Gardol

Adaptive Denial of Service Attack Mitigation

by John Walker

March 2004

Contents

Chapter 1

Introduction

Distributed Denial of Service (DDoS) attacks are one of a system administrator's worst nightmares. A perfectly functioning, highly secure, redundantly configured, carefully firewalled system with adequate growth capacity of all kinds can, in a matter of moments, be brought to its knees by being bombarded by a huge number of apparently legitimate simultaneous requests generated by by machines all over the world. Driven into total overload, the Internet bandwidth and/or server capacity saturated by the attackers, the attacked site is rendered as inaccessible to legitimate users as if its computer room were firebombed. A company whose lifeblood is online commerce can find its artery severed at the whim of a malicious attacker.

DDoS attacks are, sadly, not the stuff of science fiction or a worse-case scenario spun by security consultants peddling their services. They are, sadly, a fixture of today's Internet. The site operated by the author of this program has come under such attacks several times since the year 1999, and has been under a continuous, concerted attack generating up to half a million bogus requests to its server since January 2004. It is that attack which motivated development of this program.

A Distributed Denial of Service Attack is *Distributed* because it originates not from a single source (which could be easily identified and blocked), but from a large collection of machines *distributed* geographically. None of these machines need, by itself, mount an intensive attack upon the target. It suffices that the collection of simultaneously attacking machine, together, generate sufficient volume to harm the intended victim. In the attack against `www.fourmilab.ch` in early 2004, up to five thousand machines all around the world simultaneously hit the site, and over a period of several weeks, more than twenty thousand different machines (identified by IP address—some may be the same physical machines appearing as different dynamically assigned IP addresses) were observed as participating in the attack. And the attack on Fourmilab was *minor* as DDoS attacks go; the attack on `sco.com` in January 2004 knocked that company's site off the Web and forced them to change their domain name.

How can this happen?

Chapter 2

System Environment Parameters

Set the following parameters to correspond to the system on which you're installing the software.

2.1 Directory where Perl is installed.

⟨ Perl directory 2a ⟩ ≡
```
/usr/bin/perl◇
```
Macro referenced in 7, 33.

2.2 Program Version

⟨ Version 2b ⟩ ≡
```
0.1◇
```
Macro referenced in 19b, 21, 22, 36a.

2.3 Release Date

⟨ Release Date 2c ⟩ ≡
```
March 2004◇
```
Macro referenced in 19b, 21, 22, 29, 36a.

2.4 Web Installation Directory

⟨ Web Directory 2d ⟩ ≡
```
/ftp/webtools/gardol◇
```
Macro referenced in 39b.

2.5 Web Log File Directory

⟨ Web Log File Directory 2e ⟩ ≡
```
/vitesse/server/logs/http◇
```
Macro referenced in 39b.

2.6 Operating System Default Parameters

The following parameters specify operating system installation specific path names and commands which are substituted when an argument of "-" is given for the corresponding option on the command line. This is purely a convenience to avoid length command lines when doing production testing and may be dispensed with in the interest of cleanliness if desired.

2.6.1 Restart IP Filter Command

When the IP Filter block list is updated, the following command is used to cause IP Filter to load the updated list and put it into effect.

⟨ Restart IP Filter Command 3a ⟩ ≡
```
/etc/init.d/ipfboot reipf◇
```
Macro referenced in 20a, 25a.

2.6.2 IP Filter Configuration File Template

The following path name is used to read the IP Filter configuration file template when the default operating system configuration is delected.

⟨ IP Filter Configuration File Template 3b ⟩ ≡
```
/etc/opt/ipf/ipf_conf_template.txt◇
```
Macro referenced in 20a, 24c.

2.6.3 IP Filter Configuration File

When the IP Filter configuration is updated, it is written to this file when the default operating system configuration is selected.

⟨ IP Filter Configuration File 3c ⟩ ≡
```
/etc/opt/ipf/ipf.conf◇
```
Macro referenced in 20a, 24a.

Chapter 3

`gardol.pl`: Attack Detection and Response

3.1 Attack Detection

The following two sections of code are the heart of Gardol, and where, as a system administrator using it to respond to an attack, where you need to invest the most work. From the initial detection and analysis of the attack, you must determine a unique signature, detectable by analysis of the HTTP access log, which allows you to discriminate attacking hosts from legitimate users by their pattern of access. The code which follows applies this test on individual log items.

For reasons of efficiency, the test is performed in two phases (either of which may be void). In the first, or "quick reject" phase, fields in the HTTP log item has been parsed out into the Perl numeric capture variables but no further processing, such as computing the Unix `time()` value from the date and time in the log item has been performed. Any tests you can make at this phase which exclude the log item as representating a potential attack permit reducing the overhead of Gardol's monitoring the log. Note that you can't use pattern matches in the quick reject phase, as they would destroy the values in the capture variables, but you shouldn't be doing pattern matches in a quick reject in any case. Records which can be excluded as belonging to an attack are discarded by executing the `@<Reject the current log item as benign@>` macro.

The following table lists the variables into which fields from the HTTP log are decoded. Fields which are present in the basic CERN/NCSA "Common Log Format" are labeled as "Common" in the **Log Format** column. Fields which are present only when reading an extended log format are identified by the log format which provides them. Each log format which appears lower in the table is a superset of those which precede it. The numeric capture variables (e.g. "`$5`") are available only in the quick reject phase. Named variables are defined before the full rejection phase is performed. Items with no entry in the **Quick** column are not available in that phase. One detail: if the length of the HTTP response to the requestor was zero, the `$7` variable in the quick reject phase will be "`-`", but the `$length` variable in the full reject phase will be set to 0.

Quick	Full	Item	Log Format
`$1`	`$ip`	IP Address	Common
`$2`	`$ident`	Identification	Common
`$3`	`$userid`	User ID	Common
`$4`	`$time_date`	Date and Time	Common
`$5`	`$request`	HTTP Request	Common
`$6`	`$status`	Status Code	Common
`$7`	`$length`	Length of Response	Common
`$8`	`$referer`	`Referer`	Combined
`$9`	`$agent`	`User-Agent`	Combined
`$10`	`$cachecont`	`Cache-Control`	Forensic
`$11`	`$pragma`	`Pragma`	Forensic
`$12`	`$proxy`	`X-Forwarded-For`	Forensic
Decoded Request Date and Time			
	`$mday`	Day of Month	Common
	`$mon`	Month abbreviation (e.g. "`Mar`"	Common
	`$year`	Year	Common
	`$hour`	Hour	Common
	`$minute`	Minute	Common
	`$second`	Second	Common
	`$timezone`	Time zone offset	Common
	`$mindex`	Month index ($0 - -11$)	Common
	`$iso_date`	ISO 8861 Date and Time	Common
	`$utime`	Unix `time()` value	Common

3.1.1 Quick reject non-attack HTTP log items

Unless the attack we're under is completely unsustainable (so intense it clogs our inbound pipe), once we've blocked most of the currently-attacking IP addresses, most of the remaining accesses in the HTTP log will be legitimate (non-attack-related). We want to ignore these with as little overhead as possible, so all tests which can be made from the raw items parsed from the HTTP log item should be tested for here.

Obviously, you're going to need to customise this code to reject records which aren't part of the particular attack on your site and pass those which are. *Important:* to reduce overhead, these tests are performed directly on the numbered patterm match capture variables parsed from the HTTP log item. *Do not* use a pattern match in this code, as it will destroy the values of these variables. Pattern matches don't belong in quick-reject code anyway.

