available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Extensions to the source path isolation engine for precise and efficient log-based IP traceback

Egon Hilgenstieler [a], Elias P. Duarte, Jr. [a,*], Glenn Mansfield-Keeni [b,1], Norio Shiratori [c,2]

[a] Federal University of Paraná, Dept. Informatics, P.O. Box 19018, Curitiba 81531-980, PR Brazil
[b] Cyber Solutions Inc., ICR Bldg, 6-6-3, Minami Yoshinari, Aoba-ku, Sendai, 989-3204 Japan
[c] Tohoku University – RIEC, 2-1-1 Katahira, Aoba-ku, Sendai 980-8577, Japan

ARTICLE INFO

Article history:
Received 18 September 2009
Received in revised form
24 November 2009
Accepted 28 December 2009

Keywords:
Denial of service
Packet tracing
Traceback
Attack Response
Traffic logging

ABSTRACT

IP traceback is used to determine the source and path traversed by a packet received from the Internet. In this work we first show that the Source Path Isolation Engine (SPIE), a classical log-based IP traceback system, can return misleading attack graphs in some particular situations, which may even make it impossible to determine the real attacker. We show that by unmasking the TTL field SPIE returns a correct attack graph that precisely identifies the route traversed by a given packet allowing the correct identification of the attacker. Nevertheless, an unmasked TTL poses new challenges in order to preserve the confidentiality of the communication among the system's components. We solve this problem presenting two distributed algorithms for searching across the network overlay formed by the packet log bases. Two other extensions to SPIE are proposed that improve the efficiency of source discovery: separate logs are kept for each router interface improving the distributed search procedure; an efficient dynamic log paging strategy is employed, which is based on the actual capacity factor instead of the fixed time interval originally employed by SPIE. The system was implemented and experimental results are presented.

1. Introduction

Several attacks are reported daily in the Internet and several tools are widely available for attackers to disable network services either by exploiting software or hardware implementation bugs, or simply by flooding the network with legitimate requests. Denial of Service (DoS) attacks are frequently reported (SecurityStats.com, 2009), in which the network and/or a server is overwhelmed with heavy traffic. Other types of attacks, however, can be orchestrated using a significantly smaller amount of packets. There are attacks that can disable a network service with just a single packet (Microsoft Corporation, 2008). After an attack is detected and an attack packet is isolated, the next step is to determine its source. Unfortunately, it is not possible to reliably determine the source of a received IP packet, as the protocol does not provide authentication of the packet based on the source address field, which can be easily faked (IP Spoofing). Furthermore the Internet routing infrastructure also does not keep information about forwarded packets.

IP traceback was proposed in order to allow the discovery of the source of a packet received from an IP network (Belenky and Ansari, 2003; Gao and Ansari, 2005). Several approaches for IP traceback have been proposed, two of which have been considered the most important: Probabilistic Packet Marking (PPM), and log-based traceback. In PPM (Dean et al., 2002;

* *Corresponding author*. Tel.: +55 41 3361 3656; fax: +55 41 3361 3205.
E-mail address: elias@inf.ufpr.br (E.P. Duarte Jr.).
[1] Tel.: +81 22 303 4012; fax: +81 22 303 4015.
[2] Tel.: +81 22 217 5420; fax: +81 22 217 5426.

0167-4048/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2009.12.011

Goodrich, 2002, 2008; Savage et al., 2000; Yaar et al., 2005; Gao and Ansari, 2007; Belenky and Ansari, 2007; Wong et al., 2008), routers mark packets chosen with a predefined probability with path information, typically employing header fields that are seldom used. This approach presents several challenges, such as the complexity of path reconstruction and the convergence of the traceback operation.

In log-based traceback (Keeni, 2009; Hazeyama and Kadobayashi, 2003; Sung et al., 2008) packet logs are kept throughout the network, ideally one per segment. The SPIE architecture (Source Path Isolation Engine) (Snoeren et al., 2001, 2002) is a log-based traceback that allows the path of a packet to be traced. The logs are not kept by the routers themselves, but by a packet monitor that listens to a router interface. The set of packet monitors form an overlay network that allows the source of individual IP packets to be determined. Hybrid approaches that are at the same time log-based and employ PPM have also been presented (Cong and Sarac, 2005).

The general goal of log-based traceback is to build an *attack graph*, given an IP packet, its approximate time of receipt and its destination, which is usually called the *victim*. The attack graph consists of vertices that represent nodes (routers and hosts) that have processed the packet, and the links through which the packet was transmitted. *False positives* are the nodes of the attack graph that have not really processed the packet. False positives can occur, for example, if a router is subverted by an attacker.

Several problems arise when defining a traceback architecture. The packets can be modified during the routing process; some possible transformations are described in RFC 1812 (Baker, 1995) such as packet fragmentation, options processing, ICMP (Internet Control Message Protocol) packet processing and packet duplication. Privacy issues are also important in the project of a traceback architecture. The packet's content should be properly protected. Furthermore, as a traceback architecture possibly requires the cooperation of several autonomous system (AS) it is desirable that even the packet metadata should be protected.

In this paper we present an approach for improving the precision and efficiency of SPIE. The proposed approach allows SPIE to return an attack graph that precisely identifies the route traversed by a given packet allowing the correct identification of the attacker. We show that without the extensions SPIE can return misleading attack graphs in some particular situations, which may even make it impossible to determine the real attacker. We show that by unmasking the TTL field SPIE returns a correct attack graph that precisely identifies the route traversed by a given packet allowing the correct identification of the attacker. Nevertheless, an unmasked TTL poses new challenges in order to preserve the confidentiality of the communication among the system's components. We present two new traceback algorithms which guarantee that communication among the system's components preserves the confidentiality of the packet's information.

