Defending Against Slave And Reflector Attacks With Deterministic Edge Router Marking (DERM)

Shravan K Rayanchu
Samsung India Software Operations,
Bangalore 560052, India
Email: shravan_kr@yahoo.com

Gautam Barua
Dept. of CSE, IIT Guwahati
Guwahati 781039, India
Email: gb@iitg.ernet.in

***Abstract*— Identifying the sources of a Distributed Denial-of-Service (DDoS) attack is among the hardest problems in the Internet security area. Use of reflectors in a DDoS attack makes the problem particularly difficult as the actual sources of attack (slaves) are camouflaged. In this paper we propose a modification to the traceback of Multiple Hash DERM [13], an effective deterrent against DDoS attacks. The proposed modification - DERM Traceback Module, helps the victim not only in per-packet filtering of the attack traffic but also in identifying the nature of the attack and tracking the actual slaves involved in reflector attacks and mixed DDoS attacks. As in [13], the marking procedure at routers is simple, bandwidth overhead is nil, processing requirements at the victim are minimal, and a limited number of packets are required to carry out the traceback procedure.**

I. Introduction

Distributed Denial of Service attacks (DDoS) attacks are among one of the hardest security problems to address because they are simple to implement, hard to prevent, and difficult to trace. In such attacks, a victim is flooded with packets from a large number of sources, overloading its network, and thus denying effective use of the victim's resources. Ideally, the network traffic of an attack should include information identifying the sources. The Internet protocol (IP) specifies a header field in all packets that contains the source IP address, which would seem to allow for identifying every packet's origin. However, the lack of security features in TCP/IP specifications facilitates IP spoofing - the manipulation and falsification of the source address in the header. IP routing is stateless and is largely based on destination addresses; no entity is responsible for ensuring that source addresses are correct. Thus, an attacker could generate offending IP packets that appear to have originated from almost anywhere. IP traceback methods provide the victim with the ability to identify the address of the true source of the packets causing a DDoS attack. A perfect solution to this problem is complicated because of the use of zombies and reflectors [12]. The exact origin of an attack may never be revealed as even the MAC source addresses may be spoofed. Hence, the traceback schemes try to solve the more limited problem of *identifying the closest router(s) to the attacker(s)*.

Ingress filtering prevents packets with values in the source address field outside a specified range from entering the Internet. It can be employed to completely defeat spoofing, but as discussed in [14] it is associated with high overhead and administrative burden. While it might be impossible to prevent attackers from attempting an attack, it might be possible to lessen or eliminate the impact of an attack on victims. When prevention fails, a mechanism to identify the sources of the attack is needed at least to ensure the accountability for these attacks. This is the motivation for designing IP Traceback schemes. Related work on traceback and on reflectors can be found in [7], [1], [4], [9], [11], and [3].

II. Evaluation Metrics and Assumptions for Traceback Schemes

We now present some of the important evaluation metrics essential in comparing IP Traceback approaches. These were originally proposed in [2].

- ISP Involvement: An ideal traceback scheme must be inserted with little infrastructure and operational changes and the actual traceback process must involve little or no burden on the ISP.
- Number of attack packets needed for traceback: Once the attack has been identified, the traceback scheme should require very few packets to identify the attacker.
- Post-mortem Analysis: It must be possible to initiate the traceback procedure and identify the attacker after the attack as the victim might not be in a position to perform the analysis during the attack.
- Processing, Bandwidth and Memory requirements: Processing and memory overhead on routers must be minimal for the practical deployment of the scheme. Since bandwidth is one of the bottlenecks during flooding attacks, the scheme must not introduce additional bandwidth overhead.
- Ease of evasion: It must be very difficult for an attacker who is aware of the scheme to orchestrate an attack that is untraceable.
- Ability to handle major DDoS attacks: This reflects how well a scheme can perform traceback under severe circumstances. An ideal scheme should be able to identify all the attackers involved.
- Scalability: An ideal scheme should be easily scalable.

Assumptions mentioned here are largely borrowed from the previous schemes ([14], [1]). The following are the basic assumptions for DERM:

1) An attacker may generate any packet, 2) Attackers may be aware that they are being traced, 3) Packets may be lost or reordered, 4) An attack may consist of just a few packets, 5) Packets of an attack may take different routes, 6) Routers are both CPU and memory limited, 7) Routers are not compromised, 8) the IP address of every Ingress router of the Internet is available at every detection site.

III. Deterministic Edge Router Marking (DERM)

In [13] we had introduced Multiple Hash Deterministic Edge Router Marking (Multiple Hash DERM) , an effective packet marking technique for defence against DDoS attacks. Every router to which a computer system can connect is termed as an Ingress router. It is assumed that all Ingress routers, and only Ingress routers, participate in the marking of packets. Further, that all such routers are trustworthy, and that every victim has the IP address of all the possible Ingress routers. The ID field in an IP packet is used to identify fragments. This 16 bit field can be used to store information of the source of a packet, assuming we can tackle the fragmentation problem somehow. Since a 16 bit field cannot hold a 32 bit IP address, only partial information can be put. The scheme is to insert a hash of the IP address of the edge router, much like a message digest. In order to improve the scheme, multiple hash functions are used and multiple digests are sent in different packets. This marking procedure carried out by an edge router consists of simply writing a 16 bit representation of its IP address, $(HashMark||fid) = (HM_i(IP)||i)$ into the ID field of every outgoing packet. Here, $HashMark = HM_i(IP)$ where HM_i is one of the f universally known hash functions $HM_1, .., HM_f$ and fid is the corresponding function identifier. At the victim's end, it is assumed that some intrusion detection system (IDS) is available which identifies packets as attack packets. Reconstruction of the source address by the victim is carried out in two phases: Filtering Phase and Attacker Identification Phase. To identify an attacker, we need the IP addresses of all the possible Ingress Routers. Using these, we can map a HashMark to a list of Ingress routers. By intersecting these lists across the different HM_i, we can get the IP addresses of the Ingress Routers used to launch the attacks. For the purposes of filtering the attack packets, and of identifying the attackers, the victim maintains f $RecordTbls$. These tables are used to record the $HashMarks$ of the identified attack packets. $RecordTbl_i$ consists of the tuples $< HashMark, RECV\ bit, IngressAddList >$, where $IngressAddList$ is the list of IP addresses that when hashed with HM_i give $HashMark$. The $RECV\ bits$ in all the tables are initialized to zero before the attack. During the Filtering Phase, the $HashMarks$ of the identified attack packets are noted and the corresponding $RECV\ bits$ are set. Later, these marks are used to aid the victim in filtering the attack traffic. The Attacker Identification Phase is fairly simple. All the IP addresses that belong to $IngressAddLists$ for which $RECV\ bit$ in $RecordTbl_1$ is set to 1 are noted. These form the list of potential attackers. Taking one IP address at a time, $HM_2(IP)..HM_f(IP)$ are calculated and then it is checked whether the corresponding $RECV\ bits$ in $RecordTbl_2..RecordTbl_f$ are set to 1. If so, that particular IP is added to the $AttackerList$, else it is discarded.

