About One Monitoring Algorithm of Permutation Worms

[a]Shikhaliyev Ramiz Huseyn, and [b]Seung-Woo Seo

[a]Institute of Information Technology, Azerbaijan National Academy of Sciences, 9, F.Agayev str., Baku, Azerbaijan Republic, AZ1141,
[b]School of Electrical Engineering and Computer Sciences, Seoul National University, Seoul 151-744, Republic of Korea,

e-mail : *ramiz@science.az, sseo@snu.ac.kr*

Abstract

In recent years different strategies have emerged for increasing their rate of proliferation of worms. Therefore, for the detection of worms, particularly worms with permutation scanning, high speed monitoring and analysis of network traffic in real time is important. However, due to the emergence of computational difficulties and problems with flow storage, the solution of this problem using a deterministic algorithm becomes very difficult. Therefore, we propose a method for monitoring network traffic through the use of streaming algorithms, in particular a sliding window mechanism, which requires very little memory and computational resources.

I. Introduction

In recent years, the complexity of the mechanisms of infection and velocity of propagation of worms has risen significantly. Today, worms are capable of infecting millions of hosts in a very short time, resulting in a great amount of damage dealt. At the same time, to spread worms use different scanning strategies [1], such as scan-based preferences of local addresses, topological scanning, scanning of a pre-defined list of vulnerable sites, random scans, sequential scans, scanning based on permutations, and partial scans.

When an instance of all permutation scanning worms share a common space we can also use IP-addresses rearranged in pseudorandom order [2]. Such a permutation can be generated using any 32-bit block cipher with a pre-selected key. With this approach, there is little likelihood that the different instances of the worm will choose the same victim-host and the worms will not waste time scanning the same host several times. Each infected host, at once in search of vulnerable hosts starts scanning the space of permutations. When infected, the worms with a permutation scan starts scanning from the randomly selected point. When a distribution model using permutation scanning of worms sends a message that reaches the vulnerable hosts being scanned, the number of infected active and remote hosts varies [3]. Thus, when a host becomes infected by the worm, he or she attempts to spread the worm to other hosts to infect them and thus generates a very large number of network

connections.

Because the active hosts contribute to the rapid spread of worms with a permutation scan, their Early in the definition is very important. In particular, the earlier definition of active hosts, allow detecting and preventing the further spread of worms with a permutation scan, and resulting in minimizing the damage caused. However, this task is impossible without constant monitoring and analyzing network traffic in real time. To this end, we propose an algorithm for monitoring and analyzing network traffic that allows real-time identification of hosts that are associated with a large number of different hosts, i.e. sending the same package to a large number of different hosts. The essence of the proposed algorithm lies in the fact that over a short period of time from a given stream of pairs of IP addresses to host (s, d) available in the network traffic from s to determine the IP addresses of those who have connections with many different IP addresses from d. That is, determination s is paired with a large number of IP addresses of the d, where s and d, respectively, are the IP addresses of sources and destination hosts.

However, due to the emergence of computational difficulties and problems with flow storage, the solution for this problem using a deterministic algorithm becomes very difficult. This is mainly due to the fact that the volume and speed of continuously incoming traffic in modern computer networks are both very high and can contain millions of flows of IP packets. Therefore, to solve this problem we encourage the use of randomized streaming algorithms that use less computing power and memory than deterministic algorithms. In particular, we propose a streaming algorithm that uses a sliding window method [4, 5], which uses a very limited memory space for data storage and processing of data from a single pass, using less computational resources.

II. Algorithm for permutation worms monitoring

In this section we present a formal definition of the problem, and then describe the algorithm for determining active hosts.

