt_state
/sbin/modprobe ipt_REJECT
/sbin/modprobe ip_conntrack_ftp
/sbin/modprobe ipt_ipv4options
/sbin/modprobe ipt_length

###### Set chain policies
######
###### This is GIAC's default security posture;
###### DROP it.  This allows the ruleset to grow
###### by simply allowing in whatever new traffic
###### that is required.  Since the new rules
###### can be added directly before the DROP
###### rule in the specific chain, there is
###### no great security concern about emergency
###### maintenance to the ruleset.  Any
###### performance concerns can be addressed
###### by a thoughtful review of the entire
###### ruleset at a later time.
######
###### must contain these lines only
###### $IPTABLES -P INPUT DROP
###### $IPTABLES -P OUTPUT DROP
###### $IPTABLES -P FORWARD DROP
######
###### $IPTABLES -t nat -P PREROUTING DROP
###### $IPTABLES -t nat -P POSTROUTING DROP
###### $IPTABLES -t nat -P OUTPUT DROP
######
###### $IPTABLES -t mangle -P PREROUTING ACCEPT
###### $IPTABLES -t mangle -P OUTPUT ACCEPT

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

$IPTABLES -t nat -P PREROUTING DROP
$IPTABLES -t nat -P POSTROUTING DROP
$IPTABLES -t nat -P OUTPUT DROP

$IPTABLES -t mangle -P PREROUTING ACCEPT
$IPTABLES -t mangle -P OUTPUT ACCEPT

###### Enable/Disable (1/0) IP Forwarding
######
###### echo "0" > /proc/sys/net/ipv4/ip_forward
echo "0" > /proc/sys/net/ipv4/ip_forward


########
######## Create User chains
########

###### site-policy-chain

## No nonsense, always applied rules.
```

```
$IPTABLES -N site-policy-chain
## IP options shouldn't happen, and if they
## do, it's a "Bad Thing".  Don't even bother
## logging them; GIAC doesn't care what the
## intent was.  If this was part of an attack,
## we'll catch them some other way.  If it
## was innocent, we don't waste time on it.
$IPTABLES -A site-policy-chain -m ipv4options --rr -j DROP
$IPTABLES -A site-policy-chain -m ipv4options --ts -j DROP

#### tcp-chain

## This chain checks the tcp header it will
## only LOG and DROP bad packets.  The main
## chain will have to ACCEPT the packet.
##
## If the default logging isn't desired, the
## host will have to do tcp checks in the
## main chains.
##
## The order is unimportant except that the
## syn/fin must be last because we want to
## catch all other combinations not enumerated
## earlier in the chain.

$IPTABLES -N tcp-chain
## data in syn packet; sid: 526
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN SYN -j LOG --log-prefix "tcp_chain: syn packet
data "
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN SYN -j DROP
## null scan; sid: 623
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL NONE -j LOG --log-prefix "tcp_chain: null scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL NONE -j DROP
## xmas scan; sid: 625
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL ALL -j LOG --log-prefix "tcp_chain: xmas scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL ALL -j DROP
## cybercop scan; sid: 627
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,URG -j LOG --log-prefix "tcp_chain:
cybercop scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,URG -j DROP
## nmap fingerprint scan; sid: 629
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j LOG --log-prefix "tcp_chain:
nmap scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j DROP
## syn/fin scan; sid: 624, 630
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN,FIN SYN,FIN -j LOG --log-prefix "tcp_chain:
syn/fin scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP

#### udp-chain


#### icmp-chain
```

```
########
######## Mangle
########

########
######## NAT
########

########
######## Filtering
########

###### INPUT chain ######

## The first rules applied are the overriding,
## always enforced, site policy.
##
## Then, the order is sensitive to volume,
## both by connection-state and within
## each connection-state.
##
##     1) established connections
##     2) new connections for SSH
##     3) new connections for ping

## site policy
$IPTABLES -j site-policy-chain

## check all tcp traffic
$IPTABLES -A INPUT -p tcp -j tcp-chain

## Since we have validated the IP connection,
## pass the existing connections.
$IPTABLES -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

## log and pass the new SSH connections
## the security connections
$IPTABLES -A INPUT -i $IF_Interior -s $LOCAL_SSMS -p tcp --dport 22 -m state --state NEW -j
LOG --log-prefix "VPN NEW INPUT "
$IPTABLES -A INPUT -i $IF_Interior -s $LOCAL_SSMS -p tcp --dport 22 -m state --state NEW -j
ACCEPT
$IPTABLES -A INPUT -s $IP_SEC_SUBNET -p tcp --dport 22 -m state --state NEW -j LOG --log-
prefix "VPN NEW INPUT "
$IPTABLES -A INPUT -s $IP_SEC_SUBNET -p tcp --dport 22 -m state --state NEW -j ACCEPT
## LOG and REJECT improper internal port 22 attempts
$IPTABLES -A INPUT -s $IP_10_SUBNET -p tcp --dport 22 -m state --state NEW -j LOG --log-
prefix "VPN invalid INPUT "
$IPTABLES -A INPUT -s $IP_10_SUBNET -p tcp --dport 22 -m state --state NEW -j REJECT --
reject-with tcp-reset
$IPTABLES -A INPUT -s $IP_192_168_SUBNET -p tcp --dport 22 -m state --state NEW -j LOG --
log-prefix "VPN invalid INPUT "
$IPTABLES -A INPUT -s $IP_192_168_SUBNET -p tcp --dport 22 -m state --state NEW -j
REJECT --reject-with tcp-reset
## the remainder of port 22 attempts are from the Internet-accept
$IPTABLES -A INPUT -i $IF_Exterior -p tcp --dport 22 -m state --state NEW -j LOG --log-prefix
"VPN NEW INPUT "
```

```
$IPTABLES -A INPUT -i $IF_Exterior -p tcp --dport 22 -m state --state NEW -j ACCEPT
## any Internet port 22 connects arriving at the internal interface will be DROP'd

## ping traffic
## the ping traffic is logged and DROP'd if excessive
$IPTABLES -A INPUT -m limit ! --limit 20/hour -p icmp --icmp-type 8 -j LOG --log-prefix "VPN
excessive pings "
$IPTABLES -A INPUT -m limit ! --limit 20/hour -p icmp --icmp-type 8 -j DROP
## This ping we will respond to
$IPTABLES -A INPUT -p icmp --icmp-type 8 -j ACCEPT

## The rest should be invalid and could signal an attack
$IPTABLES -A INPUT -j LOG --log-prefix "VPN invalid INPUT " --log-level crit
$IPTABLES -A INPUT -j DROP


###### OUTPUT chain ######

## The first rules applied are the overriding,
## always enforced, site policy.
##
## Then, the order is simply by volume.  There
## are few rules because there is little traffic.

## site policy
$IPTABLES -j site-policy-chain

## Top priority is the syslog dump.
$IPTABLES -A OUTPUT -o $IF_Exterior -d $LOCAL_SSMS -p udp --dport 514 -j ACCEPT

## check all tcp traffic
$IPTABLES -A OUTPUT -p tcp -j tcp-chain

## pass the existing connections
$IPTABLES -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

## allow out valid DNS traffic
$IPTABLES -A OUTPUT -o $IF_Exterior -d $LOCAL_SSMS -p udp --dport 53 -m state --state
NEW -j ACCEPT

## allow out valid NTP traffic
$IPTABLES -A OUTPUT -o $IF_Exterior -d $LOCAL_SSMS -p udp --dport 123 -m state --state
NEW -j ACCEPT

## The VPN itself is passive; syslog & NTP above are it's only
## internally initiated IP traffic.  The rest of the NEW connections
## must be users connecting with internal resources.  We'll accept
## those, but not direct connections to the Internet because then
## non-GIAC personnel would generate traffic from GIAC's network to
## outside parties.  Corporate Counsel advises against allowing GIAC
## to be used as a relay by outsiders.
##
## For readability and performance reasons, the LOG lines
## are broader in coverage than the ACCEPT lines.
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_ADMIN_SUBNET -p tcp -m state --state NEW -j
LOG --log-prefix "VPN NEW OUTPUT "
```

