**Flexible Packet Matching**
February 01, 2010
By Steve Means, CCSP, CCNP, CCSI# 32951

One of the issues facing network and security engineers is defending the network against dynamic threats. This means traffic that cannot easily be identified by regular means, such as an application that changes layer 4 ports or tunnels within an existing protocol. Because of this, a solution is needed to match and filter based on specific information within the packet that is unique to the application you want to block.

Enter “Flexible Packet Matching”, otherwise known as FPM. FPM works by loading XML files called “Protocol Header Description Files” that define fields within the protocol header: IP, TCP, UDP, ICMP, etc…. Once defined, information within the fields can be matched and acted on using MQC.

As a simple example, we’re going to allow ICMP but block echo requests that come from 192.168.9.10 with a length greater than 500.

To enable FPM, you need to first load the PHDFs for the protocol stack you’ll be working with. We’ll be using ETHER, IP and ICMP, so those files are loaded with the “load” command:

```
load protocol system:/fpm/phdf/ether.phdf
load protocol system:/fpm/phdf/ip.phdf
load protocol system:/fpm/phdf/icmp.phdf
```

Next you have to tell FPM the correct protocol stack to examine. This is done with a class-map type stack. Notice that we start at layer 2, match the ethertype for IP, then move to layer 3, matching IP protocol 1 or ICMP.

```
class-map type stack match-all IP_ICMP
 stack-start l2-start
 match field ether type eq 0x800 next IP
 match field layer 3 IP protocol eq 1 next ICMP
```

With the stack defined, we’re now free to match the specific information we’re after, FPM uses “class-map type access-control”.

```
class-map type access-control match-all BAD_ICMP
 match field ICMP type eq 8
 match field IP length gt 500
 match field IP source-addr eq 192.168.9.10
```

Now we’ll create a “policy-map type access-control” to drop any traffic that is a part of this class. This policy map is then nested within another policy map that matches our previously defined protocol stack. It is then applied to the interface:

```
policy-map type access-control DROP_BAD_ICMP
 class BAD_ICMP
   drop
policy-map type access-control FPM
 class IP_ICMP
  service-policy DROP_BAD_ICMP
interface fa0/1
service-policy type access-control input FPM
```

Verification is simple. First we'll ping from a router with an interface IP of 192.168.2.10 with no options:

```
R2# ping 10.6.6.6
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.6.6.6, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 50/58/60 ms
```

The ping is successful as expected. It matches the stack, ICMP type and source address, but not the size. Now we can repeat the ping, upping the size:

```
R2# ping 10.6.6.6 size 1000
Type escape sequence to abort.
Sending 5, 1000-byte ICMP Echos to 10.6.6.6, timeout is 2 seconds:
….
Success rate is 0 percent (0/5)
```

This time the packets were dropped. We can further verify by checking the policy-map on the router using FPM. Notice that we have lots of packets matching the stack class-map, but only our 5 echo requests were dropped:

```
R1#sho policy-map type access-control interface
 FastEthernet0/1

  Service-policy access-control input: FPM

    Class-map: IP_ICMP (match-all)
      136 packets, 15468 bytes
      5 minute offered rate 0 bps
      Match: field ETHER type eq 0x800 next IP
      Match: field layer 3 IP protocol eq 1 next ICMP

      Service-policy access-control: DROP_BAD_ICMP

        Class-map: BAD_ICMP (match-all)
          5 packets, 5070 bytes
          5 minute offered rate 0 bps
          Match: field ICMP type eq 8
          Match: field IP length gt 500
          Match: field IP source-addr eq -1062729462
      drop
```

This is a fairly basic example that is easy to verify and learn with. FPM can be much more detailed, even going so far as to match specific strings within the data payload itself. The preferred method of applying this in the field is to sniff the traffic you're interested in, find something that is unique to this traffic and match based on it.