Watcher

A real-time intrusion detection system and ‘*superordinate firewall management*’ solution for Linux server systems.

... keep network bandits away ...

Watcher - Master manual

Revision 1.4

Philosophies

Kiss … Keep it small & 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 indispensable mark … the center of it is simple, however much the fulfillment may be complicated.” (Gilbert K. Chesterton)

Contents

Scope of this document

This document explains the Watcher framework which provides the basis for everything that is running in the Watcher framework.

For the service related modules there is a separate document ‘Watcher modules’, that explains the modules in detail; i.e. their construction, maintenance & operation, etc. In this document only the way how modules are integrated into the framework is outlined in a terse manner.

Watcher is chiefly designed for so-called ‘root servers’ that are running million-wise as ‘exposed hosts’ as rented computer systems in data centers to provide ‘internet presence’ for freelancers, small companies and private persons.

Dynamic loaders are are treated as part of the framework and are discussed in the section ‘Dynloader’.

» PREFACE

1 Preface

- **Why wait that a burglar or an attacker of a computer system can disturb your running services or overload it with pointless or even knowingly malicious requests?**

- **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, since it preserves the performance of your services.

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

A usual ‚packet‘ transferred 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 principle.

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 Why Watcher?

There are already other ‘Intrusion Detection Systems’ (IDS) and ‘Intrusion Prevention Systems’ (IPS) in the field like ‘*Fail2Ban*’, ‘*Snort*’, ‘*Suricata*’ and quite a couple of other solutions.

Why don’t we just use them? The trouble is: ‘integration’ and fitness for the purpose.

We were not willing to integrate and maintain several different IDS/IPS systems for the main services that an ‘*exposed host*’ is delivering to the open Internet.

These services are:

- Login services
- Mail Transport & Mailbox access services
- WEB services (http/https)

Those services run on some specific ‘service ports’ that must be open in the firewall for the purpose of the machine. Other service ports are **shut in the firewall anyway** and therefore there is **no need to deal with them at all** at the service level.

Watcher brings it to the point … since ‘service related’ **Watcher modules automatically track the log files in real-time** of only those services that are under attack by bandits on the Internet. And Watcher integrates this trilogy of scanners into a single framework – so there is no messing with a couple of different systems and tools.

- **Fail2ban** … only takes care of the ‘login’ services and so it is not sufficient on ‘root servers’ that also run mail transport, mailbox & WEB services.
- **Suricata** … is more a full-fledged ‘network traffic analyzer’ than an IDS/IPS and is overkill for the purpose of protecting services that run over open service ports in the firewall.
- **Snort** … is meanwhile a commercial product and owned by *CISCO-Talos*. There is a ‘community’ edition available but this is restricted by a ‘for personal use only’ policy. Due to its age Snort is pretty much outdated.

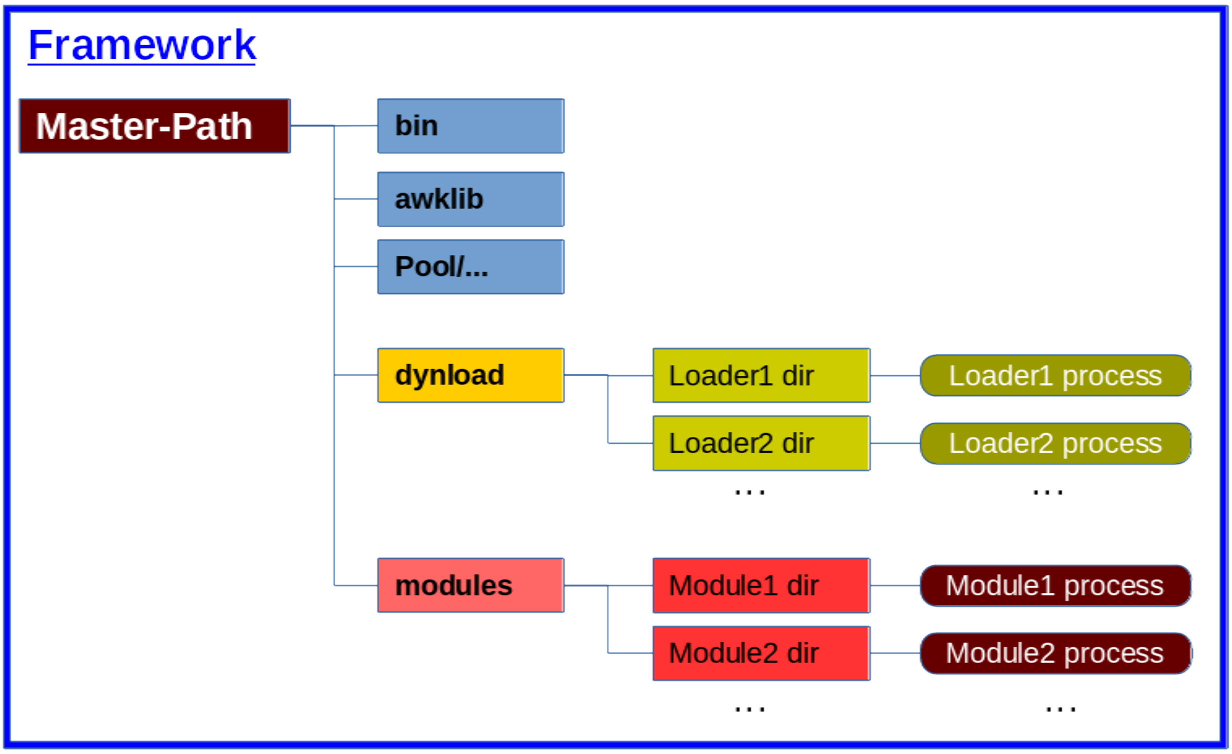
Indications of attacks can be **directly found in the service’s log files** and there is **no need for ‘scientific low-level network analysis’**.

For example, if a *Postfix MTA* complains in its log file with

> “…: **connect from unknown[**<IP addr aa.bb.cc.dd>] …”

the situation is entirely clear. It says by the pattern '\: **connect from unknown\[**', that someone coming from a *NXdomain* (Non-eXistend domain; i.e. not known by any DNS) is trying do deliver a mail message across this MTA. But MTAs do not talk to NXdomains, since they are not 'open relays' to transport mail messages for any anonymous sender that comes along.

So a Watcher module related to a service just syncronously reads **in real-time** the logs of services straight from the system-logger.



Each 'Module Process' is straight related to a so-called 'facility' of the system-logger to which the service delivers its log. Then the system-logger (*rsyslog* or *syslog-ng)* passes this log messages through a FIFO directly to the 'Module Process' of a Watcher module for determination and registration of an attacker in an exclusive database that results probably in a DROP in the firewall for an attacker, if a configurable limit of 'affairs' is reached.

So **Watcher integrates all 'scanners' (modules) into a single framework** and there is no need to operate and maintain a bunch of several different products.

1.2 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**’. These are usually so-called ‘***root servers***’ and machines that are running in the DMZ of a data center.

In such a situation a server system is exposed to 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 business or organization 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.

1.3 What does Watcher need?

1.3.1 Firewall

Since Watcher is a ‘*subordinate firewall manager*’ Watcher needs a flawlessly working firewall on the system. It is assumed that a firewall is already installed and running. The Prep routine will check this in the first stage of preparation and denies preparation of the Watcher framework if there is no suitable firewall installed and running.

A modern Linux kernel provides two different ‘firewall systems’:

- xtables …. maintained by iptables, ebtables, arptables & ipset
- nftables … maintained by ‘NetFilter Tables’ (NFT)

In Watcher Rev. 1.x series the ‘traditional’ **xtables kernel firewall** is supported, that is maintained by the *iptables* command, the *iptables-service* and the *ipset* tool.

(Watcher Rev. 2 will be fully based on ‘nftables’ that integrates IPV4 & IPV6, iptables, arptables, ebtables and ipset in a single framework that is maintained by the ‘*nft*’ tool.)

