Notes on Installing CERN's httpd on a firewall/bastion host.

Darren Reed,
Department of Architecture & Building,
University of Melbourne
13th August, 1994
darrenr@arbld.unimelb.edu.au
draft

1. INTRODUCTION....2

1.1 ENVIRONMENT. 2
1.2 AIM....2

2. THE 5 MINUTE JOB....2

3. GENERAL CONFIGURATION DETAILS. 3

3.1 DIRECTORY ACCESS....4
3.1.1 WARNING!!....4
3.2 PROTECTIONS 4
3.3 USING MAP/EXEC/PASS/REDIRECT AND FAIL. 4
3.3.1 Map....5
3.3.2 Exec....5
3.3.3 Pass and Fail....5
3.4 ACLS....6

4. PROXY SETUP. 6

4.1 WWW CLIENTS AND PROXYING....6
4.1.1 Source Routing and Proxying....6
4.2 PROXYING ON THE CERN SERVER. 7
4.2.1 Single Server 7
4.2.2 Dual Server 7
4.3 PROBLEMS....

5. CACHING....7

6. PASSWORD AND GROUP FILES....7

7. HACKING HTTPD ? "USE THE SOURCE LUKE!" :)....8

7.1 UNDOCUMENTED (?) CONFIGURATION DIRECTIVES 8
7.2 WWW LIBRARY: HTSECURE....8

8. CLOSING NOTES. 9

1. Introduction

If you don't have a copy of CERN's httpd Guide[LEE94] then I'd strongly suggest you get a copy before reading any further. There are continual references to it and it's not something to be without when configuring, tuning and running a httpd server in a sensitive environment.

1.1 Environment.

The environment in which I setup the CERN httpd is a tertiary institution which, like most education institutions it would seem, finds it pretty hard to justify spending money on firewall hardware (they're generally more interested in using the Internet, alas). To do this we run what is perhaps the simplest variation of a firewall - a single workstation with two ethernet cards and no routing from one side to the other in operation (ip_forwarding off) - a Dual-Homed gateway[MJR93] On the "inside" we have PC and Unix workstations all working over ethernet with Macs on AppleTalk/EtherTalk and on the outside is the rest of the University and the InterNet.

1.2 Aim.

The goal of setting up CERN's httpd was simple: to get it configured in such a manner that it would give full access of the web to anyone with a compliant client on the inside and allow outsiders to access our home page and other documents we wanted on display.

This document discusses issues which I was confronted with when setting up CERN's http and ones which I believe are of importance to people maintaining a machine in a firewall environment and who are running/would like to run a WWW server.

1.3 The URL's you don't see.

The URL[ref] is fast becoming a widely used and accpted form referencing any type of file on the Internet. When giving it to client software, you will typically hand it a string like:

```
http://foo.bar.edu/graphics/sphere.html
http://foo.com:8080/New/index.html
ftp://foo.baz.edu/pub/pc/net/mosaic.zip
```

etc.

What the client will do is break this up into several parts:

<service> :// <hostname> [: <port>] <filename>

The service tells it what protocol to speak, when connecting and (if there is no ":port" after the hostname) what port number to connect to. What the typical httpd will see when a client makes a request for a document is:

GET /

2. The 5 minute job

This quick list of what you need to get a minimal httpd up and functioning. It won't proxy, but it is secure. It'll get someone off your back about getting it working while you work on the real thing (depending on your firewall setup) :-). For a more detailed discussion on what httpd does and doesn't do, see later sections.

1. ftp the latest and greatest httpd from ftp.cern.ch
2. uncompress/gunzip, whatever you have to do to get the binary out.
3. put your httpd binary where you want to keep it
4. add a line to /etc/rc.local to start it at bootup
5. vi /etc/httpd.conf
6. Insert one of the sets of lines from below lines
7. save these lines as your /etc/http.conf file
8. run "httpd -v" and look for any trouble spots which need fixing (do this until it works flawlessly)
9. run httpd

Quick setup, no proxy:

```
ServerRoot        /top/dir/for/httpd
ServerType        standalone
PidFile                 proxy.pid
Hostname                <your hostname>
Port                    80
UserId                  nobody
DirAccess               off
AccessLog               /usr/adm/http-access
ErrorLog                /usr/adm/http-errors
ProxyAccessLog          /usr/adm/http-proxy
LogFormat               Common
LogTime                 LocalTime
Caching                 off
Gc                      off
Pass                    /*                    /top/dir/for/httpd/*
Fail                    *
```