Two other features are proposed to improve the efficiency of the traceback process: separate logs are kept for each router interface improving the distributed search procedure, having a strong impact on the cost of the traceback operation, as the number of requests is reduced to the minimum. Finally, an efficient dynamic log paging strategy is proposed, which is based on the actual capacity factor instead of the fixed time interval originally employed by SPIE.

The traceback system was implemented and experimental results are presented. Three metrics are evaluated: the precision of the obtained attack graph, that allows the correct determination of not only the packet source but also the entire route traversed by the packet; the cost of the traceback operation, measured by the number of requests in the network, and the time-frame from which a received packet can still be traced.

The contributions of this paper can be thus summarized as:

- We show that SPIE can return misleading attack graphs in some particular situations, which may even make it impossible to determine the real attacker.
- We show a simple solution to the problem (unmasking the TTL field) which nevertheless poses new challenges in order to preserve the confidentiality of the communication among the system's components.
- Two new traceback algorithms are proposed which guarantee that communication among the system's components preserves the confidentiality of the packet's information.
- We show that keeping separate logs for each router interface reduces the number of requests to a minimum.
- A dynamic and efficient approach for paging logs to secondary memory is proposed based on the log's actual capacity factor, instead of a fixed time interval.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 gives an overview of SPIE. In section 4 we describe the situation that causes SPIE to return incorrect attack graphs; the solution to the problem is given in section 5. Section 6 is devoted to the extensions proposed to improve SPIE's efficiency. The implementation as well as experimental results are presented in section 7. Conclusions follow in section 7.

2. Related work

The earlier approaches for tracing packets in the Internet aimed at determining the route of a packet stream rather than of a single packet. In Burch (2000) some networks are systematically flooded with packets. In this way it is possible to observe variations on the received packet stream and infer the traversed route. In other early efforts, the routers provide partial information about the route traversed to the end-hosts, employing a subset of the packets of a given flow. The end host can then reconstruct the packet path after receiving a large enough amount of packets.

Those early approaches required large amounts of packets in order to infer the traversed route. Those proposals cannot be applied to discover the source of attacks conducted with a smaller amount of packets. Later, other approaches for tracing back individual IP packets were developed.

A strategy based on the Internet Control Message Protocol (ICMP) has been proposed (Wu et al., 2001), in which a new ICMP message is defined that is sent randomly by routers along a path traversed by a given packet, either to the destination or origin of that packet. The communication overhead of this approach has been considered an important issue.

Another IP traceback technique is Probabilistic Packet Marking (PPM) (Dean et al., 2002; Goodrich, 2002, 2008; Savage et al., 2000; Yaar et al., 2005; Gao and Ansari, 2007; Belenky and Ansari, 2007; Wong et al., 2008). PPM routers mark packets chosen with a predefined probability with path information, typically employing fields that are seldom used. This approach presents several challenges, such as the complexity of path reconstruction and the convergence of the traceback operation.

Log-based traceback (Keeni, 2009) was proposed to allow individual packets to be traced. SPIE (Source Path Isolation Engine) (Snoeren et al., 2001, 2002) is an architecture that employs packet logs which are kept throughout the network, ideally one per network segment. A packet monitor listens to a router interface and maintains a log of processed packets. Logs are implemented with Bloom filters. The set of packet monitors form an overlay network that allows the source of individual IP packets to be searched.

Other IP traceback approaches have been developed which are based on SPIE. In Sung et al. (2008) only a small percentage of packets is sampled and logged in a Bloom filter. In Hazeyama and Kadobayashi (2003) layer-2 information and addresses are used in order to identify the last hop: the host from which the attack was launched. Finally, (Cong and Sarac, 2005) a hybrid approach that is at the same time log-based and employs probabilistic packet marking is presented, that allows logs to be roughly reduced to a half of their SPIE counterparts. In Hilgenstieler et al. (2007) we describe precise and efficient IP traceback in the context of the architecture described in Keeni (2009).

In Matsumoto et al. (2008) the Adaptive Bloom Filters (ABF) have been presented that feature counter bits, which are particularly efficient when the network traffic is variable, leading to a collision rate that changes dynamically and unpredictably. The number of hash functions employed by an ABF changes dynamically, and the number of hash functions used to decode the membership of each key element can be used to infer its multiplicity on a given filter. An ABF requires the same amount of space of a classical Bloom filter.

Recently a novel approach for IP traceback which employs swarm intelligence was proposed (Lai et al., 2008). This approach is similar to some of the earlier traceback efforts in that it employs flow level information to identify the origin of the attack packets. Another recent approach was presented in Castelucio et al. (2009) which consists of a network overlay for traceback on the level of autonomous systems. This approach uses the Border Gateway Protocol (BGP) update-message community attribute that enables information to be passed across autonomous systems that are not necessarily involved in the overlay network. The authors argue that the proposed system can be implemented in an incremental fashion.

3. An overview of the source path isolation engine

Log-based IP packet traceback employs packet logs that are stored throughout the network, possibly one per segment. In the SPIE architecture (Snoeren et al., 2001, 2002) logs are kept for recently processed packets. As the amount of storage is limited, newer records overwrite older ones when necessary. If a traceback operation was requested for a given packet, e.g. by an IDS (Intrusion Detection System), the request is executed by running a distributed search throughout the network logs in order to discover the routers that processed the packet, and eventually its source.

In order to log packets processed by backbone routers, massive storage space is required, even for relatively slow links and for short time frames. Storing packets from several sources also involves privacy issues. SPIE uses Bloom (1970) filters to solve both these problems, and stores information obtained from a packet hash. A packet hash should uniquely identify an IP packet. The hash also preserves confidentiality when a search is executed across several autonomous domains. In order to compute the hash of a given IP packet, SPIE only uses the invariant portion of the packet plus 8 bytes from the payload. Results presented in Snoeren et al. (2001) show that 28 bytes (20 bytes from the header and 8 from the payload) are enough to identify almost all non-identical packets. Fields that change as the packet is transmitted across the network are masked prior to the hash calculation.

