Defending against a Denial-of-Service Attack on TCP

Pars Mutaf
pars@likya.iyte.edu.tr

Department of Computer Engineering
Izmir Institute of Technology
Gaziosmanpasa Blv. No. 16 Cankaya Izmir 35230 Turkey

Abstract

In this paper we propose a real-time anomaly detection method for detecting TCP SYN-flooding attacks. This method is based on the intensities of SYN segments which are measured on a network monitoring machine, in real-time. In the currently available solutions we note several important flaws such as the possibility of denying access to legitimate clients and/or causing service degradation at the potential target machines, therefore we aim to minimize such unwanted effects by acting only when it is necessary to do so: during an attack.

In order to force the attackers to fall in a detectable region (hence, avoid false negatives) and determine the actual level of threat we are facing we also profit from a series of host based measures such as tuning TCP backlog queue lengths of our servers. Experience showed that complete avoidance from false positives is not possible with this method, however a significant decrease can be reasonably expected. Nevertheless, this requires an acceptable model for the legitimate use of services. We first explain why the Poisson model would fail in modeling TCP connection arrivals for our purpose and show that analyzing daily maximum arrival rates can be suitable for minimizing false positive probabilities.

This method can allow ISPs to determine their correct requirements to cope with this particular attack and provide more secure services to their clients.

1 Introduction

The Internet has undergone a phenomenal growth in the recent past. However, during this period the vulnerabilities found in the TCP/IP protocol suite have been subjected to significant revelation as well. Particularly, the details of a simple denial-of-service attack popularly known as “SYN-flooding” were published in two underground magazines and this attack still continues to pose a serious threat against the availability of TCP services[11]. The SYN-flooding attack exploits a common TCP implementation issue and a well-known authentication weakness found in IP, which do not seem correctable in the near future since they require the modification of the standards.

Preventive approaches such access control, are not applicable to SYN-flooding attacks since the general target is public services. Therefore, we propose a method for detecting these attacks in real-time and recover from the damage as soon as possible and in a convenient way. Network monitors are known to be able to detect such low-level network based attacks, therefore we implement this method on a network monitoring machine. The method that we propose falls in the anomaly detection category of intrusion detection systems. However, we also profit from a series of host based measures in order to force an attacker to fall in a detectable region.

The rest of this paper is organized as follows: Section 2 describes background material such as the IP and TCP protocols as well as the vulnerabilities found in these protocols which are exploited by the SYN-flooding attack, Section 3 overviews its current solutions, Section 4 explains our objectives, Sections 5-10 give the details of our approach, Section 11 proposes future work and Section 12 presents several conclusions.

2 Background

The Internet is a worldwide network that uses the TCP/IP (Transmission Control Protocol/Internet Protocol) protocol suite for communications.

IP[6] is the standard internet layer protocol of TCP/IP, which provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. IP is a connectionless protocol. Therefore, IP datagrams may get delivered out of order and there is no guarantee that a datagram successfully gets its destination. IP does not either provide address authentication. Actually, any host can send datagrams with any source IP address[2]. Therein lies most of the threat against the integrity, secrecy and availability of today’s Internet assets.

TCP[7] is the connection oriented transport layer protocol of the TCP/IP suite, designed to provide a reliable logical circuit between pairs of applications in hosts attached to the Internet. TCP assumes that it can obtain an unreliable datagram service from lower level protocols. In the Internet, this service is provided by IP. The primary purpose of TCP is to provide a reliable connection service on top of a less reliable internet communication system. For this, TCP supports facilities in the following areas: reliability, flow control, multiplexing and connections.

In order to support reliability and flow control, TCP initializes and maintains certain status information for each data stream. The combination of this information including sockets, sequence numbers, and window sizes, is called a connection. A pair of socket (4 tuple consisting of the client IP address, client port number, server IP address and server port number) specifies the two end points that uniquely identifies each TCP connection in the Internet. A TCP packet is called a “segment”.

For a connection to be established, the two TCPs must synchronize on each other’s sequence numbers. This is done by exchanging connection establishing segments carrying a SYN control bit and initial sequence numbers (ISNs). The synchronization requires each side to send it's own ISN and to receive an acknowledgment of it from the other side. Each side must also receive the other side's ISN and send an acknowledgment (ACK). This is illustrated in Figure 1.

