Analysis of a Denial of Service Attack on TCP

Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn,
Eugene H. Spafford, Aurobindo Sundaram, Diego Zamboni
COAST Laboratory
Department of Computer Sciences
Purdue University
1398 Department of Computer Sciences
West Lafayette, IN 47907–1398
{schuba,krsul,kuhn,spaf,sundaram,zamboni}@cs.purdue.edu

Abstract

*This paper analyzes a network-based denial of service attack for IP (*Internet Protocol*) based networks. It is popularly called* SYN flooding*. It works by an attacker sending many TCP (*Transmission Control Protocol*) connection requests with spoofed source addresses to a victim's machine. Each request causes the targeted host to instantiate data structures out of a limited pool of resources. Once the target host's resources are exhausted, no more incoming TCP connections can be established, thus denying further legitimate access.*

The paper contributes a detailed analysis of the SYN flooding attack and a discussion of existing and proposed countermeasures. Furthermore, we introduce a new solution approach, explain its design, and evaluate its performance. Our approach offers protection against SYN flooding for all hosts connected to the same local area network, independent of their operating system or networking stack implementation. It is highly portable, configurable, extensible, and requires neither special hardware, nor modifications in routers or protected end systems.

1. Introduction

Since September 1996, several dozen sites on the Internet have been subjected to a denial of service attack, popularly called *SYN Flooding* [4, 5, 20]. The attack exploits weaknesses in the TCP/IP (*Transmission Control Protocol/Internet Protocol*) protocol suite. This cannot be corrected without significant modifications to its protocols. These denial of service attacks can be launched with little effort. Presently, it is difficult to trace an attack back to its originator.

Several possible solutions to this attack have been proposed by others, and some implemented. We have developed an *active monitoring* tool that classifies IP source addresses with high probability as being falsified or genuine. Our approach finds connection establishment protocol messages that are coming from forged IP addresses, and takes actions to ensure that the resulting illegitimate half-open connections are reset immediately.

This paper is organized as follows. Section 2 describes background material, such as the IP and TCP protocols. Section 3 explains the SYN flooding attack. Section 4 discusses existing approaches to solve this problem, such as configuration improvements and firewall-based approaches. The technical details of our approach are described in Section 5, followed by a performance evaluation in Section 6. Sections 7 and 8 outline future work issues and present conclusions.

2. Background

We will provide a brief description of the features of the TCP/IP protocol suite that facilitate this attack. For further details see [2, 16, 17].

2.1. Internet Protocol

The *Internet Protocol* (IP) is the standard network layer protocol of the Internet that provides an unreliable, connection-less, best-effort packet delivery service. IP defines the basic unit of data transfer used throughout an IP network, called a *datagram*. The service is *unreliable*, because the delivery of datagrams is not guaranteed. Datagrams may be lost, duplicated, delayed, or delivered out of order. IP is *connection-less*, because each packet is treated independently of others — each may travel over different

paths and some may be lost while others are delivered. IP provides *best-effort* delivery, because packets are not discarded unless resources are exhausted or underlying networks fail. Datagrams are routed towards their destination. A set of rules characterize how hosts and gateways should process packets, how and when error messages should be generated, and when packets should be discarded.

2.2. Transmission Control Protocol

To ensure reliable communications for applications and services that need them, the *Transmission Control Protocol* (TCP) is available. It resides between IP and the application layer. TCP provides a reliable, connection-oriented data stream delivery service. As long as there is link layer communication between two communicating endpoints, TCP guarantees that datagrams will be delivered in order, without errors, and without duplication. It provides these services by using flow control mechanisms, such as the sliding window protocol, and adaptive retransmission techniques.

2.2.1 Three-way Handshake

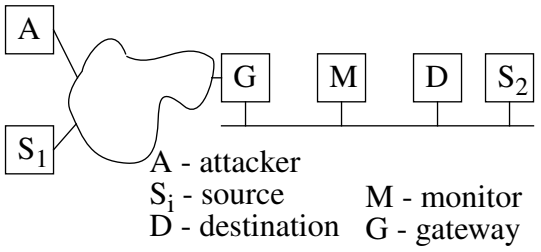

Figure 1. Generic network topology

Before data can be transmitted between a source host S_i and a destination host D, TCP needs to establish a connection between S_i and D (see Figure 1). The connection establishment process is called the *three-way handshake* (see Figure 2). The first step in the process is a SYN[1] packet that is sent from S_i to D. The second message, from D to S_i, has both the SYN and ACK flags set indicating that D acknowledges the SYN and is continuing the handshake. The third message, from S_i to D has its ACK bit set, and is an indication to D that both hosts agree that a connection has been established. The third message may contain user payload data.

The three-way handshake also initializes the sequence numbers for a new connection between S_i and D. Sequence numbers are needed by the TCP protocol to enable reliable

[1] TCP packet types are distinguished by flag bits (e.g., SYNchronize, ACKnowledgment, ReSeT) set in the TCP header code field. In the remainder of the paper we will abbreviate TCP control packets by referring to the flags set in their code field, e.g., *SYN* instead of *TCP control datagram with the SYN bit set in its code field.*

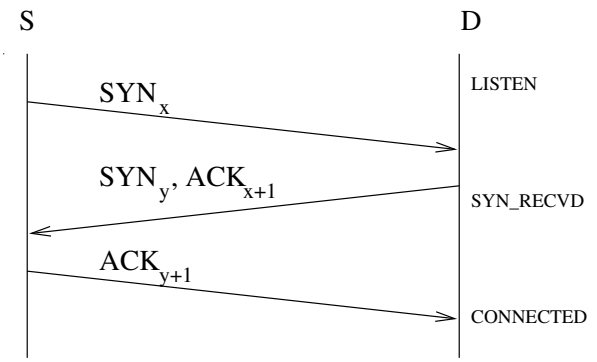

Figure 2. Three-way Handshake

packet delivery and retransmission. S_i sends an initial sequence number x with the first datagram: SYN_x. In the second message D acknowledges the first datagram with ACK_{x+1} and sends its own sequence number y: SYN_y. S_i acknowledges D's packet in the final message of the three-way handshake: ACK_{y+1}.

2.2.2 TCP Data Structures

For any TCP connection, under BSD style network code, there are three memory structures that need to be allocated by both endpoints (See [19]). The socket structure (`socket`) holds information related to the local end of the communication link: protocol used, state information, addressing information, connection queues, buffers and flags. TCP uses the Internet protocol control block structure (`inpcb`) at the transport layer to hold information such as TCP state information, IP address information, port numbers, IP header prototype and options, and a pointer to the routing table entry for the destination address. The TCP Control Block structure (`tcpcb`) contains TCP specific information such as timer information, sequence number information, flow control status, and out-of-band data. The combined size of these data structures for a single TCP connection may typically exceed 280 bytes.

Different versions of Unix use different data structures and schemes of allocation, but for the purpose of this discussion, it is sufficient to understand that every TCP connection establishment requires an allocation of significant memory resources.

2.2.3 TCP Connection Establishment

When a SYN arrives at a port on which a TCP server is listening, the above-mentioned data structures are allocated. There is a limit on the number of concurrent TCP connections that can be in a half-open connection state, called the

Operating System	Backlog	Backlog + Grace
FreeBSD 2.1.5	n.a.	128
Linux 1.2.x	10	10
Solaris 2.4	5	n.a.
Solaris 2.5.1	32	n.a.
SunOS 4.x	5	8
Windows NTs 3.51	6	6
Windows NTw 4.0	6	6

Table 1. Backlogs for some Operating Systems

SYN_RECVD state (i.e., SYN received — see Appendix A). Not enforcing this limit would lead to a different denial of service attack: an attacker could request so many connections that the target machine's memory is completely exhausted by allocating data structures for half-open TCP connections. When the maximum number of half-open connections per port is reached (see Table 1, [6]), TCP discards all new incoming connection requests until it has either cleared or completed some of the half-open connections. Overall system resources are usually sufficient for several ports to be flooded.

