The iptables-liveupdate.py utility script

How to modify iptables configurations
with no service impact on non-firewalld Linux installations.

Table of Contents

Introduction

On many Linux distributions, IP packet filtering configuration is controlled by static text files. When this is the case, the standard method of updating the currently active IP filtering configuration relies on clearing the current in-kernel filtering tables and re-initializing them from the static text files.

There exist several popular software packages that rely on dynamic filtering modifications for their operation, with no record of these modifications in the static text files. Examples of such packages are Docker and Kubernetes, which can install filtering rules on-the-fly depending on what applications or containers are active on a given Linux host or VM. As a result, on hosts/VMs that run these packages, *using the standard method to update in-kernel filtering configuration can cause service impacts by causing the loss of dynamic filtering rules which are required for proper operation of specific application components*.

It is possible to maintain in-kernel iptables configuration by hand, using commands **iptables/ip6tables** with careful use of options "-A|-I|-R|-D" (append/insert/replace/delete). This can be very error-prone and certainly impractical for modern deployments spanning many Linux hosts/VMs.

In these conditions, the script **iptables-liveupdate.py** can be used to simplify the routine maintenance of iptables filtering rules.

iptables-liveupdate.py is designed to be reasonably easy to integrate into larger automated installation environments: it is self-contained, written/tested in Python 2.7.x, and looks for iptables configuration files in a list of standard locations easily customizable in variable **filter_variants** located at the beginning of the script. The iptables configuration files themselves only need to comply to the standard syntax requirements of commands **iptables-restore/ip6tables-restore**, and may be maintained by hand or generated by any number of automated means.

1.0 Reasons to use iptables-liveupdate.py

- You need to run Linux distributions that store iptables configurations in static text files
- You need to be able to modify iptables configurations **without** clearing/reloading them

Here are some Linux distributions that use static text files for iptables configuration:

- CentOS and RedHat Enterprise Linux up to release 6, with the filtering configuration stored in files **/etc/sysconfig/iptables** and **/etc/sysconfig/ip6tables**
- CentOS and RedHat Enterprise Linux release 7, if the standard **firewalld** package is disabled and legacy package **iptables-services** is installed and activated.
- Debian Linux, with the filtering configuration stored in files **/etc/iptables/rules.v4** and **/etc/iptables/rules.v6**

2.0 Additional scripts supplied with iptables-liveupdate.py

Additional scripts are supplied to allow simple SSH-driven iptables management targeting multiple Linux hosts/VMs, which must be described in a file in JSON format. These scripts can be useful in an environment where no modern configuration management/orchestration tool is used (ex. Puppet, Chef, Ansible, etc.) The scripts in question are as follows, and should all be installed in the same directory :

Script	Description
iptables-liveupdate-driver.py	Uploads a temporary copy of **iptables-liveupdate.py** on multiple Linux hosts/VMs and executes it on each host.
iptables-backup.py	Connects to multiple hosts/VMs and downloads their “stored” and “live-in-kernel” iptables configs into a local directory.
relearn-known-hosts.py	Useful in environments where the network is trusted and multiple Linux hosts/VMs are frequently re-installed or often have new SSH host keys generated. Removes the locally-stored SSH host keys for the target hosts/VMs and re-learns them by attempting SSH connections to each host/VM.
util.py	A module used by the four scripts above.
iptables-liveupdate.py	Script with the main functionality, can be invoked independently or used by **iptables-liveupdate-driver.py** to manage multiple hosts.

3.0 Running iptables-liveupdate.py directly on a host

To run script **iptables-liveupdate.py** on a Linux host, the requirements are as follows:

- iptables-liveupdate.py must be installed in some suitable directory (ex. /usr/local/sbin)
- the host has iptables installed and active, with some configuration files present
- root privilege, either directly or through **sudo**