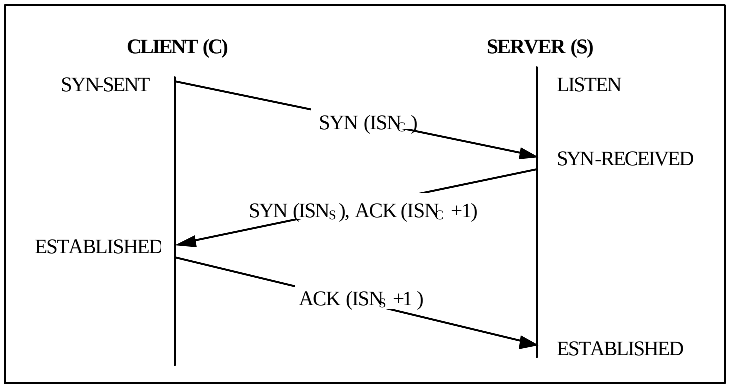


Figure 1. TCP 3-Way Handshake

A three-way handshake is necessary because the client and server sequence numbers are not tied to a global clock in the network, and TCPs may have different mechanisms for picking ISNs. The server which receives the first SYN has no way of knowing whether the segment is an old delayed or not, and thus it must ask the sender to verify this SYN.

TCP servers are concurrent. A TCP server starts a new process to handle each client therefore the listening server is always ready to handle the next coming connection request. However, there is still a chance that multiple connection requests arrive while the server starts a new process. In order to handle these incoming connection requests while the listening application is busy TCP employs a fixed length queue for the connections that have not been accepted by the server application. This is referred to as the "backlog queue". However, there is an upper limit to the number of connection requests waiting in this queue. This limit is specified by the listening application by calling the `listen()` system call.

When a new connection request arrives (i.e., a SYN segment), TCP acknowledges this if there is room for this new connection on the requested service's queue. However it should be noted that the server application will not see these new connection until the third segment of the 3-way handshake is received. If there is no room on the requested service's queue, TCP ignores the received SYN packet. By ignoring the SYN packet the server forces the client to retransmit the SYN later, hoping that the queue will then have room[9].

The SYN-Flooding attack exploits both this design and the authentication weakness in IP. The attacker generates several SYN segments with invalid source IP addresses to a target TCP server. Since the source hosts are non-existing or closed, the 3-way handshake for these connections will never complete or be broken (a valid source host would certainly reset such an unreferenced connection by sending a RST segment), resulting in several half-open connections filling up the server's backlog queue. Then, the target service becomes unavailable (interrupted) until a connection establishment timer expires. Most implementations limit the SYN-RECEIVED state to 75 seconds[1].

3 Current Solutions

In order to immunize servers against SYN-flooding attacks, two different host based measures are generally proposed: decreasing the SYN-RECEIVED state timeout value and increasing the backlog queue limit. As mentioned above, the timeout value associated with the SYN-RECEIVED state is generally 75 seconds. Decreasing this value will reduce the time a half-open connection will occupy the backlog queue, therefore the duration of denial-of-service after the attack. However, a timeout value set too short will increase the risk of aborting legitimate connections over low-speed paths. As for the second choice, which is increasing the backlog queue limit, it depends on physical memory capacities. Each entry in the backlog queue will allocate an amount of memory which may be different for different implementations of TCP. The higher the chosen queue length, the more physical memory resources will be allocated in the case of an attack. One vendor proposes a backlog queue length of 8,192 in order to be able to cope with SYN-flooding attacks[14]. However, we note that the upper limit to the length of the backlog queue is generally loosely defined and it is not generally guaranteed that a given length will suffice in all situations.

Firewall approaches do also exist which are already implemented by several vendors[12,13]. Such an approach requires a circuit-level gateway, which accepts the connection requests on behalf of the server. Once the connection is successfully established, the gateway acts as a relay between the client and server. The important advantage of this scheme is that in the case of a SYN-Flooding attack, the target server never sees the attacker's packets. The gateway simply discards incomplete connection requests. However, the gateway must be immune to SYN-Flooding. In addition, with this method, new delays are introduced to legitimate clients. This is a generic problem found in firewalls.

The "semi-transparent gateway" approach as called by Schuba et al[8], lets the connection requests pass through, however artificially completes each connection request. When a SYN segment destined for an internal host is observed, the gateway waits for the server's answer (a segment containing a SYN and ACK), then reacts by generating the necessary ACK segment on behalf of the client. This completes the 3-way handshake. If the client is legitimate, it sends its own ACK segment, resulting in a duplicate ACK. The client's ACK is silently discarded by the server, the connection is already established and data flows in both directions. In the case of an attack, the client's ACK is never seen, then (after a timeout period) the gateway sends a RST packet in order to close the connection. The obvious advantage of this scheme is that the gateway does not cause any delay to legitimate connections. However, the timeout period before breaking incorrect connections is loosely defined and a small timeout period denies access to legitimate clients. Furthermore, in the case of a long timeout period, a flood of SYNs results in a large number (undefined) of established connections at the target hosts, which represents extra loads for these hosts. Schuba et al[8] call this situation as "service degradation".

