

Hardening your Apache server using log analysis

Karel Kubat, e-tunity

2003

Contents

1	Introduction	2
1.1	Quick Installation	2
2	Procedure	2
3	Configuration	3
3.1	Apache configuration	3
3.2	The script apachelog	3
3.3	Enabling a grace period	5
3.4	Room for Improvements	6
3.4.1	Allowing some requests for remote sites	6
3.4.2	Method of closing an IP address	6
3.4.3	A more elaborate grace scheme	6
4	Results	7

1 Introduction

This article describes how to harden your Apache server using log analysis. Furthermore, a small utility *apachelog* is shown, which does the job.

The 'hardening' of the server is applicable to all situations where an Apache webserver is exposed to the outside world – i.e., most situations. It's only a matter of time before potential hostile access requests arrive, trying to connect to remote systems (e.g., for spamming purposes), or requesting HTTP documents from other servers (for purposes of anonymous surfing). Hardening is especially useful -and necessary- in situations where the Apache webserver serves as a proxy. Proxies are more vulnerable to such attempts, because they are usually meant to serve remote documents.

1.1 Quick Installation

For the impatient:

- Read section 2, the procedure. Even if you're impatient.
- Install the script *apachelog* shown in section 3.2 into */root/bin*.
- Edit your Apache configuration file *httpd.conf* and enter the commands shown in section 3.1 at the right places.
- Restart your webserver.
- Read section 3.3 and enable a *cron* action to periodically restart your firewall or to periodically flush policy chains.

2 Procedure

In this article we assume that an Apache webserver exists that

- either does not need to serve remote documents (i.e., is not a proxy),
- or is configured as a local proxy, e.g., it serves remote documents from a webserver in a LAN, beyond a firewall. The Apache webserver is in this case a gateway to the LAN. The requests it receives appear for local documents, though the proxy will actually retrieve remote ones.

In both of the above cases, the Apache webserver does not need to serve true remote requests, meaning that the following requests can be considered 'hostile':

- *GET http://what/ever/site/*: This is a request to function as a proxy and to serve a remote document.
- *HEAD http://what/ever/site/*: This is similar to the above *GET*, but only obtains the page statistics.

-
- *POST http://what/ever/site/*: The result of a form posting to a remote server.
 - *CONNECT*: This is a request to establish a transparent connection to a remote system. The connection can be to a mail server, which in fact establishes a spam port.

As for the action to take upon detecting a hostile access attempt, we make the assumption that the webserver is running on a system that understands *ipchains*. Using *ipchains* we will stop the access for the given IP source.

3 Configuration

3.1 Apache configuration

In order to make the hardening effective, the Apache configuration is slightly altered. Using the directive *CustomLog*, we redirect the logging information to a script */root/bin/apachelog*. An example is given below:

```
1 # Here we define the "custom" log format. The source IP
2 # is the first part of the log line.
3 LogFormat "%h %l %u %t \"%r\" %>s %b" common
4
5 # Here we define the custom log stream. It goes to the
6 # script apachelog.
7 CustomLog "|/root/bin/apachelog /tmp/access.log /tmp/hostile.log" common
```

The *CustomLog* stream is redirected to the script */root/bin/apachelog*, which is also supplied two arguments: the 'true' access log, and the log for hostile attempts. The last argument on the line defines the log format (common).

3.2 The script apachelog

The script *apachelog* is shown below. It is a very simple Perl script, that logs its *stdin* input to the true access log. After that, the input is analyzed for 'hostile' access attempts, see section 2.

As one more feature, the script provides at the top a number of 'friendly' IP numbers, which will never be closed.

```
#!/usr/bin/perl

use strict;

# Configuration
# -----

# Friendly IP classes, which will NEVER be closed.
```

```
my @friendlies = ("127.0.0.", "192.168.1.", "213.51.99.189");

# Method of closing an IP. Values are either ipchains or iptables.
# The wiring conforms to the SuSE-standard "ipchains" or "iptables"
# firewall usage: ipchains uses the chain input, and denies packets,
# while iptables uses the chain INPUT, and drops packets.
my $closure = "ipchains";

if ($#ARGV != 1) {
    my $nargs = $#ARGV + 1;
    die << "ENDUSAGE";
```

This is the Apache access log catcher. Use as follows:

(1) In httpd.conf, use a rule like:

```
# Definition of the "common" log format. The IP address MUST be
# the first part of the line. The request MUST be enclosed in
# double quotes.
LogFormat "%h %l %u %t \"%r\" %>s %b" common

# Here we redirect the access log to the script apachelog.
CustomLog "|/where/ever/apachelog A.log B.log" common
```

This specifies that all access will be logged to A.log, but false access tries (proxying and so on) will also be logged to B.log.

