Improving the detection of On-line Vertical Port Scan in IP Traffic

Yousra Chabchoub, Christine Fricker, Philippe Robert

Improving the detection of On-line Vertical Port Scan

Improving the detection of On-line Vertical Port Scan in IP Traffic

Yousra Chabchoub , Christine Fricker and Philippe Robert,

ISEP, 21 rue d'assas 75006 Paris Email: yousra.chabchoub@isep.fr

INRIA Paris Rocquencourt, Domaine de Voluceau 78153 Le Chesnay, France

Email: christine.fricker@inria.fr

Email: philippe.robert@inria.fr

Abstract

We propose in this paper an on-line algorithm based on Bloom filters to detect port scan attacks in IP traffic. Only relevant information about destination IP addresses and destination ports are stored in two steps in a two-dimensional Bloom filter. This algorithm can be indefinitely performed on a real traffic stream thanks to a new adaptive refreshing scheme that closely follows traffic variations. It is a scalable algorithm able to deal with IP traffic at a very high bit rate thanks to the use of hashing functions over a sliding window. Moreover it does not need any a priori knowledge about traffic characteristics. When tested against real IP traffic, the proposed on-line algorithm performs well in the sense that it detects all the port scan attacks within a very short response time of only 10 seconds without any false positive.

Keywords

Bloom filter, Internet measurements, On-line algorithms, Attack detection.

Improving the detection of On-line Vertical Port Scan in IP Traffic

INTRODUCTION

Problem statement

We address in this paper the problem of designing an on-line algorithm for identifying port scan attacks in IP traffic. A port scan is a method of determining whether particular services are available on a host or a network by observing responses to connection attempts (Devivo, 1999). The received information are exploited to identify weaknesses and vulnerabilities of the host and to launch therefore more serious attacks. Several attack tools are now available and can easily be used (see (Nmap), (Foundstone) and (Nessus)) Port scan can be launched from one or several sources. In this latter case, we are dealing with *distributed attacks*, which are more difficult to detect as the contribution of each source can be considered as legitimate traffic. According to (Staniford, 2002) port scan attacks can be classified into two categories:

1. Vertical scan consisting of scanning to *big number* of destination ports for a single destination address.
2. Horizontal scan when *many* IP addresses are scanned (generally within the same subnet), on one or several ports.

Related works

Many port scan detecting methods had been developed in the literature. In (Monowar, 2010) Monowar et al. provide a survey on a large number of detection approaches. They classified them into many types (algorithmic, threshold-based, soft computing, rule-based, and visual approaches). They also established a comparison based on type, mode of detection and accuracy of the algorithms. Their main conclusion is that methods combining data mining and threshold-based analysis are the most efficient in terms of false positive rates, scalability and robustness. Most of detection approaches are single source and can not detect distributed attacks (e.g. (Robertson, 2003) , (Roesch, 1999)). Very few methods can be applied on-line and give real time response. Moreover a common weakness of port scan detecting methods, particularly the threshold-based methods, is that their accuracy is closely dependent on traffic characteristics. Quite often, algorithms depend on constants directly related to the traffic intensity. For a limited set of traces, they can be tuned by hand to get reasonable performances. This procedure is, however, not acceptable in the context of an operational network. As a general requirement, it is highly desirable that the constants used by algorithms automatically adapt, as simply as possible, to varying traffic conditions. As an example, in (Heberlein, 1990) Heberlein et al. propose an IDS (Intrusion Detection System) where a source is considered as malicious if it contacts more than 15 other IP addresses in a given time window.

In this paper we focus on vertical port scan where many ports are scanned for a given destination address. Our objective is to design an adaptive algorithm that detects on-line this kind of attacks. The algorithm will be deployed on operators IP backbone network carrying traffic at a

very high bit rate. At this level one can have an aggregated view of the global traffic (generated by many hosts) which is very useful to detect attacks. Moreover at this level, network operator can stop the attack to avoid its propagation until the destination and therefore save network resources. An on-line analysis of IP traffic in the core network is a challenging issue. The algorithm has to perform a very quick data processing and to store only relevant information. Data analysis should be faster than the arrival rate of data stream of a real-time execution. So the processing time of a packet has to be lower than the inter-arrival packets time which is of the order of few nanoseconds. This is typically a context of data mining.

