نگاهی اجمالی به هوکهای ویندوز

محمد شمس

چکیده

استفاده از هوک، یکی از مباحث تکنیکی مطرح در زمینه برنامه‌نویسی سیستمی بوده که کاربرد بسیاری در ساخت برنامه‌های کنترل کننده سیستم عامل دارد. هوک به معنی ایجاد شنود بر روند اجرای وقایعی مانند فراخوانی توابع، ارسال پیامها و پاسخ به رخدادها در سیستم عامل می‌باشد. از این تکنیک در ساخت نرم‌افزارهایی مانند آنتی‌ویروسها، فایروالها، دیباگرها، نرم‌افزارهای جاسوسی و ویروسها استفاده می‌شود.

همچنین در این مقاله روشهایی جهت شنود بر توابع API معرفی و تشریح شده و در پایان، ساختار فایلهای اجرایی ویندوز و نحوه اجرای آنها در سیستم عامل مورد بررسی قرار می‌گیرد.

کلمات کلیدی

هوک; شنود توابع API ; سیستم پیام‌رسان; تزریق کد; ساختار فایل PE

Short view of windows hooks

Mohammad Shams

Abstract

Using hooks is one of the system programmings technical articles that widely used in operating system monitoring programs. Hook means installing a routine to monitor windows message handling mechanism and event processing. This technic is used in many applications like antiviruses, firewalls, debugers, spy programs and viruses.

Also some API monitoring technics are explained in this article, finally portable executable file structure explained too with the way of its loading.

Keywords

Hook; API monitoring; Message handling system; Code injection; PE structure

مقدمه

یکی از مباحث مهمی که در زمینه برنامه‌نویسی سیستمی مطرح می‌شود، کنترل دقیق نحوه انجام کارها توسط سیستم عامل است. بدین منظور، برنامه‌نویسان اقدام به نصب و راه‌اندازی شنودگرهایی نرم‌افزاری جهت کنترل نحوه پاسخدهی سیستم عامل به وقایع سیستم می‌کنند. بدیهی است که در حین کنترل این روند، می‌توان تغییراتی دلخواه، بسته به نیاز و هدف برنامه‌نویس، در نحوه انجام آن اعمال کرد.

۱- هوک چیست

همانطور که می‌دانیم سازوکار سیستم عامل ویندوز، یک سیستم مبتنی بر پیام است. یعنی تمام اعمالی که در ویندوز انجام می‌شود از طریق ارسال پیامهایی از بخشهای مختلف این سیستم عامل به سایر بخشها صورت می‌گیرد. به عنوان مثال وقتی که پنجره‌ای را در ویندوز تغییر مکان می‌دهیم، پیامی توسط بخش کنترل کننده اینترفیس برنامه، حاوی اطلاعات موقعیت پنجره و از طریق تابع API به نام ()SendMessage به بخشی از هسته سیستم عامل ارسال گشته و سیستم عامل بر اساس پارامترهای دریافتی، کار مورد نظر را انجام می‌دهد. با توجه به مکانیسم ذکر شده، به عمل نصب شنودگرها در سیستم پیام‌رسان سیستم عامل، هوک گفته می‌شود [1].

احتمالا در ابتدا هوکها جهت استفاده در دیباگرها و اعمال کنترلی سیستم معرفی و بوجود آمده بودند اما امروزه، برنامه‌نویسان کاربردهای فراوانی برای آنها پیدا کردند. مثلاً برنامه‌ای برای ثبت کلیدهای فشرده شده در سیستم.

هوکها بر روی انواع خاصی از پیامهای ارسالی نصب شده و شنود بر تمامی انواع پیامهای سیستم عامل به سادگی امکانپذیر نیست. ایده اصلی هوک، در واقع نوشتن تابعی است که به محض فعال شدن رویدادی خاص (Event) در سیستم عامل، اجرا می‌شود. حال این تابع می‌تواند فقط شاهد داده‌های ارسالی بوده و یا آنها را تغییر هم بدهد.

۱-۱- فواید استفاده از هوک

- **تغییر مکانیسم یا روند اجرای برنامه‌ها**: با استفاده از هوک می‌توان اعمال دلخواهی، قبل و یا بعد از فعال شدن رویدادهای برنامه انجام داد. بدین وسیله می‌توان امکاناتی به برنامه‌های آماده و کامپایل شده افزوده و یا روند پیش‌فرض آنها را تغییر داد. مثلاً جلوگیری از عملی که یک قفل نرم‌افزاری برای بررسی وجود CD در درایو انجام می‌دهد.

- **دیباگ و مهندسی معکوس**: معمول‌ترین روش دیباگ کردن برنامه‌های کامپایل شده استفاده از هوکهای سیستم عامل است. با این کار می‌توان رابطه مابین بخشهای مختلف یک نرم‌افزار و نحوه تعامل آنها با سیستم عامل را بررسی کرده و به ایرادات احتمالی آن پی برد. به این عمل اصطلاحاً مهندسی معکوس گفته می‌شود.

- **کشف قابلیتهای مخفی سیستم عامل**: تعداد زیادی از توابع API موجود در ویندوز، مخفی بوده و هیچگونه مستنداتی از طرف شرکت مایکروسافت جهت معرفی آنها ارائه نشده است که اصطلاحا به آنها Undocumented API گفته می‌شود.
در حال حاضر کتب بسیاری به این مقوله پرداخته که حاصل مدتها تحقیق و بررسی و استفاده از روشهای مهندسی معکوس بر ساختار سیستم عامل ویندوز می‌باشند. در این کتب، تعداد زیادی از توابع مستند نشده ویندوز معرفی شده و نحوه کار آنها تشریح شده است.