The script provides on-line help through the options "**-h**" or "**--help**", as follows:

```
usage: iptables-liveupdate.py [-h] [--loglevel {ERROR,WARNING,INFO,DEBUG}]
                              [--live-update]

Performs checks on the host packet filtering configurations. By default, only
read-only checks are performed. If 'live-update' is specified, and the host
has 'stored' iptables rules that are not 'live' in-kernel, the script attempts
to insert the rules 'live' in-kernel if it appears safe to do so. If 'stored'
rules are commented out with the '#DELETE#' prefix and are found in-kernel,
the script attempts to delete the rules.

optional arguments:
  -h, --help            show this help message and exit
  --loglevel {ERROR,WARNING,INFO,DEBUG}
                        set the log verbosity level, default=INFO
  --live-update         modify remote IPtables configuration on the fly
```

Note that the script will **ignore** filtering rules that are **active** in the kernel but are **absent** from the stored configuration in text files. This is an intentional design feature of the script that reflects its intended usage model: to help manage Linux hosts that have running applications that dynamically insert/delete iptables rules in the kernel tables.

3.1 Verifying consistency of stored and in-kernel configurations

In cases where the in-kernel configuration matches the stored configuration, by default, the script generates very little output:

```
[root@centos-vm1 ~]# ./iptables-liveupdate.py
timestamp INFO     Running IPtables analysis script on target host...
```