A Bloom filter is a data structure used to store a set of elements allowing a fast membership-test operation. SPIE's Bloom filter employs k distinct hashes for each packet using independent hash functions. The size of each hash result is n bits. An array of 2^n is initialized with zeros and all entries which correspond to the computed hash results are set to one. In other words: if hash(*packet*) $= y$, then Bloom[y] $= 1$. A given array can be employed for a limited period of time after which a new fresh array is started. The set of all arrays represents the traffic processed in a given interval of time.

In order to determine whether a given packet was processed or not, k hash functions are computed, and all the corresponding array bits are checked. If the packet was seen and processed, then *all* bits must be set to 1. If one or more bits are 0 then the packet was not processed. However, if all the bits are set, there is still a probability that the packet was *not* processed, this situation is called a *false positive*. The false positive rate can be controlled (Fan et al., 2000), depending on k and the capacity factor, defined as $c = m/n$, where m is the array length and n is the number of inserted elements. Fig. 1 shows a Bloom filter using k hash functions.

The SPIE architecture for IP Packet Traceback is shown in Fig. 2. The three basic components of this architecture execute the set of tasks involved in determining the origin and route traversed by a packet. These components are described below.

- DGA (*Data Generation Agent*) – This component computes and stores the hash value of the router's outgoing packets. The data is stored locally on the DGA for a fixed time interval whose length depends on storage space constraints. The DGA can be implemented in software or as a separate hardware element connected to the router through an auxiliary interface (Sanchez et al., 2001).
- SCAR (*SPIE Collection and Reduction Agent*) – This component is responsible for searching. It maintains information about a set of DGAs in a network region. After an application generates a traceback request for a given packet, SCAR will receive a request from the STM (described below) and will then forward the request to all DGAs within its region. The

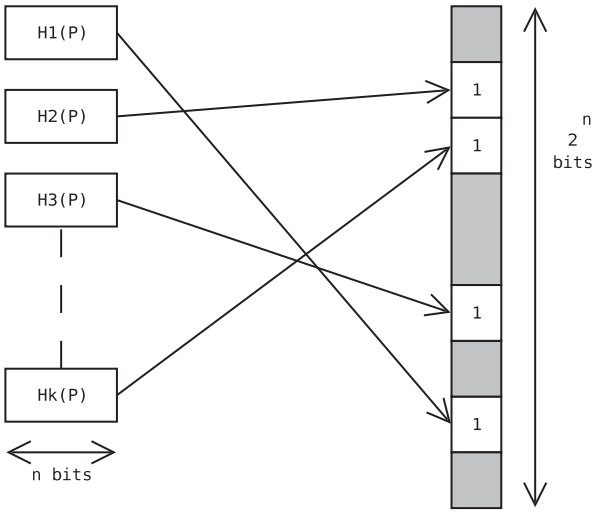

Fig. 1 – Bloom filter example.

DGAs send their filters to be evaluated by the SCAR. If the packet search succeeds, a partial attack graph is returned to the STM.

- STM (*SPIE Traceback Manager*) – This component is the front-end to the traceback mechanism and manages the system as whole. When a request is received, it is authenticated and validated and then dispatched to the selected SCARS. The STM then receives the resulting attack graphs from which it builds the complete attack graph which is returned.

The traceback procedure is described as follows. The STM receives as input the packet to be traced, the IP address of the host which is the packet's destination, called the *victim*, and the approximate time of receipt. The packet must have been sent recently enough as logs are periodically overwritten by the DGAs. The victim's address is used by the STM as a starting point for the search.

The STM, after checking the authenticity and integrity of the request, sends a request to all SCARs in its domain, which obtain from their respective DGAs the traffic data for analysis.

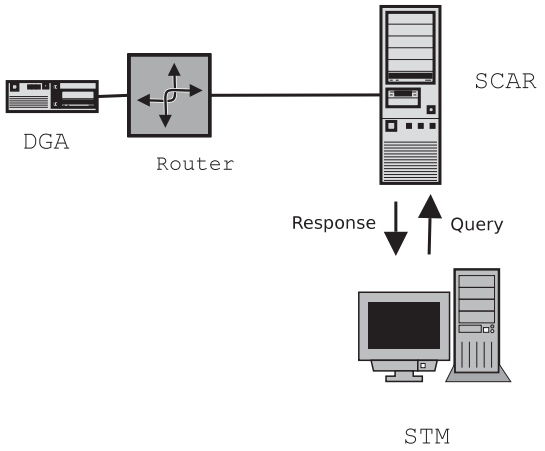

Fig. 2 – The SPIE architecture.

Time is critical, as the query must be processed before the desired log entries are removed from the DGAs.

The STM query starts at the SCAR responsible for the victim's network region. The SCAR then responds with a partial attack graph. The attack graph can complete in the SCAR region in case the source is identified within that region; or alternatively it can contain nodes at the border of this region. In that case, the STM sends a request to the SCAR that manages the network which contains that node. This process continues until there are no more possibilities of extending the attack graph, which occurs when a packet source is identified or when SPIE has searched across its complete network region. Finally, the STM builds the complete attack graph based on the partial graphs it received, and returns this graph to the client.

As mentioned above, IP packets can suffer valid transformations when they traverse the network. SPIE is capable of tracing those packets even after those transformations: before computing the hash of a packet, SPIE masks frequently changed IP headers. This operation hides most common transformations, but forces SPIE to explicitly manage the following transformations: fragmentation, NAT (*Network Address Translation*), ICMP message (*Internet Control Message Protocol*), IP in IP tunneling (Simpson, 1995) and IPsec (*IP Security*) (Kent and Atkinson, 1998).

If the original packet has to be reconstructed, additional resources are necessary. In this case SPIE keeps, together with the digest tables stored by the DGA, a packet transformation table, called TLT (*Transform Lookup Table*). The TLT is kept for the same time interval as the DGA's log entry. Each entry on the TLT consists of three fields. The first contains the resulting hash digest for the referred packet. The second field specifies the transformation type (three bits are enough to identify the transformations described above). The last field contains other packet information, which depends on the transformation used. Fig. 3 shows an example TLT entry.

