Watcher

A ‘superordinate’ firewall management solution and realtime intrusion detection and prevention system for Linux server systems

… keep network bandits away …

Watcher Modules Manual

Revision 1.3

Contents

1 Preface

Detect & react …

Watcher modules are the real ‘work horses’ in the Watcher system.

- They provide **real-time intrusion detection**
- They provide **firewall DROP measures** upon detection **in real-time**
- They directly track the system logger stream for a ‘facility’ and measure in **databases** (instead of slow linear file searches and writes) **for maximum performance**.

Watcher modules **run autonomously** if once started by the Watcher master service.

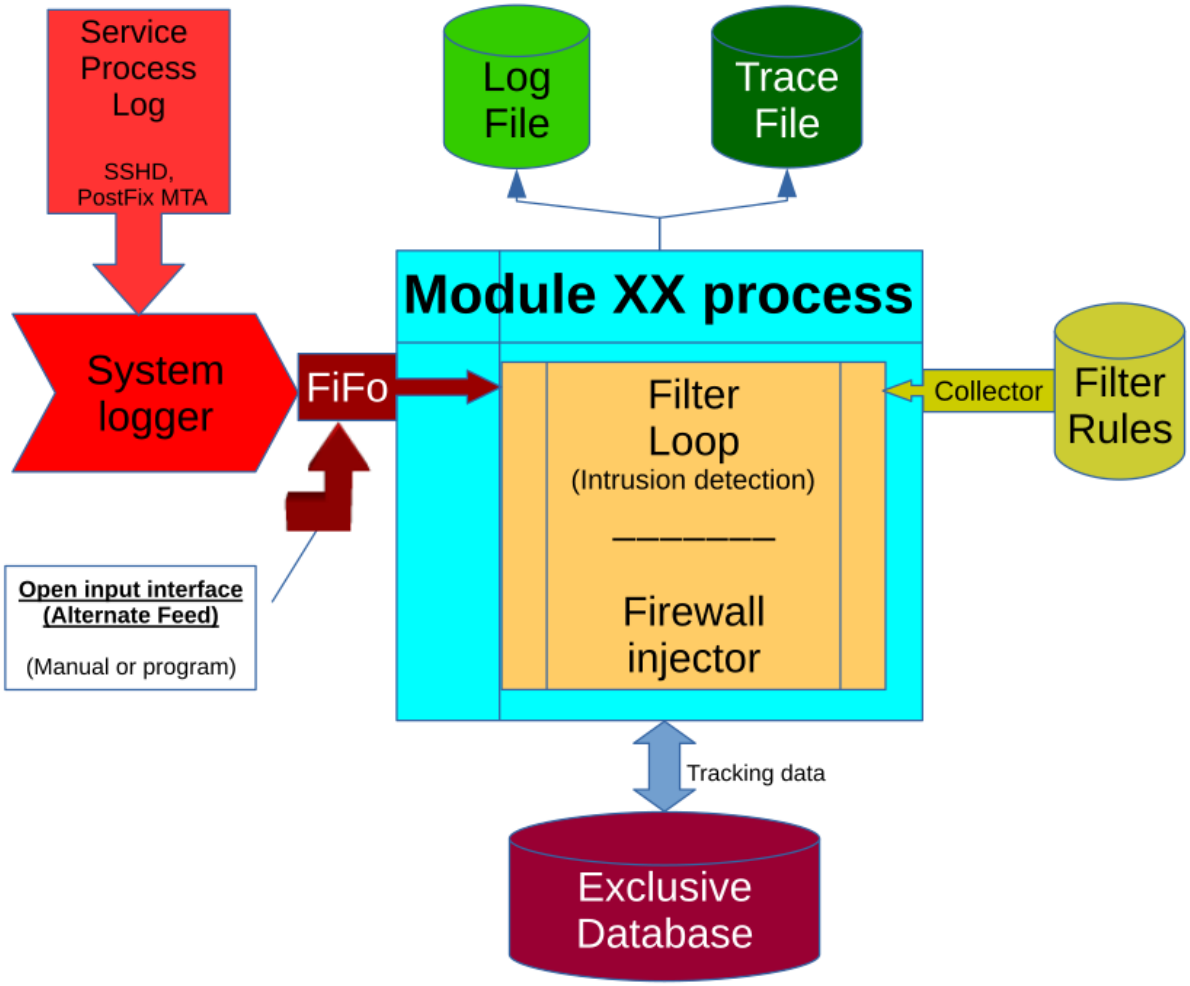
Furthermore each module has its individual:

- **Database expiration** tool (*ExpireXX*) for **database housekeeping** to keep the database compact for best performance
- **Statistics function** (*StatXX*) for measuring efficiency of all measure you have been taking.

