

Maximizing Firewall Availability

Techniques on Improving Resilience to Session Table DoS Attacks

Version 1.8, 06/15/2002

Stephen Gill

E-mail: gillsr@cymru.com

Public Release: 10/15/2002

Contents

Credits	2
Introduction	3
The Problem	3
The History	5
The Attacks	6
C ² -Flood	6
SYN-Flood	8
UDP-Flood	9
The Vendors	10
Netscreen	10
Check Point	11
Cisco	12
The Counter-Attacks	13
Checksum Validation	13
INIT Flow Timer Optimization	14
Session Table Expansion	14
Session Table Limiting	14
Rate Limiting	15
Aggressive Aging	15
TCP Proxy	17
TCP SYN Cookies	18
TCP Fastpath	18
TCP SYN Required	19
UDP / ICMP Flood Protection	19
END Session Rapid Removal	19
TCP RST	19
ICMP Echo-Reply	20
ICMP Port Unreachable Mapping	20
Proof of Concept	20
The Matrix	22
Uncovered Bugs	22
Conclusion	23
References	23

Credits

My thanks go out to the following individuals who assisted in some of the firewall testing that contributed to the writing of this paper:

- Dave Klein [dklein@netscreen.com] – Netscreen 500 testing, feature requests, and Netscreen liaison.
- Lael Black [me@laelblack.com] – Netscreen 5 testing.
- Larry Edie [ledie@attbi.com] – Cisco PIX 515 testing.

Introduction

Modern day firewalls are expected to withstand ever increasing amounts of traffic loads while providing higher resilience and lower latency. Along with these elevated goals, comes a greater responsibility to be conscientious of all aspects of security whilst optimizing the use of a finite amount of resources.

As firewall filtering technologies have improved, so have some techniques used to exploit them. One such method of abusing a firewall is to search for unique ways of expending all of a firewall's resources with a minimal amount of effort. This in turn causes a Denial of Service for all devices that sit behind it.

This paper describes a new danger, the C²-Flood (Crikey Checksum Flood), uncovered by the author that affects most stateful firewall vendors to date (e.g. Netscreen, Check Point, and Cisco). It also covers techniques, both current and future, that can be employed to protect oneself against it and similar types of DoS Attacks.

The Problem

The attacks described in this paper are adept at exploiting the very heart of most stateful firewalls: the session table. As the reader is likely already very well aware, the firewall session table is designed to increase security and performance by associating individual packets with their respective flows. The session table keeps track of the state of individual flows through the careful observation of packet direction, packet types, and implementation of flow timers.

This principle works very well for TCP traffic where flows are generally in one of three states: beginning (SYN, SYN-ACK, ACK), middle (ACK, PSH), or end (FIN, FIN-ACK, ACK, RST). However, given that UDP and ICMP are stateless, firewalls must create state information by examining packet contents and keeping close tabs on session timers.

Flows can move from the unestablished state, to the established when the firewall sees that a valid TCP handshake has taken place. Once a TCP flow reaches the established state, its timeout value is generally increased to a much greater length of time such as 30 minutes for Netscreen or 60 minutes for Check Point. Unfortunately this is much harder to do for UDP than it is for TCP given the nature of the protocol. For this reason, UDP virtual session timers generally remain fairly low, such as 40 seconds.

Session entries are generally comprised of at the very least, a 4-tuple that includes the source and destination IP and port, along with session timers, individual flow state (i.e. init, established, complete), and the protocol type (i.e. TCP, UDP, or ICMP). Other vendors might include additional

information such as corresponding interface MAC address or matching policy ID number. In theory, packets can be attributed to particular flows and thereby be allowed if they belong to that flow. The state table also serves to improve firewall performance since it can usually be safely assumed that a packet that belongs to a particular flow has already been allowed through the rulebase and thus, it does not have to be re-checked against the rulebase.

Firewalls are not capable of tracking an unlimited amount of flows that pass through them. Generally an upper bound on traffic flows is imposed due to a finite amount of memory and CPU power inherent in any architecture. Though some firewall vendors are better at others in maintaining higher limitations on the number of sessions they can track, most are not capable of handling new connections when the session table is at its maximum. If not implemented carefully, the state table can actually be used against a firewall by flooding this limited resource. In fact, the author has yet to encounter a firewall that is capable of handling **new connections** when state table is at full capacity. If a session table can be filled with illegitimate flows, it can keep a firewall from allowing legitimate flows to be established. This is the situation in which the attacks described in this paper thrive.

Consider Figure 1 below:

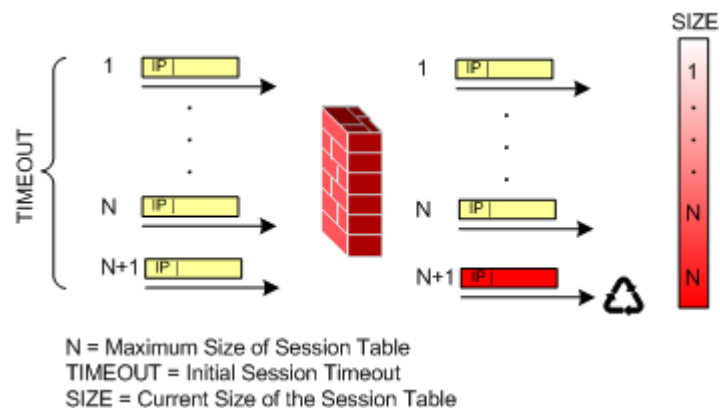


Figure 1 - Session Table Flooding

As packets belonging to new flows traverse the firewall, new session entries are added to the state table and they are set to the initial session timeout. If a greater number of flows pass through the firewall (N+1) than the maximum size of the session table (N) within the initial timeout, all sessions exceeding the limit will be dropped. The typical behavior requires that old entries be timed out of the session table before allowing new ones through. State table timers are much like a leaky bucket. If new flows are poured into the bucket too quickly, then the excess water will overflow the bucket.

Causing new connections to be dropped is not as difficult as one might imagine. Three attacks are described in this paper that can be used to flood a firewall's session table. Given the proper conditions, these streaming packet signatures can effectively block all new connections through a firewall. The C²-Flood (Crikey Checksum) is targeted directly at a firewall's state table, while the UDP-Flood, and SYN-Flood have been used by miscreants for many years for DoS attacks against particular servers.