Analysis in [13] shows that Multiple Hash DERM can handle about 3800 attackers with less than 1% false positives (wrongly identifying a node as a source of attack). It was also found that the storage requirements on the victim side were not very high and that the expected number of packets required to carry out the traceback procedure was small. However, Multiple Hash DERM cannot identify the attackers involved in a reflector attack. We propose a modification to the traceback Multiple Hash DERM in order to handle reflector attacks. We begin by discussing the characteristics of a reflector attack in the next section.

IV. Distributed Reflector Attacks

In these attacks, the attacker (also referred to as a slave from now on; many slaves carry out an attack on behalf of the real attacker: a human) does not flood the victim directly, but uses innocent servers as reflectors to achieve its goal. A reflector may be any IP host that will return a *reply* packet when sent a *request* packet. For example, all web servers, DNS servers and routers can be reflectors since they would reply with SYN ACK or RST packets in response to SYN packets. A slave spoofs the source address of the request packets with the address of the victim and sends them to the reflectors. The reflectors then send reply packets to the victim (apparent source of the request packets). As any host with a service can be exploited, it is not difficult for a slave to find many reflectors. Thus, a slave can easily flood the victim's network by sending the request packets to a very large number of reflectors. Not only do the reflectors hide the identity of actual slaves, but can also act as *amplifying subnets* as packets sent in reply may be much more in number. For instance in HTTP, the number of packets in a GET request message is small, but those in the reply message are likely to be large. Multiple slaves can co-ordinate and launch an attack. Let N_s be the number of slaves involved in an attack. Let N_r be the number of reflectors. In general, N_r is much larger than N_s. Let P_s be the number of packets sent from a single slave and P_r be the number of packets a reflector generates. Assuming no amplification (i.e a single reply packet arises from a request packet at the reflector), we have:

$$N_s P_s = N_r P_r \text{ and } N_r >> N_s$$

$$\text{Thus, } P_r = \frac{N_s}{N_r} P_s, \quad P_r << P_s$$

We see that the number of packets sent by each reflector is very less compared to that sent by each of the slaves. For example, if $N_s = 10$, $N_r = 1000$ and $P_s = 1000$ (let each slave send a single packet to each reflector), then the total number of packets sent to the victim are $10,000$ but they come from 1000 reflectors with each sending just $P_r = 10$ packets. This is so small that the local intrusion detection mechanism at the reflector cannot detect any attack.

Identifying the reflectors involved in the attack is a trivial problem, as the reflectors do not spoof the packets. Tracing the slaves behind the reflectors is much more difficult. Traceback procedures (including DERM) which employ packet marking will not provide us with the marks of the actual slaves. In case of DERM, the victim actually gets the HashMarks of the edge routers, to which the reflectors involved in the attack are connected to. This is illustrated in figure 1.

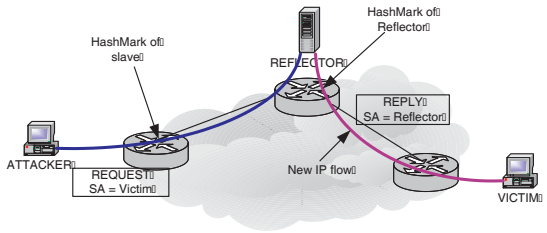

Fig. 1. Illustration of victim losing the slave HashMarks during a reflector attack

We present a traceback scheme using a DERM Traceback Module (DTM) that will help us identify even the slaves involved in a reflector attack by using the HashMarks obtained at the reflector.

Tracing the slaves using DERM Traceback Module (DTM)

Each host is required to maintain a DERM Traceback Module (DTM) which is integrated with the Intrusion Detection System (IDS) of the host. As we shall see, this module will help the victim not only in filtering the attack traffic, but also in identifying the actual attackers during slave attacks and reflector attacks. Further, as a reflector, it will help store information that will be of help to victims of reflector attacks.

V. Proactive HashMark Logging

The basic idea is to keep a log of the HashMarks received at hosts (reflectors) so that they can be made available to reflector attack victims. As discussed before, the reflector has no idea that it is being exploited and hence it cannot differentiate between legitimate packets and attack packets sent by slaves. Therefore, logging of the HashMarks must be done proactively at all hosts, i.e. irrespective of whether the packet is an attack packet. A DTM maintains a *ProactiveTbl* which contains the tuples $< SA, MarkList >$. SA is the source address of the packet received and $MarkList$ is the list of the marks that were obtained from the packets containing that particular source address. $MarkList$ is actually a list of tuples $< HashMark||fid, ArrTime >$, where $HashMark||fid$ is the identification field of the packet (i.e the HashMark and function id) and $ArrTime$ denotes the time at which the corresponding mark was received. Figure 2 illustrates this idea. Whenever the host receives a packet, DTM notes down the SA of the packet and adds the ID of the packet along with a timestamp to the corresponding $MarkList$ in the $ProactiveTbl$. Needless to say, some mechanism for purging the data in the *ProactiveTbl* is required. We do not discuss this in this paper. For a host that is going to be only a reflector, this is all that is required. However, every host can be both a victim as well as a reflector (together, or at different times). So we also need to include mechanisms to detect intrusions and to filter packets identified as attack packets. The packets are therefore then sent to the IDS after passing through the *FilterTbl* (which we explain in a short while) where attack identifying algorithms are implemented. The IDS may employ signature-based detection or anomaly-based detection as discussed in [6]. Techniques employed in identifying the attacks are beyond the scope of this work. We assume that the IDS will notify the DTM whenever it identifies an attack packet.

