Accommodating Fragmentation in Deterministic Packet Marking for IP Traceback

Andrey Belenky and Nirwan Ansari
Advanced Networking Laboratory, ECE Department, NJIT, Newark, NJ 07102, USA

***Abstract*— [1] We propose a modification to the basic Deterministic Packet Marking (DPM), a promising IP traceback scheme, to handle fragmented traffic. The modification introduces no additional bandwidth overhead, but limited additional memory requirements and processing overhead on the DPM-enabled interface.**

Index Terms **— Security, IP Traceback**

I. Introduction

Fragmented traffic constitutes between 0.25% according to [1] and 0.5% of the total IP traffic according to [2]. Though the amount of fragmented traffic is small, it does exist. Deterministic Packet Marking (DPM) [3], a recently proposed IP traceback scheme, is scalable and capable of tracing back attacks, which are composed of just a few packets. However, the basic DPM proposal did not differentiate between fragmented and non-fragmented traffic. The Identification (ID) Field of the IP header was completely replaced with a certain set of bits of the ingress interface IP address chosen at random.

In this article, we discuss why the basic DPM may not be the best way to handle fragmented traffic, and present a modification to DPM to address fragmentation. The rest of the article is structured in the following way: the basic DPM algorithm is described in Section II; IP Fragmentation and associated terminology are discussed in Section III; the difference between the upstream and the downstream fragmentation is introduced in Section IV; the limitations of the basic DPM with respect to the fragmented traffic are discussed in Section V; two modifications proposed for the basic DPM to alleviate the problem associated with fragmented traffic are presented in Section VI; the circumstances which may cause the reassembly error even with the introduced modification are described and analyzed in Section VII; finally, we conclude in Section VIII

II. Basic Deterministic Packet Marking (DPM)

This section provides the general principle behind DPM and discusses the most basic implementation of the proposed scheme.

A. DPM Principle

DPM is a packet marking algorithm. The 16-bit Packet ID field and 1-bit Reserved Flag (RF) in the IP header will be used to mark packets. *Each* packet is marked when it enters the network. This mark remains unchanged for as long as the packet traverses the network. The packet is marked by the interface closest to the source of the packet on an edge ingress router, as seen in Figure 1. The mark is a partial address information of this interface, and will be addressed later in Section II-B. The interface makes a distinction between incoming and outgoing packets. Incoming packets are marked; outgoing packets are not marked. This ensures that the egress router will not overwrite the mark in a packet placed by an ingress router.

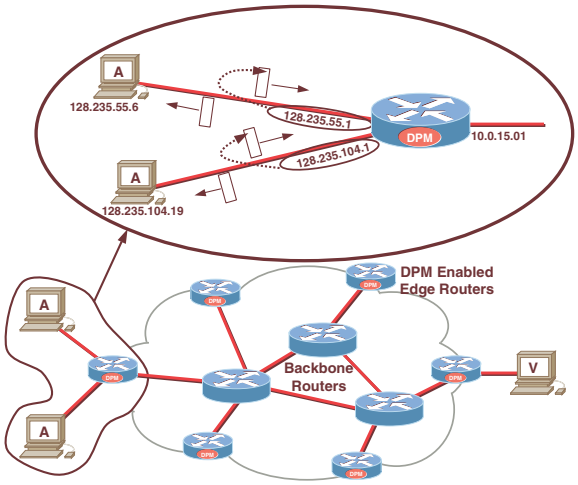

Fig. 1. Deterministic packet marking.

For illustrative purposes, assume that the Internet is a network with a single administration. In this case, only interfaces closest to the customers on the edge routers will participate in packet marking. The marking will be done deterministically. Every incoming packet will be marked. Should an attacker attempt to spoof the mark in order to deceive the victim, this spoofed mark will be overwritten with a correct mark by the very first router the packet traverses.

B. Coding of a Mark

A 32-bit IP address needs to be passed to the victim. A total of 17 bits are available to pass this information: 16-bit ID field and 1-bit RF. Clearly, a single packet would not be enough to carry the whole IP address in the available 17 bits. Therefore, it will take at least two packets to transport the whole IP address. An IP address will be split into two segments, 16 bits each: segment 0 – bits 0 through 15, and segment 1 – bits 16 through 31. The marks are prepared in advance in order to decrease

[1]This work has been supported in part by the New Jersey Commission on Higher Education via the NJI-TOWER project, and the New Jersey Commission on Science and Technology via NJWINS.