⟨ Quick reject non-attack HTTP log items 5a ⟩ ≡

```
        #   Special status 573 indicates an attack hit pre-detected by the
        #   filter in Apache.
        if ($6 != 573) {
            #   But if an apparently attacking host sends a single packet
            #   which doesn't resemble an attack, remove it from the list
            #   of attackers.  Its future behaviour will determine whether
            #   we put it back on the list of miscreants.
            if (defined($hits{$1})) {
                ⟨ Delete host from list of attackers (5b $1 ) 15a ⟩
                if ($verbose) {
                    print(STDERR "--Purged by non-homepage hit $1\n");
                }
            }
            ⟨ Reject the current log item as benign 5c ⟩
        }

        #   Quick reject hits relayed by proxy servers
        if ($12 ne '-') {
            ⟨ Reject the current log item as benign 5c ⟩
        }
    ◇
```

Macro referenced in 9b.

To make quick and regular rejection code more readable, we define a little macro to do the `next` which causes the current log item to be ignored.

⟨ Reject the current log item as benign 5c ⟩ ≡

```
    next; ◇
```

Macro referenced in 5a, 6, 9b.

3.1.2 Reject non-attack HTTP log items

Where possible, detecting and rejection of non-attack log items should be performed as a quick rejection in the previous section. More complex rejection tests which require pattern matches, access to the decoded time and date, etc. should be done here, after the items parsed from the HTTP log item have been assigned to their respective variables.

⟨ Reject non-attack HTTP log items 6 ⟩ ≡

```
#   There are no items requiring non-quick rejection here.
#   An example of a non-quick rejection test is:
#
#       if (!(($request =~ m-/earthview/cache/-) && ($status == 404))) {
#           ⟨ Reject the current log item as benign 5c ⟩
#       }
```

◇

Macro referenced in 9b.

3.2 Main Program

"gardol.pl" 7 ≡

```
#! ⟨ Perl directory 2a ⟩

    ⟨ Documentation in POD format 22 ⟩

    ⟨ Global declarations 15c ⟩

    ⟨ Process command line options 19b ⟩
    ⟨ Validate option specifications 20a ⟩

    if ($#ARGV != 1) {
        &print_command_line_help;
        exit(0);
    }

    #   HTTP Log file to monitor
    my $read_HTTP_log = $ARGV[0];

    #   IP Filter packet disposition log
    my $read_IP_Filter_log = $ARGV[1];

    ⟨ Open log files to be monitored 8a ⟩;

    ⟨ Start monitoring for restart signal 8b ⟩

    while (1) {

        ⟨ Check for restart signal and reset log files if received 9a ⟩

        ⟨ Examine newly appended HTTP log items 9b ⟩

        ⟨ Examine newly appended IP Filter log items 11b ⟩

        if ((time() - $lastupd) >= $updtime) {
            ⟨ Update filter to block attacking hosts 12a ⟩
        }
        if ($verbose) {
            print(STDERR "--Sleeping $sleepytime seconds.\n");
        }
        sleep($sleepytime);
    }

    ⟨ Utility functions 20b ⟩
◇
```

3.2.1 Open log files to be monitored

We monitor two log files: the HTTP log to track incoming requests and detect sequences of requests which identify an attacking site, and the IP Filter log to watch for continuing hits from IP addresses we've already decided to ban. (Monitoring the IP Filter log permits timing out IP addresses which have ceased to hit us, which both reduces the length of the list of blocked addresses and allows subsequent access from a one-attacking floating IP address which has now been reassigned to a benign user.)

⟨ Open log files to be monitored 8a ⟩ ≡

```
open(FL, "<$read_HTTP_log") || die "Unable to open HTTP log file $read_HTTP_log";
seek(FL,  0, 2);            # Seek to end of log file

open(FLOG, "<$read_IP_Filter_log") || die "Cannot open IP Filter log $read_IP_Filter_log";
seek(FLOG,  0, 2);          # Seek to end of IP Filter log file
```

◇

Macro referenced in 7, 9a.

3.2.2 Start monitoring for restart signal

We register to catch the "HUP" signal which, when received, causes the files we're monitoring to be closed, re-opened, and re-seeked (re-sought?) to the end. This allows monitoring to be transferred when the log files are "cycled" without killing and restarting the program. Upon receiving the signal, we simply set the variable `$restart_signal_received` which causes the closing and re-opening of the files the next time we go around the main loop. This avoids any problems due to lack of re-entrancy in the underlying system or by snatching file descriptors out from under code in this program.

⟨ Start monitoring for restart signal 8b ⟩ ≡

```
$SIG{HUP} = sub {
    $restart_signal_received++;
#print("Restart! ($restart_signal_received)\n");
};
```

◇

Macro referenced in 7.

3.2.3 Check for restart signal and reset log files if received

If a restart signal has been received since the last time we went around the main loop, close and re-open the log files we're monitoring so if they've been cycled to new files, we'll commence monitoring them.

⟨ Check for restart signal and reset log files if received 9a ⟩ ≡

```
if ($restart_signal_received) {
    close(FL);
    close(FLOG);
    ⟨ Open log files to be monitored 8a ⟩
    $restart_signal_received = 0;
    if ($verbose) {
        print(STDERR "Log files re-opened.\n");
    }
}
```

◇

Macro referenced in 7.

3.2.4 Process HTTP log items

Read all HTTP log items appended since the last time around the main loop. Each is parsed, then tested to see if it is part of the attack. If so, we add its source to the list of currently-attacking hosts and record the time of the first attack (should it so be) and most recent attack from this host.

⟨ Examine newly appended HTTP log items 9b ⟩ ≡

```
while ($l = <FL>) {

    #   Parse request record
    if ($l !~ m/$parseHTTPlog/) {
        #   Can't parse log item.  Ignore it.
        if ($verbose) {
            print(STDERR "!! Cannot parse HTTP log item: $l");
        }
        ⟨ Reject the current log item as benign 5c ⟩
    }

    ⟨ Quick reject non-attack HTTP log items 5a ⟩

    ⟨ Parse HTTP log item into variables 10 ⟩

    ⟨ Reject non-attack HTTP log items 6 ⟩

    ⟨ Add IP address to list of attacking hosts 11a ⟩
}
```

◇

Macro referenced in 7.

3.2.4.1 Parse HTTP log item into variables

If quick rejection has not excluded this HTTP log item as benign, we proceed to store the fields parsed from it into individual variables and further parse the extracted time and date field into its components, which are then re-assembled into an ISO-8601 date and time and a Unix `time()` value for arithmetical comparisons.

⟨ Parse HTTP log item into variables 10 ⟩ ≡

```
        $ip = $1;
        $ident = $2;
        $userid = $3;
        $time_date = $4;
        $request = $5;
        $status = $6;
        $length = $7;
        $referer = $8;
        $agent = $9;
        $cachecont = $10;
        $pragma = $11;
        $proxy = $12;
        if ($length eq '-') {
            $length = 0;
        }

        #   Parse date and time field
        $time_date =~ m-(\d+)/(\w+)/(\d+):(\d+):(\d+):(\d+)\s([\+\-]\d+)$-;
        $mday = $1;
        $mon = $2;
        $year = $3;
        $hour = $4;
        $minute = $5;
        $second = $6;
        $timezone = $7;
        $mindex = $mnames{$mon};
        $iso_date = sprintf("%04d-%02d-%02d %02d:%02d:%02d", $year, $mindex, $mday,
                                $hour, $minute, $second);
        $utime = timelocal($second, $minute, $hour, $mday, $mindex - 1, $year);

    #print("$mday,$iso_date,$year,$hour,$minute,$second,$timezone\n");
    #print ("$ip,$ident,$userid,$time_date,$request,$status,$length,$cachecont,$pragma,$proxy\n");
    ◇
```

Macro referenced in 9b.

3.2.4.2 Add IP address to list of attacking hosts

This HTTP log item has been deemed an attack (or attack candidate, subject to further scrutiny when it's time to decide which IP addresses should be blocked). Increment the number of hits from the IP address, save the time of this hit and, if this is the first hit from this address, save the request time as the time of the first hit as well.