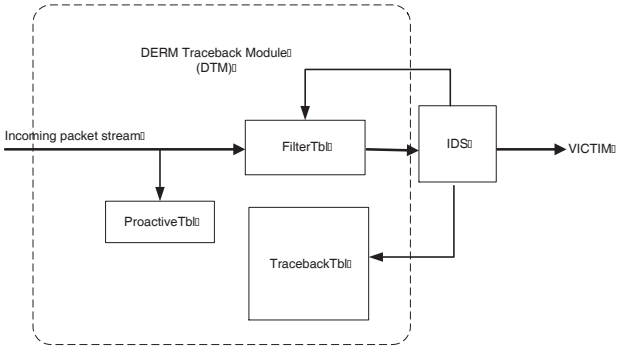

Fig. 2. DERM Traceback Module in filtering phase

Instead of using a collection of f $RecordTbls$ for filtering and traceback purposes as in Multiple Hash DERM, DTM uses two separate data structures. One is the $FilterTbl$ which is used for filtering purposes and the other is $TracebackTbl$ which is used for attacker identification. Under normal conditions, a DTM uses only the $ProactiveTbl$ for logging the HashMarks. Whenever, a DTM is informed by its IDS of an attack, it creates an instance of a $FilterTbl$ and a $TracebackTbl$.

VI. Filtering Phase of the Attack

The filtering phase consists of a mark recording procedure and the filtering procedure. We now look into the use of $FilterTbl$ and $TracebackTbl$ in these procedures.

Filtering the attack traffic using FilterTbl

DTM maintains a $FilterTbl$ which is used during the filtering phase of the attack, to filter the attack traffic. It consists of the tuples $< HashMark, RECV_1, RECV_2, .. , RECV_f >$. During the filtering phase, whenever DTM receives an attack packet from the IDS, it notes down the $HashMark$ and the function identifier (fid) from the ID field of the packet. It then sets the corresponding $RECV$ bit to 1 in the $FilterTbl$. For example, if DTM receives an attack packet with $ID = (HashMark_j||k)$, it sets the $RECV_k$ bit of $HashMark_j$ in the $FilterTbl$. This is a part of the mark recording procedure of the filtering phase. As shown in figure

2 whenever the host is under an attack, all the packets it receives first pass through the $FilterTbl$ of the DTM. The filtering procedure is very simple. It involves noting down the $HashMark$ and fid of the received packet and checking if the $RECV_{fid}$ bit corresponding to the $HashMark$ is set in the $FilterTbl$. If so, the packet is discarded, else it is sent to the IDS for further verification.

TracebackTbl in filtering phase

As shown in figure 3, $TracebackTbl$ consists of a number of $RefRecordTbls$. A $RefRecordTbl$ is identical in structure to the $FilterTbl$. It consists of the tuples $< HashMark, RECV_1, RECV_2$,.., $RECV_f >$. Each $RefRecordTbl$ is associated with a source address SA of one of the attack packets. As a part of the marking recording procedure of the filtering phase, whenever DTM receives an attack packet from the IDS, it checks if there is an instance of $RefRecordTbl$ corresponding to the SA of the packet. If so then it simply notes down the $HashMark$ and fid of the packet and sets the corresponding $RECV$ bit to 1 in that particular $RefRecordTbl$ of the $TracebackTbl$. For example, if the DTM receives an attack packet with source address SA_i, $ID = (HashMark_j||k)$, it sets the $RECV_k$ bit of $HashMark_j$ in $TracebackTbl[SA_i].RefRecordTbl$. If it does not find an instance of $RefRecordTbl$ corresponding to the SA of the packet, it creates one. Thus, the $TracebackTbl$ provides the victim with source addresses (SAs) of the attack packets and the corresponding HashMarks received from them.

VII. Types of RefRecordTbls: Information about spoofed packets

After the mark recording procedure of the filtering phase is complete, the $RefRecordTbls$ in the $TracebackTbl$ can be classified into one of the following types:

- $UnviolatedAndFull$ or $Ideal$, if for each $RECV_i$, exactly a single HashMark was received at the victim. In other words, there is exactly one $RECV$ bit set in each column of the $RefRecordTbl$. If more than one $RECV$ bit is set in a particular column, then we say that the column is $violated$. A $RefRecordTbl$ is said to be $violated$ if it has at least one column that is $violated$. Number of violations ($NumViolations$) is the number of columns that are $violated$. A $RefRecordTbl$ is $Full$ if each column has at least one $RECV$ bit set. An $Ideal$ $RefRecordTbl$ is $Full$ and has $NumViolations$ equal to zero.
- $ViolatedAndFull$, if more than one $RECV$ bit is set in at least one column of the $RefRecordTbl$ and at least one $RECV$ bit is set in every column. That is, $RefRecordTbl$ is $Full$ and has $NumViolations \geq 1$.
- $UnviolatedAndPartial$, if the $RefRecordTbl$ is not $Full$ and has $NumViolations$ equal to zero.
- $ViolatedAndPartial$, if the $RefRecordTbl$ is not $Full$ and has $NumViolations \geq 1$.

Under normal conditions, that is if the packets received at the victim are not spoofed, then all the $RefRecordTbls$ would have to be $Ideal$. This is because, the packets coming from a particular source address are always marked by one of the f HashMarks of the DERM enabled ingress edge router. We also assume that if a $RefRecordTbl$ is $Ideal$, then all the HashMarks in it belong to a particular host. This is based on an analysis which shows that if multiple slaves are involved in spoofing an IP address, they can together send very few packets before the probability that the marks received are different for the same hash function, becomes very high. It is to be remembered that the marks are obtained by a hash of the IP address of the ingress routers, which are different for the different slaves. The analysis is not shown here due to space constraints but it shows that even if only 35 packets are sent by all the slaves together the probability of receiving different marks is nearly one. However, if a $RefRecordTbl$ is $Ideal$, the victim cannot infer that the packets corresponding to the HashMarks in it were not spoofed. It may so happen that all the HashMarks came from a single slave which has actually spoofed a particular source address SA of a particular host, and during the attack no packets were received from that particular host (i.e. the host with source address SA). But if a $RefRecordTbl$ is $violated$, then the victim can infer that there has been definite spoofing and that some of the HashMarks in the $RefRecordTbl$ are definitely the HashMarks of the slaves. More precisely, at least one of the HashMarks in every $violated$ column is that of a slave.

