From Network Security To Content Filtering

Network security has evolved dramatically in the last few years not only for what concerns the tools at our disposals or the threats to which we are subjected, but in the approach itself that must be adopted to secure a network connection.

Around year 2000 a robust stateful firewall working at layer 4 (in ISO/OSI parlance) was considered a good defense in most situations. Indeed by selecting the TCP/IP addresses, ports (i.e. services) and the direction of the connections (that is of the first packet requesting the service) it was possible to prevent most attacks and reduce quite considerably the risks of unwanted network connections. To be effective this approach required that:

1. TCP/IP ports identified clearly the protocols and the kind of data which is transported through them;
2. it was possible to trust IP addresses to identify the source/destination of the network traffic to distinguish the trusted peers from the untrusted or the unknown.

Stateful firewall are still today, and will be in the future, a fundamental building block of network security, very little can be done without them. But today their efficacy is very limited and this has come about in part thanks to their success.

Indeed the diffusion of stateful firewalls in public and private companies in the first years of 2000 has forced application developers to find way to pass through them. Very often it happened to install in a company a new application and to discover that it required a modification of the ruleset of the firewall to be able to send or receive communications through it. But the modification of the ruleset of the firewall often requires management approval, security checks of the application itself etcetera. From the application point of view, in many situations it is easier if there is not the need to modify the security rules or to add new security rules, but it is sufficient just to apply security rules already in place.

Since access to web browsing in internet is usually allowed through firewalls, almost all applications today offer the possibility to tunnel their communication as if it was web traffic: some applications just use port 80, the port reserved web servers (*httpd*), other really tunnel their data communication in the *http* protocol and appear as if it was a browser connecting to a web server.

If most network traffic uses the web ports and looks like web traffic, a layer 4 stateful firewall which filters traffic based only on TCP/IP addresses and ports has very little possibility to enforce the company security policies. Moreover, if a few years ago authentication by IP addresses could have been acceptable for very low security services, today is totally unacceptable: with the extensive deployment of Network Access Translation (NAT), Virtual Private Networks (VPN) and tunnels, the presence of anonymizing services and unfortunately also of viruses, worms, zombie PCs etc. it is impossible to trust the origin of a connection only based on its source IP address.

At the same time the security of applications has improved greatly: default password have almost disappeared, trivial bugs due to bad coding practices have mostly been corrected etc., so being able to contact an application via network is today not sufficient to have access to it. Today attacks must be done at the application level: a layer 4 stateful firewall does not analyze the contents of the packets so has no way of protecting from application level attacks.

The problem of filtering traffic at layer 7

So the fact that layer 4 stateful firewall are very good in doing their job has, together with the evolution of network security, forced an epic change in the way data is transmitted in TCP/IP networks. We cannot rely on the ports used by an application to understand which kind of data is transmitted since typically port 80 is used for almost anything. So we have to look at the actual contents of each TCP/IP packet (this is usually called *layer 7 filtering*), but how can we do this? There are various issues at hand among which the most important are:

1. <u>Speed</u>: to check the port number of each packet is fast, to analyze all data transported by a packet is a very big job which can impact adversely on the speed of communication, the work load on firewalls and security devices etc.

2. Protocol Analysis: it is possible to limit the analysis to verify that the data transported is formatted according to the application protocol that is allowed, typically *http*: this will prevent applications just to use port 80 to communicate (and, as we have seen, this can be a problem) but will not protect against attacks to applications which use the *http* protocol to communicate.

3. Content Filtering: *full* content analysis seems to be the real security solution, but it is almost impossible to implement. Indeed if almost any kind of traffic can be tunneled through *http*, how can we discriminate 'good' from 'bad' content? As an extreme example consider Skype traffic which is encrypted end-to-end and can be tunneled as *https* (port 443) traffic: for example it is not an easy task to forbid only Skype traffic and allow all other kinds of https traffic. So instead of *full* content analysis what it is possible to do is *Content Filtering* that is to filter out traffic which is either known to be bad or considered to be suspicious or not standard. This still requires to analyze the full contents of each packet but looking only for known attacks or new patterns (as we will describe better below).

Proxy and Application Level Gateway (ALG)

Layer 7 filtering has traditionally been done by Proxies or Application Level Gateway. In a client-server TCP/IP connection a Proxy intercepts the packets and behaves towards the client as if it was the server and towards the server as if it was the client. It thus breaks the connection in two (this could be seen as a kind of legal Man-In-The-Middle attack) and has the possibility of checking the content of each packet as it passes by. Actually very few Proxies check the contents of each packet for possible attacks.

