Last update: 01 Apr 2009

Grumpy web & gopher server's Manual

Written by Mateusz Viste

http://www.viste-family.net/mateusz/grumpy/

Table of Contents

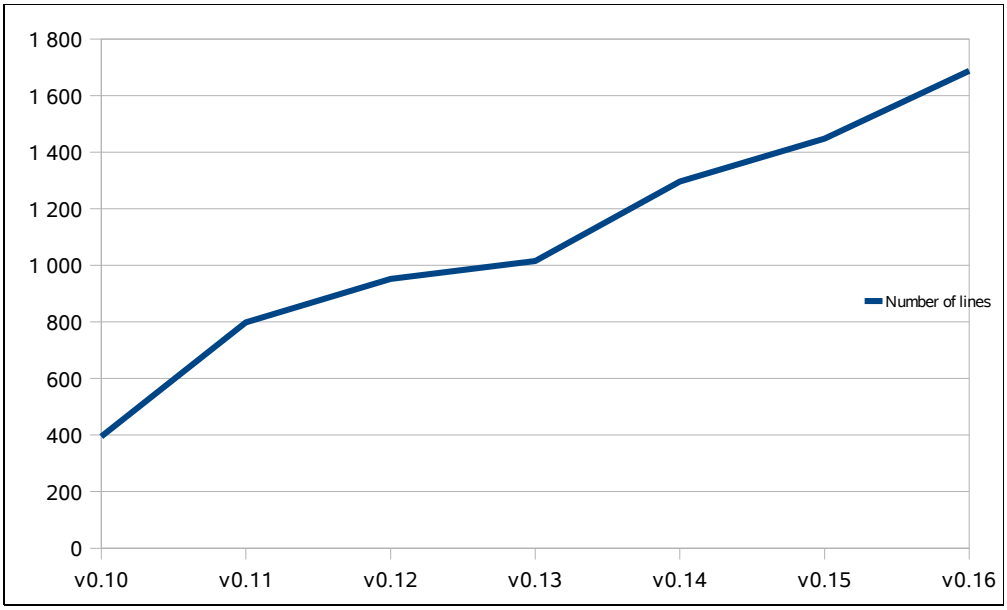
Introduction

Grumpy is a simple, easy to install, open-source web and gopher server for Linux and Windows. Grumpy is not a standalone daemon - it requires an inetd-compatible superserver to work.

Why one would prefer Grumpy over any other Linux web server, like Apache, Lighttp, Jigsaw...? Well, I wrote Grumpy primarily for fun, but it appears to have become a rather strong web server with some very good points: easy to install, lightweight, secure (as it doesn't support any dangerous features)...

The Grumpy web server is meant to be used for small projects, like home servers, some intranet applications, small-sized companies, etc. All the configuration is done via a single configuration file, which has very reasonable defaults. That makes Grumpy easily maintainable, and allows the administrator to have a full knowledge of what features are allowed/enabled on the server, and what's not. Grumpy supports CGI applications, may act as a gopher server, and is entirely written in FreeBASIC.

Grumpy is a small project, with very few developers (to tell you the truth, I'm the only one), but it keeps growing, and gets improved continuously. Below, you will see a cool graph which presents the project's growth over the time.



...Yeah, I hadn't anything better to put there. :-)

Installation (inetd)

Installing Grumpy on a Linux host is very easy. On the first place you will have to copy grumpy to /sbin or /usr/sbin, and grumpy.cfg to /etc. As mentioned previously, Grumpy needs an inetd-like superserver to work. If using the basic inetd, you will have to add the following line to /etc/inetd.conf:

```
www   stream      tcp   nowait      www-data    /sbin/grumpy      grumpy
```

Important things here are: "www", which is the name of service to bind to. You will have to check if the www service has its entry in /etc/services. "www-data" is the name of the user which has to be used to run the grumpy process. Never use root to run grumpy (or any other public daemon)! Obviously, you will have to check if the www-data user exists in your system (in Debian it exists by default), and let him write to /var/log/grumpy.log, read from /etc/grumpy.cfg and execute /sbin/grumpy. The easiest way to do that is simply to run "chown www-data file" on each file. When the inetd configuration is done, you'll have to restart the superserver (or the whole machine). In Debian, restarting inetd may be done by typing "kill -HUP `cat /var/run/inetd.pid`".

Don't forget to put an index.htm (or index.html) file into your RootDir path (by default, the RootDir is /var/www/). Note, that you don't need to use any index file if you allow to browse directories (AllowDirListing=1 in /etc/grumpy.cfg).