Each SCAR builds a partial graph, which is a subgraph of the desired graph using the topology information of its particular network region. After collecting all DGAs' traffic tables, the SCAR simulates the Reverse Path Flooding Algorithm using locally stored tables.

In order to query the DGA, the SCAR computes the hash of the specified packet and then searches the local log for a matching entry. If there is a positive match, the DGA is considered to have processed that packet. This node is then added to the attack graph and the process continues at each of its neighbors, with the exception of those already visited. However, if there is no positive match, it may be necessary to repeat the query considering log entries already discarded, created in a previous interval. Depending on network latency, an SCAR may need to query multiple tables of each DGA. Once the hash value is found, the time of receipt is considered to be

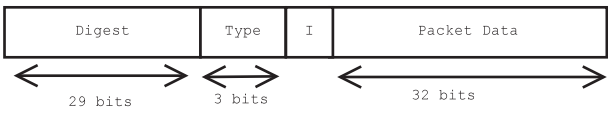

Fig. 3 – A TLT entry has enough information to reconstruct an original packet.

the most recent possible. This guarantees that the packet must have been processed previously by adjacent routers. Fig. 4 shows an example execution of the algorithm for victim V and attacker A. The traversed path of the packet is A, R07, R02, R01, V.

If the packet was not found after the search was executed on all tables available, the search is concluded in this network region but continues on other nodes. A list of all visited nodes is maintained to guarantee the algorithm termination.

The final attack graph that results from the execution of the process described above is a connected graph containing the set of nodes which are believed to have been traversed by the packet towards the victim. Assuming correct router operation, this graph should contain the nodes from the real attack graph. Due to hash collision, however, the graph can contain false positives, i.e. nodes which have not processed the packet. Furthermore as we show below, even if the nodes have really processed the packet it is possible that they appear in the wrong order in the attack graph returned by SPIE.

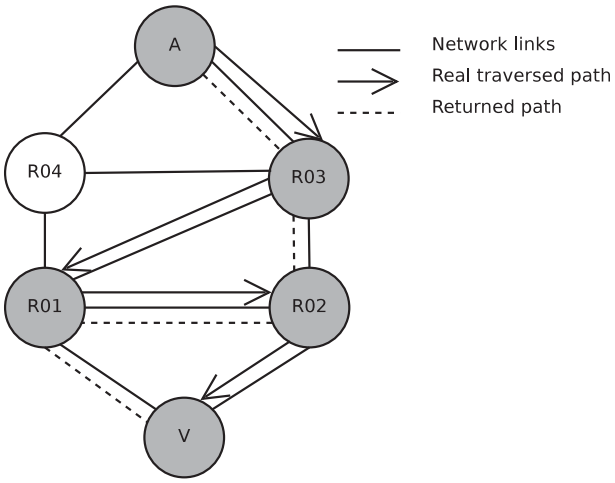

Fig. 5 – An example of a real path traversed and the incorrect attack graph returned by SPIE.

4. Incorrect attack graphs returned by SPIE

In this section we show that the attack graph returned by SPIE may contain what we call *false edges*, which correspond to links that have *not* been traversed by the packets. Fig. 5 shows an example. The arrows represent the attack path traversed by the packet that triggered the traceback process. Solid lines represent the links connecting nodes in the real network topology. The edges of the returned attack graph are represented by dotted lines. Initially the victim (node V) queries neighbor R1 and includes link R1–V in the attack graph. Note that this link was not traversed by the packet. Then, link R2–R1 is included which was traversed. From R2, a new mistake occurs and link R3–R2 is included in the attack graph, and it was not traversed by the packet. Note that link R1–R3 which was traversed by the packet is not included in the attack graph. Finally, the link to the real attacker (A–R3) is included. Despite the fact that in this example the attacker was found, we show below an example in which the attacker cannot be unambiguously determined.

SPIE can add a false edge to the attack graph whenever two neighbors of a given node have processed the packet. Using SPIE it turns out to be impossible to determine from which neighbor the packet came. In other words, let $G = (V, E)$ be the attack graph, and $T = (V', E')$ the real network topology. Whenever $u, v, w \in V$ and $(u, v), (u, w) \in E'$, SPIE *can* add either (u, v) or (u, w) to E. In this case we call (u, v) a false edge, and the corresponding attack graph V is incorrect.

Fig. 6 shows an example of an incorrect attack graph that leads to the incorrect determination of the packet source. In part a, the attacker (A) sends through a router (R), a packet to the victim (V). Any of the three attack graphs labeled b, c, and d can be returned by SPIE, depending on the implementation of the search algorithm. Special attention is required when examining attack graph c: in this case it is impossible to determine the origin of the attack.

In next section we present a solution to this problem. This solution allows routers to be sorted in the order they processed the traceback packet.

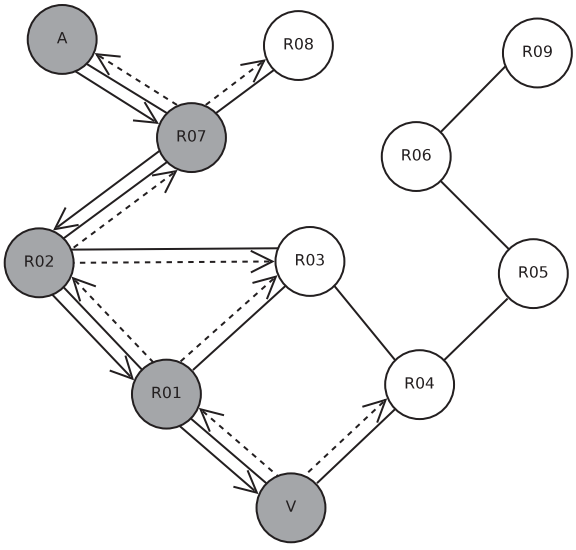

Fig. 4 – Example attack graph. Dotted lines represent the attack path. Dashed lines represent SPIE requests.