The TCP connection establishment process can be described as a state machine. Detailed below is what happens from the point of view of the destination machine (server):

1. A packet arrives at the destination machine when the TCP state machine is in the LISTEN state.

2. If the datagram checksum is incorrect, the packet will be discarded, and the client is expected to retransmit it.

3. The `tcpcb` associated with the connection is searched for. If it is not found, the server will discard the packet and will send an RST (i.e., inform the client that it reset the connection). If the `tcpcb` exists, but the TCP state machine is not in the LISTEN state, the server will discard the packet, but will not send an RST (this would, for example, be the case when the server is just coming up, but has not yet started listening).

4. If the SYN packet arrives for a socket that is in the LISTEN state, the above mentioned data structures will be allocated. However, the server will also set a flag indicating that it will destroy the connection and associated memory structures if it encounters an error. If the backlog queue is full, the server will consider this an error and will terminate the connection.

5. The packet will be ignored if it contains an RST. If it contains an ACK, it will be discarded and an RST sent to the other side. The packet will be discarded if the SYN bit is not set. Otherwise, the server copies information, such as the client's address information, into a buffer, connects its `tcpcb` to the client, and initializes its initial send sequence (ISS) number y.

6. The server now sends the second message of the three-way handshake (SYN_{x+1} and ACK_y) to the client. The state changes to SYN_RECVD. A connection establishment timer is started for this half-open connection. The connection remains in the SYN_RECVD state until either an ACK (the third message of the handshake) is received or until the timer expires. This timer is usually set to 75 seconds. During this period of time retransmissions of the first and second message of the three-way handshake may occur. When the timer expires, all memory structures associated with the connection are deallocated, and the server goes back to the LISTEN state.

3. The SYN Flooding Attack

3.1. The Attack

As mentioned above, TCP implementations are designed with a small limit on how many half-open connections per port are possible at any given time. An attacker A initiates a SYN flooding attack by sending many connection requests with spoofed source addresses to the victim machine D. That causes D to allocate resources as explained in Section 2.2.3 and, once the limit of half-open connections is reached, to refuse all successive connection establishment attempts — in particular legitimate attempts (see Figure 3). It is important to note that neither outgoing connection attempts nor connections that are already established are affected by this attack.

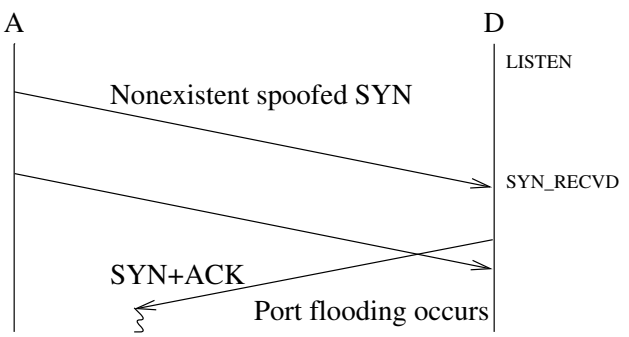

Figure 3. A system under attack

This condition exists until either the timer expires, or some connections are completed or reset. If the timer ex-

pires for a particular half-open connection, the host will reset the connection and release all resources allocated for it.

If a spoofed SYN packet contains the source address of a reachable IP host S_i, that host will receive the second message of the three-way handshake generated by D. Not expecting a SYN+ACK without having requested a connection, S_i will send a RST packet to D, and consequently cause D to reset the connection. It is therefore in the interest of an attacker to forge source addresses that do not belong to hosts that are reachable from the victim D.

If the attacker wants the denial of service condition to last longer than the timeout period, he needs to continuously keep requesting new connections from the victim machine. The amount of CPU and network bandwidth required by an attacker for a sustained attack is negligible.

The basis of the attack is that TCP/IP does not offer strong authentication on its control packets. Furthermore, there is a requirement for an inappropriately burdensome allocation of memory and computation resources on the target side.

3.2. Different Attack Modes

Typical SYN flooding attacks can vary several parameters: the number of SYN packets per source address sent in a batch (=: *batch-size*), the delay between successive batches (=: *delay*), and the mode of source address allocation (=: *mode*).

We consider only source addresses of hosts that are not reachable from D, be it because the addresses are not yet allocated, assigned, or the associated hosts are very slow in response, virtually or physically disconnected, or down. We classify three possible modes of source address allocation: The attacker can be using a single address, a short list of addresses, or no list at all[2].

Single address: The attack scripts published in the hacker magazines Phrack [6] and 2600 [8] take as a parameter a single spoofed address that is used as the source address of all SYN packets. In the absence of any defense, this mode of attack is as effective as the other three modes described.

Short list: An attacker can generate a small pool of addresses and use them as source addresses to generate SYN packets.

No list: The attacker can use a different, randomly generated source address for each successive batch of SYN packets.

[2] We list the "single address" mode separately, because it represents an important special case of the "list of addresses" mode.

4. Solutions

In our opinion a good solution should have the following characteristics:

- independence of operating system and network stack implementation of the protected end systems
- no requirement for IP or TCP protocol modifications
- capability to protect sets of machines, and not only a single machine
- no special hardware requirements
- portability
- extensibility
- configurability

The countermeasures described in this section have been proposed by others to date. None of these proposals provides all the characteristics we are looking for.

4.1. Configuration Optimization

There are several ways of reducing the likelihood and effects of an attack that involve changes in the configurations of end systems and routers.

4.1.1 System Configuration Improvements

To defend against the exhaustion of resources in the systems under attack, an obvious approach is to increase the number of resources devoted to half-open TCP connections, and to reduce the timeouts. These measures have been suggested by different sources [11], and can be summarized as:

1. Reduce the timeout period from the default to a short time, e.g., 10 seconds. This helps in pruning half-open connections from the TCP queue.
2. Significantly increase the length of the backlog queue from the default (see Table 1). This makes the system able to cope with more simultaneous half-open connections than before.
3. Disable non-essential services, thus reducing the number of ports that can be attacked.

These measures help in dealing with attacks, but also have severe shortcomings:

1. Lowering the timeouts may deny legitimate access for machines to which the round trip times exceed the timeout period.

2. Increasing the backlog leads to a potential increase in resource usage. One vendor recommends upgrading systems to a minimum of 128 MB RAM to allow them to cope with attacks.

4.1.2 Router Configuration Improvements

The measures proposed in the first reactions to the recent attacks [4], as well as several other sources [1, 9], attempt to make it difficult for packets with spoofed source addresses to traverse routers. The solutions proposed can be summarized as follows:

1. Configure external interfaces on routers to block packets that have source addresses from the internal network.

2. Configure internal router interfaces to block packets to the outside that have source addresses from outside the internal network. This limits the ability to launch a SYN flooding attack from that network, because the attacker would only be able to generate packets with internal addresses.

These measures can be effective, but only if taken in large scale. As more *Internet Service Providers* (ISPs) configure their routers appropriately, the fertile ground for launching SYN flooding attacks may be reduced. It should be noted that in mobile IP situations can be created in which legitimate addresses appear at an apparently wrong interface, in particular if a mobile node retains its IP identity while far away from home.

4.2. Infrastructure Improvements

Router configurations can be improved if the address spaces reachable over their various interfaces are disjoint and well-defined [9]. This is the case for routers that attach an organization or a local ISP to a backbone network. The address prefixes separate the *inside* and the *outside*. An example where this scheme is deployed is the international telephone system. Phone number assignment is based on the geographical location of the end system.

Currently, there are practical problems for this approach to work: in general, routers in large backbone networks with complex topology cannot make a clear distinction between inbound and outbound traffic. Packets are routed in backbones based on current link availability and load and can take numerous possible paths through the network. Genuine packets from the same source address can reach a backbone router legitimately over various interfaces.