Installation (xinetd)

If your system is running xinetd, the first steps will be similar to the inetd case: on the first place you will have to copy grumpy to /sbin or /usr/sbin, and grumpy.cfg to /etc. The behavior of xinetd is very similar to inetd. However, the configuration file has a different syntax. Some distribution has an unique configuration file for xinetd, others uses separate files for each service (these files are usually in /etc/xinet.d/). In any case, you will need to add the such (or similar) lines to let xinetd know about Grumpy:

```
service www
{
  disable = no
  socket_type = stream
  protocol = tcp
  wait = no
  user = wwwrun
  server = /sbin/grumpy
  instances = 100
  per_source = 10
  log_type = FILE /var/log/xinetd-grumpy.log
  log_on_success = HOST PID DURATION
  log_on_failure = HOST
}
```

Important things here are: "service www", which is the name of service to bind to. You will have to check if the www service has its entry in /etc/services. "wwwrun" is the name of the user which has to be used to run the grumpy process. Never use root to run Grumpy (or any other public daemon)! However, you may want to use it for troubleshooting purpose. Obviously, you will have to check if the wwwrun user exists in your system, and let him write to /var/log/grumpy.log, read from /etc/grumpy.cfg and execute /sbin/grumpy. The easiest way to do that is simply to run "chown wwwrun file" on each file. When the xinetd configuration is done, you'll have to restart the superserver (or the whole machine). In most distributions, restarting xinetd may be done by running "/etc/init.d/xinetd restart". For more details on available features, see the xinetd manual.

Don't forget to put an index.htm (or index.html) file into your RootDir path (by default, the RootDir is /var/www/). Note, that you don't need to use any index file if you allow to browse directories (AllowDirListing=1 in /etc/grumpy.cfg).

Installation (Windows)

Although being primarily written for Linux, Grumpy is available for the Windows platform, too. I wouldn't recommend you to use Grumpy (nor any server application) on the Windows platform, but it's obviously up to you to make the choice. The Windows package of Grumpy contains all things which are required to run the http and gopher servers (executable file, an inetd-like wrapper, etc). All you have to do is launch one of the batch files "start-http.bat", "start-gopher.bat" (or both). These batch files will execute the inetd wrapper, binding Grumpy to a network socket.

As explained in the "Authentication" part of this manual, the Grumpy HTTP server requires a ".grumpy.auth" to be created in the directory which has to be password-protected. You will quickly notice, that Windows Explorer doesn't allow to create files beginning by a dot. That's why you will have to use any kind of trick to create the file. One way would be to run the "echo. > .grumpy.auth" command from within the command-line.

After being started, the Grumpy server will listen on the port 80 (http) and/or 70 (gopher). If you would like to check whether it's active or not, you can simply type the "http://127.0.0.1/" address in your web browser (or "gopher://127.0.0.1/" to test the gopher server).

Keep in mind, that Windows is not case-sensitive when it comes to handling file names. Therefore, a "http://server.net/file.html" request will return the same ressource than "http://server.net/FILE.HTML" (there are big chances that you don't mind anyway).

Because Grumpy cannot share the same port with another TCP/IP application, you may need to stop or uninstall certain services first. These include (but are not limited to) other web servers, and firewall products such as BlackIce. Also, you should double-check your firewall's configuration, to avoid any filtering related trouble.