Watcher modules are tracking for several services (login, mail transport, mailbox access) the **events of real attacks** for the **distinct system** instead of flooding the firewall with hypothetical lists taken from the internet.

IP addresses will be classified by the way they resolve ‘forward’ and ‘reverse’.
An IP address that does not resolve to an FQDN by any DNS is reported as ‘NXDOMAIN’ (Non-eXistent-DOMAIN) by a DNS; i.e. not registered anywhere.

1.1 Modules - Common architecture



1.2 Watcher modules

The watcher modules WatchLG & WatchMX (with its companion WatchMB with a somewhat simplified architecture) track the input to several log files on the system. Infact they don‘t read the actual log file but are fed directly by the system logger (rsyslog, syslog-ng,…) through FIFOs (‚named pipes‘) .

This way the system logger can feed the Watcher modules in real-time

So there are no troubles with ‚tail reading‘ of system specific files. But the system logger must be configured to write the FIFOs (‚named pipes‘) that the modules need in order to read from them.

Which log files are affected on the particular system is entirely transparent to the Watcher module. The association in the system logger configuration is directly made between the system-logger's 'facility' and the module's FIFO of the specific module. This makes the Watcher module system-independent and no care must be taken, whether the system is *DEBIAN*-, *SuSE*- or *RedHat*-style.

rsyslog … (/etc/rsyslog.conf)

```
...
# The authpriv file has restricted access.
authpriv.*                              /var/log/secure
                                        |/var/log/.pipes/WatchLG

# Log all the mail messages in one place.
mail.*                                  /var/log/maillog
                                        |/var/log/.pipes/WatchMX
...
```

syslog-ng … (/etc/syslog-ng/syslog-ng.conf)

```
...
destination d_auth      {       file("/var/log/secure");
                                pipe("/var/log/.pipes/WatchLG");
                        };
destination d_mail      {       file("/var/log/maillog");
                                pipe("/var/log/.pipes/WatchMX");
                        };
...
```

(Restart the system logger after these changes)

1.2.1 Login watcher (WatchLG)

The 'login watcher" (WatchLG) is the simplest of all watcher modules.

It tracks the **failed login attempts** with a counter in the database for each IP address. If a number of maximum failed attempts (in the Watcher nomenclature called '*affairs*') it records 'DROP' in the database and immediately fires a DROP into the firewall to stop the aggressor instantly. The standard value for MAX_AFFAIRS is 5 – but can be changed in the module's WatchXX.conf file to a value of your choice.

1.2.2 Mail transport watcher (WatchMX)

The mail transport watcher tracks the input to its **exclusive FIFO in $FIFO_BASE/*WatchMX*** as you specified 'FIFO_BASE' in the 'common.conf' file.

The WatchMX module is a lot more complex than the LG watcher module that only has to take care of the system's login process. WatchMX gets everything that is logged by mail related system services (PostFix, Exim, Dbmail, DoveCot …) to the 'mail.*' facility of the system-logger that usually goes to the */var/log/maillog log file.*

Different mailing system (MTAs) with very different messaging to the system logger brings up the need for very individual filter rules for the specific MTA that is used on the system: PostFix, QMAIL, Exim, … etc.

In the module's delivery package the rules are configured for ***PostFix*** [MTA] and ***DBmail*** [mailbox service; POP, IMAP, ...]

1.2.3 Mailbox and SASL access watcher (WatchMB)

WatchMB is the companion of WatchMX.

WatchMB gets maillog messages **passed from WatchMX** that have to do with authentication to mailbox access via POP & IMAP (similar to 'login' tracking) or transport authentication via SASL to the MTA.

If WatchMX sees by a filter rule that this is an issue of break-in attempt the rule forwards the maillog line to the 'named pipe' that WatchMB reads for further processing in WatchMB.

Wrong passwords in mailbox access or wrong certificates in case of SSL/TLS transport requests are good examples for this.

