

Automating the use of dshield.org's Top 10 list

Tillman Hodgson

Initial Revision: January 30, 2003

Revision date: January 30, 2003

Contents

| | |
|--|----------|
| 1 Background | 1 |
| 1.1 What is Dshield.org? | 1 |
| 1.2 How does the Top 10 list work? | 1 |
| 1.3 What does the Top 10 list look like? | 1 |
| 2 Automatically blocking the Top 10 | 2 |
| 2.1 The idea | 2 |
| 2.2 The benefits | 2 |
| 2.3 The risks | 2 |
| 3 Using IPF for automation | 2 |
| 3.1 Obtaining the Top 10 list | 2 |
| 3.2 Modifying your IPF rules | 2 |
| 3.3 Updating the IPF ruleset with cron | 3 |
| 3.4 Testing the rules | 3 |
| 3.5 Temporarily adding rules | 3 |
| 3.6 What about duplicates? | 3 |
| 3.7 What about removing old rules? | 3 |

1 Background

1.1 What is Dshield.org?

DShield.org was officially launched end of November 2000. Since then, it has grown to be a dominating attack correlation engine with worldwide coverage. While initially run as a volunteer effort, DShield.org has recently received support from the SANS Institute.

DShield.org is the .Distributed Intrusion Detection System,. an attempt to collect data about cracker activity from all over the Internet. This IDS data is cataloged and summarized for use by the general public. It can be used to discover trends in activity and prepare better firewall rules.

DShield is regularly used by the media to cover current security events. Analysis provided by DShield has been used in the early detection of several worms, like "Ramen", "Code Red", "Leaves", "SQL Snake" and more.

It is the goal of DShield to allow access to its correlated information to the public at no charge to raise awareness and provide accurate and current snapshots of Internet attacks.

1.2 How does the Top 10 list work?

Dshield.org collects information that is voluntarily supplied by many Internet-connected organizations around the world. This information is consolidated and tracked over long periods of time.

If a particular IP address is reported as being the source of a large number of attacks by a significant number of Dshield.org users¹, and it continues to do so over at least 5 days, it will be investigated by hand and might make the Top 10 Most Wanted list.

Dshield.org also picks the strongest cases from the logs reported to it and investigations it undertakes and uses these to follow up with the ISP of the attacking IP address. This often gets a much quicker and more thorough result than individual targets attempting to complain to the ISP.

1.3 What does the Top 10 list look like?

The Top 10 list is viewable on the web at <http://www.dshield.org/top10.html>. As of 2:00PM on January 28, 2003 the Top 10 list was:

¹Thus preventing spoofed reports

| IP Address | # of lines implicating this attacker | # of hosts attacked |
|-----------------|--|------------------------|
| 202.172.234.8 | 656422 | 229558 |
| 194.17.211.132 | 234538 | 101151 |
| 217.160.110.151 | 185033 | 54375 |
| 138.121.23.4 | 501396 | 113941 |
| 157.169.10.110 | 87851 | 85841 |
| 213.33.62.110 | 82398 | 79049 |
| 211.199.143.9 | 74490 | 37847 |
| 218.30.21.40 | 381591 | 174389 |
| 208.190.30.203 | 249890 | 148779 |
| 195.218.135.218 | 177059 | 148896 |

As the extreme numbers demonstrate, these IP addresses have been proven unfriendly. It is extremely unlikely that an IP address will end up on the Top 10 list by mistake.

2 Automatically blocking the Top 10

2.1 The idea

That the security of a firewall be increased by pro-actively blocking traffic from IP addresses that have convincingly proven themselves to be hostile by being listed on the DShield.org Top 10 list.

2.2 The benefits

Known hostile IP addresses will not be able to access your network through channels that are normally allowed. For example: email, web browsing, etc. This prevents exploits or denial of service attacks from occurring by preventing the hostile IP from contacting *any* network service.

2.3 The risks

The major risk is that the IP address of a partner or convenient Internet resource may end on the Top 10 list. Due to the checks that DShield employs to prevent this from happening, it is unlikely that this would ever occur.

3 Using IPF for automation

This method of automation uses IPF, `fetch`, `cron` and a one-line `awk` script. It was implemented on FreeBSD 4.7-STABLE, but could probably be easily implemented on any other Unix-ish system that runs IPF.

3.1 Obtaining the Top 10 list

A simple cron job performs the work of obtaining the Top 10 list (via `fetch`²), massaging it to IPF rules format, and writing it to a file in `/usr/local/etc`.

```
#min hour mday month wday who command
44 6,12 * * * root fetch -q -o - \
http://feeds.dshield.org/top10-2.txt | awk '{print \
"block in log quick on r11 from", $1 "/32 to any \
group 121"}' > /usr/local/etc/ipf.top10.rules
```

Note that this is one line, and backslashes signify that the line continues.