VIII. Log Request Procedure

Once the victim is in a position to perform the traceback, it initiates the log request procedure. The victim cannot be sure if it is under a pure slave attack, pure reflector attack or a mixed attack. If it is a pure slave attack then the HashMarks in the $RefRecordTbls$ of the $TracebackTbl$ are actually the HashMarks of the slaves. In case of a pure reflector attack, the obtained HashMarks are that of the reflectors and in a mixed attack, the $RefRecordTbls$ contain the HashMarks of reflectors as well as slaves. Hence, DTM treats each source address SA in the $TracebackTbl$ as a potential reflector and sends a request for the log of HashMarks to each and every SA in the $TracebackTbl$. The log request contains the victim's source address V, the $start\ time$ and $end\ time$ of the attack. It must be noted that, during a reflector attack, the slave spoofs the source address of the victim V and sends the packets to the reflector. The replies to these packets are then directed to the victim. Even though a slave may spoof the SA of the packet, the HashMarks contained in the packets received at the reflector would be that of the ingress edge router of the slave's network. Therefore, the HashMarks in the $ProactiveTbl$ of the reflector corresponding to the source address of victim V would actually be the HashMarks of the slave. When a reflector receives a log request, it simply checks the $ProactiveTbl$ to see if there are any HashMarks corresponding to the source address V, with $startTime \leq ArrTime \leq endTime$. It then makes a $MatchList$, which is the list of all such HashMarks and sends it to the victim. The victim can then carry out the $Attacker\ Identification$

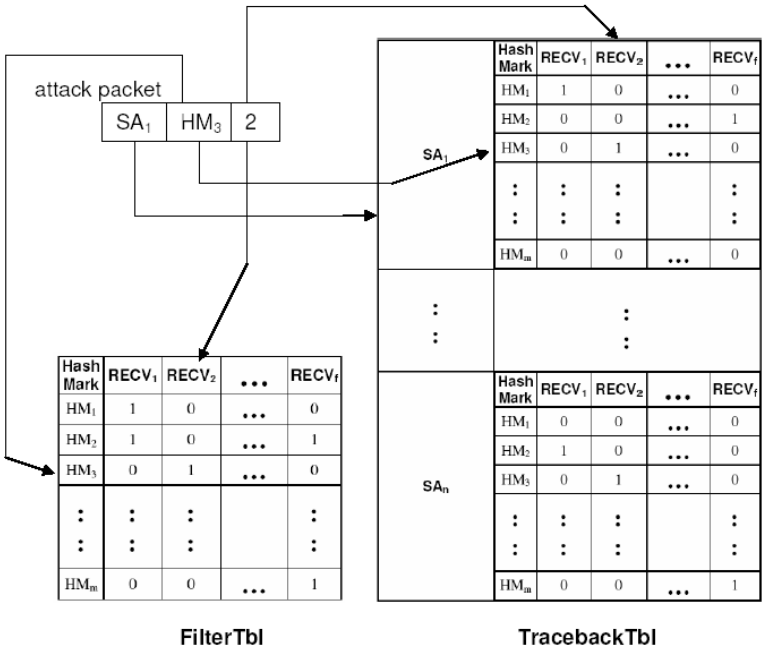

Fig. 3. Mark recording during filtering phase

procedure on the HashMarks to identify the ingress edge router of the slave. The protocol for obtaining the log of HashMarks (i.e. the $MatchList$) from a reflector is beyond the scope of this work.

Whenever the victim sends a request for logs to the potential reflectors, the responses received would be one of the following:

1) *LogFound*, if the reflector found HashMarks corresponding to the source address of the victim during the attack time. The list of these HashMarks, $MatchList$ is also obtained by the victim. 2) *LogNotFound*, if the reflector was not able to find any HashMark corresponding to the information provided in the log request. 3)*LogNotEnabled*, if the reflector does not proactively log the HashMarks. 4) *NIL*, if the victim does not receive any response or if it receives an error message.

IX. Inferring the nature of attack from TracebackTbl

In this section, we try to infer the nature of an attack by examining the $RefRecordTbl$ of a particular source address SA in the TracebackTbl and the log response received from SA. Based on the taxonomy introduced in section 5 the following cases arise:

- Case (a) If $RefRecordTbl$ is $Ideal$ and the victim receives the response: $LogFound$ along with the $MatchList$ containing a list of HashMarks. The HashMarks in the $RefRecordTbl$ belong to a single host as the $RefRecordTbl$ is $Ideal$. The HashMarks obtained in the $MatchList$ are definitely from the ingress interface of one of the slaves, since they came in the packets which had victim's source address V. Hence, this is the result of a pure reflector attack, i.e the $RefRecordTbl$ contains only the HashMarks of the single reflector and the $MatchList$ contains the HashMarks of the slaves that sent packets to this reflector. It is not possible that HashMarks in the $RefRecordTbl$ are that of a single slave and that it has sent bogus logs, for in that case the slave should not spoof the source address in the first place.
- Case (b) If the $RefRecordTbl$ is $Ideal$ and the victim receives the response: $LogNotFound$ then it can be inferred that it is the case of a pure slave attack i.e the HashMarks in the $RefRecordTbl$ belong to a single slave and that it had spoofed the source address of a host SA. This is because the $RefRecordTbl$ is $Ideal$ and no HashMarks were found at SA corresponding to the victim's source address.
- Case (c) If the $RefRecordTbl$ is $Ideal$ and the victim receives no response (or error), then this would most probably be a case of pure slave attack i.e a single slave has spoofed a source address SA randomly and there might actually be no host with source address SA.
- Case (d) If the $RefRecordTbl$ is $Ideal$ but the victim receives the response: $LogNotEnabled$, then it can only infer that all the HashMarks came from a single host. The HashMarks might be that of a reflector, which had no logging enabled or they might be that of a slave, which has spoofed the source address of a host which had no logging enabled.
- Case (e) If the $RefRecordTbl$ is $violated$ and the victim receives the response: $LogFound$ along with the $MatchList$, then it is the case of a mixed attack i.e the

HashMarks in the $RefRecordTbl$ are that of a reflector with source address SA and that of slave(s) which has spoofed the source address of this reflector.

