Some thoughts about Symbiosis Firewall Management

Originally posted to https://github.com/BytemarkHosting/symbiosis/issues/136

The Symbiosis firewall seems to work well, but I have issues with it, which may be caused by my lack of understanding of the nuances of ruby, I guess. Mostly I want to stop the bots that relentlessly look for logins and passwords, while at present I think that Exim is doing a reasonable job of keeping spam out.

This attempts to explain my understanding of what is happening, so please bear with me - and maybe you will see if I have it wrong.

What's there

I am interested in improving the action of /usr/sbin/symbiosis-firewall-blacklist which runs every 15 minutes (or so) and is responsible for creating and deleting files in /etc/symbiosis/firewall/blacklist.d. The files are named for IP addresses (and have .auto) appended to them when created by the script. IP files can be created here to put into the firewall, and are ignored by the system. The files contain port numbers, 'all' or be empty (which implies 'all') - the port numbers are loaded into iptables and block attempts to connect from the named IP to the appropriate port.

The symbiosis-firewall-blacklist script has two distinct phases:

1. It creates a list of candidate IPs (and ports) using the files in /etc/symbiosis/firewall/patterns.d to scan log files. Once it has a list of candidates it creates the files in blacklist.d.
2. It then looks for files to delete.

So let's look at phase 1:

Each file in the patterns.d directory contains the name of a log file. This file is scanned looking for a regular expression match to find likely candidates for blocking. The script uses an sqlite3 database (in /var/lib/symbiosis/firewall-blacklist-logtail.db) to remember where it was in the file the last time it looked at that file. (I suspect that this is the cause of some people's uncertainty about this system. If you add a pattern which you've found in a log file, then it won't be activated until the problem happens again and re-appears in the log file so it can be seen.) However doing this makes sense for efficiency reasons.

The script uses pattern.rb to scan the log file. When matches are found it creates a two dimensional hash

```
results[ip][port] = hit (actual code is results[ip.to_s][port] += 1)
```

so for each port in the pattern.d file, the ip is awarded a single hit. My guess is that this method was used because different pattern files may locate the same bad IP and may also repeat ports. However it means that total hit counts are multiplied by the number of ports in the matched pattern file.

So it's found some lines in a logfile matching the regular expression. Now it wants to decide if any of these entries are worthy of being blacklisted.

This happens if any of the hit counts are greater than 20 (by default), or the value supplied by the -a (--block-after). While it's doing this, it's summing all the hits for the ip and writes them to another sqlite3 database (/var/lib/symbiosis/firewall-blacklist-count.db) along with the IP and a timestamp.

It now looks to see if this IP has been really bad, and is worth blocking completely. It creates a sum of the hits recorded for this IP in the last 24 hours and if this is over 100 (default) or the value determined from -b (--block-all-after) then this IP is set up to be blocked completely for all ports.

So now if it has some candidates, it will create files with the ip address as the names and attach .auto.

Phase 2 is reasonably simple. It looks for the modification time of the file, and if it's expired it's removed.

Thoughts

The Phase 1 algorithm really needs improving and I would say that this is really an urgently needed change. In many cases the system is not reactive enough - and I see that people are using other systems because it's not good enough.

When looking for candidates, the system only 'knows' about the matches it has found in the current slice of the log file that it's inspecting for that pass. For my site, this is typically 3 or 6. My system is not busy so not a lot happens every 15 minutes. The window for selecting candidates is much too small - and is essentially avoidable by delaying attacks.

Second, the system completely ignores the history that it has carefully stored away in the sqlite3 file. Mostly I find that these robots come back from the same IP address often several days later. The system should make much better use of the history that it has gathered.

Third, the current system will theoretically generate port specific matches in iptables. But it's not clear how often this will happen, and frankly I don't care, I'd be happy to completely block every bad site from everything. As an aside it would be good if the code making the firewall understood about 'multiport dports' to make less lines in the filters.

The current selection code is:

```
       results.each do |ip, ports|
          #
          # tot up on a per-ip basis
          #
          total_for_ip = 0

          ports.each do |port, hits|
            total_for_ip += hits

            blacklist[ip] << port if hits > @block_after
          end

          #
          # Record our count
          #
          @count_db.set_count_for(ip, total_for_ip, timestamp)

          #
          # Get the hits for the last 24 hours
          #
          total_for_ip = @count_db.get_count_for(ip, timestamp - 86400)
```

```
          #
          # If an IP has exceeded the number of matches, block it from all ports.
          #
          if total_for_ip > @block_all_after
            blacklist[ip] = %w(all)
          end

         end
```

I am no ruby programmer - so this is a syntactic guess but maybe this will be an improvement - without changing patterns.rb.
If you want to correct the syntax for me please do.

```
        results.each do |ip, ports|
           #
           # get history for this ip for the last day
          # and this ought to be a parameter to the script
          # because I'd like to change this
          # (@block_period defaults to 1 I think)
           #
           historysecs = timestamp - 3600*@block_period
           past_for_ip = @count_db.get_count_for(ip, historysecs)

          #
          # tot up on a per-ip basis
          #
          total_for_ip = 0

          ports.each do |port, hits|
            total_for_ip += hits
            # notice change here to include history
            blacklist[ip] << port if hits + past_for_ip > @block_after
          end

          #
          # Record our count
          #
          @count_db.set_count_for(ip, total_for_ip, timestamp)

          #
          # Get the hits for the some period, which can also be the epoch
          # and again should be a parameter - block_all_period can perhap default to 3
           #
           historysecs = 0
           if @block_all_period > 0
              historysecs = timestamp - 3600*@block_all_period
           end
          total_for_ip = @count_db.get_count_for(ip, historysecs)
          #
          # If an IP has exceeded the number of matches, block it from all ports.
          #
          if total_for_ip > @block_all_after
            blacklist[ip] = %w(all)
          end

         end
```

So this:
a) includes the history for some period when working out the bad guys
b) includes the history for some longer period when evaluating really bad guys
c) Now has an extra lookup for each ip,

This may cause problems if this hits ip addresses that should be in use. There are no tools for removing good guys from the history file. Maybe this can be supplied as a small tool. Perhaps when an address is placed in the whitelist directory, the address should be removed from the blacklist database if it's there.

Peter Collinson (Jan 2019)