Watcher

A real-time intrusion detection system and '*superordinate firewall management*' solution

for Linux server systems

to keep network bandits away …

Watcher Master manual

Revision 1.2

Philosofies

Kiss … Keep
it
small and
simple.

Easy is beautiful.

Do one thing and do it well.

"Everything should be made as simple as possible – but not simpler"

(Albert Einstein)

"Every work of art has one indispensible mark … the center of it is simple, however much the fulfillment may be complicated."

(Gilbert K. Chesterton)

Contents

1 Preface

- **Why wait that a burglar or an attacker of a computer system can disturb your running services?**
- **Moreover, why have the services to be in charge of defending themselves from burglars and attackers?**

Having bandits rejected right at the entry before they can mess with the services on the computer system is the far better solution.

The good thing with computer networks is, that they are organized by transfering ‚packets' with determined structure and the specific services have ‚port' numbers by which they are identified.

A usual ‚packet' transfered by IP (‚Internet Protocol') for IP V4 looks about like this …

IP header			
Source IP addr **177.15.238.16**	Target IP addr. **98.73.115.4**	Protocol bits tcp, udp, ...	... other transport information …
TCP/UDP protocol header			
Source port xxxxx	Service (dest.) port **25**	Sequence number nnnnnnn	... other transport information …
Data block			

The information in the header of the packet is what the ‚routing' deals with to take the ‚data' to the 'service' for processing.

Firewalls in between use the header information to *allow* or *deny* the transport of a packet – that's all what a firewall does in principal.

So any incoming data packet identifies the ‚requestor' by its ‚*source IP address*' and the ‚*service port*' classifies what the requestor is up to do on the target system.

By tracking the requestor's behavior it is possible to identify the requestor as a normal service user or as a ‚burglar' or ‚attacker' of the target system.

1.1 What's the audience for Watcher?

Watcher is addressed to anyone who has to operate a server on the open internet these days as an '**exposed host**'.

In such a situation a server system is exposed to the access by anyone (i.e. any other system accessing the internet) which is essential for some services; e.g. Mail transport agents (MTAs like PostFix, Exim, Qmail,…), WEB services and last but not least 'console login' services to maintain and operate the own servers across the internet.

Companies and people that have rented a so-called 'root server' from a provider to hang out their domains to the open internet for presentation are facing the harsh winds that are going 'round on the open internet. SPAM, SCAM, break-in attempts and other kinds of attacks are instantly seen when the computer system is exposed to the open Internet.

1.2 What does Watcher do?

Watcher tracks the system log on the system and determines the behavior of requestors. In this sense Watcher is an 'intrusion detection' system. But Watcher does not only detect burglars & attackers but takes measures against future attempts to keep the services free from defending themselves instead of doing their usual tasks.

Watcher modules record every attempt of burglary & attack in a database and if a maximum of attempts is reached the module puts a DROP into the firewall and records this in the database as well.

There are several **watcher modules** in order to track the behavior of requestors on several different services.

- Console Logins (SSHD)
- Mail transport request to the MTA & Mail access services (POP, IMAP, …)

Ill-behavior of requestors is honored by a DROP in the firewall after some identified requests, that show hostility or pointlessness of a requestor. This way a requestor is blocked right at the ‚front door' and cannot pester a service with attacks; e.g. attempts of pushing SPAM through a mail server.

1.3 How does Watcher do this?

At system start-up (or after reboot) the system starts the Watcher service (Master) which in turn **flushes the firewall within seconds** from several resources.

Watcher runs a master process and starts several modules for specific services.

- The WatchLG module tracks aggressive login attempts.
- The WatchMX module tracks mail transport requests for signs of abuse
- The WatchMB (as a companion of WatchMX) tracks illegal access to the mail users mailboxes or illegal mail transport request usually conducted by SPAMmers.