⟨ Add IP address to list of attacking hosts 11a ⟩ ≡

```
$hits{$ip}++;
if ($verbose) {
    print(STDERR "--($hits{$ip}) $l");
}
if (!defined($ufirst{$ip})) {
    $ufirst{$ip} = $utime;
}
$ulast{$ip} = $utime;
```

◇

Macro referenced in 9b.

3.2.5 Process IP Filter log items

Once a host has been deemed an attacker and its IP address placed in the IP Filter block list, we will no longer see hits from it in the HTTP log. Nonetheless, the host may very well continue to hit us, even without the satisfaction of a reply (this is the case for the brute denial attack which motivated the development of this program). If we relied exclusively on the HTTP log, once we denied access to the IP address, its timeout would start ticking down and, without any new hit appearing in the log, would expire. The host would then be granted access again, would resume the attack, be blocked once more, and so on.

To avoid this silly cycle, we examine records appended to the IP Filter log since the last cycle. This log is written with the `ipmon` program which comes with IP Filter; the file name containing this log is the second command line argument on the call to this program. For each new IP Filter log item, we parse the IP address, date and time, and disposition of the packet. If this is a packet blocked originating from an IP address we've deemed an attacker, reset the timeout to the time the blocked packet was received and increment the count of hits from this host. This will guarantee that a blocked host will continue to remain blocked as long as it continues to hit, even though the HTTP server never sees the blocked packets.

⟨ Examine newly appended IP Filter log items 11b ⟩ ≡

```
while ($l = <FLOG>) {
    if ($l =~ m/$parse_IP_Filter/) {
        $utime = timelocal($6, $5, $4, $1, $2 - 1, $3);
        $bp = $8;
        $ip = $9;
        if (($bp eq 'b') && (defined($ufirst{$ip}))) {
            $ulast{$ip} = $utime;
            $hits{$ip}++;
            if ($verbose) {
                print(STDERR "-+($hits{$ip}) $l");
            }
        }
    }
}
```

◇

Macro referenced in 7.

3.2.6 Update filter to block attacking hosts

The filter file (`$filtfile`) consists of a sequence of configuration commands for IP Filter. It is assembled by transcribing a fixed template file (`$templatefile`) until a sentinel (`$templatesentinel`) is encountered, interpolating the list of hosts to be blocked and any commands from the `--blacklist` if any, then appending the balance of the template to complete the filter configuration file. If a `$reload_IP_Filter` command has been specified, it is executed to notify the filter to put the updated rules into effect.

⟨ Update filter to block attacking hosts 12a ⟩ ≡

```
my $now = time();

open(TE, "<$templatefile") || die "Unable to open template file $templatefile";
open(OF, ">$filtfile") || die "Unable to create IP Filter configuration file $filtfile";

⟨ Transcribe template to IP Filter configuration file 12b ⟩

⟨ Generate list of IP addresses to be blocked 13 ⟩

⟨ Add blacklisted IP addresses to block list 14a ⟩

⟨ Transcribe template to IP Filter configuration file 12b ⟩

⟨ Remove timed out hosts from list of attackers 14b ⟩

close(OF);
if ($verbose) {
    printf(STDERR "--Blocked $nblock hosts.\n");
}
$lastupd = time();

if ($reload_IP_Filter ne '') {
    system("$reload_IP_Filter\n");
}
```

◇

Macro referenced in 7.

3.2.6.1 Transcribe template to IP Filter configuration file

Copy the IP Filter configuration template to the configuration file about to be put into service, stopping when we encounter the `$templatesentinel`.

⟨ Transcribe template to IP Filter configuration file 12b ⟩ ≡

```
while (<TE>) {
    if ($_ =~ m/^$templatesentinel/) {
        last;
    }
    print(OF);
}
```

◇

Macro referenced in 12a.

3.2.6.2 Generate list of IP addresses to be blocked

Walk through the list of hosts suspected to be attacking. Any host whose last hit is longer than the timeout interval is skipped and marked for deletion from the list of attacking hosts. Hosts whose last hit was within the timeout interval are tested against the denial criterion (usually based on the number of hits or mean hit rate) and, if deemed an attacker, are listed in an IP Filter `block` rule to bar access from that IP address.

⟨ Generate list of IP addresses to be blocked 13 ⟩ ≡

```
my @toh;
my $nblock = 0;
my $k;

foreach $k (sort keys %ulast) {
    if (($now - $ulast{$k}) >= $timeout) {
        push(@toh, $k);
    } else {
        if ($hits{$k} >= $minhits) {
            my $mina = int(($now - $ufirst{$k}) / 60);
            print(OF "block in log quick from $k to any  # $mina min. $hits{$k} hits\n");
            $nblock++;
        }
    }
}
```

◇

Macro referenced in 12a.

3.2.6.3 Add blacklisted IP addresses to block list

If a `--blacklist` file is specified and exists, transcribe the IP addresses from the blacklist to the list of hosts to be blocked by the filter. IP addresses to be blacklisted appear one per line. Comments which follow a `#` and blank lines are ignored. The actual blacklist command is not validated–if it's invalid, it's up to IP Filter to issue a warning and ignore it. Passing the blacklist specification through directly permits it to be a general expression, possibly including a netmask or other modifier.

Lines in the blacklist file which begin with "`+`" (after any leading white space is discarded) are transcribed to the blacklist file literally, dropping the plus sign. This permits specification of more complicated rules than a simple IP-based block to all destinations. Again, these statements are not syntax checked, so errors may lead to warnings or more dire consequences when IP Filter attempts to digest them.

⟨ Add blacklisted IP addresses to block list 14a ⟩ ≡

```
if (($blacklistfile ne '') && (-f $blacklistfile)) {
    my $b;

    if (open(BL, "<$blacklistfile")) {
        while ($b = <BL>) {
            $b =~ s/#.*$//;
            $b =~ s/^\s*//;
            $b =~ s/\s+$//;
            if ($b ne '') {
                if ($b =~ s/^\+//) {
                    print(OF "$b  # Transcribed from --blacklist $blacklistfile\n");
                } else {
                    print(OF "block in log quick from $b to any  # Included from --blacklist $blacklistfil
                }
            }
        }
        close(BL);
    }
}
```

◇

Macro referenced in 12a.

3.2.6.4 Remove timed out hosts from list of attackers

As we scan the list of attacking hosts, any which haven't hit us since the `$timeout` interval are added to the `@toh` (timed-out hosts) lists. We can't delete them on the fly because that disrupts the `foreach` statement we use to scan the list of hosts. So...we chew through the `@toh` list here and forget everything we knew about the hosts it cites.

⟨ Remove timed out hosts from list of attackers 14b ⟩ ≡

```
foreach $k (@toh) {
    ⟨ Delete host from list of attackers (14c $k ) 15a ⟩
    if ($verbose) {
        print(STDERR "--Timeout purged $k\n");
    }
}
```

◇

Macro referenced in 12a.

3.2.6.5 Delete host from list of attackers

This little macro purges the host with the IP address of the argument from all our tables of attacking hosts.

⟨ Delete host from list of attackers 15a ⟩ ≡

```
delete($ulast{@1});
delete($ufirst{@1});
delete($hits{@1});
```

◇

Macro referenced in 5a, 14b.

3.2.7 Template

⟨ Template 15b ⟩ ≡

◇

Macro never referenced.

3.3 Global declarations

⟨ Global declarations 15c ⟩ ≡

⟨ Perl language modes 15d ⟩

```
use Time::Local;
```

⟨ Log file parsing patterns 17b ⟩

⟨ Default parameter settings 16 ⟩

⟨ Global variables 17a ⟩

◇

Macro referenced in 7.

3.4 Perl language modes

⟨ Perl language modes 15d ⟩ ≡

```
require 5;
use strict;
```

◇

Macro referenced in 15c, 33.