Indeed the simplest version of Proxies, also called *Circuit Level* Proxies, do not look at all at the content of the packets but split the connection in two and in case are able to add an authentication layer if they sit in front of a sever which does not authenticate the clients. More common Proxies, like the ones usually employed for web traffic, can do client authentication, cache common web pages and do some protocol verification. They can also use white or black lists to limit the clients or servers which can connect to them. A few Proxies are more security aware and are able to recognize also attacks and prevent them.

But the most important feature of traditional Proxies is that for each application there must be a specialized Proxy since it must implement the full application logic to be able to understand and protect the traffic. Often each Proxy requires its own server and special network configuration to be able to intercept the traffic. This does not make it simpler for the network and security managers and on top does not cover all types of traffic but only the ones for which a Proxy is available.

If we look at Proxies from the point of view of full network security, we realize that they are fundamental components but still do not give all the answers to our needs of security. Today every network security system must have at least a Web and a Email (anti-virus and anti-spam) Proxy, but we need something else to cover the general problem of Content Filtering.

From IDS to IPS

For generic network Content Filtering we turn our attention to Intrusion Detection Systems (IDS). An IDS works in a similar way to an email anti-virus (there are IDS based on recognition of unusual traffic but we do not consider them here): given a database of rules to detect malicious packets, an IDS checks all packets which are sent to it for their presence. In case an IDS finds a malicious packet, it sends an *Alert* to the *Console*, in other words it sends an alarm message to the person in charge. Once the alarm has arrived, it must be evaluated, managed and finally an appropriate response has to be initiated. Often all these activities are manual, some time part of them are automatic.

An IDS can analyze the full packet, but usually not the original one. Indeed the work of an IDS is very onerous and to avoid delays in the network traffic, an IDS receives copies of the packets in transit and analyzes the copies without worry to delay the original packets. In this way, the real packet, even if malicious, reaches anyway its destination and often the alarm arrives some time after it. In case of very heavy traffic IDS are known to *drop* packets, that is not to analyze some packets if they start to lag too much behind the real network traffic.

Another aspects of IDS is the kind of rules used to match malicious packets. Since IDS are not used to stop attacks in real time, they are used to gather information about possible attacks. In other words, since they do not interfere with the true traffic, it is considered better if they report false positives, that is alarms which actually do not correspond to real attacks, than they have false negatives, that is that they miss some attacks. Indeed it is often difficult to create rules to match malicious packets that in some situations do not match also normal packets, and there is a delicate threshold between the number of attacks undetected and the number of false attacks reported: as one decreases the other increases.

This is the reason why all alarms generated by an IDS are reported to a console where they have to be studied, correlated and finally it has to be decided if they correspond to a real attack or a false positive. IDS are in any case very useful, because they give the possibility to correlate traffic and to discover patterns, like for example of reconnaissance traffic, which can lead to an attack. But IDS are difficult to use and they often require the intervention of highly specialized personnel.

So IDS are not the final solution to the security of today network traffic. On the other side, IDS are able to analyze all kind of traffic at the application level and implement part of what Content Filtering should be.

But what if we give to an IDS the possibility to stop malicious traffic? This will be then a *reactive-IDS* which is often called an *Intrusion Prevention System* (IPS) [notice that there exist many different, albeit similar, definitions of the term IPS].

The first idea would be that the IDS sends the alarms directly to a firewall which then adds a rule to stop the malicious packets. But doing in this way the first malicious packet will for sure arrive to destination. Moreover the number of rules added to the firewall can become very large and reduce the efficiency of the firewall itself besides the risks of a self-induced denial of service attack.

The only practical way to go, is to merge firewall and IPS as follows. We can put on the same security device first a layer 4 stateful firewall which imposes the security network policies: only packets allowed by the security network policies pass the stateful firewall and reach the IPS which follows. Then the IPS scans the real packets, and not copies, and if it finds that a packet matches one or more rules, it drops, i.e. cancels, the packets (optionally it can also send a reset to the sender/receiver to stop the connection). In this way we limit the work of the IPS only to packets already filtered by the layer 4 stateful firewall but we guarantee that if a packet matches a rule, it is dropped and it does not reach its destination.

Limits of an IPS

An IPS as described has opposite problems to an IDS. Indeed for an IPS the problem of false positives is very important: there must not be false positives otherwise lawful connections will be dropped. So we must select only those rules which guarantee to match only malicious packets. Another problem for IPS is the one of delays: since an IPS works on real packets, it cannot delay them nor drop them due to heavy load. So the number of checks that an IPS can do on a packet is limited to very few. To speed up IPS various techniques are implemented: from hardware dedicate devices, to filtering only some kind of traffic as for example not filtering email traffic since anyway it has to go through the anti-virus, and, as already mentioned, reducing the number of used rules.