As long as a significant number of sites can transmit packets into the backbone networks without any source address checking, hosts are still subject to untraceable attacks. Therefore, additional backbone mechanisms should be implemented to cope with a large number of network based attacks.

The implementation and deployment of a scheme to cryptographically sign IP source addresses of all packets would allow tracing the physical transmission path of any IP packet to its source. Although this wouldn not prevent SYN flooding, the threat of tracing and subsequent prosecution should serve as a deterrent to at least casual attacks. In this case, online tracing mechanisms are especially useful, because a successful SYN flooding attack requires sustained network activity. The Internet infrastructure lacks basic mechanisms that have been present and successfully used in telephone networks for a long time.

4.3. Connection Establishment Improvements

This solution addresses the fact that TCP imposes asymmetric memory and computation requirements on the two endpoints during each connection establishment process. The destination host needs to allocate large data structures in response to any SYN packet, without any guarantee of its authenticity.

The three-way handshake requires the sequence number y to match between the second and third message to protect against accidentally reopened old connections and unauthorized access (see [3]). The destination therefore needs to either store its ISS y between sending the second message and receiving the third message, or be able to regenerate y at the time the third message of the three-way handshake is received. If there were no mechanism to regenerate y and the destination didn't store y, any host could establish a connection by sending only the third message.

One such mechanism is to calculate y as a cryptographic hash value of source and destination IP addresses, ports, the source's ISS x, and a destination specific secret key. D would calculate y in that manner and use it in its SYN+ACK message. At the time D received the third message of the three-way handshake it can recalculate y' by using its secret key, sequence number, the addresses, and the ports found in that message. If y' matches the y in ACK_{y+1}, the connection is legitimate, otherwise it is not. Note that this solution also provides some protection against sequence number prediction ([3]), because of the statistical properties of good hash functions.

Although this approach prevents the SYN flooding attack, it has considerable drawbacks. This solution requires the modification of the TCP standard and consequently every TCP implementation. It is impossible to provide the fault tolerance that TCP currently offers without the destination keeping state about each half-open connection. Furthermore, this mechanism makes it impossible for the source to include data in the third message of the three-way hand-

shake, because x needs to be part of the hash function argument. As there are only 2^{32} TCP sequence numbers, this technique introduces a small probability that an old or a single forged packet might open a connection. Section 4.4.1 discusses an extension of this approach.

4.4. Firewall Approach

As many sites connected to the Internet are already somewhat protected by firewalls, it makes sense to try to use firewalls to protect against SYN flooding. Several firewall vendors have already made products available to increase protection against the attacks [14, 15], and some other solutions have been proposed.

Firewall-based protection approaches are based on the idea that every packet destined to a host inside the firewall has to be examined by the firewall first, and thus decisions can be made on its authenticity and actions can be taken to protect the internal hosts. This can be effective if, apart from the normal blocking done by the firewall, some other specialized mechanism is put in place to deal with SYN flooding.

The drawbacks of this approach are delays on every packet for additional processing. Not every firewall product is capable of adding functionality, such as a module to protect against SYN flooding.

The two main approaches are described below.

4.4.1 Firewall as a Relay

In this approach, when a packet for an internal host is received the firewall answers on its behalf. Only after the three-way handshake is successfully completed does the firewall contact the host and establish a second connection.

1. In the case of an attack (see Figure 4), the firewall answers to the SYN sent by the attacker. Because the final ACK never arrives, the firewall terminates the connection, and the host never receives the datagram. This mode of protection is only effective if the firewall itself is not vulnerable to SYN flooding.

2. In the case of a legitimate connection (Figure 5), after the firewall receives the final ACK, it creates a new connection to the internal host on behalf of the original client. This makes the protected machines vulnerable to the new degradation of service attack described in Section 2.2.3. Once the connection is established, the firewall has to keep acting as a proxy to translate the sequence numbers in the packets that flow between the client and the server.

This method has the drawback of introducing new delays for legitimate connections. Delays are introduced by extra processing done at the firewall, both at connection establishment time and for each data packet. The obvious advantage is that the destination host never receives spoofed SYN packets.

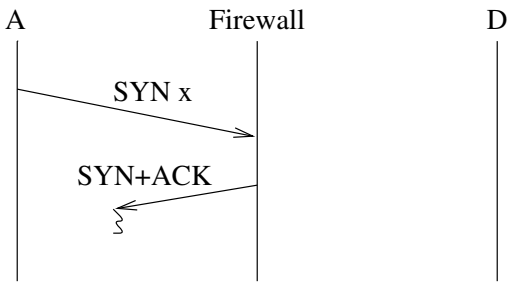

Figure 4. Attack scenario with a relay-firewall protection

An alternative approach in which the firewall could predict the sequence number that is going to be used by the host (see Section 4.3) would allow the firewall to intervene in the same manner when establishing the connection, without the need for translating sequence numbers for each data packet.

4.4.2 Firewall as a Semi-transparent Gateway

In this approach, the firewall lets SYN and ACK packets go through, but monitors the traffic and reacts to it. We call this the *semi-transparent gateway* approach.

The firewall passes SYN packets destined to internal hosts. When the host responds with a SYN+ACK packet, the firewall forwards it, but reacts by generating and sending an ACK packet that seems to come from the client. This has the effect of moving the connection out of the backlog queue in the host, thus freeing the resources that were allocated for the half-open connection.

1. In the case of an attack (see Figure 6), when the host sends the SYN+ACK, the gateway lets it pass and generates and sends the ACK that moves the connection out of the backlog queue. If the firewall has not received the legitimate ACK after some (arguably short) period of time, it will send a RST packet, terminating the connection.

2. In the case of a legitimate connection (Figure 7) the firewall generates and sends an ACK packet. When the legitimate ACK packet arrives, the firewall lets it pass, and the "duplicate" ACK packet arrives at the host. TCP is designed to cope with duplicate packets, so the duplicate packet is silently discarded. Now data can flow freely in both directions, without further firewall intervention.

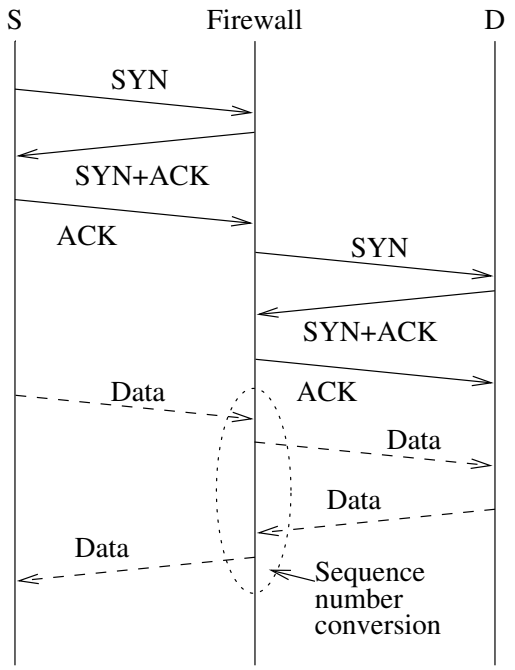

Figure 5. Legitimate connection with a relay-firewall protection

The main advantage of this approach over the previous one is that no delays are introduced for legitimate connections once they are established. The price to pay is a large number of illegitimate open connections at the destination if it is under attack. However, the limit on the number of open connections is much higher on most systems (in the order of thousands, limited only by the CPU and memory resources available at the host), so it is an extra load that most server class systems can withstand without many problems. Again, this approach requires the timeout period to be very carefully selected, so as to not deny access to legitimate hosts with long response times.

4.5 Active Monitoring

This category of solutions consists of using software agents to continuously monitor TCP/IP traffic in a network at a given place. An agent can collect communication control information to generate a view of all connections that can be observed on a monitored network. Furthermore, it can watch for certain conditions to arise and react appropriately. This category offers a general approach to detecting and reacting to a large class of network based attempts to breach security.