Assure that iptables, iptables-services & ipset tools are installed and working flawlessly.

The **‘firewalld’ is not supported,** since ‘firewalld’ is just a PYTHON wrapper around the native ‘nft’ commands that maintain a *nftables* kernel firewall.

1.3.2 System logger

The Watcher modules **read the system logger output in real time** through exclusively assigned FIFOs (‘named pipes’). This takes manual preparation of your system logger before any of the installed and activated modules can work. See the installation manual for modules on how the system logger (*rsyslog* or *syslog-ng*) must be prepared in order to work with Watcher modules properly.

1.4 What does Watcher do?

The Watcher master package provides the

- framework
- libraries (APIs for awk & bash)
- system determination and automatic preparation tasks
- start-up services

As that the Watcher master is the basis for operation and must be installed first of all.

Watcher modules automatically and dynamically track specific system logs 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. This leads to a much better performance of the services.

Watcher modules record every attempt of burglary & attack in a database and if a configurable maximum of attempts (‘affairs’ in Watcher’s nomenclature) is reached the **modules 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-box access services (POP, IMAP, …)
- WEB server log scanner (integrated with release 1.3)
- Geo-Blocking dynloader & Geo-Tracking (pseudo-)module (new with release 1.4)

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 or gaining console access to the system.

1.5 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. Since revision 1.3 it takes not even a second.

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

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

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', 'attacker' a.s.o

The GeoTrack (pseudo) module even blocks complete subnets (CIDRs), if a country is on the list in the configured 'ZONES=...' variable.

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 ../WatchMB/WatchMB
```

1.6 How is Watcher constructed?

Philosophy: KISS … Keep It Small & Simple.

Watcher avoids the use of any fancy programming languages or 3rd-party runtime environments and no compilation of source code is needed.

Watcher is **entirely 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.; 'coreutils' et al) that comes with every basic installation of a 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 often 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-8/CentOS-8 or even newer. 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. Revision 1.3 introduced extended to support for:

- SuSE Linux Enterprise Server (SLES) & openSuSE Leap
- Debian
- Ubuntu

1.7 What’s new in this revision?

The previous release Watcher 1.3 completed the trilogy of modules by implementing the module “WB” for reacting on WEB server attacks.

Watcher 1.4 provides geo-blocking and geo-tracking on top.

Note in particular that geo-blocking and geo-tracking do not block single IP addresses but **block entire sub-nets** based on the CIDR the attacker lives in. The lists for the geo-blocking dynloader already come in with CIDR notation. The GeoTrack pseudo-module determines in a sophisticated process the CIDR from official resources for an incoming IP address; i.e. the first IP address of a CIDR range triggers the blocking of the entire CIDR range.

1.7.1 Geo-Blocking & Geo-Tracking

Watcher revision 1.4 drops-in geo-blocking & geo-tracking.

The experience shows, that over 60% of all attacks are coming from Russia & China. Blocking such countries by DROPs in the firewall can increase the effectivity of attack suppression to a level of 95%. This was the motivation to implement geo-blocking and geo-tracking.

1.7.1.1 Geo-Blocking

Geo-Blocking is implemented as a ‘dynloader’ by the name ‘**geo**’.

The dynloader ‘geo’ loads lists from ‘ipdeny.com’, turn these lists into loadable file for IPSET and loads the firewall information into the ipset ‘geo’.

As typical for dyloaders they are called just once during Watcher startup in the load-phase. Consecutive runs must be implemented by an entry in the root’s crontab file to update the firewall information with the schedule of the updated lists by the list provider.

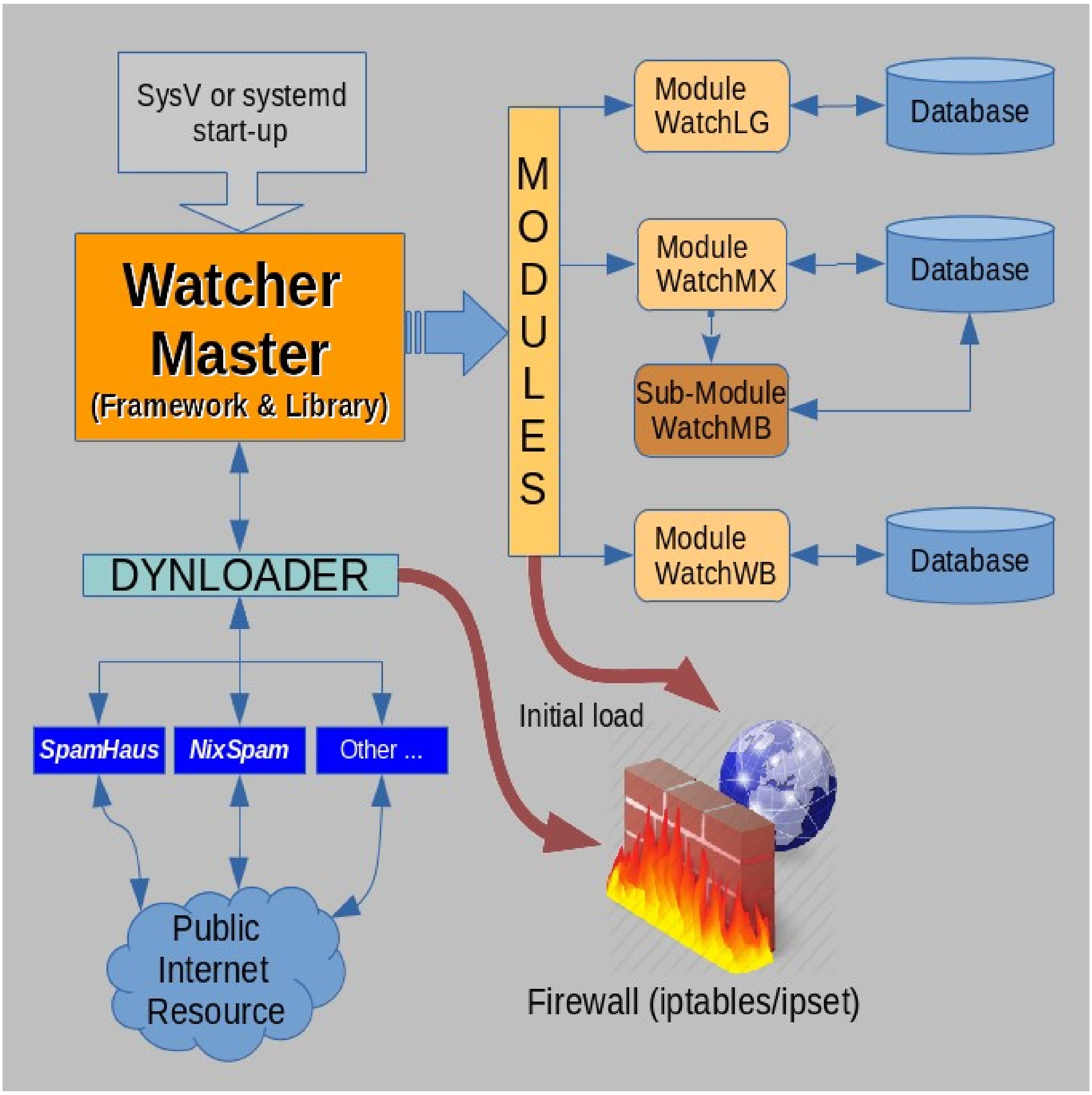
1.7.1.2 Geo-Tracking

Geo-Tracking is implemented as a (pseudo-)module ‘GeoTrack’. It’s called pseudo-module since it follows the module concept but does not serve a particular system service (Login, MTA/Mailbox, WEB server) but is fed by the regular modules every time a culprit is registered to the database of a regular module.

» Part 1 – Overview and concepts

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 ‘blacklist’ & ‘whitelist’ files that are maintained manually and provides a **local blacklist** for IP addresses and complete networks (CIDR format) 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 called ‘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 gets **fully 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 and can even restart themselves if they crash for some reason that occurred.

Modules provide:

- **Real-time intrusion detection & prevention**
- Storage of their information in **exclusive databases** (sqlite3) for fast retrieval 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
- Self-cleaning through automatic housekeeping of the databases through **database 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 separate "**Watcher modules manual".**

2.4 Dynloaders

Dynamic loaders (dynloaders) are specific programs that can fetch information from **external resources**. These 'external resources' can be anything: local files, lists from the Internet, etc.

See the section for dynloaders in 42

» PART 2 - WATCHER INSTALLATION

3 Installing Watcher

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 and automatically creates a ‘*system.conf*’ file that adapts and explains the system to the Watcher master, the dynloaders, the modules and the utility programs.

3.1 Pre-installation considerations

3.1.1 Turning off SELINUX

On a ‘root-server’ or datacenter server in the DMZ there are usually no users but the ‘superuser’ allowed and/or registered. Having SELINUX enabled does more harm than it does any good in such a situation where no non-privileged users are connecting the system. For instance, SELINUX rules forbid the system logger to write log files outside of /var/log/… and on ORACLE Linux it was found, that creating ‘named pipes’ (FIFOs) where they are needed was forbidden. Turning SELINUX ‘off’ solves all this.

Edit the file /etc/selinux/conf and change:

SELINUX=disabled

Then reboot the system before you continue.

3.1.2 Switching to syslog-ng

Practically all linux distributions come usually with ‘rsyslog’ installed as the standard system logger. But experience showed, that ‘rsyslog’ has its flaws which are not acceptable:

- hanging FIFOs
- redundant criss-cross logging to several log files for log streams from the WEB module’s

After switching to ‘syslog-ng’ all these quirks were gone and configuring works far better as ‘syslog-ng’ is smarter and excellently documented.

So it is recommended to replace ‘rsyslog’ with ‘syslog-ng’ (OCE; Open Community Edition) which is fairly easy to accomplish. See the section ‘Configure the system logger’ for details.

3.2 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 of disk 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; e.g. 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
```