```
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_ADMIN_SUBNET -p tcp --dport 80 -m state --
state NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_ADMIN_SUBNET -p tcp --dport 25 -m state --
state NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_ADMIN_SUBNET -p tcp --dport 21 -m state --
state NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_ADMIN_SUBNET -p tcp --dport 22 -m state --
state NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_ADMIN_SUBNET -p tcp --dport 445 -m state --
state NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_ADMIN_SUBNET -p tcp --dport 515 -m state --
state NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_GS_SUBNET -p tcp -m state --state NEW -j LOG
--log-prefix "VPN NEW OUTPUT "
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_GS_SUBNET -p tcp --dport 80 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_GS_SUBNET -p tcp --dport 25 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_GS_SUBNET -p tcp --dport 21 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_GS_SUBNET -p tcp --dport 22 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_GS_SUBNET -p tcp --dport 445 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_GS_SUBNET -p tcp --dport 515 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_PROD_SUBNET -p tcp -m state --state NEW -j
LOG --log-prefix "VPN NEW OUTPUT "
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_PROD_SUBNET -p tcp --dport 80 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_PROD_SUBNET -p tcp --dport 25 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_PROD_SUBNET -p tcp --dport 21 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_PROD_SUBNET -p tcp --dport 22 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_PROD_SUBNET -p tcp --dport 445 -m state --
state NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_PROD_SUBNET -p tcp --dport 515 -m state --
state NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_ES_SUBNET -p tcp -m state --state NEW -j LOG
--log-prefix "VPN NEW OUTPUT "
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_ES_SUBNET -p tcp --dport 80 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_ES_SUBNET -p tcp --dport 25 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -o $IF_Interior -d $IP_ES_SUBNET -p tcp --dport 21 -m state --state
NEW -j ACCEPT

## lastly, LOG and REJECT
$IPTABLES -A OUTPUT -j LOG --log-level crit --log-prefix "VPN invalid OUTPUT "
$IPTABLES -A OUTPUT -p tcp -j REJECT --reject-with tcp-reset
$IPTABLES -A OUTPUT -j REJECT


#### FORWARD chain ######
```

Appendix E.

```
#!/bin/sh

########
######## Template NetFilter bash script
########

#### Security Subnet Management Station
#### by Greg Leisner

########
######## macros for common values
########

###### executable

IPTABLES="/sbin/iptables"

###### interfaces

IF_Clients="eth0"
IF_Clients2="eth1"
IF_SSN="eth3"
IP_Clients="192.168.0.1"
IP_Clients2="192.168.1.1"
IP_SSN="10.254.0.5"
IP_Clients_Mask="192.168.0.0/24"
IP_Clients2_Mask="192.168.1.0/24"
IP_SSN_Mask="10.254.0.0/16"

###### local subnet hosts
######
###### except for SSMS host,
###### at least a LOCAL_SSMS
###### must be defined here

########
######## Initialization
########

###### Load modules

/sbin/depmod -a

/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_state
/sbin/modprobe ipt_REJECT
/sbin/modprobe ip_conntrack_ftp
/sbin/modprobe ipt_ipv4options
/sbin/modprobe ipt_length
```

```
###### Set chain policies
######
###### This is GIAC's default security posture;
###### DROP it.  This allows the ruleset to grow
###### by simply allowing in whatever new traffic
###### that is required.  Since the new rules
###### can be added directly before the DROP
###### rule in the specific chain, there is
###### no great security concern about emergency
###### maintenance to the ruleset.  Any
###### performance concerns can be addressed
###### by a thoughtful review of the entire
###### ruleset at a later time.
######
###### must contain these lines only
###### $IPTABLES -P INPUT DROP
###### $IPTABLES -P OUTPUT DROP
###### $IPTABLES -P FORWARD DROP
######
###### $IPTABLES -t nat -P PREROUTING DROP
###### $IPTABLES -t nat -P POSTROUTING DROP
###### $IPTABLES -t nat -P OUTPUT DROP
######
###### $IPTABLES -t mangle -P PREROUTING ACCEPT
###### $IPTABLES -t mangle -P OUTPUT ACCEPT

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

$IPTABLES -t nat -P PREROUTING DROP
$IPTABLES -t nat -P POSTROUTING DROP
$IPTABLES -t nat -P OUTPUT DROP

$IPTABLES -t mangle -P PREROUTING ACCEPT
$IPTABLES -t mangle -P OUTPUT ACCEPT

###### Enable/Disable (1/0) IP Forwarding
######
###### echo "0" > /proc/sys/net/ipv4/ip_forward
echo "0" > /proc/sys/net/ipv4/ip_forward


########
######## Create User chains
########

###### site-policy-chain

## No nonsense, always applied rules.

$IPTABLES -N site-policy-chain
## IP options shouldn't happen, and if they
## do, it's a "Bad Thing".  Don't even bother
## logging them; GIAC doesn't care what the
## intent was.  If this was part of an attack,
```

```
## we'll catch them some other way.  If it
## was innocent, we don't waste time on it.
$IPTABLES -A site-policy-chain -m ipv4options --rr -j DROP
$IPTABLES -A site-policy-chain -m ipv4options --ts -j DROP

#### tcp-chain

## This chain checks the tcp header it will
## only LOG and DROP bad packets.  The main
## chain will have to ACCEPT the packet.
##
## If the default logging isn't desired, the
## host will have to do tcp checks in the
## main chains.
##
## The order is unimportant except that the
## syn/fin must be last because we want to
## catch all other combinations not enumerated
## earlier in the chain.

$IPTABLES -N tcp-chain
## data in syn packet; sid: 526
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN SYN -j LOG --log-prefix "tcp_chain: syn packet
data "
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN SYN -j DROP
## null scan; sid: 623
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL NONE -j LOG --log-prefix "tcp_chain: null scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL NONE -j DROP
## xmas scan; sid: 625
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL ALL -j LOG --log-prefix "tcp_chain: xmas scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL ALL -j DROP
## cybercop scan; sid: 627
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,URG -j LOG --log-prefix "tcp_chain:
cybercop scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,URG -j DROP
## nmap fingerprint scan; sid: 629
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j LOG --log-prefix "tcp_chain:
nmap scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j DROP
## syn/fin scan; sid: 624, 630
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN,FIN SYN,FIN -j LOG --log-prefix "tcp_chain:
syn/fin scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP

#### udp-chain


#### icmp-chain


########
######## Mangle
########

########
######## NAT
```

```
########

########
######## Filtering
########

###### INPUT chain ######

## The first rules applied are the overriding,
## always enforced, site policy.
##
## Then, the order is sensitive to volume,
## both by connection-state and within
## each connection-state.
##
##     1) established connections
##     2) new connections for SSH
##     3) new connections for ping

## site policy
$IPTABLES -j site-policy-chain

## check all tcp traffic
$IPTABLES -A INPUT -p tcp -j tcp-chain

## Since we have validated the IP connection,
## pass the existing connections.
$IPTABLES -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

## ACCEPT client syslogs, if we get flooded,
## we'll figure out who's doing it
$IPTABLES -A INPUT -i $IF_CLIENTS -s IP_Clients_Mask -p udp --dport 123 -j ACCEPT
$IPTABLES -A INPUT -i $IF_CLIENTS2 -s IP_Clients2_Mask -p udp --dport 123 -j ACCEPT