Re-running the script with debug-level logging shows a lot more of what is happening:

```
[root@centos-vm1 ~]# ./iptables-liveupdate.py --loglevel DEBUG
timestamp INFO     Running IPtables analysis script on target host...
timestamp DEBUG    IPtables version supports the --wait option.
timestamp DEBUG    Computed ephemeral time string: 1501796554
timestamp DEBUG    Parsing exactly one IPtables file out of:
```

```
                       [ /etc/iptables/rules.v4 /etc/sysconfig/iptables ]
timestamp DEBUG    Rules for table 'filter':
timestamp DEBUG    Chain FORWARD, policy ACCEPT, NumRules=1
timestamp DEBUG    Chain INPUT, policy ACCEPT, NumRules=8
timestamp DEBUG    Chain OUTPUT, policy ACCEPT, NumRules=0
timestamp DEBUG    Parsing of IPv4 config file successful.
timestamp DEBUG    Verifying that all chains in stored rules are 'at least' defined in the kernel.
timestamp DEBUG    Creating temporary IPv4 filtering chains in-kernel.
timestamp DEBUG    Reading/Parsing in-kernel rules, table filter .
timestamp DEBUG    Reading/Parsing in-kernel rules, chain INPUT-1501796554 ..
timestamp DEBUG    Reading/Parsing in-kernel rules, chain FORWARD-1501796554 ..
timestamp DEBUG    Reading/Parsing in-kernel rules, chain OUTPUT-1501796554 ..
timestamp DEBUG    Reading/Parsing stored rules, table filter .
timestamp DEBUG    Reading/Parsing in-kernel rules, chain FORWARD ..
timestamp DEBUG    Reading/Parsing in-kernel rules, chain INPUT ..
timestamp DEBUG    Reading/Parsing in-kernel rules, chain OUTPUT ..
timestamp DEBUG    Removing temporary IPv4 filtering chains from kernel.
timestamp DEBUG    Checking Stored/In-kernel diffs for chain 'IPv4/filter/FORWARD' :
timestamp DEBUG    For chain IPv4/filter/FORWARD, default policy matches for Stored vs. In-Kernel-Actual.
timestamp DEBUG    For chain IPv4/filter/FORWARD, NumRules match for Stored vs. In-Kernel-TmpLoad.
timestamp DEBUG    For chain IPv4/filter/FORWARD, Found no duplicates in In-Kernel-TmpLoad rules.
timestamp DEBUG    For chain IPv4/filter/FORWARD, all In-Kernel-TmpLoad rules matched in Actual,
                                              no rules to delete, no further checks needed.
timestamp DEBUG    Checking Stored/In-kernel diffs for chain 'IPv4/filter/INPUT' :
timestamp DEBUG    For chain IPv4/filter/INPUT, default policy matches for Stored vs. In-Kernel-Actual.
timestamp DEBUG    For chain IPv4/filter/INPUT, NumRules match for Stored vs. In-Kernel-TmpLoad.
timestamp DEBUG    For chain IPv4/filter/INPUT, Found no duplicates in In-Kernel-TmpLoad rules.
timestamp DEBUG    For chain IPv4/filter/INPUT, all In-Kernel-TmpLoad rules matched in Actual,
                                            no rules to delete, no further checks needed.
timestamp DEBUG    Checking Stored/In-kernel diffs for chain 'IPv4/filter/OUTPUT' :
timestamp DEBUG    For chain IPv4/filter/OUTPUT, default policy matches for Stored vs. In-Kernel-Actual.
timestamp DEBUG    For chain IPv4/filter/OUTPUT, NumRules match for Stored vs. In-Kernel-TmpLoad.
timestamp DEBUG    For chain IPv4/filter/OUTPUT, no rules in In-Kernel-TmpLoad, no further checks needed.,
timestamp DEBUG    Parsing exactly one IPtables file out of:
                       [ /etc/iptables/rules.v6 /etc/sysconfig/ip6tables ]
timestamp DEBUG    Rules for table 'filter':
timestamp DEBUG    Chain FORWARD, policy ACCEPT, NumRules=1
timestamp DEBUG    Chain INPUT, policy ACCEPT, NumRules=6
timestamp DEBUG    Chain OUTPUT, policy ACCEPT, NumRules=0
timestamp DEBUG    Parsing of IPv6 config file successful.
timestamp DEBUG    Verifying that all chains in stored rules are 'at least' defined in the kernel.
timestamp DEBUG    Creating temporary IPv6 filtering chains in-kernel.
timestamp DEBUG    Reading/Parsing in-kernel rules, table filter .
timestamp DEBUG    Reading/Parsing in-kernel rules, chain INPUT-1501796554 ..
timestamp DEBUG    Reading/Parsing in-kernel rules, chain FORWARD-1501796554 ..
timestamp DEBUG    Reading/Parsing in-kernel rules, chain OUTPUT-1501796554 ..
timestamp DEBUG    Reading/Parsing stored rules, table filter .
timestamp DEBUG    Reading/Parsing in-kernel rules, chain FORWARD ..
timestamp DEBUG    Reading/Parsing in-kernel rules, chain INPUT ..
timestamp DEBUG    Reading/Parsing in-kernel rules, chain OUTPUT ..
timestamp DEBUG    Removing temporary IPv6 filtering chains from kernel.
timestamp DEBUG    Checking Stored/In-kernel diffs for chain 'IPv6/filter/FORWARD' :
timestamp DEBUG    For chain IPv6/filter/FORWARD, default policy matches for Stored vs. In-Kernel-Actual.
timestamp DEBUG    For chain IPv6/filter/FORWARD, NumRules match for Stored vs. In-Kernel-TmpLoad.
timestamp DEBUG    For chain IPv6/filter/FORWARD, Found no duplicates in In-Kernel-TmpLoad rules.
timestamp DEBUG    For chain IPv6/filter/FORWARD, all In-Kernel-TmpLoad rules matched in Actual,
                                              no rules to delete, no further checks needed.
timestamp DEBUG    Checking Stored/In-kernel diffs for chain 'IPv6/filter/INPUT' :
timestamp DEBUG    For chain IPv6/filter/INPUT, default policy matches for Stored vs. In-Kernel-Actual.
timestamp DEBUG    For chain IPv6/filter/INPUT, NumRules match for Stored vs. In-Kernel-TmpLoad.
timestamp DEBUG    For chain IPv6/filter/INPUT, Found no duplicates in In-Kernel-TmpLoad rules.
timestamp DEBUG    For chain IPv6/filter/INPUT, all In-Kernel-TmpLoad rules matched in Actual,
                                            no rules to delete, no further checks needed.
timestamp DEBUG    Checking Stored/In-kernel diffs for chain 'IPv6/filter/OUTPUT' :
timestamp DEBUG    For chain IPv6/filter/OUTPUT, default policy matches for Stored vs. In-Kernel-Actual.
timestamp DEBUG    For chain IPv6/filter/OUTPUT, NumRules match for Stored vs. In-Kernel-TmpLoad.
timestamp DEBUG    For chain IPv6/filter/OUTPUT, no rules in In-Kernel-TmpLoad, no further checks needed.
```