3.3 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 the platform on which it is running: RedHat, SuSE, Debian/Ubuntu ...
- It checks for needed programs which the watcher modules, dynloaders and utility programs depend on
- Places a start-up script in /etc/init.d/… and/or systemd configuration and enables the 'auto start' of the watcher service (master)

Just go to the installation directory (MASTER_PATH) and type in:

```
./Prep [ENTER]
```

'*Prep*' itself is started with a so-called 'she-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/watcher.service.*

3.3.1 Needed programs

Absolutely essential:

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

Additionally needed:

- ‘iptables’ & iptables-services
- ‘ipset’ (essential since Watcher 1.3)
- ‘ipcalc’ for validation of IP addresses (not available on Debian-like systems; e.g. Ubuntu)
- ‘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
- … also …
- ‘at’ for the auto-restart after a module runs into a crash
- ‘bc’ for some floating-point calculations; e.g. for tools like *Watcher-Report*, *Procrate*, etc.

3.3.2 Toolpath

Some of the tools that come with the Watcher master or the modules symlink themselves to the path provided by the TOOLS_LINK variable defined in watcher.conf */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, .bash_profile or the ./bashrc file:

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

This way all the Watcher tools are generally available from $TOOLPATH.

Note, that you must log-out and log-in again after such a change to your PATH variable.

» Part 3 – Watcher operation

4 Operation tasks

Watcher on the whole is designed for ‘minimum maintenance’. So the amount of maintenance tasks for the framework is fairly small.

4.1 Watcher master operation

4.1.1 Starting and stopping the watcher service

```
[root@hphws2 etc]# service watcher start
Starting service watcher:
Watcher V1.3 ... Running on centos 7
Language en_US.UTF-8, LC_ALL: en_US.UTF-8
Watcher-ramdisk on /root/bin/Watcher-1.3/Pool … size=20480k)

Present DynLoaders: nixspam spamhaus
Present Modules   : WatchLG WatchMB WatchMX

Loading firewall ...
      Took   0.6890 seconds
      for    16038 total firewall DROPs
      Loadrate: 23277 DROPs per second
Starting module WatchLG ...
Starting module WatchMX ...
Starting module WatchWB ...
Installed modules: WatchLG WatchMX WatchWB
[OK]
```

Possible options are:

stop	Stops all modules Note: This will **NOT unload the DROPs** from the firewall. The system remains protected until the iptables service is stopped/restarted as well.
start	This triggers the ‘Load’ programs 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. Note that the iptables service is not affected by this. Only the watcher modules and dynloaders are triggered to load their exclusive IPSETs from their resources.
reload	Restarts the iptables service prior to restart the watcher service; i.e. it starts based on a ‘plain’ setup of your firewall settings.

(full)	Deprecated. Replaced by ‘reload’ With Watcher revision 1.3 no longer needed and removed from the options.

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 and the command operates otherwise quiet unlike the ‘service’ command that reports pretty verbose and detailed.

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 for blacklisting and whitelisting; namely: ‘*blacklist*’ and ‘*whitelist*’

The format of this files is as follows:

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

(… and so on ...)
```

- Everthing after a ‘#’ is ignored
- Comments can be indented by tabs & blanks (e.g. for section headings in complex files)
- 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.

As a minimum there should be at least a whitelist file named ‘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 ...      # systemctl restart watcher
… or …        # /etc/init.d/watcher restart
```

(the latter gets you a full output of the start-up script)

5 Common framework features

5.1 Logging & Tracing

Watcher has its own log format for writing log & trace files.

The timestamp in log & trace files has ISO format with milliseconds prepended on a log line followed by the origin and PID of the logging instance.

```
2023-07-31T11:28:31.414 <origin>[<pid>]: <original log line>
```

This format is defined by the log(), clog() and trace() functions in *common.bashlib* which is sourced by every program, dynloader or module in the Watcher framework.

(excerpt from common.bashlib)

```
# log simple
log() {
        printf "%-18s %-s[%d]: %s\n" "`date +%Y-%m-%dT%T.%3N`" "$ME" "$$" "$*" >> $LOG
}

#
# Columized log
# 4 columns: 10,10,15,rest
#
clog() {
        printf "%-18s %-s[%d]: %-10s %-10s %-15s %s\n" "`date +%Y-%m-%dT%T.%3N`" "$ME" "$$" "$1"
"$2" "$3" "$4" >> $LOG
}


…
trace() {
        if [ ! -z "$TRACE" ]
        then echo "`date +%Y-%m-%dT%T.%3N` $ME[$$]: $*" >> $ME.trace
        fi
}
```

The clog() function is only used to write the log files in /*var*/log/Watch*XX*.log to get a nicer tabular output for the essential event information from modules.

```
2023-07-31T02:17:21.597 WatchLG[13831]: [inject]   NXDOMAIN   112.186.218.246 'Initial 4/5, Penalty: 4'
2023-07-31T03:24:13.959 WatchLG[13831]: [inject]   FAKEHOST   81.17.22.114    'Initial 4/5, Penalty: 4'
2023-07-31T03:24:14.080 WatchLG[13831]: [inject]   FAKEHOST   81.17.22.114    'Dropped @ 5'
2023-07-31T04:08:54.802 WatchLG[13831]: [inject]   NXDOMAIN   185.224.128.142 'Dropped @ 5'
2023-07-31T05:34:11.431 WatchLG[13831]: [inject]   NXDOMAIN   210.179.113.202 'Dropped @ 5'
2023-07-31T10:33:07.407 WatchLG[13831]: [inject]   NXDOMAIN   103.138.71.242  'Initial 4/5, Penalty: 4'
2023-07-31T11:07:49.673 WatchLG[13831]: [inject]   NXDOMAIN   159.89.203.133  'Initial 4/5, Penalty: 4'
2023-07-31T11:28:28.785 WatchLG[13831]: [inject]   FAKEHOST   45.95.147.207   'Initial 4/5, Penalty: 4'
2023-07-31T11:28:28.865 WatchLG[13831]: [inject]   FAKEHOST   45.95.147.207   'Dropped @ 5'
```

The trace() function outputs the entire log line as it was provided by the system-logger.

```
2023-07-31T11:28:28.726 WatchLG[13831]: [Loop: 1829] 'Jul 31 11:28:28 vmd28527 sshd[29371]: Address
45.95.147.207 maps to host0.ennsdomains.com, but this does not map back to the address - POSSIBLE
BREAK-IN ATTEMPT!'
2023-07-31T11:28:28.729 WatchLG[13831]: [inject] Triggered by rule ['BREAK-IN'] 'POSSIBLE BREAK-IN
ATTEMPT'
2023-07-31T11:28:28.765 WatchLG[13831]: [inject] FAKEHOST 45.95.147.207 Port:
2023-07-31T11:28:28.787 WatchLG[13831]: [inject] FAKEHOST 45.95.147.207 'Initial 4/5, Penalty: 4'
2023-07-31T11:28:28.790 WatchLG[13831]: [inject] Finished for rule ['BREAK-IN'], 73/60 ms
```

The related time zone of the Watcher timestamps is that, which is defined for your individual operating system; i.e. *localtime* in your time zone.