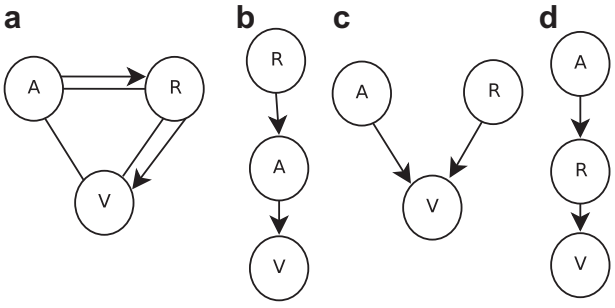

Fig. 6 – Example of a real attack and three graphs that can be returned by SPIE.

5. Extensions to SPIE

A traceback architecture can be evaluated using three metrics. The first metric is the precision, which reflects how close the attack graph returned is in comparison with the real attack route. The second metric is the number of queries employed by the traceback operation, which should be as small as possible. The third metric is the size of logs employed. In this and the next sections we describe three extensions to SPIE architecture that improve the precision of returned attack graphs featuring the exact reverse path the packet has traversed. The extensions also improve the efficiency of the traceback process, both in terms of the number of traceback queries required, and by adjusting the log size with a dynamic mechanism which assumes a variable link usage.

5.1. *Improving the traceback precision*

In order to avoid the inclusion of false edges in the attack graphs, a deceptively simple strategy can be used. This solution is based on the variation of the TTL field which, if left unmasked, will allow routers to be sorted in the order they processed the traceback packet. However, computing the hash with an unmasked TTL represents a new challenge: the search procedure which was based on a single hash computed for each packet cannot be used anymore. Furthermore, new search algorithms are needed because packet data has to be kept confidential as the search is processed.

Routers that processed the packet appear in the correct order because the TTL is decremented before the packet leaves each router. As shown in Fig. 7 we compute the hash using the invariant part of the IP Header plus the TTL (Time-to-Live) field and the first 8 bytes of the payload (28 bytes total). Other fields that change as the packet is processed are masked. Results presented in Snoeren et al. (2001, 2002) show that 28 bytes (20 bytes from the IP header plus 8 bytes from the payload) are enough to identify almost all non-identical packets. We will see later that this brings two advantages: (1) traceback is more efficient than SPIE's, as fewer queries are required; (2) traceback is more precise, it is possible to determine the complete route traversed by the packet, and to unambiguously determine its source.

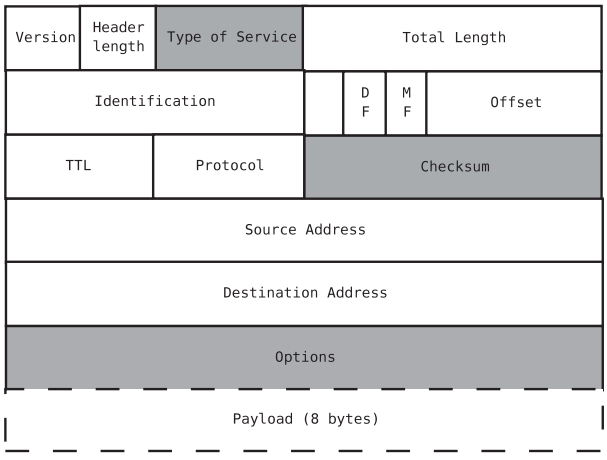

Fig. 7 – An IP packet; grey fields are masked.

Nevertheless, as the TTL field is also used as input to the hash function, the search algorithm must also consider the variation of this field: initially the first query is sent with the TTL of the received packet, then it is incremented in each subsequent query. Thus unmasking the TTL field when computing the hash of a processed packet solves the false edge problem, but raises a new problem regarding the confidentiality of the traceback.

Tracing back packets in the Internet presents several issues not all of which are purely technical. The traceback operation generally requires cooperation among different autonomous systems belonging to distinct administrative domains. An autonomous system may wish not to expose any part of the packet being traced, even if it is just a few bytes of the packet metadata. In this case a solution is to send queries only with packet hashes. This represents a challenge, if SPIE is to employ a visible TTL field.

We present two different solutions to this problem. In the first solution the search procedure is executed in a centralized way by the STM; in the second approach each SCAR recursively issues queries to its neighbors. In the first approach the STM queries the first SCAR with the packet hash with the TTL incremented by one. The SCAR, after querying its DGAs, replies with the neighbors which effectively processed the packet and the STM then queries those neighbors with the new hash computed from the original packet with its TTL incremented as appropriate. The procedure is repeated until there are no SCARs left to be queried. Fig. 8 illustrates this algorithm. Solid lines represent the node's adjacent links and the dotted lines represent the components' requests and replies. As a router may decrement the TTL field by more than 1, consecutive queries may be required with incremented TTL values until a positive reply is received or the maximum value is reached.

The second approach consists of a recursive search executed in a distributed way by the SCARs themselves. The STM sends the query to the first SCAR, which in turn sends queries to its neighbors. Eventually there will be no more neighbors left to be queried and the last SCAR will reply the received request. Each SCAR then returns the attack graph to its predecessor. At last, the STM receives the complete attack graph. In this case, as each query must be executed with a different hash because of the TTL increment, each query must be composed by a list of all necessary hashes to the forthcoming requests. In each recursion step, one of the hashes is removed from the list. The disadvantage of this approach, in

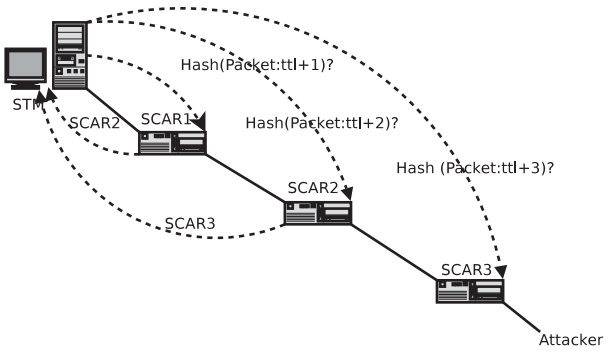

Fig. 8 – Iterative traceback algorithm.

comparison with the iterative algorithm presented above, is the increase of the request message size. Fig. 9 illustrates an example execution of this algorithm.