First, we examine a bit of the history behind an attack known as an ACK-FLOOD that achieved somewhat similar results in the past on the Check Point and Netscreen platforms.

The History

In January of 2000 the author submitted a feature request with Netscreen to modify the way their firewall code functioned when in route mode. There was vulnerability in versions of ScreenOS prior to v2.6 that allowed TCP ACK packets, assuming a rule-base match, to blindly enter the session table (albeit with 60 second timers) without requiring an initial TCP handshake. The implications of this were that the session table could be filled based on a steady stream of spoofed TCP ACK packets sourced from random addresses destined to a host behind the firewall. Service timeout granularity was also grossly inadequate and only specifiable in 60-second increments. For reasons which the author will illustrate, this drawback can become quite dangerous in today's hostile environments.

The feature request was related to a security focus advisory regarding a DoS against the Check Point connections table [1]. In much the same way, TCP ACK packets were allowed to create entries in the session table with surprisingly large timeouts. This was previously considered a 'feature' in Check Point to allow for cases that required dynamic state table rebuilding. Rather than dropping existing connections, established connections were allowed to rejoin the session table (such as after a rulebase push) through active learning based on established ACK packets traversing the firewall assuming a rulebase match. Unfortunately, these sessions were initiated with the default TCP timeout of 3600 seconds, or 1 hour. Check Point posted a hack for this in their inspect code enabling the firewall to drop non-first TCP packets if they were not in the connections table instead of matching the rulebase [2]. This feature was subsequently implemented in mainstream code 4.1 Sp2 and above.

When the firewall session table fills up, connectivity through the firewall suffers greatly because no new sessions can be established until the old entries have expired.

The Attacks

One of the primary effects of the attacks described in this paper is quite straightforward: filling the firewall session table to capacity and blocking new incoming connections. Though we do not intend to cover all of the possible attacks against state tables, they should give the reader a good idea of their potential dangers.

C²-Flood

The first of such DoS attacks is somewhat new and directly centered on desynchronizing a firewall's state table with actual data flows. The author uncovered this issue while exploring different ways that can be employed to confuse stateful filtering firewalls. This attack has been affectionately named C² (C squared) or Crikey Checksum.

The packet flood was given such a name because it simply sends packets with invalid OSI layer 4 checksums (eg. TCP, UDP, ICMP) through a firewall to any host that sits behind it. Since most firewalls do not check layer 4 checksums for performance reasons, they will happily pass this traffic along assuming there is a rulebase match. Lab testing showed that Netscreen ScreenOS 3.1.0r5, Check Point NG FP2, and Cisco PIX 6.1.1 all forwarded packets with invalid layer 4 checksums. However, when the traffic reaches its final destination, it simply gets dropped as part of the layer 4 checksum validation. The destination server is not required to send any error messages to the source, and so the firewall remains unaware that the initial flow that was added to the state table is no longer valid. This flow will remain in the state table until the initial timeout expires.

Aside from server-side checksum verification, other applications or network configurations may also contribute to this Denial of Service by silently dropping packets. DNS servers may be configured to drop invalid DNS queries and Routers may silently discard packets destined to TCP port 179 without sending ICMP unreachable messages to the source. The real source of the problem is the inability of a firewall to block malformed packets that will not elicit a response from the target host.

In truth, the C² flood is not drastically different from any other standard flood, and it poses only moderately more effective results against firewall state tables. This is because in many cases, no response or some response will not matter to a firewall if a flow is still in its embryonic phase. The real differentiators are the cases where a firewall could easily imply that a connection has terminated such as by receiving a TCP RST, a UDP DNS Query response, an ICMP Unreachable message, or an ICMP Echo-Reply. Upon receiving any of these, it is presumably safe for a firewall to remove the associated session table entry because the connections have been fully terminated.

TCP poses a special case because a SYN-ACK response from the server is still not enough to complete an embryonic session. However, there are many cases in TCP where a RST might be received in response to a SYN such as:

- TCP wrappers are configured on the destination host to disallow particular networks but the flow is allowed by the firewall.
- The destination server is not listening on the target port.
- The destination server is using BSD socket binding and restricting connections to specific IP networks or hosts.
- The destination server sends a SYN-ACK response to the supposed connection source (be it spoofed or not), and in turn, the connection source (if existent) replies with a RST.

A C^2 flood guarantees that in 100% of the above cases, an end-session response will not be received or returned through the firewall. Another point worthy of mentioning is that in some firewall implementations such as Check Point, the embryonic timer will reset when a SYN-ACK response is received from the server. Most servers will actually retry sending a SYN-ACK response several times with exponential backoff before finally giving up. Firewall implementations that reset the embryonic timer upon receipt of a SYN-ACK response will be marginally more susceptible to a standard SYN Flood than a C^2 flood. This is because the embryonic session timer will only begin to expire after the last SYN-ACK response attempt is sent by the server.

If a steady stream of such packets were sent through a firewall at a fast enough rate, then they could potentially fill the session table with useless information and keep the firewall from servicing new connections while the session table was full. The issue is uncovering how much traffic is really required to wreak havoc on a firewall's session table. In theory, if we know how large a firewall's session table is, and if we know what the initial flow timeout is, then we should be able to determine how much traffic we would need to generate in order to fill a firewall's session table.

Table 1 lists the known session table limit and initial TCP session timeout for three major firewall vendors including Netscreen, Check Point, and Cisco. It also lists the standard TCP and UDP timeouts, along with the intervals at which the garbage collection task is run to clear out old entries. Based on these figures it is possible to compute the amount of traffic required to maliciously fill such a table within that initial timeout assuming that a given flow is permitted by the rulebase. The equation is: DoS Packets/s = Max_Sessions / Timeout. Assuming 64 byte packets, the amount of traffic this equates to can be calculated by multiplying packet size with the number packets per second. Keep in mind that these calculations do not take into account the garbage collection intervals or

any other traffic that may be traversing the firewalls which can detract from the amount of traffic necessary to fill the session table.

