Flow Label Filtering Feasibility

Orla McGann and David Malone

[1] HEAnet Ltd., Brooklawn House, Shelbourne Road, Ballsbridge, Dublin 4
orla.mcgann@heanet.ie
[2] The Hamilton Institute, National University of Ireland Maynooth
david.malone@nuim.ie

Abstract. The IPv6 header has a field called the Flow Label field, which was included to help identify flows of packets, mainly for Quality of Service purposes. We propose using this field, and its randomly generated value, as another piece of state to filter related packets in a TCP connection. This will only be possible if the Flow Label field actually remains constant for the duration of connections and will only be useful if the Flow Label is set in a unpredictable way.

Key words: IPv6, Flow Label, Stateful Firewalling, SYN Cookies

1 Introduction

Stateful packet filtering currently uses a number of parameters in the TCP and IPv4/IPv6 headers to remember the "state" of a connection, in order to process packets entering or leaving the filter more accurately. For example, port numbers or sequence numbers may be examined to make sure the packet is part of a valid TCP connection.

The Flow Label is a 20-bit field that is part of the IPv6 header. It is used by a source to label certain groups of packets for easy identification and handling by routers, mainly to implement Quality of Service (QoS). These groups of packets are called a "flow". We aim to investigate the usefulness of the Flow Label for Stateful Firewalling; that is, as another method of identifying and filtering packets. Can the Flow Label be examined and stored by the firewall? This would introduce an extra piece of state that needs to be correctly guessed in order to blindly inject spoofed packets into the network, thus adding another layer of protection to the firewall.

The IPv4 header has a similar field, the IP ID field, which is used in fragmentation to identify packets that are part of the same datagram. The IP ID field was used to implement the idle scan attack that was popular in the late 1990's. This exploitation was possible due to the sequential allocation of values in the IP ID field[1].

In contrast, the Flow Label is assigned pseudo-randomly from the range 0x00001 – 0xFFFFF to a packet by the source node. It should be chosen randomly so that any set of bits within the Flow Label field are suitable for use

as a hash key by routers, for looking up the state associated with the flow. The router, or other devices using the Flow Label, should not depend on this being the case if they want to avoid an algorithmic attack on the hash table using maliciously chosen Flow Labels[2]. The Flow Label 0x000000 (zero) has a special meaning and is reserved to indicate that the packet has not been associated with any flow[3].

In IPv4, flow classifiers are based on the quintuple of source and destination addresses and port numbers, and on the transport layer type. This can be problematic as some of these fields may not always be available; either due to fragmentation or encryption. It is more efficient to base the classifiers solely on the IP header, so that the introduction of newer transport types will not require "flow aware" routers to be updated.

Thus for IPv6, a flow is uniquely identified by the source and destination addresses and the (non-zero) Flow Label. All packets belonging to the same flow should be sent with this same triplet. If there is a Priority label, Hop-by-Hop options header or Routing Header associated with the flow, they must also remain consistent throughout the flow (excluding the Next Header field in the Hop-by-Hop options header and Routing Header obviously). RFC 3697[4] states that the Flow Label is immutable and must be delivered unchanged to the destination node(s).

Routers that do not support the functions of the Flow Label are required to: set the field to zero, when a packet originates locally; pass the field on unchanged, when forwarding a packet; and ignore the field, when receiving a packet. Routers may choose to set up a flow-handling state for any packet, even when one has not been explicitly established by the source of the packet. There is no requirement that all, or even most, packets belong to flows. There may also be multiple concurrent active flows between two hosts, as well as traffic that does not belong to any flow.

In order to establish whether it is possible to use the Flow Label for stateful packet filtering, we must first check to make sure that the Flow Label actually stays constant across the lifetime of a flow in practice. Otherwise requiring a fixed Flow Label at a firewall would block legitimate packets.