A more comprehensive solution proposed by Schuba et al[8] is based on the classification of source IP addresses. The tool implementing this method is called `Synkill` and operates on a network monitoring machine. Regarding their network activities, the `Synkill` algorithm classifies the source IP addresses as: never seen (*null*), correctly behaving (*good*), potentially spoofed (*new*), and most certainly spoofed (*bad*). Addresses that are administratively configured as *good* (*bad*) are called *perfect*(*evil*). Examples for *evil* addresses are private networks 10.0.0.0, 172.16.0.0 and 192.168.0.0, which should not appear as source addresses outside their networks. Synkill can also operate as a state machine, which allows addresses (except the *perfect* and *evil* addresses) to be moved to different classes regarding their observed behaviours. Synkill immediately resets the connection attempts from *evil* or impossible addresses (such as 0.0.0.0 and 127.0.0.0) and takes one of the two possible actions mentioned above for the connection attempts from *bad* addresses: directly sending RST or immediately completing the connection and sending RST after again a short timeout period. The main disadvantage of this approach is the possibility of breaking legitimate connections in the *new* state. Another important problem arises when the attacker's IP addresses does not repeat. In this case Synkill can not use the information contained in its database and state machine. During the tests of Synkill, the authors observed considerable service degradation when attack packets contained different source IP addresses, therefore they suggest the combination of this method with the host based measures mentioned above.

4 Objectives

Our objective is the detection of a SYN-flooding attack in real-time and each time we detect an attack we want to be able to define the level of threat we are facing and act accordingly. We note here that none of the above methods were designed in this sense. In stead, they were designed to analyze the correctness of each TCP connection and act for them separately, regardless of the existence of a considerable threat. However, being able to differentiate an attack from the normal mode of operation will have considerable advantages.

The common flaw that we note in the methods mentioned above is early timeout expiries, which cause denial-of-service to legitimate clients suffering from low bandwidth. A long timeout period causes service degradation at the server as opposed to a short one that results in denial-of-service to particular legitimate clients. Currently, the common policy about this dilemma is: “clients already suffering from low quality of service should not victimize others”, therefore short timeout periods are generally preferred. One vendor calls this an “aggressive timeout”. Our statement is that the aggressive timeouts should not be employed when there is no obvious reasons to do so. In today’s Internet not every user has the chance to connect via high-speed paths, nor there is a guarantee that the others will have the same level of quality of service twenty-four hours a day. We note here, the importance of detecting a SYN-flooding attack, which will provide us the ability of employing aggressive timeouts only when it is necessary: during an attack.

To simplify the discussion, we first consider the protection of only one host: ephesus[*] which is one of the Internet servers of Izmir Institute of Technology and in order to detect a SYN-flooding attack destined for this host we use a network monitoring machine attached to the same physical network. This

[*] The name of the host has been changed for the reasons explained in Section 9.

machine analyzes the SYN segments destined for ephesus, classifies them regarding their destination ports and aims to detect it whenever a SYN-flooding attack is launched against a given service. The method that we propose depends also on the proper setting of the backlog queue length of ephesus. Therefore we also propose a method for more precisely determining this value.

Our study is based on a set of traces obtained by monitoring all SYN segments destined for ephesus during 68 days (between Monday 23/11/98 and Friday 29/01/99).

5 Detection Method

The detection method that we propose is based on the intensity measures of SYN segments. At time of writing there exists at least one reference which mentions such a possibility[10].

The parameters that can be associated with the SYN flooding attack are:

?? *T*: SYN-RECEIVED state timeout in seconds (usually 75).
?? *L*: Per-port backlog queue length.
?? *A*: Number of received SYN segments per second by a given TCP port.

We call here *A*, as the intensity of SYN segments.