Configuration file

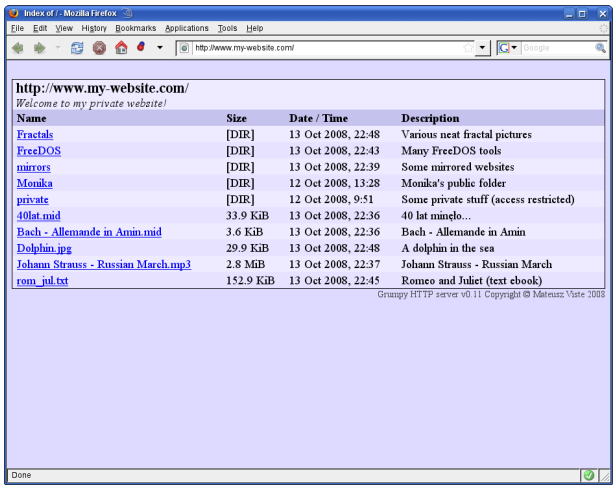
The Grumpy's configuration file should be at /etc/grumpy.cfg (Linux) or in Grumpy's directory (Windows), and made readable by the user which will run Grumpy. It's a plain-text file, containing some tokens with values. Any line beginning by the "#" character is ignored. If, for some reasons, Grumpy would be unable to access/read his configuration file, he will take default values. Here's an example configuration file:

```
#####################################################
# Configuration file for the Grumpy web server #
#####################################################

## Set the loging verbosity ##
#  0 - No loging (not recommended, unless you really don't care about logs)
#  1 - Log basic informations, like Request code/answer code. (Default)
#  2 - Maximum verbose (log full requests + a summary of the answer)
Verbose=1

## Allow/Disallow listing of directories ##
# If you allow directory listing, then Grumpy will list the content of
# directories which doesn't have any index.htm or index.html file. You will
# probably want to enable it, unless you are paranoid.
# 0 - Disabled (default)
# 1 - Enabled
AllowDirListing=0

## The server's root directory path. ##
# Default is /var/www
RootDir=/var/www

## QoS bandwith limitation ##
# This setting allows you to limit the transfer rate per file (in KiB/s).
# Eg. "QoS=5" means that Grumpy will serve files at a rate of max. 5 KiB/s.
# Note, that the QoS setting is set per file (an user could download one file
# at 5 KiB/s, two files at 10 KiB/s, etc...).
# 0 disables the QoS limitation (so Grumpy will serve files as fast as he can).
# Default is 0.
QoS=0

## Support for CGI applications. ##
# 0 - Disabled (default)
# 1 - Executes CGI, but only if the script has the *.cgi extension and is in
#     the /cgi-bin/ subdirectory
# 2 - Executes any file which have the *.cgi extension
#
# The CGI support in Grumpy is not standard. Read the Grumpy manual first.
# WARNING: AS ANY OTHER SERVER-SIDE CODE EXECUTION, THE CGI SUPPORT MAY
#          BE *VERY* DANGEROUS IF USED UNCORRECTLY. ENABLE IT ONLY IF YOU
#          KNOW EXACTLY WHAT YOU ARE DOING!
CgiSupport=0
```

```
## Virtual Hosts configuration ##
# Grumpy supports the use of virtual hosts. Any virtual host has to be
# declared below. The syntax is the following:
# vhost:my_hostname=my_directory
#
# Eg. vhost:mywebsite.com=/var/www/website/ would declare a virtual host
# called "mywebsite.com", and will redirect any requests to that host
# to the directory /var/www/website/.
#vhost:mywebsite.com=/var/www/webiste/

## Redirect domains ##
# Here you may declare any redirection for a given domain. It's particularily
# useful if you have moved your website to another domain name and wants to
# keep the content available for users (note, that crawling robots will follow
# such redirects, too). A domain redirect will generate a 301 HTTP response for
# any request destinated to the redirected domain.
# Syntax: Redirect:OldDomain=NewDomain
# Example:
# RedirectDomain:MyOldDomainName.net=www.MyNewFlashyDomainName.com

## Custom HTTP headers ##
# Below you may define few custom header. If set, these headers will be sent
# with any HTTP response to the client. If you leave them empty, they won't
# be sent.
# Example:
# x-powered-by=The Grumpy http & gopher server, (C) Mateusz Viste
x-powered-by=
x-hosted-by=
x-server-admin=
x-disclaimer=

## Banned User-Agents ##
# There you can specify some User-Agents you definitely do not want to
# accept HTTP requests from. Use that only for banning purpose, as the
# Grumpy server will be very impolite to requests comming from an
# user-agent specified below: it will make him wait for 2s, and answer
# by a 403 (Forbidden) code. All user-agents have to be delimited using
# a pipe character (|). Note, that Grumpy will check if the specified
# string is contained in the declared user-agent, therefore be careful
# when adding anything (for example, adding "net" there would make Grumpy
# deny requests from any User-Agent which contains "net" in his name).
BannedUserAgents=Nmap|libwww-perl|Morfeus|Toata dragostea|Brutus|DataCha0s

## Gopher configuration ##
# There comes the Gopher configuration. If you don't know what Gopher is,
# then don't bother about this section - you don't need it.
# The Gopher engine implemented into Grumpy needs to know three things:
# GopherRoot - that's the local path to Gopher ressources.
# GopherHostname - The public hostname the gopher server is available at.
# GopherPort - The port on which the public Gopher server listens on.
GopherRoot=/var/gopher/
GopherHostname=gopher.mydomain.net
GopherPort=70
```

Directory listing

Grumpy is able to list directories content on his own. That is, if there's no index.html nor index.htm file in the given directory, and you enabled directory listing in the configuration, then Grumpy will happily display an index, listing all files and subdirectories. Of course, if the directory requires authentication, Grumpy will ask for credentials first. Note, that the listing won't display any hidden ressource (ie file or directory beginning by a dot). It won't display the "descript.ion" file either.



I guess you are wondering now, what's that « descript.ion » file for? Well, it's a text file which contains any descriptions of the directory content. Grumpy will use it (if found) to fill the "Description" column of the listing. The structure of that file is very basic. Each entry has to be stored in one line: element[tab]description. It's very important to keep in mind that the delimiters are TABs, as it won't work with simple spaces (that's because you may have filenames containging spaces, so it would be confusing to allow spaces as delimiters). Here's a short example of a "descript.ion" file:

```
.     This directory contains some pictures of my dog
summer2008  Here are many photos from our summer 2008 holiday
ingrass.jpg Doggy in the grass
beach.jpg   Swimming dog
dream.jpg   Doggy felt asleep
```

The entry "." is describing the current directory (this description would appear in the bottom of the listing page). An important (well, maybe not THAT important, but rather "good to know") restriction of the "directory listing" feature is the fact, that Grumpy will display only the first 20'000 entries (not the first alphabetically, but rather the first in disk entries order). Therefore, if you have more than 20'000 elements (files and/or directories) in a single directory, you will probably prefer to build an index by yourself. Anyway, Grumpy will then display a big red warning stating that there are too many files to display.

HTTP Authentication

Basically, two authentication methods exist in HTTP/1.1: the "basic" method, where the username and password are transmitted in plaintext, and the "digest" method, where the username and password are replaced by their MD5 hash. Grumpy supports only the first one (don't worry, the digest one isn't much more secure anyway).

How does it work?

When a particular resource has been protected using basic authentication, Grumpy sends a 401 Authentication Required header with the response to the request, in order to notify the client that user credentials must be supplied in order for the resource to be returned as requested.

Upon receiving a 401 response header, the client's browser, if it supports basic authentication, will ask the user to supply a username and password to be sent to the server. If you are using a graphical browser, such as Netscape or Internet Explorer, what you will see is a box which pops up and gives you a place to type in your username and password, to be sent back to the server. If the username is in the approved list, and if the password supplied is correct, the resource will be returned to the client.