FIREWALL	MAX SESSIONS	TIMER (s) TCP	UDP	ICMP	GARBAGE	DOS PACKETS/s	DOS Kbytes/s	DOS Kbps
Netscreen 5	1,024	60	60	60	10	17	1	8
Netscreen 5XP	2,000	60	60	60	10	33	2	16
Netscreen 25	4,000	60	60	60	10	67	4	32
Netscreen 50	8,000	60	60	60	10	133	9	72
Netscreen 200	128,000	60	60	60	10	2,133	136	1,088
Netscreen 500	250,000	60	60	60	10	4,167	267	2,136
Netscreen 1000ES	300,000	60	60	60	10	5,000	320	2,560
Netscreen 1000SP	500,000	60	60	60	10	8,333	533	4,264
Check Point 4.1	25,000	60	40	30	0	417	27	216
Check Point 4.1 (tweaked)	50,000	60	40	30	0	833	53	424
Check Point NG	2,499,000	60	40	30	0	41,650	2,666	21,328
Cisco PIX 501	3,500	120	120	na	30	29	2	16
Cisco PIX 515E	125,000	120	120	na	30	1,042	67	536
Cisco PIX 525	280,000	120	120	na	30	2,333	149	1,192
Cisco PIX 535	500,000	120	120	na	30	4,167	267	2,136

Table 1 – Session Table DoS Estimate

A constant stream of spoofed TCP, UDP, or ICMP traffic equal to or above the threshold listed in the 'DoS Packets' column could guarantee that the state tables for each of the devices would remain full. Additional traffic would not be allowed to pass if there is no room to create new session entries on the firewall. As the reader can easily observe, this DoS potential is usually far less than the amount of traffic that a firewall is capable of handling. Hence, firewall resources can be easily taxed with a relatively low constant traffic stream.

For example, the Netscreen 1000ES is a Gigabit rated firewall. However, assuming default system parameters, its state table can be filled with a constant TCP stream of only 2.6 Megabits per second!

Using techniques similar to the C² flood, many other traffic signatures could be generated to desynchronize a firewall's state table. IDS evasion techniques are not far behind this approach – in fact, they are quite similar [9]. Packets that do not follow general protocol guidelines are prime candidates for such a line of attack. A few such examples might include the following: UDP packets with invalid header lengths, TCP packets with misconfigured flags, or ICMP echo-requests with a 'Zero' code field.

SYN-Flood

The TCP SYN-Flood has been around for quite some time, yet it continues to plague networks throughout the Internet on a daily basis. One major and perhaps lesser known effect of a SYN-Flood is to fill the state table of a firewall assuming no additional protections have been added to guard

against such an attack. Each inbound TCP SYN packet causes the firewall to add an entry into the state table to track the connection after it has passed a rulebase inspection.

Some existing mechanisms have proven helpful but not ultimately bulletproof when it comes to protecting servers against TCP SYN-Floods. When performing SYN-Flood resiliency testing on a Netscreen 500 running ScreenOS 3.1.0r5 it was discovered that not only did the TCP-Proxy feature not work, but that incoming flows were added to the state table even if they exceeded the predefined flood threshold. Other platforms were visibly susceptible to session table flooding caused by SYN-FIN floods even with TCP-Proxy enabled. This has been reported to the vendor and should be corrected in the next release of their code.

Running the same tests on Check Point NG FP2 using the SYNDefender Passive SYN Gateway showed no marked signs of improvements in session table protection. More information is available on this under TCP Proxy later in this paper. It is not known for sure whether or not the PIX suffers from similar issues with adding entries to the state table even if they exceed the SYN-Flood threshold.

Other mechanisms that proxy the initial TCP handshake similar to Netscreen's TCP Proxy have been evaluated for speed and resilience by third parties. Testing has shown that though Netscreen is generally capable of handling a greater amount of SYN Flood packets per second than Check Point Firewall-1 and Cisco PIX (when the code is functioning properly), this area could still use considerable improvement [5].

UDP-Flood

Much like the TCP SYN-Flood, the UDP-Flood has also been used by miscreants to fill up pipes and consume network resources. It generally consists of small UDP packets sourced from different addresses, destined for a specific IP and port. A suitable target might be a DNS server that filters everything but inbound connections to UDP port 53.

Though the UDP protocol is stateless by design, stateful firewalls attempt track UDP flows by storing session information in the state table much like they would for TCP connections. Careful attention must be given to session timers and flow direction since there is no inherent mechanism in the protocol for issues such as session setup and teardown. A stateful firewall must infer what is happening in a UDP flow since there are no state flags like there are for TCP.

Since state information is stored for every UDP connection, it is equally possible to negatively affect the firewall session table through the use of a UDP Flood. Netscreen provides a mechanism for limiting the amount of

UDP packets that are destined to a unique IP and port which is called UDP Flood Protection. However, in lab testing, it was shown that the UDP Flood Protection feature was broken and did not in fact block inbound connections that exceeded the predefined threshold. Interestingly enough, only connections that matched a 3-tuple of source port **and** destination IP and port would activate the UDP Flood Protection mechanism. This flaw has been reported to the vendor and should be fixed in subsequent releases of their code.

If no flood protection mechanism is in place, a firewall session table could be easily filled by generating spoofed packets at a rate that exceeds the size of the session table within the initial UDP timeout. This is identical to the effect of a TCP SYN-Flood described above. Based on the UDP initial timeouts summarized in Table 1, the number of packets necessary to fill a firewall session table is calculated using the same formula used for TCP.

The Vendors

In the interest of ascertaining what the general session table behavior was across the firewall market, the three largest firewall competitors were chosen as the base platforms for data discovery: Netscreen, Check Point, and Cisco. More extensive testing was performed on the Netscreen platform which served as the primary testbed for further vulnerability testing and analysis.

Following is a quick run down on viewing the session table on each of these platforms, along with relevant details that might differ between architectures.

Netscreen

TCP, UDP, and the ICMP protocol are all tracked statefully on the Netscreen products. This means that for each of these packets an entry will be added to the state table to keep track of the connection throughout its lifetime assuming it has initially matched a firewall policy. Some vendors do not keep track of state for ICMP packets, the merits of which are outside the scope of this paper.