(Rule file of WatchMX in order to forward to WatchMB; i.e. 'pass-on rules' for WatchMB)

```
RULE="Mailbox-Breaker"
Pattern='Error\:\[pop3\]'
#--------
      result=`echo "$REPLY" | grep "$Pattern"`
      if [ ! -z "$result" ]
      then  echo  "$REPLY" >> $FIFO_BASE/WatchMB
            return 2   # Flag forward action with ret-code 2
      fi
#--------

RULE=SMTPS
Pattern='SSL_accept error from unknown\['
#--------
      result=`echo "$REPLY" | grep "$Pattern"`
      if [ ! -z "$result" ]
      then  echo  "$REPLY" >> $FIFO_BASE/WatchMB
            return 2   # Flag forward action with ret-code 2
      fi
#--------
```

Note: WatchMX and the companion process WatchMB share the same database but have separate FIFOs in $FIFO_BASE/… (default: */var/log/.pipes)*

1.2.4 Web access watcher WatchWB

The module WatchWB is new with Watcher revision 1.3.

WatchWB is the most complex scanner in the series, since WEB servers can establish and control **many 'instances' (vhosts)** on a single physical server.

For the sake of security every web server instance must write its own exclusive 'ErrorLog' and 'AccessLog' to which only the customer for this instance (vhost) should have access.

i.e. a logging structure for the individual customers (vhosts) must be established on the web server:

- ➔ ***/var*/www/logs/...**
  - ➔ **.../customer1/...**
    - ➔ error.log
    - ➔ requests.log
  - ➔ **.../customer2/…**
    - ➔ error.log
    - ➔ request.log

  … and so on …

Writing everything into a single central error-log and access-log would be easy for the web server administrator. But this would also means, that all customers must have access to such a central log information and so all customers can see what happens in other customers web instances – which is a serious infringement of common security rules, though.

So the separation of logging information must look like this in the 'vhost-customerXX' definition in the webserver configuration:

```
(Example: /etc/httpd/conf.d/vhost-customer1.conf)
<VirtualHost *:80>
    ServerName  comserve-it-services.de
    ServerAlias www.comserve-it-services.de

    DocumentRoot        /var/www/html/Joomla/comserve
    ErrorLog            /var/www/logs/comserve/error.log
    CustomLog           /var/www/logs/comserve/requests.log combined
.
. and so on ...
.
```

But Watcher modules expect to be fed by their exclusive FIFOs and the trouble with the logging from an Apache web server is, that it can only pipe to PROGRAMs – but not to FIFOs directly.

The solution is having the Apache web server piping to ‘logger’ with an exclusive syslog ‘facility’.

First of all the system logger (*rsyslog* or *syslog-ng*) must be prepared to accept the web server (httpd) log for an exclusive facility. For the purpose the **user defined facility ‘local2’** was chosen.

rsyslog

Provide a line for the chosen facility ‘local2’ in */etc/rsyslog.conf*

```
local2.*                |/var/log/.pipes/WatchWB
```

syslog-ng

Provide a configuration file in */etc/syslog-ng/conf.d/httpd.conf with the following contents:*

```
#
## WEB server facilities ...
#
template iso_date {
   template("${ISODATE} ${HOST} ${MSGHDR}${MSG}\n");
   template_escape(no);
};

destination    d_weblog { pipe("/var/log/.pipes/WatchWB" template(iso_date));  };
filter         f_weblog { facility(local2);    };
log            { source(s_sys);  filter(f_weblog);  destination(d_weblog); };
```

Ahead from this preparation of the system logger the web sites in the ‘vhost-xxxx’ files can be configured …

To connect the web server request logs for an instance a site specific ‘LogFormat’ and ‘CustomLog’ clause is needed in the vhost configuration for the instance.

```
#
# Fed to the systemlogger that provides the timestamp
# So '%t' can/should be ommited from the LogFormat
#
LogFormat "%h %l %u \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\" [Instance: %v:%p]@joomla" instancelog
CustomLog "|/bin/logger -t httpd -p local2.notice" instancelog
```

Add the above line sequence after the 'CustomLog' clause that writes the log for the instance to a local file.

Note, that the LogFormat for this webserver instance must provide a tag for the instance that is logging. This is done by the '[Instance: %v:%p]*@instancetype*' tag where '%v' is the vhost derived from the 'Servername' specified in the vhost configuration file and '%p' is the port number on which the particular request took place. Attach this 'instance tag' at the end of 'LogFormat' line along with the @instancetype marker, where instancetype is the one that you have assigned with the ***WBinstance* tool** to this instance (site) when you have configured your WEB instances in the WatchWB module during the module preparation.