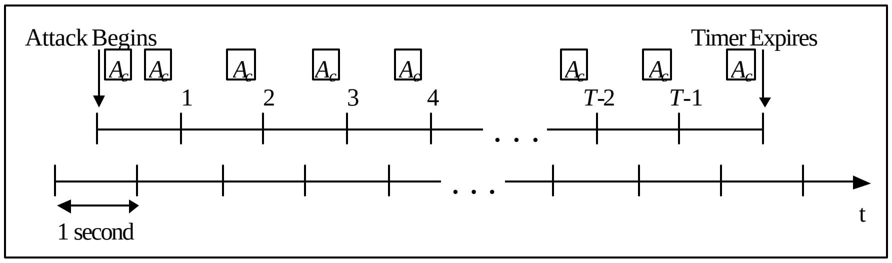
In order to succeed in a SYN-flooding attack, the minimum number of SYN segments that an attacker must send in *T* seconds is *L*. Thus, the average intensity of SYN segments in *L* seconds must be L / T. However this is a minimum value and the higher the chosen rate, the more effective the attack will be. We note that, this situation is of considerable advantage in the detection of an attack. An additional parameter needed for our detection method is:

?? A_c: the maximum acceptable (critical) *A* value.

Then, our network monitor computes *A* for each second and for each port of ephesus, and whenever it observes the condition $A > A_c$ satisfied, it considers the situation as an attack and acts accordingly.

6 Backlog Queue Length Requirements

In this section we describe the necessary size of the backlog queue for ephesus. Briefly, we aim to protect this host against undetectable attacks (hence, avoid false negatives) and decrease the probability of having false positives.


Attack Begins
Timer Expires
A_c A_c A_c A_c A_c A_c A_c A_c
1 2 3 4 . . . T-2 T-1
1 second
t
Figure 2. Possible Response of an Attacker

6.1 False Negatives

The possible response of an attacker against this method is illustrated in Figure 2. It should be noted that the maximum number of SYNs that an attacker will be able to send (without being detected) before the SYN-RECEIVED timer expires is:

$$N = T ? A_c + A_c$$

Therefore, in order to be immune to this worst case (the most intensive but undetectable attack), the backlog queue length of ephesus should be at least N +1, which gives:

$$L = T ? A_c + A_c + 1$$

Rearranged to collect the terms multiplied by A_c, this becomes:

$$L (A_c , T) = A_c ? (T+1) + 1$$

where T is a configurable parameter which is generally 75 seconds. As for A_c, it can be chosen so that the probability of having a false positive ($A > A_c$ but, in fact this is not an attack) is minimum. We want to be able to minimize this probability, because a false positive will cause the network monitor to act unnecessarily.

We note here that if L is too large, an undetectable attack may result in another form of denial-of-service: service degradation, since in this case too much of the system resources will be allocated by half-open connections. However, such an attack requires knowledge about A_c and the correct synchronization to the monitor's one-second time intervals. It is also possible to leave a doubt as to the exact time intervals of the monitor. In addition, there is no obvious way for an attacker to understand whether his/her attack is detected or not.

6.2 False Positives

Observations showed that the probability of receiving large numbers of SYNs in a given time interval is likely to be small during normal operation. Therefore we note that the larger the chosen A_c value the smaller the probability of false positives will be. This is illustrated in Figure 3. In this figure, ? is the area limited by A_c and represents the probability of false positives.

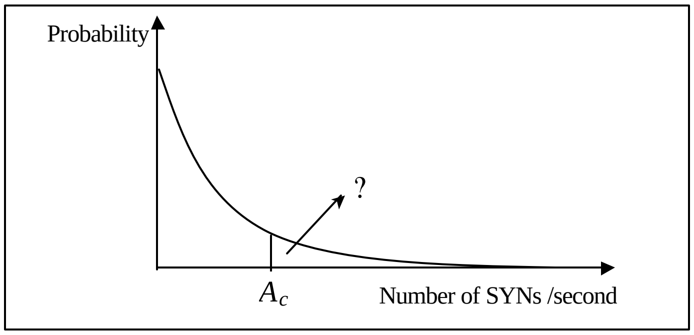


Figure 3. Probability of False Positives

However in practice there will be an upper limit to A_c since it comes as a factor in the calculation of the backlog queue length L needed to be immune to undetectable attacks as formulized above (it should be noted such a setting in L forces the attackers to fall in the ? region, which guarantees that we will not have any false negatives). Therefore, a proper statistical model is needed for precisely determining A_c. Nevertheless, there are important limitations in modeling SYN arrivals and setting L accordingly, which will be covered in the next section.