In order to obtain the maximum number of session table entries for a Netscreen, one can issue the *'get session'* command on the CLI and look for the maximum entry. This command will also display the current size of the session table along with every entry it contains. Figure 2 presents the partial view of the session table on a Netscreen 5XP, of which the maximum and allocated sessions are highlighted. The best method of determining if the firewall is running out of session table entries is to check the *'alloc failed'* counter.

```

NAT, sessions 7, allocated 8, maximum 2048, alloc failed 0
id 68, vsys id 0, flag 00004000/00, pid 0, did 2, time 177
0(21):10.10.10.10/3880->16.136.233.129/5050,6,00d059aacec4,vlan 0,tun 0
1(20):1.134.159.2/1046<-16.136.233.129/5050,6,00206f0fb97f,vlan 0,tun 0
id 79, vsys id 0, flag 00004000/00, pid 0, did 2, time 178
0(21):10.10.10.10/3886->16.188.8.52/5190,6,00d059aacec4,vlan 0,tun 0
1(20):1.134.159.2/1051<-16.188.8.52/5190,6,00206f0fb97f,vlan 0,tun 0

```

Figure 2 - Netscreen 5XP State Table

The initial flow timer can be tweaked with the “*set flow init <time>*”, where time is specified in 10 second increments, no smaller than 20 seconds. Since the garbage collection routine runs every 10 seconds, the smallest initial timer actually equates to somewhere between 20 and 30 (20+10) seconds. For backwards compatibility, if a number between 1 and 5 is entered, then the output is interpreted as being in minutes not in seconds.

During testing, it was found that this feature was broken on the Netscreen 5 running ScreenOS 2.6.1r7.1. If the user specifies a flow init timer in seconds, it actually gets interpreted as minutes instead! Thus, the smallest configurable init timer is 60 seconds instead of the standard 20. Testing also showed that the output of the ‘*get session*’ command in ScreenOS 3.1r0.5 displays timers in 10 second increments rather than in seconds as was previously known. Keep this in mind when working with Netscreen products on the CLI, since this can actually cause quite a bit of confusion.

Check Point

The latest release of Check Point software, Check Point NG, now also allows for the configuration of the initial TCP, ICMP, and the end TCP session timeout in addition to the already available UDP virtual session timeout.

Individual flow timers can be viewed by perusing the state table through the command line with the “*fw tab -t connections -u*” command visible in Figure 3. The last two numbers listed for every flow will be the current and maximum timeout value for that flow. Adding a trailing “-s” will simply return the number of active entries in the state table.

```

localhost:
----- connections -----
dynamic, id 8158, attributes: keep, sync, expires 60, refresh, limit 2499000,
hashsize 65536, kbuf 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30, free function
c4070b0c 0
<00000000, 0a0a0a75, 00001073, 0a0a0af8, 00000016, 00000006; 0001c001, 00004000,
00000001, 00000e10, 00000000, 3cdd5681, 00000000, f800000a, 000007b6, 00000001,
00000001, ffffffff, ffffffff, 00000000, 00000000, 00000000, 00000000, 00000000,
00000000, 00000000, 00000000, 00000000, 00000000, 00000000; 3598/3600>
<00000001, 0a0a0af8, 00000016, 0a0a0a75, 00001073, 00000006> -> <00000000, 0a000075,
00001073, 0a0a0af8, 00000016, 00000006> (00000005)
<00000001, 42869f81, 00000208, 42869f87, 00000208, 00000011> -> <00000000, 42869f81,
00000208, 42869f87, 00000208, 00000011> (00000002)
<00000001, 42869f87, 00000208, 42869f81, 00000208, 00000011> -> <00000000, 42869f81,
00000208, 42869f87, 00000208, 00000011> (00000005)
<00000000, 42869f81, 00000208, 42869f87, 00000208, 00000011; 00010001, 00004000,
00000001, 00000028, 00000000, 3cdd54a2, 00000000, f800000a, 000007b6, 00000000,
ffffffff, ffffffff, ffffffff, 00000000, 00000000, 00000000, 00000000, 00000000,
00000000, 00000000, 00000000, 00000000, 00000000; 38/40>

```

Figure 3 - Check Point NG FP2 State Table

The Check Point FW-1 state table is not as easy to understand at first glance due to hexadecimal notation and the lack of column headers. Once the administrator is familiar with the significance of each entry, it is much easier to troubleshoot [10].

A standard Check Point Firewall-1 4.1 installation is capable of handling up to 25,000 concurrent connections in its state table. If tweaked, it can be configured to handle upwards of 50,000 connections [3]. In theory, Check Point NG FP2 is now capable of processing up to 2,499,000 connections, as displayed by the limit statement above. This particular firewall gateway has been tweaked to its maximum capacity through a new option in the FW-1 gateway properties window. In practice, this presupposes the availability of sufficient system memory – 1 to 4 GB of Ram in this case. This enhanced limit may warrant further research on the performance impact of similar session table lookups.

Up until version Firewall-1 v. 4.0, ICMP was not tracked statefully and required manual modification of the INSPECT code [6]. FireWall-1 4.1 and above *theoretically* perform stateful inspection of ICMP if the policy property is enabled. However, some reports have been received about this not working as it should.

Cisco

The Cisco PIX on the other hand does not keep state for ICMP. This was verified by performing a “*show conn*” to view the session table and observing that entries for pings never made it into the table. This fact coupled with 30 second garbage collection intervals makes it a bit more difficult to measure the effects of a full session table. Figure 4 contains the output of a Cisco PIX session table. Curiously, the output lists 9 entries including TCP and UDP connections, yet only the TCP connections are counted. How flows are counted towards the maximum number of session entries is not entirely certain.

6 in use, 6 most used							
TCP	out	10.165.201.1:80	in	10.3.3.4:1404	idle	0:00:00	Bytes 11391
TCP	out	10.165.201.1:80	in	10.3.3.4:1405	idle	0:00:00	Bytes 3709
TCP	out	10.165.201.1:80	in	10.3.3.4:1406	idle	0:00:01	Bytes 2685
TCP	out	10.165.201.1:80	in	10.3.3.4:1407	idle	0:00:01	Bytes 2683
TCP	out	10.165.201.1:80	in	10.3.3.4:1403	idle	0:00:00	Bytes 15199
TCP	out	10.165.201.1:80	in	10.3.3.4:1408	idle	0:00:00	Bytes 2688
UDP	out	10.165.201.7:24	in	10.3.3.4:1402	idle	0:01:30	
UDP	out	10.165.201.7:23	in	10.3.3.4:1397	idle	0:01:30	
UDP	out	10.165.201.7:22	in	10.3.3.4:1395	idle	0:01:30	

Figure 4 - Cisco PIX State Table

ICMP traffic such as echo-requests will not be negatively affected when the state table is at capacity because they do not require that state information be stored. The long garbage collection intervals also create certain periods where traffic does in fact make it through the firewall since so much time has elapsed after the last session table cleanup. Nevertheless, the default initial timeout of 120 seconds for TCP and UDP is twice as much as the Netscreen and Check Point.

