Fuzzy Firewalls

Hime Aguiar e Oliveira Junior, Maria Augusta Soares Machado
IBMEC - Av. Rio Branco no. 108 - 9o andar – Centro –
Rio de Janeiro – RJ – BRASIL CEP 20040
hime@engineer.com, mmachado@ibmecrj.br

Abstract

This paper presents a fuzzy control approach for improving present firewall architectures. Taking into account not only network packets contents and connection status information to take filtering decisions, it's possible to make firewall behavior adaptive and to use it to increase operating systems' immunity to attacks .
Keywords: Fuzzy Logic, Network Security, Fuzzy Control, Firewalls.

1. Introduction

Network and computer security is a main concern in present days, and the most common means of trying to get it is through the use of the so-called firewalls [18]. A typical firewall has basically two types of mechanisms to achieve its aim :

- Static packet filtering [13].
- Tracking of connection states based on protocol specifications

In addition, "crisp" (the term "crisp" is used as the oposite of "fuzzy" [17]) rules are defined and used to "tell" the firewall what is allowed to pass or not through the network interface. This way, we can block all incoming echo-request ICMP packets of a given set of hosts and admit GET SNMP packets of another one. Note that, in this scheme, the rules force the firewall to behave in a static (all or nothing) manner, not leaving room for adaptive dynamics, based on parameters collected from the network traffic. A very known kind of network threat is the "denial of service" attack, that consists of bombarding network servers with enormous amounts of well-formed requests, driving them to their knees. That could occur using several protocols like ICMP (echo-request), UDP (SNMP GET request), TCP (SMTP transactions), etc.. The key idea is that there is a certain threshold after which the computational resources will be exhausted and weird things could happen because, unfortunately, most operating systems have bugs and are not prepared to degrade gracefully. Even in a bug-free operating system, response time would be so large that such a situation is unacceptable. On the other side, connection state tracking is effective in several situations, but a skilled hacker could very well to trigger a great number of "well-behaved" sessions and be allowed to cause serious damages.

Another issue to be addressed is that the quality of security in a given site is dependent of network administration staff's skills relative to internals protocols: in other words, if people in charge of setting up the firewalls forget something, we can have bad surprises. But, what to do to ensure maximum usability, keeping the site at good levels of security ?

The answer seems to be in the use of Soft Computing techniques (in particular, Fuzzy Logic [1,4]) a particular caso being qualitative modeling [2]) to build a model of an "intelligent" and adaptive firewall that changes its behavior depending on the dynamical conditions of networks and computational resources.

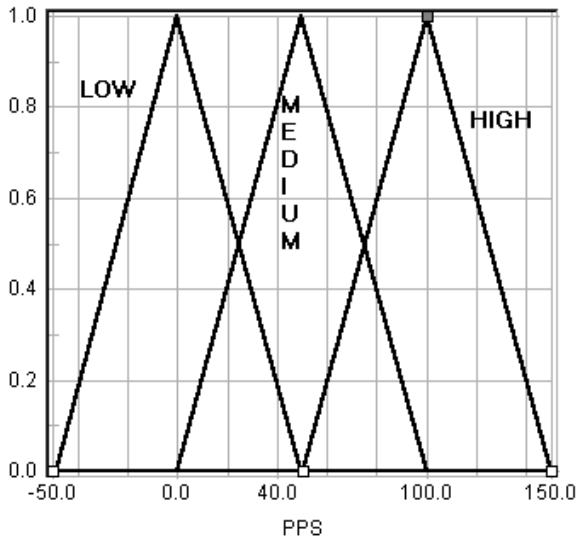

Fig.1 Functions for MEDIUM, LOW BUT NOT SO MUCH and HIGH BUT NOT SO MUCH

For the sake of simplicity, we'll focus the discussion on the protection against denial of service attacks, examining two simple and well-known situations : ping flooding and SMTP/POP server resource starvation. Our purpose is to show how fuzzy logic control can be applied to solve the problems above, in particular, those related to resource starvation..

2. Echo-request flooding

To cope with this, we propose to build tables, containing statistical, real-time information about ICMP-related traffic.In addition, the user could be allowed to set up the maximum level of ICMP echo-request packet acceptance (in packets per second, for instance).