A natural solution to cope with the huge amount of data in IP traffic is to use hash tables. A data structure using hash tables, a Bloom filter, proposed by B. Bloom (Bloom, 1970) in 1970, has been used to test whether an element is a member of a given set. Bloom filters have been used in various domains : In (McIlroy, 1982) and (Mullin & Margoliash, 1990) they are used to represent words of a dictionary with a small memory. They make the search in the dictionary much faster. Bloom filters are also very useful for some distributed database applications. With a simple representation of a table content in a Bloom filter, we can speed up significantly semi-join operations and reduce the overhead communication between machines as they only exchange Bloom filters (see (Bratbergsengen, 1984) and (Mullin, 1990)). Many distributed network applications rely on Bloom filters to improve their performance. We can for example mention the distributed web cache sharing proposed by Fan et al. in (Fan, 2000) or the detecting loops algorithm introduced by Whitaker and Wetherall (Whitaker, 2002), in the context of packet switching. Bloom filters are well adapted to many other applications such as multicast (Gronvall, 2002) resource routing in peer-to-peer networks (Druschel, 2001), (Stoica, 2001) and queue management (Dilip, 2001).
Bloom filters have been used by Estan and Varghese (Estan, 2002) to detect large flows. A Bloom filter consists of k tables of counters indexed by k hash functions. The general principle is the following : for each table, the flow ID of a given packet (that is the addresses and port numbers of the source and the destination) is hashed onto some entry and the corresponding counter is incremented by 1. Ideally, as soon as a counter exceeds the value C, it should be concluded that the corresponding flow has more than C packets. Unfortunately, since there is a huge number of small flows, it is very likely for instance that a significant fraction (i.e. more than C for example) of them will have the same entry, incrementing the same counter, thereby creating a false large flow. To avoid this problem, Estan and Varghese (Estan, 2002) propose to periodically erase all counters. Without any a priori knowledge on traffic (intensity, flow arriving rate, etc.), which is usually the case in practical situations, the erasure frequency can be either

1. too low, and, in this case, the filters can be saturated : Because of the large number of small flows, many of them may be hashed on the same entry of the hash table and, therefore, the corresponding counter is increased accordingly, and consequently creating a false large flow.

2. too high and a significant fraction of large flows can be missed in this case : Indeed, the value of the counter of a given entry corresponding to a large flow with a low throughput may not reach the value C if the value of this entry is set to 0 too often.

The efficiency of the algorithm is therefore highly dependent on the period Δ of the erasure mechanism of the filters. This quantity is clearly related to the traffic intensity.

Organization of the paper

Starting from Estan and Varghese's algorithm, we propose an adaptation to the context of port scan attacks. An algorithm based on Bloom filter with a new adaptive refreshment scheme is proposed. The organization of this paper is as follows: A detailed description of the algorithm detecting port scan is given in the next section. The algorithm proposed is tested against experimental data collected from IP backbone network in the third Section. Concluding remarks are presented in the last section.

ALGORITHMS WITH BLOOM FILTER

In this section, we describe our new algorithm designed to identify on-line port scan attacks in the Internet IP traffic. According to the port scan definition given above, the number of different destination ports per destination address is an excellent observable to control in order to detect this kind of attacks. That is why the proposed algorithm is based on an on-line counting of the number of distinct destination ports used for each destination address.

Let us first define the *flow* as the set of the those packets with the same destination address. The *flow size* is then naturally defined as the number of different destination ports used during a given *time window* Δ. Hence the objective of the algorithm is to identify flows which get "much" larger than the others.

The algorithm is divided into two parts:

1. The counting mechanism: In this part IP traffic is filtered and only significant observables for port scan detection are stored.
2. The decisional module: The objective of this part is to detect automatically suspicious behavior in the filtered data derived from the counting mechanism.

The problem of on-line counting the number of distinct elements (cardinality) in a given set has been already addressed in the literature. One can mention the Probabilistic counting and the Hyperloglog algorithms proposed by Flajolet et al. (Flajolet, 2007), (Flajolet, 1985) or the minCount algorithm of Giroire and Fusy (Giroire, 2007). These are very efficient algorithms in term of memory usage and speed. Unfortunately, they are not adapted to our context as one has to run them for each destination address, which is unrealistic given the potential number of destination addresses, especially with the transition to IPv6.

The specificity of the port scan detection problem is that the counting mechanism has to be performed for each destination address. This brings an additional difficulty compared to the classical counting problems. The proposed solution is based on an improvement of Bloom filter that takes into account this new challenge.

