Last update: 11 Sep 2010

Mótsognir – The mighty gopher server

User manual

Written by Mateusz Viste

gopher://gopher.viste-family.net/1/projects/motsognir/

Table of Contents

Introduction

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

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.

I wrote Mótsognir primarily for fun, but it appears to have become a strong gopher server implementation with some very nice points: easy to install, lightweight, secure... At first, Mótsognir was born as a gopher module incorporated into my Grumpy web server[1]. After several months of development, the gopher support in the Grumpy web server appeared to become a very robust gopher implementation, so I decided to make it a standalone server.

The Mótsognir gopher server is meant to be used for small projects (like home servers), but should scale well on bigger architectures as well. All the configuration is done via a single configuration file, which has very reasonable defaults. That makes Mótsognir easily maintainable, and allows the administrator to have a full knowledge of what features are allowed/enabled on the server, and what's not. Mótsognir supports also server-side CGI applications, is plainly compatible with UTF-8 filesystems, and is entirely written in FreeBASIC[2].

1 http://www.grumpy-server.net/
2 http://www.freebasic.org/

Installation (Linux with inetd)

Installing Mótsognir on a Linux host is very easy. On the first place, you will have to copy the motsognir executable to /sbin (or /usr/sbin, if you prefer), and the motsognir.cfg configuration file to /etc. As mentioned previously, Mótsognir needs an inetd-like superserver to work. If you are using the basic inetd (included by default in many Linux distributions), you will have to add the following line to /etc/inetd.conf:

```
gopher      stream      tcp   nowait      root  /sbin/motsognir
```

Important things here are: "gopher", which is the name of service to bind to. You will have to check if the gopher service has its entry in /etc/services. "root" is the name of the user which has to be used to run the motsognir process. Never use root to run motsognir (or any other public daemon) – the example above is just that: an example. Obviously, you will have to check if the user you choosed to run motsognir exists in your system, and let him write to /var/log/motsognir.log, read from /etc/motsognir.cfg and execute /sbin/motsognir. The easiest way to do that is simply to run "chown user file" on each file, where "user" is the username you choosed to run motsognir, and "file" is the file you want to set permissions on. When the inetd configuration is done, you'll have to restart the superserver (or the whole machine). In Debian, restarting inetd can be done by running the command "kill -HUP `cat /var/run/inetd.pid`".

Installation (Linux with 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 the motsognir executable to /sbin or /usr/sbin, and motsognir.cfg to /etc. The behavior of xinetd is very similar to inetd. However, the configuration file has a different syntax. Some distributions have a unique configuration file for xinetd, other use separate files for each service (these files are usually placed in /etc/xinet.d/). In any case, you will need to add such (or similar) lines somewhere in the configuration of xinetd to let it know about Mótsognir:

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

Important things here are: "service gopher", which is the name of the service (port) to bind to. You will have to check if the gopher service has its entry in /etc/services. "root" is the name of the user which has to be used to run the grumpy process. Take care to never use root to run Grumpy (or any other public daemon). The above example is just a dumb example, for you to get the general idea. However, you may want to run Mótsognir under the root account for troubleshooting purposes. Obviously, you will have to check if the user you choosed exists in your system, and let him write to /var/log/motsognir.log, read from /etc/motsognir.cfg and execute /sbin/motsognir. The easiest way to do that is simply to run "chown user file" on each file, where "user" is the username you want the gopher service to run on, and "file" is the file you want to set permissions on. When the xinetd configuration is done, you'll have to restart the superserver (or the whole machine). In most distributions, restarting xinetd can be done by running "/etc/init.d/xinetd restart". For more details on available features, see the xinetd manual.

Installation (Windows)

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

As explained in the "Authentication" part of this manual, the Mótsognir gopher server looks for ".motsognir.auth" files in directories which have 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 some kinds of tricks to create the file. One way would be to run the "echo. > .motsognir.auth" command from within the command-line command.

After being started, the Mótsognir server will listen on the port TCP/70. If you would like to check whether it's active or not, you can simply type the "gopher://127.0.0.1/" URL in your gopher browser, to see whether you get something in return or not.