After that, we are ready to build the following fuzzy rule-base and terms:

- IF <ICMP echo-request rate> IS LOW THEN <ICMP echo-request acceptance> IS HIGH.
- IF <ICMP echo-request rate> IS MEDIUM THEN <ICMP echo-request acceptance> IS MEDIUM.
- IF <ICMP echo-request rate> IS HIGH THEN <ICMP echo-request acceptance> IS LOW.

- - - - - - - - - - - - - - - - - - - -

etc.

- -

where the fuzzy input terms LOW, MEDIUM and HIGH are given by and the output fuzzy terms LOW, MEDIUM and HIGH are represented in the Figure 1.

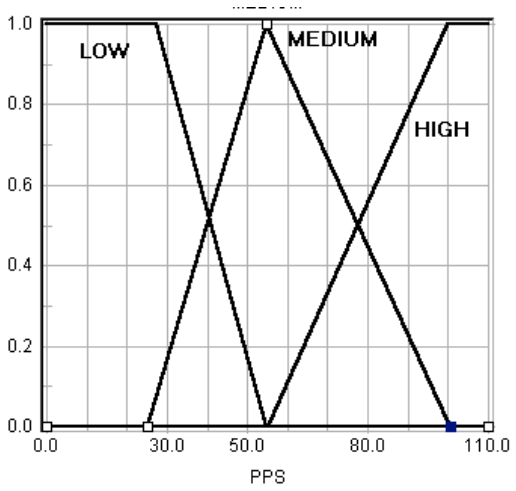

Figure 2 Functions representig the fuzzy concepts LOW, MEDIUM and HIGH.

With the usual t-norm, t-conorm, fuzzy inference and COG (Center Of Gravity) defuzzification, we have a (very simplified) Mamdani fuzzy system [3,8,10] that implements the desired control scheme.

Obviously we could have chosen a TSK (Takagi-Sugeno-Kang) system [19], replacing the consequent parts by adequate expressions and simplifying even more (in my viewpoint) the actual implementation.

To verify that the fuzzy controller really does its job, we'll make a little calculation showing its output for some representative values of the input.

Suppose that the user adjusted the maximum allowed echo-request input rate at 100 pps (packets per second). If, at a given sampling interval, the actual rate is lesser than 25 pps, the acceptance rate will be 100 %, resulting in the standard behavior (all packets will pass). If, during another period, the incoming rate is 50 pps, the acceptance rate will be 50 % (one packet pass and the following is rejected, for example). When the incoming rate is 100 pps or higher, the acceptance rate will be 0 %, meaning full blocking of incoming ICMP echo-request packets.

Such a control law seems sensible, taking into account that – according to the user's perception – a denial of service attack is in progress. Note that immediately after the end of attack the standard behavior is restored and "authentic" traffic can flow again, without human intervention.

3. Resource starvation in SMTP/POP servers

The strategy, in this case, is to allow incoming requests in such a way that the given server can stay responsive and "healthy". It is of no value to establish more sessions than the existent resources could afford. As in the previous case, we can get an upper limit on the simultaneous number of e-mail transactions from the user or to estimate it from operating system and firewall parameters.

A (very simple) fuzzy rule-base in this case could contain the following rules:

- IF <number of present SMTP/POP sessions> IS LOW AND <data rate of SMTP/POP traffic> IS LOW THEN ACCEPTANCE IS HIGH.
- IF <number of present SMTP/POP sessions> IS LOW AND <data rate of SMTP/POP traffic> IS HIGH THEN ACCEPTANCE IS MEDIUM.
- IF <number of present SMTP/POP sessions> IS MEDIUM AND <data rate of SMTP/POP traffic> IS HIGH THEN ACCEPTANCE IS LOW.
- IF <number of present SMTP/POP sessions> IS HIGH AND <data rate of SMTP/POP traffic> IS HIGH THEN ACCEPTANCE IS VERY LOW.

.....

etc.

where linguistic variables and fuzzy terms will be defined according the users' needs.

Obviously, in a real life situation the fuzzy system will not be so simple and could use different elements. As complexity rises, we believe that neuro-fuzzy modeling is the right way to go.

But anyone could argue : it suffices to build a crisp decision table to get to the same results !