Counting with Bloom filter

The starting point is the algorithm based on Bloom filters designed by Estan and Varghese (Estan, 2002). The objective of this algorithm is to identify flows with more than *C* packets, called large flows. Notice that flows size is in this context simply defined as its total packets number. The filter, see Figure 1, consists of *k* stages. Each stage *i* contains *m* counters taking values from 0 to *C*. It is assumed that *k* independent hash functions $h_1, h_2 \ldots, h_k$ are available. The total size of the memory used for the filter is denoted by *M*, recall that *M* should be of the order of several Mega-Bytes. An additional auxiliary memory is used to store the identifiers of detected large flows.

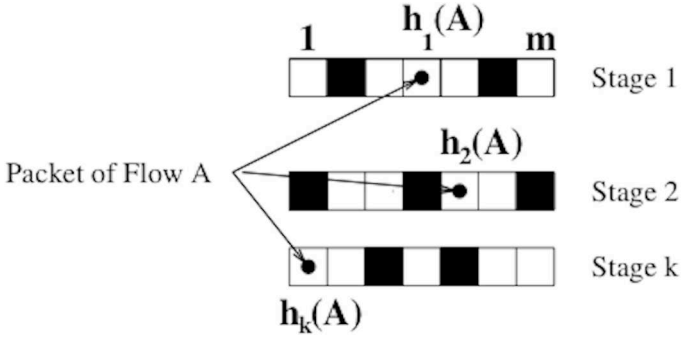

Figure 1. A Bloom filter

The algorithm works as follows: All counters are initially set equal to 0; if a packet belonging to a flow *A* is received then:

- If *A* is already registered in the memory storing large flows IDs then next packet is considered.
- If not, let *min(A)* the minimum value of the counters at the entries $h_1(A), \ldots, h_k(A)$ of the *k* hash tables.
    - If *(min(A) < C)*, all the corresponding counters having the value *min(A)* are incremented by 1.
    - If *(min(A) = C),* the flow of A is added to the memory storing large flows IDs. The flow is detected as a large flow.

The algorithm as such is of course not complete since small flows can be mapped repeatedly to the same entries and create false large flows. This is the classical problem of collisions in the context of Bloom filters. One has therefore to clear the filters from the influence of these undesirable flows. Estan and Varghese (Estan, 2002) proposed to erase all counters of the filters on a periodic basis (5 seconds in their paper):

Estan and Varghese's refreshing mechanism:

- Every Δ time units all counters are re-initialized to 0.

Ideally, the constant Δ should be directly related to the traffic intensity. On the one hand, if the refreshment mechanism occurs frequently, the counters corresponding to real large flows are

decreased too often and a significant fraction of them may not reach the value *C* and therefore many large flows will be missed. On the other hand, if Δ is too large then, because of their huge number, small flows may be mapped onto the same entry and would increase the corresponding counter to the value *C*, creating a false large flow. This periodic refreshing mechanism could perfectly work if there would be a way to change the value of Δ according to the order of magnitude of the number of small flows. Such a scheme is however not easy to implement in practice.

In (Chabchoub, 2009), a Bloom filter associated to *k* hash functions is used to identify large flows, in a different way: For each received packet, only one counter, chosen at random among the smallest counters, is incremented by 1. A new refreshing mechanism is also introduced. It consists of decrementing periodically all positive counters by one. The objective is to eliminate progressively small flows without damaging large ones. In fact the bound between small and large flows is very thin: more or less than *C* packets.
In the context of attacks detection, we aim intuitively to erase all flows related to standard traffic in order to keep only very large flows that are very likely to correspond to some attacks. As there is a significant difference between these two kinds of flows, we can apply a more aggressive refreshing mechanism on the filter.

We propose in the next subsection a new adaptive refreshing mechanism that needs no a priori knowledge about traffic characteristics and that automatically varies according to traffic intensity.

Estan and Varghese's algorithm cannot be directly applied to the port scan detection problem. In fact, for a given flow, only packets having a new destination port number should be counted (cf flow size definition given before). This means that, for each flow one has to store already used destination port numbers.