Because the HTTP protocol is stateless, each request will be treated in the same way, even though they are from the same client. That is, every resource which is requested from the server will have to supply authentication credentials over again in order to receive the resource. Fortunately, the browser takes care of the details here, so that you only have to type in your username and password one time per browser session - that is, you might have to type it in again the next time you open up your browser and visit the same web site.

Along with the 401 response, certain other information will be passed back to the client. In particular, it sends a name which is associated with the protected area of the web site. This is called the realm, or just the authentication name. The client browser caches the username and password that you supplied, and stores it along with the authentication realm, so that if other resources are requested from the same realm, the same username and password can be returned to authenticate that request without requiring the user to type them in again. This caching is usually just for the current browser session, but some browsers allow you to store them permanently, so that you never have to type in your password again.

How to set up authentication?

All you have to do is put a ".grumpy.auth" file in the directory which access to has to be restricted (that file will contain any authorized credentials). Note, that Windows Explorer won't let you create a file beginning with a dot – if you run Grumpy on Windows, you will have to use some tricks to do it, like creating the file from the command-line, with the command "echo. > .grumpy.auth".

Such .grumpy.auth file would have the following structure (any line beginning by a "#" character is ignored):

```
# Authentication file for Grumpy
realm=Password, please!
John:blahblah
Kim:white kitty
```

The realm token defines the realm (surprising, isn't it?). Usually the user's browser will display the realm in the popup window asking for credentials. Then, we have the list of users (with their passwords) which are allowed to access the given location. The syntax is very simple: "user:password". From the example above, we see that the users allowed to access the location are "John" and "Kim". John's password is "blahblah", while Kim's is "white kitty". Note, that the system is case-sensitive! A single ".grumpy.auth" file restricts the access to its own directory, as well as any subdirectories (unless these subdirectories have their own ".grumpy.auth" files). Of course, the ".grumpy.auth" file is not retrievable via HTTP nor gopher. If someone request it, Grumpy will output a 404 error. Take care to not leave any (renamed) copy of the file ".grumpy.auth", as it would be readable by anyone!

Brute-force protection

The HTTP authentication scheme is by nature vulnerable to brute-force attacks. The attacker would just have to try various combinations of logins and passwords the fastest he is able to. Grumpy is aware of such nasty behavior, and apply a simple (but efficient) protection against that issue: any request asking for a protected content and coming with false credentials will be responded with a delay of 2 seconds. Note, that 2 seconds is a very long time to wait for a brute-force attacker.

Attention:

The password is passed from the client to the server in plain text across the network. Anyone listening somewhere in the packet's way with any variety of packet sniffer will be able to read the username and password in the clear as it goes across.

Not only that, but remember that the username and password are passed with every request, not just when the user first types them in. So the packet sniffer need not to be listening at a particularly strategic time, but just for long enough to see any single request come across the wire.

And, in addition to that, the content itself is also going across the network in the clear, and so if the web site contains sensitive information, the same packet sniffer would have access to that information as it went past, even if the username and password were not used to gain direct access to the web site.

Is there any solution to these security risks? Of course! The big trouble here is the possibility to capture clear traffic directly from the wire. The ultimate solution is simply to encipher your http traffic into SSL packets. That's what we commonly call "HTTPS" (that is, HTTP over SSL). Read the HTTPS-related chapter of this manual to discover how to turn Grumpy into a secure HTTPS server.

CGI support

Grumpy supports CGI application, although it covers only a limited amount of functionalities. The only supported CGI programs are applications which output text data (binary data won't be interpreted properly by the Grumpy's CGI parser). Fortunately, CGI scripts outputting binary stuff are rather rare. Note, that CGI support is not available on the Windows port of Grumpy.

The CGI standard defines few ways of providing input data to the CGI application: the GET and POST methods. The GET method transfers data via URL parameters, thus the amount of data is limited (typically no more than 256 bytes). The POST method is more efficient, and transfers data as a response's payload. Grumpy supports only the GET method (on the other hand, it makes him more secure).

Let's see how does CGI work.

Each time a client requests the URL corresponding to your CGI program, the server will execute it in real-time. The output of your program will go more or less directly to the client.

The web server (in our case Grumpy), may provide some informations to the CGI application - either by launching it with some command-line parameters, or setting some environment variables. Basically, if the client's request doesn't contain any (non-encoded) "=" character in the URL, then all URL parameters will be used as command-line parameters of the CGI script. For example, a "GET /cgi-bin/myscript.cgi?param1&param2" request will let Grumpy execute "myscript param1 param2". If any "=" character is found in the request, then no command-line parameters are provided. In any case, several environment variables may be set:

QUERY_STRING	The URL parameters, as provided by the client
SERVER_SOFTWARE	The name and version of the server software
SERVER_NAME	The server's hostname, DNS alias, or IP address, used for self-referencing links
GATEWAY_INTERFACE	The revision of the CGI specification, as supported by the server
HTTP_USER_AGENT	The client's user-agent
SCRIPT_NAME	Script name (for self-referencing links)
REQUEST_METHOD	The request method (GET, HEAD, POST...)
SERVER_PROTOCOL	The HTTP protocol version of the client's query
HTTP_VERSION	Same as SERVER_PROTOCOL
HTTP_COOKIE	The client's cookie
HTTP_REFERER	The HTTP referer
AUTH_USER	The authenticated user (if there was any authentication)
REMOTE_USER	Same as AUTH_USER
AUTH_TYPE	The authentication type used to authenticate the user (BASIC, DIGEST...)