3.2 Example: Using iptables-liveupdate.py to add filtering rules

To demonstrate the functionality on this sample CentOS configuration, let us edit file **/etc/sysconfig/iptables** and add three new rules, shown in bold in this output:

```
#
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p udp -m state --state NEW -m udp --dport 53   -j ACCEPT
-A INPUT -p udp -m state --state NEW -m udp --dport 123  -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22   -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 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 8000 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 8080 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Running the script in read-only mode shows the detection of the rules that are stored in the text files but are absent from the in-kernel iptables configuration:

```
[root@centos-vm1 ~]# ./iptables-liveupdate.py
timestamp INFO     Running IPtables analysis script on target host...
timestamp INFO     For chain IPv4/filter/INPUT, 2 rules in Stored config can be safely inserted
                       live in-kernel within other matched rules.
timestamp INFO     For chain IPv4/filter/INPUT, 1 rule in Stored config can be safely inserted
                       live in-kernel within other matched rules.
timestamp WARNING  Plan for rule modifications for chain 'IPv4/filter/INPUT' :
                       (ALL DELETEs will be performed first! )
timestamp WARNING  For chain IPv4/filter/INPUT, insert at position 4:
                       '-p udp -m state --state NEW -m udp --dport 53   -j ACCEPT'
timestamp WARNING  For chain IPv4/filter/INPUT, insert at position 5:
                       '-p udp -m state --state NEW -m udp --dport 123  -j ACCEPT'
timestamp WARNING  For chain IPv4/filter/INPUT, insert at position 9:
                       '-p tcp -m state --state NEW -m tcp --dport 8000 -j ACCEPT'
timestamp INFO     Solutions found for all unmatched Stored rules.
```

The script output suggests that live modification of the iptables rules is possible and safe, so we run the script in live-update mode:

```
[root@centos-vm1 ~]# ./iptables-liveupdate.py --live-update
timestamp INFO     Running IPtables analysis script on target host...
timestamp INFO     For chain IPv4/filter/INPUT, 2 rules in Stored config can be safely inserted
                       live in-kernel within other matched rules.
timestamp INFO     For chain IPv4/filter/INPUT, 1 rule in Stored config can be safely inserted
                       live in-kernel within other matched rules.
timestamp WARNING  Plan for rule modifications for chain 'IPv4/filter/INPUT' :
                       (ALL DELETEs will be performed first! )
timestamp WARNING  For chain IPv4/filter/INPUT, insert at position 4:
                       '-p udp -m state --state NEW -m udp --dport 53   -j ACCEPT'
```

```
timestamp WARNING  For chain IPv4/filter/INPUT, insert at position 5:
                       '-p udp -m state --state NEW -m udp --dport 123  -j ACCEPT'
timestamp WARNING  For chain IPv4/filter/INPUT, insert at position 9:
                       '-p tcp -m state --state NEW -m tcp --dport 8000 -j ACCEPT'
timestamp INFO     Solutions found for all unmatched Stored rules.
timestamp WARNING  Starting live update for 'IPv4/filter/INPUT' ...
timestamp WARNING  Live update completed.
```

Running the script again shows that the in-kernel rules have now been updated:

```
[root@centos-vm1 ~]# ./iptables-liveupdate.py
timestamp INFO     Running IPtables analysis script on target host...
```

OK, so this is not a big deal, right?

This test Linux host happens to be running Docker… Doing a raw output of the iptables configuration, just for the “**filter**” table, gives a more accurate picture of the situation. If we had naively executed the command “**service iptables restart**” to add our three rules, all the lines in bold would have been lost!