2 Examining the Flow Label for consistency

A packet capture program[3] was written to track the IPv6 packets on the network. The program extracted the source and destination addresses; ports; transport type and the Flow Label from the IPv6 and TCP headers of each packet. Then a table of active connections was created for listing each flow; as determined by the quintuple of source and destination addresses, source and destination ports, and transport type and each packet was compared to this table. The program then checked to make sure the Flow Labels matched for each packet in a connection; if they did not, the program logged a message. If

[3] The libpcap library is part of the TCPdump program (`http://www.tcpdump.org`)

there was no such flow, it was added to the table of active connections. The program also deleted ("flushed") any stale flow once the connection had been closed or timed out. This allowed us to see whether the Flow Label was kept constant for the lifetime of a TCP connection.

Initial testing with FreeBSD and Debian Linux using the program showed that a Flow Label of "00000" was set by the local FreeBSD machine when establishing connections to both the FreeBSD and Debian hosts. The KAME IPv6 stack, used by FreeBSD, was supposed to set the Flow Label on TCP connections, so we submitted a small patch to implement this.

```
/* update flowinfo - draft-itojun-ipv6-flowlabel-api-00 */
inp->in6p_flowinfo &= ~IPV6_FLOWLABEL_MASK;
if (inp->in6p_flags & IN6P_AUTOFLOWLABEL)
    inp->in6p_flowinfo |=
        (htonl(ip6_randomflowlabel()) & IPV6_FLOWLABEL_MASK);
```

Further testing showed that Debian Linux always sets a Flow Label of "00000", whether it establishes the flow or it is responding to a connection attempt from another host. A stranger phenomenon was noted on FreeBSD however: a Flow Label was set by the remote machine on the first response packet. But, another different Flow Label was then sent when the transfer of data began. This problem was identified as a problem with FreeBSD's SYN Cookies/SYN Caching implementation, where the Flow Label on the SYN/ACK was uninitialised, and then correctly set once the connection was fully established. We discuss this further in Section 3.

Further tests were run to see if the problem identified by our initial observations was widespread, or just local to particular implementations of IPv6 and operating systems that use TCP optimisations. A long list of IPv6 enabled websites[5] was used to generate TCP flows from various IPv6 enabled platforms and the capture program was used to analyse if the Flow Label was being set consistently.

Table 1 contains a detailed break down of the setting of the Flow Label by the various hosts observed. Of the 1105 hosts that sent TCP packets, 427 of them (38.64%) set the Flow Label to something other than "00000"; but only 84 of these consistently set the Flow Label during the flow. The remaining 343 set the Flow Label once during the three-way handshake and then again once the transfer of data started, as was noted in our initial observations. The remaining 678 hosts (61.36%) set the Flow Label to "00000" consistently. These hosts are most likely using some Linux distribution or OpenBSD. OpenBSD explicitly states in its source code that it leaves the Flow Label field set to zero so that it does not require any state management.

The operating system of the hosts analysed was determined from two tests that were performed on each host. The first test probed each host using the `netcat` program[6] to collect the banners that are displayed by hosts when they are remotely accessed. The second test checked to see if the hosts responded to

TCP Hosts and the Flow Label	Hosts	%
Hosts with Flow Label set to something other than 0	427	38.64
Hosts with the Flow Label = 0	784	
Hosts consistently setting Flow Label to 0	678	61.36
Hosts who set the Flow Label to 0 and set it again after handshake	106	
Hosts who set the Flow Label (but not necessarily consistently)	321	
Hosts who consistently set the Flow Label (never changes)	84	19.67
Hosts who inconsistently set the flow (incl. setting it to 0)	343	80.33

Table 1. Table of consistency of Flow Label Setting on TCP packets captured for each host

ICMPv6 Node Information solicitations. This is a feature that is currently only implemented in the KAME IPv6 stack[7], which is used by FreeBSD, OpenBSD and NetBSD. These tests served as a crude form of OS finger printing.