When it comes to answer to the client, the CGI application will output a (partial) HTTP header, which should contain at least the "Content-Type" statement. This header will be followed by an empty line (which tells to the server that the header is over), and then will come the document which has to be transmitted to the client. Grumpy will analyze the header transmitted by the CGI application and will complete it with all lacking informations (the status code, HTTP version, etc).

As we just said, Grumpy is parsing the HTTP header returned by the CGI applications, and completes it when necessary. But there are situations where such behavior is unwanted (for example if the script undertakes to print the entire HTTP response including all necessary header fields.). That's what we call NPH scripts (NPH stands for "No Parsed Headers"). Grumpy is supporting such CGI applications - all we have to do, is name these particular programs starting with "nph-" (like "nph-myscript.cgi"). Grumpy is thereby instructed not to parse the headers (as it would normally do) nor add any which are missing. If you are going to use NPH, be sure to read and understand the HTTP spec (http://www.w3.org/Protocols/). Your headers should be complete and accurate, because you're instructing the Grumpy web server not to correct them or insert what's missing.

Here are some useful CGI references:

- The Common Gateway Interface <http://hoohoo.ncsa.uiuc.edu/cgi/>
- CGI Made Really Easy <http://www.jmarshall.com/easy/cgi/>
- CGI and Environment Variables <http://www.cs.cf.ac.uk/Dave/PERL/node200.html>

Virtual Hosts configuration

Virtual hosting is a method that servers such as web servers use to host more than one domain name on the same computer, often on the same IP address. For example, it is often desirable for companies sharing a web server to have their own domains, with websites accessible as www.company1.com and www.company2.com, without requiring the user to know any extra path information. For www.company1.com, the server would send the HTML file from the directory /var/www/user/Joe/site/, while requests for www.company2.com would make the server serve pages from /var/www/user/Mary/site/.

With web browsers that support HTTP/1.1 (as nearly all now do), upon connecting to a webserver, the browsers send the address that the user typed into their browser's address bar (the URL). The server can use this information to determine which web site, as well as page, show to the user.

Okay, but how do I configure Grumpy to act such smart way? Well, you will be surprised to see how easy it is! The only thing to do, is add a vhost entry to your grumpy.cfg configuration file. There comes an example:

```
vhost:www.mysite.com=/var/www/mysite/
```

That short entry is telling Grumpy to look into the /var/www/mysite/ directory for any request addressed to www.mysite.com. Of course, any other requests will still be answered using the default RootDir directive. You may add several vhost entries into your configuration file.

Domain redirection

If one wants to permanently forward an entire web site to a new server (or hostname) permanently and have the search engines update their database, one should use a 301 redirect. The 301 redirect code is the most efficient and search engine friendly method for webpage redirection. The code "301" is interpreted as "moved permanently". So much for the theory.

Let's say, that you are using the domain www.blahblah.org. One day, you want to change your domain name to www.mumbo-jumbo.net. You will definitely get in troubles, as all people who bookmarked your site will still come to the first domain, and search engines will do the same. There Grumpy comes to your help – it allows you to configure a redirection for an entire domain. It'll be done using a directive of the following form:

```
RedirectDomain:www.blahblah.org=www.mumbo-jumbo.net
```

With such directive, any request addressed to www.blahblah.org will be redirected to www.mumbo-jumbo.net. Note, that the full URL will be preserved during the redirection! For example, a client requesting http://www.blahblah.org/image/pic.png will be redirected (via a 301 code) to http://www.mumbo-jumbo.net/image/pic.png. The whole process is completely transparent for the end-user.

Another cool way of using domain redirects is setting a redirect for your domain name to your www host. That's how I did it for my domain:

```
RedirectDomain:grumpy-server.net=www.grumpy-server.net
```

This way any user coming to grumpy-server.net will be automatically switched to www.grumpy-server.net.

Obviously, you can't redirect a single page with that method, as it's applied to the whole domain. If you would like use a similar mechanism to redirect clients coming to a specific page of your web site, I would recommend you to put such code in the <head> section of your html file:

```
<meta http-equiv="Refresh" Content="0; URL=http://www.company.com/dir1/">
```

This commands the browser to refresh the page immediately with the new specified URL. It forwards a single page only and not the entire domain. It can forward the default home page of the domain, giving the appearance of forwarding the domain.

HTTP headers customization

First of all, what a HTTP header is? HTTP Headers form the core of an HTTP request, and are very important in an HTTP response. They define various characteristics of the data that is requested or the data that has been provided (data encoding, caching, authentication...). The headers are separated from the request or response body by a blank line. HTTP headers are invisible to the final user, as the user will see the document transfered in the HTTP body, without caring about the way it came in.