Issuing the “*show timeout*” will display the global system timeout settings but the initial session timeout is not configurable. Unfortunately this causes the firewall to be at an even greater risk of reaching the maximum limit on its session table, while providing no avenue for rectification.

The Counter-Attacks

After reading about how easy it is to fill the session table on even the highest rated and most successful firewalls on the market today, the administrator might be wondering what if anything can be done to protect himself against such dangers. There are in fact quite a few methods of improving a firewall’s resilience to session table based DoS attacks, though most of them require feature additions to existing code. The suggestions proposed in this paper have been submitted to Netscreen, Check Point, and Cisco to address some of their state related limitations.

Some of the ideas proposed towards state table protection are not new, but have been included in this paper for the sake of completeness. Others may already be available in some firewall products. The goal of this section is to educate the firewall vendor and security administrator on methods of protection. Recommendations on mitigation strategies are not listed in order of importance.

Checksum Validation

The C² Flood has demonstrated how easy it is to desynchronize a firewall’s session table with what is actually happening on the wire. While a firewall may be keeping track of an invalid connection, the destination server may already have dropped the session due to an invalid OSI layer 4 checksum. Evaluating checksums at layer 4 increases the level of

security on a firewall by having the capability to disallow packets failing a simple checksum test.

To date, most vendors such as Netscreen, Cisco, and Check Point have chosen **not** to evaluate layer 4 checksums likely due to the amount of increased overhead involved in such a task. Interestingly enough, Netscreen's TCP Proxy and Check Point's SYNRelay both do not perform Checksum Validation for TCP SYN packets and will gladly respond with a TCP SYN-ACK to a SYN with an invalid layer 4 CRC.

The tradeoffs between performance and security are often revisited. A greater firewall differentiator is the ability to perform traffic validation and normalization such as checksum verification at higher levels in the protocol stack **without** suffering from the symptoms of state table flooding or performance degradation.

INIT Flow Timer Optimization

Each of the firewall platforms evaluated displayed excessively high default system parameters for initial flow timers. Even worse, some platforms did not offer the ability to change these parameters. Rather than selecting a 60 or 120 second default flow timer, a more realistic value would be in the range of 10 to 20 seconds with the option of tuning according to individual need. Realistically, it should not take more than 10 seconds to establish a flow. Bringing a 60 second timer down to 10 seconds could potentially require six times as much nefarious traffic to fill the session table.

Session Table Expansion

On the other end of the spectrum, the size of the session table has a lot to do with how quickly it can fill up. Though it would potentially require a significant increase in memory and CPU cycles, a larger session table configured either dynamically or statically would allow a firewall to track a greater number of concurrent connections. The amount of system memory and CPU cycles available to perform entry lookups, flow ageing, and garbage collection may be the biggest barrier to entry for this optimization, not just the software that is running on it.

Session Table Limiting

The ability to limit entries in the session table based on source IP, destination IP, or even on a per policy basis would greatly reduce the risk of overrunning the state table. Currently, Netscreen products enjoy the ability to limit flows based on the IP address they originate from, but not the IP address they are destined to. Source and destination IP session limiting would be generic global flags with a specific number of flows allowed to be sourced from or destined to IP addresses on either side of the firewall.

Check Point supposedly has the ability to perform session table limiting via its INSPECT code through the netquota and serverquota features though the author has not performed any testing with these features. They will hopefully be made available in the GUI in the future.

Having this ability on a per policy basis could also be considered an additional method of rate shaping or traffic policing, since only a specific number of flows would be allowed to pass at any given time; those exceeding the threshold would be dropped.

The unfortunate drawback to this approach is that it makes it more difficult to harm a firewall, but much easier to restrict valid new connections to a server. The number of connections when limited per destination would presumably be much smaller than the size of the entire session table. Thus, a smaller amount of traffic would be necessary to fill the number of sessions allowed to a particular destination. However, when the firewall health is at stake it may be prudent to chose network survivability over server availability.

Rate Limiting

Similar to session table limiting, rate limiting restricts predefined traffic flows to configured bandwidth parameters. Generally, traffic parameters that can be defined in a firewall policy or router access-list can also be rate limited. Rate limiting can be performed on upstream routers such as using Cisco IOS's Committed Access Rate, or on firewalls that support rate limiting such as Check Point and Netscreen.

As a general rule of thumb, TCP flows comprise a vast majority of IP traffic. For this reason, UDP and ICMP traffic can usually be safely rate limited on edge networks. This technique can usually cut down a large percentage of floods that are based on these two protocols but it requires a good understanding of individual network traffic baselines. If a server or network device is being overrun, rate-limiting is a sure way of limiting the size of the network pipe based on a traffic flow, potentially allowing room for legitimate traffic to pass that does not match the same pattern.

Aggressive Aging

The term aggressive aging was coined by the author in an effort to describe the need for optimization of timers within a firewall's session table. Rather than disallowing new connections when the state table is full, a stateful firewall should have the capability of aggressively timing out its oldest entries to make room for new connections. In theory, the oldest connections are those that are least likely to resurface, thus they should not take priority over new connections. Established connections on the other hand should have the greatest priority of all over embryonic

connections since they have the highest likelihood of legitimacy. Aggressive aging could further be enhanced by allowing users to selectively enable aging on specific policies rather than using it globally.

Though the decision to implement such an algorithm would be completely up to the vendor, perhaps following Cisco's TCP Intercept architecture found in their IOS would be a great place to start. TCP Intercept offers a high and low watermark for enabling and disabling the automatic removal of entries from its TCP state table. Additionally, a knob is available to enable the removal of random connections instead of the oldest ones. More details surrounding Cisco's TCP Intercept are described later in this paper.