7 Limitations in Modeling SYN Arrivals

1. The Poisson model will fail

In the classical teletraffic theory, call arrivals are often modeled as Poisson processes. We would like to have such an analytical traffic model for TCP call arrivals (connection arrivals) because they are easier both to convey and analyze. However, a number of past studies by others have shown that analytic models and specifically the Poisson model (which is interesting for our purpose), does not suite well in today's Internet in all situations. Although we have not further examined the validity of these findings for our case (e.g., establishing statistical tests), we have reasonable pre-known evidences that using a Poisson model would be difficult.

First, our observations showed that we can not reasonably hope to model connection arrivals using homogeneous Poisson processes, which require constant rates. This is also observed by Paxson and Floyd[3]. Figure 4 shows the variation of hourly connection arrival rates at ephesus (obtained from the first 30 days portion of our traces). The important point that we note in this figure is that the connection arrival rate changes regarding both the hour of the day and the day of the week. For example, the arrival rates are generally smaller around midnights and during weekends.

Paxson and Floyd[3] note that during fixed length intervals (for example 1 hour or 10 minutes) the arrival rates can be assumed constant and the arrivals within each interval can be modeled by a homogeneous Poisson process. Such a model is referred to as a "nonstationary Poisson process". User session arrivals are likely to be nonstationary Poisson processes since a user session arrival corresponds to the time when a human decides to use

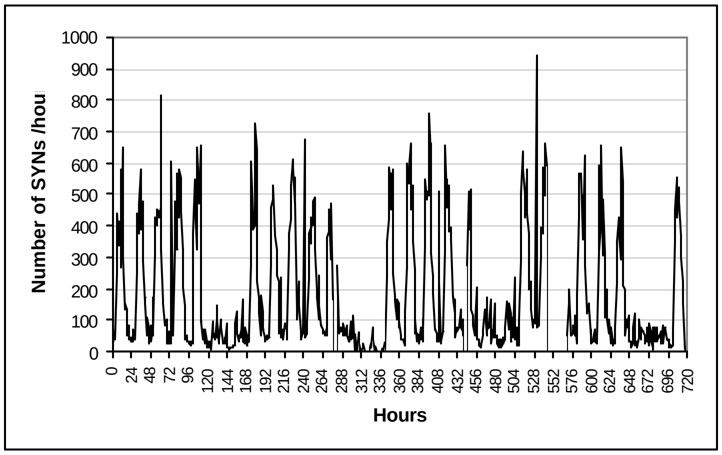


Figure 4. Hourly Variation of SYN Arrival Rates

the network for a specific task, hence they are generally uncorrelated. However, for our purposes we are interested in connection arrivals rather than user session arrivals. Paxson and Floyd[4] note that individual TCP connections that comprise each session are not well modeled by a Poisson process. TELNET and RLOGIN connection arrivals generally correspond to a new user therefore are likely to be close to uncorrelated, memoryless arrivals and can be described using nonstationary Poisson processes. However, X11, FTP data transfer and HTTP sessions generally consist of more than one connections which do not have this property therefore are not Poisson[3]. For example, an HTTP session consists of several connections corresponding to a user selecting links to other pages on the same server. Furthermore, in a HTTP session, not every connection is necessarily initiated by the user. The images to appear on a WWW page for example, are transferred immediately after the transfer of the text of the page. Finally, SMTP connection arrivals are not Poisson since they are machine -initiated and can be timer-driven. Furthermore, SMTP connections may be perturbed by mailing list explosions in which one connection immediately follows another and the timer effects due to using the DNS to locate MX records[3].

2. Backlog queue length is static

Another limitation that we note is that L is static. Currently no TCP/IP implementation provides a means for dynamically changing this parameter (furthermore, this would require the server and the monitor to communicate each other). However, as noted above, the clients' behaviour change in time and adapting L to these changes is not possible.

3. Different services have different characteristics

We note that, each service has different characteristics and possibly different clients. Therefore, it would be hard to find a single model that fits in all services. However, in most systems a single deamon, `inetd` issues the `listen()` system call (hence, determine the backlog limit) on behalf of the daemons for which it watches ports.

8 Estimating A_{MAX}

For the reasons explained above, we prefer setting L regarding the maximum SYN arrival rate that is likely to occur in the near future. Therefore, we propose:

$$A_c = A_{MAX}$$

Figure 5 shows the changes in the maximum SYN arrival rates at services provided by ephesus over one-day intervals.