Quick setup, allowing proxy:

```
ServerRoot              /top/dir/for/httpd
ServerType              standalone
PidFile                 proxy.pid
Hostname                <your hostname>
Port                    80
UserId                  nobody
DirAccess               off
AccessLog               /usr/adm/http-access
ErrorLog                /usr/adm/http-errors
ProxyAccessLog          /usr/adm/http-proxy
# The internet at large.
Protection              Prot-def              {
                        Mask          @(*)
}
Protect                 /*                    Prot-def
# our private subnets.
Protection              Prot-proxy    {
                        Mask          @(10.*)
}
Protect                 http:*        Prot-proxy
Protect                 ftp:*         Prot-proxy
LogFormat               Common
LogTime                 LocalTime
Caching                 on
CacheRoot               /top/dir/for/cache
CacheOnly               http:*
CacheAccessLog          /usr/adm/http-cache
Gc                      on
Pass                    /*                    /top/dir/for/httpd/*
Pass                    http:/*
Fail                    *
```

3. General Configuration Details.

CERN's httpd was the automatic choice for my situation - it provides built in SOCKS support and also supports being configured to work with other proxy servers. Many of the configuration options I describe below may be described somewhere as "default" but when security is at risk, better to be explicit about what you want (rather than trust a large program such as httpd) and put it in your configuration file anyway.

I've had a look at the NCSA WWW server in recent days, and although it probably does a good job and does it well, writing the configuration file for the server is much like writing an HTML document. Although this maybe a powerful abstraction, it is much too complex for my liking in an environment where security is critical. It does provide a few nice abstractions which allow the order of "deny/permit" to be changed (to name but one), but I find it much more reassuring knowing that the server will do it one way and the same way every time, regardless.

3.0pre6 is the latest version (at this time) and has better reporting of problems with your rule (configuration) file than earlier versions. I used a precompiled binary, as can be found on various ftp hosts (which, yes, isn't that sound but time was of the essence) and have used the source code, so far, just to read and for reference in learning how it works (or trying to :-) and look for potential security problems/holes/hidden features.

3.1 Directory Access

Being in an education environment, it seemed appropriate that "UserDir" be defined to let staff (or students) create their own home pages for show and linking into the department's. Be *VERY* careful with this! It appears to allow a reference such as:

```
http://server/~user/..
```

whereas opening

```
http://server/~user
```

will give a "Parent directory" which is NOT "~user/..", but the server root, IF permitted. However, "~user/../.." will NOT be allowed as httpd protects against the use of "/../" in a path, but not if it ends in "/..". Whilst this is not a terrible security leak, it's less than desirable. Creating a symbolic link in the WWW home directory to a directory outside this subdirectory allowed me out:

```
% ln -s /etc      ~/WWW/
```

and retrieving the passwd file was eventually possible. It is possible to turn *off* `raw' directory acces via this option in your httpd.conf file:

```
DirAccess                       off
```

This allows personal setup of files for display on the WWW reduces the threat to your file system from unauthorised access.

3.1.1 WARNING!!

Symbolic links are **NOT** protected against and thus a user can easily do:

```
% ln -s /etc/passwd       ~/WWW/
```

and your /**etc**/**passwd** file becomes available with the following URL:

```
http://server/~user/passwd
```

or even:

```
% ln -s / ~/WWW/root
```

and you're wide open.

3.2 Protections

Protections can be setup on the basis of hostnames or users/groups or a combination of these two. I would discourage everyone from using the username/password protection, on **ANY** document. It might be more secure (in some cases) than host, but I will gaurantee you you'll get innocent users typing in usernames and passwords believing it will get them somewhere (a student I was showing WWW to tried to access something which I assured him he wouldn't get access to and typed in his login and password, only to fail, which I made him change after this). It would be nice if httpd could filter this out even if it just automatically failed permissions! We goto enough trouble to hide passwords from the bad-guys only to have something like this entice people into giving it away, unlike anonymous ftp which has clients which will login for you and for which the procedure is rather well known amongst those on and off the Internet.

When creating Protection definitions, I would recommend using embedded Protections inside the rule file rather than have separate files. This makes it easier to manage (less files) and easier to work out what you're doing. REMEMBER that you can't put a comment inside the {}'s!