To the author's knowledge, Netscreen, Check Point, and Cisco do not yet implement aggressive aging for their state tables, although Cisco does provide a similar technique through TCP Intercept. An example of the effects of aggressive aging on a state table can be found in Figure 5 below.

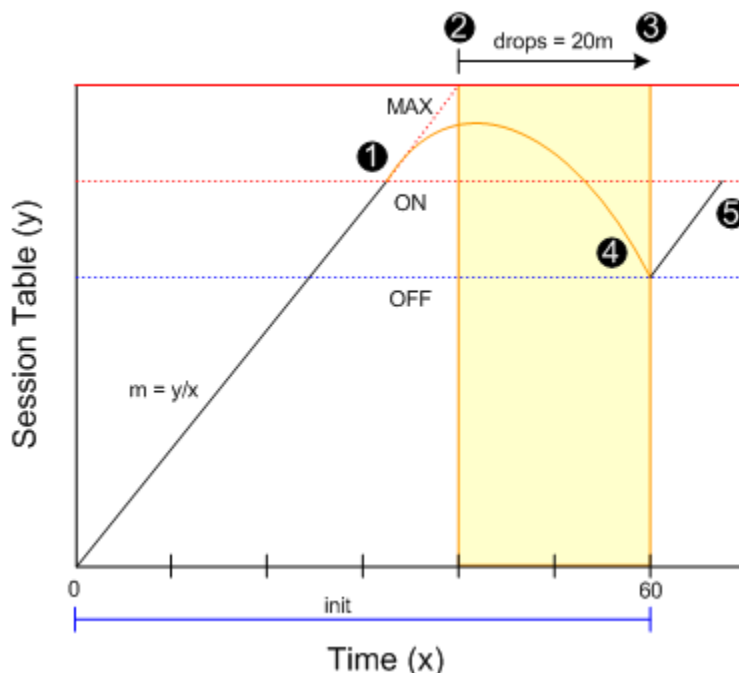


Figure 5 - Aggressive Aging Comparison

In step (1), as new entries are added to the session table, it will eventually fill up to capacity if enough old entries have not sufficiently timed out. For the amount of time between phases (2) and (3), all new connections are dropped because the firewall is unable to service new requests when the session table is full. The number of connections dropped can be calculated by multiplying the time by the rate at which new connections are coming in ($20 * m$). This is a far cry from the optimal scenario.

If aggressive aging were enabled on this firewall with an enable watermark of Y(1), old entries could begin to be aggressively timed out until the disable watermark was reached at Y(4). At this point, all would return to normal and aggressive aging would be disabled until needed once again.

TCP Proxy

One feature that is available in most firewall platforms to date is akin to Netscreen's TCP Proxy, Cisco's TCP Intercept, and Check Point's SYNDefender and SYNRelay. Vendors have chosen to implement this feature in slightly different ways, some with greater success than others. When enabled, the initial TCP handshakes for connections exceeding a minimal threshold are proxied by the firewall with the assumption that it is capable of handling inbound traffic at a much higher rate than a server would. Connections exceeding the TCP Proxy maximum threshold should be dropped and not tracked in the session table.

As previously noted, testing has shown that this is not necessarily the case for Netscreen's current ScreenOS. TCP connections are still tracked if they exceed the size of the TCP Proxy packet queue, which results in the potential flooding of the state table even with this feature enabled. Future revisions of ScreenOS will hopefully be addressing this particular issue, by restricting the entries in the state table to match what is tracked in the TCP Proxy queue.

This problem also seems to present itself with Check Point's SYNDefender. Enabling this feature showed no visible signs of improvement in session table attack mitigation. Additionally, the current maximum number of connections that can be tracked is limited to 10,035. Entries that exceed the SYNDefender threshold will not be policed. It should be noted that SYNRelay, a more CPU intensive SYN Flood protection mechanism, displayed better resilience at limiting the number of session table entries but at the cost of much greater performance degradation and higher latency.

Of key importance is that global firewall performance is not degraded by enabling this additional option. In many cases in the past, enabling this feature on a production firewall, be it Cisco, Check Point, or Netscreen, has only resulted in decreased firewall performance with little added benefit. It is highly recommended that the security administrator fully understand the dynamics of this feature and perform independent throughput testing in accordance with his or her network bandwidth requirements before enabling it on a production firewall.

TCP SYN Cookies

TCP SYN cookies are a newer technique used to prevent SYN flooding attacks. The SYN cookies concept was originated by D. J. Bernstein and Eric Schenk in 1996, and it is now a part of Linux and FreeBSD [11]. It has been gaining in popularity due to its ability to provide improved protection against TCP SYN Flooding while using a much smaller amount of resources. Interestingly enough, it does not maintain traffic state until the initial TCP handshake has completed.

A TCP SYN Cookie is comprised of the following: an incremental 64 second counter (t), a TCP MSS (Maximum Segment Size) value, and a hash of a server-selected secret function of the client IP and port, the server IP and port, and the incremental counter (t). When the firewall notices that traffic has reached a predefined SYN-flood threshold, it can send back a SYN-ACK to the client(s) with an Initial Sequence Number of the TCP SYN Cookie for that connection. When it receives the final ACK in the TCP handshake, it checks that the secret function works for a recent value of the counter (t). The chances of creating a valid fake ACK response packet are very slim. At this point it can rebuild the original TCP SYN from the encoded MSS value in the Cookie and patch the flow with the server. After the flood is subsided, the firewall can resume normal operations by disabling the use of SYN Cookies.

A few limitations to the TCP SYN Cookies approach are that while under attack, a firewall must reject TCP options such as large windows (RFC 1323) and selective ACK (RFC 2018), and it must use one of the eight MSS values that it can encode. This may result in decreased network efficiency during an attack, but it is certainly better than passing no packets at all!

As of the time of this writing, TCP SYN Cookies are not yet available in production code on any of the vendors evaluated.

TCP Fastpath

An alternative to proxying inbound TCP connections at the firewall is to configure it so that it only inspects packets that have the TCP SYN bit set for a particular service. Though this decreases security somewhat, it may be a viable alternative for improving firewall performance and optimizing state table usage. By only checking packets with the SYN bit, the firewall no longer needs to keep state information for packets matching that service or policy. Packets without the SYN bit still must match the rulebase, but are assumed to be coming from established TCP connections.