5.2 Debugging aid

All programs, dynloaders and modules have debugging information in the preamble of the code straight after the shebang:

```
#!/bin/bash
if [ "$1" == 'debug'  ]; then set -x;   shift; fi
if [ "$1" == 'debug2' ]; then set -xvT; shift; fi
(rest of the program code)
```

So in case of any troubles you may start a program (tool or dynloader) with either '*debug*' or '*debug2*' as the first argument after the program name:

```
<program> debug  (arguments, if any)
<program> debug2 (arguments, if any)
```

With 'debug2' function calls are traced as well.

<u>Example</u>:

```
[root@vmd28527 nixspam]# . Dynload nixspam
/root/bin/Watcher-1.4/dynload/nixspam
[root@vmd28527 nixspam]# ./nixspam debug2
```

This will output the entire run (including all sourced components and function calls) to your console.

5.3 Runtime IPsets

Dynloaders and modules create their IPsets on-the-fly through their ‘*Load routine*’ which the watcher service **calls once during service start**.

In case of a module the ‘*Load routine*’ is a symbolic link to a special program called *LoadXX* that knows hows to retrieve the DROP information from the modules’s database and how to convert the data into a ‘*ipset.Loadfile-$ME-DB*’, that is in turn placed in the pool directory ($POOL).

```
[root@vmd28527 Watcher-1.4]# . Module lg
/root/bin/Watcher-1.4/modules/WatchLG
[root@vmd28527 WatchLG]# ls Load*
Load    LoadLG
[root@vmd28527 WatchLG]# ls -l Load*
lrwxrwxrwx 1 root root    6 May  1 21:03 Load -> LoadLG
-rwxr-xr-x 1 root root 1253 Jun 14 11:59 LoadLG
```

In case of a dynloader the ‘Load routine’ is a symbolic link as well but to the main program of the dynloader.

```
[root@vmd28527 WatchLG]# . Dynload spamhaus
/root/bin/Watcher-1.4/dynload/spamhaus
[root@vmd28527 spamhaus]# ls -l Load*
lrwxrwxrwx 1 root root 8 May  1 21:03 Load -> spamhaus
```

The static and manually maintained ‘blacklist’ and ‘whitelist’ files are (re)loaded by the Watcher toolbox commands ‘Blacklist’ and ‘Whitelist’, that are called during service startup.

```
[root@vmd28527 Watcher-1.4]# ipset -n list
WatchLG-DB
WatchMX-DB
WatchWB-DB
GeoTrack-DB
spamhaus
nixspam
geo-cn
geo-ru
…
hijackers
blacklist
whitelist
tarpit
custody
validate
```

###

Modules reflect the attackers history stored in their databases in IPsets named ‘Watch*XX*-DB’. These IPsets are loaded by the ‘Load’ routine of the watcher service and **never change at runtime** afterwards. The same applies for the (pseudo-)module GeoTrack.

At runtime only the ‘tarpit’ and ‘custody’ IPsets are used by modules and change.

The ‘validate’ IPset is a special case for systems, that do not have a decent ‘ipcalc’ command which can validate an IP address like the *ipcalc* command in RHEL-style systems.

5.3.1 The ‘tarpit’ set

The ‘tarpit’ IPset is a ***timed ipset*** in the firewall with a DROP target.

It is the most important IPset in the Watcher frame work, since it provides **the means against (D)DOS attacks.** [(distributed) denial of service attacks]

It takes a timeout value on assignment of an IP address, that is counted down from the moment of assignment until it is zero. If zero is reached, the entry will be removed from the IPset and an incoming IP address will be accepted by the firewall again. So the **tarpit IPset is self-cleaning.**

The regular modules (Watch<*XX*>) use the tarpit IPset in their *Initial* & *Count* section of the log scanner for each affair, that a suspected IP address has caused with its attempt to access your server system until the counted number of affairs exceeds the configured MAX_AFFAIRS for the module.

The timeout value is logarithmic by ‘power of 2’ and calculated in the module code as:

`… timeout $((2**AFFAIRS * TIME_SLICE)) …`	i.e. $2^{affairs}$ * TIME_SLICE

This means, that with each affair the timeout value doubles; i.e. TIME_SLICE is multiplied by 2, 4, 8, 16, 32 with increasing affairs of an IP address.

```
[root@vmd28527 WatchWB]# ipset -t list tarpit
Name: tarpit
Type: hash:ip
Revision: 4
Header: family inet hashsize 1024 maxelem 65536 timeout 60 comment
Size in memory: 11808
References: 2
Number of entries: 7
```

[Note: Without ‘-t’ (terse mode) a complete list of ‘Members’ in the IPset is shown as well and the comment shows you the remaining timeout in seconds, the module type and the affair count of the bandit.]

Also note, that the type of the ‘tarpit’ IP set is ‘hash:ip’. This means, that the ‘tarpit’ IPset can only take single IP addresses. This is just in case to block a supposedly attacking IP address in a timely manner.

5.3.2 The ‘custody’ set

The ‘custody’ IPset is the **final destination for malicious IP addresses**, that have exceeded their affair counter in a module’s database beyond the MAX_AFFAIRS value that is configured in the modules configuration file.

Unlike the ‘tarpit’ IPset where the entries timeout after a while the ‘custody’ IPset keeps the bandits firmly in custody as the name implies; i.e. attacking IP addresses are kept ‘in jail’ after the module has registered the bandit in its database with a ‘DROP’ state and the bandit dares to reoccur.

The trace output of the module’s injector (inject function of the module) reports this as:

```
… : [inject] DROPed culprit <IPADDR> re-occured ... taken into custody
```

The ‘custody’ IP set is **not normally self-cleaning**, since bandits gather with time (weeks, months, ...)

You may check the filling state of the IPset with the *ipset* command once in a while:

```
[root@vmd28527 WatchWB]# ipset -t list custody
Name: custody
Type: hash:net
Revision: 6
Header: family inet hashsize 1024 maxelem 65536 comment
Size in memory: 1385
References: 2
Number of entries: 8
```

[Note: Without ‘-t’ (terse mode) a complete list of ‘Members’ in the IPset is shown as well and the comment shows you the module type and the affair count of the bandit.]

If you see ‘Number of entries:’ being several thousands after some time (weeks, months,…) you can simply ‘reload’ the watcher service with “`# service watcher reload`”. This will clear the ‘custody’ IPset and all dropped bandits can be found in the Watch*XX*-DB ipset after the IPsets were reloaded by the ‘Load’ phase of the watcher reload.

Note also, that the type of the ‘custody’ IPset is ‘hash:net’. This means, that an entry in the custody IPset can be an entire subnet in CIDR format and not just a single IP address like in the ‘tarpit’ IPset.

5.3.3 The ‘validate’ (pseudo-)set

The ‘validate’ IPset is a pseudo IPset, that is not linked with any firewall chain.