Each module has its own database where all the illegal access attempts are stored during scanning of the requestor's behavior. So it is easily possible to restore the firewall with DROP information for IP addresses that were identified in the past as ‚burglar', ‚SPAMmer', a.s.o

On a running system it looks like this ...

```
[root@hphws2 Watcher]# ps -ef| grep -i watch
root     2873     1  0 07:32 ?        00:00:00 /bin/bash modules/WatchLG/WatchLG
root     2874     1  0 07:32 ?        00:00:00 /bin/bash modules/WatchMX/WatchMX
root     2875     1  0 07:32 ?        00:00:00 /bin/bash modules/WatchMB/WatchMB
```

1.4 How is Watcher constructed?

Philosophy: KISS … Keep It Small and Simple.

Watcher avoids the use of any fancy programming languages or 3rd-party runtime environments

Watcher is **based on onboard tools** that any modern UNIX-like operating system provides anyway or that are publicly available as reliable OpenSource components.

Essentially Watcher uses GNU bash, GNU awk and relies on other GNU implementations of the UNIX tool-set (like GNU grep, etc.) that come with every Linux system.

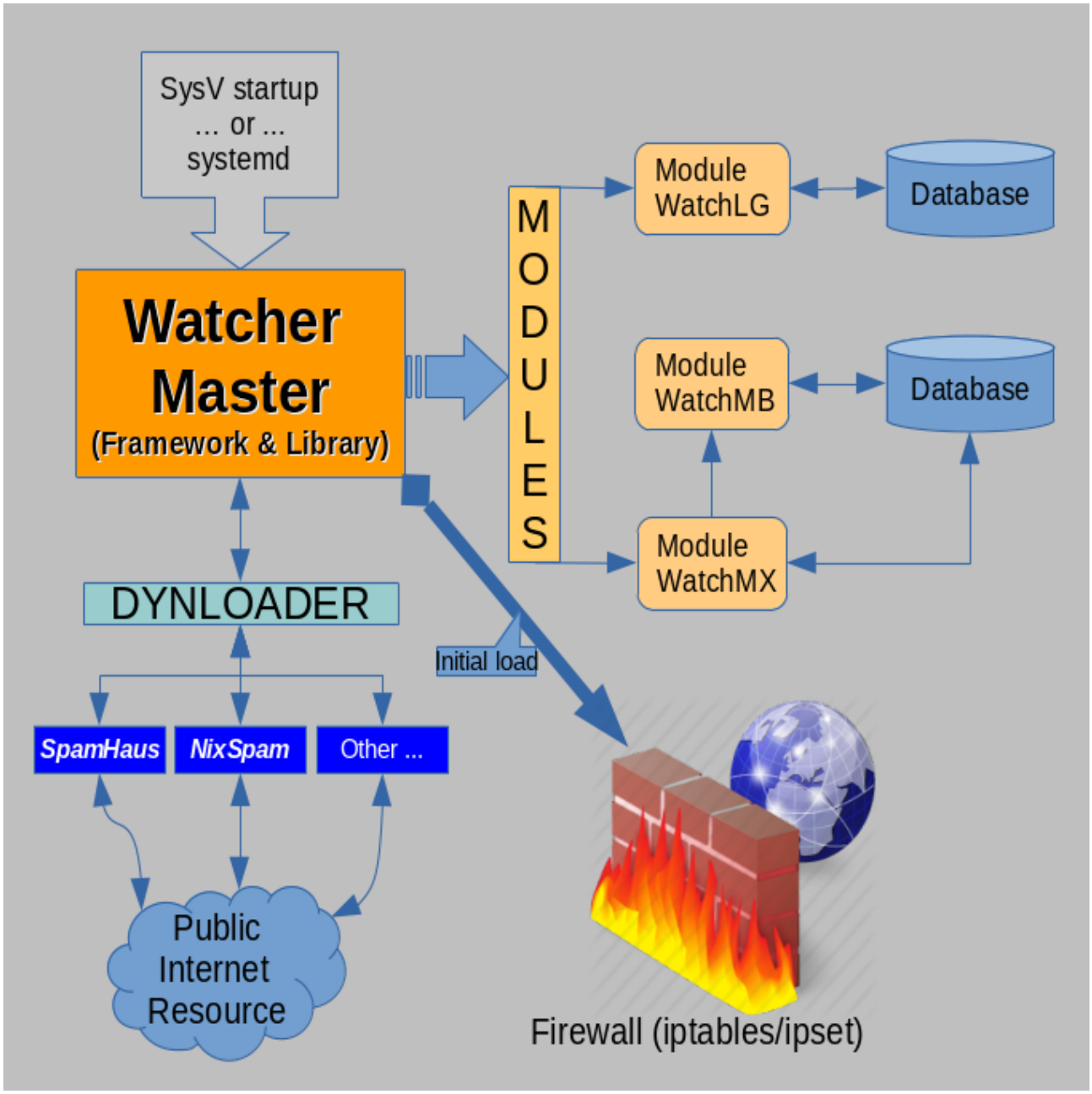
The Watcher development started in 2013 on an old SuSE 9.7 Linux system as a mere hack just to prevent break-in attempts to a rental 'root server' in a data center.