## The 191.168 networks don't do DHCP,
## but rather use static ARP, host IP's,
## network masks and routing tables.

## pass new DNS and NTP connections
$IPTABLES -A INPUT -i $IF_SSN -s $IP_SSN_Mask -p tcp --dport 53 -m state --state NEW -j
ACCEPT
$IPTABLES -A INPUT -i $IF_SSN -s $IP_SSN_Mask -p udp --dport 53 -m state --state NEW -j
ACCEPT
$IPTABLES -A INPUT -i $IF_CLIENTS -s $IP_Clients_Mask -p udp --dport 53 -m state --state
NEW -j ACCEPT
$IPTABLES -A INPUT -i $IF_CLIENTS2 -s $IP_Clients2_Mask -p udp --dport 53 -m state --state
NEW -j ACCEPT
$IPTABLES -A INPUT -i $IF_SSN -s $IP_SSN_Mask -p udp --dport 123 -m state --state NEW -j
ACCEPT
$IPTABLES -A INPUT -i $IF_CLIENTS -s $IP_Clients_Mask -p udp --dport 123 -m state --state
NEW -j ACCEPT
$IPTABLES -A INPUT -i $IF_CLIENTS2 -s $IP_Clients2_Mask -p udp --dport 123 -m state --state
NEW -j ACCEPT

## log and pass the new SSH connections
$IPTABLES -A INPUT -i $IF_SSN -s $IP_SSN_Mask -p tcp --dport 22 -m state --state NEW -j
```

```
LOG --log-prefix "SSN NEW INPUT "
$IPTABLES -A INPUT -i $IF_SSN -s $IP_SSN_Mask -p tcp --dport 22 -m state --state NEW -j
ACCEPT

## ping traffic
## the ping traffic is logged and DROP'd if excessive
$IPTABLES -A INPUT -i $IF_SSN -s $IP_SSN_Mask -m limit ! --limit 20/hour -p icmp --icmp-type
8 -j LOG --log-prefix "SSMS excessive pings "
$IPTABLES -A INPUT -i $IF_SSN -s $IP_SSN_Mask -m limit ! --limit 20/hour -p icmp --icmp-type
8 -j DROP
## This ping we will respond to
$IPTABLES -A INPUT -i $IF_SSN -s $IP_SSN_Mask -p icmp --icmp-type 8 -j ACCEPT

## The rest should be invalid and could signal an attack
$IPTABLES -A INPUT -j LOG --log-prefix "SSMS invalid INPUT " --log-level crit
$IPTABLES -A INPUT -j DROP


###### OUTPUT chain ######

## The first rules applied are the overriding,
## always enforced, site policy.
##
## Then, the order is simply by volume.  There
## are few rules because there is little traffic.

## site policy
$IPTABLES -j site-policy-chain

## check all tcp traffic
$IPTABLES -A OUTPUT -p tcp -j tcp-chain

## pass the existing connections
$IPTABLES -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

## Allow access to clients; the user will
## be logged in to this host via the SSN,
## or this will be a scripted connection.
$IPTABLES -A OUTPUT -i $IF_CLIENTS -s $IP_Clients_Mask -p tcp --dport 22 -m state --state
NEW -j LOG --log-prefix "SSN NEW OUTPUT "
$IPTABLES -A OUTPUT -i $IF_CLIENTS -s $IP_Clients_Mask -p tcp --dport 22 -m state --state
NEW -j ACCEPT
$IPTABLES -A OUTPUT -i $IF_CLIENTS2 -s $IP_Clients2_Mask -p tcp --dport 22 -m state --
state NEW -j LOG --log-prefix "SSN NEW OUTPUT "
$IPTABLES -A OUTPUT -i $IF_CLIENTS2 -s $IP_Clients2_Mask -p tcp --dport 22 -m state --
state NEW -j ACCEPT

## lastly, LOG and DROP
$IPTABLES -A OUTPUT -j LOG --log-prefix "SSMS invalid OUTPUT " --log-level crit
$IPTABLES -A OUTPUT -p tcp -j REJECT --reject-with tcp-reset
$IPTABLES -A OUTPUT -j REJECT


#### FORWARD chain ######
```

Appendix F.

```
#!/bin/sh

########
######## Template NetFilter bash script
########

#### End User Workstation
#### by Greg Leisner

########
######## macros for common values
########

###### exexecutable

IPTABLES="/sbin/iptables"

###### interfaces

IF="eth0"
IP= <supplied by a helper function in the real script>
IP_MASK= <supplied by a helper function in the real script>

###### local subnet hosts
######
###### except for SSMS host,
###### at least a LOCAL_SSMS
###### must be defined here
LOCAL_SSMS= <supplied by a helper function in the real script; all SSMS's are IP=(network +
1)>

########
######## Initialization
########

###### Load modules

/sbin/depmod -a

/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_state
/sbin/modprobe ipt_REJECT
/sbin/modprobe ip_conntrack_ftp
/sbin/modprobe ipt_ipv4options
/sbin/modprobe ipt_length

###### Set chain policies
######
###### This is GIAC's default security posture;
###### DROP it.  This allows the ruleset to grow
```

```
###### by simply allowing in whatever new traffic
###### that is required.  Since the new rules
###### can be added directly before the DROP
###### rule in the specific chain, there is
###### no great security concern about emergency
###### maintenance to the ruleset.  Any
###### performance concerns can be addressed
###### by a thoughtful review of the entire
###### ruleset at a later time.
######
###### must contain these lines only
###### $IPTABLES -P INPUT DROP
###### $IPTABLES -P OUTPUT DROP
###### $IPTABLES -P FORWARD DROP
######
###### $IPTABLES -t nat -P PREROUTING DROP
###### $IPTABLES -t nat -P POSTROUTING DROP
###### $IPTABLES -t nat -P OUTPUT DROP
######
###### $IPTABLES -t mangle -P PREROUTING ACCEPT
###### $IPTABLES -t mangle -P OUTPUT ACCEPT

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

$IPTABLES -t nat -P PREROUTING DROP
$IPTABLES -t nat -P POSTROUTING DROP
$IPTABLES -t nat -P OUTPUT DROP

$IPTABLES -t mangle -P PREROUTING ACCEPT
$IPTABLES -t mangle -P OUTPUT ACCEPT

###### Enable/Disable (1/0) IP Forwarding
######
###### echo "0" > /proc/sys/net/ipv4/ip_forward
echo "0" > /proc/sys/net/ipv4/ip_forward


########
######## Create User chains
########

###### site-policy-chain

## No nonsense, always applied rules.

$IPTABLES -N site-policy-chain
## IP options shouldn't happen, and if they
## do, it's a "Bad Thing".  Don't even bother
## logging them; GIAC doesn't care what the
## intent was.  If this was part of an attack,
## we'll catch them some other way.  If it
## was innocent, we don't waste time on it.
$IPTABLES -A site-policy-chain -m ipv4options --rr -j DROP
$IPTABLES -A site-policy-chain -m ipv4options --ts -j DROP
```

```
#### tcp-chain

## This chain checks the tcp header it will
## only LOG and DROP bad packets.  The main
## chain will have to ACCEPT the packet.
##
## If the default logging isn't desired, the
## host will have to do tcp checks in the
## main chains.
##
## The order is unimportant except that the
## syn/fin must be last because we want to
## catch all other combinations not enumerated
## earlier in the chain.