- Case (f) If the $RefRecordTbl$ is $violated$ and the victim receives the response: $LogNotFound$, then this is definitely a case of pure slave attack. Since the $RefRecordTbl$ is $violated$, we can infer that the HashMarks belong to multiple slaves.
- Case (g) The $RefRecordTbl$ is $violated$ and the victim receives no response. This is similar to case (c) except that the HashMarks belong to multiple slaves.
- Case (h) If the $RefRecordTbl$ is $violated$ and the victim receives the response: $LogNotEnabled$, then the victim can only infer that some of the HashMarks are that of slaves. It might be a mixed attack, i.e. the HashMarks are that of a reflector SA and a slave which spoofed the source address SA or it might be the case of pure slave attack i.e the HashMarks might belong to multiple slaves.
- Case (i) The $RefRecordTbl$ is $UnViolatedAndPartial$ and the victim receives the response: $LogFound$. This might be because the reflector received a small number of packets from the slave(s). It is also possible that some of the HashMarks may belong to the slave(s).
- Case (j) The $RefRecordTbl$ is $UnViolatedAndPartial$ and the victim receives the response: $LogNotFound$. The HashMarks in $RefRecordTbl$ definitely belong to slave(s). This is the case where the slave(s) have sent very few packets with this SA.
- Case (k) The $RefRecordTbl$ is $UnViolated AndPartial$ and the victim receives no response. Then the HashMarks might belong to single or multiple slaves.
- Case (l) The $RefRecordTbl$ is $UnViolatedAndPartial$ and the victim receives the response: $LogNotEnabled$. The Hashmarks may belong to a reflector or they might belong to single or multiple slaves.

Let us now examine the marks in the $MatchList$. The victim can maintain a table, similar to $RefRecordTbl$ in order to analyze the HashMarks obtained from the reflectors. If the table happens to be $Ideal$, then the victim can infer that the reflector has received packets from only a single slave. We can simply carry out the $Attacker\ Identification$ procedure (as in Multiple Hash DERM) and add the identified slave to the $AttackerList$. In case the table is $violated$, then the reflector has received packets from multiple slaves. Instead of using a different table for each source address (like $RefRecordTbls$) it would be better if we copy the HashMarks from all the $violated$ slave tables to a single table. To understand this, recall that $N_s << N_r$ and that HashMarks got from each slave at a particular reflector are very few: that is, the tables would generally be $ViolatedAndPartial$. The slaves spread their HashMarks across all the $MatchLists$. Moreover, the HashMarks from a particular $MatchList$ are generally from multiple slaves. Hence it would be better if the victim copies all the marks of $violated$ tables to a single table for the purpose of $Attacker\ Identification$.

X. The Traceback procedure

Based on the above analysis, the victim carries out the following traceback procedure. For case (a), the victim simply deletes the $RefRecordTbl$ as the HashMarks are that of a reflector. For the cases (b) and (c), the victim carries out the $Attacker\ Identification$ procedure on the HashMarks (briefly explained in section II). For this purpose DTM uses a table, $SlvRecordTbl$. Figure 4 shows the structure of the $SlvRecordTbl$. The $IngressAddList$ refers to the list of IP addresses of Ingress routers whose which to that particular HashMark. Consider case (d): the HashMarks might be those of a single slave or a reflector. There is nothing much the victim can do when there is no logging enabled, so it copies the HashMarks to the $SlvRecordTbl$ and carries out the $Attacker\ Identification$ procedure. That is, we risk an increase in false positives while identifying the attackers so as to cut down the false negatives, that is, so as to identify all the slaves. Coming to case (e), it is certain that some of the HashMarks are that of the slaves. To identify those HashMarks, DTM collects the HashMarks of the reflector. These can be sent by the reflector itself along with the $MatchList$ or the victim can acquire the HashMarks by itself in the ICMP echo replies in response to the request it sends. After the victim gets the $ReflectorMarkList$, it can $classify$ the HashMarks of the $RefRecordTbl$ into $DefiniteSlaveMarks$ and $PotentialSlaveMarks$. $DefiniteSlaveMarks$ are the HashMarks in the $RefRecordTbl$ other than those in the $ReflectorMarkList$. $PotentialSlaveMarks$ are the HashMarks in the $RefRecordTbl$ other than the $DefiniteSlaveMarks$.

The victim should not delete the $PotentialSlaveMarks$ simply because they happen to be the HashMarks of the reflector. It may be the case that the attacker has carefully chosen the slaves so that they have some HashMarks in common with the reflector. Deleting those common marks would then result in an increase in the false positives while carrying out the $Attacker\ Identification$ procedure as we would not get all the $f\ HashMarks$ of the slave. But at the same time, we should be able to differentiate the HashMarks of slaves from those of the reflectors among the $PotentialSlaveMarks$. For this purpose, the victim copies the $DefiniteSlaveMarks$ to $DefSlvTbl$ and $PotentialSlaveMarks$ to the $PotSlvTbl$. The $DefSlvTbl$ and $PotSlvTbl$ are identical in nature and consist of tuples $< HashMark, CardinalNumber >$. Figure 4 shows their structure. The $CardinalNumber$ of a HashMark is the number of $RefRecordTbls$ across which the HashMark appears. That is, it is the number of different source addresses which carried the same HashMark among the packets received at the victim. Assuming a uniform distribution of HashMarks in the interval $[0, 2^d)$ (d is the number of bits used for a HashMark, typically 12 to 14; 16-d bits are used to identify the hash function being used), the only reason why some HashMarks have a higher $CardinalNumber$ is due to the fact that the HashMarks belong to slaves which