The approach is attractive, because of its low costs and high flexibility. It neither requires new hardware to be purchased and installed, such as a certain firewall router product with the desired functionality, nor does it demand software modifications to the protected end systems. On typical multiple access local area networks such a tool can only be reactive in character, because it has no capability of blocking any "offensive" traffic as a firewall does.

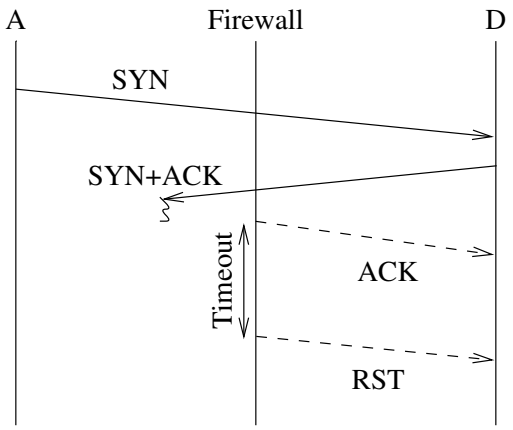

Figure 6. Attack with semi-transparent gateway firewall protection

At the time of this writing, we know of one publicly available tool [12] that is claimed to perform active monitoring (other than our tool called `synkill`). According to user-definable parameters, the program monitors the local network for SYN packets that are not acknowledged after a certain period of time, and frees the allocated resources by sending matching RST packets. Our tool called `synkill` falls into the same category and is explained in the following Section 5.

5. Active Monitor — `synkill`

We have developed a software tool that can lessen the impact of SYN flooding attacks, and in many cases defeat attacks completely. It provides all characteristics as described in Section 4.

5.1. Description

The program requires the ability to monitor and inject network traffic to and from the machines it is protecting. Ethernet is an example for a networking technology that satisfies this requirement. The program is called a *monitor*, because it reads and examines all TCP packets on the LAN after setting its network interface into promiscuous mode. The program is called *active*, because it can generate TCP packets in response to observed traffic and inject them into the

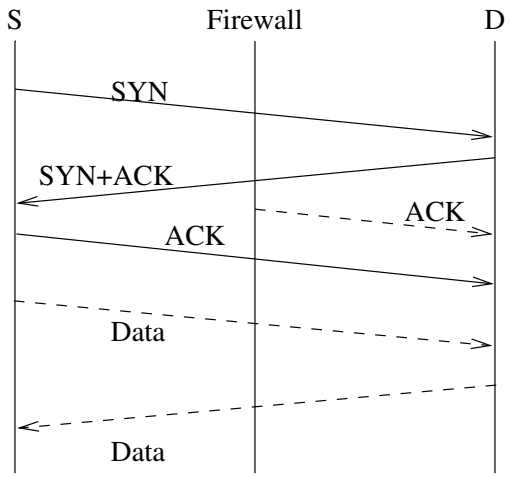

Figure 7. Legitimate connection with semi-transparent gateway firewall protection

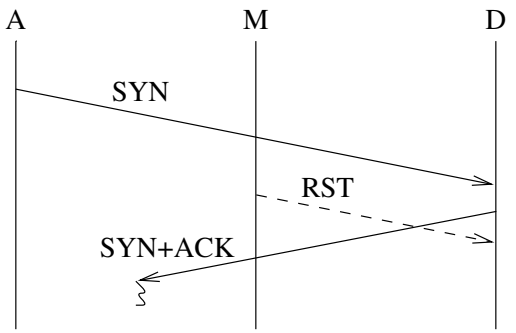

Figure 8. Attack scenario: `synkill` generates RST packet in response to *bad* or *evil* IP source addresses. The connection at D is immediately moved into the CLOSED state and resources are released.

network. In the following sections we will refer to the algorithm, and its implementation as `synkill`.

5.1.1 Algorithm

The `synkill` algorithm classifies the source IP addresses of TCP packets as never seen (=: *null*), belonging to correctly behaving (=: *good*) hosts, as potentially spoofed addresses (=: *new*), or as most certainly spoofed addresses (=: *bad*). This classification is based on observed network traffic and administratively supplied input. Addresses that are administratively configured as *good* (*bad*) are called *perfect* (*evil*).

`Synkill` performs several processing steps on every TCP packet that is observed on the local area network, and handles asynchronous events, such as administrative input and timer expirations. TCP packet processing can be divided into:

- address prefiltering, where the program classifies the observed address as impossible, unassigned, or administratively configured as *perfect* or *evil* (see Section 5.1.2)
- a decision process based on a state machine to determine correct state membership and actions (see Section 5.1.3).

The program can take two possible actions:

- `Synkill` sends RST packets whenever it observes connection establishment attempts from impossible, *bad*, or *evil* IP addresses or networks (See Figures 8, 9, and 11). The purpose of this action is to release the resources allocated at the destination machine for connection establishments.
- `Synkill` completes TCP connections by generating the third message of the three-way handshake, and sending it to the destination (See Figures 9, 10, and 11). The purpose of this action is to move a connection quickly from the SYN_RECVD to the CONNECTED state. This is useful if `synkill` considers the connection establishment attempt to be illegitimate. This approach is similar to the semi-transparent gateway solution described in Section 4.4.2 and is also potentially subject to the new degradation of service attack described in 2.2.3. The execution of this action is optional.

5.1.2 Operation

In addition to the address classification, `synkill` performs the following processing steps.

- process administrative input (asynchronously)
- handle *expiry* events (asynchronously)
- handle *staleness* events (asynchronously)
- send RST for all impossible addresses (e.g., net 0.0.0.0 or 127.0.0.0)
- send ACK to complete observed SYN+ACK connections
- send RST for all *evil* addresses (e.g., nets 10.0.0.0, 172.16.0.0, and 192.168.0.0; see [18])

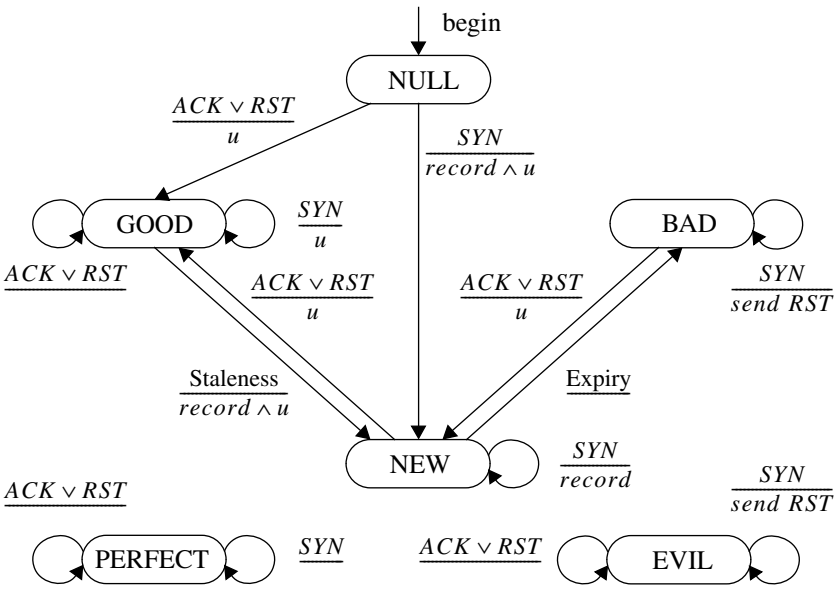

Figure 12. The `synkill` finite state machine. The diagram does not contain the optional action of sending ACK packets.