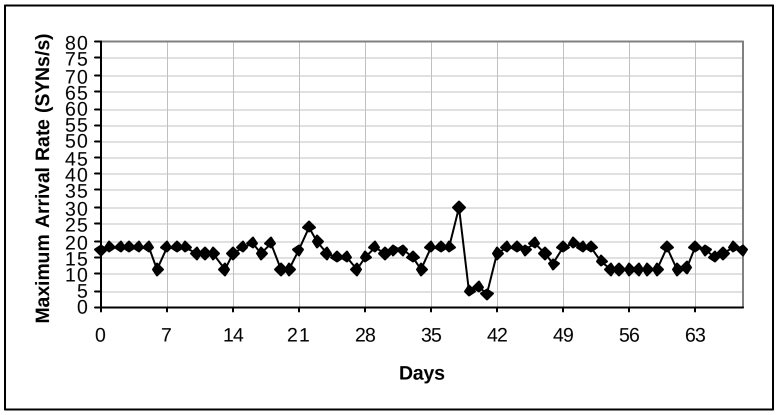


Figure 5. Daily Variation of Maximum SYN Arrival Rates

In this figure, we note that maximum SYN arrival rates are generally bound within the interval 15-20 (48 of 68). Therefore, we can reasonably hope that over one-day intervals the maximum SYN arrival rates will be fairly consistent. Furthermore, some of the deviations from the consistent behaviour can be related to known social events. For example, the maximum arrival rates generally decrease during weekends and holidays (the days 39-41 correspond to new-year vacation and we observe low arrival rates during these days). Therefore, in order to capture the most likely maximum arrival rate we can look at working days. The low arrival rates during days 54-59 are known to be caused by the degradation in our ISP's quality of service and the spike in day 38 is related to the new-year traffic (just before the vacation). Nevertheless, there is no obvious reason for the spike that we observe in day 23. As a result, completely avoiding false positives seems not possible with this method, however we can expect a significant decrease.

Figure 6 shows the theoretical numbers of false positives that would occur during the 68 days from which our traces were obtained, for A_c values between 15-20. We note that even for the worst case A_c value which is 15 we can except a significant decrease in the number of false positives (the monitor would act for only 110 seconds during 68 days) and for A_c values of 19 and 20 the number of false positives are negligible (monitor reaction for 9 and 5 seconds during 68 days, respectively).

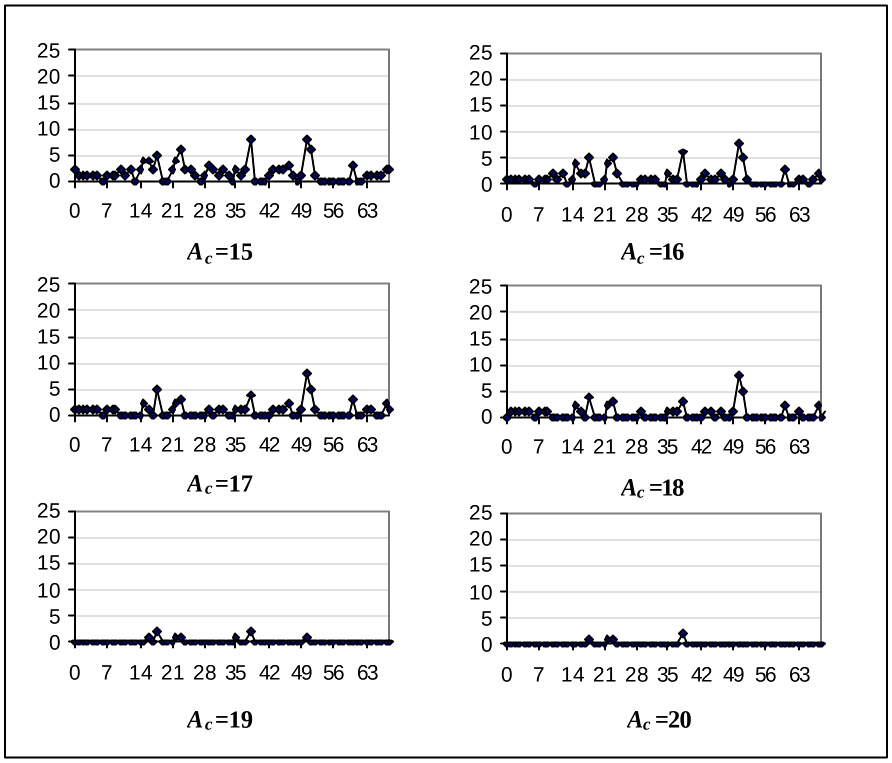


Figure 6. Number of False Positives per Day

9 Different Levels of Threat

Regarding the intensities of observed SYN segments, we currently define three different levels of threat. Our network monitor takes different actions for each level:

Level 0: A ? A_c

In this level, where no attack is observed, the network monitor does not react against the connections passing by. Therefore there is no risk of breaking legitimate connections.

