ISSN 2278 – 0211 (Online)

Denial of Service Forwarding Anomalies in Bloom Filter

R. Mary Rifo Nisha
M.E. Student, Department of Computer Science and Engineering
PET Engineering College, Vallioor, India
P. Krishna Kumar
Associate Professor, Department of Computer Science and Engineering
PET Engineering College, Vallioor, India

Abstract:
Bloom-filter-based forwarding has been suggested to solve several fundamental problems in the current Internet, such as routing-table growth, multicast scalability issues, and denial-of-service (DoS) attacks by botnets. The proposed protocols are source-routed and include the delivery tree encoded as a Bloom filter in each packet. The network nodes forward packets based on this in-packet information without consulting routing tables and without storing per-flow state. DDoS attacks does not seek to breach data integrity or privacy; they can be conducted without the requirement of identifying vulnerabilities to exploit the application.

Keywords: *Multicast, network protocols, network-level security and protection*

1. Introduction

IN-PACKET Bloom- filter -based forwarding has been pro-posed as a solution for several problems in the current Internet, including routing-table growth and scalable multi-cast [1], denial-of-service (DoS) resistant forwarding [2], and information-centric network design [3], [4]. Its use has also been proposed for data centers [5], [6] and as an enhancement for Multi-Protocol Label Switching (MPLS) [7]. There are several implementations, including one for *NetFPGA*.

The idea behind in -packet Bloom -filter-based forwarding is relatively simple: The delivery tree is stored in the packet header as a set of *forwarding- hop identifiers* (FHIDs), which can be either nodes, links, or in–out interface pairs on the delivery tree. The set of the FHIDs in the delivery tree is encoded as a Bloom filter [10] data structure, which enables efficient testing of set membership. Network nodes forward packets by checking which potential FHIDs, e.g., outgoing links, are in the Bloom filter.

In comparison to standard IP routing-table lookup or solutions that use Bloom filters to encode routing tables [11] or destination addresses [12], [13], in-packet Bloom-filter -based for-warding requires little computation at the routers. It can also scale to much larger numbers of multicast groups than the cur-rent IP multicast protocols because the network nodes do not need to store any per-group or per-flow state.

These properties make it attractive to use the same protocol for both unicast and multicast.

Existing Bloom-filter-based protocols have proposed three security mechanisms: 1) limiting the number of items stored in the Bloom filters; 2) centralizing Bloom filter computation and making forwarding-hop identifiers secret; 3) using cryptographically computed per-flow forwarding-hop identifiers.

However, perhaps surprisingly nobody until today has thoroughly evaluated the security of these proposals. In this paper, we start this work by analyzing the denial-of-service resistance of the existing Bloom-filter-based forwarding architectures. Many variations of such protocols have been proposed in the literature (see Section II). Since no single protocol has yet been standardized, the analysis in this paper aims to cover all the different variants of encoding the delivery tree in to the in-packet Bloom filter. To do this, we have created a unified connectivity-graph model that makes it possible to analyze different protocol variants with a single model.

We evaluate the effectiveness of the security mechanisms against denial-of-service attacks using three different attacks. First, we show that static link-identifiers can be reverse-engineered. This was hypothesized in [2]. However, our paper is the first to show this is actually the case. Second, we show that distributed packet-flooding attacks can get around the pro-posed security mechanisms in existing literature. Third, some protocol variants are vulnerable to attacks that prevent nodes from leaving a multicast group. These attacks show that most of the abstract security claims presented in the literature do not hold under detailed analysis.

The attacks are distributed. That is, they require the attacker to have access to a botnet consisting of at least hundreds of compromised computers. This is a reasonable assumption be-cause real DoS attacks on the current Internet are commonly launched from botnets. It

is also estimated that a large portion (16%–25%) of Internet hosts belong to botnets [14]. Moreover, the abstract security claims made about the Bloom-filter-based protocols cover distributed denial-of-service attacks.
Our work shows that Bloom-filter-based forwarding needs further improvements on security before deployment on open networks. However, it should be noted that the security mechanisms proposed in the literature do increase the cost of DoS attacks and, thus, they may be useful in combination with further security solutions.