If we compare an IPS to and IDS, we see that the two practically work in opposite regimes:

- an IDS should not have false negatives (i.e. attacks undetected) but can have many false positives; it usually uses as many rules as possible, works off-line on copies of the packets, can drop packets under heavy load and sends alerts after the real malicious packets are already delivered;
- an IPS should not have false positives but can have false negatives; it uses a limited number of rules, works in-line on the real packets, cannot drop packets under heavy load but directly drops the malicious packets if they match a rule.

So how are we standing with network security today?

From this discussion it should be obvious that we do not have the perfect solution, the tool or approach to network security which allows us to control all traffic. Content filtering is difficult and we must adopt different and parallel approaches to be able to reach a reasonable confidence that the data which transit in our networks is not malicious. Today we definitely need to use all tools at our disposal, from the old layer 4 stateful firewall, the basis of all network security, to the specialized Proxies, the IDS and the IPS. Managing all these tools is not always an easy task, even if the latest security devices do include most of them and allow to manage them in a coherent and unified way.

Snort-inline

Since the IPS is the latest kid on the block, we conclude by briefly indicating how one can build a very simple open-source IPS to verify its functioning. Snort [1,2] is probably the most famous IDS, besides being open-source. It has also a version, called Snort-inline [3] which is a reactive IDS, that is, according to our definition, an IPS. Installing Snort-inline on a Linux machine is an easy and well documented process. Here we are interested in discussing a few important points of the configuration.

First of all, the layer 4 Linux stateful firewall, called iptables, sends to Snort-inline only packets for the port 80, that is typically web traffic. As indicated in Table 1, this is done by sending to the QUEUE target the packets directed to port 80 instead of forwarding them directly, which would be done with the target ACCEPT.

The configuration of Snort-inline in Table 2 contains only rules and commands which can be related to web traffic. Comparing this configuration with that for a typical Snort IDS (the configuration file is the same for Snort and Snort-inline) one can see that the one for Snort-inline is reduced to the only necessary items.

The only differences in the configuration between Snort and Snort-inline are in the rules themselves. The typical Snort action in a rule, see the first line in Table 3, is *alert,* that is to send an alert, whereas Snort-inline adds other possible actions to execute if a packet matches a rule. Among the most common actions, see Table 3, there are: *drop* which drops the packet and sends an alert, *sdrop* which silently drops the packet, *reject* which drops the packet and sends a reset to the sender of the packet.

From this simple configuration it should be obvious that the main differences between Snort and Snort-inline are that:

- Snort-inline receives the real packets from the firewall, whereas Snort works on copies of the packets obtained as a sniffer with *libpcap*
- Snort-inline either allows the packet to continue to transit or drops the packet, whereas Snort just sends an alert
- in the configuration of Snort-inline one should limit oneself to the real necessary commands and rules.

Andrea Pasquinucci

PhD CISA CISSP

http://www.ucci.it/

References

[1] www.snort.org

[2] www.sourcefire.com

[3] snort-inline.sourceforge.net

```
# activate ip_queue
modprobe ip_queue
# add rule to send packets to snort_inline
iptables -A FORWARD -i eth0 -p tcp --dport 80 -j QUEUE
```

```
# send to snort_inline also the return packets
iptables -A FORWARD -o eth0 -p tcp --sport 80 -j QUEUE
```

Table 1. Iptables rules to send packets to Snort-inline in Linux

```
### Network variables
var HOME_NET 1.2.3.0/24
# Ports you run web servers on
var HTTP_PORTS 80
# Path to your rules files (this can be a relative path)
var RULE_PATH /etc/snort_inline/rules

### preprocessors
preprocessor flow: stats_interval 0 hash 2
preprocessor stream4: disable_evasion_alerts, stream4inline, \
                      enforce_state, memcap 134217728, timeout 3600, \
                      truncate, window_size 4000
preprocessor stream4_reassemble: both
preprocessor http_inspect: global \
                      iis_unicode_map unicode.map 1252
preprocessor http_inspect_server: server default \
                      profile all ports { 80 } oversize_dir_length 500

### Logging alerts
output alert_fast: snort_inline-fast
# Include classification & priority settings
include $RULE_PATH/classification.config
include $RULE_PATH/reference.config

### The Drop Rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-php.rules
include $RULE_PATH/sql.rules
```

Table 2. An over-simplified configuration of Snort-inline

```
alert udp any any <> any 53 (msg: "Alert DNS";)
drop tcp any any -> $HOME_NET 80 (msg:"drop TCP port 80";content:”...”)
sdrop tcp any any -> $HOME_NET 80 (msg:"sdrop TCP port 80";content:”...”)
reject tcp any any -> $HOME_NET 80 (msg:"reject TCP port 80";content:”...”)
```

Table 3. Example of rules for Snort-inline (the *content* part of the rule is not displayed)