We don't think so and it's easy to see that viewing the firewall as a discrete-time control system and the host as the plant under control, we have all the right tools to tailor the dynamic characteristics of the closed-loop system, including stability-related ones.

4. Self-organizing maps and load monitoring

To help us in the difficult task of critical condition identification of network resources utilization, we have employed Kohonen's self-organizing maps as a pattern recognition/decision making tool. This was done because, in certain conditions, isolated parameters cannot tell by themselves if the overall system is under true stress or not. So, the fuzzy control approach is supported by sound global information about network activity. To clarify this point, suppose that, in the example about ping flooding, we'd like to improve our fuzzy rules, substituting them by

- IF <ICMP echo-request rate> is LOW AND <global system stress> IS LOW THEN <ICMP echo-request acceptance> IS HIGH.
- - - - - - -
- - - - - - - -

IF <global system stress> IS HIGH THEN <ICMP echo-request acceptance> IS MEDIUM.

- - - - - - -
- - - - - - - -
- - - - - - - -

where the new input linguistic variable <global system stress> is obtained from a SOM-based decision maker. It's easy to see that such a fuzzy rule-base allows us take more realistic decisions, taking into account the overall situation.

It's crucial to understand that denial of service attacks are successful if and only if the target system has its resources exhausted; if we can predict dangerous patterns before the worst occurs, it's possible to take evasive actions – exactly what the above fuzzy rules intend to do. In the sequel, we'll explain how the pattern recognizer was implemented.

5. Diagnosing load conditions through SOMs

The problem was handled by defining characteristic features of typical networked environments, like UDP,TCP,ICMP incoming packet rates (among others) and sampling them for significant extents of time. From this, a large and representative sample of approximately 150,000 9-dimensional tuples was generated and used as an input to one of Kohonen's algorithms, giving as a result a grid of codebook vectors (4,800 cluster centers). Each one of these vectors represents a given load state, to which we can attach a load or "global system stress" degree, according to our perception or to another arbitrary criterion. As the clustering takes place in a 9-dimensional space and 4,800 centers were synthesized , we have used an automatic scoring scheme.

After convergence, the (cluster center, score) tuples were stored in tables, to be used in run-time by the fuzzy inference engine. The overall process is very simple: during its operation, the module gets present rates of the diverse pre-established features and find the nearest neighbor among the previously found cluster centers. After that, the corresponding score is the key to compute the membership values relative to the input fuzzy term. The real-time module may be implemented in several ways: we believe that the better way is to use internal firewall statistics. Another choice would be to launch a detached process which, by means of a packet capture package, would make the recognition task. Anyway, the run-time overhead is very low, according to our experience.

6. A fuzzified firewall : the Linux / Netfilter / Iptables case

Linux [18] has had firewalling capabilities for a very long time. in the 2.0 kernels, we had the ipfwadm package, which did basic fire walling. With 2.2, ipchains was implemented and offered a significant number of improvements over 2.0. With the 2.4 kernel, all of the old fire walling code was ripped out and a new system implemented. Rather than simply provide fire walling capabilities, the development team decided to simply write a system which can take plug-ins, allowing any fire walling system to be used with Linux. This subsystem is known as *netfilter* [11,12], and allows *ipfwadm* and *ipchains* to be used with the 2.4 kernel series using the ports of each of these to *netfilter (that is about to change in new releases of 2.6 kernel).* However, a far more powerful fire walling system was

implemented, known as *iptables* [5,6]. Netfilter is the system compiled into the kernel which provides hooks into the IP stack which loadable modules (iptables is one) can use to perform operations on packets. As netfilter uses modules for the filtering, you can use an ipchains module to provide exactly the same capabilities as the kernel level ipchains code in 2.2, or even the module for ipfwadm from 2.0. Netfilter is there all of the time, as long as it is compiled in, whether or not you are using any fire walling modules at all. IPTables is split into two parts; The user-space tools and the kernel-space modules. The kernel-space modules are distributed with the main kernel, and you compile them as you would any other module, be it sound drivers, a file system or USB support. There is the main ip_tables module, as well as modules specifically for NAT, logging, connection tracking and so on. These modules perform the appropriate function on the packets which they get sent by netfilter, depending on the rules which they have in their rule-list, or chain. The user-space iptables code comes in the form of a binary called 'iptables', which is distributed separately from the main kernel tree, and is used to add, remove or edit rules for the modules.