2. Problem Statement

This section gives an overview of the Bloom-filter-based for-warding proposals. Section II-A provides background information on Bloom filters, and Section II -B surveys the proposed protocols. (For readers not previously familiar with the concepts, a brief tutorial is provided in Appendix A.) Section II-C discusses the previous work on denial-of-service attacks and countermeasures in this family of protocols.

2.1. Bloom Filters

Bloom filter [10] is a probabilistic space-efficient data structure for storing sets. Its basic operations are *membership testing* and *element addition*, but not element removal. Bloom filter is implemented as a bit array with fixed length m and a small number k of hash functions that map data items to indexes $[0, m-1]$ in the bit array. A data item is added to the set by computing the k hash functions on it and setting the corresponding k bits in the bit array to 1.

2.2. Bloom-Filter-Based Multicast Forwarding

Three methods have been proposed for Bloom-filter-based multicast forwarding: 1) using Bloom filters in the multicast routers to reduce the space required for storing the multicast forwarding table; 2) source routing where the multicast de-livery-tree is encoded as a Bloom filter in the packet headers (e.g., [1] and [3]); and 3) storing the list of receivers in the packet header as a Bloom filter.

2.3. Known DoS Problems and Solutions

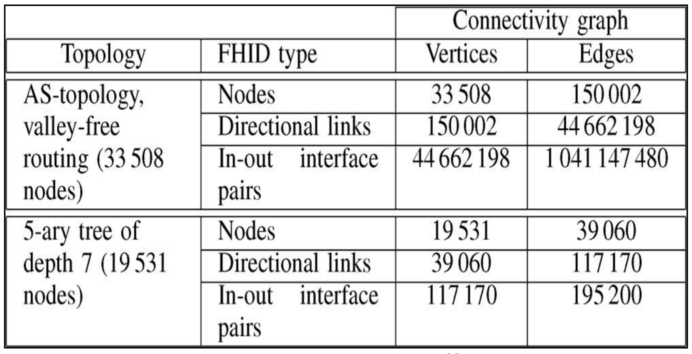
Multicast enables the sender to reach a large number of receivers even though it only sends each packet once. The use of Bloom filters creates a probabilistic element in packet for-warding; packets may be forwarded over links they were not intended as well as over the intended links. (However, false negatives, i.e., packets not forwarded over intended links, are not possible.) These two—traffic amplification and potential for false positives—create potential security problems.

3. Reverse-Engineering Attack

We will now start with the security analysis of Bloom - filter forwarding. In Section III-A, we explain how different variants of the link identifiers can be modeled in a uniform way. The model will be used in all the following sections including Section III-B, in which we show that an attacker with a botnet is able to reverse-engineer the supposedly secret link identifiers.

3.1. Connectivity Graph

As seen in Section II-C, the data items stored in the Bloom filter do not need to be just link identifiers. They could equally identify the *nodes* on the path, the *links* on the path, or the *incoming–outgoing interface pairs* on the path. Moreover, the identifiers could be *symmetric*, i.e., the same regardless of the direction in which the packets travel, or they could be *directional*, i.e., different in the upstream and downstream directions. All these variants provide sufficient information for the forwarding nodes to pass the packet onto the next-hop node. To avoid confusion, we will use the term *FHID* to denote all these different types of identifiers. This paper, however, focuses on directional identifiers because they give stronger sender access control. (Symmetric identifiers would be suitable for groups in which all members are allowed to send.)



		Connectivity graph	
Topology	FHID type	Vertices	Edges
AS-topology, valley-free routing (33 508 nodes)	Nodes	33 508	150 002
	Directional links	150 002	44 662 198
	In-out interface pairs	44 662 198	1 041 147 480
5-ary tree of depth 7 (19 531 nodes)	Nodes	19 531	39 060
	Directional links	39 060	117 170
	In-out interface pairs	117 170	195 200

Table 1: Connectivity Graph Sizes in Two Different Network Topologies

3.2. Reverse-Engineering FHIDs

The secrecy of forwarding-hop identifiers is critical to the security of Bloom-filter forwarding. For this reason, we first investigate the difficulty of reverse- engineering the identifiers. If the attacker knows all the FHIDs, it can flood specific targets with packets.