Marking procedure at router R, edge interface A:
for y = 0 **to** 1
$Marks[y]$.Seg_Num := y
$Marks[y]$.A_bits := $A[y]$
for each incoming packet w
let x be a random integer from [0,1]
write $Marks[x]$ into w.Mark

Ingress address reconstruction procedure at V:
for each attack packet w
$IngressTbl[w.\text{Mark.Seg\_Num}]$:= w.Mark.Seg_Num

Fig. 2. Pseudo code for the basic DPM algorithm.

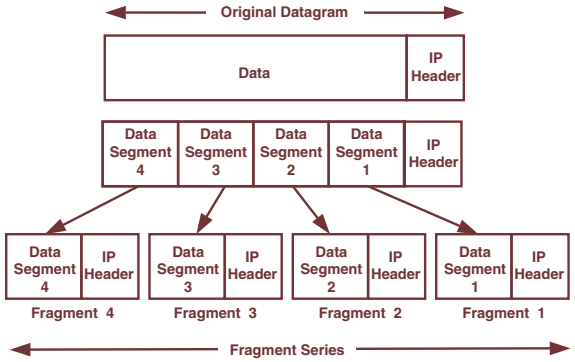

Fig. 3. IP fragmentation.

the per packet processing. Each mark has two fields: Segment Number and Address bits. With probability of 0.5, the 17-bit field comprised of the ID field and RF of each incoming packet will be populated with either of those two marks. It is necessary to introduce this randomness into the scheme so that sophisticated attackers would not send exactly every other packet to the victim, and by doing that creating a situation when only one part of the address is available to the victim.

C. Formal Description of Basic DPM

In this section, we introduce the formal pseudo-code for DPM. As seen in Figure 2, all edge interfaces on all edge routers will place either the first or the last 16 bits in every incoming packet in the ID field, and set the reserved flag to the appropriate value. At the victim, we suggest that the table matching the source addresses to the ingress addresses is maintained. The victim would check to see if the table entry for a given source already exists, and create it if it did not. Then, it would write appropriate bits, depending on the value of the reserved flag, into the ingress IP address value.

III. IP Fragmentation Background and Terminology

Terminology used to describe different aspects of fragmentation is largely adopted from [2].

Fragmentation is a feature of Internet Protocol (IP) to enable transport of packets across the networks with different Maximum Transfer Unit (MTU). *Path MTU* is the smallest MTU of all the links on a path from a source host to a destination host as described in [4]. When a packet enters a network, whose MTU is smaller than the packet length, the packet has to undergo a process of *fragmentation*.

Figure 3 illustrates this process and introduces several important terms. The *original datagram* is an IP datagram that will be fragmented because its size exceeds the MTU of the next link. A *Packet Fragment*, or simply a *fragment*, refers to a packet containing a portion of the payload of an original datagram. While the datagram and packet are synonymous, the terms, *original datagram* and *packet fragment*, will be used for clarity. A *fragment series*, or simply a *series*, is an ordered collection of fragments resulted from a single original datagram.

When fragmentation occurs, each fragment becomes a valid IP packet. All the fragments have their own IP header. Most of the fields of the IP header of the fragments are inherited from the original datagram IP header. The fields of interest are the ID field, Flags, and Offset. The ID field is copied from the original datagram to all the fragments. The Source Address (SA), Destination Address (DA), Protocol (P), and ID, are used by the destination to distinguish the fragments of different series [5], [6]. The ID field of all the fragments, which resulted from a single datagram, must have their ID field in the IP header set to the same value for proper reassembly. The More Fragments (MF) flag is set to '1' in every fragment except the last one. This flag indicates that more fragments to follow. The last fragment has MF set to '0' to indicate that it is the last fragment in the series. Finally, the offset field of the IP header is set to the position of the data in the fragment with respect to the beginning of data in the original datagram. The offset is measured in the units of 8 bytes.