3.4.1 Default parameter settings

The following variables contain parameters which control the operation of the program. All of these are defaults which can be overridden by command line options.

⟨ Default parameter settings 16 ⟩ ≡

```
#   HTTP transfer log parsing pattern
my $parseHTTPlog = $parse_forensic;

#   How long to sleep between scans of new log items
my $sleepytime = 10;

#   Frequency of update dumps in seconds
my $updtime = 5 * 60;

#   Time out hosts after this many seconds without a hit
my $timeout = 30 * 60;

#   Minimum hits from IP address to deem an attacker
my $minhits = 2;

#   Where to read IP Filter configuration file template
my $templatefile = 'ipf_conf_template.txt';

#   Sentinel marking where rules are interpolated in ipf.conf file
my $templatesentinel = '⟨ IP Filter Template Sentinel 28a ⟩';

#   Where to read explicit list of IP addresses to blacklist
my $blacklistfile = '';

#   Where to write IP Filter configuration file
my $filtfile = 'ipf.conf';

#   Command to load new rule set into IP Filter
my $reload_IP_Filter = '';
```

◇

Macro referenced in 15c.

3.4.2 Global variables

The following variables are global to the entire `main` program. We could make many of these more local, but the entire program is sufficiently short and straightforward we'd probably only end up obfuscating things in the interest of "purity."

⟨ Global variables 17a ⟩ ≡

```
#   Processing arguments and options

my $verbose = 0;            # Verbose output for debugging

#   Handy constants

my %mnames = split(/,/, "Jan,1,Feb,2,Mar,3,Apr,4,May,5,Jun,6,Jul,7,Aug,8,Sep,9,Oct,10,Nov,11,Dec,12");

#   Utility variables

my $lastupd = 0;            # Time of last update dump
my $restart_signal_received = 0; # Nonzero if HUP restart signal received

my ($l, $ip, $ident, $userid, $time_date, $request, $status, $length, $referer,
       $agent, $cachecont, $pragma, $proxy, $time_date, $mday, $mon, $year,
       $hour, $minute, $second, $timezone, $mindex, $iso_date, $utime, $bp);

my (%hits, %ufirst, %ulast);
```

◇

Macro referenced in 15c.

3.4.3 Log file parsing patterns

The following declarations define the regular expression patterns used to parse the different log file formats we read.

⟨ Log file parsing patterns 17b ⟩ ≡

```
⟨ Common log parsing pattern 18a ⟩
⟨ Combined log parsing pattern 18b ⟩
⟨ Forensic log parsing pattern 18c ⟩

⟨ IP Filter log parsing pattern 19a ⟩
```

◇

Macro referenced in 15c.

3.4.3.1 Common log parsing pattern

The following expression parses an Apache HTTP log in the CERN/NCSA "common log file" format, which is defined in the standard Apache `httpd.conf` file as follows:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
```

⟨ Common log parsing pattern 18a ⟩ ≡

```
my $parse_common = qr/^(\d+\.\d+\.\d+\.\d+)\s+(\S+)\s+(\S+)\s+\[(.*)\]\s+"(.*)"\s+(\d+)\s+([\-\d]+)"/;
```

◇

Macro referenced in 17b.

3.4.3.2 Combined log parsing pattern

The following expression parses an Apache HTTP log in the "combined log file" format, which contains the same 8 initial fields as the common log file format, plus quoted fields which specify the "`Referer`" and "`User-Agent`" from the HTTP request, or "`-`" if these items did not appear in the request.

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

⟨ Combined log parsing pattern 18b ⟩ ≡

```
my $parse_combined = qr/^(\d+\.\d+\.\d+\.\d+)\s+(\S+)\s+(\S+)\s+\[(.*)\]\s+"(.*)"\s+(\d+)\s+([\-\d]+)"/;
```

◇

Macro referenced in 17b.

3.4.3.3 Forensic log parsing pattern

The following expression parses an Apache HTTP log in the special "forensic log" format developed to diagnose and respond to the 2004 attack against **www.fourmilab.ch**. This format consists of the "`combined`" log format with quoted fields appended which contain the "`Cache-Control`", "`Pragma`", and "`X-Forwarded-For`" HTTP request header field specifications (or "`-`" if the field is absent).

To create a log in forensic format, add the following to your Apache `httpd.conf` file, adjusting the location where the log should be written in the `CustomLog` statement as appropriate.

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" \"%{Cache-Control}i\" \"%{Pragma
CustomLog /files/server/logs/http/forensic_log forensic
```

⟨ Forensic log parsing pattern 18c ⟩ ≡

```
my $parse_forensic = qr/^(\d+\.\d+\.\d+\.\d+)\s+(\S+)\s+(\S+)\s+\[(.*)\]\s+"(.*)"\s+(\d+)\s+([\-\d]+)\s"(
```

◇

Macro referenced in 17b.

3.4.3.4 IP Filter log parsing pattern

We track records appended to the IP Filter log in order to reset the inactivity timeout whenever a packet from an attacking host is blocked by the filter rules we've put into effect. These records look like the following:

```
05/03/2004 16:32:51.670599 hme0 @0:909 b 219.77.161.203,45931 -> 193.8.230.138,80 PR tcp len 20 48 -S I
```

and we're interested in parsing the date and time, whether the packet was blocked ("b") or passed ("p"), and the source IP address.

⟨ IP Filter log parsing pattern 19a ⟩ ≡

```
my $parse_IP_Filter = qr-^(\d+)/(\d+)/(\d+)\s+(\d+):(\d+):(\d+)\.(\d+)\s+\w+\s+\@\d+:\d+\s+(\w)\s+(\d+\.\
```

◇

Macro referenced in 17b.

3.4.4 Process command line options

We use the `Getopt::Long` module to process command line options.

⟨ Process command line options 19b ⟩ ≡

```
use Getopt::Long;

GetOptions(
            'blacklist=s' => \$blacklistfile,
            'copyright' => sub { print("This program is in the public domain.\n"); exit(0); },
            'help' => sub { &print_command_line_help; exit(0); },
            'ipfconf=s' => \$filtfile,
            'ipfsentinel=s' => \$templatesentinel,
            'ipftemplate=s' => \$templatefile,
            'ipfupdate=s' => \$reload_IP_Filter,
            'minhits=i' => \$minhits,
            'polltime=i' => \$sleepytime,
            'timeout=i' => \$timeout,
            'updtime=i' => \$updtime,
            'verbose' => \$verbose,
            'version' => sub { print("Version ⟨ Version 2b ⟩, ⟨ Release Date 2c ⟩\n"); exit(0); }
          );
```

◇

Macro referenced in 7.

3.4.5 Validate option specfications

Validate the option specifications before we begin processing. Pre-checking them avoids ugly pratfalls with the input half-processed.

⟨ Validate option specifications 20a ⟩ ≡

```
{
    my $ok = 1;

    if (($blacklistfile ne '') && (!(-f $blacklistfile))) {
        print(STDERR "Blacklist file $blacklistfile does not exist.\n");
        $ok = 0;
    }

    if ($reload_IP_Filter eq '-') {
        $reload_IP_Filter = '⟨ Restart IP Filter Command 3a ⟩';
    }

    if ($templatesentinel eq '-') {
        $templatesentinel = '⟨ IP Filter Configuration File Template 3b ⟩';
    }

    if ($templatefile eq '-') {
        $templatefile = '⟨ IP Filter Configuration File 3c ⟩';
    }

    if (!$ok) {
        die("Invalid option specification(s)");
    }
}
```

◇

Macro referenced in 7.

3.5 Utility Functions

The following utility functions are defined in the main program context to handle matters such as command line processing.

⟨ Utility functions 20b ⟩ ≡

```
⟨ Print command line help information 21 ⟩
```

◇

Macro referenced in 7.