It is a timed IPset with a fixed timeout value of just 1 second.

Its only purpose is - as the name implies – to validate an IPV4 address and reply the return code of a ‘blind assignment’.

Since the *ipset* command validates supplied IP addresses on the ‘add’ sub-command the pseudo IPset ‘validate’ is provided.

The return code on a ‘blind assignment’ is not equal ‘0’, if the IP address is invalid:

```
[root@vmd28527 WatchLG]# ipset -q add validate 277.1.2.3; echo $?
1
[root@vmd28527 WatchLG]# ipset -q add validate 177.1.2.3; echo $?
0
```

This is chiefly needed for DEBIAN-ish Linux distributions (Debian, Ubuntu, …), that do not have a *ipcalc* command which can do IP address validation.

Second, IP address validation is essentially needed as plausibility check for manually typed-in IP addresses. This is the case for the ‘emergency injectors’ of modules:

- LGinjector
- MXinjector
- WBinjector

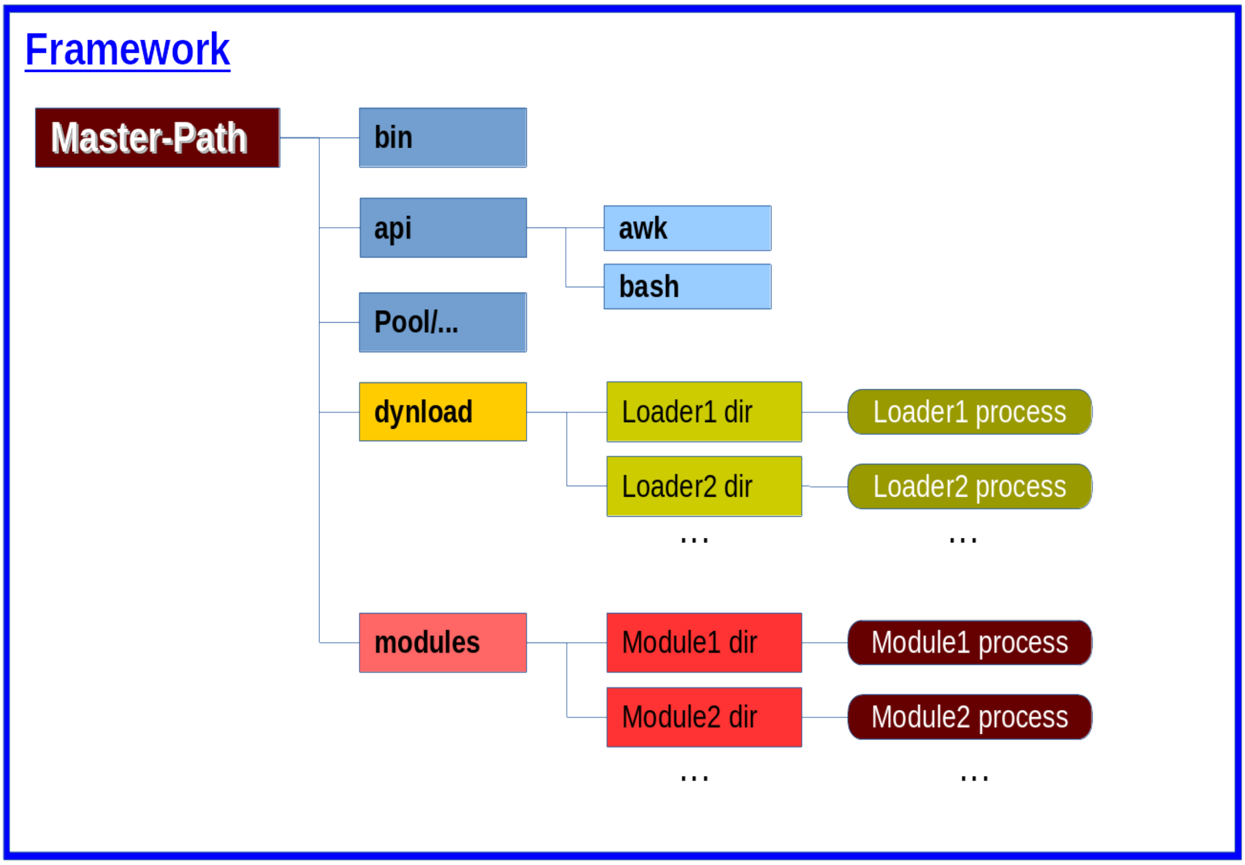
These know how to write the particular database of a module and allow manual input of an IP address on the command line:

```
XXinjector IP_addr tag
```

Then an entry is made to the module’s database and the questionable IP address is straight injected into the ‘custody’ IPset.

6 Appendix A - Additional information

6.1 Watcher directory structure



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

6.2 Watcher Toolbox

In $MASTER_PATH/bin the Watcher framework provides a couple of handy tools. These tools are symlinked to a common path that is usualy in your PATH variable (default /usr/local/sbin defined by TOOLS_LINK in watcher.conf). The configuration variable TOOLS (also in watcher.conf) defines a list of programs to be provided in $TOOLS_LINK. Watcher automatically symlinks all the configured tools to the TOOLS_LINK directory when it (re)starts.

Note that all Watcher tools are starting with an uppercase letter.

6.2.1 Informational commands

6.2.1.1 Trace

Trace [LG|MX|MB|WB|GE]
Trace [lg|mx|mb|wb|ge]

The ‘Trace’ command directly shows the module’s trace file of choice. You don’t have to go to the module’s path or have to type in the module’s trace file path.

6.2.1.2 Watcher-Report

Watcher-Report (no options)
Watcher-Report -d

Watcher-Report is a complete rewrite in Watcher-1,4 to provide information of records with separation of CIDRs, nIPs (normal/single IPs) and xIPs (extended IP range of CIDRs)