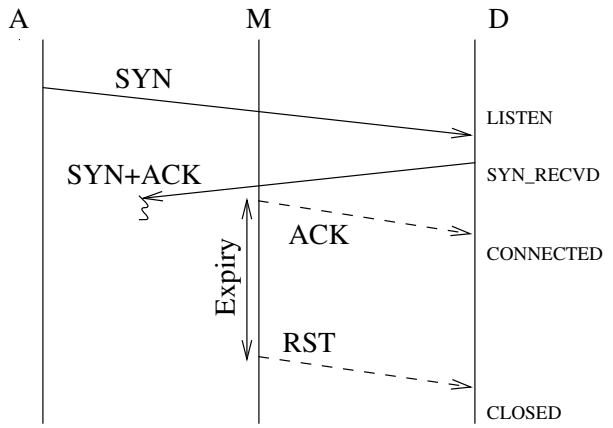

Figure 9. Attack scenario: `synkill` sends an ACK packet to complete the connection. After *expiry* has passed, `synkill` generates a RST.

5.1.3 State Machine

After the preprocessing steps are taken, `synkill` operates as a state machine (see Figure 12). The source address of each TCP packet is examined to determine the set membership of the address (*null*, *new*, *bad*, or *good*). *Null* addresses are not saved explicitly, because it is not practical to keep data structures for all possible IP addresses. If an address is not present in the database, it is considered to be in state *null*.

Figure 12 depicts the state machine. The symbol *u* denotes when the timestamp of a given address is updated. These timestamps are used to generate timer events (see below). *Record* denotes where datagram information (IP addresses, ports, and sequence numbers) is recorded, so that a RST can be generated later if necessary. There are several distinct sets of events: observed TCP packets, timer events, and administrative commands:

1. Observed TCP packets

   SYN TCP packets with the SYN bit set are the initial message of any TCP connection establishment attempt. The state machine is designed to ignore SYNs for addresses that are in the *new*, *good*, or *perfect* states. For addresses in the *bad* or *evil* states, a RST packet is generated and sent.

   The very first packet received from an address with its SYN bit set is moved into the *new* state to indicate suspicion. As soon as further valid TCP traffic from that address is observed (ACK, RST) the address is moved into the *good* state.

   ACK, RST If `synkill` receives a valid ACK or RST packets from an address, it means that the host generates valid packets and the address can be considered *good*. The address is moved into the *good* state.

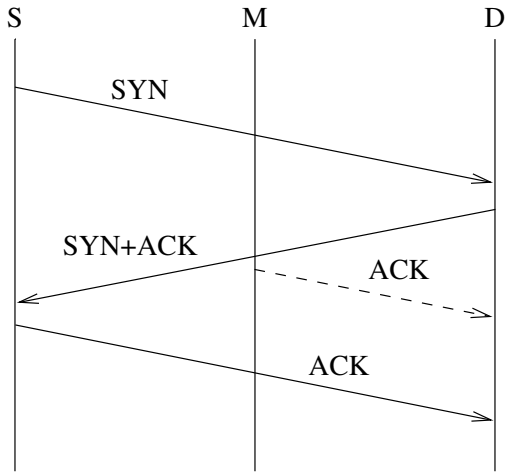

Figure 10. Normal access scenario: `synkill` generates and sends an ACK packet to complete the pending connection. The "duplicate" ACK from the original source A reaches D later and is ignored.

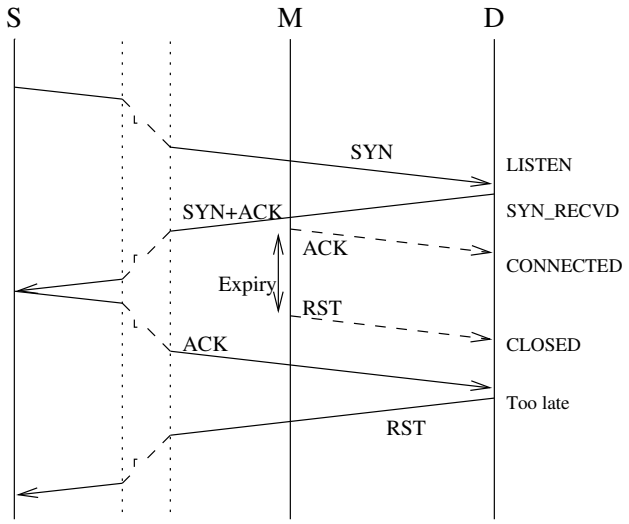

Figure 11. Normal access scenario: the connection delay between a valid source machine and the destination is large. `Synkill` will generate a RST too early if the *expiry* timer value is not chosen carefully!

2. Timer events

expiry An *expiry* event occurs if the timer associated with an address in the *new* state expires. This means that `synkill` has not observed any valid TCP traffic from that address. The address is therefore moved into the *bad* state and RST packets are generated and sent for all SYN packets from that address that were observed while the address was in the *new* state. Ideally, the expiry timer should be much smaller than the current 75 seconds timeout. The smaller the chosen value the more likely it is for legitimate connections to be erroneously denied by `synkill`.

Because RSTs are sent after the SYN was observed, the destination machine will respond with a SYN+ACK and thus trigger the third message of the three-way handshake. This third message (an ACK) will cause `synkill` to reclassify the observed address as *good*. Subsequent connection establishment attempts will therefore succeed.

staleness The notion of staleness was introduced as a mechanism to allow addresses in the *good* state to leave the *good* state after no TCP traffic was observed from that address for a period of time, i.e., the staleness period. This allows `synkill` to correctly classify spoofed IP addresses as *bad* even if they were once *good* — as long as they first became stale.

This can be implemented either with explicit timer events, or with a timestamp per address that is examined the next time the address is processed.

5.2. Implementation

We have implemented this algorithm in the programming language C with a Tcl/Tk graphical user interface. The program can execute in the foreground or as a daemon. Its output can be redirected to syslog. Currently, the program's classification database can grow to over 47600 entries and it garbage collects database entries if the database is filled beyond a certain watermark. It utilizes the Packet Capture library from the Lawrence Berkeley National Laboratory, a high level interface to packet capture systems, to make all packets on the monitored network accessible in a highly portable manner.

There is a rich set of administrative commands to manipulate the address classification database, display statistics and modify the configuration of `synkill`. Refer to the manual page for details.

5.3. Discussion

The philosophy behind our approach was to build a tool that can detect the conditions of a SYN flooding attack and

react appropriately to defeat, or at least lessen the impact of, the attack. `Synkill` neither requires any special hardware (such as particular firewall products), nor certain certain operating systems, network stacks, or even modifications in the protected end systems. Our software is highly portable, extensible, and easily configurable.

In our testbed, we successfully protected a set of hosts of a wide variety of vendors and operating systems against the attack. Section 6 details some of the operational characteristics of the `synkill` application. Furthermore, the active monitor approach allows for replication of the software to improve reliability and performance because of decentralized and distributed action.

6. Performance of `Synkill`

6.1. Experimental Evaluation

The performance of the `synkill` application was evaluated using the configuration illustrated in Figure 1. The attacker A performs a SYN flooding attack against machine D. The `synkill` application runs on machine M protecting all hosts on the local area network. Host S_2 evaluates the accessibility of D in the following way: S_2 starts 25 processes that attempt to establish connections to the target computer simultaneously. Each of these processes performs one hundred sequential attempts with a random delay between zero and four seconds. The machines utilized for the evaluation environment are SUN Sparc Ultra 1 workstations with 32 MB of RAM, running Solaris 2.5.1.

Two metrics are considered during the performance tests: 1) success rates and 2) average delays. A success was defined as a connection attempt that succeeds, regardless of the delay incurred. The success rate was determined by dividing the number of successful connections by the number of connections tried in a given time interval. Delay was defined as the time that was required to establish a successful connection; hence, delays were only computed for successful connections. The time intervals for both measures was determined by measuring the total time taken by each test case and dividing this number by thirty (30).

Upon successful connection completion, the connection is closed immediately. Typical TCP connections do not exhibit this behavior. However, we are interested only in determining how many connection establishments can succeed under attack. To simulate maximum contention, we performed all connection establishment attempts against a single port on the server.