3.5.1 Print command line help information

⟨ Print command line help information 21 ⟩ ≡

```
    sub print_command_line_help {
        print << "EOD";
Usage: gardol.pl [ options ] HTTP_log_to_monitor IP_Filter_log_to_monitor
       Options:
             --blacklist f   Add IP addresses in blacklist file f to block list
             --copyright     Print copyright information
             --help          Print this message
             --ipfconf f     Write IP Filter configuration to file f
             --ipfsentinel s Interpolate host blocking commands at this sentinel in IP Filter template
             --ipftemplate f Read IP Filter configuration template from file f
             --ipfupdate c   Use shell command c to update IP Filter configuration
             --minhits n     Require n consecutive attack packets before blocking host
             --polltime n    Poll log files for new items every n seconds
             --timeout n     Timeout and unblock hosts after n seconds of inactivity
             --updtime n     Update list of blocked sites every n seconds
             --verbose       Print verbose debugging information
             --version       Print version number
Version ⟨ Version 2b ⟩, ⟨ Release Date 2c ⟩
EOD
   }
◇
```

Macro referenced in 20b.

3.6 Documentation in POD format

⟨ Documentation in POD format 22 ⟩ ≡

```
=head1 NAME

gardol - Adaptive Denial of Service Attack Mitigation

=head1 SYNOPSIS

B<gardol.pl>
[I<options>]
I<HTTP_log_to_monitor>
I<IP_Filter_log_to_monitor>

=head1 DESCRIPTION

B<gardol> is a Perl program which monitors an HTTP log file to
detect accesses which identify IP addresses participating in a
Distributed Denial of Service (DDoS) attack and, in conjunction
with the
S<B<IP Filter>>
package, dynamically block them from attacking a server.

⟨ Options documentation 23a, ... ⟩

=head1 VERSION

This is B<gardol> version ⟨ Version 2b ⟩, released ⟨ Release Date 2c ⟩.
The current version of this program is always posted at
http://www.fourmilab.ch/webtools/gardol/.

=head1 AUTHOR

John Walker
(http://www.fourmilab.ch/)

=head1 BUGS

Please report any bugs to bugs@fourmilab.ch.

=head1 SEE ALSO

B<IP Filter> (http://coombs.anu.edu.au/~avalon/),
B<nuweb> (http://nuweb.sourceforge.net/),
S<Literate Programming> (http://www.literateprogramming.com/).

=head1 COPYRIGHT

This program is in the public domain.

=cut
```

◇

Macro referenced in 7.

3.6.1 Options

Here we document the command-line options.

⟨ Options documentation 23a ⟩ ≡

```
=head1 OPTIONS

All options may be abbreviated to their shortest
unambiguous prefix.

=over 5
◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.1 `--blacklist` *filename*

⟨ Options documentation 23b ⟩ ≡

```
=item B<--blacklist> I<filename>

Whenever the list of IP addresses to be blocked is updated, the
IP addresses listed in the specified I<filename> (one per line, with
blank lines and any text following a "#" character ignored) will be
added to the list of IP addresses to be blocked.
◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.2 `--copyright`

⟨ Options documentation 23c ⟩ ≡

```
=item B<--copyright>

Display copyright information.
◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.3 `--help`

⟨ Options documentation 23d ⟩ ≡

```
=item B<--help>

Display how to call information.

◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.4 `--ipfconf` *filename*

⟨ Options documentation 24a ⟩ ≡

```
=item B<--ipfconf> I<filename>

The IP Filter configuration file containing the list of IP addresses
to be blocked is written to the specified I<filename>.
The default configuration file name is C<B<ipf.conf>> in
the current directory.  If a I<filename> of "B<->" is specified,
the installation default template location of
C<B<⟨ IP Filter Configuration File 3c ⟩>> is specified.

◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.5 `--ipfsentinel` *string*

⟨ Options documentation 24b ⟩ ≡

```
=item B<--ipfsentinel> I<string>

The list of IP addresses to be blocked is interpolated into the
B<--ipftemplate> file when a line containing the I<string> is
read.  The default sentinel is:

S<C<⟨ IP Filter Template Sentinel 28a ⟩>>

◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.6 `--ipftemplate` *filename*

⟨ Options documentation 24c ⟩ ≡

```
=item B<--ipftemplate> I<filename>

The specified I<filename> is used as the template for the IP
Filter configuration file.  Lines of the template are copied
to the configuration file, with the list of attacking IP
addresses to be blocked interpolated at the specified
B<--ipfsentinel>.
The default template file name is C<B<ipf_conf_template.txt>>
in the current directory.  If a I<filename> of "B<->" is specified,
the installation default template location of
C<B<⟨ IP Filter Configuration File Template 3b ⟩>> is specified.

◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.7 `--ipfupdate` *command*

⟨ Options documentation 25a ⟩ ≡

```
=item B<--ipfupdate> I<command>

The specified I<command> will be executed to inform
IP Filter when changes are made in the configuration
file, instructing it to load the new rules.  The default
command is blank, which disables reloading of the filter
rules.  Specifying a I<command> of "B<->" sets the reload command
to the IP Filter installation default of
C<B<⟨ Restart IP Filter Command 3a ⟩>>.

◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.8 `--minhits` *n*

⟨ Options documentation 25b ⟩ ≡

```
=item B<--minhits> I<n>

A total of I<n> consecutive attack packets (default 2) from a
given host will be required before its IP address is added to
the block list..

◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.9 `--polltime` *t*

⟨ Options documentation 25c ⟩ ≡

```
=item B<--polltime> I<t>

The HTTP and IP Filter log files will be checked for newly appended
records every I<t> (default 10) seconds.  Log records are read and
processed until the end of the log file is encountered, whereupon
B<gardol> sleeps for I<t> seconds before checking whether additional
items have been added.

◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.10 `--timeout` *t*

⟨ Options documentation 26a ⟩ ≡

```
=item B<--timeout> I<t>

Hosts from which neither an attack packet nor a packet blocked by
IP Filter has been received in I<t> seconds (default 1800 seconds--30
minutes) are timed out and access from their IP addresses re-enabled.
This prevents floating IP addresses from remaining blocked once an
attacker ceases to hit from them.

◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.11 `--updtime` *t*

⟨ Options documentation 26b ⟩ ≡

```
=item B<--updtime> I<t>

The IP Filter configuration file is updated, adding host newly added
to the block list and removing those which have timed out, every
I<t> seconds (default 300--five minutes).  The
B<--updtime> should be sent sufficiently short to respond to
the changing population of attacking hosts in a timely
manner, but not so short as to create too much overhead updating
and reloading the filter configuration.

◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.12 `--verbose`

⟨ Options documentation 26c ⟩ ≡

```
=item B<--verbose>

Generate verbose output to indicate what's going on.

◇
```

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

3.6.1.13 `--version`

⟨ Options documentation 27 ⟩ ≡

```
=item B<--version>

Display version number.

=back
```

◇

Macro defined by 23abcd, 24abc, 25abc, 26abc, 27.
Macro referenced in 22.

Chapter 4

ipf_conf_template.txt: IP Filter Configuration File Template

⟨ IP Filter Template Sentinel 28a ⟩ ≡

```
#### INSERT ATTACK_DENIAL RULES HERE ####◇
```

Macro referenced in 16, 24b, 28b.

"ipf_conf_template.txt" 28b ≡

```
## ipf.conf - config file for ipfilter
##
## pass all local traffic
pass in quick on lo0 all
pass out quick on lo0 all

## By default, pass all traffic (this is appropriate for a machine
## already situated behind a firewall).

pass in on hme0 all
pass out on hme0 all

## Ephemeral rules incorporated by Gardol
```

⟨ IP Filter Template Sentinel 28a ⟩

```
## End ephemeral rules incorporated by Gardol

##
## end of ipfilter ruleset
◇
```

Chapter 5

index.html: Main Web Page

This is the main Web page for Gardol. It contains the user documentation and the initial request form.