With this preparation of the system logger and configuration of the web server instance in the vhost file the system logger and web server can/must be both restarted:

 # service <system logger> restart

 # service <web server> restart

… or …

 # systemctl restart <system-logger>

 # systemctl restart <webserver>

where:

- <system logger> is the system logger that you are using – either 'rsyslog' or 'syslog-ng'.
- <web server> is the name of the web server service for your particular system.
  - on RHEL and clones this is '*httpd*'
  - on Debian and offsprings (like Ubuntu) this is usually '*apache2*'

Finally check in your *WatchWB.trace* file in the module's directory the logging result that you get:

 # cd <MASTER_PATH>/modules/WatchWB

 # tail -100f WatchWB.trace

… or just call the helper program 'Trace' with a module tag; e.g.

 # Trace {LG|MX|WB}

If everthing is fine do all the same for other web server instances by modifying the individual vhost-xxxx file for the instance as explained above.

1.3 Modules - Common utilities

1.3.1 Database Expiration

With time the database get filled more and more with attacker's IP addresses and DROP information that in turn will fill the firewall more and more.

Experience shows that a lot of break-in attempts are coming from a NXDOMAIN (Non-eXistend-Domain); i.e. an IP addresss that is nowhere registered by a legal Domain-Name-Service (DNS).

Keeping these addresses in the database forever is not such a good idea.

So each watcher module has an 'ExpireXX' program that can be started via crontab on a regular basis to cleanup the database. (where XX is the module token 'LG' or 'MX')

(crontab entries for a weekly cleanup at 00:00 [midnight] on sundays)

0 0 * * 0	<installdir>/modules/<modulename>/ExpireXX
… or ...	
0 0 * * 0	<installdir>/modules/<modulename>/ExpireXX <days>

ExpireLG has a standard value of EXPIRATIONDAYS=30 configured in the WatchLG.conf file.

To override this standard value ExpireLG can take a commandline parameter to run it with a lower value to cut down the level in the database.

To set a higher value it is recommended to increase the value in the configuration file.

The ExpireXX for the particular module dynamically removes the DROP entries in the firewall. So there is no need to restart the watcher service after an ExpireXX program ran.

Note, that each ExpireXX is sym-linked to the flat name 'Expire' and so can be called by its flat name; i.e.

0 0 * * 0	<installdir>/modules/<modulename>/Expire
… or ...	
0 0 * * 0	<installdir>/modules/<modulename>/Expire <days>

1.3.2 Statistics output

Each module has a utility program ‘StatXX’ to output a statistics file for the individual module.

The StatXX program generates a *.csv file from its module database that can be mailed to a configured REPORTMAIL email address. See section “Dealing with statistics files” for details.

To configure delivery of statistics data you need to configure a CRONTAB entry in the super-users’s crontab, that conducts the delivery.

(root’s crontab ...)

```
#--- Statistics   : Once a week
40 02 * * 0 cd /root/bin/Watcher/modules/WatchMX && ./StatMX >/dev/null 2>&1
50 02 * * 0 cd /root/bin/Watcher/modules/WatchLG && ./StatLG >/dev/null 2>&1
```

Like the ‘ExpireXX’ tool the ‘StatXX’ tool is sym-linked to its flat name as ‘Stat’ and so can be called by its flat name ‘Stat’.

```
#--- Statistics   : Once a week
40 02 * * 0 cd /root/bin/Watcher/modules/WatchMX && ./Stat >/dev/null 2>&1
50 02 * * 0 cd /root/bin/Watcher/modules/WatchLG && ./Stat >/dev/null 2>&1
```

2 Installation manual

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

If it comes to installed modules the most important component on the system is the system logger by which the ‘module readers’ are fed – instead of ‘tail reading’ system specific log files in /var/log/…

2.1 Preparation

In each module installation directory (modules/WatchXX/... below the Watcher master path) you will find a program named ***Prep*** like in the installation directory of the Watcher master.

The *Prep* script for a module does not have much to do. But it will initialize the database from the ‘Schema’ template for a particular module.

Just go to the module installation directory and type in: **./Prep [ENTER]**

This will establish the exclusive database for the module.

2.2 System-logger

The modules don’t do any ‘tail reading’ or scanning of log files in /var/log/…

Instead the module processes are directly fed by the system-logger (rsyslog, syslog-ng, …) through FIFOs (‘named pipes’) located on the filesystem with a base path declared as ‘FIFO_BASE’ in the ‘common.conf’ file of the MASTER_PATH as:

- *$FIFO_BASE/*WatchLG
- $FIFO_BASE/WatchMX & $FIFO_BASE/WatchMB (companion process of WatchMX)

The benefit from this is, that **log messages don’t get lost** if any of the Watcher modules is going offline for a while: e.g. for database maintenance, update or whatever. The system logger continues to fill the FIFO with messages that it has picked up for a ‘facility’ from a particular service process; e.g. from the *Postfix* mail transport agent [MTA].