This simulates a scenario where 25 hosts perform, on the average, one TCP connection establishment attempt every two seconds. This means the accessed server must service 750 requests a minute — about an order of magnitude more than the authors' departmental Web server.

Test	Defense	Attack Configuration		
		mode	delay sec.	batch-size
1	none	None		
2	none	Single Addr.	10	100
3	`synkill`	Single Addr.	10	100
4	`synkill`	Single Addr.	1	20
5	`synkill`	20 Addrs.	1	2
6	`synkill`	No list	1	10

Table 2. Summary of test cases used for performance evaluation of `synkill`

6.2. Explored Evaluation Space

We use six test cases to evaluate the performance of the `synkill` program. The test cases are summarized in Table 2. The terms *mode*, *delay*, and *batch-size* are explained in Section 3.2. They are used to characterize instances of SYN attacks.

6.3. Evaluation Results

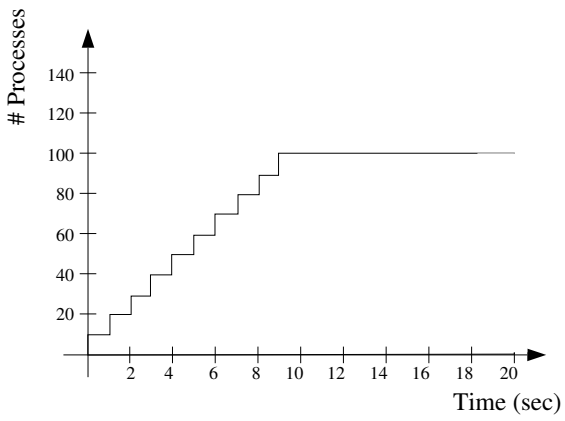

Figure 13. Process growth for the attack in case 6.

The first two test cases are included as points of reference. Test 1 executes the evaluation scripts without D being under a SYN flooding attack. The second test runs the evaluation scripts with D being under attack, but without any defenses. Figures 14 and 15 show the success rates and average delays for these two test cases.

In the second test case the attacker sends twenty batches of one hundred spoofed SYN packets each with a delay of ten seconds between batches. Note that the areas marked

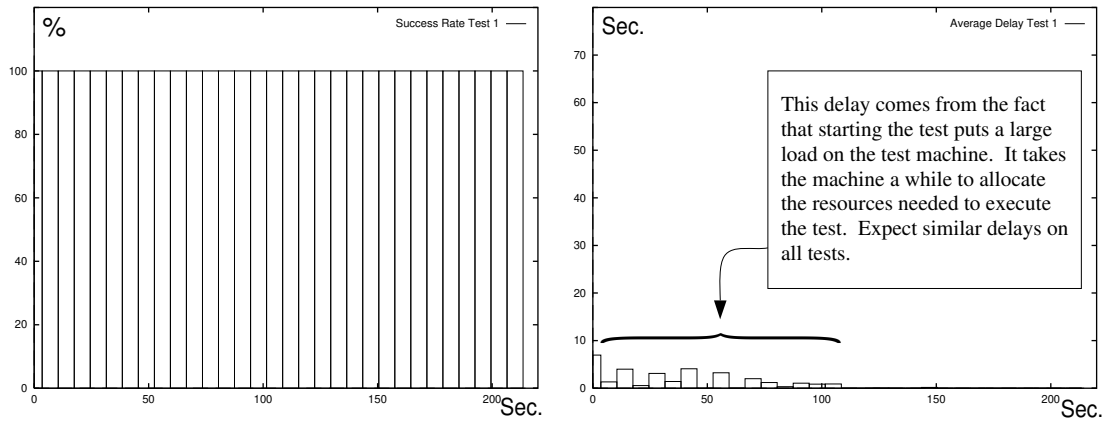

Figure 14. Test case 1: Connection establishment success rates and average delays for normal operation of D (without attack).

with the letter α correspond to a small window of opportunity that the evaluation program has when the attacked machine releases the first set of blocked ports. The delay in this case indicates that, on the average, these were successes in the first TCP retry attempt. The area marked with the letter β shows how once the attack has stopped the connections succeed but only after a very large delay.

In test cases 3 and 4 `synkill` protects the target machine against a single address SYN flooding attack of different delays and batch-sizes. In both cases similar performance results can be observed. `Synkill` learns the spoofed address, classifies it as *bad*, and releases half-open connections from that address as soon as they are observed. All legitimate connections succeed, and only small delays are observed.

Test case 5 evaluates access to a machine under SYN flooding attack using a list of 20 spoofed addresses, 400 batches, a batch-size of 2 and a delay of 1 second. Figure 18 shows the success rates and average delays in this test case. Note that the only noticeable effect the attack has on the machine protected by `synkill` is a small increase in the delay experienced during connection establishment. A load increase of the attacked machine is responsible for this delay.

Finally, test case 6 consists of evaluating the performance of `synkill` during an attack in which spoofed addresses are not repeated. The attack script sends one thousand batches of ten SYN packets each, with a delay of one second, using a new address for every batch. In some sense this is the worst case scenario for `synkill`, because it cannot utilize its learned knowledge of *bad* addresses and reset future connections that use the *bad* addresses as spoofed source addresses.

The measurements of this test case are displayed in Figure 19. We observed considerable delays and some failures in connection establishment attempts. They happened because the attacked machine ran out of swap space and empty process table entries to handle further incoming connections. For this attack the load in the attacked machine increased dramatically and at one point had ten processes waiting for attention in the ready queue. These observations suggest that even better performance of `synkill` can be expected if the host-based configuration optimizations discussed in Section 4.1.1 are used in conjunction with `synkill`.

At 32 MB or RAM and only 40 MB of swap space, our test machine quickly reached the point where not enough `inetd` processes could be started to handle all the artificially completed connections. At first it seems like no amount of memory or disk space can resist the continuous creation of `inetd` processes to handle an attack of this type. However, there is a maximum number of processes that will be created, and for this particular attack, and assuming that the time `synkill` waits to timeout a new IP address is ten seconds, the number should be roughly 100. As shown in Figure 13, each second ten new SYN packets appear and these trigger the creation of ten `inetd` processes. After ten seconds, IP addresses from the first batch will time out and appropriate RST packets will be generated. These will release ten processes, just in time for the next batch of SYN packets.

Typical SYN attacks have low packet rates, and part of the attractiveness of this attack comes from the fact that a small number of packets per minute (as little as 10) is sufficient for a successful denial of service attack. As shown here, the `synkill` approach can be an effective defense for SYN packet rates close to a thousand SYN packets per

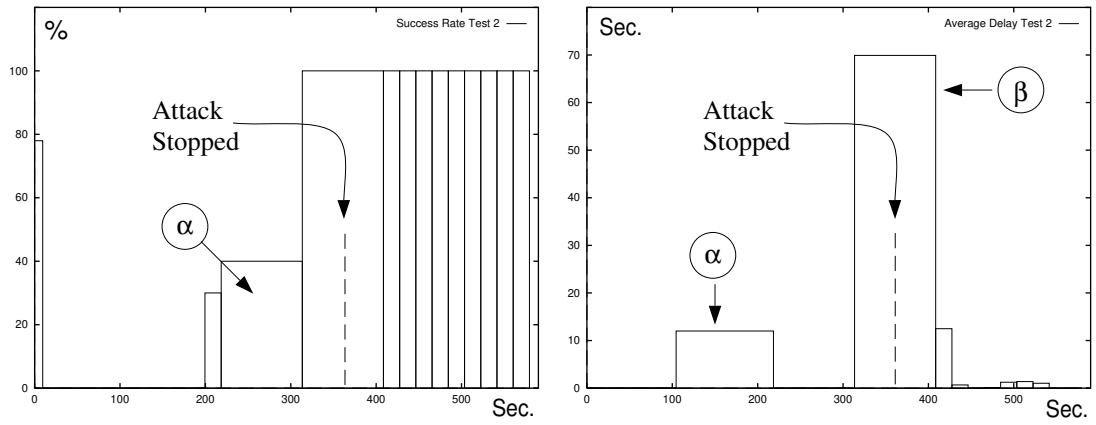

Figure 15. Test case 2: Connection establishment success rates and average delays while D is temporarily under attack, without active defense by `synkill`.

