A Connection Management Protocol for Stateful Inspection Firewalls in Multi-Homed Networks

Jin-Ho Kim
Samsung Electronics
Email: jhkim@netlab.snu.ac.kr

Saewoong Bahk
School of Electrical Engineering and Computer Science
Seoul National University
Email: sbahk@netlab.snu.ac.kr

Heejo Lee
Department of Computer Science and Engineering
Korea University
Email: heejo@korea.ac.kr

***Abstract*— To provide network services consistently under various network failures, enterprise networks increasingly utilize path diversity through multi-homing. As a result, multi-homed non-transit autonomous systems (ASes) become to surpass single-homed networks in number. In this paper, we address an inevitable problem that occurs when networks with multiple entry points deploy stateful inspection firewalls in their borders.**

In this paper, we formulate this phenomenon into a state-sharing problem among multiple firewalls under asymmetric routing condition. To solve this problem, we propose a stateful inspection protocol that requires very low processing and messaging overhead. Our protocol consists of the following two phases: 1) Generation of a TCP SYN cookie marked with the firewall identification number upon a SYN packet arrival, and 2) State sharing triggered by a SYN/ACK packet arrival in the absence of the trail of its intial SYN packet. We demonstrate that our protocol is scalable, robust, and simple enough to be deployed for high speed networks. It also transparently works under any client-server configurations. Last but not least, we present experimental results through a prototype implementation.

I. Introduction

Enterprise networks increasingly leverage path diversity through multi-homing because a single Internet Service Provider (ISP) is not enough to provide consistent performance. Today, multi-homed non-transit autonomous systems (ASes) surpass single-homed networks in number [11], [3]. However, since BGP policies provide very coarse control over communication paths and the global effect on its traffic is difficult to predict or control, asymmetric routing is a real possibility for multi-homed networks. Even before the current advent of multi-homed networks, it was shown that routing paths are often asymmetric in the Internet [5].

Meanwhile, most enterprise networks need to deploy firewalls in their border to protect themselves from illegitimate traffic. Recently, what is called the stateful inspection firewall has become the de facto industry standard. It enables flexible and fine-grained control over incoming traffic, and the filtering decision can be dynamically made based on the need of the outgoing traffic. Namely, a stateful inspection firewall intercepts a packet and updates its internal state table with the extracted connection state information (source and destination addresses and port numbers), based on which the filtering decision is made for the incoming traffic [9]. However, this means that the stateful inspection has a topological restriction: outgoing and incoming traffic of a connection should pass through a single firewall.

The failure of meeting this restriction due to the possibility of asymmetric routing leads to the connection establishment problem. Namely, if an AS is with multiple entry/exit points (henceforth, MEPs), outgoing and incoming flows may go through different firewalls. In this case conventional stateful inspection designed for a single entry point (henceforth, SEP) does not work as desired. In this paper, we address this inevitable problem that occurs when MEP networks deploy firewalls in their border. In particular, we explore a state sharing method between firewalls used in the MEP networks and the stateful inspection mechanism to filter out packets from suspicious connections.

The main contributions of this work are three-fold. First, we formulate the problem of firewalling for an MEP network. Second, we propose a method of stateful inspection for firewalls in the MEP network, and design an effective mechanism for state information exchange among firewalls in it. Our proposed mechanism is scalable because its complexity has nothing to do with the increase of firewalls in number. Third, through a prototype implementation, we verify that the high-speed packet filtering is achievable in the MEP network.

The paper is organized as follows. Section II describes the state sharing problem in asymmetric routing environments, then discusses state sharing methods between firewalls during the connection establishment procedures. In section III, we propose an algorithm using the modified SYN cookies by considering the state synchronization problem. Sections IV discusses performance issues and implementation results respectively, followed by the conclusion in section V.

II. State Exchange for Connection Establishment

This section describes the state sharing problem in MEP environments, and considers two state sharing approaches that take different steps during the TCP handshaking. Then, we briefly review the principles of a SYN cookie [10] for our use in transferring connection information among firewalls.