$IPTABLES -N tcp-chain
## data in syn packet; sid: 526
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN SYN -j LOG --log-prefix "tcp_chain: syn packet
data "
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN SYN -j DROP
## null scan; sid: 623
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL NONE -j LOG --log-prefix "tcp_chain: null scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL NONE -j DROP
## xmas scan; sid: 625
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL ALL -j LOG --log-prefix "tcp_chain: xmas scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL ALL -j DROP
## cybercop scan; sid: 627
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,URG -j LOG --log-prefix "tcp_chain:
cybercop scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,URG -j DROP
## nmap fingerprint scan; sid: 629
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j LOG --log-prefix "tcp_chain:
nmap scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j DROP
## syn/fin scan; sid: 624, 630
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN,FIN SYN,FIN -j LOG --log-prefix "tcp_chain:
syn/fin scan "
$IPTABLES -A tcp-chain -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP

#### udp-chain


#### icmp-chain


########
######## Mangle
########

########
######## NAT
########

########
######## Filtering
```

```
########

###### INPUT chain ######

## The first rules applied are the overriding,
## always enforced, site policy.
##
## Then, the order is sensitive to volume,
## both by connection-state and within
## each connection-state.
##
##     1) established connections
##     2) new connections for SSH
##     3) new connections for ping

## site policy
$IPTABLES -j site-policy-chain

## check all tcp traffic
$IPTABLES -A INPUT -p tcp -j tcp-chain

## Since we have validated the IP connection,
## pass the existing connections.
$IPTABLES -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

## log and pass the new SSH connections
## the security connections
$IPTABLES -A INPUT -i $IF -s $LOCAL_SSMS -p tcp --dport 22 -m state --state NEW -j LOG --
log-prefix "user NEW INPUT "
$IPTABLES -A INPUT -i $IF -s $LOCAL_SSMS -p tcp --dport 22 -m state --state NEW -j
ACCEPT
$IPTABLES -A INPUT -i $IF -s $IP_SEC_SUBNET -p tcp --dport 22 -m state --state NEW -j LOG
--log-prefix "user NEW INPUT "
$IPTABLES -A INPUT -i $IF -s $IP_SEC_SUBNET -p tcp --dport 22 -m state --state NEW -j
ACCEPT

## ping traffic
## the ping traffic is logged and DROP'd if excessive
$IPTABLES -A INPUT -i $IF -m limit ! --limit 20/hour -p icmp --icmp-type 8 -j LOG --log-prefix
"user excessive pings "
$IPTABLES -A INPUT -i $IF -m limit ! --limit 20/hour -p icmp --icmp-type 8 -j DROP
## This ping we will respond to
$IPTABLES -A INPUT -i $IF -p icmp --icmp-type 8 -j ACCEPT

## Lastly, LOG and REJECT the rest.
## Because all the rules have -i $IF, this script
## will not work for multi-homed hosts.
$IPTABLES -A INPUT -j LOG --log-prefix "user invalid INPUT "
$IPTABLES -A INPUT -p tcp -j REJECT --reject-with tcp-reset
$IPTABLES -A INPUT -j REJECT


###### OUTPUT chain ######

## The first rules applied are the overriding,
## always enforced, site policy.
```

```
##
## Then, the order is simply by volume.  There
## are few rules because there is little traffic.

## site policy
$IPTABLES -j site-policy-chain

## Top priority is the syslog dump.
$IPTABLES -A OUTPUT -o $IF -d $LOCAL_SSMS -p udp --dport 514 -j ACCEPT

## check all tcp traffic
$IPTABLES -A OUTPUT -o $IF -p tcp -j tcp-chain

## pass the existing connections
$IPTABLES -A OUTPUT -o $IF -m state --state ESTABLISHED,RELATED -j ACCEPT

## allow out valid DNS traffic
$IPTABLES -A OUTPUT -o $IF -d $LOCAL_SSMS -p udp --dport 53 -j ACCEPT

## allow out valid NTP traffic
$IPTABLES -A OUTPUT -o $IF -d $LOCAL_SSMS -p udp --dport 123 -j ACCEPT

## allow out valid DHCP traffic
$IPTABLES -A OUTPUT -o $IF -d $LOCAL_SSMS -p udp --dport 67 -j ACCEPT

## The host itself is passive; syslog & NTP above are it's only
## internally initiated IP traffic.  The rest of the NEW connections
## must be users connecting with other hosts.
##
## We don't test --dport or -d because we don't
## really know who the user is and what their
## permited activities are.
##
## We will LOG each users' NEW connections
## for analysis and enforcement reasons.
$IPTABLES -A OUTPUT -o $IF -m state --state NEW -j LOG --log-prefix "user NEW OUTPUT "
$IPTABLES -A OUTPUT -o $IF -m state --state NEW -j ACCEPT

## Lastly, LOG and REJECT the rest.
## Because all the rules have -o $IF, this script
## will not work for multi-homed hosts.
$IPTABLES -A OUTPUT -j LOG --log-prefix "user invalid OUTPUT "
$IPTABLES -A OUTPUT -p tcp -j REJECT --reject-with tcp-reset
$IPTABLES -A OUTPUT -j REJECT


#### FORWARD chain ######
```

Appendix G.

```
#
# GIAC static networks (192.168.x.x) arp (/etc/ethers)
# and hostname mappings (/etc/hosts)
#
# During boot, the arp entries are registered with:
#
#          arp -f /etc/ethers
#
# The host name mappings will be automatically found.


# 192.168.0.0/16 Exterior Gateway subnet

# /etc/ethers

192.168.0.1     00:15:02:00:17:01
192.168.0.2     00:15:02:00:17:02
192.168.0.3     00:15:02:00:17:03
192.168.0.4     00:15:02:00:17:04

# /etc/hosts

127.0.0.1       localhost.localdomain   localhost
192.168.0.1     alpha.egs.giac.com      alpha
192.168.0.2     beta.egs.giac.com       beta
192.168.0.3     gamma.egs.giac.com      gamma
192.168.0.4     delta.egs.giac.com      delta

# 192.168.1.0/16          External Services subnet

# /etc/ethers

192.168.1.1     00:15:02:00:19:01
192.168.1.2     00:15:02:00:19:02
192.168.1.3     00:15:02:00:19:03
192.168.1.4     00:15:02:00:19:04
192.168.1.5     00:15:02:00:19:05

# /etc/hosts

127.0.0.1       localhost.localdomain   localhost
192.168.1.1     epsilon.ess.giac.com    epsilon
192.168.1.2     zeta.ess.giac.com       zeta
192.168.1.3     eta.ess.giac.com        eta
192.168.1.4     theta.ess.giac.com      theta
192.168.1.5     iota.ess.giac.com       iota

# 192.168.2.0/16          VPN Gateway connection

# /etc/ethers

192.168.2.2     00:15:02:00:15:02
192.168.2.3     00:15:02:00:15:03
```

```
# /etc/hosts

127.0.0.1       localhost.localdomain   localhost
192.168.2.2     kappa.vgs.giac.com      kappa
192.168.2.3     lambda.vgs.giac.com     lambda

# 192.168.3.0/16        Non-VPN Gateway connection

# /etc/ethers

192.168.3.2     00:15:02:00:15:02
192.168.3.3     00:15:02:00:15:03

# /etc/hosts

127.0.0.1       localhost.localdomain   localhost
192.168.3.2     mu.nvgs.giac.com        mu
192.168.3.3     nu.nvgs.giac.com        nu
```

Appendix H.

```
### GIAC Enterprises
### Redhat Kickstart-base platform
### copyright 2003
###

### what we are doing
text
install
cdrom
#nfs --server 192.168.1.1 --dir /usr/kickstart/

### setup the harddrive
zerombr yes
clearpart --all
part /  --onprimary=1 --badblocks --fstype ext3 --size 1000
part swap --onprimary=2 --badblocks --size 256

### generic platform build
auth --enablemd5 --enableshadow
### Linux only platform build
#auth --enablemd5 --enableshadow  --enableldap --ldapserver= --ldapbasedn= --enableldaptls --
enablecache
### windows cooperative platform build
#auth --enablemd5 --enableshadow --enablesmbauth --smbservers= --smbworkgroups= --
enableldap --ldapserver= --ldapbasedn= --enableldaptls --enablecache

### user interface stuff
keyboard us
lang en_US
langsupport --default en_US
timezone --utc America/Chicago

### for non-desktops
skipx
### for desktops
#xconfig

### misc configuration settings
firewall --high --ssh --dhcp
mouse --emulthree
network --bootproto dhcp

### the following passwords must be made unique by host
### scale for password              1234567890
bootloader --location=mbr --password=Pf7v#@63RD
rootpw                          KR@1am59A$