(2) When false accessing is attempted, the IP will be closed using ipchains. To use e.g. iptables instead of ipchains, please adapt this script.

You invoked apachelog as:

```
apachelog @ARGV
with $nargs arguments.
```

```
ENDUSAGE
}
```

```
# Log a line to the NORMAL ACCESS attempts logfile.
sub accesslog ($) {
    open (OF, ">>$ARGV[0]")
    or die ("cannot append to $ARGV[0]: $!");
    print OF ($_[0]);
    close (OF);
}
```

```
# Log a line to the HOSTILE ACCESS attempts logfile.
sub hostilelog ($) {
    open (OF, ">>$ARGV[1]")
    or die ("cannot append to $ARGV[1]: $!");
    print OF ($_[0]);
    close (OF);
}
```

```

# Main loop starts here.
my $line;
LINE: while ($line = <STDIN>) {

    $line =~ s/\n//;
    $line =~ s/\r//;

    # Log it to the true access log.
    accesslog ("$line\n");

    # Let's see if this is an illegal request. Loop to the next line if
    # it isn't.
    next LINE unless ($line =~ /\ "GET http/i or
        $line =~ /\ "POST http/i or
        $line =~ /\ "HEAD http/i or
        $line =~ /\ "CONNECT /i);

    # Yup.. illegal request. Get the IP address.
    my $ip = $line;
    $ip =~ s/[ \t].*//;

    # Does the request originate from a friendly IP?
    my $f;
    foreach $f (@friendlies) {
next LINE if ($ip =~ /^$f/);
    }

    # It's a truly hostile attempt!
    hostilelog ("$line\n");

    # Determine the closure command.
    my $cmd;
    if ($closure eq "ipchains") {
        $cmd = "ipchains -I input 1 -s $ip -p tcp -j DENY";
    } elsif ($closure eq "iptables") {
        $cmd = "iptables -I INPUT 1 -s $ip -p tcp -j DROP";
    } else {
die ("apache log: closure method \"$closure\" not supported!\n");
    }

    my $ret = system ($cmd) >> 8;
    hostilelog ("WARNING: [$cmd] stopped with $ret, " .
"failed to close IP $ip\n") if ($ret);
}

```

3.3 Enabling a grace period

We strongly suggest a 'grace period' when installing the above shown procedure. There are several reasons:

- You probably don't want to block a source IP forever just because they once launched a 'hostile' access attempt.

-
- Several users may be hidden behind one IP address. When one launches a hostile attempt, then the access to your site will be denied also to all the others.

How long the grace period should be, is up to you. Usually, restarting your firewall each night does nicely. If your system doesn't use a firewall, then emptying (flushing) the input chain will do the job.

3.4 Room for Improvements

This section describes some ideas on improvements. For more information don't hesitate to contact us via info@e-tunity.com.

3.4.1 Allowing some requests for remote sites

The above script *apachelog* is by no means perfect. E.g., your setup may require you to allow the serving of requests for a given remote site *http://my.other.site/* but not from any other site.

In that case, the detection of what's 'hostile' would be more elaborate. E.g., the following code would do the trick:

```
1 # Let's see if this is an attack.
2 next LINE unless ($line =~ /GET http/i or
3                 $line =~ /POST http/i or
4                 $line =~ /HEAD http/i or
5                 $line =~ /CONNECT/i);
6
7 # Is it an allowed request?
8 next LINE if (/my.other.site/);
```

3.4.2 Method of closing an IP address

Also, a further improvement may be required in the *system()* call that blocks the source IP. In this example, we use *ipchains* to insert a rule in the chain *input* at slot #1. The rule must match the given IP and the *tcp* protocol. When activated, the request is denied. Your setup may require a slightly different approach. E.g., you might need *iptables* instead of *ipchains*. The script *apachelog* contains more information.

3.4.3 A more elaborate grace scheme

As a final hint for an improvement, the script could be rewritten so that per suspect IP number, the closure is only done when say more than 5 hostile attempts are made without intervening valid attempts from the same IP number. This could help in situations where many users hidden behind one IP address are trying to access your site, but only one of them is trying to misuse

it. The code wouldn't close the access right away, but would prevent closure upon receipt of valid intermediate requests.

4 Results

So far, our results have been very satisfactory. We've managed to successfully block access attempts that we consider hostile. Furthermore, the *ipchains* action that is triggered leads to a DENY, meaning that the source system will get no answer from our site, until they timeout. This at least somewhat slows such misuseage of the Net.