"index.html" 29 ≡

⟨ HTML header section 30a ⟩

```
<body bgcolor="#FFFFFF" >

<center>
<h1><img src="figures/gardol.png" width="512" height="214" alt="GARDOL"></h1>
<h2>Adaptive Denial of Service Attack Mitigation</h2>
<h3>by <a href="/">John Walker</a></h3>
</center>


 <p>


<div class="bodycopy">
```

⟨ Introductory text 30b ⟩

⟨ Questions to answer 31 ⟩

```
<h2>Download <cite>Gardol</cite>:   <a href="gardol.tar.gz"><tt>gardol.tar.gz</tt></a> source archive</h2
<h2>Read <cite>Gardol</cite> <a href="gardol.pdf">source code</a></h2>

<p>
<hr>

<h3><a href="/">Fourmilab Home Page</a></h3>

<address>
by <a href="/">John Walker</a><br>
```

⟨ Release Date 2c ⟩

```
</address>
<center>
<em>This document is in the public domain.</em>
<br> 
</center>

</div>

</body>
</html>
```

◇

5.1 HTML Header Section

⟨ HTML header section 30a ⟩ ≡

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="en">
<head>
<title>Gardol: Adaptive Denial of Service Attack Mitigation</title>
<style type="text/css">
    DIV.bodycopy {
        margin-left: 10%;
        margin-right: 10%
    }
</style>
<XXbase href="http://www.fourmilab.ch/" >

<meta name="keywords" content="gardol, denial, service, attack, mitigation, tool, response">
<meta name="description" content="Gardol: Adaptive Denial of Service Attack Mitigation">
<meta name="author" content="John Walker">
<meta name="robots" content="index">

</head>
```

◇

Macro referenced in 29.

5.2 Introductory Text

⟨ Introductory text 30b ⟩ ≡

◇

Macro referenced in 29.

5.3 Questions to Answer

⟨ Questions to answer 31 ⟩ ≡

```
<h2>Questions to Answer</h2>

When an apparent distributed denial of service attack hits your
site, it's easy to panic, especially if it effectively knocks
your site off the Web and the boss is running around like a
beheaded chicken urging you to "do something".  This is
even more severe at sites like Fourmilab, where I have to play
both the rle of the chicken and the beleaguered system
administrator simultaneously.  Still, before you can ``do anything''
that's likely to improve the situation, you need to carefully determine
the facts of the matter, which process you can begin by answering the
following questions.

<h3>Are we really under attack?</h3>

You may have decided your site is under attack because you've observed
a large bulge in the hit rate on your server, seen your inbound
and/or outbound network bandwidth hit the peg, or watched the CPU
load on your server reach saturation.

<table align="right">
<tr><td align="center">
<img src="figures/2004-01.png" width="209" height="200" alt="Web access log: January 2004">
<br>
<small><em>
An attack on Fourmilab.
</em></small>
</table>

The first question to ask yourself is whether these symptoms are
actually the result of a DDoS attack or something else
entirely.  Certainly, all of these things are characteristic
of a DDoS attack, but they do not, by themselves, indicate an
attack is actually underway.  The chart at the right shows the onset
of the
<a href="http://www.fourmilab.ch/documents/security/incidents/ddos_2004-01/" target="_blank">DDoS
attack against Fourmilab</a>
in January 2004.  Prior to
the attack, the hits per day had been at the typical level of
500,000 to 650,000 per day.  As the attack began, hits per day
increased to over a million per day and stayed at that level.
Analysis of the request log showed the additional hits were
completely stylised requests for nothing but the home page,
originating from thousands of different IP addresses all over
the world, with each requestor repeating at a more or less
constant rate, and several hundred new IP addresses "recruited"
into the attacking population each hour.  The attack requests were
easily distinguished from legitimate requests for the home page
because they contained request header variables which no browser
was known to send.  In this case, an unexpected bulge in request rate
did indeed indicate an attack.  But this is not always the case.

<br clear="right">

<p>
```



```
Consider the access graph at the left, covering a week in April 2003.  Once
again, requests were running at the typical rate with the usual slowdown
during the week-end when, on the 13th of April, the hit rate exploded
to more than double the usual average rate.
```

Chapter 6

badbot.pl: BadBot Attack Simulator

Bad bot, bad bot!
How 'ya gonna fight?
When somebody aims it,
At your site.

In the immortal words of Kelvin R. Throop, "If you haven't tried it, it doesn't work." **BadBot** allows you to simulate distributed denial of service attacks against your site, allowing you to debug attack detection code in **Gardol**. Running **Gardol** nonprivileged, simply monitoring the HTTP log file and writing a filter list in a private directory, allows you to test detection of simulated attacks mounted with **BadBot** without disrupting the operation of your site in any way.

BadBot is controlled by a sequence of commands (they look like options, but they're really imperative commands processed in sequence) on its command line. You can submit `GET`, `HEAD`, and `POST` requests to Web servers, wait for a specified number of seconds, and loop repeating a sequence of commands until the program is killed. The `--help` command prints a list of all available commands. For example, to reproduce the every four minute hit on the home page attack which struck Fourmilab in early 2004, the following command will suffice.

```
perl badbot.pl --set Cache-control=no-cache \
               --set Referer=- \
               --agent '' \
               --get http://www.victim.org/ --wait 240 --loop
```

6.1 Main program

"badbot.pl" 33 ≡

```
#! ⟨ Perl directory 2a ⟩

    ⟨ Perl language modes 15d ⟩
    use LWP;

    my $browser = LWP::UserAgent->new();
    $browser->env_proxy();

    my $verbose = 0;
    my $settings = '';

command:
    for (my $n = 0; $n <= $#ARGV; $n++) {
        my $cmd = $ARGV[$n];

        if ($cmd !~ m/^\-/) {
            $cmd = '--get';
            $n--;
        }

        if ($cmd =~ m/^\-/) {
            my $opt = $cmd;
            $opt =~ s/\-+//;
#print("$n: command $opt\n");

            ⟨ Agent command handler 35e ⟩
            ⟨ Get command handler 34b ⟩
            ⟨ Head command handler 35a ⟩
            ⟨ Help command handler 36a ⟩
            ⟨ Loop command handler 36b ⟩
            ⟨ Post command handler 35c ⟩
            ⟨ Set command handler 37a ⟩
            ⟨ Wait command handler 38a ⟩
            ⟨ Verbose command handler 37b ⟩

            die("Unknown command $cmd");
        }
    }

◇
```

6.2 Command handlers

The following sections handle the various command line options (which are actually command, not modal options). Each tests for the option, then does whatever action it requests. If the option takes an argument, it is scanned and validated within the option handling code. Note that each command handler must end with "`next command`" to avoid falling through into the unknown command code.

6.2.1 URL Fetch commands

The following macro retrieves the URL in `$arg` with the protocol given in the argument. If the user has requested one or more header variables be set with the `--set` command, we build the fetch method call as a string and execute it with `eval`, making a special case for POST requests, which require a dummy form data array argument before the list of header variables. It's ugly, but it gets you there.

⟨ URL Fetch command 34a ⟩ ≡

```
if ($verbose) {
    my $c = "@1";
    $c =~ tr/a-z/A-Z/;
    print("Fetching $arg with $c protocol.\n");
}
my $reply;

if ($settings eq '') {
    $reply = $browser->@1($arg);
} else {
    my @pform;
    my $parg = ("@1" eq 'post') ? ', \@pform' : '';
    my $s = '$reply = $browser->@1($arg' . $settings . ')';
    eval($s);
}
if ($verbose) {
    printf("%s:  Status: %s  Content length: %d\n",
        ($reply->is_success ? "Success" : "Failure"),
        $reply->status_line, length($reply->content));
}
next command;
```

◇

Macro referenced in 34b, 35ac.

6.2.1.1 `--get` *url*

The `--get` command retrieves the specified *url* with the "`GET`" protocol.

⟨ Get command handler 34b ⟩ ≡

