iptables-converter Documentation

Release 0.9.9

Johannes Hubertz

Nov 12, 2017

Contents

CHAPTER 1

what's up here?

iptables-converter is a python script to convert a shellscript file containing iptables commands, usually your firewall startup script, to an iptables-restore format. The latter has exactly one big advantage: speed!

1.1 iptables-converter - intro

1.1.1 Default operating

Assume a plain file with following contents:

```
iptables -F
iptables -t nat -F
iptables -N USER_CHAIN
iptables -A INPUT -p tcp --dport 23 -j ACCEPT
iptables -A USER_CHAIN -p icmp -j DROP
iptables -P INPUT DROP
iptables -t nat -A POSTROUTING -s 10.0.0.0/21 -p tcp --dport   80 -j SNAT --to-source␣
↪192.168.1.15
iptables -t nat -A PREROUTING  -d 192.0.2.5/32 -p tcp --dport 443 -j DNAT --to-
↪destination 10.0.0.5:1500
```

As times goes by, the script will grow. The more lines the longer will it take to be loaded. There should be a quicker way of getting things done. Using iptables-save we easily can save the actual ruleset from the kernel to a file. To load it's content into the kernel again is a very quick action compared to the loading of the originating shellscript. So the idea came up to have a converter just for saving time.

iptables-converter by default reads a file **rules**, using comandline parameter `-s` any other file. After having read completely, output is written to stdout for full flexibility. Given the above file as input the following is printed out:

```
*raw
:OUTPUT ACCEPT [0:0]
:PREROUTING ACCEPT [0:0]
COMMIT
```

```
*nat
:OUTPUT ACCEPT [0:0]
:PREROUTING ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A PREROUTING -d 192.0.2.5/32 -p tcp --dport 443 -j DNAT --to-destination 10.0.0.
↪5:1500
-A POSTROUTING -s 10.0.0.0/21 -p tcp --dport 80 -j SNAT --to-source 192.168.1.15
COMMIT
*mangle
:FORWARD ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
COMMIT
*filter
:FORWARD ACCEPT [0:0]
:INPUT DROP [0:0]
:USER_CHAIN - [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -p tcp --dport 23 -j ACCEPT
-A USER_CHAIN -p icmp -j DROP
COMMIT
```

As a file this might be read by iptables-restore, which works immediately.

Usage example

So you probably may want to run the converter from within a shell script or the like:

```
#!/bin/bash

set -e
INPUT_FILE=rules
OUTPUT_FILE=iptables-converted

# needs to be executable as indicator that writing has ended
[ ! -r $INPUT_FILE ] && exit 0
[ ! -x $INPUT_FILE ] && exit 0

iptables-converter.py -s $INPUT_FILE > $OUTPUT_FILE

# do it only once!
mv $INPUT_FILE $INPUT_FILE}.old

iptables-restore < $OUTPUT_FILE
echo "$INPUT_FILE successfully converted and loaded"
exit 0
# EoF
```

1.1.2 Error handling

Shell functions and shell commands

As the file read is not interpreted in any way, there are few known error conditions:

1. the file contains some shell variables, indicated by '$', this leads to an errormessage and exits immediately with returncode 1.
2. the file contains some shell functions, indicated by '(' and/or ')', this leads to an errormessage and exits immediately with returncode 1.

If you have such a file, and you want to speed up by converting, please execute it and feed the output as a file to iptables-converter.

Non exsitent user chains

iptables-converter does some more error-checking while reading input.

Normal behavior is to raise an errror, if any append or insert statement to an userdefined chain is not preceeded by a corresonding creation statement '-N'. This may be changed to a more smooth handling with an additional commandline option '- - sloppy'. Having this, a non existent userchain is created on the fly when the first append statement is seen. So it is set as first entry gracefully.

Inserting into an emtpy chain anyhow raises an error as iptables-restore would do it later on trying to set the files content into the kernel.

Just to mention it: **iptables -E xyz** and **iptables -L** are not implemented and throw exceptions for now!

1.2 iptables-converter - tests

Untested software, that means software which isn't accompanied by automated functional tests, is assumed to be broken by design. As iptables-converter is written in python, use of the popular unittests is done for your convienience.

Two testclasses are build: Chains_Test and Tables_Test accordingly to the two classes from which the iptables-converter script is build from.

Runing the tests with nosetests:

```
nosetests  ... --with-coverage
.......................
Name                 Stmts   Miss  Cover   Missing
----------------------------------------------------
iptables_converter     189     27    86%   25-26, 167-177, 240-253, 257-258
----------------------------------------------------------------------
Ran 23 tests in 0.019s
```

1.2.1 Chains_Test(unittest.TestCase)

The tests are enumerated to assure a predefined sequence of evaluating for cosmetical reason.