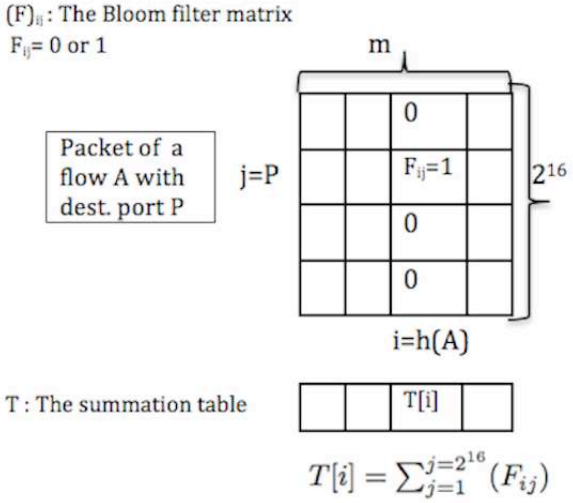

Figure 2. Counting flows sizes with a Bloom filter matrix

Thus the filter is modified as follows, see Figure 2:

The filter $(F)_{ij}$ consists of $m \times 2^{16}$ matrix of bits where $F_{ij} \in \{0,1\}$. A table T of m counters is also added. T_i maintains the number of distinct destination ports per column: $T_i = \sum_{j=1}^{2^{16}} F_{ij}$.

The counting mechanism is performed as follows: All counters are initially equal to 0. If a packet belonging to a flow A, with a destination port P is received then $(F)_{ij}$ and T are adapted in the following way:

- The cell F_{ij}, where $i = h(A)$ and $j = P$, is considered. If $(F_{ij} = 0)$ then
  - $F_{ij} = 1$
  - $T_i = T_i + 1$

Notice that Estan and Varghese (Estan, 2002) algorithm uses *k* hashing functions in order to reduce the impact of collisions consisting of mapping several flows to the same counter via a given hashing function. Consequently this counter over estimates flows size. In the context of the port scan detection algorithm, the size of a flow *A* can be over estimated if and only if these two conditions are satisfied:

1. *A* collides with an other flow *B*; *(h(A) = h(B))*
2. *A* and *B* have different destination ports

In practice this is an unusual situation as the commonly used destination ports are not very numerous, even if the number of different ports can theoretically reach 65,536 as a port number is encoded on two bytes. Moreover, for the port scan detection problem one does not focus on an exact estimation of flows sizes but only on identifying very large flows. So unlike for Estan and Varghese, this new context is much less sensitive to the problem of collisions. That is why we prefer saving memory by using only one hash function in our algorithm.

Adaptive Refreshing Mechanism

The presented algorithm cannot run indefinitely. In fact flows have a limited lifetime and ideally old flows have to be erased from the filter to keep an efficient counting mechanism. So the filter needs to be sometimes refreshed. The general principle is the following: If the state of filter is declared as overloaded then all the cells in the filter $(F)_{ij}$ and in the summation table *T* are reinitialized to 0.

The following "RATIO" criterion is proposed to declare the state of the filter as overloaded:
Let us define *r* as the proportion of non-null counters in the summation table *T*; the filter is overloaded when *r* is above some threshold *R*. This refreshing mechanism is clearly adaptive as its frequency only depends on traffic variations: As long as the state of the filter is not overloaded, nothing occurs and if there is a peak of activity, the filter gets quite quickly filled and the refreshment mechanism is automatically executed. More precisely, as *R* is computed on the table *T* and not globally for the filter, the peak of activity consists of a big number of destination addresses received in a short time window. Notice that the occurrence of a vertical

port scan (where only one destination address is involved) has no impact on the refreshing frequency. This is very important to could detect the attack.

The key parameter of this refreshing mechanism is the filling up threshold R. It is a very sensitive parameter for the algorithm. For a fixed m (size of the summation table T), there is clearly a trade-off in the choice of the value of R: On the one hand, with a small R, collisions will be minimized so the counting mechanism will be more accurate. But on the hand, if we refresh the filter too frequently (lets say every second), the impact of a port scan attack on the traffic characteristics can not be seen as the attack will be fractionated over many sliding windows. An intermediate solution is to fix R in an initialization step of the algorithm as follows: R is the filling up rate reached after a given duration Δ. It is a variable threshold depending on the traffic type considered. In practice Δ is set to one minute, which is a reasonable duration given that a port scan attack lasts several minutes.

The attacks detection mechanism

As explained above, the adaptive refreshing mechanism proposed gives an estimation of the number of different destination ports for each destination address over a sliding window of one minute. Recall that this observable is defined in the second section as flows size. The objective of the attacks detection mechanism is to identify "large" flows. The term "large" has to be properly defined. Roughly speaking, this means that such a flow is much larger than the other "normal" flows. Again, because of the variation of traffic, an adaptive scheme has to be devised to properly define these concepts.

The main idea of the algorithm is to evaluate a varying average m_n of the largest flow in several sliding time windows of length Δ. The quantity m_n describes "normal flows"; it is periodically updated in order to adapt to varying traffic conditions. It is a weighted average that takes into account all its past values to follow carefully traffic variations but not too closely. If a flow in the n^{th} time window is much larger than m_{n-1}, it is considered as an attack, and the moving average is not updated for this time window.