In summary, most of the hosts that inconsistently set the Flow Label either used the FreeBSD or NetBSD operating system. Hosts that consistently set the Flow Label to something other than "00000" were mainly running the patched version of FreeBSD or Solaris. Hosts that consistently set the flow label to zero were either using OpenBSD or some version of Linux.

We note two possible biases in these results. Firstly, a FreeBSD machine is always a part of the connections that were captured, as it was used to run our tests. Secondly, all of the connections captured are responses to connections initiated by the machine conducting these tests. No observations are made of how the Flow Label is set by these remote hosts when they initiate connections to other machines. Monitoring the connections at a popular IPv6 web server might produce different results.

In order to discuss what is actually occurring when different Flow Labels are being set during the TCP connections, and how the problem was fixed for FreeBSD, we must discuss SYN Cookies and SYN Caching. A solution to this problem that was later integrated into FreeBSD will also be described.

3 Fixing FreeBSD's SYN Cookie and SYN Cache Implementation

SYN Cookies[8] were devised as a means of alleviating the problem of SYN flooding[9] without the need for major changes to the TCP protocol. They are created by making a MD5 hash[10][4] from a number of parameters that are found in the SYN and ACK packets of the 3-way handshake. These parameters include the source and destination addresses and ports, and the Maximum Segment Size (MSS). A time-based server selected secret is also included in

[4] Of course another hashing algorithm could be used, but this is the one originally suggested.

the hash. This Cookie is then sent as the 32-bit TCP initial sequence number (ISN) by the server when responding to new connection requests. When the host receives the ACK for this SYN packet, it can rebuild the MD5 hash from this ACK packet. This removes the need to store the ISN and other information about the connection during the handshake.

The main problem with using SYN Cookies to store connections is that other TCP features, such as TCP options, cannot be used. The server remembers nothing about the connection while waiting for the returning ACK, so options that the originating host sets are lost when the server rebuilds the connection from the information stored in the SYN Cookie. If the ACK received matches the ISN sent, then the SYN Cookies will be the same.

A second method for optimising a TCP implementation to prevent SYN flooding, SYN Caching, was developed by David Borman for BSDi[11]. This was then integrated into FreeBSD by Jonathan Lemon[12]. SYN Caching is similar to the original method of storing connection information in a Protocol Control Block(PCB) structure, but instead a much smaller PCB is allocated which only remembers the key information for the setup of the connection (such as TCP options like window scaling and timestamps). FreeBSD now passes all SYNs it receives through SYN Caching, failing over to SYN Cookies if the SYN Cache is full. So, in FreeBSD traditional TCP connection establishment is now defunct.

It is easy enough to keep the Flow Label constant for a simple TCP implementation, as another variable can just be added to the TCP PCB structure to remember the Flow Label. When TCP optimisations are used to accept connections, these techniques must also keep the Flow Label constant. One simple option would be to set the Flow Label to zero for all such connections, but this is not a rather limited implementation.

The problem we noted previously arose because the code in the FreeBSD kernel did not set the Flow Label correctly for packets generated by the SYN Cache/SYN Cookie code; the Flow Label was not initialised on the SYN/ACK packet. It was only being set when the first packet of data was being ACKed. Instead, whatever junk that was in the memory allocated to the packet was being sent as the Flow Label.

Modifications were made to the kernel code to rectify this problem and were subsequently incorporated into FreeBSD. We will describe these modifications here.

In the SYN Cache case, the Flow Label can be generated either sequentially or randomly and stored as a variable added to the syncache structure (`u_int32_t sc_FlowLabel`). We are free to choose either for the SYN Cache case. There is less potential for collisions if the Flow Label is set sequentially, but there is potential for abuse, just as with the IPv4 IP ID field, if sequential Flow Labels are used.