Iptables has three built-in lists of rules – or chains – for filtering., so there is INPUT, OUTPUT and FORWARD chains. INPUT applies to all packets destined for the local machine, OUTPUT for packets which originated locally and FORWARD for packets which are sent to our machine, but are not actually for it. We can, if we want, create our own chains to organize our rules into different groups based on other rules. We create a chain with **iptables -N <chain-name>** and delete it with **iptables –X <chain-name>**. After this, they behave just like the three default chains, and we can flush them with **iptables -F <chain-name>** or list their rules with **iptables -nL <chain-name>**. Using iptables we can perform three actions on the chains which alter their rules. We can either add, insert or delete rules, using **-A**, **-I** and **-D**, respectively, followed by the chain name. So, if we wanted to add another rule to the end of the INPUT chain we would use **iptables -A INPUT**. Not much used, as we need to specify which packets we want the rule to apply to.

Matching source and destination IPs and ports is the most straightforward things to do. If, for example, we want to block all connections to port 80, over tcp, to a local machine we would do:

iptables -A INPUT -p tcp --dport 80 -j DROP

-p sets the IP protocol used, be it TCP, ICMP, UDP or one of the other more unusual protocols, and **--dport** specifies the destination port of the packet. We can, of course, use **--sport** to specify a source port, but that is rarely used as connections normally use a high source port, unless they are from a specific service, which has packets coming from a specific port.

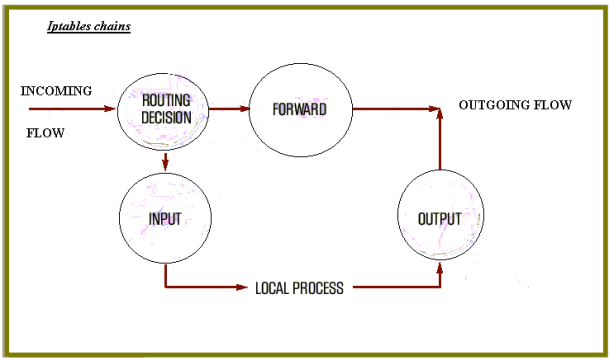

Figure 3 Schma of the information flow

As with every firewall we have notice, the rules are Aristotelian (they don't allow a rule to be partially satisfied), that is, one of two conditions is met : a given packet flow is totally blocked or totally freed to pass through the filters. But, in many cases, DoS (Denial of Service) attacks consist of using perfectly legal services with enormous packet rates, exhausting server and communication resources and, frequently, causing interrupts in normal service.

The question is: **How to maintain services publicly available and to block DoS attacks at the same time?**

The answer seems to be related to the degree at which the traffic hits the server infrastructure – if packet rate is low or moderate, then everything is ok. If traffic is excessive, we are under DoS attack. So, it is necessary to design mechanisms that can "understand" and take evasive actions in adverse conditions and remove barriers when conditions are favorable.

To this end, the Netfilter/Iptables system was endowed with a extension (the fuzzy match) that allowed the user to translate his own degree perception into rules, so that the problems above could be solved in a simple manner. In truth, the original Aristotelian firewall was fuzzified by the ***fuzzy*** kernel module.

The fuzzy match allows you to match packets according to a dynamic profile implemented by means of a simple Fuzzy Logic Controller (FLC). It implements a TSK FLC (Takagi-Sugeno-Kang Fuzzy Logic Controller) and the basic idea is that the match is given two parameters that tell it the desired filtering interval.

- When the packet rate is below `lower-limit', the rule will never match.
- Between `lower-limit' and `upper-limit', matching will occur according an increasing (mean) rate.
- Finally, when the packet rate comes to `upper-limit', (mean) matching rate attains its maximum value, 100%.

Taking into account a variable sampling rate of approximately 100ms (on a busy machine), the module

presents good adaptation to a fast to changing traffic patterns.