The following variables are used:

- As before, r is the proportion of non-zero counters in the summation table T.
- S is a multiplicative detection threshold. Roughly speaking, an attack is declared when an observation is S times greater than the "normal" behaviour. The value of S is fixed by the administrator.
- R_s and R are thresholds for the variable r. The constant R_s is independent of traffic and taken once and for all equal to 90% and R is a variable threshold depending on the traffic type considered.
- α is the updating coefficient for averages; $\alpha = 0.85$ in our experiments.
- Δ is the duration of the initialization phase (1 minute in the paper). It is in fact a bound for the time before which an attack should be detected.
- m_n is the weighted moving average for the n^{th} time window.

Initialization phase:

- All counters in the filter matrix $(F)_{ij}$ and the summation table T are set to 0.
- $(F)_{ij}$ and T are progressively updated when packets are received

  After a duration Δ, evaluate the variable r
  - If $r \times R_s$ then $R := r$ else $R := R_s$.
  - Find the highest counter in T: $m_1 := max_i T[i]$.

Detection phase: the n^{th} time window

- At the beginning $(F)_{ij}$ and T are initialized to 0.
- The Bloom filter is progressively updated with packets arrivals by using their destination address and destination port.

  If a counter exceeds Sm_{n-1}, an attack is declared.

  If $r \times R$
  - Find the highest counter in T: $\max_n$.
  - If $\max_n < S\, m_{n-1}$

    $m_n = \alpha\, m_{n-1} + (1-\alpha)\max_n$
  - Start the $(n+1)$th time window.

Table 1. Algorithm for port scan detection

The algorithm starts with an initialization phase of length Δ in order to evaluate the threshold *R*. At the end of this phase, *R* will be definitively fixed for the rest of the experiment. In addition, as this phase corresponds to the first time window, the moving average m_1 will be initialized as the biggest counter obtained. We implicitly assume that there is no attack during the initialization phase. See Table 1 for the description of the algorithm.

Note that an alarm is declared during the n^{th} time window as soon as the value of a counter becomes greater than Sm_n. In this case the moving weighted average m_n is not updated, it keeps its old value, computed in the previous time windows. Both attacker and victim can be identified as the attack is always raised by the last performed packet and we can simply store and update the IP header of the last received packet.

At the beginning, the first time window is fixed (its duration is Δ) but, since the evolution depends on the occupation rate of the summation table *T*, the duration of the other time windows is variable. If traffic characteristics are not much varying, time windows duration remain around one minute. In this case, an attack is detected at the latest after one minute so that the network administrator can react quickly.

EXPERIMENTAL RESULTS

Dataset

To evaluate and validate the attack detection algorithm described in the previous section, we run experiments on a traffic trace captured in the IP backbone network of Orange Labs in December 2007 in the context of the ANR-RNRT OSCAR project. This traffic trace contains some port scan attacks. Its global characteristics are given in Table 2.

Nb. IP packets	Duration	Nb. Flows
32.10^6	67 minutes	250.10^3

Table 2. Characteristics of the traffic used for attack detection

A simple analysis of the trace shows that the global characteristics of the traffic (rate, number of flows per minute) do not vary too much (see Figure 3 and 4). Thus Port scan attacks have no impact on the global traffic parameters.