A. State Sharing Problem

We let $FW(c, s)$ denote the firewall on the one-way routing path from client host c to server host s, where either c or

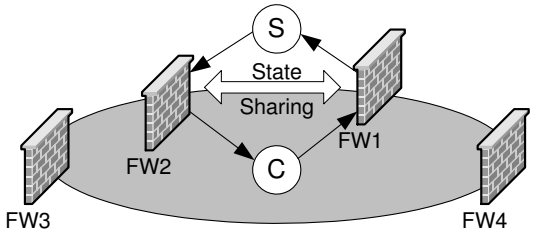


Fig. 1. State sharing problem for asymmetric routing.

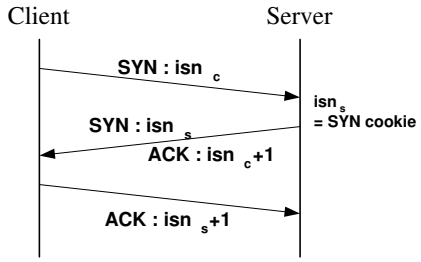


Fig. 2. TCP 3-way handshaking with original SYN cookie.

s is in the considered AS to be protected by firewalls. We call the routing path of a connection is symmetric when $FW(c,s) = FW(s,c)$. Otherwise, it is asymmetric. State sharing between firewalls is needed when a routing path is asymmetric for any client-server pair (c,s). For simplicity, let FW_x denote $FW(c,s)$ and FW_y denote $FW(s,c)$.

Fig. 1 shows an example of network with 4 firewalls, $FW_x = FW_1$ and $FW_y = FW_2$. To do stateful inspection of a TCP connection, FW_1 needs to share the state information with FW_2. Assume that a SYN packet, referred to as SYN(c,s), passes through FW_1 and its corresponding SYN/ACK packet, referred to as SYNACK(s,c), arrives at FW_2. To check the validity of SYNACK(s,c), FW_2 needs to know the state information of this connection, otherwise it will drop the SYNACK(s,c). Thus, a state sharing mechanism is required for firewalls to protect the MEP network against possible attacks. Our goal is to minimize the cost for state exchange among firewalls.

B. Two State Sharing Methods

We can consider two possible ways of state sharing for a TCP connection according to the initiator of state sharing.

1) SYN-Initiated State Sharing: In a SYN-initiated algorithm, a firewall receiving a SYN(c,s) initiates information exchange by sending new connection information to all other firewalls. A drawback of this method is that it is hard to know in advance which firewall will receive the SYNACK(s,c). In case of symmetric routing, state sharing among firewalls is not necessary for connections. But if a firewall cannot convince the routing path a priori, it should send the state information to every other firewall which does not need it except the designated one. Such a SYN-initiated algorithm incurs $m-1$ extra packets per connection, where m is the number of firewalls. Thus, messaging overhead of $m-1$ packets per SYN packet renders firewalls vulnerable to denial of service (DoS) attacks with $O(m)$ amplification.

In addition to the redundant messages, a SYN-initiated approach has a race condition between state sharing and SYNACK(s,c) arrival.

2) SYN/ACK-Initiated State Sharing: Under the SYN/ACK-initiated state sharing, FW_y initiates information exchange upon receiving a SYNACK(s,c). If FW_y receives a SYNACK(s,c) and it has not seen the corresponding SYN(c,s) before, *i.e.*, $FW_y \neq FW_x$, it requests the connection information to the other firewalls.

The advantage of this method is that extra messages are not generated for symmetric routing cases. The messaging overhead of SYN/ACK-initiated sharing is proportional to $\alpha \cdot (m-1)$, $0 \leq \alpha \leq 1$, where α is the portion of asymmetric routing. The SYN/ACK-initiated sharing has negligible messaging overhead when $\alpha \approx 0$.

Nevertheless, it is still susceptible to DoS attacks like floods of spoofed SYN/ACK packets. Fake SYN/ACK packets amplify the number of messages because a single SYN/ACK packet invokes $m-1$ control packets. This amplification effect causes DoS of the firewalling network.

These drawbacks motivate us to devise a novel approach without incurring the amplification effect. There are two objectives of our protocol design:

1) *Only a valid SYN/ACK initiates state sharing,*
2) *FW_y inquires only FW_x instead of all the others.*

In this case, the messaging overhead is only proportional to α and independent of m. To achieve these objectives, upon receiving a SYNACK(s,c), the firewall needs to check the validity of the SYNACK(s,c), and identify which firewall the corresponding SYN(c,s) passed.