### finally
reboot

%packages
@ Network Support
-rwho
-micq
```

```
-MAKEDEV
-lokkit
-ipchains
-pam_krb5
-telnet
-radvd
-yp-tools
-apmd
-krbafs
-finger
-up2date
-rhn_register
-lilo
-ypbind
-rsh
-ash
-wget
-rdate
-sendmail
-ftp
-sendmail-cf
-talk
-whois
-rusers
-nfs-utils
-raidtools
-reiserfs-utils
-parted
-dhcpcd
-ncftp
-mkbootdisk
-dosfstools
-syslinux
-mailx
-procmail
-rmt
-python-xmlrpc
-m4
-cpio
-authconfig
-ed
-mailcap
-netconfig
-nmap
-ntsysv
-pcre
-time
-utempter
###
### deducts only for generic
###
-openldap
-openldap-clients


libcap
```

```
ntp
openssl096
pump
stunnel
sysreport
traceroute
tripwire
xinetd


%post
```

Appendix I.

GIAC physical security standards

The explicit promise of this project is to deliver a secure host. No operating system of any vendor can promise to do that without the underlying hardware being secured. From NIST 800-7:

All software security depends on hardware security. If the hardware can be stolen or surreptitiously replaced, secure software will not help. When computers filled a room, stolen computers were not a big problem. Now that laptop and palmtop computers are the fastest growing market, physical security is at least as important as software security.
Some of the most common problems are:

- equipment and removable media is stolen or replaced;
- security can be circumvented by changing hardware setup parameters;
- systems can be booted by unauthorized users;
- systems can be booted from unauthorized software;
- boot media can be re-written by unauthorized software, and
- unauthorized software can be executed from removable media.
- Some of the safeguards which can be taken are:
- locked doors and secured equipment;
- lockable cases, keyboards, and removable media drives;
- key or password-protected configuration and setup;
- password required to boot;
- password required to mount removable media;
- read-only boot media, and
- storing removable media in secured areas.

The steps GIAC requires taken to avoid physical circumvention of software security are:

1) never purchase hardware with the first harddrive swappable unless the host is to be deployed in a locked and sealed enclosure.
2) all host firmware (BIOS) be configured to boot only from the first harddrive; don't let hosts be booted from floppy or cdrom.
3) all host firmware (BIOS) be password protected by random computer generated codes and those codes be physically secured by Computer Operations; the BIOS ought to be hard to change!
4) all hosts have locked cases or are in locked enclosures with Computer Operations physically securing the keys; keep the bad guys out.
5) all host cases and enclosures be sealed with a GIAC custom, tamperproof, serialized label that will self-destruct if broken; detect covert intrusion.
6) require supervisory personnel to verify integrity of seals (5) periodically.
7) the physical media (i.e. cable) and switching nodes must also be secured, but those issues are not addressed here.

Appendix J.

GIAC approved automated scanning tools

Nmap
www.nmap.org

nessus
www.nessus.org

Appendix K.

```
##
## trimmed and annotated tcpdump capture file
##

##
## the following is the nmap section trimmed of all the actual nmap packets except for
## the ones that triggered DNAT rules and the resulting packet with a new destination
## host.
##
## The nmap output file was simply:
##
# nmap (V. 3.00) scan initiated Sat Feb  1 15:16:01 2003 as: nmap -e eth0:6 -S
# 102.17.94.36 -sS -P0 -F -v -n -oN nmap_extest.txt 102.17.94.35
# All 1150 scanned ports on  (102.17.94.35) are: filtered
#
# Nmap run completed at Sat Feb  1 15:39:05 2003 -- 1 IP address (1 host up)
# scanned in 1384 seconds
##
## The NetFilter log file has corresponding entries for each of the scans that were
## DNAT'd.
##

15:20:00.230310 somebody.nowhere.com.63218 > 102.17.94.35.ssh: S [tcp
sum ok] 763580494:763580494(0) win 4096

(ttl 51, id 58343, len 40)
15:20:00.231332 somebody.nowhere.com.63218 > gamma.egs.giac.com.ssh: S
[tcp sum ok] 763580494:763580494(0) win

4096 (ttl 50, id 58343, len 40)
15:20:06.251861 somebody.nowhere.com.63219 > 102.17.94.35.ssh: S [tcp
sum ok] 2513326235:2513326235(0) win 4096

(ttl 51, id 37929, len 40)
15:20:06.252789 somebody.nowhere.com.63219 > gamma.egs.giac.com.ssh: S
[tcp sum ok] 2513326235:2513326235(0) win

4096 (ttl 50, id 37929, len 40)
15:25:01.314433 somebody.nowhere.com.63218 > 102.17.94.35.http: S [tcp
sum ok] 763580494:763580494(0) win 4096

(ttl 51, id 22813, len 40)
15:25:01.315642 somebody.nowhere.com.63218 > zeta.ess.giac.com.http: S
[tcp sum ok] 763580494:763580494(0) win

4096 (ttl 50, id 22813, len 40)
15:25:07.334341 somebody.nowhere.com.63219 > 102.17.94.35.http: S [tcp
sum ok] 2513326235:2513326235(0) win 4096

(ttl 51, id 37509, len 40)
15:25:07.335413 somebody.nowhere.com.63219 > zeta.ess.giac.com.http: S
[tcp sum ok] 2513326235:2513326235(0) win

4096 (ttl 50, id 37509, len 40)
```

```
15:25:25.399540 somebody.nowhere.com.63218 > 102.17.94.35.smtp: S [tcp
sum ok] 763580494:763580494(0) win 4096

(ttl 51, id 52077, len 40)
15:25:25.400875 somebody.nowhere.com.63218 > eta.ess.giac.com.smtp: S
[tcp sum ok] 763580494:763580494(0) win

4096 (ttl 50, id 52077, len 40)
15:25:31.421261 somebody.nowhere.com.63219 > 102.17.94.35.smtp: S [tcp
sum ok] 2513326235:2513326235(0) win 4096

(ttl 51, id 47560, len 40)
15:25:31.422446 somebody.nowhere.com.63219 > eta.ess.giac.com.smtp: S
[tcp sum ok] 2513326235:2513326235(0) win

4096 (ttl 50, id 47560, len 40)
15:25:55.509606 somebody.nowhere.com.63218 > 102.17.94.35.ftp: S [tcp
sum ok] 763580494:763580494(0) win 4096

(ttl 51, id 38581, len 40)
15:25:55.511137 somebody.nowhere.com.63218 > theta.ess.giac.com.ftp: S
[tcp sum ok] 763580494:763580494(0) win

4096 (ttl 50, id 38581, len 40)
15:26:01.529174 somebody.nowhere.com.63219 > 102.17.94.35.ftp: S [tcp
sum ok] 2513326235:2513326235(0) win 4096

(ttl 51, id 8616, len 40)
15:26:01.530193 somebody.nowhere.com.63219 > theta.ess.giac.com.ftp: S
[tcp sum ok] 2513326235:2513326235(0) win

4096 (ttl 50, id 8616, len 40)
15:33:27.111606 somebody.nowhere.com.63221 > 102.17.94.35.ftp: S [tcp
sum ok] 763580494:763580494(0) win 4096

(ttl 51, id 38260, len 40)
15:33:27.112583 somebody.nowhere.com.63221 > theta.ess.giac.com.ftp: S
[tcp sum ok] 763580494:763580494(0) win

4096 (ttl 50, id 38260, len 40)
15:33:33.134567 somebody.nowhere.com.63222 > 102.17.94.35.ftp: S [tcp
sum ok] 2513326235:2513326235(0) win 4096

(ttl 51, id 59350, len 40)
15:33:33.135477 somebody.nowhere.com.63222 > theta.ess.giac.com.ftp: S
[tcp sum ok] 2513326235:2513326235(0) win

4096 (ttl 50, id 59350, len 40)
15:33:57.219927 somebody.nowhere.com.63221 > 102.17.94.35.smtp: S [tcp
sum ok] 763580494:763580494(0) win 4096

(ttl 51, id 60741, len 40)
15:33:57.221303 somebody.nowhere.com.63221 > eta.ess.giac.com.smtp: S
[tcp sum ok] 763580494:763580494(0) win