2.1 A formal statement of the problem

Let be network traffic, is modeled as a stream consisting of a pair of addresses of the source-destination (s, d) received sequentially (in which the flows $d_i \neq d_j$). Also let h_1 and h_2 be a random hash function [6] which allows for the selection of uniform random samples consisting of different pairs of source-destination addresses. Here, the hash function h_1 maps a pair of source-destination address (s, d) to $[0, 1]$, i.e. $h_1(s, d) \rightarrow [0, 1]$, a hash function h_2 maps the source address s to $[0, 1]$. Moreover, the hash functions h_1 and h_2 must meet the requirements of fast computation and minimal collision. Thus, each pair of source-destination addresses, as well as individual addresses of sources with equal probability, can be mapped to any value in the interval $[0, 1]$. At the same time we assume that the hash functions are independent and their use ensures that the probability of inclusion in the sample does not depend on the number of occurrence of individual pairs of source-destination addresses in a stream. That is, each pair is equally likely to be included in the sample. Also, suppose we are given two hash tables T_1 and T_2, which are respectively associated with the hash functions h_1 and h_2 where hash table T_1 is designed to detect and exclude duplicate pairs of source-destination addresses, and the hash table is used to calculate T_2, the number of different destinations that are associated with which each source address.

Since the thread can be packages in the title are the same pair of source-destination addresses, the mapping carried out by a hash functions h_1 and h_2 can lead to collisions [6]. In the event of conflict, two or more of the same pair of source-destination addresses are associated with the same cell of the hash tables. Therefore, in order to solve the problem of collisions for each hash function h_1 and h_2, we use hash tables with blocks [6] that may contain records that are displayed as hash functions in the same address in the hash tables. This will allow

detecting and eliminating of duplicate pairs of source-destination addresses, and count the number of different destinations that are associated with which each source address. At the same time, the number of blocks in the hash table defines the lower boundary of the memory required for the determination of active hosts in the stream.

It is required to identify the active hosts on the last W packets, that is, to identify hosts that are associated with many different hosts in W packets, where W is the length of the sliding window. This should use less memory than what is required to store all the pairs of IP addresses in the current window.

2.2 Description of the algorithm for determining active hosts

First, we define the active hosts as hosts with addresses s, which in the window W with N pairs of source-destination addresses of hosts, the number of connections with different hosts addresses of d exceeds a predetermined threshold k. At the same time, in a given set of packets N, N/k of active hosts and their need for storage space $\Omega(N/k)$, which gives us a lower limit necessary for a finding of active hosts.

A simple approach to reduce the memory usage is a random removing pairs of source-destination addresses of the stream, that is, from a stream of pairs of addresses containing only a small fraction of randomly chosen pairs of source-destination addresses. In this case, the basic idea is that the sample will get the source addresses s, which are connected with many different addresses, appointments d.

The selection of samples from a stream of pairs of source-destination addresses is based on hashing [6] the flow of a pair of source-destination addresses. Each part of a pair of source-destination addresses is hashed using the hash function h_1 and hash values obtained for each pair of source-destination addresses are used to calculate the index location in the hash table T_1. These indices are used to check for the presence of a pair of source-destination addresses in the hash table T_1.

In the sliding window, we consider only the last N pairs of source-destination address in which the source addresses associated with a variety of assignment addresses. That is, the oldest couple will be removed upon the arrival of a new pair of source-destination addresses. This is achieved by adding a hash table T_1 timestamp n for each pair of source-destination addresses, that is, each pair stored with a timestamp, which may be the arrival time or number of the logical sequence of network packets (i.e., the triplet (s, d, n) is stored in the hash table T_1. Upon arrival, followed by a pair of source-destination addresses, the coming out of the window, a pair of timestamps is reset. In the case of the provision of several identical pairs in the current window is set to the timestamp of the last pair. With each entry of a new pair of source-destination addresses, n_curr timestamp in the hash table T_1 is used to search in pairs with a time stamp n_curr-W and is removed from the table. After that, the hash table T_2 value count, which is designed to count the number of links to various addresses, decrements by one. At the same, the only streams stored are those whose pairs of source-destination addresses (s, d) have a packet counter greater than $\epsilon * k$, the error of the actual number of unique compounds sources s, where ϵ is the error threshold. That is, for the deviation of the actual number of different connections for each source s, the counter can be $\epsilon * k$. As a result, they are considered a source address if the active number exceeds a predetermined threshold k.