```
if ($opt =~ m/^g/) {
    ⟨ Get command argument 38b ⟩
    ⟨ URL Fetch command (34c get ) 34a ⟩
}
```

◇

Macro referenced in 33.

6.2.1.2 `--head` *url*

The `--head` command retrieves the specified *url* with the "`HEAD`" protocol.

⟨ Head command handler 35a ⟩ ≡

```
if ($opt =~ m/^hea/) {
    ⟨ Get command argument 38b ⟩
    ⟨ URL Fetch command (35b head ) 34a ⟩
}
```

◇

Macro referenced in 33.

6.2.1.3 `--post` *url*

The `--post` command retrieves the specified *url* with the "`POST`" protocol.

⟨ Post command handler 35c ⟩ ≡

```
if ($opt =~ m/^p/) {
    ⟨ Get command argument 38b ⟩
    ⟨ URL Fetch command (35d post ) 34a ⟩
}
```

◇

Macro referenced in 33.

6.2.2 `--agent` *agent_name*

The `--agent` command sets the `User-Agent` field for requests.

⟨ Agent command handler 35e ⟩ ≡

```
if ($opt =~ m/^a/) {
    ⟨ Get command argument 38b ⟩
    $browser->agent($arg);
    if ($verbose) {
        print("User-Agent set to ", $browser->agent(), "\n");
    }
    next command;
}
```

◇

Macro referenced in 33.

6.2.3 --help

The --wait command prints how-to-call information.

⟨ Help command handler 36a ⟩ ≡

```
    if ($opt =~ m/^hel/) {
        ⟨ Echo command without argument 38c ⟩
        print << 'EOD';
Usage: badbot.pl [ commands ]
       Commands:
             --agent name       Set User-Agent to name
             --get url          Retrieve url with GET protocol
             --head url         Retrieve url header with HEAD protocol
             --help             Print this message
             --loop             Loop executing commands
             --post url         Retrieve url with POST protocol
             --set item=value   Add item to request header with specified value
             --verbose          Print verbose debugging information
             --wait n           Sleep for n seconds
        A command which doesn't begin with a "-" is submitted as a
        URL with GET protocol.
Version ⟨ Version 2b ⟩, ⟨ Release Date 2c ⟩
EOD
        next command;
    }
◇
```

Macro referenced in 33.

6.2.4 --loop

The --loop command, which must be last on the command line, causes BadBot to loop forever over the previous commands.

⟨ Loop command handler 36b ⟩ ≡

```
    if ($opt =~ m/^l/) {
        ⟨ Echo command without argument 38c ⟩
        if ($n != $#ARGV) {
            die("The --loop command must be the last on the command line");
        }
        $n = -1;
        next command;
    }
◇
```

Macro referenced in 33.

6.2.5 `--set` *item=value*

The `--set` command causes a header variable to be included in the HTTP request with the specified *item* name and *value*. Any number of `--set` commands may be specified; each supplying a different header field to the request. To keep `--set` commands from accreting without bound when commands are executed in a `--loop`, they are removed from the command line after being processed.

⟨ Set command handler 37a ⟩ ≡

```
        if ($opt =~ m/^s/) {
            ⟨ Get command argument 38b ⟩
            if ($arg !~ m/^([^=]+)=(.*)$/) {
                die("Syntax error in $cmd option argument");
            }
            $settings .= ", '$1' => '$2'";
    #print("Before N = $n ", join(', ', @ARGV), "\n");
            splice(@ARGV, $n - 1, 2);
            $n -= 2;
    #print("After N = $n ", join(', ', @ARGV), "\n");
            if ($verbose) {
                print("Request settings: $settings\n");
            }
            next command;
        }
    ◇
```

Macro referenced in 33.

6.2.6 `--verbose`

The `--verbose` command sets a flag which causes progress and status messages to be printed on standard output as commands are executed.

⟨ Verbose command handler 37b ⟩ ≡

```
        if ($opt =~ m/^v/) {
            $verbose = 1;
            ⟨ Echo command without argument 38c ⟩
            next command;
        }
    ◇
```

Macro referenced in 33.

6.2.7 `--wait` *seconds*

The `--wait` command causes the program to sleep for the requested number of *seconds*.

⟨ Wait command handler 38a ⟩ ≡

```
if ($opt =~ m/^w/) {
    ⟨ Get command argument 38b ⟩

    if ($arg !~ m/^\d+$/) {
        die("Argument to $cmd must be numeric.\n");
    }
    if ($verbose) {
        print("Waiting $arg seconds.\n");
    }
    sleep($arg);
    next command;
}
```

◇

Macro referenced in 33.

6.2.8 Get command argument

The next command line argument is taken and stored into `$arg`. If no argument is specified, an error message is issued and we give up.

⟨ Get command argument 38b ⟩ ≡

```
if ($n == $#ARGV) {
    die("Argument missing after $cmd command");
}
my $arg = $ARGV[++$n];
⟨ Echo command with argument 39a ⟩
```

◇

Macro referenced in 34b, 35ace, 37a, 38a.

6.2.9 Echo command without argument

If `--verbose` is set, this code echoes the current command (which has no argument) to standard output.

⟨ Echo command without argument 38c ⟩ ≡

```
if ($verbose) {
    printf("%3d: %s\n", $n, $cmd);
}
```

◇

Macro referenced in 36ab, 37b.

6.2.10 Echo command with argument

If `--verbose` is set, this code echoes the current command and its argument to standard output.

⟨ Echo command with argument 39a ⟩ ≡

```
        if ($verbose) {
            printf("%3d: %s %s\n", $n - 1, $cmd, $arg);
        }
    ◇
```

Macro referenced in 38b.

Chapter 7

Makefile

This is the `Makefile` for Gardol. Of course, generating the `Makefile` from the Nuweb invites infinite regress, since it's the `Makefile` which invokes `nuweb` to create.... But as long as we include the generated `Makefile` in the source distribution, all will be well, and we do that below, in the definition of the `Makefile` in the Nuweb. **Slap!** Thanks—I needed that.

Since, in the interest of preserving formatting in the LaTeX code documentation, we edit this file with hardware tabs disabled, we must cope with the regrettable detail that `make` uses tabs as a significant character.

"Makefile.mkf" 39b ≡

```
WEBDIR = ⟨ Web Directory 2d ⟩

LOGDIR = ⟨ Web Log File Directory 2e ⟩

PROGRAMS = gardol.pl

duh:
        @echo "Please choose: check dist publish test weblint"

⟨ Extract source code from Nuweb 40a ⟩

⟨ Installation 41c ⟩

⟨ Source distribution 40b ⟩

⟨ Documentation 41a ⟩

⟨ Testing 41b ⟩
◇
```

7.1 Extract source code from Nuweb

All of the source code for Gardol, its support files, documentation, and the tools used to build it are defined in the Nuweb file `gardol.w`. Processing this file with `nuweb` suffices to extract all the contents, so we can use the Perl source code `gardol.pl` as a proxy for all the files generated from the Nuweb program. Any `Makefile` target which requires a file from the Nuweb can simply specify `gardol.pl` as a dependency and be sure everything is up to date.

One little detail... since the `Makefile` itself is defined here, when you make a change you must first do something that processes the Nuweb ("`make check`" is a good choice) before the `Makefile` will contain the changes you made.

⟨ Extract source code from Nuweb 40a ⟩ ≡