It the Watcher module comes back online and operational after it has been stopped for some maintenance action, then the FIFO is read just with some delay but no loss occurs, since the **FIFOs buffer the messages** from the services.

The FIFO buffer size is determined by a system parameter ‘fs.pipe-max-size’ in the Linux kernel and has a usual size of 1 MiB:

```
# sysctl -a | grep pipe-max
fs.pipe-max-size = 1048576
```

The FIFO size can be temporarily changed/extended by:

```
# sysctl fs.pipe-max-size = <new size in bytes>
```

In order to survive a reboot this must be specified in '/etc/sysctl.conf' to overwrite the system default.

2.3 Logrotate

Watcher modules write individual log- and trace-files.

For housekeeping of the log- and trace-files *logrotate* configuration files should be set up. In particular if tracing is configured to be '*on*' for a module the trace output files can grow relatively quick to tremendous sizes as they track the processing of the **Real-time Intrusion Detection System (RIDS).**

On RedHat-style systems logrotate configuration files are in */etc/logrotate.d.*

By making use of the 2-letter module tokens (LG,MX,WB) and naming conventions all Watcher related logrotates can be formulated in just one file:

[root@vmd28527 logrotate.d]# cat Watcher

```
# /etc/logrotate.d/Watcher
#
# Module log files ...
#
/var/log/Watch??.log {
        monthly
}

#
# Module trace files ... in $MASTER_PATH/modules
#|..MASTER_PATH..|
#vvvvvvvvvvvvv
/root/bin/Watcher/modules/Watch??/Watch??.trace {
        weekly
}
```

2.3.1 Log files

Log files are kept below /var/log/… as <*Module name*>.log

The log file path is configured in *.conf file in the modules installation directory.

If a log file does not exist if the watcher module starts then the particular log file will be created automatically and set to R/W access exclusively for the super-user.

2.3.2 Trace files

Trace files are kept in the **module directory(!)** as <*Module name*>.*trace*

If the trace file does not exist when the watcher module starts then the particular trace file is established and set to R/W access exclusively for the super-user.

Take care that trace files are only being written, it the TRACE variable is set to a NON-EMPTY string in the modules configuration file. i.e. specifying 'TRACE=' turns tracing 'off'; specifying 'TRACE=hooray' turns tracing 'on'. So don't get fooled by setting "TRACE=0" and thinking tracing is now 'off' ...

Tracing is not necessarily needed for normal operation. But tracing is very helpful for checking new individual filter rules.

Take into account, that tracing generates pretty much output and the trace files can grow rapidly to tremendous sizes. The logrotate should probably be set to 'weekly' and/or a 'maxage' no longer than 30 days. Also note that writing that many tracing information to disk will degrade performance of the Watcher module a little.

Furthermore, if tracing is turned 'off' for a module the 'Trace' utility program provides no output and terminates with a notification, that tracing for the module is turned 'off'.

[root@vmd28527 ~]# Trace LG
Tracing for LG is off

2.3.2.1 The UNTREATED rule

The UNTREATED rule is an internally hard-coded rule inside of each module code.

If none of the configured custom rules matched in the ‘filter’ function then the UNTREATED rule acts as a ‘catch all’ and outputs the log line into the module’s trace file:

Examples from an excerpt of the *WatchLG.trace file:*

1	20201115T08:02:23 WatchLG[13117]: [UNTREATED] 'Nov 15 08:02:23 vmd28527 sshd[20561]: Unable to negotiate with **139.162.247.102** port 50038: no matching host key type found. Their offer: ssh-dss [preauth]'
2	20201116T10:51:51 WatchLG[1654]: [UNTREATED] 'Nov 16 09:51:33 vmd28527 polkitd[535]: Loading rules from directory /etc/polkit-1/rules.d' 20201116T10:51:51 WatchLG[1654]: [UNTREATED] 'Nov 16 09:51:33 vmd28527 polkitd[535]: Loading rules from directory /usr/share/polkit-1/rules.d' 20201116T10:51:51 WatchLG[1654]: [UNTREATED] 'Nov 16 09:51:34 vmd28527 polkitd[535]: Finished loading, compiling and executing 9 rules'

If you find “[UNTREATED]” remarks in a module’s trace file you have 2 options:

1. Write a rule, that handles the event in the ‘filter’ function
   Note: This makes only sense if there is an IP address in the log line as shown in example 1.