There is a simple example of a HTTP response:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Grumpy/0.17
Connection: close
```

Then, why one would want to customize anything there? To tell you the truth, there's no "practical" point in doing that, as the vast majority of users won't ever notice it anyway. However, it might be a cool information for people looking to the transferred data at the HTTP level (yes, it happens to me very often).

Grumpy lets you define few custom headers: X-Powered-By, X-Hosted-By, X-Server-Admin, X-Disclaimer. For example, a customized server response could look like:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Grumpy/0.17
Connection: close
X-Disclaimer: The administrator of this server is not aware of its content!
```

To make Grumpy answer with such "X-Disclamer" header, you would have to add the following directive to the configuration file (grumpy.cfg):

```
x-disclaimer=The administrator of this server is not aware of its content!
```

Last, but not least, please keep in mind that the RFC 2616 specify that HTTP headers shouldn't contain characters out of the ISO-8859-1 set, unless specifically encoded.

Customizing error pages

Grumpy does support customization of the returned error pages. The following error pages are customizable: 400, 401, 403, 404. By default, when one of these errors occurs, Grumpy returns a 4xx code with a html page containing a bare explanation of what happened. You can replace these defaults page by anything you like, by creating a directory called ".errors" in the root directory of your web server (or the root of your virtual host server if you are using the virtual host feature), and place one or several files named "400.htm", "401.htm", "403.htm" and "404.htm". If you're using virtual hosts, then each virtual host will use a distinct set of custom error pages.

Note, that if you are going to use any pictures or links on your custom error pages, you will need to use full (absolute) paths only (relative paths won't work, as the remote browser won't be aware of the existence of the ".errors" subdirectory).

Example

Say, you would like to change the default 404 error page to something nicer than just "Not found". We assume that your root path is /var/www/. You will have to make your custom 404 html page first, name it "404.htm" and put it into the directory /var/www/.errors/ (you will probably have to create the ".errors" directory before). That's all!

Banning specific user-agents

You have probably already noticed, that when a web server is put online, there are many suspicious requests coming to it. These requests are often generated by some web spiders, spam email-collectors, etc. These requests often contains a specific HTTP header called "User-Agent", which is meant to declare what software has forget the request.

Grumpy is able to block (ban) any chosen User-Agent. In its default configuration, Grumpy bans some HTTP security scanners. If you would like to modify the list of banned User-Agents, you will have to edit the "BannedUserAgents" parameter of the configuration file. That option takes as a value a list of User-Agents to ban, separated by pipe characters ("|"). Say, you would like to ban any User-Agent containing words "Microsoft" or "IE 6". All you would have to do, is write the following line in the grumpy.cfg file:

```
BannedUserAgents=Microsoft|IE 6
```

Be careful when adding banned user-agents, as Grumpy is checking if the specified string is contained in the user-agent header declared by the remote machine. It means, that adding "a" as a banned user-agent would ban all user-agents containing the letter "a" (like "Mozilla", "Opera", "Arachne", etc...). Banned user-agents are checked in a case-insensitive way.

The Grumpy's banning behavior is quite impolite, therefore you shouldn't use it for any other purpose than banning a dangerous or ill-intentioned user-agent! Technically speaking, when Grumpy gets a request coming from a banned user-agent, it makes the remote machine waiting for 2 seconds, sends a 403 error code (403 means "Forbidden"), and closes the connection.

Attention: Banning user-agents is not a protection of any kind, and shouldn't be used as such! Banning system will probably discourage most of script kiddies, and save a bit of your bandwidth, CPU processing power and HDD activity. Don't expect anything more.

Turn Grumpy into a HTTPS server

What is it all about?

Reading the title of this chapter, you probably though "Wow, does Grumpy really support SSL ciphering?". Well... the simple answer is "yes", although it is not technically true.

I won't explain here how does SSL works, nor how certificate are used to encipher and authenticate hosts, as these subjects are far beyond the scope of this manual. However, we will see how to create a SSL certificate using OpenSSL, and how to wrap Grumpy into an SSL communication.

First of all, we need to know (barely) what's the benefit of using SSL. We all know that HTTP communication is transferred as clear-text over the wires. That means that anybody who has a physical access to your wire, will be able to see what transit between you and the HTTP server you are connected to. Obviously, that includes any passwords you could provide to authenticate yourself. The solution to this problem is quite simple: use an end-to-end encryption mechanism, which will transport your HTTP traffic (in this case, the encryption mechanism will be SSL). Of course, if you don't care about that (eg. you host a website with public accesses only), then you won't have much benefits from SSL (apart authenticating the server to avoid any possible spoofing, but again – it won't be discussed here). You probably already guessed, that HTTP encrypted over a SSL tunnel is what we commonly know as "HTTPS".

Here we are again: How do I turn Grumpy into a HTTPS server? As said before, Grumpy itself doesn't provide any SSL support, as it works on clear-text communication only. However, we can use any SSL wrapper to tunnel Grumpy through a secure (enciphered) channel. For the needs of this manual, we will discover how to configure a HTTPS server using Grumpy, the Stunnel wrapper, and the OpenSSL suite in a Linux environment (really, it is not as hard as it sounds).

Install the Stunnel wrapper

There's no general guide-line for installing Stunnel on your server system. It will mostly depends of your operating system. You may install it from sources, or apply the Stunnel package, or copy the required binaries, etc... If you're using a Debian Linux operating system, you can easily install the whole Stunnel package using the command below:

```
apt-get install stunnel
```

On other systems, you will probably have equivalent commands. Check your system's handbook for details.