5.2. *Improving the traceback efficiency*

In this section we present two extensions to SPIE that improve the efficiency of the traceback process. Initially we present a dynamic approach for determining the moment to store a log in secondary memory. Then we discuss the benefits of employing one filter per routing interface, instead of one per router.

5.2.1. *Using dynamic logs for improving the storage management*

In order to use storage space efficiently, the DGA employs a new dynamic approach for paging the Bloom filters to secondary memory. SPIE pages the Bloom filter using a fixed pre-determined period of time. The DGA we propose, on the other hand, receives as parameter the maximum allowed capacity factor and the paging occurs only when this limit is reached. The capacity factor is related to how much filled up the Bloom filter is. Thus, the period in which a Bloom filter remains in main memory is variable, it could be longer when network traffic is low and shorter when more traffic is processed. When the maximum capacity factor is reached, the Bloom filter is paged to secondary memory together with control data that specifies the time interval for the stored information.

If the traffic on a given segment is highly variable, the improvement in the efficiency can be dramatic, in comparison with employing a fixed time interval. If the traffic is very high during the fixed time interval, the number of false positives increases, making the search process inefficient. On the other hand, if the traffic is low, the Bloom filter may be prematurely stored in secondary memory.

5.2.2. *Keeping one bloom filter per network interface*

Another extension we propose to SPIE improves the efficiency of the traceback process. In previous approaches a Bloom filter was kept for each router, and whenever the router returned a positive result to a given query, that would be forwarded to *all* its neighbors. In the proposed extension a Bloom filter is kept for each router interface. This extension assumes memory is cheap and that it is feasible to keep one Bloom filter per segment connected to the router. Thus for each traced packet the exact interface through which it was received can be identified, and thus only one query has to be sent to one neighbor, plus false positives. Please note that even if more than one neighbor has processed the packet, because of the TTL field only one interface will return a positive result.

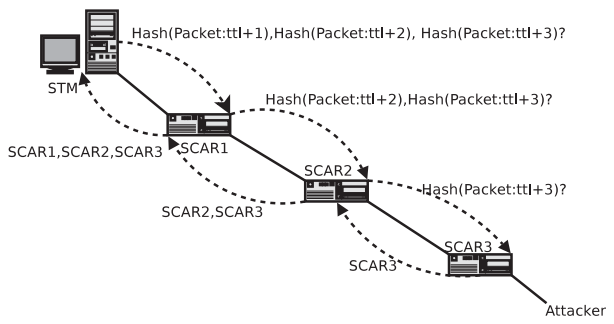

Fig. 9 – Recursive traceback algorithm.

As for each query processed by a single DGA several Bloom filters are consulted, the false positive rate as a whole is higher. Analytically we can demonstrate that, for each router with n interfaces and a Bloom filter with p false positives rate, the probability of obtaining at least one false positive is equal to $1-(1-p)^n$. For each false positive one more request will be executed by the STM. This is still a very low rate.

6. Implementation and results

In this section an implementation of the proposed system is described, as well as experimental results. The DGA was implemented as a sniffer using the PCAP (Packet Capture) library (tcpdump/libpcap, 2009). There is a thread for each network interface that maintains a Bloom filter. When the DGA receives a request it queries each Bloom filter, using shared memory. A configuration file is used to set the DGA's neighbors. The DGA reply contains the address of the neighbor that processed the packet; more than one address is returned in case false positives occur. The SCAR's parameters include the nearest DGA address (IP address and port) and a binary file that contains the packet and its time of receipt. The file format is the same as used in the PCAP library, which can be generated by the *tcpdump* tool.

Experimental results were obtained by running several realistic test scenarios on the implemented system. We compared the number of queries generated with other approaches, discounting false positives. An evaluation was also performed showing the relation of the number of interfaces, the false positive rate of an individual Bloom filter, and the total false positive rate taking into account the whole set of Bloom filters employed.

The experiments were carried out on an Ethernet LAN on which we simulated networks with Internet-like topologies generated with the Waxman (1988) method, each network had 50 nodes. In the Waxman method, the probability that a pair of nodes is connected by an edge varies according to their distance. Random attacks were generated with paths of sizes 10, 15, 20, 25 and 30. Synthetic traffic was inserted in the respective Bloom Filters. An attack path was simulated by inserting the same packet in several Monitors in the network, changing only the TTL field.

Two metrics were evaluated. The first metric was the attack graph precision, in terms of the number of false edges. The second metric was the number of queries generated by the traceback algorithm. Measures were obtained for (1) SPIE running the Reverse Path Flooding algorithm – in this case, a list of visited nodes was kept to guarantee that the algorithm completes; (2) SPIE with two of our extensions, namely unmasking the TTL plus using our log search algorithms and employing dynamic logs; (3) SPIE with the full extensions proposed, including keeping one log per interface.