```
[root@centos-vm1 ~]# iptables -t filter --list-rules
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
-N DOCKER
-N DOCKER-ISOLATION
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p udp -m state --state NEW -m udp --dport 53 -j ACCEPT
-A INPUT -p udp -m state --state NEW -m udp --dport 123 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 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 8000 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 8080 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j DOCKER-ISOLATION
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
-A DOCKER-ISOLATION -j RETURN
```

Note that the Docker service also creates dynamic rules in the “**nat**” table, which would also have been lost if we had executed “**service iptables restart**”.

3.3 Example: Using iptables-liveupdate.py to remove filtering rules

As mentioned previously, by default, a filtering rule that is present in the kernel but absent from the configuration files is ignored. To request deletion of in-kernel rules, they must be present in the configuration files, but prefixed with the string “#DELETE#”.

For example, to delete the three rules added in the previous example, we modify /etc/sysconfig/iptables as follows:

```
#
*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
#DELETE# -A INPUT -p udp -m state --state NEW -m udp --dport 53   -j ACCEPT
#DELETE# -A INPUT -p udp -m state --state NEW -m udp --dport 123  -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22   -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 80   -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 443  -j ACCEPT
#DELETE# -A INPUT -p tcp -m state --state NEW -m tcp --dport 8000 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 8080 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

In this case, running the script in live-update mode produces the following output:

```
[root@centos-vm1 ~]# ./iptables-liveupdate.py --live-update
timestamp INFO     Running IPtables analysis script on target host...
timestamp WARNING  Plan for rule modifications for chain 'IPv4/filter/INPUT' :
                       (ALL DELETEs will be performed first! )
timestamp WARNING  For chain IPv4/filter/INPUT, delete at position 4:
                       '-p udp -m state --state NEW -m udp --dport 53   -j ACCEPT'
timestamp WARNING  For chain IPv4/filter/INPUT, delete at position 4:
                       '-p udp -m state --state NEW -m udp --dport 123  -j ACCEPT'
timestamp WARNING  For chain IPv4/filter/INPUT, delete at position 7:
                       '-p tcp -m state --state NEW -m tcp --dport 8000 -j ACCEPT'
timestamp INFO     Solutions found for all unmatched Stored rules.
timestamp WARNING  Starting live update for 'IPv4/filter/INPUT' ...
timestamp WARNING  Live update completed.
```

And running it again shows that filtering lines flagged with #DELETE# are ignored if absent from the in-kernel rules:

```
[root@centos-vm1 ~]# ./iptables-liveupdate.py --live-update
timestamp INFO     Running IPtables analysis script on target host...
```

3.4 Structuring modifications that iptables-liveupdate.py can handle

The script implements a simplified algorithm similar to the Linux “**diff**” command. To achieve this, the script looks for *anchor points*:

In this context, an *anchor point* is defined as a contiguous group of one or more filtering rules that are present, exactly identical, both in the kernel and in the stored configuration files.

In situations where the script fails to find satisfactory *anchor points*, it will report errors and refuse to modify the in-kernel iptables configuration.

Accordingly, for each group of rules you wish to insert or delete from an iptables chain, ensure that the group is bracketed above and below by an *anchor point*. A rule group can also be successfully inserted/deleted at the very beginning of an iptables chain if immediately followed by an *anchor point*. This also works for a rule group at the very end of an iptables chain if immediately preceded by an *anchor point*.

Note that iptables-liveupdate.py does not support the presence of multiple identical rules in the stored configuration files. This could confuse the built-in “diff” algorithm and produce unexpected results. The script checks for this condition and reports an error if it occurs.

4.0 Targeting multiple hosts with the additional scripts

If your installation can derive benefits from the additional scripts, they should all be installed in the same directory, as mentioned earlier. So you should have a directory containing the following five files :

iptables-liveupdate-driver.py
iptables-liveupdate.py
iptables-backup.py
relearn-known-hosts.py
util.py

4.1 Specifying a list of hosts in a JSON config file

Here is a sample JSON config file for three hosts:

```
[
  { "name":"host-1", "ip":"10.128.129.130", "user":"admin",   "state":"on" },
  { "name":"host-2", "ip":"172.16.1.1",     "user":"SRCUSER", "state":"on" },
  { "name":"host-3", "ip":"192.168.1.1",    "user":"root",    "state":"off" }
]
```

For each host in the file, four fields must be specified:

Field	Description
ip	Contains the IP address used to setup an SSH connection to the host. Intended to support IPv4 and IPv6 addresses, but only IPv4 addresses have been tested.
name	Contains a descriptive name for the host, and is used during logging, If you use script **relearn-known-hosts.py**, it helps for this name to be resolvable using the locally-available name resolution system (usually DNS), since that script attempts to learn the SSH host key of a host by connecting **both** to the specified IP address **and** to the specified name.
user	During SSH or SCP connections, the remote user to specify for the target host, which should be **root** or some user that has **SUDO ALL** privileges. For ease of use with the scripts, it is preferable that the target user be accessible without a password, typically with SSH key-based authentication. If set to **SRCUSER**, it means to connect to the target host using the same user name that is running the scripts locally.
state	Scripts will only process a host in the list if this field is set to **on**. Other values will cause the host to be silently ignored by the scripts.

4.2 Taking a backup of filtering configs with iptables-backup.py

In this example, we have a file called **host-list.json** that contains the definition of two hosts named **c-01** and **d-01**, running CentOS and Debian respectively. Running the script successfully would look like this:

```
$ ./iptables-backup.py host-list.json
timestamp  INFO     Starting IPtables backup script
timestamp  INFO     Target directory for backup is ./iptables-backup-20170804-021857 ...
timestamp  INFO     Attempting to get IPtables config for host 'd-01' ...
scp: /etc/sysconfig/iptables: No such file or directory
scp: /etc/sysconfig/ip6tables: No such file or directory
timestamp  INFO     Attempting to get IPtables config for host 'c-01' ...
scp: /etc/iptables/rules.v4: No such file or directory
scp: /etc/iptables/rules.v6: No such file or directory
```

```
$ ls -lR ./iptables-backup-20170804-021857
./iptables-backup-20170804-021857:
total 8
drwxrwxr-x. 2 myuser mygrp 4096 Aug  3 20:19 c-01
drwxrwxr-x. 2 myuser mygrp 4096 Aug  3 20:19 d-01

./iptables-backup-20170804-021857/c-01:
total 48
-rw-------. 1 myuser mygrp 566 Aug  3 20:19 centos-v4-rules
-rw-------. 1 myuser mygrp 709 Aug  3 20:19 centos-v6-rules
-rw-rw-r--. 1 myuser mygrp 818 Aug  3 20:19 in-kernel-v4-rules-filter
-rw-rw-r--. 1 myuser mygrp  94 Aug  3 20:19 in-kernel-v4-rules-mangle
-rw-rw-r--. 1 myuser mygrp 295 Aug  3 20:19 in-kernel-v4-rules-nat
-rw-rw-r--. 1 myuser mygrp  38 Aug  3 20:19 in-kernel-v4-rules-raw
-rw-rw-r--. 1 myuser mygrp  51 Aug  3 20:19 in-kernel-v4-rules-security
-rw-rw-r--. 1 myuser mygrp 418 Aug  3 20:19 in-kernel-v6-rules-filter
-rw-rw-r--. 1 myuser mygrp  94 Aug  3 20:19 in-kernel-v6-rules-mangle
-rw-rw-r--. 1 myuser mygrp  76 Aug  3 20:19 in-kernel-v6-rules-nat
-rw-rw-r--. 1 myuser mygrp  38 Aug  3 20:19 in-kernel-v6-rules-raw
-rw-rw-r--. 1 myuser mygrp  51 Aug  3 20:19 in-kernel-v6-rules-security

./iptables-backup-20170804-021857/d-01:
total 48
-rw-r--r--. 1 myuser mygrp 183 Aug  3 20:18 debian-v4-rules
-rw-r--r--. 1 myuser mygrp 184 Aug  3 20:18 debian-v6-rules
-rw-rw-r--. 1 myuser mygrp  51 Aug  3 20:19 in-kernel-v4-rules-filter
-rw-rw-r--. 1 myuser mygrp  94 Aug  3 20:19 in-kernel-v4-rules-mangle
-rw-rw-r--. 1 myuser mygrp  76 Aug  3 20:19 in-kernel-v4-rules-nat
-rw-rw-r--. 1 myuser mygrp  38 Aug  3 20:19 in-kernel-v4-rules-raw
-rw-rw-r--. 1 myuser mygrp  51 Aug  3 20:19 in-kernel-v4-rules-security
-rw-rw-r--. 1 myuser mygrp  51 Aug  3 20:19 in-kernel-v6-rules-filter
-rw-rw-r--. 1 myuser mygrp  94 Aug  3 20:19 in-kernel-v6-rules-mangle
-rw-rw-r--. 1 myuser mygrp  76 Aug  3 20:19 in-kernel-v6-rules-nat
-rw-rw-r--. 1 myuser mygrp  38 Aug  3 20:19 in-kernel-v6-rules-raw
-rw-rw-r--. 1 myuser mygrp  51 Aug  3 20:19 in-kernel-v6-rules-security
```