For successful reassembly, the destination has to acquire all of the fragments of the original datagram. A tuple (SA, DA, P, ID) is used to determine if the fragments belong to the same original datagram, MF is used to indicate the number of fragments, and Offset is used to determine the correct order of reassembly. Notice that the fragments may come out of order but reassembly will still be successful because the destination would be able to determine that the fragment belongs to a given series, and its position relative to other fragments.

Since DPM uses the Identification field for its purposes, it may create a situation where reassembly will fail. We first examine the effect of the basic DPM on reassembly and then introduce a technique to deal with the undesirable effects. Finally, we analyze the performance of the techniques in terms of the probability of reassembly error.

IV. Upstream vs. Downstream Fragmentation

Fragmentation can happen *upstream* or *downstream* from the point of marking according to [1]. These two situations have to be considered separately.

Upstream fragmentation is known to the DPM-enabled interface. The DPM-enabled interface can identify a packet to be a fragment by examining its MF and Offset. DPM

can employ a different strategy for marking these packets. Downstream fragmentation is unknown to DPM. The DPM-enabled interface has no knowledge if the datagrams, marked by DPM, are being fragmented anywhere along the path.

V. Shortcomings of the Basic DPM

Recall that the DPM-enabled interface, running the basic DPM randomly inserts either the first or second 16 bits of its address in the ID field, and sets the reserved flag (RF) to '0' or '1' to indicate which group of bits the ID field contains.

Fragmentation downstream from the DPM-enabled interface causes few problems for reassembly. The router, which is going to perform fragmentation, will simply insert the content of the ID field of the original datagram into every fragment. At the destination, reassembly will be successful since the ID field will be the same for every fragment in the series. The fact that the ID field (set by the originating host) was replaced/marked by DPM is not known to the destination, and is thus irrelevant for the purpose of reassembly. The only problem, which may occur for the datagrams fragmented downstream, is the problem of *interlacing*, defined and explained in Section VII.

Upstream fragmentation will cause more problems for the reassembly at the destination. In the case of upstream fragmentation, a datagram is fragmented by a router or a host before it reaches the DPM-enabled interface. In this case, a series of fragments of the original datagram will reach the DPM-enabled interface. Since the basic DPM does not distinguish between fragments and non-fragments, the scheme will randomly replace the ID field of all the fragments with either the first or second 16 bits of the interface address. This will cause fragments to have different ID fields when they arrive to the destination. Fragments with different ID fields will be considered to be parts of different datagrams. The reassembly will eventually "timeout" since the destination will never get all the fragments necessary for the reassembly of what it considers to be two separate series. The probability of all fragments in a series of two fragments having the same ID field after marking is 0.5. For a series of three fragments, 0.25, etc. Clearly, the rate of reassembly errors caused by upstream fragmentation is unacceptable. To avoid this situation, the modification to the basic DPM is introduced.

VI. Fragment-Persistent DPM

It is essential for proper reassembly that all of the fragments of the original datagram have the same ID field. The basic DPM marks packets probabilistically, randomly choosing between the first and the last 16 bits of the ingress IP address. This random behavior must be suspended when processing fragments. In order to accomplish this task, DPM has to keep track of the fragments, which pass through. If the first fragment which DPM encounters (which does not have to be the fragment with offset 0) is marked with the first or last 16 bits, then the rest of the fragments of this datagram must be marked with the same bits. This information has to be stored as a table at the DPM enabled interface and checked every time a new fragment arrives. To identify fragments belonging to the same original datagram, DPM should check if the tuple of the four fields utilized by the reassembly function (SA, DA, P, ID) is the same as any other it marked within the maximum reassembly timeout of 120 seconds.

Marking procedure at router R, edge interface A:

for y = 0 **to** 1
  $Marks[y]$.Seg_Num := y
  $Marks[y]$.A_bits := $A[y]$
for each incoming packet w
  let x be a random integer from [0,1]
  if w.MF == '1' OR w.offset $\neq 0$ **then**
    if $FragTbl[SA, DA, P, ID] == NIL$ **then**
      create $FragTbl[SA, DA, P, ID]$
      $FragTbl[SA, DA, P, ID]$:= x
    else
      x := $FragTbl[SA, DA, P, ID]$
  write $Marks[x]$ into w.Mark

Fig. 4. Pseudo code for the fragment-persistent DPM.