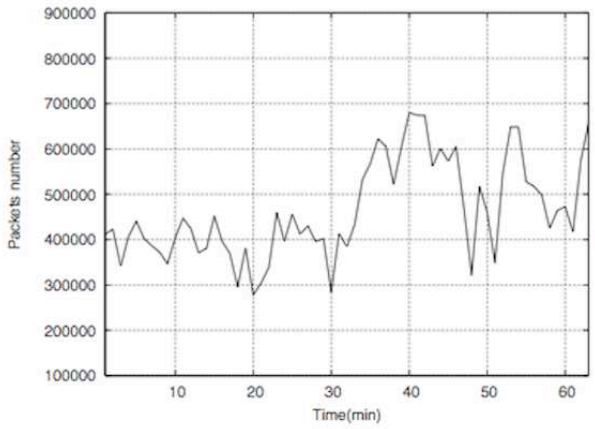

Figure 3. Total packets number per minute

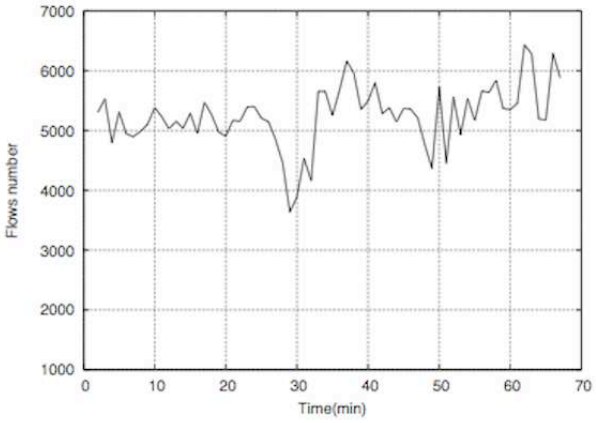

Figure 4. Total flows number per minute

More specific experiments at flow level are performed in Figure 5 to identify port scan attacks. Recall that a flow is defined as a set of packets with the same destination address. Figure 5 shows the real size of the largest flow (i.e. the number of its different destination ports) in a sliding window of one minute. Notice that these experiments are done off-line. They are based on an exhaustive exact counting, which is unrealistic in an on-line context. According to this figure, the largest flow has about 150 different destination ports. This size has also a small variance. But one can clearly detect two peaks, which correspond to two port scan attacks. They

occur at the 27th and 58th minute. The attacked address is 10.0.0.3 in the two cases. This address has up to 6000 different destination ports per minute. The used destination ports are mainly composed of the so-called in (Devivo, 1999) Well Known Ports (0-1023) and some Registered Ports (1024-49151).

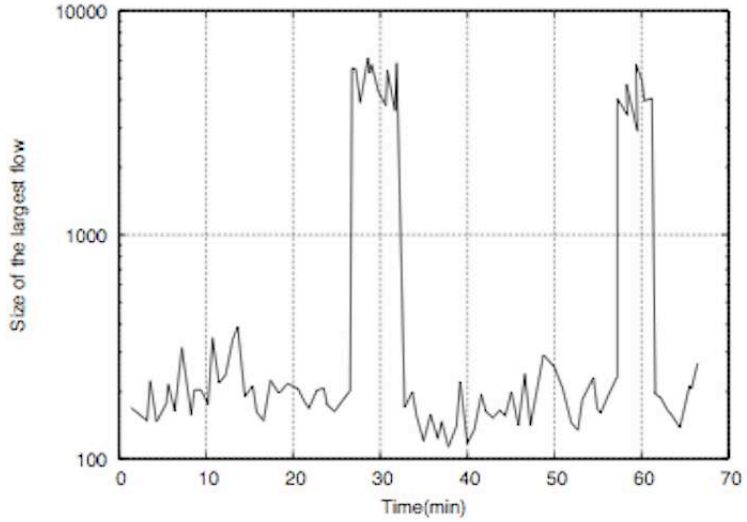

Figure 5. Size of the largest flow per minute (2 port scan attacks against 10.0.0.3)

Flow experiments and performance analysis

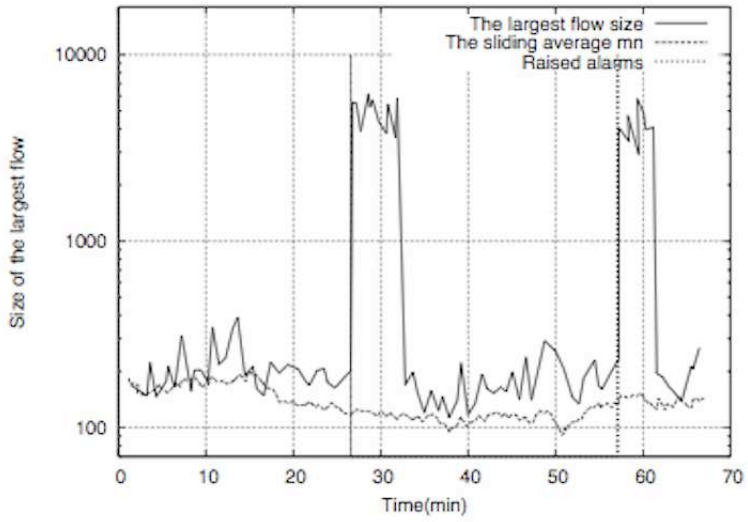

Figure 6. Detected attacks with m =1024, S =2

Figure 6 shows the results of the proposed algorithm on the traffic trace. The size *m* of the summation table *T* is taken equal to 1024. So the total size of the Bloom filter matrix is equal to 67 MB. After the initialization phase of one minute, the filling up reached 47%, and the parameter *R* is fixed to this value. The multiplicative detection threshold *S* is set to 2. The choice

of this value is mainly based on the small variance of the largest flow size showed before. One can see in Figure 6 that the moving weighted average m_n is always close to the real size of the largest flow except in case of attacks. In fact when attacks occur, m_n is not updated for the current time window and all packets addressed to the attacked destination are no more inserted into the Bloom filter until the end of the attack. This avoids overloading the filter with useless information. The moving average m_n is therefore updated in the following window. Thus the attack has no impact on m_n and the algorithm is able to deal with several simultaneous attacks.