```
gardol.pl:    gardol.w
        nuweb gardol
        chmod 755 $(PROGRAMS)
        unexpand -a <Makefile.mkf >Makefile
```

◇

Macro referenced in 39b.

7.2 Source distribution

⟨ Source distribution 40b ⟩ ≡

```
dist:   $(PROGRAMS) pdf
        rm -f gardol.tar gardol.tar.gz
        tar cfv gardol.tar gardol.w gardol.pl Makefile ipf_conf_template.txt index.html gardol.pdf \
                drop.sty badbot.pl
        gzip gardol.tar
```

◇

Macro referenced in 39b.

7.3 Documentation

⟨ Documentation 41a ⟩ ≡

```
view:   gardol.pl
        latex gardol
        nuweb gardol
        latex gardol
        xdvi -s 0 gardol

viewman: gardol.pl
        pod2man gardol.pl >ZZgardol.1
        groff -X -man ZZgardol.1
        rm -f ZZgardol.1

pdf:    gardol.pl
        sed 's///' <gardol.tex >ZZgardol.tex
        pdflatex ZZgardol
        pdflatex ZZgardol
        mv ZZgardol.pdf gardol.pdf
        rm -f ZZgardol*

viewpdf: pdf
        acroread gardol.pdf
```

◇

Macro referenced in 39b.

7.4 Testing

⟨ Testing 41b ⟩ ≡

```
check:  $(PROGRAMS)
        perl -c gardol.pl
        perl -c badbot.pl
        weblint index.html

test:   $(PROGRAMS)
        perl gardol.pl --verbose $(LOGDIR)/forensic_log $(LOGDIR)/ipfilter_log

weblint:    gardol.pl
        weblint index.html
```

◇

Macro referenced in 39b.

7.5 Installation

⟨ Installation 41c ⟩ ≡

```
publish:    dist
        cp -p index.html gardol.tar.gz gardol.pdf $(WEBDIR)
        cp -p figures/* $(WEBDIR)/figures
```

◇

Macro referenced in 39b.

Chapter 8

Indices

Three sets of indices can be created automatically: an index of file names, an index of macro names, and an index of user-specified identifiers. An index entry includes the name of the entry, where it was defined, and where it was referenced.

8.1 Files

8.2 Macros

⟨ IP Filter log parsing pattern 19a ⟩ Referenced in 17b.
⟨ Installation 41c ⟩ Referenced in 39b.
⟨ Introductory text 30b ⟩ Referenced in 29.
⟨ Log file parsing patterns 17b ⟩ Referenced in 15c.
⟨ Loop command handler 36b ⟩ Referenced in 33.
⟨ Open log files to be monitored 8a ⟩ Referenced in 7, 9a.
⟨ Options documentation 23abcd, 24abc, 25abc, 26abc, 27 ⟩ Referenced in 22.
⟨ Parse HTTP log item into variables 10 ⟩ Referenced in 9b.
⟨ Perl directory 2a ⟩ Referenced in 7, 33.
⟨ Perl language modes 15d ⟩ Referenced in 15c, 33.
⟨ Post command handler 35c ⟩ Referenced in 33.
⟨ Print command line help information 21 ⟩ Referenced in 20b.
⟨ Process command line options 19b ⟩ Referenced in 7.
⟨ Questions to answer 31 ⟩ Referenced in 29.
⟨ Quick reject non-attack HTTP log items 5a ⟩ Referenced in 9b.
⟨ Reject non-attack HTTP log items 6 ⟩ Referenced in 9b.
⟨ Reject the current log item as benign 5c ⟩ Referenced in 5a, 6, 9b.
⟨ Release Date 2c ⟩ Referenced in 19b, 21, 22, 29, 36a.
⟨ Remove timed out hosts from list of attackers 14b ⟩ Referenced in 12a.
⟨ Restart IP Filter Command 3a ⟩ Referenced in 20a, 25a.
⟨ Set command handler 37a ⟩ Referenced in 33.
⟨ Source distribution 40b ⟩ Referenced in 39b.
⟨ Start monitoring for restart signal 8b ⟩ Referenced in 7.
⟨ Template 15b ⟩ Not referenced.
⟨ Testing 41b ⟩ Referenced in 39b.
⟨ Transcribe template to IP Filter configuration file 12b ⟩ Referenced in 12a.
⟨ URL Fetch command 34a ⟩ Referenced in 34b, 35ac.
⟨ Update filter to block attacking hosts 12a ⟩ Referenced in 7.
⟨ Utility functions 20b ⟩ Referenced in 7.
⟨ Validate option specifications 20a ⟩ Referenced in 7.
⟨ Verbose command handler 37b ⟩ Referenced in 33.
⟨ Version 2b ⟩ Referenced in 19b, 21, 22, 36a.
⟨ Wait command handler 38a ⟩ Referenced in 33.
⟨ Web Directory 2d ⟩ Referenced in 39b.
⟨ Web Log File Directory 2e ⟩ Referenced in 39b.

8.3 Identifiers

Sections which define identifiers are underlined.

Chapter 9

Development Log

2004 March 1

Created Nuweb `gardol.w` from initial hacked version of Perl program.

2004 March 3

Added command line options to override all internal configuration variable settings.

2004 March 5

Removed hard-coded pattern definition for HTTP log parsing. The patterns for parsing common, combined, and forensic log formats are now defined as regular expression variables, and a configuration variable, `$parseHTTPlog`, is set to the pattern to be used. Similarly, the pattern used to parse the IP Filter log is now defined by a variable `$parse_IP_Filter`.

Integrated logo for Web page and first cut of section headings for the Web document.

Debugged distribution archive creation and the "`publish`" target in the `Makefile` which creates the Web tree.

Integrated manual-page template documentation in POD format, included at the head of the emitted `gardol.pl` file.

2004 March 10

Added a `--blacklist` facility, which permits specifying a file containing IP addresses (or more general IP Filter "`from`" expressions) which are unconditionally copied into the block list. The blacklist is re-read from the file every time the filter list is updated, so it can be changed at any time. An error is reported if the `--blacklist` option is specified and the blacklist file does not exist, an error is reported, but should the blacklist file not exist at the time of an update, it will simply be ignored. This allows users to rename blacklist files to switch them on and off and/or swap blacklists without causing an error.

2004 March 12

Added code to catch the `HUP` signal and, after receiving it, the next time around the main monitoring loop close, re-open, and re-seek the log files to the end. This allows following log files are they are cycled by renaming and restarting the programs that write them.

The current build is now in production on Vitesse. It's run from a shell script which specifies all the file locations. There is no need for `$prime_time` to override built-in path name defaults, so it has been removed.

Added the ability to transcribe arbitrary IP Filter rules from the `--blacklist` file to the filter definition. Any line whose first nonblank character is a plus sign is transcribed explicitly, dropping the plus sign. No syntax or other checking is performed–the text is simply copied into the filter configuration file.

Changed the default for `--ipfupdate` to the null string, disabling automatic update. It's better not to have reloading the IP filter a built-in default.

2004 March 13

The command line option documentation in the POD document was getting unwieldy, so I broke it up into one section per option. This results in a nice list of options in the table of contents, which will be linked in the PDF.

Broke the `Makefile` up into logical sections of targets for build, installation, documentation, test, etc. Added an explanation to the build target about how we use `gardol.pl` as a proxy for all files generated from `gardol.w` and the quirk about having to make twice before a change in the `Makefile` takes effect.

The "`--Timeout`" diagnostic was printed on standard error unconditionally. Since it is redundant given the "`--Timeout purged`" message issued in `--verbose` mode, I just removed it.

2004 March 13

Added "`-`" as a default argument for the `--ipfupdate`, `--ipfconf`, `--ipftemplate`, and `--ipfupdate` options. The default argument selects the file names configured as Operating System Default Parameters (2.6).

2004 March 15

Added the BadBot attack simulator program. Cleanup and more documentation to follow, as always.