A security administrator might choose to enforce TCP Fastpath for a highly used firewalled service such as HTTP where state information is not

as important to save on precious state table entries. This has proven particularly useful in specific instances of session table flooding caused by inbound DoS attacks on port 80. Even if it becomes a temporary measure to keep the firewall from dropping new connections, this feature can be quite effective because it disables state checking for that TCP service. To the author's knowledge this feature is only available on the Check Point platform to date.

TCP SYN Required

SYN Required is very similar to TCP Fastpath, except that it tracks the state of TCP connections. The benefit of this feature is that a session cannot be added to the state table without first going through the initial TCP handshake. This is no different than standard TCP behavior and thus it is generally recommended, except in asynchronous routing situations. A firewall must be able to see the entire TCP handshake (SYN, SYN-ACK, ACK) in order to properly keep state for a TCP connection.

Versions of Netscreen's ScreenOS 2.6 and above added the command, "*set flow tcp-syn-check*" to require an initial TCP SYN before allowing a connection to enter the session table. However, this flag is not enabled by default. Some network scenarios such as those with circular routing may require that this feature be left disabled. However, by neglecting to enable this feature, the administrator runs a greater risk of potential session table flooding. The Cisco PIX is not vulnerable to ACK Floods.

More advanced port scanning applications such as Nmap [8] have the ability to stealthily scan for open ports using non-SYN TCP packets. An added benefit of the SYN Required feature is that it would essentially block such scans from functioning.

UDP / ICMP Flood Protection

A firewall is not able to proxy UDP or ICMP connections such as with TCP because they are stateless protocols. However, it should still have the ability to limit and generate appropriate alarms for inbound UDP and ICMP connections destined to a single IP according to predefined thresholds. As was previously shown, a UDP Flood could be used to maliciously fill a firewall's session table and cause it to drop new inbound connections. Connections exceeding a certain bandwidth or packet rate should not be allowed to enter the session table.

END Session Rapid Removal

TCP RST

Stateful firewalls should handle TCP RST packets by immediately marking the appropriate entry in the state table for deletion if one exists. Testing on the Check Point platform showed that this was not in fact the case.

TCP RST packets are treated like TCP FIN packets when in fact they don't need to be. Connections that are abruptly closed with a TCP RST do not need to be acknowledged whereas connections that are cleanly terminated require full acknowledgement from both sides. A larger window is necessary for clean closures to ensure that both sides receive all packets in the TCP close sequence. Since TCP RST packets are not acknowledged, their session can be immediately marked for deletion without any loss of functionality.

ICMP Echo-Reply

Similar to TCP RST packets, ICMP Echo-Replies are the second and last packet in an ICMP ping flow. For this reason, firewalls that track ICMP statefully can safely mark the session for deletion once an ICMP Echo-Reply is received in response to an ICMP Echo-Request. Testing on the Check Point platform showed that this was not in fact the case. ICMP Echo-Replies remain in the state table until the timeout expires when they don't need to.

ICMP Port Unreachable Mapping

Testing on the Netscreen and Check Point platforms has shown that ICMP port unreachable messages are not used to mark corresponding flow entries for deletion. Somewhat akin to TCP RST messages, ICMP port unreachable messages serve to inform an end station that a port is not reachable. Port unreachable messages should be matched against the appropriate state table entry and marked for deletion instead of waiting for the standard session timeout to expire.

Firewalls should NOT blindly remove entries from the session table without full verification that the ICMP data containing the original TCP header and sequence numbers fall within acceptable sequence number ranges for that flow. This sanity check would keep miscreants from being able to easily spoof ICMP Unreachable messages to negate specific flows on a firewall. Unless the firewall already tracks acceptable TCP sequence numbers, the cost of implementation will likely outweigh the benefits of such an addition.

Proof of Concept

To simulate each of the attacks described in this paper, the author developed a very simple program in C named scooter [3] using the standard Libnet [7] libraries. Scooter was designed to generate the following types of packet streams: C² Floods, TCP Floods (setting any TCP Flag), UDP-Floods, and ICMP Floods. The program makes use of some of the rate-limiting features available in the FreeBSD kernel on the dumynet interface to restrict traffic flows to a specified bandwidth capacity. This technique allows one to generate the smallest amount of traffic necessary to cause the session table flooding conditions.

It should be noted that bandwidth calculations made by the scooter program are not analogous to standard packet sizes. The program does not account for data that resides below the IP layer. For this reason, it is easiest to calculate breaking point estimates on packets per second. Figure 6 displays the overall syntax and help screen generated by scooter.

```

NAME
    ./scooter v. 1.1 04/06/2002 - written for FreeBSD 4.5 i386 by [scooby]
USAGE
    ./scooter -p {udp | tcp | icmp} -s src_ip/mask:port -d dst_ip/mask:port
    [-c count] [-m meter] [-b bw] [-f flags] [-i] [-v]
DESCRIPTION
    Generates UDP/TCP packets
    -p > Protocol: udp, icmp, or tcp
    -s > Source IP/Bit Mask:Port (-1 = random port)
    -d > Destination_IP/Bit_Mask:Port (-1 = random port)
    -c > Number of packets to generate (unlimited = default)
    -m > Print packet rate after m packets (15000 = default)
    -b > Bandwidth rate-limiting for flow in K, M, KB, MB, B
    -f > TCP flags: sfarup (syn,fin,ack,rst,urg,psh) (null = default)
    -i > Ignore checksum
    -v > Verbose
EXAMPLES
    To generate 10 UDP packets from 10.0.0.0/16 ports 0-65535 to 10.0.0.0/24
    port 53 at a rate of 100KB/s with a summary after every 5 packets:
    ./scooter -p udp -s 10.0.0.0/16:-1 -d 10.0.0.0/24:53 -c 10 -b 100KB -m 5

    To generate continuous TCP SYN-FIN packets from 10.0.0.0/16 ports 0-65535
    to 10.1.1.1/32 port 53 at a rate of 20Kbit/s while ignoring checksums:
    ./scooter -p tcp -f sf -s 10.0.0.0/16:-1 -d 10.1.1.1/32:53 -b 20K -i
NOTES
    In order to perform rate limiting on FreeBSD this program makes
    use of the dummynet interface built into the kernel. Rate limiting
    will not work if you have not compiled your kernel with this support.

    Keep in mind that rate-limiting bandwidth numbers will only take into
    account data at the IP layer and above. This does NOT include the
    Ethernet Headers. For a 64 byte empty UDP packet per second, the
    bandwidth would equal only 28 bytes, or 28B.

    Currently ports are not taken into account when rate-limiting.

    Initial pp/s output may be misleading with smaller METERS until output
    buffer space is filled up. Default output queue is 50.
FUTURE
    Some nice additions to this program might include the following:
    - Multi-threading
    - Support for other ICMP Types
    - Random TCP Flags
SEE
    dummynet(4), ipfw(8)