Figure 4 illustrates the required modifications to DPM to support fragmentation. If the packet is not a fragment, then it would be treated as in the basic scheme. If, however, the packet is a fragment, then DPM determines if it is the first fragment in the series that it sees. If it is the first one, then the process is identical to a non-fragment case, but, in addition, DPM stores the value to which it assigned the RF. This would allow to set the ID field and reserved flag of all the remaining fragments in this series to the same values as the first fragment. Reconstruction procedure at the victim will not change and will be identical to the reconstruction procedure of the basic DPM.

A. Expected number of packets required for reconstruction

The attacker can send even infinitely many packets with the same ID field, thus making DPM believe that they belong to the same series. This is called an *infinite series*. The invalid traffic would be noticed only by the destination at the reassembly, but for (D)DoS attacks it would be enough that invalid packets occupy the resources of the victim. In this situation, the victim will never recover the full ingress address.

To remedy this situation, another simple modification in addition to fragment persistence must be introduced. The modification is based on the findings in [2], where it was determined from the real traffic traces that the longest series on the Internet is 44 fragments. DPM should recognize the fact that if the number of fragments in the series exceeds 44, it is, in all likelihood, an attack, or a result of some errors. In either case, such traffic is not expected to be properly reassembled. So, after DPM has persistently marked 44 fragments of a single series with the same 16 bits, any following fragments from the same series will be marked randomly by either the first or the second 16 bits of the address of the DPM-enabled interface.

In order to implement this modification, the table, which DPM has to keep for fragments, where the value of RF

corresponding to (SA, DA, P, ID) is kept should also keep a counter, which should be incremented every time a fragment with a given tuple is encountered. Once this counter exceeds 44, marking persistence should be suspended and randomness is reinstated. Figure 5 illustrates this concept with a pseudo code.

Marking procedure at router R, edge interface A:
for y = 0 **to** 1
$Marks[y]$.Seg_Num := y
$Marks[y]$.A_bits := $A[y]$
for each incoming packet w
let x be a random integer from [0,1]
if w.MF == '1' OR w.offset $\neq 0$ **then**
if $FragTbl[SA, DA, P, ID] == NIL$ **then**
create $FragTbl[SA, DA, P, ID]$
$FragTbl[SA, DA, P, ID]$.Mark_num := x
$FragTbl[SA, DA, P, ID]$.counter := 1
else
if $FragTbl[SA, DA, P, ID]$.counter < 45 **then**
x := $FragTbl[SA, DA, P, ID]$.Mark_num
$FragTbl[SA, DA, P, ID]$.counter++
write $Marks[x]$ into w.Mark

Fig. 5. Pseudo code for the fragment-persistent DPM with fragment counter.

With this modification, in the worst case, if the attacker will be sending the offending traffic in series of 44 fragments each, the DPM will be fragment-persistent, and the victim will get one part of the address in 44 packets. According to the solution to the Coupon Collector Problem, the expected number of datagrams required to reconstruct the address, $E[D]$ is given by $k(ln(k) + 0.577)$, where k is the number of segments that the address is split into. Since $k = 2$, $E[D] \approx 2.54$. Rounding this number up to the nearest integer yields 3. Consequently, the expected number of packets, $E[P]$, to obtain the whole ingress address will be 89. This number is derived from the fact that all of the fragments of the first two datagrams would have to be collected, and a single packet from the third datagram in order for the destination to be able to assemble the whole ingress address.

VII. Fragment Interlacing

Interlacing is a result of *out of order arrivals*, which occurs in the Internet. It would also be a sole cause of reassembly errors for the DPM modifications introduced in Section VI. In this section, we define *interlacing* and analyze the probability of its occurrence.