4096 (ttl 50, id 60741, len 40)
15:34:03.241173 somebody.nowhere.com.63222 > 102.17.94.35.smtp: S [tcp
```

```
sum ok] 2513326235:2513326235(0) win 4096

(ttl 51, id 11312, len 40)
15:34:03.242177 somebody.nowhere.com.63222 > eta.ess.giac.com.smtp: S
[tcp sum ok] 2513326235:2513326235(0) win

4096 (ttl 50, id 11312, len 40)
15:34:27.329744 somebody.nowhere.com.63221 > 102.17.94.35.http: S [tcp
sum ok] 763580494:763580494(0) win 4096

(ttl 51, id 62906, len 40)
15:34:27.331058 somebody.nowhere.com.63221 > zeta.ess.giac.com.http: S
[tcp sum ok] 763580494:763580494(0) win

4096 (ttl 50, id 62906, len 40)
15:34:33.349641 somebody.nowhere.com.63222 > 102.17.94.35.http: S [tcp
sum ok] 2513326235:2513326235(0) win 4096

(ttl 51, id 65210, len 40)
15:34:33.350836 somebody.nowhere.com.63222 > zeta.ess.giac.com.http: S
[tcp sum ok] 2513326235:2513326235(0) win

4096 (ttl 50, id 65210, len 40)
15:38:16.150829 somebody.nowhere.com.63221 > 102.17.94.35.ssh: S [tcp
sum ok] 763580494:763580494(0) win 4096

(ttl 51, id 22977, len 40)
15:38:16.152359 somebody.nowhere.com.63221 > gamma.egs.giac.com.ssh: S
[tcp sum ok] 763580494:763580494(0) win

4096 (ttl 50, id 22977, len 40)
15:38:22.174030 somebody.nowhere.com.63222 > 102.17.94.35.ssh: S [tcp
sum ok] 2513326235:2513326235(0) win 4096

(ttl 51, id 29272, len 40)
15:38:22.175366 somebody.nowhere.com.63222 > gamma.egs.giac.com.ssh: S
[tcp sum ok] 2513326235:2513326235(0) win

4096 (ttl 50, id 29272, len 40)
```

##
The ping test of the exterior interface.
##
Oops! This is wrong. The first 5 pings were unanswered, then the remaining
ones were all responded to. The source of the problem is obvious. While the
LOG rule must negate '!' the limit match to get those not corresponding, the
ACCEPT rule ought to do exactly the opposite. Dropping the '!' in the ACCEPT
rule will fix this.
##
The script command captured the following from the test host:
##
Script started on Sat Feb 1 16:16:14 2003
#
#
[root@localhost root]# ping -c 30 -I eth0:6 102.17.94.35

```
# PING 102.17.94.35 (102.17.94.35) from 102.17.94.36 eth0:6: 56(84) bytes of data.
# 64 bytes from 102.17.94.35: icmp_seq=6 ttl=64 time=0.506 ms
# 64 bytes from 102.17.94.35: icmp_seq=7 ttl=64 time=0.321 ms
# 64 bytes from 102.17.94.35: icmp_seq=8 ttl=64 time=0.322 ms
# 64 bytes from 102.17.94.35: icmp_seq=9 ttl=64 time=0.320 ms
# 64 bytes from 102.17.94.35: icmp_seq=10 ttl=64 time=0.337 ms
# 64 bytes from 102.17.94.35: icmp_seq=11 ttl=64 time=0.318 ms
# 64 bytes from 102.17.94.35: icmp_seq=12 ttl=64 time=0.307 ms
# 64 bytes from 102.17.94.35: icmp_seq=13 ttl=64 time=0.324 ms
# 64 bytes from 102.17.94.35: icmp_seq=14 ttl=64 time=0.327 ms
# 64 bytes from 102.17.94.35: icmp_seq=15 ttl=64 time=0.330 ms
# 64 bytes from 102.17.94.35: icmp_seq=16 ttl=64 time=0.311 ms
# 64 bytes from 102.17.94.35: icmp_seq=17 ttl=64 time=0.306 ms
# 64 bytes from 102.17.94.35: icmp_seq=18 ttl=64 time=0.319 ms
# 64 bytes from 102.17.94.35: icmp_seq=19 ttl=64 time=0.317 ms
# 64 bytes from 102.17.94.35: icmp_seq=20 ttl=64 time=0.334 ms
# 64 bytes from 102.17.94.35: icmp_seq=21 ttl=64 time=0.318 ms
# 64 bytes from 102.17.94.35: icmp_seq=22 ttl=64 time=0.310 ms
# 64 bytes from 102.17.94.35: icmp_seq=23 ttl=64 time=0.316 ms
# 64 bytes from 102.17.94.35: icmp_seq=24 ttl=64 time=0.318 ms
# 64 bytes from 102.17.94.35: icmp_seq=25 ttl=64 time=0.325 ms
# 64 bytes from 102.17.94.35: icmp_seq=26 ttl=64 time=0.316 ms
# 64 bytes from 102.17.94.35: icmp_seq=27 ttl=64 time=0.308 ms
# 64 bytes from 102.17.94.35: icmp_seq=28 ttl=64 time=0.342 ms
# 64 bytes from 102.17.94.35: icmp_seq=29 ttl=64 time=0.314 ms
# 64 bytes from 102.17.94.35: icmp_seq=30 ttl=64 time=0.333 ms
#
# --- 102.17.94.35 ping statistics ---
# 30 packets transmitted, 25 received, 16% loss, time 29013ms
# rtt min/avg/max/mdev = 0.306/0.327/0.506/0.045 ms
#
#
# Script done on Sat Feb  1 16:17:58 2003
##

16:18:00.060440 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:01.073939 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:02.074236 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:03.074490 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:04.074773 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:05.075057 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:05.075491 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27615, len 84)
16:18:06.075323 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:06.075570 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27616, len 84)
16:18:07.075594 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:07.075846 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
```

```
(ttl 64, id 27617, len 84)
16:18:08.075878 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:08.076128 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27618, len 84)
16:18:09.076146 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:09.076409 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27619, len 84)
16:18:10.076419 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:10.076666 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27620, len 84)
16:18:11.076698 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:11.076934 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27621, len 84)
16:18:12.076964 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:12.077215 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27622, len 84)
16:18:13.077231 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:13.077491 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27623, len 84)
16:18:14.077523 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:14.077781 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27624, len 84)
16:18:15.077786 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:15.078028 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27625, len 84)
16:18:16.078061 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:16.078296 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27626, len 84)
16:18:17.078345 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:17.078591 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27627, len 84)
16:18:18.078611 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:18.078855 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27628, len 84)
16:18:19.078887 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:19.079147 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27629, len 84)
16:18:20.079166 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:20.079417 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27630, len 84)
16:18:21.079430 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:21.079669 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27631, len 84)
```

```
16:18:22.079706 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:22.079950 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27632, len 84)
16:18:23.079990 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:23.080235 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27633, len 84)
16:18:24.080246 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:24.080504 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27634, len 84)
16:18:25.080529 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:25.080772 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27635, len 84)
16:18:26.080811 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:26.081048 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27636, len 84)
16:18:27.081076 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:27.081339 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27637, len 84)
16:18:28.081351 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:28.081592 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27638, len 84)
16:18:29.081637 somebody.nowhere.com > 102.17.94.35: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:18:29.081894 102.17.94.35 > somebody.nowhere.com: icmp: echo reply
(ttl 64, id 27639, len 84)