2. Drop a line into the ‘superflous_map’ file that resides in the ‘rules’ directory along with the rules.
   Note: Entries in the ‘*superflous_map*’ are REGEX expressions and special REGEX characters must be escaped. So to suppress lines from the ‘polkitd’ plus the following process number in square brackets you have to formulate a line as polkitd\[in the *superflous_map* file.

3 Operation Manual

3.1 Writing rules

Prior to revision 1.2 the filter rules were hard-coded in each particular Watcher module and are configured for the log output of ‘Postfix’ and ‘DBmail’.

With Watcher revision 1.2 the **dynamic rule system** was introduced. This means, that rules now reside in external files and will be dynamically assembled to a filter chain, if the module starts.

In Watcher 1.3 the decision block was simplified and replaced by an internal bash command:

‘[[…]]’ along with the expression match operator ‘=~’

This tremendously speeds up the traversal of the ‘rule rope ladder’ in the ‘filter’ function.

Watcher V1.2 - old	Watcher V1.3 - new
`RULE=root-login` `Pattern=": Failed password for root"` `#--------- Decision block ------` `result=`echo "$REPLY"\|grep -E "($Pattern)"` `if [! -z "$result"]` `then : echo "--- Matched rule $RULE ---"` `inject` `return $?` `fi`	`RULE=root-login` `Pattern=": Failed password for root"` `#------- Decision --------` `if [[ “$REPLY” =~ “$Pattern” ]]; then inject; return $?; fi`
• Pattern was actually a REGEX • Transport of the log line by ‘echo’ • Call of transient program ‘grep’	• Pattern is now a simple string • String compare uses an internal bash function • The decision is formulated as a ‘one-liner’
	Benefit: About 30 times faster

The rule sets are still pre-configured for ‘Postfix’ & ‘DBmail’ and can now be adapted to a specific MTA and/or POP/IMAP mailbox service of choice; e.g. if the also popular mailbox service ‘*DoveCot*’ is used on a particular system.

3.1.1 Rule file format

Rules can be ‘order dependent’; i.e. a **more specific** rule has to **precede a more common** rule. Therefore rules are stored in files with a 3-digit number prefix in the range of 000 to 999:

```
[root@vmd28527 rules]# ls -1 [0-9]*
090-Break_in.rule
100-root-login.rule
150-NonPriv-invalid.rule
160-NonPriv-failed-existing.rule
```

It is possible to store several rules in a single rule file as long as the order of rules follows the '*more specific rule before more common* rule' requirement is taken into account.

```
[root@vmd28527 rules]# cat 100-root-login.rule
RULE=root-login
Pattern=": Failed password for root"
#------- Decision --------
if [[ "$REPLY" =~ "$Pattern" ]]; then inject; return $?; fi
```

```
[root@vmd28527 rules]# cat 110-simple_user-login.rule
RULE=simple-login
Pattern=": Failed password for"
#------- Decision --------
if [[ "$REPLY" =~ "$Pattern" ]]; then inject; return $?; fi
```

These two rules can be combined in a single file if you like that better:

```
[root@vmd28527 rules]# cat 100-failed-login.rule
RULE=root-login
Pattern=": Failed password for root"
#------- Decision --------
if [[ "$REPLY" =~ "$Pattern" ]]; then inject; return $?; fi

RULE=simple-login
Pattern=": Failed password for"
#------- Decision --------
if [[ "$REPLY" =~ "$Pattern" ]]; then inject; return $?; fi
```

3.1.2 Testing new or changed rules

The 'rules' directory contains two small **scripts *'check-rule'* and *'check-all-rules'*,** that you can run on the 'rules' directory in order to check, that a rule is syntactically clean or all your rules are syntactically clean. It is strongly advised that you use this syntax-check as the rules are assembled into the 'filter' function by the ./mkfilter script and then the generated 'filter' function is sourced by the module. With syntactically wrong rules the module might fail starting or can behave erratically.

3.2 Dealing with statistics files

Each module provides a statistics script ‘StatXX’.

The module creates an output from your module’s database contents as a relation of ‘introduced’ (detected) and ‘dropped’ IP addresses in the time range that the expiration process of the module has left.