Generate your own SSL certificate

SSL is based on a public key infrastructure, which means that we will need a pair of keys – a private one, and its public equivalent. Sorry for this techy language... it's not so important anyway

– all you have to remember, is that you will need a certificate file, which will contain your keys, plus some other informations (Diffie-Hellman parameters...). Before starting, you will have to check whether your system has an OpenSSL installation or not (it's enough to check if you get something after typing the "openssl" command from within your shell). If not, install it now.

To generate a valid SSL certificate containing both your private and public key, you can use a command of the following syntax:

```
openssl req -new -x509 -days 365 -nodes -config stunnel.cnf -out stunnel.pem
-keyout stunnel.pem
```

This creates a private key, and a self-signed certificate. The arguments mean:

-days 365	make this key valid for 1 year, after which it's not to be used anymore (you may want to increase this value)
-new	Generate a new key
-x509	Generate an X509 certificate (self sign)
-nodes	Don't put a password on this key (otherwise you would have to manually type the certificate's password at each Stunnel call)
-config stunnel.cnf	the OpenSSL configuration file to use (you will find it somewhere in your Stunnel installation)
-out stunnel.pem	where to put the SSL certificate
-keyout stunnel.pem	put the key in this file (keep the same than above)

The OpenSSL generator will ask you few questions, and one of them will be to give a "Common Name" (or CN) to your certificate. This is quite important, as this parameter is carefully checked by all modern web browsers. The "Common Name" has to be exactly the machine's host name, as used by the final client to access your server (in most cases it will be its FQDN, like "www.mysecuresite.net", but it could be also its IP, if clients will access the site typing the site's IP in their URL bar). If the certificate's Common Name would not match the FQDN used by the remote client to reach your site, the user's browser will display a warning, telling that the certificate presented by your server is wrong.

Note, that Stunnel will often need some DH parameters, too. DH stands for "Diffie-Hellman", it is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. These DH parameters have to be in the same file than the server's certificate. To generate DH parameters for our freshly created certificate, we will use the following command:

```
openssl gendh 512 >> stunnel.pem
```

It may happen that your specific installation of Stunnel doesn't require DH parameters, but it won't harm to have them in the file anyway.

If you would like to know how to generate a more specific certificate (choose the algorithm,

key length, etc..), go read the OpenSSL documentation.

The procedure explained above will provide you with a custom, self-signed certificate. Anyone can make a self-signed certificate. It is a totally valid SSL certificate. However most SSL clients (browsers) wish to verify the identity of the organization that signed the certificate. These SSL clients often have a hard-coded list of organizations (Certificate Authorities) that sign keys after doing background checks, etc.

Since the key and certificate you just generated are not in the hard-coded list that your SSL client uses, you will get either an error or warning message when attempting to connect to your HTTPS website. If you wish to interact with third-party clients that have hard coded lists of acceptable Certificate Authorities, and you do not want annoying dialog boxes popping up for the user on the first (or all) connections, then yes, you will have to have your key signed by a valid Certificate Authority. Unfortunately, it won't be free. It won't be cheap either.

Configure the HTTPS access

Once you installed Stunnel, and generated an appropriate certificate for your server, you have to configure your system to listen on a specific port and forward any incoming connections to Grumpy, passing first through the Stunnel wrapper. There are several ways to achieve that, I will present one of them, which is using the xinetd super-server.

In fact, configuring xinetd to use Stunnel is not harder than configuring a bare (HTTP) Grumpy server. The whole point is to tell xinetd to pass any streams coming to a specific port (usually, you will want to use the standard TCP/443 port) to STUNNEL, which will have to be instructed to forward the stream to Grumpy, assuring the encryption/decryption on the fly.

Here is a simple example of a xinetd section for a HTTPS service using Stunnel and Grumpy:

```
service https
{
    disable = no
    socket_type = stream
    protocol = tcp
    wait = no
    user = www-data
    server = /usr/bin/stunnel
    server_args = -l /sbin/grumpy -p /etc/ssl/certs/stunnel.pem
    instances = 100
    per_source = 20
    log_type = FILE /var/log/xinetd-grumpy-ssl.log 500K 10M
    log_on_success = HOST PID DURATION
    log_on_failure = HOST
}
```

Of course, you may have to adjust it depending of your system's configuration. Important things here are "https", "www-data" and (obviously) the paths to the Stunnel and Grumpy binaries, and your certificate file. "https" is the name of the service on which xinetd has to listen on (typically, "https" should be binded to the port 443 via your "/etc/services" file). "www-data"

is the user which has to be used to run the Stunnel instance. For testing purpose, you could use the root user, although it is not recommended to run any public service with root rights!

If you would like to have more details, go read the documentation of xinetd and/or Stunnel.

Why can't I use SSL with virtual hosts?

The reason is very technical, and a somewhat "chicken and egg" problem. The SSL protocol layer stays below the HTTP protocol layer and encapsulates HTTP. The SSL session is a separate transaction, that takes place before the HTTP session has begun. The server receives an SSL request on a specific IP. Since the SSL request does not contain any Host: field, the server has no way to decide which SSL certificate to use, because he would need to know the content of the "host" HTTP header. Obviously, this "host" header can't be provided until the SSL session is established... You can, of course, use Virtual Hosting to identify many non-SSL virtual hosts (all on port 80, for example) and then have a single SSL virtual host (on port 443).