Fig. 10 shows results in terms of the number of false positives including false edges. It can be seen that SPIE returns attack graphs with the largest number of false edges. Meanwhile the

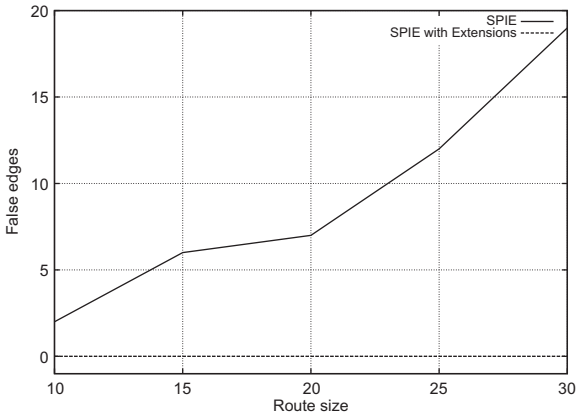

Fig. 10 – False edges returned by SPIE and SPIE with extensions.

attack graphs returned by SPIE with the proposed extensions do not show any false edges.

Fig. 11 shows the number of queries employed in the traceback. It can be seen that employing one Bloom filter per node requires a larger number of queries, because all neighbors of a node that has processed a given packet are also queried. This problem is solved by having one Bloom filter per interface, from which the neighbor that has actually processed the packet can be directly determined. Thus, the number of queries is the minimum possible, which is the total number of nodes in the attack path. According to the results shown in Fig. 7 we observe that the total number of queries is improved by at least 60% in comparison with SPIE.

The false positive rate is computed for all Bloom filters at a DGA (one per interface) that represent the traffic in a given period of time. As each Bloom filter has a false positive rate p that can be computed with its capacity factor c and the number of hash functions employed k, the probability of at least one false positive occurring at a DGA that has n interfaces is $1-(1-p)^n$.

Table 1 shows the false positive rate for a DGA with several Bloom filters (one for each interface) with capacity factor 13 using 8 hash functions and varying the number of interfaces. The false positive rate for a single Bloom filter is 0.199%. However, if the node has, for instance, 10 interfaces the false positive rate increases to 1.972%, and if it has 20 interfaces, 3.906%. The last two columns of Table 1 show the total number of queries generated from this node by 100 traceback requests, including the false positives and considering both an approach with only one log per node and the proposed approach.

Table 1 – False positive rates for SPIE, and SPIE with extensions, with and without one log per interface.

#Interfaces	False positive rate	#Queries 1 log/router	Queries 1 log/interface
1	0.00199	100	100
2	0.00398	200	100
3	0.00596	300	100
4	0.00794	400	101
5	0.00991	500	101
6	0.01188	600	101
7	0.01385	700	101
8	0.01581	800	102
9	0.01777	900	102
10	0.01972	1000	102
11	0.02167	1100	102
12	0.02362	1200	102
13	0.02556	1300	103
14	0.02750	1400	103
15	0.02944	1500	103
16	0.03137	1600	103
17	0.03330	1700	103
18	0.03522	1800	104
19	0.03714	1900	104
20	0.03906	2000	104

The impact of false positives is felt as an increase in the number of queries issued: each false positive causes an extra query. It is possible to conclude that despite the fact that the proposed approach causes an increase in the number of false positives, the total number of queries issued is still much smaller than that of the previous approaches. If the router has 12 interfaces, for example, from a minimum of 100 requests up to 102 requests would be generated; SPIE would generate 1200 requests in this case.

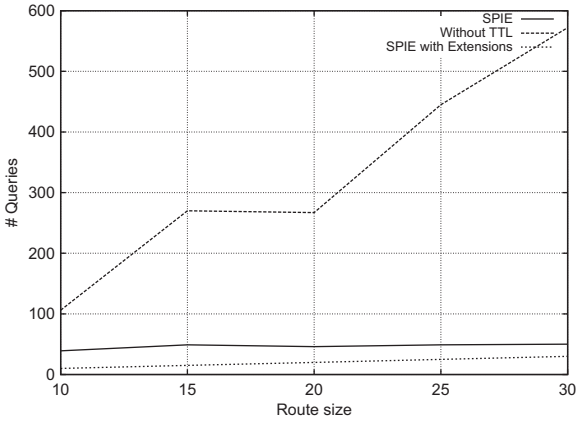

Fig. 11 – Number of queries SPIE, and SPIE with extensions, with and without one log per interface.

7. Conclusions

The ability to determine the path traversed by an IP packet is increasingly important in the Internet. In this work we described extensions that improve both the precision and efficiency of the classical SPIE architecture for log-based IP traceback. We showed that SPIE can return incorrect attack graphs that contain edges not traversed by the packet and which may even avoid the determination of the real attacker. We solve this problem by unmasking the TTL field and proposing new distributed log search strategies that keep the traceback confidentiality as no sensitive packet information is communicated. Two new traceback algorithms are proposed which guarantee that communication among the system's components preserves the confidentiality of the packet's information. We show that keeping separate logs for each router interface reduces the number of requests to a minimum.

A dynamic and efficient approach for paging logs to secondary memory is proposed based on the log's actual capacity factor, instead of a fixed time interval.

Future work includes adding the ability to handle tunneled packets, and also adapting the proposed architecture for hybrid networks.

Acknowledgments

This work was partially supported by the Research and Development of Dynamic Network Technology program of the Japanese NiCT, and grant 311221/2006-8 from the Brazilian Research Agency (CNPq).

REFERENCES

Baker F. Requirements for IP Version 4 routers. Request for Comments 1812. IETF; 1995.

Belenky A, Ansari N. On IP traceback. IEEE Communications Magazine 2003;41(7):142–53.

Belenky A, Ansari N. On deterministic packet marking. Computer Networks 2007;51(10):2677–700.

Bloom BH. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 1970;13:422–6.

Burch H, Cheswick B. Tracing anonymous packets to their approximate source. In: Proceedings of the fourth symposium on operating systems design and implementation (OSDI 2000). San Diego; 2000. p. 319,327.

Castelucio A, Salles RM, Ziviani A. An AS-level overlay network for IP traceback. IEEE Network 2009;23(1):36–41.

Cong C, Sarac K. IP traceback based on packet marking and logging. In: IEEE international conference on communications (ICC); 2005. p. 1043–7.