We first define an *out of order* arrival as an arrival of a packet to the destination, ahead of another packet from the same source to the same destination, which was sent earlier. In [7], the rate of out of order arrivals was measured by TCP sequence numbers, and thus is valid only for TCP traffic. However, since reordering in any protocol is a result of IP reordering, we conclude that reordering of packets for other protocols (i.e., UDP, ICMP, etc.) will have the same rate as the TCP traffic, namely 0.3% according to [7]. *Interlacing* is defined by an out of order arrival, when a fragment of a series, which originated at a later time, arrives before one or more fragments of another series, which originated at an earlier time, and the two series have the same tuple (SA, DA, P, ID). We are only concerned with the datagrams going between the same pair of hosts and with the same protocol, that undergo fragmentation. While ordinarily, the ID field would distinguish between fragments of different series, DPM replaces the ID field, and may eventually cause reassembly errors for those series.

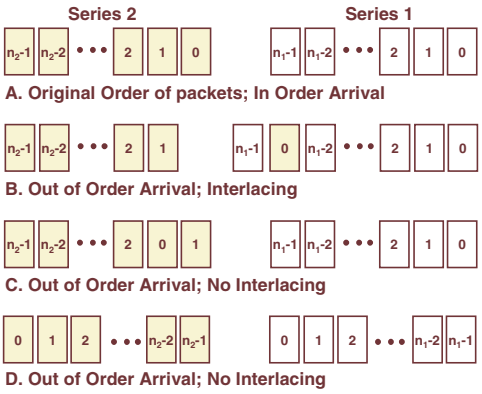

Fig. 6. Fragment interlacing.

Clearly not every out of order packet arrival, even fragments, will cause interlacing. Refer to Figure 6.A, which illustrates the case when fragments of two series arrive without any reordering to the destination. This is an ideal situation, and will not cause any reassembly errors for the downstream fragmentation. Figure 6.B illustrates the situation when a single out of order arrival occurs. If these two fragment series were traveling from the same source to the same destination, the protocol for both series was the same, and the ID fields were the same, the reassembly at the destination would fail. Let us examine closely what would happen at the destination. When fragment 0 of Series 1 arrives to the destination, the destination allocates the memory for the maximum possible IP packet, which is 64KByte, since at the receipt of the first fragment, the destination does not know how large the original datagram is. Next, fragment 1, 2, and all the fragments including $n_1 - 2$ arrive. Next, fragment 0, of Series 2, with the same ID field arrives to the destination. Destination will treat this packet as a retransmission, and replace the original fragment 0 of Series 1. Assuming that the size of fragment 0 from Series 1 and fragment 0 from Series 2 are the same, the destination will expect to complete the series to finish reassembly. When fragment $n_1 - 1$ arrives, the destination will reassemble the whole original datagram of Series 1, but with fragment 0 coming from the wrong series. IP reassembly function will not notice this error and will pass this corrupted datagram to the protocols up the stack. Depending on the protocols involved, this error will be noticed on some higher layer, and appropriate actions will be taken.

When reassembly finishes, the memory allocated is released. Next, fragment 1 of Series 2 will arrive. The destination will go through the process of allocating memory for the datagram, and will wait for all the fragments to arrive to complete reassembly. Since fragment 0 has already arrived to the destination and was processed as a part of Series 1, the destination never receives it again. After some time, 60 to 120 seconds as specified in [8], the destination will "timeout" and release the resources allocated for fragments of Series 2. Thus, a single out of order arrival, with interlacing, causes errors in two datagrams. It is necessary to point out that the reassembly function may be implemented differently on different operating systems. There is only a limited set of rules, which are specified in [8] and recommended in [6]. However, in general, a single interlacing of fragments, which are parts of the series with the same ID, will cause errors in both original IP datagrams.

Figure 6.C shows a case of out of order arrival when no interlacing happens. Such reordering will not cause any errors at reassembly. Finally, Figure 6.D shows a special case of reordering, called *reverse* ordering. Some routers, when fragmenting an original datagram, reverse the order of fragments when sending them to the destination. Such reordering does not cause interlacing unless packets arrive out of order in addition to the reverse ordering.

In order to analyze the effects of interlacing, we make a simplified assumption that out of order arrivals can only affect two series. We also assume that for a pair of affected series only one fragment arrives out of order, and causes errors in both series. Making this assumption ensures that every interlacing causes errors.