Later on Watcher **development continued on a CentOS system** (CentOS-6 in 2016 and since 2018 on CentOS-7) whereas CentOS is known for its "old-fashioned" orientation. "*Latest-is-greatest*" is not the way of CentOS but ***reliability, stability & maintainability***. So CentOS is running with all tools at least one release level behind the actual releases .

By growing Watcher on such a basis it is pretty much guaranteed, that Watcher works just fine on newer systems like RHEL/CentOS-8. This also keeps ways open to easily adapt Watcher to other Linux distributions (Debian, Ubuntu, …) or even other operating systems (AIX, HPUX, Solaris, ...) for which GNU renditions of the UNIX tool-set are available.

2 Overview

2.1 Overall architecture



2.2 Watcher master

The ‘Watcher master’ provides the **framework and library** for other components like ‘dynamic loaders’ (dynloader) and ‘modules’ that address specific services on a particular server system. It

acts as a ‘one-shot’ service for the initial load of the firewall at system startup or after reboot and then dies.

The watcher master gets started by the operating system if it is enabled as a service.

In turn the master starts the configured modules as these are enabled in the *watcher.conf* configuration file if these dynloaders or modules are installed at all.

The master process does **not** use a database.

In the first place Watcher master provides global blacklisting tables that are maintained manually and provides a **local blacklist** for IP addresses and complete networks that are to be DROPed consequently because of known ill-behavior.

In addition the master provides mechanisms to request DROP lists from global resources (like **SpamHaus** or **NixSPAM**, etc.). Such programs are known as ‘dynamic loaders’ (DynLoader)

In principal the Watcher master can run without any modules using only its own static blacklisting and the worked out information from common/external resources.

But with installed modules the tracking of burglars, attempts of system abuses & attacks get tremendously more dynamic.

2.3 Watcher modules

The Watcher modules are the real work horses in the Watcher system.

Modules are autonomous processes that get started and stopped by the Watcher master service. Once started they run independently in the Watcher framework.

Modules provide:

- **Real-time intrusion detection**
- Storage of their information in **exclusive databases** for fast retrival and update (instead of linear file searches that turn out getting slower-and-slower as data grows)
- **Statistic reports** to get measurement data of the efficiency of the measures
- Automatic cleanup of the databases through **expiration.**
- Autonomous feed of the firewall with DROPs.
- **Autonomous logging** and **extended tracing for the intrusion detection**.

For a detailed explanation of the Watcher modules see the “**Watcher modules manual”.**

3 Installation manual

The watcher service takes some basic system resources so that it can work.

In its preparation routine ./Prep it checks whether the needed system components are present

3.1 Install path & unpacking

Before the Watcher master package is unpacked a choice must be made where to locate and unpack the package. The Watcher system can run from any path you like.

Good choices are:

- */opt/…*
- */usr/local/…*
- */root/bin/…* (or just /root/…); provided your root filesystem is not short o space

A Watcher system that ran for some time consumes about 100-120MB. The most filesystem consumption comes from module databases and the *.trace files of the modules that live directly in the module path. So calculating ~200..250MB of filespace is a good choice to be prepared of future extensions: i.e. additional modules or if you plan to roll your own dynloader.

The master package unpacks to a relative path ‘Watcher/...” and all Watcher packages (master & modules) are stored in simple ‘*.tar’ files. To unpack the package file just type:

 # tar xvf <package>.tar

To continue installation change to the installation directory with:

 # cd Watcher

3.2 Preparation

In the installation directory (master path) you will find a program named ***Prep***

The PREP script will handle a big part of the installation tasks for you.