Watcher-Report without options prints out a list with the current filling state of IPSETs, databases and contribution by dynloaders. It summarizes the detail values in the ‘Grand totals’ section at the end and points out the manually maintained black- and white-list.
This is useful for a quick ad-hoc overview at a console.

```
================================================================================
      Watcher-Report2 of vmd123606.contaboserver.net, almalinux 9.5
      Date: 2025-03-12 16:14:54
================================================================================

______ Internal IP sets ________________________________________________________
                                   |   Records    CIDRs     nIPs            xIPs
Internal IPset 'custody'           :       176        0      176               0
Internal IPset 'tarpit'            :         1        0        1               0
________________________________________________________________________________
            Totals internal IPsets »       177        0      177               0

______ Dynloader _______________________________________________________________
                                   |   Records    CIDRs     nIPs            xIPs
DROPS contrib. by 'spamhaus'       :      1352     1352        0      15,458,048
DROPS contrib. by 'spamhaus-asn'   :       174        0      174               0
________________________________________________________________________________
       Total DROPS from dynloaders »      1526     1352      174      15,458,048

______ Module databases ________________________________________________________

----- Login ------------------------
                                   |   Records    CIDRs     nIPs            xIPs
                        NX Domains .       217        0      217               0
                        Fake hosts .        96        0       96               0
                        True hosts .        37        0       37               0
```

```
DROPs in module DB 'WatchLG-DB'    :      350        0      350               0

----- Mail -------------------------
                                   |  Records    CIDRs     nIPs            xIPs
                        NX Domains .     1240        0     1240               0
                        Fake hosts .       51        0       51               0
                        True hosts .      115        0      115               0

----- Mailbox ----------------------
                                   |  Records    CIDRs     nIPs            xIPs
                        NX Domains .        5        0        5               0
                        Fake hosts .        1        0        1               0
                        True hosts .       20        0       20               0
DROPs in module DB 'WatchMX-DB'    :     1432        0     1432               0

----- WEB --------------------------
                                   |  Records    CIDRs     nIPs            xIPs
                         Aggressor .       57        0       57               0
                               Bot .     5414        0     5414               0
                         Destroyer .       27        0       27               0
                          Exploits .       11        0       11               0
                         Forbidden .       35        0       35               0
                    Illegal-joomla .      496        0      496               0
                 Illegal-Wordpress .        7        0        7               0
DROPs in module DB 'WatchWB-DB'    :     6047        0     6047               0
_______________________________________________________________________________
               Total of module DBs »     7829        0     7829               0

_____ Geo Tracker _____________________________________________________________
                                   |  Records    CIDRs     nIPs            xIPs
Pseudo module DB 'GeoTrack-DB'     :        1        0        1               0
_______________________________________________________________________________
           Total DROPs in GeoTrack »        1        0        1               0

_____ Geo blockings ___________________________________________________________
(ZONES='cn ru kp by er br in za ir eg et')
                                   |  Records    CIDRs     nIPs            xIPs
DROPs contrib. by ipdeny 'geo-cn'  :     5500     5500        0     343,160,576
DROPs contrib. by ipdeny 'geo-ru'  :     8383     8383        0      44,948,224
DROPs contrib. by ipdeny 'geo-kp'  :        1        1        0           1,024
DROPs contrib. by ipdeny 'geo-by'  :       95       95        0       1,789,440
DROPs contrib. by ipdeny 'geo-er'  :        1        1        0           4,096
DROPs contrib. by ipdeny 'geo-br'  :     4604     4604        0      81,499,904
DROPs contrib. by ipdeny 'geo-in'  :     6726     6726        0      41,673,472
DROPs contrib. by ipdeny 'geo-za'  :     1499     1499        0      27,087,104
DROPs contrib. by ipdeny 'geo-ir'  :     1629     1629        0      10,953,984
DROPs contrib. by ipdeny 'geo-eg'  :      171      171        0      24,147,968
DROPs contrib. by ipdeny 'geo-et'  :       15       15        0         368,640
_______________________________________________________________________________
               Total geo blockings »    28624    28624        0     575,634,432


##### Grand totals ############################################################
###############################################################################

_____ Blacklist (manually maintained in /root/bin/Watcher) ____________________
                                   |  Records    CIDRs     nIPs            xIPs
DROPs in 'blacklist'               :       14       12        2      33,714,688
```

```
_______________________________________________________________________________
                  Blacklist totals »        14       12        2   33,714,688

===== Total firewall =========================================================
                                   |  Records    CIDRs     nIPs          xIPs
_______________________________________________________________________________
           Total drops in firewall »    38171    29988     8183   624,807,168

_____ Whitelist (manually maintained in /root/bin/Watcher) ____________________
                                   |  Records    CIDRs     nIPs          xIPs
ACCEPTS in 'whitelist'             :       18        1       17    16,777,216
_______________________________________________________________________________
                  Whitelist totals »        18        1       17   16,777,216

===== [ Watcher-Report2 finished! ] ===========================================
```

6.2.2 Firewall commands

6.2.2.1 Blacklist

Blacklist
(no parameters)

Will reload the IPSET ‘blacklist’ with changes of the manually maintained ‘blacklist’ file in MASTER_PATH’ file after changes.

Call this command after you have made changes to the ‘blacklist’ file in $MASTER_PATH

See also ‘Whitelist’

6.2.2.2 Whitelist

Whitelist
(no parameters)

Will reload the IPSET ‘whitelist’ with changes of the manually maintained ‘whitelist’ file in MASTER_PATH’ file after changes.

Call this command after you have made changes to the ‘whitelist’ file in MASTER_PATH

See also ‘Blacklist’

6.2.3 Measuring commands

6.2.3.1 Attackrate (experimental)

```
Attackrate [-d|-t] [LG|MX|WB]
-d … data from databases
-t … data from trace files
```

Evaluates the minimum, maximum, average & median of ‘time between attacks’ to a service. This shows the efficiency of a Watcher module.
At the beginning of using Watcher the attack rate might be high (short times between attacks). With time (days, weeks, months) this time will increase from milliseconds to some minutes or probably several hours, since more-and-more attackers get a DROP in the firewall and cannot even see your server anymore.

6.2.3.2 Procrate

```
Procrate [LG|MX|MB|WB|GE]
Procrate [lg|mx|mb|wb|ge]
```

Gives you some performance information about the ‘processing rate’ of a Watcher module.
Note that Procrate collects the data from already ‘log rotated’ trace files by searching for lines ‘Finished for rule ...’ and then picks the processing times from the line end to calculate the minimum, maximum, average & median from the collected data.

6.2.4 Navigation commands

Navigation commands are used in two flavours:

- info
- jump

Calling these by their simple name shows the information about the requested item; i.e. calling ‘Masterpath’ will show the contents of the internal $MASTER_PATH variable

If sourced as ‘. Masterpath’ the command does a real ‘cd’ to the information shown when called in ‘info’ mode. This is because all navigation commands do a ‘cd <info output>’ as the last statement. But a ‘cd’ in a script file does not affect your interactive shell. This is why command must be sourced into your interactive shell for a jump to the target directory.

6.2.4.1 Masterpath

```
  Masterpath
. Masterpath
```

(no parameters)

Without an argument prints the $MASTER_PATH variable.
So “ls `Masterpath`” shows all files in the MASTER_PATH directory directly.

When sourced as ‘. Masterpath’ brings you directly to the MASTER_PATH. So it is shorthand for: “cd `Masterpath`”

6.2.4.2 Module

```
  Module [LG|MB|MX|WB|GE] / [lg|mb|mx|wb|ge]
. Module [LG|MB|MX|WB|GE] / [lg|mb|mx|wb|ge]
```

Parameter is the module token in any case

Without a parameter returns a list of modules that are present (existing module directories) in the module path $MASTER_PATH/modules
When sourced as ‘. Module <parm>’ it does a real jump to the selected module path specified by <parm>

Instead of a module token (LG,MB,MX,WB,GE) you can specify any of the real module directory names that will be returned, if the program is called without a parameter; i.e.
‘Module GeoTrack’ will work as well.

6.2.4.3 Dynload

```
  Dynload [<dynloader name>]
. Dynload [<dynloader name>]
```

Parameter is the dynloader name.

Without any parameter shows a list of available dynloaders in $MASTER_PATH/dynload.

Specify a dynloader name to check for the dynloader path.

For a real jump to the dynloader path source the program by prepending a ‘.<space>’ on the commandline.

A “cd `Dynload <*dynloader name*>`” works as well to jump to the dynloader path.

6.2.4.4 Rules

```
  Rules [LG|MB|MX|WB] / [lg|mb|mx|wb]
```

```
. Rules [LG|MB|MX|WB|GE] / [lg|mb|mx|wb|ge]
```

Parameter is the module token.

Check existence of the ‘../*rules*’ directory of a module or jump to it.

Without a parameter returns a list of modules that are present (existing module directories) in the module path $MASTER_PATH/<module> that have a ‘../*rules*’ sub-directory.

For a real jump to the rules path of the module prepend a ‘.<space>’ on the commandline.

A “cd `Rules <*module name*>`” works as well to jump to the ‘../*rules*’ path of the module

6.2.5 Other commands

6.2.5.1 Check

```
Check {-l} LG|MX|WB (any case)
```

```
Check {-p} LG|MB|MX|WB|GEO (any case)
```

Check is a typical post-installtion tool to checkout the communication with a module. But it can be used in other situations as well; e.g. in in case of communication troubles with a module.

First of all, a module should be able to receive messages over its FiFo (named pipe) in /var/log/.pipes/<module>…

Secondly, the system-logger should be able to supply the module with messages from the service at the modules’s FiFo.

Check provides two methods ‘-p’ and ‘-l’ to check this out. In both cases you should have a second console//terminal opened that runs a ‘Trace <module token>’ for the module that you are checking.

Step #1.) Check -p <module token>
This will check that the FiFo file exists and is is of typ ‘p’ (named pipe). If these conditions are met, a test string is sent with ‘echo …’ into the FiFo, which is reflected in the other console with the running trace for the module. If the test strings comes out in the trace then everything is fine. Otherwise you have to troubleshoot this rudimentary condition.