##
## Selective scans using invalid tcp flags.  Each test first tried the DNAT'd port 22 and
## then a nonDNAT'd port (55555).  As these are correctly unanswered by the Border Router.
##
## The output from nmap was of two kinds: ACK scans and all others.
##
# nmap (V. 3.00) scan initiated Sat Feb  1 16:29:44 2003 as: nmap -e eth0:6 -S
# 102.17.94.36 -sA -P0 -p55555 -v -n -oN nmap_extest_sA.txt 102.17.94.35
# Interesting ports on  (102.17.94.35):
# Port       State      Service
# 55555/tcp  filtered    unknown
#
# Nmap run completed at Sat Feb  1 16:30:20 2003 -- 1 IP address (1 host up)
# scanned in 36 seconds
##
# nmap (V. 3.00) scan initiated Sat Feb  1 16:26:41 2003 as: nmap -e eth0:6 -S
# 102.17.94.36 -sN -P0 -p55555 -v -n -oN nmap_extest_sF.txt 102.17.94.35
# Interesting ports on  (102.17.94.35):
# Port       State       Service
# 55555/tcp  open        unknown
#
# Nmap run completed at Sat Feb  1 16:26:53 2003 -- 1 IP address (1 host up) scanned in 12
```

seconds
##

```
16:22:29.736534 somebody.nowhere.com.48721 > 102.17.94.35.ssh: F [tcp
sum ok] 0:0(0) win 3072 (ttl 54, id 51575,

len 40)
16:22:35.740904 somebody.nowhere.com.48722 > 102.17.94.35.ssh: F [tcp
sum ok] 0:0(0) win 3072 (ttl 54, id 26608,

len 40)
16:23:53.637199 somebody.nowhere.com.36535 > 102.17.94.35.55555: F [tcp
sum ok] 0:0(0) win 4096 (ttl 47, id

36383, len 40)
16:23:59.644010 somebody.nowhere.com.36536 > 102.17.94.35.55555: F [tcp
sum ok] 0:0(0) win 4096 (ttl 47, id

51974, len 40)
16:25:36.980393 somebody.nowhere.com.61760 > 102.17.94.35.ssh: FP [tcp
sum ok] 0:0(0) win 2048 urg 0 (ttl 37, id

5051, len 40)
16:25:42.992212 somebody.nowhere.com.61761 > 102.17.94.35.ssh: FP [tcp
sum ok] 0:0(0) win 2048 urg 0 (ttl 37, id

39242, len 40)
16:26:03.285591 somebody.nowhere.com.35006 > 102.17.94.35.55555: FP
[tcp sum ok] 0:0(0) win 4096 urg 0 (ttl 59,

id 61981, len 40)
16:26:09.287724 somebody.nowhere.com.35007 > 102.17.94.35.55555: FP
[tcp sum ok] 0:0(0) win 4096 urg 0 (ttl 59,

id 20816, len 40)
16:26:34.317996 somebody.nowhere.com.39943 > 102.17.94.35.ssh: . [tcp
sum ok] win 1024 (ttl 56, id 57786, len

40)
16:26:40.326292 somebody.nowhere.com.39944 > 102.17.94.35.ssh: . [tcp
sum ok] win 1024 (ttl 56, id 16601, len

40)
16:27:16.699809 somebody.nowhere.com.46275 > 102.17.94.35.55555: . [tcp
sum ok] win 2048 (ttl 49, id 64896, len

40)
16:27:22.708051 somebody.nowhere.com.46276 > 102.17.94.35.55555: . [tcp
sum ok] win 2048 (ttl 49, id 53321, len

40)
16:29:19.801485 somebody.nowhere.com.62643 > 102.17.94.35.ssh: . [tcp
sum ok] ack 3090964817 win 4096 (ttl 51,

id 28633, len 40)
16:29:25.821589 somebody.nowhere.com.62644 > 102.17.94.35.ssh: . [tcp
sum ok] ack 3090964817 win 4096 (ttl 51,
```

```
id 51209, len 40)
16:29:31.843266 somebody.nowhere.com.62645 > 102.17.94.35.ssh: . [tcp
sum ok] ack 3090964817 win 4096 (ttl 51,

id 7094, len 40)
16:29:37.864901 somebody.nowhere.com.62646 > 102.17.94.35.ssh: . [tcp
sum ok] ack 3090964817 win 4096 (ttl 51,

id 18098, len 40)
16:29:43.886572 somebody.nowhere.com.62647 > 102.17.94.35.ssh: . [tcp
sum ok] ack 3090964817 win 4096 (ttl 51,

id 14237, len 40)
16:29:49.888359 somebody.nowhere.com.62648 > 102.17.94.35.ssh: . [tcp
sum ok] ack 3090964817 win 4096 (ttl 51,

id 10075, len 40)
16:30:19.916278 somebody.nowhere.com.54653 > 102.17.94.35.55555: . [tcp
sum ok] ack 3324693154 win 4096 (ttl 51,

id 18389, len 40)
16:30:25.928117 somebody.nowhere.com.54654 > 102.17.94.35.55555: . [tcp
sum ok] ack 3324693154 win 4096 (ttl 51,

id 34920, len 40)
16:30:31.949717 somebody.nowhere.com.54655 > 102.17.94.35.55555: . [tcp
sum ok] ack 3324693154 win 4096 (ttl 51,

id 42551, len 40)
16:30:37.973069 somebody.nowhere.com.54656 > 102.17.94.35.55555: . [tcp
sum ok] ack 3324693154 win 4096 (ttl 51,

id 33858, len 40)
16:30:43.993068 somebody.nowhere.com.54657 > 102.17.94.35.55555: . [tcp
sum ok] ack 3324693154 win 4096 (ttl 51,

id 47142, len 40)
16:30:50.014670 somebody.nowhere.com.54658 > 102.17.94.35.55555: . [tcp
sum ok] ack 3324693154 win 4096 (ttl 51,

id 41275, len 40)


##
## This is an interior ping of the firewall's interior interface.  It correctly
## elicits no response.  The NetFilter log records all the packets.
##
## The script command captured the following from the test host:
##
# Script started on Sat Feb  1 16:32:53 2003
#
#
# [root@localhost root]# ping -c 30 -I eth0:5 192.168.0.2
# PING 192.168.0.2 (192.168.0.2) from 192.168.0.3 eth0:5: 56(84) bytes of data.
#
# --- 192.168.0.2 ping statistics ---
```

```
# 30 packets transmitted, 0 received, 100% loss, time 29574ms
#
#
# Script done on Sat Feb  1 16:42:49 2003
##

16:39:51.450334 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:39:52.464976 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:39:53.485313 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:39:54.505513 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:39:55.525866 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:39:56.546072 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:39:57.566429 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:39:58.586631 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:39:59.606986 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:00.627209 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:01.647882 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:02.667764 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:03.688006 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:04.708512 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:05.728664 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:06.749038 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:07.769363 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:08.789466 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:09.809780 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:10.830016 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:11.850347 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:12.870588 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:13.890895 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:14.911144 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:15.931458 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:16.952067 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
```

```
(DF) (ttl 64, id 0, len 84)
16:40:17.972024 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:18.992278 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:20.012477 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
16:40:21.032732 gamma.egs.giac.com > 192.168.0.2: icmp: echo request
(DF) (ttl 64, id 0, len 84)
```

Appendix L.

NetFilter log

```
Feb  1 15:16:18 leisuxII kernel: eth1: Promiscuous mode enabled.
Feb  1 15:16:18 leisuxII kernel: device eth1 entered promiscuous mode
Feb  1 15:20:00 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.0.3 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=58343 PROTO=TCP SPT=63218 DPT=22 SEQ=763580494 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:20:06 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.0.3 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=37929 PROTO=TCP SPT=63219 DPT=22 SEQ=2513326235 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:25:01 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.2 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=22813 PROTO=TCP SPT=63218 DPT=80 SEQ=763580494 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:25:07 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.2 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=37509 PROTO=TCP SPT=63219 DPT=80 SEQ=2513326235 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:25:25 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.3 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=52077 PROTO=TCP SPT=63218 DPT=25 SEQ=763580494 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:25:31 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.3 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=47560 PROTO=TCP SPT=63219 DPT=25 SEQ=2513326235 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:25:55 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.4 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=38581 PROTO=TCP SPT=63218 DPT=21 SEQ=763580494 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:26:01 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.4 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=8616 PROTO=TCP SPT=63219 DPT=21 SEQ=2513326235 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:33:27 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.4 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=38260 PROTO=TCP SPT=63221 DPT=21 SEQ=763580494 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:33:33 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.4 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=59350 PROTO=TCP SPT=63222 DPT=21 SEQ=2513326235 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:33:57 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.3 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=60741 PROTO=TCP SPT=63221 DPT=25 SEQ=763580494 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:34:03 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.3 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=11312 PROTO=TCP SPT=63222 DPT=25 SEQ=2513326235 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:34:27 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.2 LEN=40 TOS=0x00 PREC=0x00 TTL=50
```