Keep in mind, that Windows is not case-sensitive when it comes to handling file names. Therefore, a request for "gopher://server.net/0/file.txt" will return the same resource than "gopher://server.net/0/FILE.TXT" (there are big chances that you don't mind anyway).

Also, you should check your firewall's configuration, to avoid any filtering-related trouble (basically, you need to open your TCP/70 port for incoming connections).

Configuration file

The Mótsognir's configuration file is located at /etc/motsognir.cfg (Linux) or in Mótsognir's directory (Windows), and should be readable by the user which will run the motsognir service. The configuration file is a plain-text file, containing some tokens with values. All lines beginning with the "#" character are ignored (and can be used to put some comments in the configuration file).

If, for some reasons, Mótsognir can't access/read his configuration file, he will load default values. Here is an example of a (rather well commented) configuration file:

```
########################################################################
#                                                                      #
#     CONFIGURATION FILE FOR THE MOTSOGNIR GOPHER SERVER               #
#                                                                      #
########################################################################


## Server's hostname ##
# The hostname the gopher server is reachable at. This setting is
# mandatory. The gopher protocol is heavily relying on self-pointing
# links.
GopherHostname=gopher.mydomain.net

## Gopher TCP port ##
# The TCP port on which the public Gopher server listens on.
# Usually, gopher servers are published on port 70.
GopherPort=70

## Root directory ##
# That's the local path to Gopher resources.
# Default path: /var/gopher/ on Linux, and .\gopher\ on Windows.
GopherRoot=

## Set the loging verbosity ##
#  0 - No loging (not recommended, unless you really don't care about logs)
#  1 - Log basic informations, like Requests and type of answers. (Default)
#  2 - Log extended informations (requests + a summary of the answer + info)
#  3 - Debug verbose (maximum logging, mostly for debugging purpose)
Verbose=1

## Specify custom log files ##
# By default, Motsognir logs any events to /var/log/motsognir.log on
# Linux, and .\motsognir.log on Windows. Below, you can specify a custom
# log files to write to.
GopherLogFile=

## CGI support ##
# The line below enables/disables CGI support. Read the manual
# for details.
# Possible values: 0 (disabled) or 1 (enabled). Disabled by default.
```

```
GopherCgiSupport=0

## Caps.txt support ##
# Caps.txt is a specific file-like selector, which allows a gopher client to
# know more about the server's implementation (for example what the path's
# delimiter is, where is the server located, etc). When enabled, Motsognir
# will answer with caps-compatible data to requests for "/caps.txt".
# Caps support is enabled by default (CapsSupport=1).
CapsSupport=1

## Caps additionnal informations ##
# If Caps support is enabled, you can specify there some additional
# informations about your server. These informations will be served
# to gopher clients along with the CAPS.TXT data.
# Example:
#  CapsServerArchitecture=Linux/i386
#  CapsServerDescription=This is my server
#  CapsServerGeolocationString=Dobrogoszcz, Poland
CapsServerArchitecture=
CapsServerDescription=
CapsServerGeolocationString=

# [End of file here]
```

Directory listings

As any other gopher server, Mótsognir will present to gopher clients listings of available directories with a specific presentation. A specific requirement of the Gopher protocol is that it needs to provide a "type" for every resource. To detect that gopher type, Mótsognir is simply basing on the file's extension. Below is a table containing all relations between gopher filetypes and real file extensions (at least that's the way Mótsognir 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	-

Then, once all filetypes present in a given directory are known, Mótsognir will send a directory listing to the remote gopher client.

Descript.ion files

When asked to list the content of directory, Mótsognir will look after the "descript.ion" files (if found in the listed directory). If found, the "descript.ion" will be parsed to output descriptions for listed items. A "descript.ion" file is just a basic text file, with the following structure (of course, you will have to replace every <tab> with a real tabulation):

```
filename1.txt<tab>This is the description of the first file
filename2.txt<tab>This is the description of the second file
filename3.txt<tab>This is the description of the third file
...
```

Link files (gopherlinks)

The Gopher protocol is able to handle linking to external resources in a very neat way: links are part of the protocol itself, not part of the document (as in HTTP). That's why creating gopher links requires a special trick. For the purpose of gopher links, Mótsognir 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 can 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 (the "Server" token is not used for HTTP links).