Level 1: $A_c < A < L \;/\; T_A$

When an attack is detected to fall in this level, the monitor reacts against the SYN segments by sending RSTs for these connection requests after an aggressive timeout period (T_A). This guarantees that the attacker will not be able to fill the target server's queue.

This kind of reaction has the advantage of not causing service degradation at the target server because no ACK is used. It should be noted that the security administrators will generally prefer SYN-flooding attacks at this level. However, this requires an increase in L and/or a decrease in T_A. The former may cause extra costs, nevertheless with this method it is possible to decrease T_A since reaction against legitimate clients is minimized at level 0.

Level 2: A ? L / T_A

When a detected attack falls in this level, the monitor reacts for artificially completing the attacker's connection requests by sending the necessary ACK segments. These connections are reset if the correct ACK is not seen within an aggressive timeout period. This again guarantees that the attacker will not be able to fill the target server's queue.

This reaction has the disadvantage of causing service degradation at the server as described in Section 3. However, again a smaller T_A can reduce the duration of this service degradation.

An important point to note here is that the A_c, L and T_A values can be kept secret from the attacker. In this case these values can be thought as the keys of the proposed algorithm. Even if the attacker has full knowledge about the technique used, he/she will not know the actions taken against his/her attack, hence its consequences. Therefore the attractiveness of the attack will be diminished.

10 Model of Operation

Our network monitor can operate in two different modes: *training* and *defence*. The *training* mode consists of observing the normal client behaviour. Running in this mode, requires several administratively supplied parameters:

?? The duration of the *training* period in days.
?? SYN-RECEIVED state timeout in seconds (normally 75).
?? Aggressive timeout period in seconds.
?? The list of the hosts wanted to be protected.

Regarding this information, the network monitor observes the SYN arrival rates, computes $A_c = A_{MAX}$ and outputs a backlog queue length, separately for each host in the list. For the reasons noted above, we prefer working days (and when the quality of service is normal) for running our network monitor in this mode.

We use this information for configuring the backlog queue lengths of our servers, and start the network monitor in *defence* mode. In this mode, the network monitor uses the A_c values that it computed in the end of the training period. Then, whenever the $A > A_c$ condition is observed for one of the servers, it records the necessary information (IP addresses, port and sequence numbers) about the suspicious set of connection attempts observed in that second. Given a set of suspicious connection attempts, the network monitor can take one of the actions described above.

For each set, the percentage of connections that were reset can be also logged in order to provide security administrators with a means for differentiating between true and false positives. It should be noted that we can expect a small percentage of reset connections for false positives. Therefore, by looking at these logs, security administrators can decide whether the number of false positives is acceptable or not. These decisions can lead to heuristic changes in A_c and in this situation the network monitor is input a larger A_c value and it outputs the necessary L. However, the backlog queue must be reconfigured in this situation.

11 Future Work

It is also possible to analyze the distribution of RTTs (round-trip times) to the clients in order to be able to determine a more correct aggressive timeout period. RTTs can be modeled more easily since they are reported to be roughly Poisson[15]. Furthermore, in this case we will be able to change the aggressive timeout period regarding the changing behaviour of RTTs. The possible response of an attacker against this improvement can be training the network monitor with small RTTs (if possible) in order to cause denial-of-service to legitimate clients. However, our further response can be considering only correctly established connections when learning RTTs. Given the difficulty of sequence number attacks against up to date TCP implementations the attacker will therefore have to use his / her own (real) IP address. In this case we can locate the attacker.

Another possible improvement can be combining this method with `Synkill`. Although this method does not rely on source IP addresses, the `Synkill` approach will be certainly useful when the attacker's IP addresses repeat. However, with this combination we will not have to employ the large database of `Synkill` since IP address comparisons will be needed only during an attack. These entries can be discarded afterward since we do not react during normal operation.

12 Conclusion

In this paper we proposed a method for detecting SYN-flooding attacks in real-time. This method can allow ISPs to determine their correct requirements to cope with this attack. The main advantage of the method resides in its capability of reducing the probability of breaking legitimate connections during normal operation and service degradation both during normal operation and an attack.