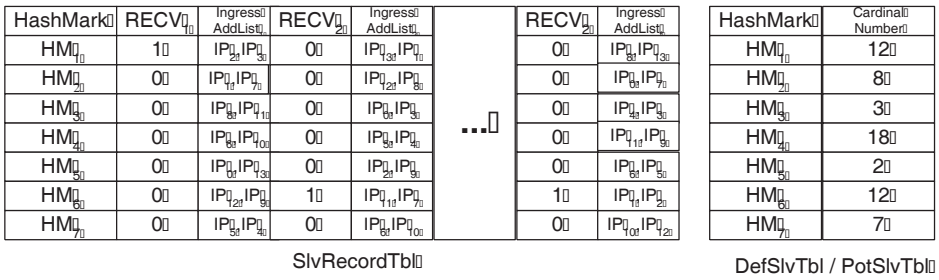

Fig. 4. Structure of $SlvRecordTbl$, $DefSlvTbl$ & $PotSlvTbl$

have spoofed the packets sent to the victim with many source addresses. Hence, among the Hashmarks in the $PotSlvTbl$ ($PotentialSlaveMarks$), the $CardinalNumber$ denotes the probability of them belonging to the slaves. Greater the $CardinalNumber$ of a HashMark, higher the chance of it being a slave HashMark.

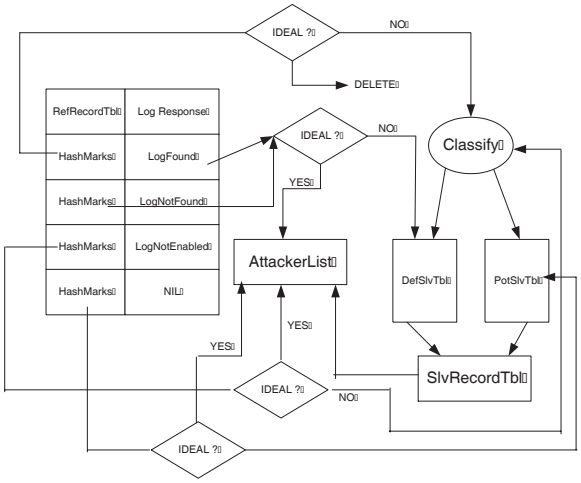

Fig. 5. The Traceback Procedure

Consider case (f). Since all the HashMarks are that of slaves, the victim simply copies all the HashMarks to the $DefSlvTbl$. For case (g), the HashMarks most probably belong to multiple slaves and hence all the HashMarks are copied to the $PotSlvTbl$. Coming to case (h), since some of the HashMarks are definitely that of slave(s), this case can be treated as similar to case (e), i.e the $ReflectorMarkList$ is collected and the HashMarks are classified into $DefiniteSlaveMarks$ and $PotentialSlaveMarks$. They are then copied to the respective tables, that is, the $CardinalNumber$ of these HashMarks is increased by one. Cases (i),(j),(k),(l) are treated similarly to the cases (e),(f),(g),(h) respectively. For cases (a), (e) the HashMarks that are obtained from the $MatchLists$ of the reflectors are first copied to the $SlvRecordTbl$. If the $SlvRecordTbl$ is $Ideal$, then the $Attacker\ Identification$ procedure is carried out and the identified slave is added to the $AttackerList$. If the $SlvRecordTbl$ is $violated$, then the HashMarks belong to multiple slaves. Therefore, they are copied to the $DefSlvTbl$ ($CardinalNumber$ is increased). After dealing with all the $RefRecordTbls$ of the TracebackTbl, the victim is left with some HashMarks that definitely belong to the slaves in $DefSlvTbl$ and some HashMarks that might belong to slaves in $PotSlvTbl$. In order to carry out the $Attacker\ Identification$ procedure, all these HashMarks must be copied to the $SlvRecordTbl$. However, we would like to keep the false positives during the $Attacker\ Identification$ to be less than 1% of N, the total number of attackers. Thus, we would like to have the HM_{MAX} number of HashMarks where HM_{MAX} is the expected number of different HashMarks the victim would collect if attacked by N_{MAX} attackers.

$$HM_{MAX} = f * 2^d \left(1 - \left(1 - \frac{1}{2^d}\right)^{N_{MAX}}\right)$$

In [13] it was found that for $d = 12$ and $f = 16$, we get $N_{MAX} = 3800$. For these values of d,f and N_{MAX}, we get $HM_{MAX} = 39,623$.

Only HM_{MAX} number of HashMarks must be copied to the $SlvRecordTbl$ in order to keep the false positives below 1%. For this purpose, the $DefSlvTbl$ and $PotSlvTbl$ are sorted by the $CardinalNumber$. Then, until the number of HashMarks in $SlvRecordTbl$ reach HM_{MAX}, the HashMarks from $DefSlvTbl$ and then those from $PotSlvTbl$ are copied in the order of decreasing $CardinalNumber$. Finally, the $Attacker\ Identification$ procedure is carried out on the $SlvRecordTbl$ and the identified attackers are added to the $AttackerList$. Figure 5 illustrates the traceback procedure.

XI. Discussion on how DTM tackles slave and reflector attacks

A. Pure Slave Attacks

In this kind of attack, all the slaves flood the victim directly. In order to hide their identity, the slaves resort to random spoofing. Thus, the $RefRecordTbls$ are mostly $violated$. When the log request procedure is carried out, then the responses may be $LogNotFound$, $LogNotEnabled$ or NIL. In all three cases if the $RefRecordTbl$ is $Ideal$, then DTM carries out the $Attacker\ Identification$ procedure and the slave is identified. If the $RefRecordTbl$ is

$violated$ or $UnViolatedAndPartial$, then the HashMarks are $classified$. At first, HashMarks of the slaves that spoof many SAs will be copied into the $SlvRecordTbl$ and then in the end will be copied the HashMarks of the slaves which have spoofed only a single address. In any case, all the slaves are identified and the number of false positives remain less than 1% of N_s if $N_s \leq 3800$. If $N_s \geq 3800$ we may not identify the slaves which have not spoofed across many source addresses, however we may increase HM_{MAX} and thus identify all the slaves involved in the attack at the risk of increasing the false positives.

The minimum number of packets each slave should send so that the victim gets all the f HashMarks is given by

$$E(f) = f(\frac{1}{f} + \frac{1}{f-1} + \frac{1}{f-2} + ... + 1)$$

or

$$E(f) = f(\log(f) + 0.577)$$

For f=16, this is about 720, a relatively small number.

