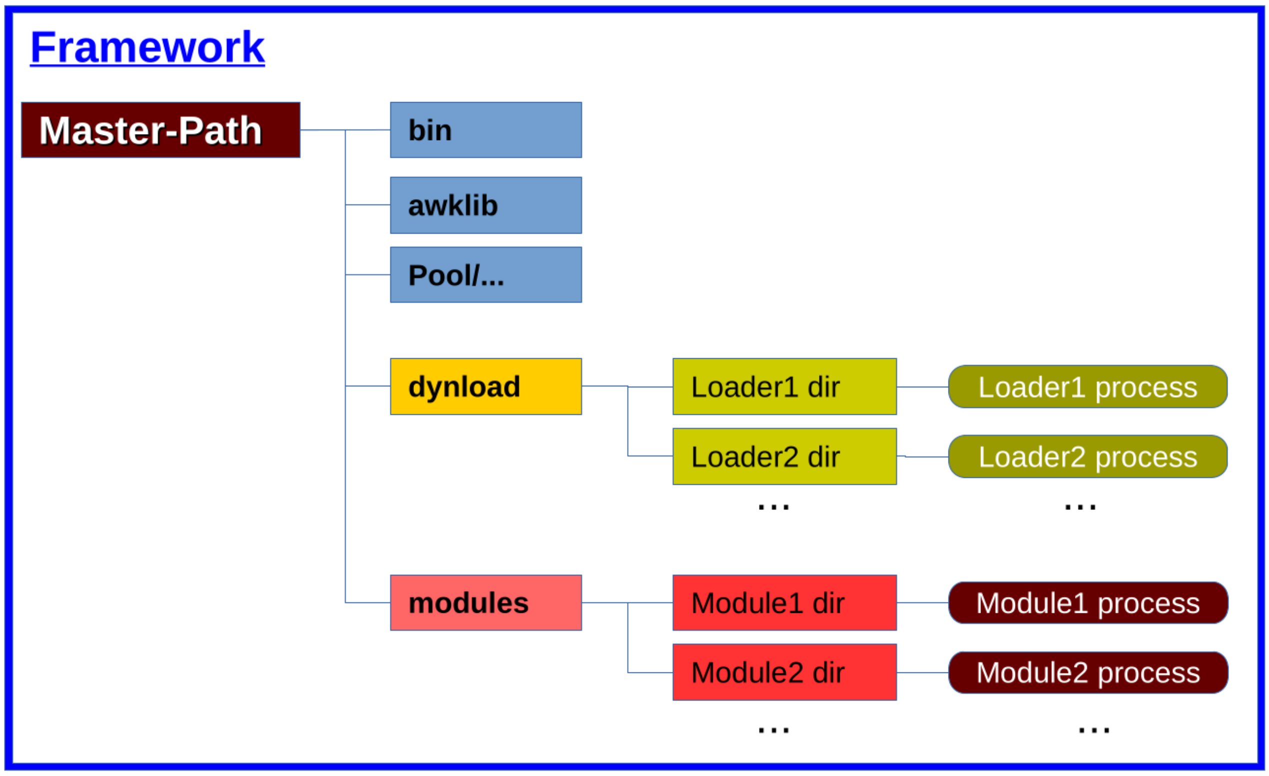
Appendix A

A Appendix A – Additional information

A.1 Watcher directory structure



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

A.2 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 IPSET that the symlinked ‘Load’ program outputs for the exclusive ‘ipset.Loadfile-XXXX’ that is then in turn filled-in into exclusively assigned IPSET. This process is **fully dynamic since Watcher release 1.3** and ‘reload’ of the entire firewall is no longer needed which speeds up the loading of the firewall tremendously. The entire load time for all dynloaders and modules is with Watcher release 1.3 usually below a second.

```
[root@vmd28527 Loadfiles]# head -10 ipset.Loadfile-LG
-exist create WatchLG-DB hash:ip comment
add WatchLG-DB 175.6.35.82 comment "DB,Login,NXDOMAIN"
add WatchLG-DB 106.54.253.9 comment "DB,Login,NXDOMAIN"
add WatchLG-DB 221.181.185.198 comment "DB,Login,NXDOMAIN"
add WatchLG-DB 176.112.79.111 comment "DB,Login,NXDOMAIN"
add WatchLG-DB 51.15.81.22 comment "DB,Login,FAKEHOST"
add WatchLG-DB 106.12.219.184 comment "DB,Login,NXDOMAIN"
add WatchLG-DB 221.181.185.159 comment "DB,Login,NXDOMAIN"
add WatchLG-DB 176.122.164.94 comment "DB,Login,TRUEHOST"
add WatchLG-DB 95.165.131.164 comment "DB,Login,TRUEHOST"
.
. (and so on)
.
```

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
```

So one must not know the individual name of the loader routine. It can be just called by the name ‘Load’ with the full path.

This makes the entries in the CRONTAB for the updates easy to read.

(‘root’ CRONTAB)

```
#
#============ Watcher ==========================================
#
*/5 * * * *      /usr/local/sbin/freeme >/dev/null 2>1&
#--- DynLoader          : Once per hour
10 * * * *       cd /root/bin/Watcher/dynload/spamhaus      && ./Load
30 * * * *       cd /root/bin/Watcher/dynload/nixspam       && ./Load
… and so on …
```

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.

For the consecutive updates of provided data a CRONTAB entry is needed that matches the timely update policy of the external provider.