The cost of the method will depend on the desired availability rate. If an ISP provides its services to a large number of clients, it is apparent that it will need to employ a larger backlog queue. This method provides a means for determining this value regarding the behaviour of the clients and several site security policies. These policies can be choosing from possible levels of threat or choosing a probability for denying service to legitimate clients.

The performance of the method depends on the correct estimation of A_c. A small A_c will result in a large number of false positives. Then, the network monitor will act for more legitimate connections, resulting in a risk of denying service to more legitimate clients. Therefore ongoing server workload characterization studies pursued by others will certainly help in better performance in the future. However, we will need network level SYN traces. For example the publicly available busy WWW server connection arrival traces found in ITA (Internet Traffic Archive) [16] were useless in modeling false positives in our case because they were obtained from application layer logs.

Heuristic changes in A_c may be also useful. For example, multiplying A_c by a factor of two will almost guarantee a minimum number of false positives. However, this will double the necessary backlog queue length as well. Stevens[10] analyzed the number of SYNs received per second by a commercial ISP's busy WWW server, which was providing service for 22 organizations. He observed $A_c = A_{MAX} = 20$ during one day and for A_c= 20?2 = 40, this would require L = 3041 (if T = 75). We note that this is less than the half of the backlog queue length proposed by one vendor mentioned in Section 3. Therefore, we conclude that such a detection method can considerably decrease the primary storage requirements necessary for coping with SYN-flooding attacks.

There is also an important problem with our approach: given the drastic development rate of the Internet and the computer sector, the highest arrival rates will also increase over long term. Such increases can be related to administrative policies as well. For example, installation of a large mailing list or a popular WWW service will certainly cause new clients to arrive, therefore increase the arrival rates. Running the network monitor in the *training* mode may be necessary whenever such an increase in the SYN arrival rates is expected or observed.

Finally, we note in this method several differences from the general definition of anomaly detection systems. First, this method is deprived of the primary advantage of anomaly detection methods: capability of detecting unknown attacks, instead it focuses on only a known vulnerability. Second, anomaly detection systems generally suffer from the fact that they can be trained by the attackers so that eventually, intrusive activities are considered normal. However, in our case such attacks can be avoided by considering only established connections during the training. The possible response of an attacker can be establishing connections very frequently, however given the difficulty of sequence number prediction attacks against up to date TCP implementations, this can be too easily detected: too many and frequent connections from a same host.

References

[1] daemon9, route, infinity, *IP-Spoofing Demystified*, Phreak Magazine, Vol. 7, Issue 48, File 14 (1996).

[2] R. T. Morris, *A Weakness in the 4.2BSD UNIX TCP/IP Software*, Computing Science Technical Report 117, AT&T Laboratories (1985).

[3] V. Paxson, S. Floyd, *Wide-Area Traffic: The Failure of Poisson Modeling*, IEEE/ACM Transactions on Networking, 3 (3) (1994) pp. 226--244.

[4] V. Paxson, S. Floyd, *Why We Don't Know How to Simulate The Internet*, Proceedings of the 1997 Winter Simulation Conference, Atlanta, GA (1997).

[5] P. A. Porras, A. Valdes, *Live Traffic Analysis of TCP/IP Gateways*, Proceedings of the Internet Society Symposium on Network and Distributed System Security (March 1998).

[6] J. Postel, editor, *Internet Protocol*, RFC791 (1981).

[7] J. Postel, editor, *Tranmission Control Protocol*, RFC793 (1981).

[8] C. L. Schuba et al, *Analysis of a Denial of Service Attack on TCP*, IEEE Symposium on Security and Privacy (1997).

[9] W. R. Stevens, *TCP/IP Illustrated, Volume 1, The Protocols*, Professional Computing Series, Addison Wesley (1994).

[10] W. R. Stevens, *TCP/IP Illustrated, Volume 3, TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols*, Professional Computing Series, Addison Wesley (1994).

[11] Computer Emergency Response Team, *TCP SYN Flooding and IP Spoofing Attacks*, CERT Advisory: CA 96-21 (September 1996).

[12] C.P.S.T. Ltd., *TCP SYN Flooding Attack and the Firewall-1 SYNDefender* (October 1996).

[13] L. S. Laboratories, *Livermore Software Lab. Announces Defense against SYN Flooding Attacks* (October 1996).

[14] SUN Microsystems, *SUN's TCP SYN Flooding Solutions*, SUN Microsystems Security Bulletin: #00136 (October 1996).

[15] D. Mills, *Internet Delay Experiments*, RFC 889 (1983).

[16] Internet Traffic Archive, data available at URL: http://ita.ee.lbl.gov