I won't go into further details about protection setup, the various documents available do this quite nicely, except to mention that you **MUST** have a rule to **ALLOW** access as the default option is for **NO-ACCESS** - unfriendly, maybe, but secure and easy to forget!

*"That which is not expressly premitted is prohibited"[*MJR93*]*

3.2.1 WARNING!!

When reporting an error, it would appear that the CERN httpd translates the path back as the full pathname, with the error (such as "fopen failed") rather than simply reporting that your URL failed. This may be an unacceptable risk to some who may not wish to disclose where user home directories are.

3.3 Using Map/Exec/Pass/Redirect and Fail.

These five directives are applied in sequential order as they appear in your httpd.conf file. The * is a **minimal** match, meaning that if I have:

```
http://foo.bar.edu/root/help/home.html
```

and put it through:

```
Pass       http://*/*                    /*/cache/*
```

it will come out as:

```
/foo.bar.edu/cache/root/help/home.html
```

The order in which they appear in your rule file is upto you, although I would strongly they appear in the order MAP, REDIRECT. EXEC, PASS and then FAIL. Of these five, only MAP is a fall-through directive, meaning that as soon as any of the other four are found to match a URL being checked by httpd, it stops/aborts further processing of these directives on the URL.

3.3.1 Map

If you're running on a dual homed host, you might wish to simplify things a bit for yourself by converting URL's refering to one name to the other (and the same with both IP#'s). To do this, use the following:

```
Map     /firewall-inner.domain/*    /firewall/*
Map     /firewall.domain/*          /firewall/*
Map     /10.0.0.51/*                /firewall/*
Map     /127.0.0.1/*                /firewall/*
Map     /localhost/*                /firewall/*
Map     /localhost.domain/*         /firewall/*
```

and if the machine's hostname is not a fully qualified domain name[ref], then you may wish to add these too:

```
Map     /firewall-inner/*           /firewall/*
```

and then if you add a "www" CNAME[ref], then you'll need one like this too:

```
Map     /www.domain/*               /www.domain/*
```

unless you wish to have a separate set of files for each of these URLs, they otherwise all refer to the same host and you will more than likely want them all to look at the same tree. The other way to do this is to create a few symbolic links in your httpd root directory, each representing the various ways to make a reference to your httpd host. I chose not to use this method, should, in future, httpd someday have an option to not follow symbolic links and to keep the directory cleaner.

You may also need something like this:

```
Map     http:*                      http:/*
```

if you have problems with proxying. I am given to understand the cause of the problem I had with not having this (an httpd setup to use another for proxying would send "GET http:/outsidehost/path" to the firewall httpd, which is doing the proxying, and that would try connect to 0.0.0.0 - missing the double /'s) to be a bug which shouldn't be :-).

If you have httpd's sharing diskspace via NFS, you can remove the need to query the remote server:

```
Map     http://other-host.domain/*  /nfs/dir/for/other/root/*
```

Remember! This is **NOT** a match and abort rule!

3.3.2 Exec

If you're setting up fancy WWW pages which contain "active" regions in gif files, then you'll want to have the "cgi-bin" executables available and will probably need something like this:

```
Exec    http://firewall/cgi-bin/*                   /cgi-bin/*
```

Firewall administrators, be warned! If you're acting as a proxy for an intenal WWW server which is NOT under your control, be wary of what is setup there (just as any other host on your internal network). It is not beyond possibility to have someone setup a "finger" or other type of search or network query which will disclose information you're trying to keep in. As this (and other services) are all enabled through the HTTP protocol and httpd, it is effectively the same as a cracker encapsulating "finger" or some other service through DNS[].

3.3.3 Pass and Fail

Pass and **Fail** separately determine what you do and don't wish to be accessed. For starters, you'll need to allow access to your "httpd-internal" icons directory and the root directory for your httpd server.

The Pass directive has some of the functionality of the Map directive builting, meaing that you can shorten your httpd.conf file with efficient combinations in Pass directives. For example:

```
Pass    http://firewall-inner/*             http://firewall/*
```

is roughly equivalent to:

```
Map     http://firewall-inner/*             http://firewall/*
Pass    http://firewall/*
```

IF you have turned on "UserDir", then you **will require** a separate rule for user directories to be passed along by httpd. As explained in the CERN http guide[1], ~is **NOT** matched by *'s.