1. A tables group is build first, filter is choosen. The predfined chains are given as parameter to the chains object, then their existance and the default policy is prooved.
2. Setting policy drop into the filter chains is prooved for each chain, an invalid policy keyword is tried and exeption raising is pooved.
3. Append a rule (a valid iptables-statment) into each chain, try to use an invalid filter group and the exception raising for that.
4. Insert rules and then flush them, proof emptiness. Then check exception raising for flushing an invalid filter group

5. Create a userdefind chain and verify existance in the objects dictionay. Check exception raising on creating a predefined chain.
6. Inserting a rule into an empty chain necessarily fails, exception is verified.
7. Inserting a rule into a nonexisting chain fails with exception.
8. Inserting a rule into a nonempty chain works and is verified.
9. Appending three rules to a chain works and their existance in chain object dictionary is prooved.
10. Try to remove a predefined chain raises exception.
11. This test is removed (commented) for reason of practicability. It's intention was, to check if removal of a nonexisting chain raises exception. The code in the chain object is commented as well, as it is needed to achieve a clean status of the chains from any status. So it was not a good idea to raise an exception just for completeness.
12. Creation and successful removal of an userdefined chain.
13. Just look if an illegal command raises an exception.

1.2.2 Tables_Test(unittest.TestCase)

1. Create a Tables object and verify the completeness of all the predefined chains.
2. Verify correctness of a given iptables -t nat command.
3. Verify correctness of a given iptables -t mangle command.
4. Verify correctness of a given iptables -t raw command.
5. Inserting a rule into a nonexisting chain raises exception.
6. Try to read a nonexisting file raises exception.
7. Read empty file and verify result.
8. Read file reference-one and verify result.
9. Read a buggy file with shell variables.
10. Read a buggy file with shell functions.
11. Read a file without '-N' in sloppy mode without raising error.

1.3 iptables-converter - python classes

iptables_converter.py: convert iptables commands within a script into a correspondig iptables-save script

default filename to read is rules, to read some other file, append: -s filename

output is written to stdout for maximum flexibilty

Author: Johannes Hubertz <johannes@hubertz.de> Date: 2017-02-05 version: 0.9.9 License: GNU General Public License version 3 or later

Have Fun!

class iptables_converter.**Chains**(*name*, *tables*, *sloppy=False*)
this is for one type of tables

put_into_fgr(*content*)
fill this line into this tabular

reset()
name is one of filter, nat, raw, mangle, tables is a list of tables in that table-class

class iptables_converter.**ConverterError**(*message*)
on accidential case of error show given reason

class iptables_converter.**Tables**(*fname='reference-one'*, *sloppy=False*)
some chaingroups in tables are predef: filter, nat, mangle, raw

put_into_tables(*line*)
put line into matching Chains-object

read_file(*fname*)
read file into Tables-object

reset(*fname*)
all predefined Chains aka lists are setup as new here

table_printout()
printout nonempty tabulars in fixed sequence

iptables_converter.**main**()
main parses options, filnames and the like one option (-s) may be given: input-filename if none given, it defaults to: rules

class iptables_converter.**Chains**(*name*, *tables*, *sloppy=False*)
this is for one type of tables

put_into_fgr(*content*)
fill this line into this tabular

reset()
name is one of filter, nat, raw, mangle, tables is a list of tables in that table-class

class iptables_converter.**Tables**(*fname='reference-one'*, *sloppy=False*)
some chaingroups in tables are predef: filter, nat, mangle, raw

put_into_tables(*line*)
put line into matching Chains-object

read_file(*fname*)
read file into Tables-object

reset(*fname*)
all predefined Chains aka lists are setup as new here

table_printout()
printout nonempty tabulars in fixed sequence

class iptables_converter_tests.**Chains_Test**(*methodName='runTest'*)
some tests for class Chain

test_01_create_a_chain_object()
Chain 01: create a Filter group, f.e. filter

test_02_prove_policies()
Chain 02: check 3 valid policies, 1 exception

test_03_tables_names()
Chain 03: 3 cases OK, 1 Exception

test_04_flush()
Chain 04: flush filter group, 2 rules and an invalid chain

test_05_new_chain()
Chain 05: create a new chain in filtergroup,

test_06_new_existing_chain_fails()
Chain 06: create an exsiting chain should fail

test_07_insert_rule_fail()
Chain 07: insert a rule into an empty chain fails

test_08_insert_rule_fail()
Chain 08: insert a rule into a non_existing chain fails

test_09_insert_rule_works()
Chain 09: insert a rule into a nonempty chain works at start

test_10_append_rule()
Chain 10: append a rule to a chain

test_11_remove_predef_chain()
Chain 11: try to remove a prefined chain

test_12_remove_chain()
Chain 12: try to remove an existing chain

test_13_illegal_command()
Chain 13: try an ilegal command

class iptables_converter_tests.**Tables_Test**(*methodName='runTest'*)
Tables: some first tests for the class

test_01_create_a_tables_object()
Tables 01: create a Tables object, check chains

test_02_nat_prerouting()
Tables 02: nat PREROUTING entry

test_03_mangle_table()
Tables 03: mangle INPUT entry

test_04_raw_table()
Tables 04: raw OUTPUT entry

test_05_not_existing_chain()
Tables 05: INPUT to not existing chain

test_06_read_not_existing_file()
Tables 06: read non existing file

test_07_read_empty_file()
Tables 07: read empty file (in relation to iptables-commands)

test_08_reference_one()
Tables 08: read default file: reference-one, check chains

test_09_shell_variables()
Tables 09: read buggy file with shell variables

test_10_shell_functions()
Tables 10: read buggy file with shell functions

test_11_reference_sloppy_one()
Tables 11: read sloppy input file: reference-sloppy-one, check chains

Contents:

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index