In the SYN Cookie case, the cookie is derived from the source and destination IP addresses and ports and the MSS, as well as a random secret, which are then passed through an MD5 hash. From this hash, 25 bits of the resulting

output are used for the Cookie, and another 7 bits are used for the window-size. These 32 bits are sent as the ISN. We use an additional 20 bits of the MD5 hash to generate the Flow Label. When the ACK is received, the hash can be recreated, as all the original information from the connection will be returned in the ACK packet, and this can be compared to the original Cookie sent as this is stored in the acknowledgement number as the ISN+1. We can do this because we are free to choose any (random) Flow Label for the SYN Cookie case. It is not possible to choose a sequentially increasing Flow Label. We must use the value generated from the hash because it is always possible to recreate the hash, therefore it is always possible to recreate the Flow Label.

Strictly speaking the Flow Label (and addresses) should uniquely identify a flow, but this is not enforced by KAME. The risk of collisions may be reduced by checking for existing flows with that ID and if one exists, resorting to a zero Flow Label (the use of a zero Flow Label could be encoded in a SYN Cookie in a similar way to that used to store TCP Maximum Segment Size[12]).

Now, with the addition of this code, when the ACK is returned a full PCB is constructed and the Flow Label is copied from the SYN Cache, or from the regenerated cookie, allowing subsequent data packets to be sent using the same Flow Label.

The addition of this code means that FreeBSD now sets the Flow Label at the beginning of a TCP connection, and it remains constant across the lifetime of the connection. Any operating system that implements techniques like SYN Caching or Cookies will need to use similar measures to ensure the Flow Label is consistent across the flow.

While completing the analysis of the packet capture program output, it was observed that on a number of occasions a patched machine did not consistently set the Flow Label. After examining the connections in question, it was established that the Flow Label sometimes changed from its properly set Flow Label value to "00000". These inconsistencies only occurred when the patched machine sent a RST packet in response to receiving a packet that corresponded to a flow that had already been shut down.

RSTs are sent when the host receives packets for a connection it does not know anything about, or when a remote host tries to connect to a port that no service is listening on. Hence, it knows nothing of the state of the connection (or lack thereof) so it does not know what to set the Flow Label to so it sets it to "00000". The FreeBSD machine sent RSTs with a Flow Label of "00000" in response to ACKs that were received after the connection had closed or timed out, so it no longer had the Flow Label state stored.

4 Using the Flow Label for Stateful Filtering

Once the setting of the Flow Label is consistent, either set to "00000" or some other value, it is possible to use this information as another piece of state remembered by the firewall. Consider the scenario where host A sends a SYN

packet to host B to initiate a connection with the Flow Label set. The packet filter creates a new state for this connection from the source address, destination addresses, ports and the Flow Label for direction A to B.

Host B sends back a SYN/ACK packet to host A with its own Flow Label set. This is the final piece of state remembered by the packet filter: the Flow Label for direction B to A. Host A sends the final ACK to host B to complete the three way handshake. The firewall checks the Flow Label to make sure it matches the one stored from the original SYN packet. If it does match, the packet is let through. As with normal stateful packet filters, once the packet matches the state stored in the firewall all subsequent packets in the connection are passed through the firewall.

Depending on how strict a firewall wants to be, we suggest a number of actions that can be taken when a packet with an inconsistent Flow Label are seen. It may:

- block packets with a different Flow Label from that recorded.
- replace the stored Flow Label if the Flow Label has changed from `00000` to another value. Then, block packets if the Flow Label changes again.
- Allow the Flow Label to change once, assuming that the label set on the SYN or SYN/ACK (depending on direction) might not have been set correctly. Thereafter, block any packets with a different Flow Label set (for that connection). This might be restricted to the the second packet of the connection.

Flaws in old implementations will cause problems until the patched code for consistent flows propagates. Each of the different strategies proposed to deal with packets from hosts that inconsistently set the Flow Label are problematic in some way.