- It checks for needed programs which the watcher depends on
- Places a start-up script in /etc/init.d/…

Just go to the installation directory and type in:

./Prep [ENTER]

‘*Prep*’ itself is started with a so-called ‘hash-bang’ (#!) to the GNU bash normally found in */bin/bash.*

If there is no GNU bash installed on your system 'Prep' fails with an error message from the shell that you are using:

(your shell) ./Prep: /bin/bash: bad interpreter: No such file or directory

If this should happen install a GNU bash V3.x or newer on the system which is **absolutely essential**. Not even '*Prep*' can start without.

If 'Prep' detects other missing things it will tell you and the missing things must be installed on the system before *Prep* will continue with further preparation for the Watcher framework.

If the check for needed tools passed successfully then *Prep* continues with the preparation of the startup in */etc/init.d* and prepares the Watcher startup in traditional SysV manner. When it detects a systemd-style startup concept on the system a 'watcher.service' file will be also installed in */usr/lib/systemd/system.*

3.2.1 Needed programs

Absolutely essential:

- GNU bash V3 or newer
- GNU awk V4 or newer
- GNU grep V2.2 or newer

Additionally needed:

- 'iptables' & iptables-services
  Note: The firewalld will not do for an **automated firewall management**!

- 'ipset' (since Watcher 1,2)
- 'ipcalc' for validation of IP addresses
- 'realpath' since all Watcher components check WHERE they are running and determine their own file position and name for identification in log-files and such.
- 'sqlite3' if any of the **modules** is in use; the watcher-master itself does not need a database

3.2.2 Toolpath

Some of the tools that come with the Watcher master or the modules sym-link themselves to the $TOOLPATH variable as */usr/local/sbin* to be widely available.

```
[root@vmd28527 Watcher]# ls -l /usr/local/sbin/
total 8
lrwxrwxrwx 1 root root  28 Nov 17 10:28 freeme -> /root/bin/Watcher/bin/freeme
lrwxrwxrwx 1 root root  44 Nov 17 10:28 LGinjector -> /root/bin/Watcher/modules/WatchLG/LGinjector
lrwxrwxrwx 1 root root  44 Nov 17 10:28 MXinjector -> /root/bin/Watcher/modules/WatchMX/MXinjector
```

To easily access these tools the PATH variable can be extended in the superuser's .profile in .bash_profile or the ./bashrc file:

export PATH=/usr/local/sbin:$PATH

4 Operation Manual

4.1 Watcher master operation

4.1.1 Starting and stopping the watcher service

```
[root@hphws2 etc]# service watcher start
Starting service watcher:
Installed modules: WatchLG WatchMB WatchMX

Loading firewall ...
Took 1.091000 seconds
for 20741 total firewall drops
Loadrate: 19010 DROPs per second
Starting module WatchLG ...
Starting module WatchMX ...
[OK]
```

Possible options are:

stop	Stops all modules
start	Requests all Loadfile information from dynloaders, modules and static files (blacklist, whitelist, ...) and finally calls FillFW to set up the firewall with complete initial DROP information from all resources before the modules get started. Finally reads the local manually maintained 'blacklist' and 'whitelist' files
restart	Does a 'stop' followed by 'start' in one rush
reload	For internal use by the dynloaders if a dynloader has changed its Loadfile-xxxx in the load pool (../../Loadfiles) Never use 'restart' from within a dynloader script to 'restart' the Watcher service if you have provided a custom dynloader – this will lead to recursion and can crash your system within seconds. Use '**service watcher reload**' on exit of your custom dynloader. (See Appendix A on how to roll your own custom dyloader.)
full	Restarts the iptables service as well prior to restart the watcher service; i.e. it starts

	with a ‘blank’ firewall setup.

If your particular system has a systemd-style startup system the ‘systemctl’ command might be used to control the Watcher startup:

 # systemctl <option> watcher

The difference is, that all output from *systemctl* goes to the /*var*/log/messages log file.