In more detail, algorithm for active hosts detection, which in fact is a heavy distinct hitters finding algorithm [7, 8, 9], consists of the following steps:

Step 1. Initially hash tables T_1 and T_2 are empty. Triples (s, d, n) are stored in the hash table T_1, where n is the numbers of pairs of source-destination address (s, d) in the stream. The hash table T_2 stores the tuple $(s, count)$, where s is the source address and $count$ is a counter containing the number of different destination addresses to which are connected s. Address pairs of source-destination (s, d) receive a sequence of n (where $n = 1, 2, 3, \ldots$).

Step 2. Additions to the window of a new pair of

addresses of the source-destination (s, d). Whenever a new pair (s, d) with the number n_{curr} is added, the last pair (s, d) with the number $n_{last} = n_{curr} - W$ is removed from the window.

Step 3. The algorithm checks if the triple (s, d, n_{last}) of this pair is in the hash table T_1. If it exists, it is removed.

Step 4. The source address s is searched for in the hash table T_2 If found, then counter of its connections with different destination addresses d decrements by one. That is, the tuple $(s, count)$ (s,count) is replaced by $(s, count - 1)$.

Step 5. If the counter connection count source s equals zero, then the source will no longer appear in the sample and the tuple $(s, count)$ is removed from the hash table T_2.

Step 6. Additions to the hash table T_1 of a new pair of addresses of the source-destination(s,d). If, $h_1(s, d) \geq \epsilon * k$, the algorithm searches hash table T_1 for the tuple (s, d). If (s, d) is present, the algorithm replaces it with the triple (s, d, n_{curr}) and continues to step 2. Otherwise, the algorithms adds the triple (s, d, n_{curr}) to the hash table T_1.

Step 7. The algorithm searches for source address s in the hash table T_2. If it is not present, the algorithm resets the counter is reset to the amount that is in the hash table T_2 where $(s, 1)$ is the written deuce with source s, and first destination d. If the source s in the hash table T_2 is not present, then the value of its counter connected with different destination addresses d increments by one.

Step 8. If the source s connecting counter $count \geq k$, then source s is the active host.

Ⅲ. Conclusion

To detect worms with a permutation scan, which can be spread with great speed in computer networks, it is very important to have high-speed monitoring and analysis of network traffic in real time. However, due to the emergence of computational difficulties and problems with the storage of flow, the solution of this problem using a deterministic algorithm becomes very difficult. Therefore, monitoring network traffic requires fast streaming algorithms, which require very little memory and computational resources. Accordingly, in this document, we propose a method for detecting worms using permutation scanning as a streaming algorithm using a sliding window method, which uses very limited memory space for data storage and processing of data from a single pass, using less computing resources.

Reference

[1] C. Smith, A. Matrawy, S. Chow and B. Abdelaziz, Computer Worms: Architecture, Evasion Strategies, and Detection Mechanisms, Journal of Information Assurance and Security, 4 (2009), pp. 69-83.

[2] N.Weaver. Potential Strategies for High Speed Active Worms: A Worst case Analysis, 2002. http://www.icsi.berkeley.edu/~nweaver/worms.pdf

[3] Manna, P.K, Shigang Chen, Ranka, S.; Inside the Permutation-Scanning Worms: Propagation Modeling and Analysis, IEEE/ACM Transactions On Networking, Vol. 18, no. 3, June 2010, pp. 858-870.

[4] Mayur Datar and Rajeev Motwani. The sliding-window computation model and results. Data Streams The Kluwer International Series on Advances in Database Systems, 2007, Volume 31, pp.149-167.

[5] C. Aggarwal (editor), Data Streams: Models and Algorithms, Springer Verlag, 2007.

[6] Knuth, D. E., The Art of Computer Programming, Volume 3: Sorting and Searching, 2nd ed. Addison-Wesley, 1998. ISBN 0-201-89685-0.

[7] S. Venkataraman, D. X. Song, P. B. Gibbons, and A. Blum. New streaming algorithms for fast detection of superspreaders. In NDSS, 2005.

[8] N. Bandi, D. Agrawal, A. El Abbadi, Fast Algorithms for Heavy Distinct Hitters using Associative Memories, 7th International Conference on Distributed Computing Systems (ICDCS'07).

[9] T. Locher, Finding Heavy Distinct Hitters in Data Streams, 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), San Jose, California, USA, June 2011.