To run the protocol in an efficient manner, we manipulate the initial TCP sequence number (ISN) of a SYN packet, which is similar to SYN cookies [10]. SYN cookies were originally suggested by Bernstein to defend a TCP server against SYN flooding attacks with IP spoofing [2].

C. Original SYN Cookie

The SYN cookie was designed to defend against SYN flooding attacks. It is a particular choice of initial TCP sequence number (ISN), which represents connection state. Hosts send out SYN/ACK packets each with the SYN cookie instead of keeping all the connection information to achieve the scalability. Thus, hosts using SYN cookies do not have to drop connections even if the backlog queue fills up [10].

Fig. 2 shows a TCP 3-way handshaking with a SYN cookie. Let isn_c and isn_s denote the ISNs sent by a client and a server, respectively. Upon receiving a SYN packet, the server generates a SYN cookie[1] according to the following.

$$isn_s = hash(sec1, saddr, sport, daddr, dport)$$

[1]The format of SYN cookie that we are using is different from the original one. The modified SYN cookie format will be discussed in section III-C.

$$
\begin{aligned}
&+ \quad isn_c + (time * 2^{24}) \\
&+ \quad (hash(sec2, saddr, sport, daddr, dport, \\
&\quad time) \bmod 2^{24}) + mss_{index}, \qquad (1)
\end{aligned}
$$

where $time$ represents a 5-bit counter increasing every 64 seconds, mss_{index} represents an encoded value of the MSS in [0,7], $sec1$ and $sec2$ represent secret keys which only the server knows, and $hash$ represents a cryptographic hash function such as MD5 or SHA-1. $saddr$, $sport$, $daddr$ and $dport$ represent source address, source port, destination address, and destination port of the SYN packet, respectively. If a client receives the SYN/ACK packet with the SYN cookie, *i.e.*, isn_s, it sends an ACK packet with the acknowledge number of $isn_s + 1$. A server receiving the ACK packet checks its acknowledge number by using the following.

$$
\begin{aligned}
mss_{index2} &= acknum - seqnum - time * 2^{24} \\
&- \quad hash(sec1, saddr, sport, daddr, dport) \\
&- \quad (hash(sec2, saddr, sport, daddr, dport, \\
&\quad time) \bmod 2^{24}), \qquad (2)
\end{aligned}
$$

where $acknum$ and $seqnum$ represent the acknowledge number and the sequence number of ACK packet respectively. If mss_{index2} is in [0,7], the ACK packet is considered as legitimate, and the server creates a connection with the MSS corresponding to mss_{index2}. In case that a packet is forged, mss_{index2} tends to be very different from a value in [0,7].

III. Distributed Stateful Inspection Protocol

In this section, we propose a distributed stateful inspection protocol for coordination of multiple firewalls in an AS.

A. Protocol Design

The requirements of our protocol design for connection state exchange are as follows:

- It needs to be secure as much as a SEP firewall.
- It generates low overhead to maintain connection states by minimizing the number of extra control packets, the amount of computation and data storage.
- It guarantees no race condition.
- It is compatible with the Internet infrastructure.

In this protocol, we take advantage of SYN cookies to securely exchange connection information between the two associated firewalls. As the SYN cookie was originally applied for SYN/ACK packets, we modify it to make it applicable to SYN packets. Unlike the original SYN cookie, the modified SYN cookie uses the firewall ID instead of mss_{index} to securely record the cookie sender's information. The verification of SYN/ACK packets is performed by a keyed-hash function. To use this function, all the firewalls in the AS need to share the same secret key. Our proposed protocol illustrated in Fig. 3 is described as follows.

1) C sends a SYN packet which arrives at FW_x.
2) FW_x examines the packet according to its packet filtering rules. If the requested connection is valid, continue the next step.