4.3 Running iptables-liveupdate-driver.py to check/update filtering on multiple hosts

Once a proper JSON config file has been built containing multiple host definitions, script **iptables-liveupdate-driver.py** can be used to maintain iptables configurations on all the hosts with one command:

```
$ ./iptables-liveupdate-driver.py -h
usage: iptables-liveupdate-driver.py [-h]
                                     [--loglevel {ERROR,WARNING,INFO,DEBUG}]
                                     [--timeout TIMEOUT] [--live-update]
                                     json_config_file_name

On a list of remote hosts, check differences between 'stored' and 'in-kernel'
IPtables IPv4 and IPv6 configurations. For this to work, for each host, the
username used must have SCP write-access to its home directory (without SUDO),
and SUDO ALL access. If '--live-update' is specified, the script ALSO modifies
the 'in-kernel' IPtables rules to make them satisfy the requirements of the
'stored' IPtables rules.

positional arguments:
  json_config_file_name
                        filename of JSON configuration with host list

optional arguments:
  -h, --help            show this help message and exit
  --loglevel {ERROR,WARNING,INFO,DEBUG}
                        set the log verbosity level, default=INFO
  --timeout TIMEOUT     set the SSH connection timeout in seconds, default=5
  --live-update         modify remote IPtables configuration on the fly



$ ./iptables-liveupdate-driver.py host-list.json
timestamp INFO     Starting IPtables liveupdate driver script...
timestamp INFO     Acting on host 'd-01' with address 'sample-ip-address-1' ...
timestamp INFO     Running IPtables analysis script on target host...
timestamp INFO     Acting on host 'c-01' with address 'sample-ip-address-2' ...
timestamp INFO     Running IPtables analysis script on target host...
```

4.4 Re-learning SSH host keys with script relearn-known-hosts.py

This script has substantial implications for SSH end-to-end security. For environments where its use is justified, here is an example of its execution:

```
$ ./relearn-known-hosts.py -h
usage: relearn-known-hosts.py [-h] [--loglevel {ERROR,WARNING,INFO,DEBUG}]
                              [--timeout TIMEOUT]
                              json_config_file_name

Forget and re-learn destination SSH host keys. Useful in environments where
the network is trusted and hosts are frequently re-created or re-installed,
and there is no tracking of SSH host keys. USING THIS SCRIPT WITHOUT
UNDERSTANDING AND CONTROLLING ITS SECURITY IMPLICATIONS BREAKS THE SSH MAN-IN-
THE-MIDDLE ATTACK DETECTION SYSTEM!

positional arguments:
  json_config_file_name
                        filename of JSON configuration with host list

optional arguments:
  -h, --help            show this help message and exit
  --loglevel {ERROR,WARNING,INFO,DEBUG}
                        set the log verbosity level, default=INFO
  --timeout TIMEOUT     set the SSH connection timeout in seconds, default=5
```

```
$ ./relearn-known-hosts.py host-list.json
timestamp INFO     Removing known_hosts SSH keys for all hosts in host-list.json ...
timestamp INFO     Removal complete.
timestamp INFO     Re-learning known_hosts entries for all active host IPs ...
```

5.0 Conclusion

The tools described in this document are very specialized and certainly not for general use. It is hoped that their open-source availability can help at least some IT professionals have easier work days :-)