Last thoughts

Keep in mind, that using SSL encryption is a very demanding (cpu-eating) process. Using SSL on top of Grumpy will slower your server's response times (latency), as well as its overall speed. My final advice would be "do not use SSL if you don't need it".

Besides that, please note, that the Grumpy server does not contain any cryptography itself. However, please remember that import and/or export of cryptographic software, code providing hooks to cryptographic algorithms, and discussion about cryptography is illegal in some countries. It is imperative for you to know your local laws governing cryptography. I am not liable for anything you do that violates your local laws.

Gopher server

Yes, Grumpy features an embedded Gopher server, too!

Gopher is a distributed document search and retrieval network protocol designed for the Internet. Its goal is to function as an improved form of Anonymous FTP, enhanced with hyperlinking features similar to that of the World Wide Web.

Setting up the Gopher support in Grumpy is trivial. There are three settings to configure in the grumpy.cfg file:

```
GopherRoot=/var/gopher/
GopherHostname=gopher.mydomain.net
GopherPort=70
```

Then, you will have to set up your superserver (inetd or xinetd...) to listens on your Gopher port (in most cases you will want to use the port TCP/70), and bind it to /sbin/grumpy, adding the "--gopher" parameter to grumpy. A xinetd configuration file could look like that:

```
service gopher
{
      disable = no
      socket_type = stream
      protocol = tcp
      wait = no
      user = wwwrun
      server = /sbin/grumpy
      server_args = --gopher
      instances = 10
      per_source = 2
      log_type = FILE /var/log/xinetd-grumpy.log
      log_on_success = HOST PID DURATION
      log_on_failure = HOST
}
```

If you are using the classic (oldish) inetd, you will have to add a line to your inetd.conf configuration file similar to that one:

```
gopher   stream   tcp   nowait    wwwrun   /sbin/grumpy    grumpy --gopher
```

Do not forget to check if the service "gopher" is properly declared in your /etc/services file. Note, that Grumpy will look at any "descript.ion" files (if found in the browsed directory) when serving Gopher content. Besides that, the Gopher protocol needs to describe the type of any resource. Grumpy is simply basing on the file's extension to assign a Gopher type to the resource. Below is a table containing all relations between gopher filetype and real file's extension (at least

that's the way Grumpy handles them):

Gopher type	Description	Files binded to this gopher type
0	Plain text file	*.txt
1	Directory listing	All directories
2	CSO search query	-
3	Error message	-
4	BinHex encoded text file	-
5	Binary (PC-DOS) archive file	-
6	UUEncoded text file	-
7	Search engine query	-
8	Telnet session pointer	-
9	Binary file	All files which doesn't fit into any other category
g	GIF image file	*.gif
h	HTML file	*.htm, *.html, *.gopherlink containing an "URL" selector
i	Informational message	-
I	Image file (other than GIF)	*.jpg, *.jpeg, *.png, *.bmp, *.pcx, *.ico, *.tif, *.tiff, *.svg, *.eps
s	Audio file	*.mp3, *.mp2, *.wav, *.mid, *.wma, *.flac, *.mpc, *.aiff, *.aac
P	PDF file	*.pdf
M	MIME encoded message	-
;	Video file	-

The Gopher protocol is handling linking to external resources in a very neat way: links are part of the protocol itself, not part of the document. That's why creating gopher links requires a special trick. For the purpose of gopher links, Grumpy uses files with the extension *.gopherlink. Let's say, we would like to put a link to a gopher site located at gopher://mygopher.server.net/1/myfolder. We would create a file (say, "link-to-my-server.gopherlink") with the following content:

```
Server=mygopher.server.net
Selector=/myfolder
Type=1
Port=70
Description=This is a link to my folder on my gopher server
```

The only parameter of the file which is really required is (obviously) "Server". All other parameters will be set to default values (no selector, type=1, port=70). If no description is provided in the gopherlink file, then the server's address will be used. Note, that you may add links to HTTP servers, too. For a link pointing at http://www.mydomain.com/stuff.htm, you'll have to use a very short gopherlink file, containing just the "Selector" token: "Selector=URL:http://www.mydomain.com/stuff.htm", and optionally a description. In this case, the "Server" token is not used.

Last, but not least, there are situations where you would like to have the absolute control on what (and how) the server will display a directory. That's why Grumpy is supporting gophermaps. If Grumpy finds a gophermap in a directory, then it doesn't check the directory content, and simply outputs to the user the content of the gophermap. A gophermap must be named "grumpy.gophermap", and must contain gopher entries as described in the RFC 1436. There's an example of a "grumpy.gophermap" file (don't forget about TABs!):

```
iWelcome to my gopher server!	fake	null	0
i	fake	null	0
0About my server	/about.txt	mygopher.domain.net	70
1Download	/download	mygopher.domain.net	70
1A link to a friend's server	friend.domain.net	70
hMy Website	URL:http://mywebsite.com
```

Note, that you may omit the server's address and server's port parts. Grumpy will then use his general settings.

Legal mumbo-jumbo

http://www.viste-family.net/mateusz/grumpy/

Trademarks

Unix is a registered trademark of UNIX System Laboratories, Inc. Windows, WindowsNT, and Win32 are registered trademarks of Microsoft Corp. All other product names mentioned herein are the trademarks of their respective owners.