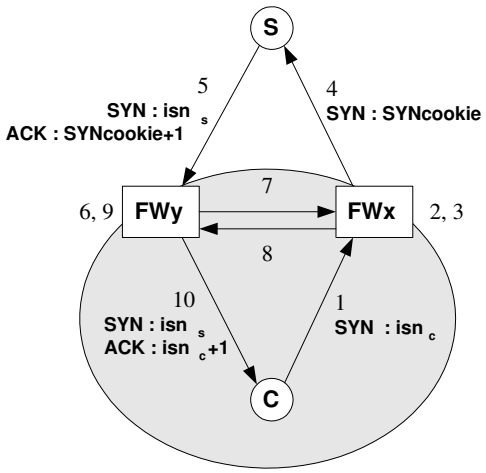


Fig. 3. Connection State Exchange Protocol.

3) FW_x replaces isn_c with the modified SYN cookie, and keeps[2] the connection information[3] at the state table.
4) FW_x sends the modified SYN packet to S.
5) S sends C a SYN/ACK packet which will go through FW_y.
6) FW_y examines the SYN/ACK packet and extracts the firewall ID of FW_x. If the packet is invalid, it will be dropped. If $FW_x = FW_y$, go to step 9.
7) FW_y forwards the SYN/ACK packet to FW_x.
8) FW_x checks the connection information of the packet and sends it to FW_y with the SYN/ACK packet. If there is no corresponding connection information, FW_x drops the packet.
9) FW_y updates its state table and replaces the acknowledge number of the SYN/ACK packet with $isn_c + 1$.
10) FW_y sends the modified SYN/ACK packet to C.

Therefore FW_x and FW_y share the connection information, and forthcoming packets for the connection including the corresponding ACK can pass through the two associated firewalls directly.

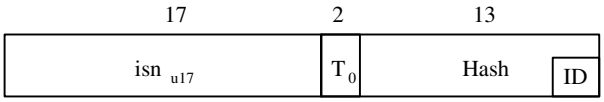
B. Reincarnation of a TCP Connection

In [7], there are regulations about TCP connection reincarnation which are given as follows.

> When a connection is closed in active state, it *MUST* linger in TIME-WAIT state for a time 2*MSL (Maximum Segment Lifetime). However, it *MAY* accept a new SYN from the remote TCP to reopen the connection directly from TIME-WAIT state, if it:
> - assigns its ISN for the new connection to be larger than the largest sequence number it used on the previous connection incarnation, and

[2]The original SYN cookie does not have this storing process.

[3]Source address, source port, destination address, destination port, and difference of the sequence numbers between isn_c and the SYN cookie.



Isn_{u17} : Upper 17bits of isn received
T_0 : 2 LSBs of time counter, Time
Hash : Hash(Sa,Sp,Da,Dp,Time,isn_{u17},Secret Key)
ID : Firewall ID

Fig. 4. Modified SYN cookie.

- returns to TIME-WAIT state if the SYN turns out to be an old duplicate.

When S stays in TIME-WAIT state, it may accept a new SYN packet sent by C. To support the reincarnation, the ISN in the new SYN packet should be larger than the latest sequence number, denoted by SN_{prev}, which was sent for the previous connection incarnation.

First of all, our protocol is designed to minimize processing and control overheads rather than to support fast reincarnation [7] of closed TCP connections. So we build our protocol on the following two premises. First, fast reincarnation is not used often. This is because there is scarcely any protocol or application that requires fast reincarnation. Second, hosts who want fast reincarnation would generate ISNs of larger than $SN_{prev} + 32,768$.

The number 32,768 comes from the following reasons. 4.4BSD-Lite adds 64,000 to the ISN of each connection in addition to timer increase [12], while NetBSD-1.5.2 adds 16,777,216. FreeBSD-4.5 and Linux 2.4.17 use 1MHz timer so every 33ms the ISN is increased by 32,768. For OSes which use 250KHz timer proposed in [6], it takes 132ms to increase the ISN by 32,768. However these instances are not directly related to our case of fast reincarnation because they are just concerned with increasing ISNs as a part of connection setup.

Nevertheless, fast reincarnation is made possible by increasing the ISN by 'at least' 32,768 from SN_{prev}. The increment of smaller than 32,768 is not enough to secure a hash field of our modified SYN cookie which will be explained next.