Step #2.) Check -l <module token>
With a succeeding step #1 you may now check that the system-logger provides the services logs (in realtime) to the module’s FiFo.
‘Check -l …’ unleashes a ‘logger -p <facility>.info “<Test string>”’ and the system-logger pushes this test string into the module’s FiFo. Again, there should be an adequate output in the trace in the other console/terminal.

If there is no output from the ‘Check’ command you have to troubleshoot the system-logger; e.g. by checking the system-logger’s setup (e.g. /etc/syslog-ng/syslog-ng.conf)

6.2.5.2 Freeme

```
Freeme [IP addr]
```

‘Freeme’ is part of the ‘**self-lockout prevention**’ feature.

Without an argument removes the IP address of your home or office router that your ISP has assigned to you from all databases, DB-ipsets & and the ‘custody’ ipset.

If an IP address is given it does the same for the specified IP address; i.e. it removes this IP address from all module databases and IPSETs.

6.2.5.3 MyFritzbox

MyFritzBox [-v]

This command is part of the ‘**self-lockout prevention**’ feature.
It is only applicable for users with a *FritzBox* router at home or in the office.

FritzBox users must have registered their router at AVM in the ‘myfritz.net’ where they get their ‘box ID code’.

1G1111

The ‘box ID code’ then must be placed in ‘/root/.BoxID’ as the plain address on a single line as it was assigned by AVM.
e.g. ***xyzabc4711blablup0815.myfritz.net***

After editing this file set the access mode to ‘600’ and owner ‘root.root’

```
# chmod 600 /root/.BoxID
# chown root.root /root/.BoxID
# ls -l ~/.BoxID
-rw------- 1 root root 29 Jul 12  2021 /root/.BoxID
```

The result of calling the program is the current plain IP address of your router and will be written to /var/log/DYN_IP for reference of the Watcher modules in order to accomplish the ‘**self-lockout prevention**’

With the ‘-v’ option the result is also printed to the display.

For more details see the section ‘Self-lockout prevention’ in the modules documantation.

6.2.5.4 MyRouter

MyRouter [-v]

This command is part of the '**self-lockout prevention**' feature.

This is for users with a *generic* router (other than a registered FritzBox) at home or in the office.

In order to have the command work properly you must first register the router with a 'dynamic DNS provider'.

The *MyRouter* command is used to determine the IP address of generic routers at home or in the office and it works for *FritzBox* routers as well.

Then configure the provider relation in the common.conf file in the Watcher MASTER_PATH like:

```
DYN_PROVIDER=ddnss.de
DYN_ADDRESS=itcomserve.ddnss.de
```

The MyRouter command will then request the IP address of your router, that has the FQDN specified in $DYN_ADDRESS.

The result is stored in /var/log/DYN_IP to be referred to by the Watcher module(s) for 'self-lockout prevention'.

With the '-v' option the result is also printed to the display.

For details see the section 'Self-lockout prevention' in the module documentation.

6.2.5.5 Refresh

Refresh LG|MB|MX|WB (any case)

Reload the rules and superflous_map for a module specified by the module token.

Beginning with Watcher 1.4 it is no longer needed to restart the entire watcher services, if changes were made to rule files and/or the 'superflous_map'.

The *Refresh* command will rebuild a new 'filter' function from the active rules of a module and the loads the new filter function into the running Watcher module on-the-fly by sending a HUP signal to the module process.

```
trap refresh            HUP             # React on a 'HUP' for the filter refresh
⁝
refresh() {
local funtag="[${FUNCNAME[0]}]"

        trace   "$funtag Reloading filters for $ME ..."
        logger  "$ME[$$]: $funtag Reloading filters for $ME ..."
        load_filter "$funtag `date --iso=seconds`" $COMPRESS_FILTER
}
```

The load_filter function in turn will (re)create a new filter function and (re)creates the SUPERFLOUS variable in one rush.

```
load_filter() {
⁝
        … filter rebuild and reload …
⁝
        # Create/refresh superflous_map as well
        SUPERFLOUS=`mk_superflous`
}
```

The refresh is reported to the console and in trace files.

```
[root@vmd28527 WatchLG]# Refresh lg
Refresh: Refreshing WatchLG, PID: 5123

Trace file LG:
⁝
2023-08-04T21:54:29.370 WatchLG[5123]: [refresh] Reloading filters for WatchLG ...
2023-08-04T21:54:29.549 WatchLG[5123]: [load_filter] Using filter compressed
2023-08-04T21:54:29.580 WatchLG[5123]: [mk_superflous] (Re)building superflous_map
⁝
```

In the modules configuration file a variable ‘COMRESS_FILTER=’ can be set to remove all comment lines and empty lines from the rules, that were read. This results in a very compact footprint in memory for the resulting filter function.

The old ‘filter’ file is kept in file ‘filter.old’ and then a ‘context diff’ (diff -c …) across these two files is created to reflect the changes.

```
[root@vmd28527 WatchLG]# ls -l filter*
-rw-r--r-- 1 root root 2648 Aug  4 21:54 filter
-rw-r--r-- 1 root root 1779 Aug  4 21:54 filter.compress
-rw-r--r-- 1 root root  642 Aug  4 21:54 filter-diffs
-rw-r--r-- 1 root root 2646 Aug  2 16:38 filter.old
```

6.2.5.6 Restart

Restart LG|MB|MX|WB|GEO (any case)

With the ‘Restart’ tool you can **restart a module** (or pseudo-module like GeoTrack) separately without restarting (or reloading) the entire Watcher process from ground up.

Modules are quite autonomous components in the Watcher framework with little or no impact on other components. If changes were made to the WatchXX code, the WatchXX.conf file or its ‘private.bashlib’ a module must be restarted that the changes can take effect. Previously this could only achived by restarting or reloading the entire Watcher process.

The ‘Restart’ command nows allows to restart a module separately w/o affecting other modules that are running in the framework.

6.2.5.7 Untreated

Untreated LG|MB|MX|WB (any case)

The ‘Untreated’ command selects the lines marked as ‘[UNTREATED]’ from the trace files of a module’s trace files and lists them on display. [UNTREATED] tags are written to the trace (and also the log files) if the filter could not find a matching rule for the scanned log line from a service (login, mail MTA mailbox service or WEB server).

Such [UNTREATED] tags should be kept very low and you may consider to provide a rule in the module’s ‘.../rule’ directory.. This will suppress noisy messages in the trace and log files of the module for a better overlook of the trace and Watcher logs (<module_path>/WatchXX.trace and /var/log/WatchXX.log).

6.2.6 Goodies

6.2.6.1 hook-watcher

The ‘hook-watcher’ script can be sources in the MASTER_PATH and links the entire Watcher world into your current interactive shell:

```
[root@vmd123606 ~]# cd `Masterpath`
[root@vmd123606 Watcher-1.4]# . hook-watcher
««« Linked into Watcher world »»»
[root@vmd123606 Watcher-1.4]#
```

From here all the global configuration data and functions from common.bashlib are at your fingertips.

[for example the mk-ipset() function from MASTER_PATH common.bashlib]

```
mk-ipset ()
{
    local funtag="[${FUNCNAME[0]}]";
    local setname=$1;
    local settype=$2;
    case $FIREWALL in
        iptables)
            $IPSET -exist create $*;
            if $IPTABLES -t filter -C INPUT -m set --match-set $setname src -j
```

```
DROP > /dev/null 2>&1; then
                : echo "$setname already linked with xtables";
            else
                $IPTABLES -t filter -I INPUT -m set --match-set $setname src -j
DROP > /dev/null 2>&1;
            fi;
            if $IPTABLES -t filter -C FORWARD -m set --match-set $setname src -
j DROP > /dev/null 2>&1; then
                : echo "$setname already linked with xtables";
            else
                $IPTABLES -t filter -I FORWARD -m set --match-set $setname src
-j DROP > /dev/null 2>&1;
            fi
        ;;
        firewalld)
            firewall-cmd --permanent --new-ipset=$setname --type="$settype";
            firewall-cmd --permanent --zone=public --add-source=ipset:$setname;
            firewall-cmd --reload
        ;;
        *)
            logger "[$ME:$$] Error - firewall unknown ...";
            trace "Error - firewall unknown ..."
        ;;
    esac
}
```

Configuration data from ‘watcher.conf’ for instance can also be requested instantly

[e.g. checking the configured ramdisk size]

```
[root@vmd123606 Watcher-1.4]# echo $RDSIZE
20m
```

6.3 Dynloaders

‘Dynamic loaders’ (dynloader) 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 and then turns them into ‘loader format‘ that IPSET understands to configure the firewall.