Moreover our algorithm is very reactive and has a very short response time as it detects the attacks in the first 10 seconds. Let us recall that the algorithm raises an alarm as soon as a counter in T becomes too high compared to m_n. It means that is does not wait the end of the time window to declare the attack.

Notice finally that this algorithm can be easily implemented on a core router as its total required memory is less than 100MB and it is well adapted to an on-line analysis. The counting mechanism used in the proposed algorithm gives an estimation of the size of the largest flow. More precisely, it can over estimate the real size because of collisions in the bloom filter. As the objective is the identification of very large flows, the counting mechanism has not to be "too" accurate. But the overestimation can also lead to some false positives (false raised alarms) beyond a given limit. The efficiency of the counting mechanism depends closely on the filling up threshold R.

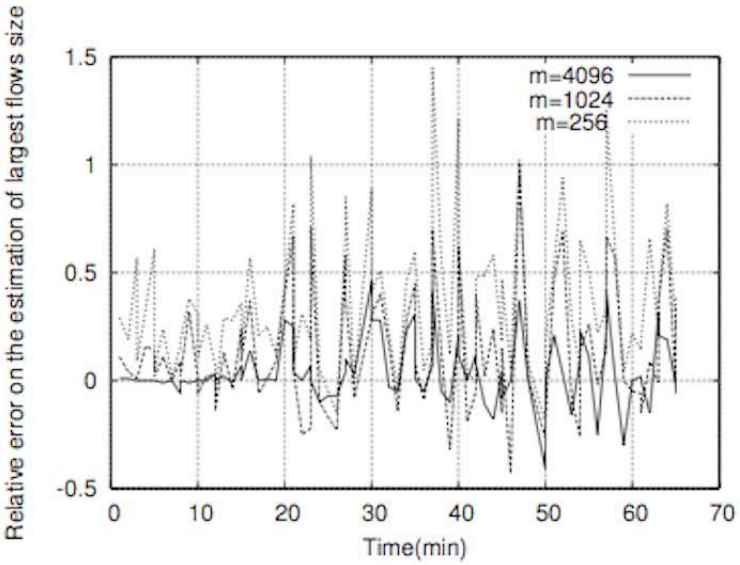

Figure 7. Impact of m on the counting mechanism accuracy

In Figure 7 the relative error on the estimation of the largest flow is plotted for different values of R. The value of R is automatically fixed after the initialization phase of one minute. Therefore it only depends on the size of the filter m. Three values of m are tested (256, 1024 and 4096). The algorithm gives no false positives in the three cases. It is clear that the estimation is more accurate for small values of R. But, a static highest filling up threshold R_s must be defined to limit the impact of the over estimation. R_s is taken to 90%.

The proposed algorithm can also deal with slow port scan attacks. In this case the attack is performed progressively by adding a small number of destination ports each time (see (Dabbagh, 2011) for more details). The weakness of the threshold-based algorithms is that they can miss the attack if the attacker is just below the threshold. To detect slow port scan attacks, one has to consider a larger time scale and a more aggregated traffic. More precisely, two Bloom filters will be used and two time windows will be considered: Δ_1 = 1 minute and Δ_2 = 5 minutes. The algorithm will be performed separately for these two Bloom filters which are totally independent. Each packet will be registered twice in the two filters, and slow attacks will be only identified using Δ_2. The required resources (memory and processing capacity) are obviously doubled but the algorithm can still be implemented on a standard router for an online analysis.

Remark on threshold

The algorithm uses the variable *S,* called the multiplicative detection threshold. It is related to the network administrator's decision about the precise definition of an anomalous behaviour. In the experiments, there is a set of events which will be qualified as "attacks" for a smaller values of *S*. Note however that, for some large but "milder" variations, the qualification as attack will depend on the particular value of this parameter. There is no way to avoid this situation in our view. This is the role of the administrator to define the level of abnormality in traffic.