If the second premise is fulfilled, the upper 17 bits of the ISN sent by C becomes larger than SN_{prev}. Therefore, it makes sense to leave the upper 17 bits intact to support reincarnation properly.

C. Modified SYN Cookie

The proposed SYN cookie format is depicted in Fig. 4. It is assumed that all firewalls share a secret key and a synchronized time counter which increases every 16 seconds. The upper 17 bits of the SYN cookie are taken from that of the ISN of the received SYN packet. T_0 is the two least significant bits of the time counter which enables a firewall to extract the time of the SYN cookie made. The firewall receiving the SYN cookie extracts the time according to the following equation and uses it as an input to a hash function that validates the SYN cookie.

$$time_{input} = time_{curr} + 1 - ((time_{curr} + 1 - (SC >> 13)) \bmod 4), \quad (3)$$

where $time_{curr}$ is the current time and SC is the SYN cookie. This time is effective for about 40 seconds after the generation of SYN cookie. We will explain the meaning of (3) and time synchronization in section IV-C.

The modified SYN cookie also uses a keyed-hash function to defend against DoS attacks of SYN/ACK packet flooding. But it has the output length of 13 bits differently from the original one of 32bits. So the probability of randomly guessing the correct hash value in an attempt is 2^{-13}. Additionally we include the firewall ID by sacrificing the length of hash output. For instance, if an AS is with 4 firewalls, 2 bits are required to identify a firewall so that the probability of randomly guessing the correct hash becomes 2^{-11}. This implies that four out of 2^{13} fake packets can be classified as legitimate and three of them will be forwarded to other firewalls. These packets, however, will be filtered out by receiving firewalls because their state information is not in the table. The other packet will be also dropped by the first firewall due to the same reason. So the flooding attack is tolerable because it incurs only one temporary extra packet per $2^{13}/(m-1)$ attacking packets.

IV. Performance

In this section, performance issues are discussed. We implemented the proposed protocol using the netfilter [4] in the Linux kernel[4] and conducted experiments for our proposed protocol on a testbed with three PC firewalls.

A. Processing Overhead

As the algorithm contains no loops, its complexity is $O(1)$ per packet. However it contains the hash function and the packet modification code, which need to be optimized.

1) Hash function: For a normal connection, the hash function is called twice. One is for the modified SYN cookie generation, and the other for verification. If attackers send a lot of SYN/ACK packets, firewalls have to call the hash function frequently. As the hash function affects overall performance very much, it should be chosen carefully. There exist commercial network security processors that achieve multi-gigabit throughput using MD5 and SHA-1 hash functions. Also, fast hash functions like UMAC and MMH can achieve multi-gigabit performance even on a 350MHz Pentium II PC [1].

2) Sequence Number Translation: Since the firewall changes isn_c, it needs to translate all the sequence numbers of packets in one direction and all the acknowledge numbers in the opposite direction, and recalculate their TCP checksums. The sequence number (SN) translation can be made possible by one addition or subtraction simply, and TCP checksum (CS) can be recalculated by a simple arithmetic [8].

$$CS_{new} = \sim (\sim CS_{old} + \sim SN_{old} + SN_{new}) \quad (4)$$

[4]Linux kernel version 2.4.19 is used for implementation.

where $\sim a$ means one's complement of a.

Therefore, the sequence number translation has negligible impact on the performance.

B. Control Packet Overhead

It is obvious that the proposed protocol requires extra control packets at minimum for connection establishment. It needs two packets for the request-and-reply for state exchange only when the routing path is asymmetric.

C. Security

Firewalls with MEPs should be as secure as SEP firewalls. We consider two types of attacks in view of security depending on how the firewall reacts when it receives SYN and SYN/ACK packets.

1) DoS attacks: There are two possible DoS attacks, SYN flooding attack and SYN/ACK flooding attack. On receiving SYN and SYN/ACK packets, firewalls execute hash functions. As mentioned before, we need to choose a hash function with high performance.

2) Fake SYN/ACK attack: The effectiveness of the proposed protocol with respect to verifying SYN/ACK packets is measured under a flooding attack of fake SYN/ACK packets. We generated 1,000,000 SYN/ACK packets with random acknowledge numbers. Among 1M forged packets, only 251 packets which are very close to the expected value of 244[5] were forwarded to the other firewalls, and they were all dropped at the receiving firewalls. This verify that no fake SYN/ACK packets could succeed to penetrate into our network.