The statistics script creates a *.csv file that is sent to the configured report mail address covered by the $REPORTMAIL variable. This variable is set to a global REPORTMAIL variable in the watcher master configuration ‘watcher.conf’ and can be overwritten in the module’s WatchXX.conf file located in the module path. For instance, if you (as the systems administrator) not interested in the statistics of attacks on the mail system but the mail system administrator wants to see efficiency of the firewall related with the mail system attacks.

To get statistics mailed setup a CRONTAB entry in the super-user’s crontab:

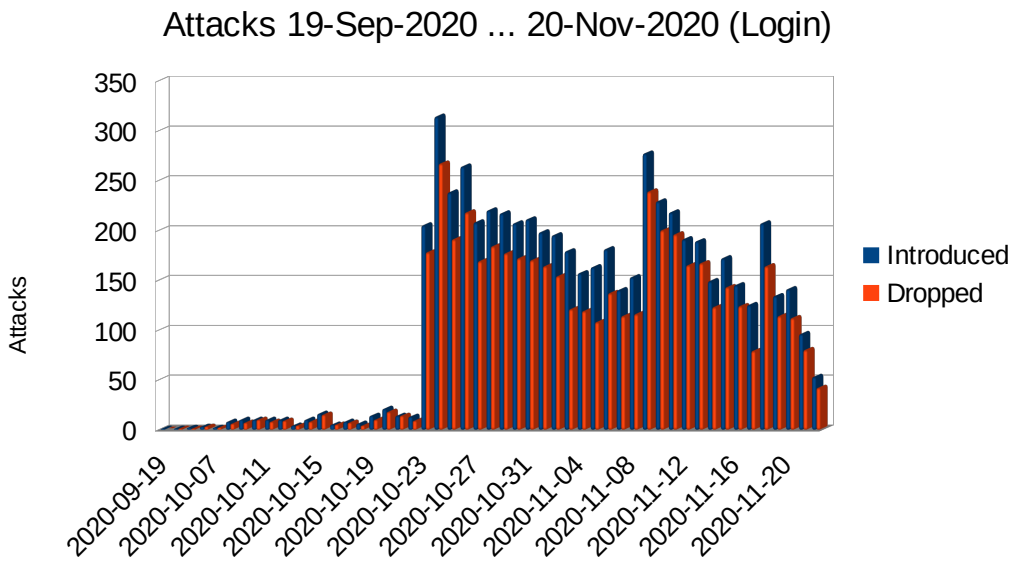
```
#============ Watcher ========================================
...
#--- Statistics  : Once a week
40 02 * * 0     cd /root/bin/Watcher/modules/WatchMX     && ./StatMX >/dev/null 2>&1
50 02 * * 0     cd /root/bin/Watcher/modules/WatchLG     && ./StatLG  >/dev/null 2>&1
```

The configured recipient will then regularly find a Statistics file in *.csv format in his mailbox with the subject: “Statistics-XX <a timestamp>”

By clicking on the attachment your favorite spread-sheet program should open and offer the CSV file to be read into a new spread-sheet.

After the statistics data was read-in you may then select from the ‘diagram functions’ the creation of a diagram of your choice.

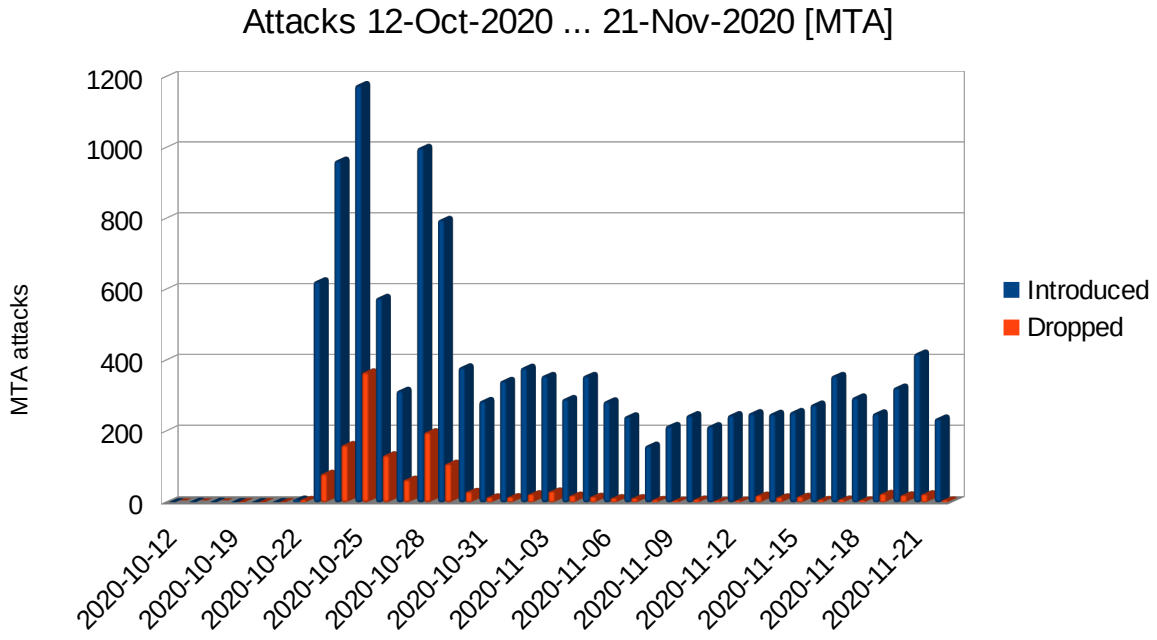
You may then annotate the diagram to your taste, include trendlines, calculated average and so on. Finally you may save and/or print the document and store it to review the efficiency of your measures.