Gophermaps

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 Mótsognir is supporting gophermaps. If Mótsognir finds a file called "gophermap" (without any extension) in a directory, then it doesn't check the directory content, and simply outputs to the user the content of the gophermap. A gophermap file must contain gopher entries as described by the RFC 1436. There's an example of a gophermap file (of course <tab> have to be replaced by real tabs!):

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

Note, that you can omit the server's address and server's port parts. If you don't specify a port, Mótsognir provides the one your server is using (usually 70). If you don't specify a host, Mótsognir provides your server's hostname. If you specify a relative selector (not beginning by a / character) instead of an absolute path, Mótsognir sticks on the path of the currently browsed directory (but only if the host part is omitted, or pointing to your own server).

Therefore, a simpler form of the above gophermap could look like that:

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

Authentication

The gopher protocol doesn't provide any standardized methods for authentication. However, Mótsognir provides his own mechanism to perform login/password authentication over gopher.

The authentication can be requested on a per-directory basis. It means that some directories on the server can be protected by a password, and other not. Note, that if a directory is protected, then all its subdirectories becomes protected, too.

To protect a given directory, you need only to put a file called ".motsognir.auth" in it. This file will contain all logins/passwords that are allowed to access the content of this directory. Here is an example of such a ".motsognir.auth" file:

```
# Some comments
john:crazydwarf
angela:kitty
# A comment line again
mark:a complex passphrase
```

Any line which begins with a pound character (#) is ignored. Login/password couples are stored in the form "login:password". Passwords are allowed to contain spaces or special characters. Both logins and passwords are case-sensitive.

Warning: Altough it might seem cool to perform authentication stuff over gopher, keep in mind that the authentication method used by Mótsognir is not secure (probably as much secure as the basic HTTP authentication method used by web servers, but this still is not really secure). The password provided by the user at authentication time will transit in clear text over the network (so it could be catched by anyone who is between the user and the gopher server). Besides, after credentials are checked by the server, Mótsognir provides an authorization token to the user - this token will be visible in the URL as a string of hexadecimal characters.

CGI support

Mótsognir supports CGI application, which allows to run custom scripts and applications interacting with the gopher client. The only supported CGI programs are applications which output text data (binary data won't be interpreted properly by the Mótsognir's CGI parser). Fortunately, CGI scripts outputting binary stuff are rather rare. Note, that CGI support is not available on the Windows port of the Mótsognir server.

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, then the output of your program will go more or less directly to the client. In fact, when it comes to answer to the client, the CGI application will output a gopher response (ie. a plain text file for file type #0, a directory listing for file type #1, etc...). This response will be catched by Mótsognir, and forwarded to the gopher client as being the request's answer.

The gopher server (in our case Mótsognir), may provide some informations to the CGI application, by setting some environment variables (note, that for security reasons – and unlike some other CGI implementations - Mótsognir will never feed CGI scripts with any command-line parameters).

Mótsognir will set several environment variables, which can (and should) be used by the called CGI script. Here is the complete list of these variables:

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
SCRIPT_NAME	Script name (for self-referencing links)
AUTH_USER	The authenticated user (if there was any authentication)
REMOTE_USER	Same as AUTH_USER

Note, that the QUERY_STRING variable will contain data inputed by the user. For type #7 items, it will contain the search string (on type #7 items, the gopher client usually asks the user for a query, using some kind of a pop-up). For any other item's type, the QUERY_STRING variable will contain the part of the URL after the first "?" character (if any). For example, for a request on "gopher://mygopher.server.com/0/myprog.cgi?hellothere", the QUERY_STRING variable will contain the data "hellothere".

Some other environment variables may be also set by your superserver (that is, inetd/xinetd, or any mechanism alike). Here are some additional CGI variables which might be set (check the documentation of your superserver for the list):

SERVER_PORT	The port number to which the request was sent
REMOTE_HOST	The hostname making the request.
REMOTE_ADDR	The IP address of the remote host making the request
REMOTE_IDENT	User name retrieved from the server via the identification mechanism described by the RFC 931

If you want to use CGI scripts on your Mótsognir server, you will have to enable CGI support in the Mótsognir's configuration file, and name your CGI program with the extension *.cgi.