B. Pure Reflector Attacks

Here, all the slaves send packets to the reflectors with the source address of the victim. Thus, in pure reflector attacks, the $RefRecordTbls$ are all $Ideal$. The response to log request procedure may be $LogFound$ or $LogNotEnabled$. In the first case, if the $SlvRecordTbl$ is $Ideal$, then this reflector has received the packets only from a single slave and it is readily identified. If the $SlvRecordTbl$ is $violated$, then the HashMarks are $classified$. In the second case if the $RefRecordTbl$ is $Ideal$, then the reflector is incorrectly identified as a slave else the HashMarks are $classified$. Thus, the efficacy of the scheme depends on the number of reflectors which have logging enabled.

Let L be the factor of reflectors which have logging enabled. If P_s is the flood from each slave, then the average number of packets each reflector receives is P_s/N_r

The effective number of total packets (HashMarks) the victim receives is

$$\left(\frac{P_s}{N_r}\right) LN_r = P_s.L$$

Thus, The minimum number of packets each slave should send so that the victim gets all the f HashMarks is given by

$$P_s * L = f(\log(f) + 0.577)$$

or

$$P_s = f(\log(f) + 0.577)/L$$

If half the reflectors have logging enabled and the number of hash functions is 16, then about 1500 packets need to be sent by each slave. This is also a reasonably small number.

XII. Related work and Comparison

There is very little work in the literature to handle attacks that use reflectors. A brief description of work in DDoS based on assumptions similar to ours, is given below.

A. The Pushback Protocol

The main idea behind the pushback protocol [10] is the fact that if routers can detect the packets belonging to an attack, they can then drop only those packets and thus the DDoS problem would be solved. However, routers cannot tell with a total certainty whether a packet belongs to a *good* or a *bad* flow; Thus the goal is to develop heuristics that help in identifying most of the bad packets, without interfering with the good ones. Bad traffic is characterized by an *attack signature* which we strive to identify; what can be really identified is the *congestion signature*, which is the set of properties of a subset of traffic identified as causing problems. The idea is to provide the routers with a congestion signature so as to drop the attack traffic before it consumes the bandwidth all the way upto the victim. *Poor* traffic consists of packets that match the congestion signature but are not really a part of the attack. The amount of poor traffic depends on the congestion signature employed. The authors in [8] use the destination address (victim's address) as the congestion signature. Thus even legitimate traffic destined for the victim is automatically dropped. One of the advantages of DERM is that it can be used as an effective congestion signature in the pushback protocol.

B. Packet Marking Schemes

In these schemes, basically some traceback data is inserted in each packet so that a victim can use this information to identify the attacker. To be effective, packet marking should not increase a packets' size. Furthermore, packet-marking methodologies must be secure enough to prevent attackers from generating false markings. We will now look into some existing schemes that fall under this category.

Probabilistic Packet Marking (PPM)

In this scheme [14] the routers probabilistically decide whether or not to mark packets. As with most of the packet marking schemes, the 16 bit ID field in the IP header is used for this purpose. The routers mark a packet with the partial address information about the edges in the attack path. A victim reconstructs the attack path with these marked packets. The main assumption in PPM is that if the routers mark the packet with a small probability, then it would receive sufficient packets during the attack that would contain the markings from all edges. There are many advantages of DERM over PPM. The number of packets required in DERM in order to identify the attacker is much less ($E(f)$ packets) as compared to PPM which requires a large number of packets. In PPM, the victim can reconstruct the path to the attacker based on multiple packet markings, but there is no guarantee that an individual packet will contain a marking that can identify an attacker. Thus, if the victim receives a packet with a marking sufficiently far away from the attacker on the attack tree, then that packet may be from a non-attack node that happens to intersect the attack path at that point. Hence, per-packet & victim filtering cannot be done in PPM. One of the major disadvantages of PPM is *Mark Spoofing*. If

an attacker injects a packet with erroneous information and no router on the path marks the packet then the spoofed marks from the attacker would also reach the victim. This completely confuses a victim during a DDoS attack and is the main reason why PPM is not scalable to DDoS attacks. There is no possibility of mark spoofing in DERM as each and every packet is deterministically marked. Fragment reassembly in PPM becomes particularly difficult during DDoS attacks wherein a large number of permutations have to be tried out. A study indicated that PPM would not be able to resolve attack paths if there are more than 10 attackers at the same distance from the victim. Though DERM does not give us the information about the entire attack path, tools like traceroute can be used to find the paths once the IP address of the router closest to the attacker is identified.

Deterministic Packet Marking (DPM)

In DPM [1], only the edge routers participate in the marking procedure. As in DERM, interfaces are used as atomic units of traceback. DPM tries to construct the ingress address of the router closest to the source by fragmenting the IP address and sending it in two packets. The ID field is used to carry one half of an IP address and the RF bit is used to denote whether it is the first half or the second half of the IP address. In the initial proposal, the source address in the IP packet was used to construct the IP address of the router. This has a very serious consequence when the attacker uses many randomly generated source addresses. Thus the scheme was modified [3] by also sending a hash of the IP address along with the fragmented IP address to aid the victim in the reassembly procedure. Since a hash is also sent along, the number of fragments now increase. Constructing an IP address from fragments would require trying out all the possible permutations. Not only would this require much processing overhead, but would also result in a high number of false positives while assembling the fragments, especially in the case of a DDoS attack where multiple fragments from multiple attackers are collected. This problem is avoided by DERM as information about the attacker is sent in a single packet. Multiple Hash functions further reduces the false positives by cross verification of HashMarks in multiple tables. But the tradeoff in DERM is the need to construct the tables, which requires collecting all the edge router IP addresses. However, as already mentioned, this needs to be done only once and then it can be distributed to every other host. One more advantage DERM offers is per-packet filtering. Although it is assumed that once the attack path is traced, upstream routers will install filters to drop the attack traffic, it might not be always possible. Hence, the victim must also be capable of filtering the traffic. In DERM, sending the information about an attacker in a single packet helps in deciding whether or not to drop a packet during an attack. However, per-packet filtering is not possible in DPM.