Moreover, even partial knowledge of the secret identifiers may enable the attacker to create routing loops or cause flow duplication intentionally

The attacker is thus able to reverse-engineer most FHIDs at the core of the network, but fewer of those close to the network edge. This means that the attacker is able to create routing loops or amplifying routes in the core network. On the other hand, the reverse-engineering attack does not seem to enable the flooding of arbitrary edge nodes. Periodic updating of the identifiers can make the attack somewhat more difficult because the attacker will have to repeat the reverse- engineering regularly. We can nevertheless conclude that it is wrong in principle to assume that the identifiers can be kept secret.

4. Injection Attack

This section presents a form of injection attack against Bloom- filter- based multicast. We show that if the attacker has control over just one node that can send to a target, then a significant portion of the botnet will be able to send packets to the target. Section IV-A explains the basic principle of the attack, and the following sections show that it is possible under increasingly stringent security assumptions.

4.1. Basic Injection

The target T has subscribed to a data flow from a compromised node, the bot B . These packets are sent with the path filter F in the packet header. The goal of the attacker is to enable another bot B_b to also send to the target T. If this succeeds with B_b , the attacker can try to do the same for all nodes in the botnet.

$$F_x = F_a \vee F_b. \qquad (1)$$

4.2. Injection with Permutations

The injection attack becomes more interesting if the for-warding nodes permute the Bloom filters. The secret permutations certainly make it impossible to combine the filters with

(1). For example, when each router in Fig. 4 permutes the filter, B_b cannot use $F \vee F_b$ to send packets to T because the bits of will be in a wrong order when they reach the point where the two paths meet. In order for them to be permuted into the right locations, the filter would have to traverse the path from B_a to the meeting point.

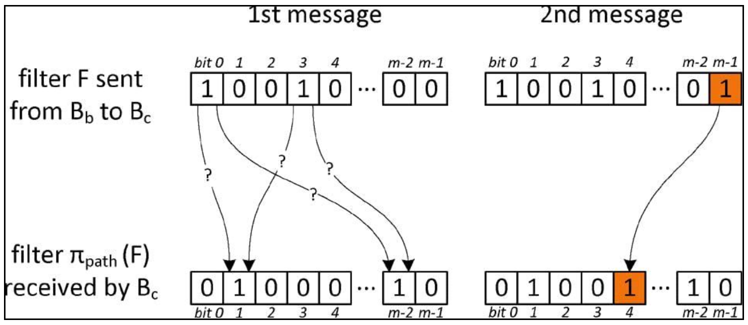


Figure 1

Fig. 5. Solving (partially) the composite permutation between two bots by turning 0-bits to 1-bits.

The second relation enables the packet to traverse from there to the target. Now, the attacker can compute the following filter from the composite permutations and filters that it already knows:

$$F_x = (\pi_c \bullet \pi_b)^{\ 1} ((\pi_c \bullet \pi_a)(F_a)) \vee F_b. \qquad (2)$$

It is clear that F_x satisfies the first subset relation. It also satisfies the second relation

$$F_x \supseteq (\pi_c \bullet \pi_b)^{\ 1} ((\pi_c \bullet \pi_a)(F_a)) \qquad (3a)$$

$$\Rightarrow \pi_b(F_x) \supseteq \pi_b \left((\pi_c \bullet \pi_b)^{-1} ((\pi_c \bullet \pi_a)(F_a))\right) \qquad (3b)$$

$$= \left(\pi_b \bullet \pi_b^{\ 1} \bullet \pi_c^{-1} \bullet \pi_c \bullet \pi_a\right) (F_a) \qquad (3c)$$

$$= \pi_a(F_a). \qquad (3d)$$

$$F_x = (\pi_c \bullet \pi_b)^{\ 1} \underbrace{(\pi_c \bullet \pi_a)(F_a \vee F_c)}_{F'} \vee F_b. \qquad (4)$$

This filter clearly satisfies the two subset relations required for an injection. However, it is not obvious from the above formula how the attacker is able to calculate its value. The key insight is that the attacker can set all of the unknown bits to 1. This is done using (5b) and (6b), where the $\neg$ symbol denotes bitwise NOT operation (i.e., bitwise complement)

$$F' = (\pi_c \bullet \pi_a)(F_a \vee F_c) \quad (5a)$$

$$= \underbrace{(\pi_c \bullet \pi_a)(F_a \wedge \neg F_c)}_{\text{left}} \vee \underbrace{(\pi_c \bullet \pi_a)(F_c)}_{\text{right}} \quad (5b)$$

$$F_x = (\pi_c \bullet \pi_b)^{-1}(F') \vee F_b \quad (6a)$$

$$= \underbrace{(\pi_c \bullet \pi_b)^{-1}(F' \wedge \neg(\pi_c \bullet \pi_b)(F_b))}_{\text{left}} \vee \underbrace{F_b}_{\text{right}} . \quad (6b)$$

The right sides of (5b) and (6b) are known to the attacker, and it can calculate the left sides because the bits whose mapping in the two composite permutations is unknown are zeroed out.