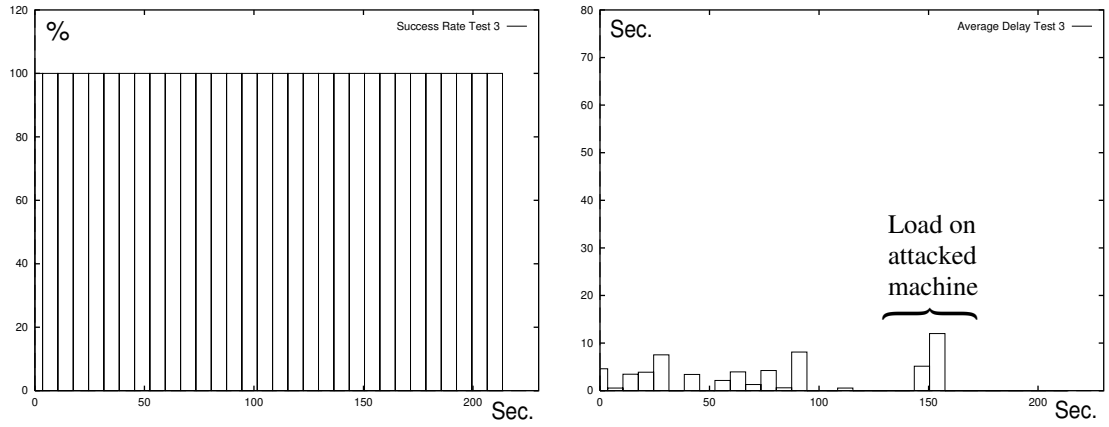

Figure 16. Test case 3: Connection establishment success rates and average delays.

minute (roughly 20 per second) in which spoofed addresses are not repeated. Higher volumes will result in a denial of service similar to that of opening thousand of telnet connections, and the attractiveness of the SYN attack will be diminished, if not gone.

7. Future Improvements

This section describes future improvements to `synkill` that would make the tool more effective against improved SYN flooding attacks.

7.1. Connection State Tracking

Currently, it is possible for an attacker to "teach" `synkill` *good* addresses that are in fact spoofed, by spoofing ACK or RST packets. That could be exploited to first teach `synkill` a spoofed address and then use that same address for a SYN flooding attack. Although `synkill` artificially completes each connection, thus avoiding port flooding, the attacker may still be able to start a large number of server processes in the target machine. This again leads to the degradation of service attack described in Section 2.2.3.

`Synkill` could respond to this improved attack by keeping state about all observed TCP connections on the LAN. That would make successful SYN flooding for an attacker as hard as sequence number prediction attacks. Fur-

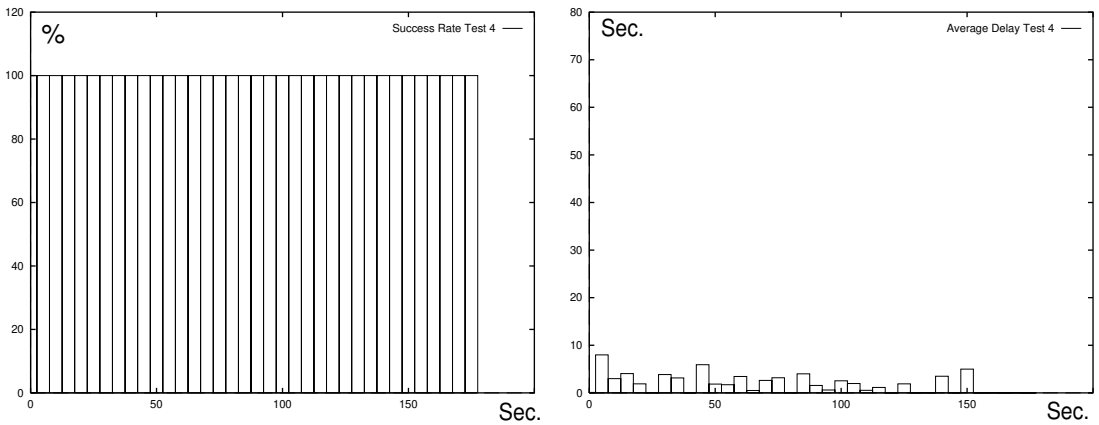

Figure 17. Test case 4: Connection establishment success rates and average delays.

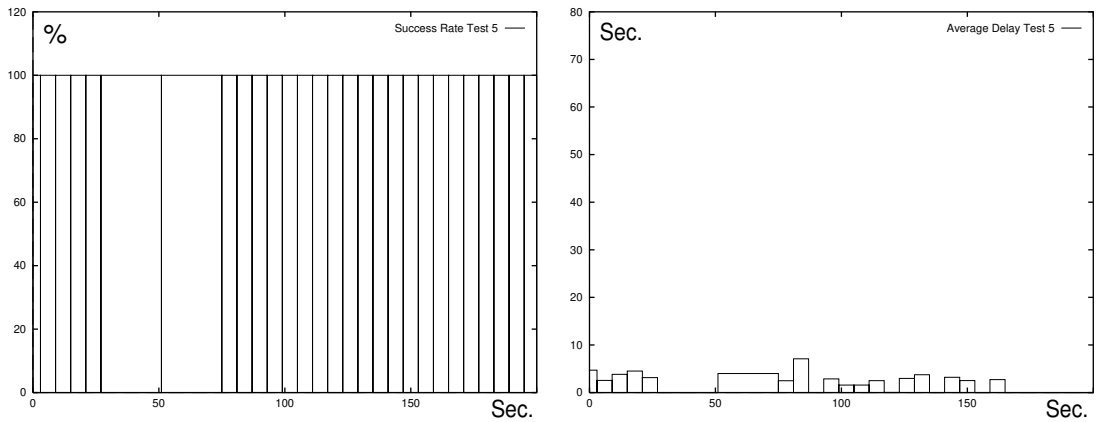

Figure 18. Test case 5: Connection establishment success rates and average delays.

thermore, this approach would facilitate the detection of other classes of network based attacks (see e.g., [3]).

7.2. Multiple-network Monitoring

Currently, `synkill` is implemented to monitor a single network interface. It may be desirable to allow the tool to monitor several network interfaces simultaneously, thus allowing the sharing of the acquired address classification database.

7.3. Attack Interval and Source Address Prediction

The basic idea of this approach is to protect against attacks based on timing or random number generator artifacts of the attack scripts, and not the generic attack method.

Our analysis of SYN flooding attack software showed that the delays between successive SYN packets in one batch, and between successive batches are almost constant. This same timing behavior can be observed at the targeted hosts, because all spoofed packets travel the same route over the internetwork and in a stable internetwork only little jitter is introduced.

The `synkill` software could therefore measure inter arrival times and use statistical models to predict the most likely arrival time of the next spoofed SYN packet. All SYN packets that fit into the predicted arrival times would be considered spoofed and immediately reset. The obvious response of the attacker will be to vary the delay between successive SYN packets.