By that 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 an 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, if nothing has changed on external resources.

```
[root@vmd28527 Pool]# 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 symlink 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 or from $MASTER_PATH with ‘dynloader/<loadername>/Load’

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

```
(‘root’ CRONTAB)

#
#============ Watcher ========================================
#

MASTER_PATH=/root/bin/Watcher

⁞
*/5 * * * *        /usr/local/sbin/Freeme >/dev/null 2>1&
#--- DynLoader   : Once per hour
10 * * * *         $MASTER_PATH/dynload/spamhaus/Load
30 * * * *         $MASTER_PATH/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 the 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
modules/WatchWB/Load
dynload/spamhaus/Load
#dynload/nixspam/Load
dynload/geo/Load
```

6.3.1 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 make changes to it. So subsequent calls to a dynloader must be configured through CRONTAB entries that match the schedule of information by the external providers.

(excerpt of crontab for ‘root’)

```
MASTER_PATH=/root/bin/Watcher
⁞
# --- DynLoader       : Once per hour
10 * * * *     $MASTER_PATH/dynload/spamhaus/Load
30 * * * *     $MASTER_PATH/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:

$POOL/ipset.**Loadfile-<dynloader name>**

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 all at the same minute; i.e. keep them some minutes apart.

```
(crontab ‘root’)
MASTER_PATH=/root/bin/Watcher
⋮
# --- DynLoader       : Once per hour
10 * * * *     $MASTER_PATH/dynload/spamhaus/Load
30 * * * *     $MASTER_PATH/dynload/nixspam/Load
```

6.4 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 own dynloader.

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

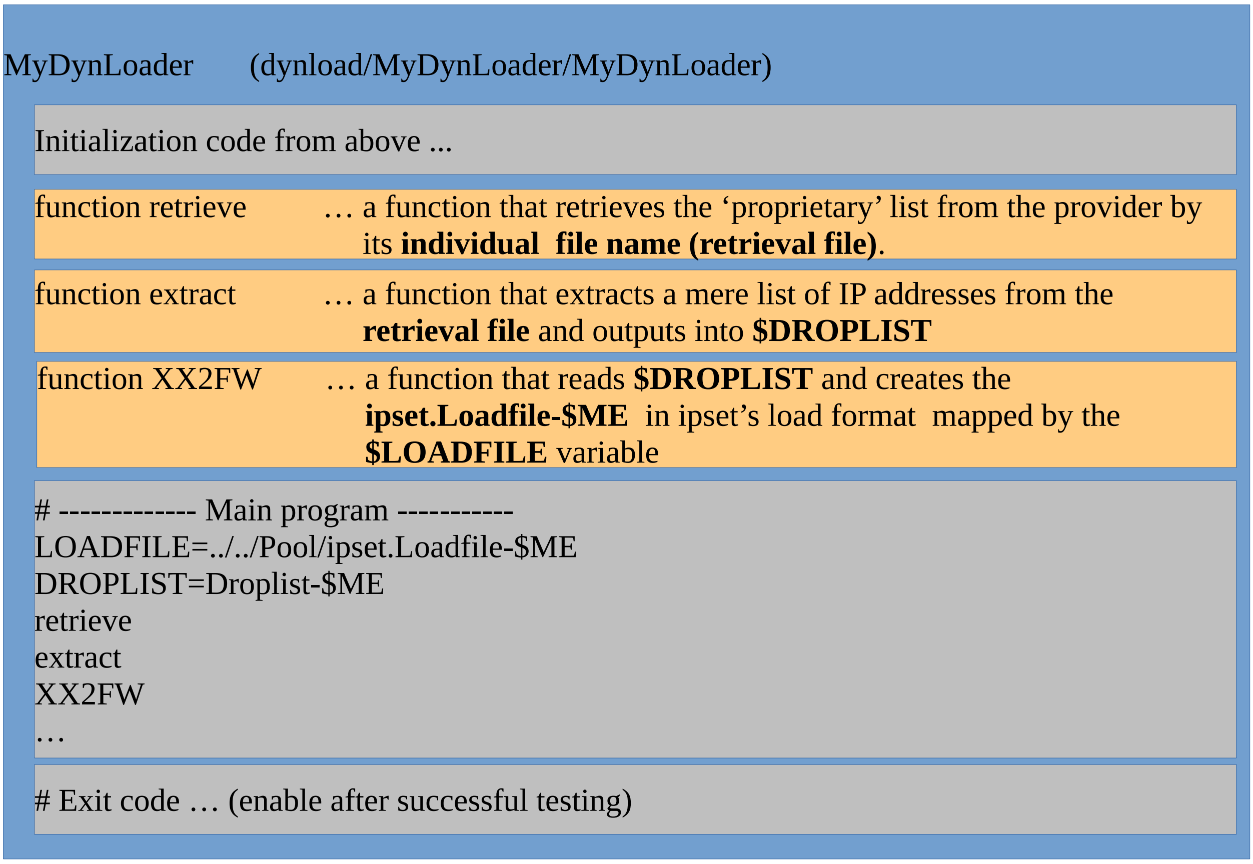
It is assumed that the provider delivers the ‘attacker list’ as a simple *.txt file. Infact it can be any format. But that is abstracted/encapsulated by the ‘extraction function’ in your dynloader code.

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
if [ "$1" == "debug2" ]; then set -xvT ; 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=../../Pool/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 MASTER_PATH.

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

Last but not least create a CRONTAB entry in the crontab file for the superuser ‘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.

7 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 a little 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 below *'/etc/default/…'* or probably 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.

7.1 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 xtables firewall setup (maintained by the ‘iptables’ command 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

*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
# Login ...
-A INPUT -p tcp -m state --state NEW -m tcp --dport  22 -j ACCEPT
# Mail transport & mailbox services ...
-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 110 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 143 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 587 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 993 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 995 -j ACCEPT
# WEB service (httpd) ...
-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
```

7.2 Ubuntu (Server edition)

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 fulfilled you are done.

7.3 SuSE

SuSE is preferring the ‘firewalld’ as the favorite 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 have adopted RPM (RedHat-Package-Manager) as the software cataloging system and uses the system configuration in /etc/sysconfig/… like all RedHat-style systems.

So to get an ‘iptables-services’ package into a SuSE is fairly easy with some easy to manage 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 with ‘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”
   }
   ```

4. The package 'insserv-compat' must be installed for compatibility with the legacy SysV startup scheme.
   # zypper install insserv-compat

5. SuSE Linux keeps the 'iptables' and 'ipset' programs in */usr/sbin* instead of */sbin* where these programs normaly reside.
   So symlinking these two to /sbin/... sets things right:
   # ln -s */usr/sbin/iptables /sbin/iptables*
   # ln -s */usr/sbin/ipset /sbin/ipset*

With these little tweaks the iptables-service rpm package for a RHEL system works perfectly on a SuSE system.

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

```
# sample configuration for iptables service

*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
# Login ...
-A INPUT -p tcp -m state --state NEW -m tcp --dport  22 -j ACCEPT
# Mail transport & mailbox services ...
-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 110 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 143 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 587 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 993 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 995 -j ACCEPT
# WEB service (httpd) ...
-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**.