Sample statistics diagram (Login, LG module data)

3.2.1 Interpreting statistics diagrams

Below is a statistics diagram with data from the MX (mail) module's database.



(Sample mail attacks diagram)

The diagram looks a bit odd at first view, since it shows a tremendous number of ‘detects’ (*introduced*) with a relatively little number of firewall DROPs – but this is fairly normal for mail attacks. SPAMers tend to use the scheme ‘*fire-and-forget*’; i.e. they fire their attempt once against a mail server and never come back as legal MTAs usually do to retry the mail transfer. Although NXDOMAINs (and FAKEHOSTs as well) get a preset of *MAXAFFAIRS-1* that results in a firewall DROP on second attempt there is no second attempt that would cause the DROP in the firewall for the incoming IP address. This is why the ‘introduce’ values are so high.

4 Troubleshooting

Nearly all programs in the Watcher suite have a particular heading in the two lines straight below the ‘shebang’ .

```
#!/bin/bash
if [ "$1" == 'debug'  ]; then set -x;   shift; fi
if [ "$1" == 'debug2' ]; then set -xvT; shift; fi
```

This means, that you may call a program, with a first parameter of either ‘debug’ or ‘debug2’. For programs that usually take parameter there parameters then follow the ‘debug/debug2’ key as usual.

- ‘debug’ will just turn on the usual Bash execution tracing.
- ‘debug2 additionally turns on verbose Bash tracing (code dump) and tracing of functions.

So one can see what the code is doing on a particular platform (RHEL, Debian, …) or you can debug a self-constructed dynloader to fitness.

4.1 Troubleshooting a module

For the modules (WatchLG, WatchMX, …) some care must be takes as modules read from their assigned FIFOs and there can only be one reader process on the FIFOs end – otherwise multiple readers would steal FIFO contents from one another and the result is reading trash.

To troubleshoot a module there a two possibilities.

1. Stop watcher entirely: ‘# service watcher stop’
   The drawback is, that all module processes stop processing their FIFOs, which you probably not want in the first place. But this option makes sense, if a bug in the common code (*common.bashlib*) is suspected.
2. Kill a specific Watcher process: ‘# killall Watch*XX*’

Note, that regardless which way you stop a Watcher module, this will not affect the firewall. The services also continue reporting to the system logger which now buffers all messages that are coming from the services. There is no loss …

To do your realtime tracing of the module to see for the trouble go to the module path:

```
# cd $MASTERPATH/module/WatchXX
```

Modules take no parameters as they are configured by their corresponding *.conf file inside the module path. So you might check if the *.conf file is free from syntax errors at first simply 'sourcing' it:

```
source  WatchXX.conf
```

If there are no syntax errors pointed out everything is ready for a debug session.

Then start the module manually in a debug mode of your choice:

```
./WatchXX debug … (or debug2)
```

A lot of information will rush across the screen and even a lot more , if debug2 was chosen as the debug mode as this tracks traversing of functions as well. Scroll back in the output and see for any errors that Bash or any of the 'transient calls' to external programs have caused.

You might want to open a second terminal to see what the 'trace' functions write to the modules *.trace file. Then start a 'Trace XX' command in this other terminal to get a condensed output from the debug session.