Obviously, blocking packets that have different Flow Labels set during the same connection is the safest solution. It prevents the possibility of spoofed packets circumventing the the packet filter. On the other hand, there are hosts that inconsistently set the Flow Label at present. There will be a lot of false positives until the patched code propagates, with packets from legitimate hosts being blocked by the filter.

The second option — allowing the Flow Label to change once from "`00000`" to some other value — provides for the correct handling of packets from un-patched hosts by the packet filter. But, this also means that there is an increased possibility of packets circumventing the packet filter. There is also no provision for packets from hosts running the un-patched version of FreeBSD or NetBSD, where the Flow Label is set to whatever was previously stored in the memory, which may not necessarily be zero.

The final solution proposed covers the correct filtering of packets through the packet filter from all hosts, including packets from un-patched machines. Unfortunately, allowing a change to the Flow Label value stored in state opens up the possibility for attackers to blindly inject packets into the connection by replacing the Flow Label in the final ACK of the 3-way handshake with one

of its own choosing, which will block all other legitimate packets from the real host.

It should be noted though, that someone blindly injecting packets into the network and trying to inject their own packets will have to create a packet that matches all the other information stored in state: the source and destination addresses and ports and also be in the expected range of the TCP sequence and acknowledgement fields, which is no mean feat. Carefully checking these fields before accepting a Flow Label change could make these techniques useful while still accommodating unpatched machines.

5 Conclusion

The Flow Label field of the IPv6 header has potential uses other than for QoS. Its use for Stateful Filtering is problematic because of a number of issues in the implementation of the Flow Label in the various IPv6 stacks. Linux and OpenBSD default to setting the Flow Label to "`00000`", which does not reduce the ability to guess firewall state information. FreeBSD and NetBSD have had flaws in their implementation with the setting of the Flow Label, which would require tweaks to a potential stateful packet filter design.

Once the use of Flow Labels becomes more widely implemented and such implementation problems are fixed, the Flow Label should provide a useful additional piece of state to keep the attackers guessing.

References

1. Salvatore (Antirez) Sanfilippo. Idle scan [online]. `http://wiki.hping.org/8`. Last visited 26/09/2005.
2. Scott A. Crosby and Dan S. Wallach. Denial of Service via Algorithmic Complexity Attacks. In *USENIX Security 2003*, aug 2003. `http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach_UsenixSec2003.pdf`.
3. Steve E. Deering and Robert M. Hinden. *RFC 2460: Internet Protocol version Six (IPv6)*, dec 1998. `http://www.faqs.org/rfcs/rfc2460.html`.
4. Jarno Rajahalme, Alex Conta, Brian E. Carpenter, and Steve E. Deering. *RFC 3697: IPv6 Flow Label Specification*, mar 2004. `http://www.faqs.org/rfcs/rfc3697.html`.
5. Sander Jonkers. IPv6 Enabled Sites / IPv6 Accessible Sites [online]. `http://www.prik.net/list.html`. Last visited 23/05/2005.
6. Giovanni Giacobbi. The GNU Netcat – Official homepage [online]. `http://netcat.sourceforge.net/`. Last visited 28/09/2005.
7. KAME project. Webpage of Kame Project [online]. `http://www.kame.org`. Last visited 23/05/2005.
8. Dan J. Bernstein. SYN Cookies. `http://cr.yp.to/syncookies.html`. Last visited 28/09/2005.
9. daemon9. IP-Spoofing Demystified. *Phrack*, 48, sep 1996. `http://www.phrack.org/show.php?p=48&a=14`.

10. Ronald L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*, apr 1992. http://www.faqs.org/rfcs/rfc1321.html.
11. David Borman. TCP-SYN and delayed TCB allocation. http://www.postel.org/pipermail/end2end-interest/2003-May/003118.html. Last visited 23/05/2005.
12. Jonathan Lemon. Resisiting SYN flood DoS attacks with a SYN cache. In *BSDCon 2002*, feb 2002. http://people.freebsd.org/~jlemon/papers/syncache.pdf.