3) Replay Attacks: We use the current time as an input of the hash function to prevent replay attacks. After 64 seconds, a SYN cookie will be regarded as invalid and the packet will be dropped accordingly. The proposed protocol prevents an attacker from guessing the SYN cookie without having seen it recently, which is the same as the original SYN cookie. To prevent the replay attack, it is assumed that all firewalls have a synchronized time counter, which increases every 16 seconds. We inserted the two least significant bits of the time counter, $time_{org}$, and extracted $time_{input}$ by using (3). Then, the $time_{input}$ has the following property.

$$\begin{aligned} & time_{org} = time_{input} \\ & \quad \Longleftrightarrow \; time_{org} - 1 \leq time_{curr} \leq time_{org} + 2. \end{aligned} \tag{5}$$

The reason we allow $(time_{org} - 1)$ and $(time_{org} + 2)$ is to tolerate time asynchronization between firewalls for up to 16 seconds. If the time asynchronization period does not exceed 16 seconds, the SYN cookie is valid for 16 to 64 seconds.

D. Scalability

A good feature of our proposed protocol is its scalability because it runs independently of the number of firewalls in an AS. This is caused by the fact that a connection exchange occurs just between the two associated firewalls with their own identifiers. Our protocol has low complexity and does not have looping problems. Desirably it does not generate any unnecessary control packets for symmetric connections.

[5]In case of $m = 3$, the expected value is $1,000,000 * (3-1)/2^{13} \approx 244$.

V. Conclusion

The state sharing problem occurs among firewalls in the MEP network as the Internet intrinsically allows connections to experience asymmetric routing paths. Conventional firewalls have topological restrictions since they have no ability to do stateful inspection in MEP environments. Such a problem has hindered deploying stateful inspection firewalls in the MEP environments.

In this paper, we described the state sharing problem between two associated firewalls in the MEP network. Also we proposed a distributed stateful inspection protocol which exchanges connection information and checks its validity by using the modified SYN cookie. Differently from the original cookie, our cookie contains the field of firewall ID to indicate which firewall the SYN packet passed through, and a hash value to check the validity of a connection request. When the firewall receives a SYN/ACK packet, it examines the hash value and extracts the firewall ID from the acknowledge number to defend against DoS attacks employing fake SYN/ACK flooding.

The proposed protocol is scalable because the control messages are exchanged between the two associated firewalls of an asymmetric connection, regardless of the number of entry points in an AS. The protocol requires low processing and messaging overhead, thereby it can be applicable to current Internet environments. For future work, we left the case of supporting UDP sessions in the MEP network.

References

[1] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, *UMAC: Fast and Secure Message Authentication*, Advances in Cryptology - CRYPTO99, 1999.

[2] *TCP SYN Flooding and IP Spoofing Attacks*, CERT Advisory CA-1996-21, September 1996.

[3] J. Johnson, *BGP is a reachability protocol*, a NANOG presentation, June 2002, `http://www.nanog.org/mtg-0206/ppt/jerm2/`

[4] Netfilter Homepage, `http://www.netfilter.org`

[5] V. Paxson, *End-to-End Routing Behavior in the Internet*, Proceedings of ACM SIGCOMM, 1996.

[6] J. Postel, *Transmission Control Protocol*, STD 7, RFC 793, September 1981.

[7] R. Braden, *Requirements for Internet Hosts - Communication Layers*, STD 3, RFC 1122, October 1989.

[8] A. Rijsinghani, *Computation of the Internet Checksum via Incremental Update*, RFC 1624, May 1994.

[9] *Stateful Inspection Technology*, Check Point Tech note, `http://www.checkpoint.com/products/solutions/technologies.html`

[10] SYN Cookies Homepage, `http://cr.yp.to/syncookies.html`

[11] D. Vukadinovic, P. Huang, and T. Erlebach, *A Spectral Analysis of the Internet Topology*, Technical Report ETH-TIK-NR 118, 2001.

[12] G. Wright and W. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*, Addison-Wesley, 1995.