4.1.2 Maintaining static lists (blacklist, whitelist)

In order to provide individual lists of IP addresses the Watcher master directory ($MASTER_PATH) holds files blacklisting and whitelisting; namely: ‘blacklist’ and ‘whitelist’

The format of is files is as follows:

#comment
<tabs/blanks>
<tabs/blanks> #comment
<IP addr or CIDR>
<IP addr or CIDR> #comment

(… and so on ...)

- Everthing beyond a ‘#’ is ignored
- Comments can be indented by tabs & blanks (e.g. for section heading)
- Empty lines are ignored
- Data lines with single IP addresses or CIDRs are taken from the first column.

  A comment may follow separated by tab and/or blank characters

See the ‘blacklist’ & ‘whitelist’ *-sample files in the $MASTER_PATH.

Take care that all IP addresses or CIDRs are valid. Otherwise the initial firewall load might fail as iptables will grok on these.

As a minimum there should be at least a whitelist file ‘whitelist’ present with your essential addresses provided.

```
[root@vmd28527 Watcher]# cat whitelist

<your host IP here>     # This host
127.0.0.1               # This localhost
10.0.0.0/8              # This VPN
```

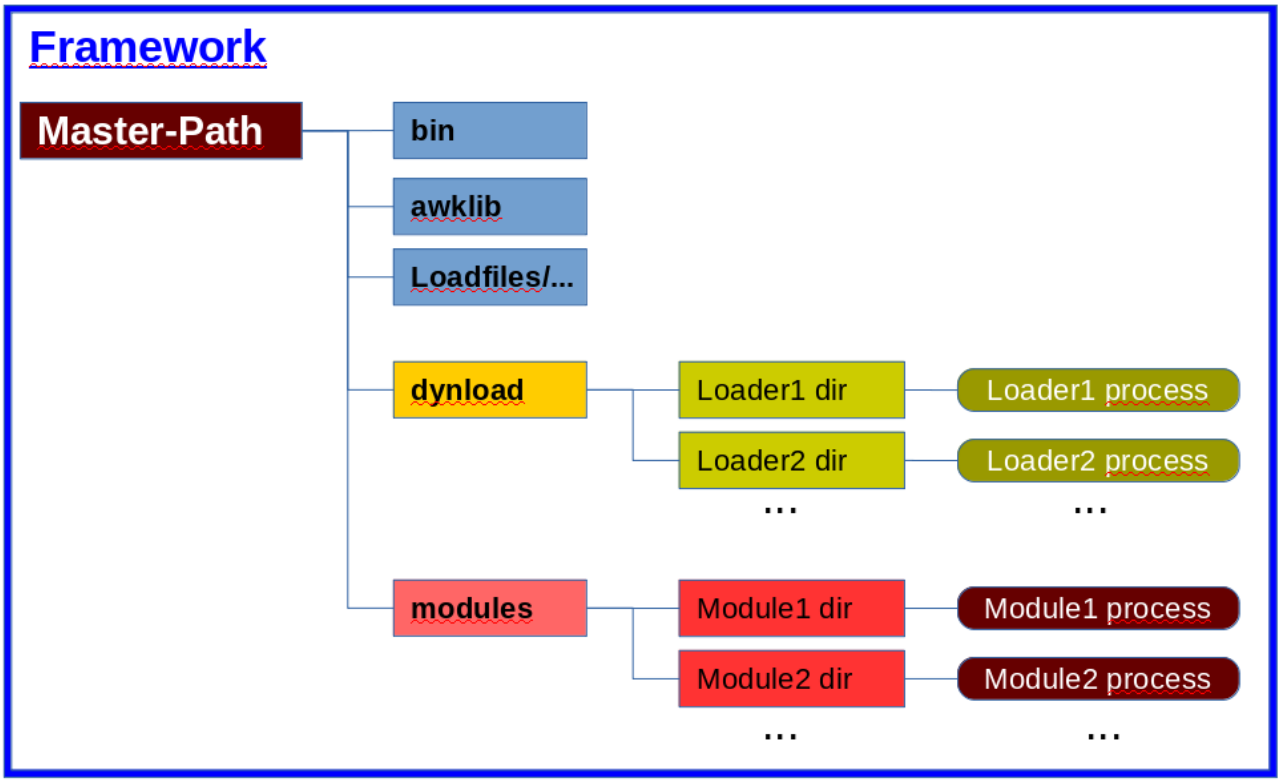
This will prevent getting locked-out by the firewall on your own addresses.