Fail does the opposite of Pass and causes a "permission denied" to be returned by httpd whenever it matches a URL against the Fail command. For example, you could make up rules to prevent "/..":

```
Fail    */..
Fail    *~*/..
```

In case someone gets ideas about how cool using Exec is on an internal httpd, you might wish to consider using the following rule on the firewall httpd:

```
Fail    */cgi-bin/*
Fail    */htbin/*
```

but as they can make up arbitary places to put binaries, and there is no difference (as far as the protocol being passed through is concerned) between a text file and an executable in a "GET" command passed to and forwarded by a proxy server. Because of this, you may (quite reasonably) wish to not allow any interior httpd's access to the outside unless they are run by yourself exclusively - for security reasons.

[1] See section 4.6.1, page 20

3.3.4 When to Pass, Map, Execute and Fail

With these various bits and pieces, you might well try combining them in your rule file, only to find, for some bizarre reason they don't quite work as expected. The order in which they appear is *very* important. For example:

```
Map        http://firewall/*      /*
Map        /*      /home/of/www/*
Pass       /httpd-internal-icons/*
```

is going to result in icons not being lookup up correctly. Indeed, doing

```
Map        http://firewall/*      /*
Pass       /*      /home/of/www/*
Exec       /cgi-bin/*     /home/of/www/cgi-bin/*
```

is going to cause cgi-bin's to fail too and what you probably want is:

```
Map        http://firewall/*      /*
Exec       /cgi-bin/*     /home/of/www/cgi-bin/*
Pass       /*      /home/of/www/*
```

3.4 ACLs

The presence of ACLs makes customising protections in finger detail possible than might typically be used for files, as having per file protections in the rule file would make it harder to read what your general aim was.

I recommend against using the directive "**ACLOveride**" in the rule file, as the presence of the ACL file causes all other protection setups in the rule file to be ignored. This is especially true if you have "**UserDir**" defined and present in the rule file.

3.5 Logging

Through use of "AccessLog", "ErrorLog", "ProxyAccessLog" and "CacheAccessLog", all transactions made by httpd can be recorded. If you had some free disk space, and needed something to fill it, direct these there :) The typical home page can generate a dozen or more transactions with httpd, depending on the number of in-line images. The logging is fairly exhaustive, who, what, when, where, error code and size. The error code recorded is just the 3 digit decimal number[HTTP93].

4. Proxy Setup.

The CERN httpd is quite a flexible piece of software when applied to proxy services. The typical setup, I imagine, will be similar to what I've done with users connecting to a httpd and it making requests on their behalf.

4.1 WWW Clients and Proxying

Now, depending on your client, the options available maybe different. The NCSA-Mosaic client (v2.0 or later) is lovely in this respect - it allows a different proxy definition for each of the popular service it uses; ftp/wais/gopher/http/ news. Personally, I am quite content and happy to let them all use the same proxy providor - in this case httpd itself. I would imagine if you were using the TIS firewall toolkit (or were otherwise concerned about overgrown programs running on the firewall/bastion), you'd use a small program which handled proxy requests.

There are several ways to setup proxying for the NCSA Mosaic clients: a script (or just manually) setting "http_proxy", "ftp_proxy", etc, environment variables (NOTE: case is *IMPORTANT* for environment variables!) or just edit /usr/lib/X11/app-defaults/Mosaic and add a few lines like:

```
*.httpProxy:                    http://firewall/
*.ftpProxy:                     http://firewall/
```

If you're used to reading the URL's, that is doing what it suggests - using port 80 on the firewall host for ftp. I would recommend against using this is a mechanism for allowing ftp to the Internet as httpd is transaction based and continual spawning of ftpd's is not likely to be viewed upon kindly (an option to have it hold the connection open for 5-10 minutes would be nice while you do ls's, etc).

For the PC, there is a section that you can add lines to, specying the same as above and on Macs, it's a preferences menu.

4.1.1 Source Routing and Proxying

Through use of source routing (more than the other attacks, using routing protocols or lower layer protocols)[BEL89], there is potential for people to connect to your httpd gateway in your DMZ and use your httpd server as if they were someone from the inside.