```
ID=62906 PROTO=TCP SPT=63221 DPT=80 SEQ=763580494 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:34:33 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.1.2 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=65210 PROTO=TCP SPT=63222 DPT=80 SEQ=2513326235 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:38:16 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.0.3 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=22977 PROTO=TCP SPT=63221 DPT=22 SEQ=763580494 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 15:38:22 leisuxII kernel: NEW Incoming to GIAC IN=eth0 OUT=eth1
SRC=102.17.94.36 DST=192.168.0.3 LEN=40 TOS=0x00 PREC=0x00 TTL=50
ID=29272 PROTO=TCP SPT=63222 DPT=22 SEQ=2513326235 ACK=0 WINDOW=4096
RES=0x00 SYN URGP=0
Feb  1 16:11:41 leisuxII kernel: Bdr Router invalid OUTPUT IN= OUT=eth1
SRC=192.168.0.2 DST=65.43.19.26 LEN=60 TOS=0x00 PREC=0x00 TTL=64
ID=45054 DF PROTO=UDP SPT=1091 DPT=53 LEN=40
Feb  1 16:11:41 leisuxII kernel: Bdr Router invalid OUTPUT IN= OUT=eth1
SRC=192.168.0.2 DST=206.141.192.60 LEN=60 TOS=0x00 PREC=0x00 TTL=64
ID=45055 DF PROTO=UDP SPT=1091 DPT=53 LEN=40
Feb  1 16:11:41 leisuxII kernel: Bdr Router invalid OUTPUT IN= OUT=eth1
SRC=192.168.0.2 DST=65.43.19.26 LEN=60 TOS=0x00 PREC=0x00 TTL=64
ID=45055 DF PROTO=UDP SPT=1091 DPT=53 LEN=40
Feb  1 16:11:41 leisuxII kernel: Bdr Router invalid OUTPUT IN= OUT=eth1
SRC=192.168.0.2 DST=206.141.192.60 LEN=60 TOS=0x00 PREC=0x00 TTL=64
ID=45055 DF PROTO=UDP SPT=1091 DPT=53 LEN=40
Feb  1 16:11:41 leisuxII kernel: Bdr Router invalid OUTPUT IN= OUT=eth1
SRC=192.168.0.2 DST=65.43.19.26 LEN=60 TOS=0x00 PREC=0x00 TTL=64
ID=45055 DF PROTO=UDP SPT=1091 DPT=53 LEN=40
Feb  1 16:11:41 leisuxII kernel: Bdr Router invalid OUTPUT IN= OUT=eth1
SRC=192.168.0.2 DST=206.141.192.60 LEN=60 TOS=0x00 PREC=0x00 TTL=64
ID=45055 DF PROTO=UDP SPT=1091 DPT=53 LEN=40
Feb  1 16:11:41 leisuxII kernel: Bdr Router invalid OUTPUT IN= OUT=eth1
SRC=192.168.0.2 DST=65.43.19.26 LEN=60 TOS=0x00 PREC=0x00 TTL=64
ID=45056 DF PROTO=UDP SPT=1091 DPT=53 LEN=40
Feb  1 16:11:41 leisuxII kernel: Bdr Router invalid OUTPUT IN= OUT=eth1
SRC=192.168.0.2 DST=206.141.192.60 LEN=60 TOS=0x00 PREC=0x00 TTL=64
ID=45056 DF PROTO=UDP SPT=1091 DPT=53 LEN=40
Feb  1 16:18:00 leisuxII kernel: ISP: excessive pings IN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=24844 SEQ=256
Feb  1 16:18:01 leisuxII kernel: ISP: excessive pings IN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=24844 SEQ=512
Feb  1 16:18:02 leisuxII kernel: ISP: excessive pings IN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=24844 SEQ=768
Feb  1 16:18:03 leisuxII kernel: ISP: excessive pings IN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=24844 SEQ=1024
Feb  1 16:18:04 leisuxII kernel: ISP: excessive pings IN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
```

```
TYPE=8 CODE=0 ID=24844 SEQ=1280
Feb  1 16:26:34 leisuxII kernel: ISP: null scanIN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=40 TOS=0x00 PREC=0x00 TTL=56 ID=57786 PROTO=TCP
SPT=39943 DPT=22 WINDOW=1024 RES=0x00 URGP=0
Feb  1 16:26:40 leisuxII kernel: ISP: null scanIN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=40 TOS=0x00 PREC=0x00 TTL=56 ID=16601 PROTO=TCP
SPT=39944 DPT=22 WINDOW=1024 RES=0x00 URGP=0
Feb  1 16:27:16 leisuxII kernel: ISP: null scanIN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=40 TOS=0x00 PREC=0x00 TTL=49 ID=64896 PROTO=TCP
SPT=46275 DPT=55555 WINDOW=2048 RES=0x00 URGP=0
Feb  1 16:27:22 leisuxII kernel: ISP: null scanIN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=40 TOS=0x00 PREC=0x00 TTL=49 ID=53321 PROTO=TCP
SPT=46276 DPT=55555 WINDOW=2048 RES=0x00 URGP=0
Feb  1 16:36:10 leisuxII kernel: ISP: excessive pings IN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=42508 SEQ=256
Feb  1 16:36:11 leisuxII kernel: ISP: excessive pings IN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=42508 SEQ=512
Feb  1 16:36:12 leisuxII kernel: ISP: excessive pings IN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=42508 SEQ=768
Feb  1 16:36:13 leisuxII kernel: ISP: excessive pings IN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=42508 SEQ=1024
Feb  1 16:36:14 leisuxII kernel: ISP: excessive pings IN=eth0 OUT=
MAC=00:03:6d:14:29:ed:00:40:63:c1:9d:d3:08:00 SRC=102.17.94.36
DST=102.17.94.35 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=42508 SEQ=1280
Feb  1 16:39:51 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=256
Feb  1 16:39:52 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=512
Feb  1 16:39:53 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=768
Feb  1 16:39:54 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=1024
Feb  1 16:39:55 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=1280
```

```
Feb  1 16:39:56 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=1536
Feb  1 16:39:57 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=1792
Feb  1 16:39:58 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=2048
Feb  1 16:39:59 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=2304
Feb  1 16:40:00 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=2560
Feb  1 16:40:01 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=2816
Feb  1 16:40:02 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=3072
Feb  1 16:40:03 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=3328
Feb  1 16:40:04 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=3584
Feb  1 16:40:05 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=3840
Feb  1 16:40:06 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=4096
Feb  1 16:40:07 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=4352
Feb  1 16:40:08 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=4608
Feb  1 16:40:09 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=4864
Feb  1 16:40:10 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
```

```
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=5120
Feb  1 16:40:11 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=5376
Feb  1 16:40:12 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=5632
Feb  1 16:40:13 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=5888
Feb  1 16:40:14 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=6144
Feb  1 16:40:15 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=6400
Feb  1 16:40:16 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=6656
Feb  1 16:40:17 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=6912
Feb  1 16:40:18 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=7168
Feb  1 16:40:20 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=7424
Feb  1 16:40:21 leisuxII kernel: Bdr Router invalid INPUT IN=eth1 OUT=
MAC=00:03:6d:13:37:fd:00:40:63:c1:9d:d3:08:00 SRC=192.168.0.3
DST=192.168.0.2 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=43276 SEQ=7680
Feb  1 16:43:34 leisuxII kernel: device eth1 left promiscuous mode
```