4.3. Injection with Unknown Topology
The key idea is that we can estimate the number of FHIDs in a path by counting the 1-bits in the filter. We can also deter-mine the approximate number of common FHIDs in two paths by computing their dot product (i.e., the number of 1-bits in the bitwise AND). Generally, the value x of the dot product of two Bloom filters, which share c common elements and thus have approximately ck bits at corresponding positions, follows the hypergeometric
Probability distribution

$$P(X = x) = \binom{\rho_1 m - ck}{x - ck}\binom{m - \rho_1 m}{\rho_2 m - x}\binom{m - ck}{\rho_2 m - ck}^{1} . \quad (7)$$

The distribution has the mean value $ck + (\rho_1 m - ck)(\rho_2 m - ck)(m - ck)_{-1}$. By calculating several dot products after periodic identifier updates and comparing the dot products to the mean values for different c, the attacker can determine the length of the shared paths with ever increasing confidence.

4.4. Injection with Flow-Specific FHIDs
Our analysis thus far has covered FHIDs that are static or periodically updating and the same for all data flows. We now turn to flow-specific identifiers. They are implemented by including some flow-specific information from the packet headers in the input to the Bloom-filter hash functions (e.g., the z-Formation [2]). The flow-specific information can be a sender or multicast-group identifier, or there can be a special identifier space for naming the flows. For unicast flows, both endpoint names could be used in a way similar to the TCP connection identifiers. The reasoning behind this defense mechanism is that the FHIDs and Bloom filters for different flows will be independent of each other. Consequently, the attacker cannot reverse-engineer FHIDs by computing the bitwise AND of path filters or concatenate paths with the bitwise OR. It might thus appear that flow-specific identifiers prevent the attacks discussed so far in this paper. This is unfortunately not the case.

5. Resubscription Attack
The final attack presented in this paper targets the distributed filter discovery defined by Särelä *et al.* [19]. The path Bloom filter is the bitwise OR of the FHIDs on the path. Recall that, in the distributed path discovery, this value is computed by sending a join packet from the subscriber to the publisher. The join packet starts with an all-zero filter, and the FHIDs are added to it hop by hop.
The problem with the distributed filter discovery is that nothing forces the subscriber to start the join packet off with an empty filter. A malicious subscriber may set the initial filter in the join packet to any filter value that it knows. For example, when some honest subscribers leave the multicast group, the malicious subscriber can observe this as a change in the Bloom filter of the received multicast packets. It can then send a join packet with the old multicast filter as the initial filter value. This will prevent any nodes from leaving the multicast group.

6. Discussion
One lesson from the attacks described in this paper is that the Bloom filters and their operations are not secure crypto-graphic constructs. It is possible to analyze and combine them in various unexpected ways in order to derive information and new filters that are useful in attacks. The correlation attack suggested by Rothenberg *et al.* [2] can be implemented, and we have seen other, even more effective attacks that combine for-warding paths in clever ways.
The natural question to ask at this point is whether any variant of Bloom-filter forwarding escapes the attacks presented in this paper. Some do. If the FHIDs and filters are computed by a trusted topology manager as a function of unique flow identifiers that are chosen by the topology manager, then the filters for all flows are independent of each other and cannot be combined with bitwise operations as required by the reverse- engineering or traffic-injection attacks. Moreover, if the FHIDs are updated frequently, that minimizes the impact of the resubscription attack. Unfortunately, the flow-specific identifiers require the for-warding nodes to compute the hash functions for each packet, which has serious implications on the forwarding performance. The cryptographic computation would completely negate the original argument that Bloom-filter forwarding is efficient be-cause only simple bitwise operations on the filters are needed at each forwarding hop.