There are two things which make it possible and both can be overcome - separately:

1. httpd not discarding TCP connections which have IP options present in them

2.having your router drop all packets from the outside which claim to be from the inside (this should be one of the first rules put into a router where packet filtering is possible).

4.2 Proxying on the CERN server.

There are (again) a couple of ways to setup the proxy service used by the server (and there looks to be a new one in the wind). The first is the same as the client: use the environment variables set before running the server. I personally don't like this, as if you've busy making changes or make an upgrade in a hurry, you may not run the right script or set the right variables yourself. The other option is adding the proxy definitions to your rule file, something like this:

```
http_proxy                      http://firewall/
ftp_proxy                       http://firewall/
```

If you look in your ErrorLog file, you may notice something about overwriting an environment variable (even if you didn't set it :). It is a nice comfort knowing it is overwriting the envinonment varibales with these from the rule file, even if they didn't exist before (well, maybe it is a bug, but it is reassuring :).

When you setup the protections, you should have specified a few lines like this:

```
Pass                            http:*
Pass                            ftp:*
```

to allow for passing through of the above. If you have problems, you may need a hack like this to allow it though:

```
Pass                            http:*          http:/*
```

You might also want to turn proxying off (using "no_proxy") for the firewall/bastion itself and any internal WWW servers.

4.2.1 Single Server

The first option you can take with proxying with the CERN server is to have the server do both proxying AND act as a WWW server. If you take this step, you'll need to use the "Protect" directive to setup protections for *everything* you're proxying, else *anyone* else can use GET and use your WWW server to proxy for them! ie, if you do it wrong, someone could use your WWW server to perform some sort of activity using *your* firewall as the staging tool! To do it correctly, you'll need some protection lines like this:

```
Protection        Proxying-Prot  {
                  ServerId       ProxyServer
                  Mask           @(10.*,127.0.0.1,192.1.1.*)
}
Protecttion       Internet-Prot  {
                  ServerId       InternetServer
                  GetMask        @(*)
}
Protect           http:*         Proxying-Prot
Protect           ftp:*          Proxying-Prot
Protect           /*             Internet-Prot
```

4.2.2 Dual Server

The other option is two use two WWW servers (preferably on different hosts, one on your bastion and the other further inside your internal network). Again, you'll need protection setup to allow selective use of it as a proxying agent.

CERN's httpd also allows for the server to be configured to use remote proxy agents[2]. So it is possible to have an "outer" httpd (on the firewall) and an inner one. In my environment, this seems to provide little gain, but if you're using the dual-router and bastion host configuration, this provides an easy plan for filtering: a single rule to allow anyone to your WWW port on the bastion from outside, and a single IP and port pair from the inside to the bastion (you can deny everything else).

5. Caching.

You will most definately want to turn this on. The documents are pretty self explanatory about how this works and some good examples are available.

Garbage cleaning goes hand in hand with caching so you'll want to enable this.

What the defaults are is a MUST for checking up on. Everyone schedules backups, /tmp cleaning, etc at different times, so it is likely that garbage disposal will need alterations from the defaults.

6. Password and Group files.

I don't trust the ideas behind these. They don't appear to make it very clear how/where it gets used or when to expect prompts (or maybe I need to read some more). They quite clearly state that there is no correllation between users in password files for http and those for unix, but it worries me with httpd running as root and changing its uid to "nobody" about which nobody and nogroup it is. On top of this, the extra password files are the bits which require passwords (see above for comments) for accessing documents.

They're also vulnerable to all the old and current problems of passwords which are entered across the network, which are slowly being solved with programs such as S/Key.

---

[2] See section 4.14, page 36

7. Hacking httpd ? “Use the source Luke!” :)

To gain further insights into how httpd *really* worked, I did what all good hackers do and went and read the source (not all of it yet :)! Not all of it yet, but what I found there was very interesting indeed! Most of this is probably due to this being a beta release so I’m not sure that it is really anyone’s fault. The only reason that I mention the following is that although it is beta software, people will install it if it does the correct job and is reliable and secure and crackers probably are familiar with it, even if you’re not.