Caps.txt support

Mótsognir supports caps.txt since version 0.99.1. Caps.txt is a file-like selector, which allows a gopher client to know more about the server's gopher implementation (like what is the path delimiter character, how are structured server's paths, what the server's location is, etc).

Caps.txt support is configurable via the Mótsognir's configuration file, using following tokens:

```
## Caps.txt support ##
# Caps.txt is a specific file-like selector, which allows a gopher client to
# know more about the server's implementation (for example what the path's
# delimiter is, where is the server located, etc). When enabled, Motsognir
# will answer with caps-compatible data to requests for "/caps.txt".
# Caps support is enabled by default (CapsSupport=1).
CapsSupport=1

## Caps additionnal informations ##
# If Caps support is enabled, you can specify there some additional
# informations about your server. These informations will be served
# to gopher clients along with the CAPS.TXT data.
# Example:
#  CapsServerArchitecture=Linux/i386
#  CapsServerDescription=This is my server
#  CapsServerGeolocationString=Dobrogoszcz, Poland
CapsServerArchitecture=
CapsServerDescription=
CapsServerGeolocationString=
```

If you would like to have full access to what Mótsognir sends in Caps.txt data, you might consider disabling the caps.txt support in Mótsognir (CapsSupport=0), and simply host your own caps.txt file in the server's root. Here is an example of such custom caps.txt file:

```
CAPS
CapsVersion=1
ExpireCapsAfter=3600
PathDelimiter=/
PathIdentity=.
PathParent=..
PathParentDouble=FALSE
PathKeepPreDelimeter=FALSE
ServerSoftware=Motsognir
ServerSoftwareVersion=0.99.1
ServerArchitecture=Linux/i386
ServerDescription=This is my gopher server
ServerGeolocationString=Dobrogoszcz, Poland
```

Frequently asked questions (FAQ)

Q: Does Mótsognir support special (nationalized) character sets in file names?

A: Yes, it does. Mótsognir implements support for UTF-8 encoded URLs, therefore it is able to handle any existing language. Note, that it requires the local server's filesystem to be using UTF-8 too, otherwise only the basic ASCII set will be handled.

Q: Can I use Mótsognir for commercial purpose, or adapt it to my own needs?

A: Yes, you definitely can. Mótsognir is released under the GNU/GPLv3 license, therefore everyone is free to use it, modify it, and even sell it. However, you can't claim that you are the author of this software, and you must provide the source code of any modification you do on it (and you can't EVER drop the GPL licensing). Note, that if you add any feature or fix to Mótsognir, I would be happy to add your code to the official release.

Q: Is there any way to run server-side applications on Mótsognir?

A: Mótsognir supports executable CGI scripts, which allows to run custom server-side scripts (not available on the Windows platform).

Q: What's the maximum file size that Mótsognir can serve?

A: Mótsognir itself can serve files which are up to 8 exbibytes (EiB) big. However, there are chances that your filesystem will limit you much sooner (for example EXT3 supports files up to 2 TiB of size, while EXT4 supports files up to 16TiB).

Q: What does "Mótsognir" stand for?

A: In Norse mythology, Mótsognir is the father of the Dwarves. Mótsognir is the creation of Odin and his brothers, Vili and Vé, who fashioned him out of Ymir's blood and bones in the form of a maggot. He got a roughly humanoid appearance and a human-like intelligence, which the rest of the Dwarves later inherited.

Q: Does Mótsognir support the HTTP protocol?

A: No. Mótsognir is a gopher server. Gopher is a protocol very different from HTTP. However, if you send by mistake a HTTP request to Mótsognir (for example using a URL like http://yourserver:70/), he will politely answer to you with a HTTP error message, explaining what the mistake was.

Legal mumbo-jumbo

gopher://gopher.viste-family.net/1/projects/motsognir/

Trademarks