If you have made changes to the ‘blacklist’ and/or ‘whitelist’ file(s) restart the watcher service.

 # service watcher restart
… or … **# /etc/init.d/watcher restart**
… or ... **# systemctl restart watcher**

Appendix A

5 Watcher directory structure



Seen from a loader or module process the Master-Path is always 2 levels up; i.e.: ../../

6 Dynloaders

'Dynamic loaders' (dynload) are utility programs that read **IP address lists from external resources**: i.e. from outside the Watcher framework. This can be files on local filesystems, remote systems or from somewhere on the Internet of which the dynloader knows how to retrieve them.

A dynloader is kind of a 'translator' that conducts a transition of a proprietary external list format into a compliant format for the Watcher flushloader "*FillFW*", since FillFW expects a loadfile in a **clean iptables format**:

```
[root@vmd28527 Watcher]# grep DROP Loadfiles/Flushload | head
-I INPUT 1 -s 185.126.116.0/22      -j DROP -m comment --comment spamhaus,drop
-I INPUT 1 -s 41.77.240.0/21        -j DROP -m comment --comment spamhaus,drop
-I INPUT 1 -s 77.36.62.0/24         -j DROP -m comment --comment spamhaus,drop
-I INPUT 1 -s 131.108.16.0/22       -j DROP -m comment --comment spamhaus,drop
-I INPUT 1 -s 138.59.4.0/22         -j DROP -m comment --comment spamhaus,drop
-I INPUT 1 -s 199.84.56.0/22        -j DROP -m comment --comment spamhaus,drop
-I INPUT 1 -s 157.186.0.0/16        -j DROP -m comment --comment spamhaus,drop
-I INPUT 1 -s 185.248.132.0/22      -j DROP -m comment --comment spamhaus,drop
```

Every dynloader has a 'Load' routine which is usually just a sym-link to the main program:

```
[root@vmd28527 Watcher]# cd dynload/spamhaus/
[root@vmd28527 spamhaus]# ls -l Load spamhaus
lrwxrwxrwx 1 root root    8 Nov  5 11:24 Load -> spamhaus
-rwxr-xr-x 1 root root 1995 Nov 18 21:55 spamhaus
```

This load routine is automatically called **once** by the Watcher service program during service start when the Watcher service includes 'loader conf' from the Watcher MASTER_PATH.

```
[root@vmd28527 Watcher]# cat loader.conf
# ------------------------------------------------------------
# loader.conf (used by FILLFW - the Firewall filler)
# Startup lines for the modules and DynLoader (dynamic loaders)
#
# To disable load from a module or DynLoader just clamp it off with
# a comment character in the first column
# ------------------------------------------------------------
modules/WatchLG/Load
modules/WatchMX/Load
dynload/spamhaus/Load
#dynload/nixspam/Load
```

6.1 Repeated dynloader calls

Dynloaders are called only once during startup of the Watcher service.

The Watcher service cannot know how external resources schedule their provisioning and changes to it. So subsequent calls to a dynloader must be configured through CRONTAB entries that match the schedule of informations by the external providers.

```
(crontab 'root')
# --- DynLoader       : Once per hour
10 * * * *      cd /root/bin/Watcher/dynload/spamhaus      && ./Load
30 * * * *      cd /root/bin/Watcher/dynload/nixspam       && ./Load
```

The 'Load' routine of the dynloader then manages to retrieve refreshed information.

This will create a new 'Loadfile-<*dynloader name*>' in the framework's loadpool:

../../Loadfiles/**Loadfile-<dynloader name>** (as seen from the dynloader)

For the firewall the retrieval of new/changed information from an external provider is not seen. Hence the 'Load' routine must trigger the Watcher service to collect the information in the load

pool. This is the job for ‘FillFW’ that creates a (new) flushload file ‘Flushload’ from all current Loadfile-<XXXX> files in the load pool directory.

Therefore the dynloader’s ‘Load’ routine recalls the Watcher service on exit with the **‘reload’ option** to get the changes of a specific provider worked-in into the static part (filter/INPUT chain) of the firewall.