Consider two series of packets with the same ID, sent back to back. The first series has n_1 packets and the second series has n_2 packets. If a position of any fragment changes with respect to other fragments, out of order arrival has occurred. A fragment from any series can arrive out of order in $n_1 + n_2 - 1$ positions. If a fragment from the first series arrives out of order, then there are n_2 possible positions to cause a reassembly error, and if there is a single reordering, the probability that the packet from Series 1 arrived out of order is $\frac{n_1}{n_1+n_2}$. Similarly, a packet from the second series would cause a reassembly error in n_1 cases, and the probability of fragment from the second series arriving out of order would be $\frac{n_2}{n_1+n_2}$. Let event *I* denote "Interlacing", and let event *POO* denote "Packet Out of Order". Therefore the probability of Interlacing given an out of order packet is:

$$P[I|POO] = \frac{n_2 \times \frac{n_1}{n_1+n_2}}{n_1+n_2-1} + \frac{n_1 \times \frac{n_2}{n_1+n_2}}{n_1+n_2-1} = \frac{2n_1n_2}{(n_1+n_2)(n_1+n_2-1)}.$$

$P[I|POO]$ will be the highest when $n_1 \approx n_2$, reaching its highest value of 0.6 when n_1 and n_2 have the value of either 2 or 3. According to statistics in [2], series comprised of two and three fragments constitute more than 99% of all series, so we can assume for practical purpose, the series consists of two packets.

$$P[I|POO] = \frac{P[I \cap POO]}{P[POO]}$$

Interlacing of fragments can only occur as a result of out of order arrivals. In other words, out of order arrivals are the only cause for the reassembly error. Therefore, $P[I \cap POO] = P[I]$.

$$P[I|POO] = \frac{P[I]}{P[POO]}$$

$$P[I] = P[I|POO] \times P[POO]$$

$P[POO]$ is 0.3% according to [7]; therefore, if the largest value of $P[I|POO] = 0.6$ is considered, the probability of interlacing of fragments $P[I]$ is 0.18%, given the fragmented traffic. We must keep in mind that every instance of fragment interlacing will affect not one but two packets.

Introducing marking persistence for fragments will result in all fragments of a datagram having the same ID field. The fact that the ID field was not set by the host is irrelevant for the reassembly. For this modification, the upstream and the downstream fragmentation would have exactly the same effect on the reassembly error rate. The **only** condition which would result in reassembly error would be fragment interlacing discussed in this section. It was derived that in the worst case, under the least favorable circumstances, the probability of interlacing for the fragmented traffic is 0.18%. Multiplying this probability by the probability of fragmentation, which is 0.5% according to [2], results in probability of interlacing caused by DPM for all traffic be 0.0009%. Also, taking into consideration that two packets are affected by each instance of interlacing, the probability of reassembly error due to DPM is 0.0018%.

VIII. Conclusions

In this article we have presented the modification to the basic DPM scheme described in [3]. While this modification does not totally eliminate reassembly errors due to fragmentation, it drastically decreases the probability of the reassembly error caused by DPM. The modification is easy to implement, and requires little additional processing and memory.

References

[1] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network support for IP traceback," *IEEE/ACM Trans. Networking*, vol. 9, no. 3, pp. 226–237, June 2001.

[2] C. Shannon, D. Moore, and K. C. Claffy, "Beyond folklore: observations on fragmented traffic," *IEEE/ACM Trans. Networking*, vol. 10, no. 6, pp. 709–720, Dec. 2002.

[3] A. Belenky and N. Ansari, "IP traceback with deterministic packet marking," *IEEE Commun. Lett.*, vol. 7, no. 4, pp. 162–164, Apr. 2003.

[4] J. Mogul and S. Deering, "Path MTU discovery," RFC 1191, Nov. 1990.

[5] J. Postel, "Internet protocol," RFC 791, Sept. 1981.

[6] D. D. Clark, "IP datagram reassembly algorithms," RFC 815, July 1982.

[7] Y. Zhang, V. Paxson, and S. Shenker, "The stationarity of internet path properties: Routing, loss and througput," Aciri technical report, AT&T Center for Internet Research, May 2000.

[8] R. Braden, "Requirements for internet hosts – communication layers," RFC 1122, Oct. 1989.