```

Figure 6 - Scooter syntax and output

For the primary victim platform the author chose one of the leading firewall products on the market, the Netscreen 500. This firewall is theoretically capable of handling approximately 750 Mbps traffic loads in a vanilla environment. The test network consisted of a 200 rule policy with an any any permit rule at the bottom while the firewall was set to 'route' mode. All traffic was generated from one server on the untrusted side of the firewall while ping requests were generated from a second server to the same victim behind the firewall. The loss of ping replies and the output of the "get session" were used as metrics for session table flooding. The output of the Unix tcpdump program was used on both the source and destination servers to verify traffic forwarded by the firewall.

The Matrix

The very same session table flooding was also confirmed on a Cisco PIX 515 v6.1.1 and Nokia IP 440 running Check Point NG FP2. The PIX and Nokia served as an additional proof of concept test bed and allowed the author to confirm initial session timeouts and garbage collection intervals. Though the session flooding issue plagues all three firewall vendors evaluated, Check Point NG FP2 offers the largest feature set of protective mechanisms listed in Figure 7.

FEATURES	Netscreen 3.1.0r5	Cisco PIX 6.1.1	Check Point 4.1	Check Point NG FP2
Checksum Validation				
INIT Flow Timer Customization	•		○	•
Session Table Expansion			○	•
Session Table Limiting			○	○
Rate Limiting	•		•	•
Aggressive Aging				
SYN Required	•	•	•	•
TCP Proxy	•	•	•	•
TCP SYN Cookies				
TCP Fastpath			•	•
UDP / ICMP Flood Protection	•			
END Session Rapid Removal	○	?		

Figure 7 - Firewall Vendor Features Matrix

Netscreen's ScreenOS is not far behind with commitment by the vendor to implement further enhancements for state protection in future releases of their code. Finally, the Cisco PIX seems to provide the smallest selection of features for countering session table flooding. It should be noted that some features listed above may be much more effective than others for countering session table attacks and should not be weighted equally. As an example, TCP SYN Cookies, when available, may provide a much greater amount of protection for TCP traffic than several other features combined.

Uncovered Bugs

The proof of concept code not only confirmed the dangers of session table flooding, but also led to the discovery of several important bugs in current versions of Netscreen's ScreenOS. Following is a quick synopsis of the problems uncovered during testing, most of which have been mentioned in this paper:

- TCP Proxy is broken in ScreenOS 3.1.0r5 on NS500 and potentially several other versions and platforms. Packets with the SYN+ACK bits set can be used to fill the session table despite the 'tcp-syn-required' and TCP-Proxy settings.
- Entries exceeding the TCP Proxy queue are tracked in the state table in ScreenOS 3.1.0r5 on NS500 and potentially several other versions and platforms.

- UDP Flood is broken in ScreenOS 3.1.0r5 on NS500 and potentially several other versions and platforms. Packets must currently match a 3-tuple (source port + destination IP + destination port) to kick off the alarm, not the expected tuple (destination IP + destination port).
- Initial UDP service timeout setting is broken in ScreenOS 3.0.1r2 on NS5XP ScreenOS 2.6.1r7.1 on NS5 and potentially several other versions and platforms. UDP packets do not take on the same init flow timeout value as TCP.
- Service Timeout granularity on CLI is incorrectly documented in ScreenOS 3.1.0r5 on NS500 and potentially other versions down to 2.6 and other platforms. The CLI states that timeouts are to be specified in minutes when it is in fact expecting an entry in 10's of seconds.
- Check Point's SYNRelay and Netscreen's TCP Proxy do not perform Checksum Validation for TCP SYN packets.

Conclusion

The dangers of firewall session table flooding are quite extraordinary to which this paper has presented several potential countermeasures. It behooves the firewall vendor to implement the techniques on countering session based attacks and the security administrator to make full use of them.

References

- [1] Security Focus. "FireWall-1, FloodGate-1, VPN-1 Table Saturation Denial of Service Vulnerability", July 1999.
<http://online.securityfocus.com/bid/549>
- [2] Gill, Stephen. "Nokia HA Synchronization and ACK Protection", 04/18/2001. <http://www.cymru.com/gillsr/documents/nokia-high-availability.pdf>
- [3] Gill, Stephen. "Scooter Packet Generator", scooter.c, April 2002.
<http://www.cymru.com/gillsr/code/scooter-1.1.tar.gz>
- [4] Check Point. "How to improve diminished performance caused by exhaustion of the connections table", September 1999.
<https://support.checkpoint.com/public/idsearch.jsp?id=3.0.698764.2304823&QueryText=%28%22concurrent+connections%22%29&>
- [5] Tech Mavens. "Countering SYN Flood Denial-of-Service Attacks", August 2001.
<http://www.tech-mavens.com/synflood.htm>
- [6] Phoneboy. "Stateful PING and Traceroute with FW-1", November 2001.

<http://www.phoneboy.com/faq/0066.html>

[7] Sourceforge. "Libnet Libraries."
<http://sourceforge.net/projects/libnet/>

[8] Fyodor's Nmap port scanner. <http://www.insecure.org/nmap/index.html>

[9] Handley, M., Paxson V., and C. Kreibich, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics", May 2001. <http://www.icir.org/vern/papers/norm-usenix-sec-01-html/norm.html>

[10] Spitzner, Lance, "Understanding the FW-1 State Table" November 2000. <http://www.enteract.com/~lspitz/fwtable.html>

[11] Bernstein, DJ, "TCP SYN Cookies." <http://cr.yp.to/syncookies.html>