Dean D, Franklin M, Stubblefield A. An Algebraic approach to IP traceback. ACM Transactions on Information and System Security (TISSEC) 2002;5(2):119–37.

Fan L, Cao P, Almeida J, Broder AZ. Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Transactions on Networking 2000;8:281–93.

Gao Z, Ansari N. Tracing cyber attacks from the practical perspective. IEEE Communications Magazine 2005;43(5): 123–31.

Gao Z, Ansari N. "A practical and robust inter-domain marking scheme for IP traceback,". Computer Networks 2007;51(3): 732–50.

Goodrich MT. Efficient packet marking for large scale IP traceback. In: The 9th ACM conference on computer and communications security (CCS); 2002. pp. 117–26.

Goodrich MT. Probabilistic packet marking for large-scale IP traceback. IEEE/ACM Transactions on Networking (TON) 2008; 16(1):15–24.

Hazeyama H, Kadobayashi Y. A layer-2 extension to hash-based IP traceback. IEICE Transactions on Information and Systems 2003;E86-D(11):2325–33.

Hilgenstieler E, Duarte Jr. EP, Keeni GM, Shiratori N. Improving the precision and efficiency of log-based IP packet traceback. In: Proc. of the 50th IEEE Globecom, Washington DC, USA; 2007.

Keeni GM. An architecture for IP packet tracing, http://www.cysol.co.jp/contrib/draft-glenn-ippt-arch-01.txt; 2009.

Kent S, Atkinson R. Security architecture for the Internet protocol. Request for Comments 2401. IETF; 1998.

Lai G, Chen C, Jeng B, Chao W. Ant-based IP traceback. Expert Systems with Applications 2008;34(4):3071–80.

Matsumoto Y, Hazeyama H, Kadobayashi Y. Adaptive Bloom filter: space efficient counting algorithm for unpredictable network traffic. IEICE Transactions on Information and Systems 2008;E91-D(5).

Microsoft Corporation. Stop 0A in Tcpip.sys when receiving out of Band (OOB) data, http://support.microsoft.com; 2008.

Sanchez LA, Milliken WC, Snoeren AC, Tchakountio F, Jones E, Kent ST, et al. Hardware support for a hash-based IP traceback. In: Second DARPA information survivability conference and exposition; 2001.

Savage S, Wetherall D, Karlin A, Anderson T. Practical network support for IP traceback. In: Proceedings of the ACM special interest group on data communications 2000 (SIGCOMM'2000); 2000, p. 295–306.

SecurityStats.com. Security Statistics, http://www.securitystats.com/; 2009.

Simpson W. IP in IP tunneling. Request for Comments 1853. IETF; 1995.

Snoeren AC, Partridge C, Sanchez LA, Jones CE, Tchakountio F, Kent ST, et al. Hash-based IP traceback. In: Proceedings of the ACM special interest group on data communications 2001 (SIGCOMM'2001); 2001, p. 3–14.

Snoeren AC, Partridge C, Sanchez LA, Jones CE, Tchakountio F, Schwartz B, et al. Single-packet IP traceback. IEEE/ACM Transactions on Networking 2002;10(6):721–34.

Sung M, Xu J, Li J, Li L. Large-scale IP traceback in high-speed Internet: practical techniques and Theoretical Foundation. IEEE/ACM Transactions on Networking 2008; 16(6):1253–66.

tcpdump/libpcap "TCPDUMP Public Repository", http://www.tcpdump.org/; 2009.

Waxman BM. "Routing of multipoint connections". IEEE Journal of Selected Areas in Communications; 1988:1617–22.

Wong TY, Wong MH, Lui JCS. "A precise termination condition of the probabilistic packet marking algorithm". IEEE Transactions on Dependable and Secure Computing 2008;5(1):6–21.

Wu SF, Zhang L, Massey D, Mankin A, Wu CL. "Intention-Driven ICMP Trace-Back," Internet Draft. IEFT; 2001. draft-wu-itrace-intention-00.txt.

Yaar A, Perrig A, Song D. FIT: fast Internet traceback. In: the twenty-fourth annual joint conference of the IEEE computer and communications Societies (INFOCOM); 2005, p. 1395–1406.

Egon Hilgenstieler received both his Master and B.Sc. Degrees in Computer Science from Federal University of Parana, Brazil. Research interests include Computer Networks and Security.

Elias P. Duarte Jr. is an Associate Professor at Federal University of Parana, Curitiba, Brazil. His research interests include computer networks and distributed systems, their dependability, management and algorithms. He has chaired or served on the program committee of several conferences including DSN, ICDCS, SRDS, ISPA, IM, NOMS, among others. He received his Ph.D. in Computer Science from Tokyo Institute of Technology, Japan, 1997; M.Sc. in Telecommunications from the Polytechnical University of Madrid, Spain, 1991, and B.Sc. in Computer Science from Federal University of Minas Gerais, Brazil, 1987. He is a member of The Brazilian Computing Society and the IEEE.

Glenn Mansfield-Keeni received his Ph.D. in Logic programming, from Tohoku University, Japan. He is currently President/CEO of Cyber Solutions Inc., Sendai, Japan, and a senior visiting researcher at the Research Institute of Electrical Communications, Tohoku University, Sendai, Japan. His areas of interest include expert systems, computer networks and their management, security, etc. He is a member of the ACM, the IEEE Communications Society and is an active member of the IETF.

Norio Shiratori received his doctoral degree from Tohoku University, Japan in 1977. Presently he is a professor of the Research Institute of Electrical Communication, Tohoku University. He has engaged in research related to symbiotic computing paradigms between human and information technology. He was recipient of the IPSJ Memorial Prize Winning paper award in 1985, the Telecommunication Advancement Foundation Incorporation award in 1991 and many others. He was the vice president of IPSJ in 2002 and a fellow of IEEE, IPSJ and IEICE.