Overview of fverify:

an iptables policy auditing tool

Source: https://github.com/notem/Linux-Firewall-Verification-Utility

Nate Mathews
Michael Rhodes

Introduction

Network-level packet filters are vital to maintaining a strong security posture in any home or enterprise network. Firewalls are the first and/or last line of defense against network level attacks. Enterprise firewalls may have to handle many gigabits of traffic per second, while efficiently parsing and determining how to handle each packet based on a given policy. Due to the large number of hosts and services in enterprise networks, their firewalls may consist of thousands of rules. For even the most experienced administrator, auditing firewall policies can be a daunting and time consuming tasks. That is why we developed a tool that efficiently and easily parses firewall rules to determine if the firewall satisfies a given policy. This report was written to describe how our tool works and how to continue development for the tool.

The tool

Fverify is a command-line python tool to audit firewall policies in iptables. It was designed to quickly and efficiently audit firewalls with thousands of rules. To run this tool, the iptables rules and a policy must be provided. The usage statement can be seen in *figure 1*. The policy is provided the same way a rule is created in iptables. The only required options for the policy is the target ('-j' or '--jump') and the chain to look at ('-A' or '--append'). Other options, such as the source and destination addresses and ports, or protocols can also be specified. The order of the options entered does not matter, and any extra options added to the policy will be ignored. The iptables rules are provided using the '-file' option. This file must consist of the iptables-save output. An example usage of the tool could be: *fverify -A INPUT -p udp -src 192.168.1.1 -j DROP -file rules.txt*

```
Usage: fverify [policy] -file [filename]
        Specifying a policy:
                use the same options you would use to create a rule in iptables
        Required parameters:
                -j | --jump [target]    specify the target (ADD/DROP)
                -A | --append [chain]   specify the chain to look at
                -file [filename]        file containing iptables-save output
        Optional parameters:
                source/destination ports (e.g. --dports, --source-port)
                source/destination addresses (e.g. -s, -d, --src-range)
                protocol(s) (e.g. -p, --protocols)
```

Figure 1. Usage statement

Algorithm

This tool implements the algorithm for formal verification of firewall tables, as described in [1]. The algorithm is implemented in the *C* language and used in a Python 3 module to allow python scripts to efficiently execute the firewall property verification algorithm. The efficient performance of the algorithm is gained through the usage *projection* and *slicing*. The algorithm implementation supports firewalls composed of five-tuple rules. Internally, the firewall is represented using three contiguous arrays of 32-bit integers. As a consequence of this, the algorithm is rather cumbersome to use with in *C*. The python module wrapper simplifies the process of building a firewall and verifying properties. The python module exposes several functions:

- ***add(...)*** may be used to append firewall rules (represented as tuples) to firewall.
- ***verify(...)*** may be used to verify a property against the current firewall.
- ***witness()*** can be used to retrieve the stored witness value produced as a byproduct of the previous verify procedure.
- ***clear()*** can be used to reset the firewall.
- ***size()*** may be used to retrieve the number of rules the firewall contains.

Program files

Our tool consists of four python scripts:

- ***fverify.py*** - primary script capable of parsing an *iptables* export file and evaluate property specified by the user.
- ***setup.py*** - used to compile and install the *firewall_verifier* python module.
- ***benchmark.py*** - used to test performance of the algorithm.
- ***test.py*** - demonstrates the correctness of the firewall verification algorithm on a small firewall (note: this script does not *prove* the correctness of the implementation).

The source code for the python module is divided into three files:

- ***algorithm.c*** - full implementation of the firewall verification algorithm 5-field firewalls.
- ***algorithm.h*** - header file which describes the correct usage of the functions provided.
- ***python.c*** - provides an interface to use the algorithm in python scripts.

Benchmarking

A small script for benchmarking the application is provided. This benchmarking script will generate a firewall of a configurable size and compare the firewall to a collection of randomly generated properties. The script outputs the time-to-completion of each of these processes. Below is a sample of the results produced by this benchmarking script on a *Intel(R) Celeron(R) CPU 3215U @ 1.70GHz* processor.

Trial	Number of Rules	Properties Tested	Max Time	Average Time
1	10,000	100	288 ms	28.1 ms
2	10,000	100	540 ms	143 ms
3	10,000	100	364 ms	66.0 ms
4	100,000	100	35.5 s	6.87 s
5	100,000	100	55.3 s	16.9 s
6	100,000	100	47.5 s	12.8 s

Limitations

- This tool currently only looks at the options specified in the required and optional parameters of the usage statement. For instance, it ignores options from most extension modules, such as matching TCP connections and filtering by MAC address. More options can be added by adding more tuples to the algorithm.
- This tool currently doesn't work with rules that don't jump to a target. For example, it doesn't work with rules that use -g or --goto instead of -j or --jump.

As a consequence of these limitations, we do not recommend that this tool be used in non-academic settings. The application would need to be further developed to support more fields used by iptables.

Future work

- The algorithm implementation could be modified to utilize concurrency when available.
    - The primary bottleneck of the verification algorithm is when candidate witness packets are produced and tested against a firewall slice.
    - Another strategy would be to parallelize the slice creation and processing component. This method may be substantially easier to implement.
- Add more tuples to the algorithm to provide a better representation of actual iptables policies. For example, tuples could be created for source and destination MAC addresses or to track TCP connection states. An ideal implementation would allow the library user to specify the number of fields to support when initializing the verifier module.
- The ability to parse configurations of other firewall applications and frontends. The application currently only supports exports from the *iptables* frontend.
- To make this tool easier to use and more widely available. The code could be better packaged for distribution. It could be added to an apt repository or integrated with pip.

- The tool could be modified to be more modular. This would allow later developers to more easily add support for additional *iptables* filtering fields.

References

[1] H. Bhattacharya, "On the Modular Verification and Design of Firewalls", Ph.D, The University of Texas at Austin, 2012.