Similarly, the random numbers used in many published exploitation routines are generated by cryptographically weak standard library routines. They do not follow good

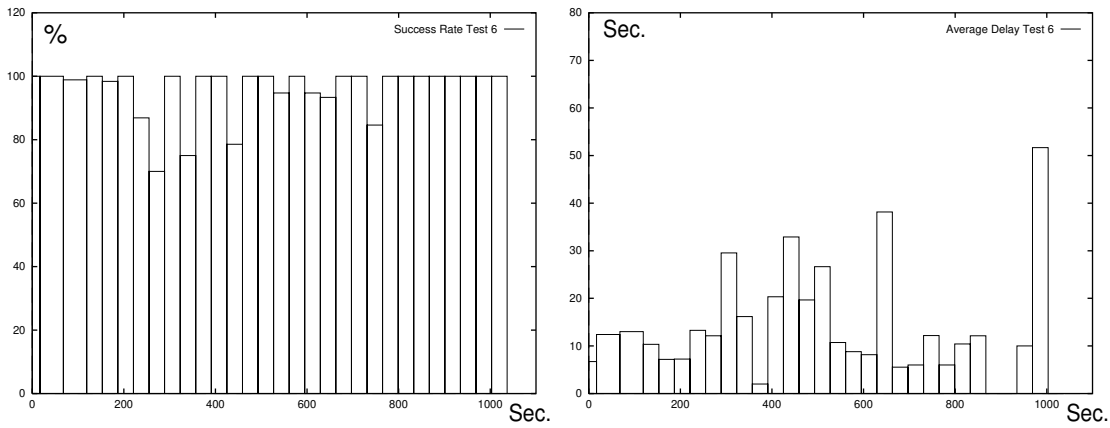

Figure 19. Test case 6: Connection establishment success rates and average delays.

cryptographic practices as described in [7, 10, 13]. We could implement a number of algorithms that automatically detect and predict pseudo random number sequences generated by simple common generators and use predicted pseudo-random IP addresses to identify malicious packets quickly. Again, there is an obvious countermeasure on the side of the attacker to harden attack implementations against these artifacts.

7.4. Trusted Address Space Ranges

Once the source address filtering mechanisms discussed in Sections 4.1.2 and 4.2 become more widely implemented a limited IP address space will be available for spoofed source addresses. `Synkill` could incorporate information about these secured address space ranges and automatically include them in its address preprocessing steps.

8. Conclusions

This paper has described and analyzed a network based denial of service attack, called *SYN flooding*. It has contributed a detailed analysis of this attack and a description and discussion of existing and proposed countermeasures. Furthermore, it has introduced a new solution approach, explained its design, and evaluated its performance.

The design is based on the philosophy that this active anomaly detection tool can detect the conditions of a SYN flooding attack and react appropriately to defeat, or at least lessen the impact of, an attack. `Synkill` neither requires any special hardware (such as particular firewall products), nor certain operating systems, network stacks, or even modifications in the protected end systems. Our software is highly portable, extensible, and easily configurable.

Our evaluation of the tool shows that `synkill` is capable of effectively protecting all machines on a LAN against a wide range of attack configurations. Many of the lessons learned from this study can be applied to the protection against other denial of service attacks.

Acknowledgments

We would like to thank Gustavo Rodriguez-Rivera for help with the design of the garbage collector and members of the security seminar at the Department of Computer Sciences at Purdue University for fruitful technical discussions. Suggestions from the anonymous referees helped improve the presentation.

References

[1] Cisco Systems Inc. *Defining Strategies to Protect Against TCP SYN Denial of Service Attacks*, September 1996.

[2] D. E. Comer. *Internetworking with TCP/IP*. Prentice–Hall, Englewood Cliffs, New Jersey, third edition, 1995.

[3] Computer Emergency Response Team (CERT), Carnegie Mellon University, Pittsburgh, PA. *IP Spoofing Attacks and Hijacked Terminal Connections*, Jan. 1995. CA-95:01.

[4] Computer Emergency Response Team (CERT), Carnegie Mellon University, Pittsburgh, PA. *TCP SYN Flooding and IP Spoofing Attacks*, Sept. 1996. CA-96:21.

[5] E. Corcoran. Hackers strike at N.Y. Internet Access Company. *The Washington Post*, Sep. 12, 1996.

[6] daemon9, route, and infinity. Project neptune. *Phrack Magazine*, 7(48), 1996.

[7] D. E. Eastlake, S. D. Crocker, and J. I. Schiller. *RFC–1750 Randomness Recommendations for Security*. Network Working Group, Dec. 1994.

[8] J. Fairlane. Flood warning. *2600*, 13(2):6–11, Summer 1996.

[9] P. Ferguson. Network ingress filtering. Internet draft, Cisco Systems, Inc., September 1996.
[10] S. Garfinkel and G. Spafford. *Practical UNIX & Internet Security*. O'Reilley & Associates, Inc. Sebastopol, CA., second edition, 1996.
[11] M. Graff. *Sun Security Bulletin 00136*. Mountain View, CA, Oct. 1996.
[12] Internet Security Systems. *RealSecure User's Guide and Reference Manual*, 1996. Available at `http://iss.net/RealSecure`.
[13] D. E. Knuth. *The Art of Computer Programming, Volume 2*. Addison–Wesley Publishing Company, Inc., second edition, 1981.
[14] L. S. Laboratories. *Livermore Software Labs. Announces Defense agains SYN Flooding Attacks*, October 1996.
[15] C. P. S. T. Ltd. *TCP SYN Flooding Attack and the* FireWall-1 *SYNDefender*, October 1996.
[16] J. Postel. *RFC–791 Internet Protocol*. Information Science Institute, University of Southern California, CA, Sept. 1981.
[17] J. Postel, editor. *RFC–793 Transmission Datagram Protocol*. Information Sciences Institute, USC, CA, Sept. 1981.
[18] Y. Rekhter, B. Moskowitz, D. Karrenberg, and G. de Groot. *RFC–1597 Address Allocation for Private Internets*. Network Working Group, Mar. 1994.
[19] R. W. Stevens and G. R. Wright. *TCP/IP Illustrated, Volume 2, The Implementation*. Prentice–Hall, Englewood Cliffs, New Jersey, 1995.
[20] B. Ziegler. Savvy Hacker Tangles Web For Net Host. *The Wall Street Journal*, Sep. 12, 1996.

A TCP State Machine

Figure 20 depicts the TCP state machine (Figure courtesy of Douglas E. Comer, [2]).

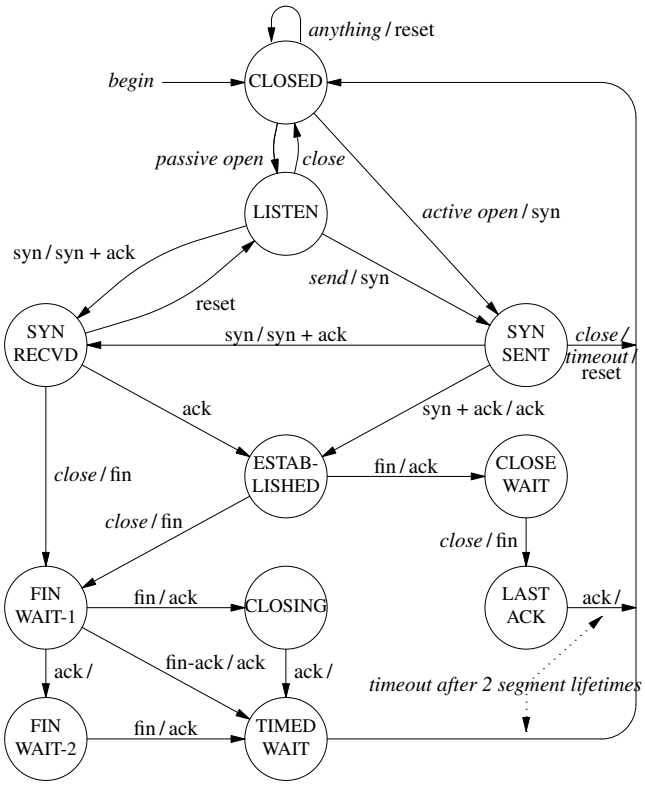

Figure 20. The TCP finite state machine