Pi and StackPi : Path Identification Mechanisms

In these schemes [4][5], each packet is marked deterministically by the routers along its path towards the destination. The StackPi mark created by a router is a $d\ bit$ message digest of the concatenation of the IP addresses of the previous router and the current router. The value $d = 2$ is used in the proposed scheme. Since the routers discard the oldest mark and insert the new mark at each point, only the last eight marks $(16/2 = 8$) made by the routers along the path reach the destination. In any case, the packets travelling along the same path will have the same marking so that the victim needs only to identify the StackPi marks of the attack packets and filter out all further packets with the same marking. It is important to note that every packet in the internet is individually routed; multiple paths exist. Thus, in StackPi, multiple marks arise for hosts at an arbitrary distance from the victim that has traffic routed over multiple paths. This would affect the filtering. Since, the ingress interface of the router closest to the source would be the same even if multiple paths exist, DERM would not give rise to multiple marks. In StackPi, if there are not enough marking routers on the path or if the host is close to the destination then some of the initially random bits in the marking field may not be overwritten when the packet reaches its destination. An attacker can therefore generate different markings (which would not be overwritten) and confuse the victim. This problem of legacy routers is lessened in DERM as we need only the edge routers to participate in the scheme unlike StackPi which requires all the routers to do so. Also, an attacker which is very close to the victim will not be a problem in DERM. Compatibility with the pushback protocol is one of the advantages DERM provides. The congestion signature is simple to check and thus can be implemented in upstream routers. The StackPi marks cannot be used directly as a congestion signature since the marks are changed at each and every router. Another problem is that in StackPi only 8 markings are recorded when d=2 (the value used by the authors). Since most of the paths are around 15 to 20 hops in the internet, the initial marks are lost and thus packets coming from two different hosts in a network are most likely going to have the same marks. This is same as in DERM where the edge router markings would be identical and so there is no disadvantage in using edge router markings only. Finally as the authors in StackPi state, the lower bound in their results occurs when the distribution of IP addresses over the StackPi marks is uniform, in which case the probability of an attacker spoofing becomes $1/2^{16}$. This is the case with DERM too when the hash marking function HM is assumed to be ideal and there is only one hash function. Thus, DERM has the same results as StackPi while the implementation is much easier as only the edge routers have to participate in the scheme and the marking procedure is a one-time process. As is the case with StackPi, DERM also requires constructing a table which consists of mapping the HashMarks with the IP addresses. This task is much easier in DERM as we have to collect the edge router IP addresses and simply hash them to get the corresponding

HashMarks. Also, since these marks are only dependent on the edge routers we can have the table constructed at a particular location and distribute them. But in StackPi, the marks from a particular source will be different for two different hosts. Hence, each host has to construct its own table. Moreover, in StackPi we are dealing with addresses of hosts (not edge routers) which will be much more difficult to maintain.

XIII. Conclusion

In this paper, a modification to the traceback procedure of Multiple Hash DERM has been proposed to account for reflector attacks. The basic assumptions are that Ingress routers (routers to which end nodes connect) participate in marking packets, all such routers are trustworthy, and that every victim has the IP address of all the possible Ingress routers. As in DERM, the marking procedure for the routers is simple and the processing overhead at the victim during reconstruction is also little. The victim maintains a DERM Traceback Module (DTM) which carries out the reconstruction in two phases, a filtering phase and an attacker identification phase. The filtering phase involves setting a flag in a filter table and a traceback table based on marks in identified attack packets and then using the marks in the filter table for filtering the attack traffic. In the attacker identification phase, DTM follows a comprehensive traceback procedure using the traceback table which helps the victim to identify the slaves even in case of reflector attacks and mixed attacks. The traceback procedure consists of identifying the marks of slaves with the help of reflector responses and copying them to the slave record table to carry out the attacker identification procedure. The attacker identification procedure consists of simply noting down the IP addresses of ingress packets and checking them against the table entries to see whether the corresponding flag bits are set to 1. As in [13], about 3800 attackers can be handled with less than 1% false positives and the expected number of packets required to identify an attacker is also small.

For further work, a secure protocol to download hashMarks from reflectors needs to be developed. An implementation needs to be done to show the feasibility of this scheme. In particular, schemes and algorithms to maintain and search the database of all Ingress routers need to be developed and implemented.

References

[1] N. Ansari A. Belenky. IP Traceback with Deterministic Packet Marking. *IEEE communication letters*, 7, April 2003.

[2] N. Ansari A. Belenky. On IP Traceback. *IEEE Communications Magazine*, 41(7):142–153, July 2003.

[3] N. Ansari A.Belenky. Tracing Multiple Attackers with Deterministic Packet Marking (DPM). In *Proceedings of the IEEE PACRIM '03, Victoria, B.C., Canada*, August 2003.

[4] Adrian Perrig Abraham Yaar and Dawn Song. Pi: A path identification mechanism to defend against DDoS attacks. *In IEEE Symposium on Security and Privacy*, May 2003.

[5] A. Perrig A.Yaar and D. Song. StackPi: A New Defense Mechanism against IP Spoofing and DDoS Attacks. *Technical Report, Carnegie Mellon University, USA*, 2003.

[6] Rebecca Bace and Peter Mell. Intrusion Detection Systems. *NIST Special Publication on IDS*, SP 800-31, November 2001.

[7] Roy Chang. Defending against Flooding-based Denial-of-Service Attacks: A Tutorial. *IEEE Communications Magazine*, 40(10):42–51, October 2002.

[8] Ratul Mahajan et. al. Controlling high bandwidth aggregates in the network. *Computer Communication Review*, 32(3):6273, July 2002.

[9] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. *RFC 2267*, January 1998.

[10] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of the Symposium on NDSS 2002, San Diego, CA*, February 2002.

[11] Hideyuki Tokuda Nobuhiko Nishio, Noriyuki Harashima. Reflective Probabilistic Packet Marking Scheme for IP Traceback. *IPSJ JOURNAL*, 44(8), 2003.

[12] Vern Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *Computer Communication Review*, 31(3), 2001.

[13] Gautam Barua. Shravan K. Rayanchu. IP Traceback using Deterministic Edge Router Marking (DERM). *International Conference on Distributed Computing and Internet Technology, Bhubaneswar, India (accepted for presentation)*, December 2004.

[14] A. Karlin. S.Savage. Practical Network Support for IP Traceback. *ACM SIGCOMM*, pages 295–306, 2000.