7.1 Undocumented (?) Configuration Directives

None of these were found documented in the CERN httpd guide[LEE94] but WERE found to be accepted by httpd:

```
DnsLookup          <on|off>
AcceptHack <on|off>       (notes in the source code say no documenting:
                                  should be obvious why but what does it do ?)
Linger             <on|off>
ParentUserId       <UserId>
ParentGroupId      <GroupId>
ProxyAccessLog     <filename>
SuffixCaseSense    <on|off>
RuleCaseSense      <on|off>
UseProxy           (doesn't work due to fuzzy matching on "httpproxy", etc,
                    keywords)
SetUid             (semantics still vague according to the source)
SecurityLevel      high/normal
Dirshowhtmltitles is new
FtpDirInfo         (top/bottom/-ve ?) is new
DirCaseSense              is new
CacheFiles         is new
CacheRefreshInterval      is new
```

And when using Disable/Enable keywords, the available options are:

GET, POST, PUT, DELETE (all found in examples)
HEAD, CHECKIN, CHECKOUT, SHOWMETHOD, LINK, UNLINK

For documentation on these, you’ll have to read the IETF draft on HTTP[HTTP93]] or RFC once published.

Keywords recognised by httpd as service types (eg, httpd, gopher) and for which it can/will perform a proxy service for:

FTP, NEWS, GOPHER, WAIS, WHOIS (depends on compile option), HTTP, FILE
TELNET, TN3270, RLOGIN

So be *very* careful of using the sample configurations which list anything like this:

Pass *

of course using

Fail *

is safe but you won’t get far if it is at the top :)

7.2 WWW Library: HTSecure

The CERN httpd package includes an implementation of the WWW library (alas, minus documentation). In search of security realted bits and pieces, I found one item of interest:

a global variable called “HTSecure”

If you’ve got the source handy or compiled it yourself, you can have a quick peek at what its value is and what it does. If you have a precompiled binary, don’t worry, you can still change/check the value of this variable with adb’s help (or your system’s equivalent):

```
% echo "HTSecure?" | adb httpd
```

and to change its value:

```
% echo "HTSecure?W 1" | adb -w httpd
```

(again, your exact syntax to do this may not be the same).

HTSecure appears to affect 4 different parts of the WWW library and the behaviour of each is described for (a) HTSecure non zero and (b) HTSecure is 0:

1. /htbin

(a) allow allows programs in “/htbin/” to be executed. For (b), no check is made upon the path for sanity. If httpd is used with HTSecure set, you might like to put the “cgi-bin” directory inside “/htbin”, amongst the approved executeables;

2. REMOVE

(a) prevents the DELETE method from being used to remove files overriding “Enable DELETE”, whilst (b) leaves it up to the usual configuration;

3. file:

(a) prevents **ANY** file from being accessed, either through “file://” or just the normal “GET /...” being presented to httpd from a remote client. If you were running a proxy-only server, then this would be a desirable feature. (b) is the default, allowing files to be retrieved by clients;

4. telnet: and rlogin:

(a) prevents use of httpd as a proxy agent for telnet/rlogin whilst (b) leaves it upto the rule file.

8. Closing Notes.

I've not been playing with CERN's httpd for very long (read since the start of August) so I don't know many of the intricate details. However, I believe anything which is integral to security should be simple to configure so that it is easy to understand and less bits which can go "wrong". I've more than likely got things which need changing in my own rule file too.

CERN's httpd is a mixture, it is big (ie buggy :) but easy to get a `secure' configuration setup and operating. I like it's simplicity over and above it being big. Remember, the current version, 3.0pre6 is only a beta (but alas is the best to use, in my experience so far) so it maybe worth working out would be required (from a firewall point of view) to properly implement proxy preotections.

If people have other notes on using CERN's httpd (or indeed any other WWW servers) and would like to pass them onto me, I'll do some more work on this rather rushed document (and try to stay upto date on httpd) I hope this is of use.

[LEE94] Tim Berners-Lee and Ari Loutonen, “CERN http Guide”, 4th May 1994

[MJR93] Marcus J Ranum, “Thinking About Firewalls”, proceedings of the Second World Conference on Systems Management and Security (SANSII), 1993. ftp://ftp.tis.com/pub/firewalls/firewall.ps.Z

[HTTP93] Tim Berners-Lee, “Hyptertext Transfer Protocol: A stateless search, retrieve and manipulation protocol”, working draft, 5th November 1993.

[BEL89] Steven M. Bellovin, “Security Problems in the TCP/IP Protocol Suite”, Computer Communications Review, Vol 19, No. 2, pp 32-48, April 1989.