```
[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
```

A.2.1 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.

(excerpts of crontab for '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 'ipset.Loadfile-<*dynloader name*>' in the framework's loadpool:

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

For the firewall the retrieval of new/changed information from an external provider is **fully transparent** (i.e. not seen).

With the consequent introduction of IPSETs in Watcher release 1.3 this process is **fully dynamic** and there is no longer any need to 'reload' the entire 'chain' in the firewall if changes happened.

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; i.e. keep them some minutes apart.

(crontab 'root')

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

A.3 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/bash
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 variable 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 outputs into **$DROPLIST**
function XX2FW	… a function that reads **$DROPLIST** and creates the **ipset.Loadfile-$ME** in ipset’s load format mapped by the **$LOADFILE** variable

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

```
# Exit code … (enable after successful testing)
```

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.

Check your code thoroughly before you integrate it in your loader.conf and/or have it fired up by a CRONTAB entry.

If the ipset.Loadfile-$ME from your custom dynloader produces a clean file to be read by ipset, 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.

B Appendix B – Other systems

Watcher was initially developed on CentOS which is a RedHat-style Linux distribution. So all the examples show how things are organized and configured for such a RedHat-style system.

For other Linux distributions like Debian (and its offsprings like Ubuntu) things may be different due to the differences in system organization.

For instance configuration of system services is found on RedHat-style systems below ‘/etc/sysconfig/…’ but a Debian-style system has organized this to be to be below *‘/etc/default/…’* or somewhere else. Also package names may essentially differ among different distributions. For example, what a RedHat-style package delivers in package ‘iptables-services’ can be found in the software repository for a Debian-style system by the package name ‘netfilter-persistent’.

This chapter relates to these differences and explains the changes for other systems.

B.1 Debian

Debian is a top-level distribution style and so is the guideline for the offsprings like Ubuntu and the offsprings of offsprings like Linux Mint (unsupported by Watcher)

In Debian the ‘iptables-services’ package to fill the initial firewall setup is ‘netfilter-persistent’. So in order to have Watcher work properly you have to install the ‘netfilter-persistent’ package:

```
# apt install netfilter-persistent
```

The firewall setup of a Debian-style system is kept in */etc/iptables/rules.v4*. This file must be edited to reflect your situation.

As a starting point the following can be taken

```
# sample configuration for iptables service
# you can edit this manually or use system-config-firewall
# please do not ask us to add additional ports/services to this
default configuration
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 25 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 443 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 5900 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 5901 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 5902 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

B.2 Ubuntu

Ubuntu is an offspring of Debian. So Ubuntu shares much with the Debian organization. Here we explain the specifics for Ubuntu.

At the time of this writing there were no things, that go beyond the Debian specifics. So if the Debian specifics are full-filled you are done.

B.3 SuSE

SuSE is prefering the ‘firewalld’ as the favourite firewall management system. This is definitely a very bad idea.

‘firewalld’ is not a linux firewall system at all. ‘firewalld’ is a PYTHON wrapper around the native ‘nft’ commands that manage a modern ‘nftables’ kernel firewall. Even worse is, that SuSE did never supply an ‘iptables-services’ package, that loads an xtables firewall configuration by use of the ‘iptables’ command.

SuSE systems luckily has adopted RPM as the software cataloging system and uses the system configuration in /etc/sysconfig/… like all RedHat-style systems.

So to get a ‘iptables-services’ package into a SuSE is fairly easy with some easy tweaks.

1. Get an ‘iptables-services’ package from any RHEL repository; e.g. that of CentOS 8.

   https://centos.pkgs.org/8/centos-baseos-x86_64/iptables-services-1.8.4-17.el8.x86_64.rpm.html

2. Install the *.rpm with the ‘--*nodeps*’ option into the SuSE system:

   ```
   # rpm -ivh --nodeps iptables-services-1.8.4-17.el8.x86_64.rpm
   ```

The first start of ‘service iptables start’ will produce some errors on missing things, that must be fixed.

1. 1. The directory ‘/var/lock/subsys’ must be established:

   ```
   # mkdir /var/lock/subsys
   ```

2. The file /etc/init.d/functions is missing and must at be at least ‘touched’ to get rid of the error

   ```
   # touch /etc/init.d/functions
   ```

3. For the ‘iptables-service’ only two trivial functions are referenced in this file and the following text must be inserted exactly as shown into the file /etc/init.d/functions:

```
success() {
        echo “Success”
}

failure() {
        echo “Failure”
}
```

Finally you just have to edit the ‘iptables’ load file in /etc/*sysconfig/*… to fit your needs.

```
# sample configuration for iptables service
# you can edit this manually or use system-config-firewall
# please do not ask us to add additional ports/services to this
default configuration
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 25 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 443 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 5900 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 5901 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 5902 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

(The blue lines are added here to the sample configuration to have the ports for the most important services opened plus some basic VNC ports for remote management of the machine.)

Fill-in other configuration for your situation that you can transform from the ‘firewalld’ setup into native iptables commands.

You don’t have to take care of any IPSETs, that Watcher uses, since Watcher (and in particular the Watcher modules) will create the IPSETs dynamically ‘on-the-fly’ and links them with the firewall in a **fully dynamic manner**.