Other dynloaders (or modules) are not affected by this. But you should keep in mind that the ‘refresh calls’ from external providers should run with a little time lag to avoid concurrency; i.e.: don’t run them at the same minute ...

(crontab ‘root’)

```
# --- DynLoader       : Once per hour
10 * * * *      cd /root/bin/Watcher/dynload/spamhaus      && ./Load
30 * * * *      cd /root/bin/Watcher/dynload/nixspam       && ./Load
```

---

7 Rolling your own custom dynloader

What if you have some IP address list from other providers with ‘badfinger’ listings, that you would like to integrate with Watcher?

Here a schematic concept is shown on how to roll your dynloader.

To get a clue of the construction you may verify with the included dynloaders ‘spamhaus’ & ‘nixspam’.

It is assumed that the provider delivers the ‘attacker list’ as a simple *.txt file. Infact it can be any format. But that is abstracted/encapsulated by the ‘extraction function’ in your dynloader code.

First of all your code should contain the usual ‘initialization block’ in the program heading that is common for all processes in the Watcher concept for ‘positioning’ and ‘naming’.

```
#!/bin/sh
if [ "$1" == "debug" ]; then set -x; fi
#
# Plug MyDynLoader lists dynamically into firewall ...
#
#-------------------------
REALPATH=`realpath $0`        # Where is my absolute file position?
WHERE=`dirname $REALPATH`     # What is my path?
ME=`basename $REALPATH`  # What’s my program name?
cd $WHERE                # Finally hook to the path where we are
called
#-------------------------
# Get common definitions and library functions from the master.
source ../../common.conf

trap cleanup 0 1 2 9 15
cleanup() {
        logger "$ME[$$]: Finished."
}
```

You may add to the cleanup() function as you like; e.g. removing temporary files that your code has generated. But do not place your ‘exit code’ here.

The program code for a dynloader is fairly straight forward on the whole.

MyDynLoader (dynload/MyDynLoader/MyDynLoader)

Initialization code from above ...

function retrieve	… a function that retrieves the 'proprietary' list from the provider by its **individual file name (retrieval file)**.
function extract	… a function that extracts a mere list of IP addresses from the **retrieval file** and output into **$DROPLIST**
function XX2FW	… a function that reads **$DROPLIST** and creates the **Loadfile-$ME** in iptables load format mapped by the **$LOADFILE** variable

```
# ------------ Main program -----------
LOADFILE=../../Loadfiles/Loadfile-$ME
DROPLIST=Droplist-$ME
retrieve
extract
XX2FW
```

```
# Exit code … (enable after successful testing)
# service watcher reload
```

Include the grey parts of the coding scheme at first as these comprise the Init, Main & Exit sections.

Then build-up your functions step-by-step and check the results that you get. The critical point is the creation of the dynloader's $LOADFILE as this goes to the load pool and will be loaded with the next (re)start or reload

Check your code thoroughly.

If the Loadfile-$ME from your custom dynloader produces a clean file to be read by iptables, then symlink your custom dynloader script to the common name '*Load*' in the dynloader's path:

 # ln -s MyDynLoader Load

With this preparation you can include your custom dynloader to the '*loader.conf*' file in the MASTER_PATH.

---

modules/WatchLG/Load
modules/WatchMX/Load
dynload/spamhaus/Load
#dynload/nixspam/Load
dynload/MyDynLoader/Load

Last but not least create a CRONTAB entry in the 1crontab file for the super-user ‘root’ to get your custom dynloader started on a regular basis:

(crontab ‘root’)
--- DynLoader : Once per hour

```
10 * * * *    cd /root/bin/Watcher/dynload/spamhaus        && ./Load
30 * * * *    cd /root/bin/Watcher/dynload/nixspam         && ./Load
50 * * * *    cd /root/bin/Watcher/dynload/MyDynLoader     && ./Load
```

Whether your custom dynloader really runs regularly you may check /*var*/log/messages file, since at least the cleanup() function writes to the log file when it lately has ‘Finished’.

Finally:

Don’t forget to enable the service reload in the exit code of your custom dynloader!