I chose two runs a day (in case one run fails for whatever reason), and I run it at 44 minutes after the hour—please select a different minute to run on so that `dshield.org` isn't hit by spikes in traffic.

The result of this cron job is a file, `/usr/local/etc/ipf.top10.rules`, that looks like this:

```
block in log quick on r11 from 202.172.234.8/32 to any group 121
block in log quick on r11 from 194.17.211.132/32 to any group 121
block in log quick on r11 from 217.160.110.151/32 to any group 121
block in log quick on r11 from 213.33.62.1/32 to any group 121
block in log quick on r11 from 157.169.10.11/32 to any group 121
block in log quick on r11 from 216.205.94.116/32 to any group 121
block in log quick on r11 from 66.250.140.10/32 to any group 121
block in log quick on r11 from 208.190.30.203/32 to any group 121
block in log quick on r11 from 218.30.21.40/32 to any group 121
block in log quick on r11 from 24.159.31.102/32 to any group 121
```

This is a nicely formed IPF ruleset. Note that group 121 is specified, this is important to the integration with IPF.

3.2 Modifying your IPF rules

In order to allow rules to be inserted into the middle of the ruleset, we need to use the “groups” feature of IPF. The following `/etc/ipf.rules` skeleton can be used to put groups into effect. Note that you need to put your normal IPF rules in, under group 120.

```
#####
### Notes:
### * The internal interface is simply trusted in/out
### * Group 100 is for the external interface
### * 110 is outgoing traffic originating with this host
### * 120 is incoming traffic destined for this host
### * 121 is the black-list for incoming traffic
###
### Group 121 is loaded by group 120 first, so additions to
### 121 will actually end up in the middle of the rules,
```

²wget would probably also work, but `fetch` is native to FreeBSD

```

### which is very useful.
#####

#####
# Inside Interface
#####

pass in quick on rl0 all keep state
pass out quick on rl0 all keep state

#####
# Group 100 (External Interface)
#####

block out log body on rll all head 110
block in log body on rll all head 120

#####
# Group 110 (External Interface, outgoing)
#####

#-----
# Allow out all TCP, UDP, and ICMP traffic & keep state on it
# so that it's allowed back in automatically
#-----
pass out quick on rll proto tcp from 24.72.10.212/32 \
    to any keep state group 110
pass out quick on rll proto udp from 24.72.10.212/32 \
    to any keep state group 110
pass out quick on rll proto icmp from 24.72.10.212/32 to \
    any keep state group 110

#####
# Group 120 (External Interface, incoming)
#####

#-----
# Block the black-listed IPs group. Note that this means
# that group 121 occurs /before/ the rest of group 120. We
# also don't log it - we know they're bad already.
#-----
block in on rll all head 121 group 120

#-----
# All your other normal firewall rules go here
#-----
# your rules here ...

#-----
# Finally, block politely all other traffic
# Note that the default block policy for this group is thus
# not reached for TCP or UDP connections
#-----
block return-rst in log body quick on rll proto tcp \
    from any to any group 120
block return-icmp-as-dest(port-unr) in log body quick \
    on rll proto udp from any to any group 120

#####
# Group 121 (External Interface, black-listed incoming)
#####

#-----
# This group is separate, see /usr/local/etc/ipf.rules
#-----

```

3.3 Updating the IPF ruleset with cron

This should be done at a time when your other daily cron jobs have already finished but before there is any significant Internet traffic. This is because touching the firewall rules has the potential (however unlikely) for causing trouble and minimizing the potential impact mitigates this somewhat. Somewhere around

5:00AM seems reasonable for many people.

```

#min hour mday month wday who command
00 5 * * * * root ipf -f \
    /usr/local/etc/ipf.top10.rules

```

This cron job loads the rules in `/usr/local/etc/ipf.top10.rules` into the current IPF ruleset without first flushing the ruleset. Normally, this would append the rules to the end of the ruleset, which is likely after your `pass` rules would have already permitted access. However, these rules are in group 121 which are explicitly checked before any `pass` rules. Effectively, this inserts the rules into the middle of the ruleset.

3.4 Testing the rules

To see the “hit count” for your rules, try something like `ipfstat -hin | grep "group 121"`. This lists all the rules in group 121 with the first field being a count of how many times that rule has been “hit” by a packet.

3.5 Temporarily adding rules

Group 121 now becomes useful for temporarily blocking access to hostile hosts. Simply echo an IPF rule that specifies group 121 to the `ipf` command to have to take effect. For example:

```

echo "block in quick from 1.2.3.4/32 to any group 121" \
    | ipf -f -

```

3.6 What about duplicates?

Duplicates don’t cause a problem as IPF will automatically discard them. You may see an error message (`#:ioctl(add/insert rule): File exists3`), but everything will work correctly.

3.7 What about removing old rules?

You can use the `-r` option with `ipf` to remove a rule.

³Where # is the rule number that was duplicated