CONCLUDING REMARKS

We have presented in this paper a novel adaptive algorithm for identifying vertical port scan attacks in Internet traffic. A new approach using a two-dimensional Bloom filter is developed. Unlike Estan and Varghese approach where the filter is periodically erased, we propose a new original "RATIO" criteria to refresh the filter according to traffic intensity. The proposed algorithm has been successfully tested against a real traffic trace (captured in the IP backbone network of orange Labs) containing some port scan attacks.

References

Bloom, B. (1970) Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, vol. 13(7), pp. 422–426.

Bratbergsengen, K. (1984) Hashing methods and relational algebra operations. *In Proceedings of the Tenth International Conference on Very Large Data Base*, pp. 323-333.

Chabchoub, Y., Fricker, C. & Mohamed, H. (2009): Analysis of a Bloom Filter algorithm via the supermarket model. *International Teletraffic Congress*

Dabbagh, M., Ghandour, A., Fawaz, K., Hajj, W., & Hajj, H. (2011) Slow port scanning detection. *In Proceedings of the 7th conference on Information Assurance and Security (IAS)*, pp. 128-133.

Devivo, M., Carrasco, E., Isern, G., & Devivo, G. O. (1999). A review of port scanning techniques. *SIGCOMM Computer Communications Review*

Dilip, D. K., Feng, W.C. Shin, K. G. & Saha, D. (2001) Stochastic fair blue: A queue management algorithm for enforcing fairness. *In Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM-01)*, vol. 3, p. 1520-1529.

Druschel, P. & Rowstron, A. (2001) Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. *In Proceedings of the Eighteenth ACM Symposium on Operations Systems Principles*, p. 188-201.

Estan, C. & Varghese, G. (2002) New directions in traffic measurement and accounting. *In Proceedings of the ACM 2002 SIGCOMM*

Fan, J. A. L., Cao, P., & Broder, A. Z. (2000) Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, vol. 8(3), p. 281-293.

Flajolet, P., Fusy, E., Gandouet, O. & Meunier, F. (2007) Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *In Proceedings of the 13th conference on analysis of algorithm (AofA 07)*, pp. 127-146.

Flajolet P & Martin, N. (1985) Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, pp. 182-209.

Foundstone, a division of mcafee superscan from http://www.foundstone.com/us/resources/proddesc/superscan.htm.

Giroire, F. & Fusy, E. (2007) Estimating the number of active flows in a data stream over a sliding window. *In Proceedings of the Fourth Workshop on Analytic Algorithmics and Combinatorics (ANALC0)*, D. Applegate, Ed. New Orleans: SIAM, pp. 223-231.

Gronvall, B. (2002) Scalable multicast forwarding. Poster in *the ACM 2002 SIGCOMM Conference*.

Heberlein, T., Dias, G., Levitt, K., Mukherjee, B., Wood, J., & Wolber, D. (1990). A network security monitor. *Proceedings of RISP90.*

McIlroy, M. D. (1982) Development of a spelling list. *IEEE Transactions on Communications*, vol. 30(1), pp. 91–99.

Monowar, H., Bhattacharyya, D. K., & Kalita, J. K. (2010). Surveying port scans and their detection methodologies. *Computer Journal*.

Mullin, J. K. (1990) Optimal semijoins for distributed database systems. *IEEE Transactions on Software Engineering,* vol. 16(5).

Mullin, J. K., & Margoliash, D. J. (1990) A tale of three spelling checkers. *Software Practice and Experience*, vol. 20(6), pp. 625-630.

Nessus: Tenable network security inc. Columbia. nessus. http://www.nessus.org/nessus/.

Nmap, from http://nmap.org

Robertson, S., Siegel, E.V., Miller, M., & Stolfo, S. J. (2003). Surveillance detection in high bandwidth environments. *In proceeding of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX-III)*.

Roesch, M. (1999) Snort-lightweight intrusion detection for networks. *In Proceedings of LISA99*, Seattle, WA, USA.

Staniford, S., Hoagland, J. A., & McAlerney, J. M. (2002). Practical automated detection of stealthy portscans. *Journal of. Computer. Security.* , vol. 10, pp. 105-136.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001) Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review (Proceedings of the ACM 2001 SIGCOMM Conference),* vol. 31(4), p. 149-160.

Whitaker, A. & Wetherall, D. (2002) Forwarding without loops in Icarus. *In Proceedings of the fifth IEEE Conference on Open Architectures and Network Programming (OPENARCH)*, p. 63-75.