For example, if you wish to avoid generic Denial Of Service attacks, you could use the following rule:

iptables -A INPUT -m fuzzy --lower-limit 100 --upper-limit 1000 -j REJECT

- Below the 100 pps (packets per second) rate, the filter is inactive.
- Between 100 and 1000 pps the mean acceptance rate drops from 100% (when we are at 100 pps) to 0% (when we are at 1000 pps).
- Above 1000 pps the acceptance rate keeps constant at 0%.

Supported options for the fuzzy patch are :

--upper-limit n
-> Desired upper bound for traffic rate matching.

--lower-limit n
-> Lower bound over which the FLC starts to match.

For additional details, please see [11].

Conclusions

In our viewpoint, present firewall technology will be greatly improved by the application of fuzzy logic control techniques.

To cope with complexity, neuro-fuzzy techniques can be employed, allowing us to build large rule-bases without expert knowledge.

So, enterprises could benefit largely from the use of Soft Computing techniques applied to security devices like the one described above that, at the best of our knowledge, was the very first operational fuzzy firewall in existence.

References

[1].BARRETO, J. M .:*Inteligência Artificial no Limiar do Século XXI*. RôRôRô Edições, Florianópolis, 2001.

[2].BARRETO, J M., and M. De NEYER, Qualitative and Quantitative Models of Systems. In *Mathematical Modelling in Science and Technology*, R. Hanus, P. Kool, S. Tzafestas (edts.), Jorge C. Baltzer AG, Scientific Pub. Co." p. 269-274, 1991.

[3].BARRETO, J. M., De NEYER, M. & GOREZ, R., Fuzzy control of a non linear plant: the case of a fluid mixer, In *Proc. IEEE/MELECON Mediterranean Conference*, B. Zajc and F. Solina (edts.), Ljubljana, Slovenia, pp. 807-811, 1991.

[4].BRAGA, M. J. F., BARRETO, J. M. & MACHADO M. A.: *Conjuntos Nebulosos em Análise de Risco*, Artes e Rabiscus, Rio de Janeiro, 1995.

[5].COULSON, D.: Network Security – Iptables, 2003 www.davidcoulson.net/writing/lxf/38/iptables.pdf.

[6].COULSON, D.*: Network Security – Iptables*, Part 2, 2003 www.davidcoulson.net/writing/lxf/39/iptables.pdf

[7].De NEYER M., R. GOREZ & J. M. BARRETO: Disturbance Rejection Based on Fuzzy Models. In *Decision Support Systems and Qualitative Reasoning* M. G. Singh e L. Travé-Messuyès (edts.), North-Holand, Amsterdam, 215-220, 13-15/03/1991.

[8].DRIANKOV, D. & PALM R. (Editors). *Advances in Fuzzy Control*. Physica Verlag, 1998.

[9].KAZABOV, N. K.L *Foundations of neural networks, fuzzy systems, and knowlegde engineering*, The MIT Press, Massachussets, 1996.

[10].MAMDANI, E. H., *Application of fuzzy algorithms for control of simple dynamic plant*, Proc. IEE, vol.121, nº12, pp.1585-1588, 1974.

[11].Netfilter Extensions HOWTO – (www.netfilter.org/documentation/HOWTO//netfiltr-extensions-HOWTO.html)

[12].Netfilter Hacking HOWTO – (www.netfilter.org/documentation/HOWTO//netfiltr-hacking-HOWTO.html)

[13].Packet filtering HOWTO – (www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.html)

[14].Networking concepts HOWTO – (www.netfilter.org/documentation/HOWTO//networking-concepts-HOWTO.html)

[15].TANSCHEIT, R. *Controle de Processo por Lógica Nebulosa*. Mestrado em Engenharia Elétrica - Instituto Militar de Engenharia., 1978.

[16].TANSCHEIT, R; BARRETO, J. M.. Controle de Misturador de Fluídos na Matemática Nebulosa. In: *II Congresso Brasileiro de Automática*, Florianópolis. 1978.

[17].ZADEH, L .A., Fuzzy Sets, *Information and Control*, vol.8, nº1, pp.338-353, Jan 1965.

[18].ZIEGLER, R.:. *Linux Firewalls*. New Riders, 2002.

[19].ZHAO, J. *System Modeling, Identificationand and Control Using Fuzzy Logic*. Doutorado em Ciências Aplicadas – Université Catholique de Louvain, 1995.