All the defense mechanisms that do not require per -packet hash computation at the forwarding nodes are less secure against the injection attack. Two proposed mechanisms standout be-cause they can significantly reduce the probability of two paths meeting in a way that allows a bot to participate in the injection attack. First, inbound–outbound interface pairs are a better FHID type than node or link identifiers because they result in a larger connectivity graph and more independent paths across the network. Second, the hop-count-dependent FHIDs proposed in LIPSIN restrict meeting points of two paths F and $F_{\bullet}$ to be at the same distance from the beginning of the two paths. This will, however, work only if the filters are computed by a trusted topology manager so that the attacker cannot tamper with the hop count used for filter creation. Moreover, the defense is fully effective only if ingress filtering is deployed to prevent the attacker from setting nonzero initial hop counts in the packets that it sends. Both of these defense mechanism increase the number of bots needed for an overwhelming flooding attack, but their effectiveness depends on the circumstances such as the exact network topology and routing.

7. Conclusion

Our central conclusion is that Bloom- filter-based multicast is resistant to distributed packet flooding only under very stringent assumptions, i.e., when the link identifiers (or other for-warding-hop identifiers) are flow-specific and cannot be forged by the end-hosts. In practice, this requires per-packet crypto-graphic computation at the forwarding nodes and trusted infrastructure for discovering the Bloom filters, which would negate many of the advantages of the proposed Bloom-filter-based protocols. The protocol variants that do not implement these security mechanisms suffer from distributed DoS vulnerabilities comparable to the current Internet. This paper can be seen as a reminder that broad security claims should be presented with extreme care and that vague claims are often shown false after a more careful study.

8. Acknowledgment

I would like to thank my guide Prof. P .Krishna Kumar for assisting me in this paper work.

9. References

i. M. Ain, S. Tarkoma, D. Trossen, and P. Nikander, “Conceptual architecture of PSIRP including subcomponent descriptions,” PSIRP project, Deliverable D2.2, 2008.

ii. CAIDA, La Jolla, CA, USA, “The CAIDA AS relationships dataset,” Jan. 20, 2010 [Online]. Available: http://www.caida.org/data/active/as-relationships/

iii. L. Gao, “On inferring autonomous system relationships in the In-ternet,” IEEE/ACM Trans. Netw., vol. 9, no. 6, pp. 733–745, Dec. 2001.

iv. S. Ruj, M. Stojmenovic, and A. Nayak, “Privacy Preserving Access Control with Authentication for Securing Data in Clouds,” Proc. IEEE/ACM Int’l Symp. Cluster, Cloud and Grid Computing.

v. 5. C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, “Toward Secure and Dependable Storage Services in Cloud Computing,” IEEE Trans. Services Computing, vol. 5, no. 2, pp. 220-232, Apr.- June 2012.

vi. J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, “Fuzzy Keyword Search Over Encrypted Data in Cloud Computing,” Proc. IEEE INFOCOM, pp. 441-445, 2010.

vii. X. Boyen, “Mesh Signatures,” Proc. 26th Ann. Int’l Conf. Advances in Cryptology (EUROCRYPT), pp. 210-227, 2007.

viii. D. Chaum and E.V. Heyst, “Group Signatures,” Proc. Ann. Int’l Conf. Advances in Cryptology (EUROCRYPT), pp. 257-265, 1991.

ix. H.K. Maji, M. Prabhakaran and M. Rosulek, “Attribute-Based Signatures: Achieving Attribute-Privacy and Collusion-Resis- tance,” IACR Cryptology ePrint Archive, 2008.

x. H.K. Maji, M. Prabhakaran and M. Rosulek, “Attribute-Based Signatures,” Topics inCryptology-CT-RSA,vol.6558,pp.376-392,2011.

xi. A. Beimel, “Secure Schemes for Secret Sharing and Key Distribu- tion,” PhD thesis, Technion, Haifa, 1996.

xii. A. Sahai and B. Waters, “Fuzzy Identity-Based Encryption,” Proc.Ann. Int’l Conf. Advances in Cryptology (EUROCRYPT), pp. 457-473,2005.

xiii. V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data,” Proc. ACM Conf. Computer and Comm. Security, pp. 89-98, 2006.

xiv. S. Yu, C. Wang, K. Ren, and W. Lou, “Attribute Based Data Sharing with Attribute Revocation,” Proc. ACM Symp. Information, Computer and Comm. Security (ASIACCS), pp. 261-270, 2010.