یکی از نکاتی که قبل از استفاده از هوک می‌بایست به آن توجه شود اینست که هدف، شنود بر یک برنامه خاص بوده یا به منظور بررسی عملی، در کل سیستم عامل است.

مثلا اگر بخواهیم پارامترهای ارسالی در تابع ()SendMessage به پنجره‌های ویندوز را بدست آوریم، می‌بایست یک هوک سراسری (Global hook) نصب شود، اما اگر هدف فقط بررسی یک برنامه تنها باشد نیازی به اینکار نبوده و از یک هوک محلی (Local hook) استفاده می‌شود. به عبارتی یک هوک سراسری، رویدادهای تمام پروسه‌های موجود را شنود می‌کند.

۲- ساختار کلی هوک

از دو بخش سرویس دهنده (Hook server) و درایور تشکیل می‌شود. درایور کار اصلی را انجام داده و پیامهای مورد نظر را از سیستم دریافت می‌کند. سرور نیز در تعامل با درایور بوده و بنا به خواست برنامه‌نویس، برای نحوه برخورد با پیامها تصمیم می‌گیرد.

در یک سیستم هوک سراسری، درایور در یک فایل DLL و کاملاً مجزا از قسمت سرور قرار می‌گیرد. به عبارتی برنامه سرور اقدام به راه‌اندازی درایور از درون فایل DLL می‌نماید.

به طور دقیق‌تر سرور می‌بایست با استفاده از تابع ()LoadLibrary و به طور پویا، اقدام به بارگزاری فایل DLL نموده و Handle آن را بدست آورد. سپس با استفاده از تابع ()GetProcAddress اشاره‌گر مربوط به تابع نصب هوک را بدست آورده و از آن استفاده کند. با اینکار درایور با استفاده از تابع ()SetWindowsHookEx تابع هوک را به عنوان اولین حلقه از زنجیره هوکهای سیستم نصب می‌کند.

تابع ()SetWindowsHookEx اشاره‌گر مربوط به آدرس هوک نصب شده را به عنوان مقدار بازگشتی برمی‌گرداند. این مقدار اهمیت زیادی دارد زیرا می‌بایست با استفاده از تابع ()CallNextHookEx این آدرس را به ادامه زنجیره هوکها متصل کنیم. همچنین پس از پایان کار برای حذف هوک از این آدرس استفاده می‌شود [2].

در مورد هر نوع از پیامهای ارسالی در سیستم عامل، ممکن است تعداد زیادی هوک به شکل یک زنجیره به هم متصل نصب شده باشند. پس از فعال شدن رویداد مربوطه و ارسال پیام مورد نظر، سیستم عامل اولین حلقه از این زنجیر را فراخوانی کرده و پس از آن هر کدام از هوکها باید پس از انجام کارشان، حلقه بعدی را توسط تابع CallNextHookEx() فراخوانی کنند. بدیهی است که اگر این عمل به درستی انجام نشود، زنجیره مذکور قطع شده و هوکهای بعدی هیچ پیامی دریافت نمی‌کنند. به عنوان مثال ممکن است که دو برنامه به طور همزمان اقدام به شنود بر داده‌های ارسالی توسط دستگاه ماوس کنند. در این میان می‌بایست هر دوی این برنامه‌ها، داده‌های مورد نیاز خود را بدرستی دریافت نمایند.
شایان ذکر است که استفاده گسترده از هوکها شدیداً، سرعت پردازش سیستم عامل را کند می‌کند. زیرا با نصب زنجیره‌ای از هوکها میزان پردازشی که سیستم در ازای ارسال هر پیام انجام می‌دهد افزایش می‌یابد و در این حین توابع مربوط به تمام هوکها می‌بایست تک‌تک پردازش شده تا در نهایت پیام مربوطه به مقصد نهایی برسد. در نتیجه تنها زمانی از هوکها استفاده می‌شود که راه دیگری وجود نداشته باشد. همچنین اگر کوچکترین خطایی در حین کار برنامه سرویس دهنده یا درایور آن رخ دهد، ممکن است روند اجرای سیستم عامل مختل گردد. پس از نصب هوک سراسری، ویندوز فایل DLL مربوطه را در فضای آدرس (Address space) مربوط به تمام پروسه‌ها نوشته و آن را به عنوان قسمتی از روند ارسال پیام، مسیردهی می‌کند. (شکل ۱)

پس از پایان عمل شنود و یا زمانی که نیاز به خروج از برنامه سرویس دهنده باشد، هوک نصب شده باید از زنجیره هوکها خارج شود. برای اینکار از تابع UnHookWindowsHookEx() استفاده می‌شود. این تابع اشاره‌گر دریافت شده از SetWindowsHookEx() را به عنوان پارامتر ورودی دریافت کرده و اقدام به حذف هوک می‌کند.
هنگامی که این تابع فراخوانی می‌شود، سیستم عامل صبر می‌کند تا تمام پروسه‌هایی که در حال استفاده از تابع هوک هستند پردازش خود را تمام کرده و سپس اقدام به حذف هوک از فضای آدرس پروسه‌ها می‌نماید. ساختار توابع ذکر شده در جداول ۱ تا ۶ تشریح شده است.

۲-۱- پیامهای مورد استفاده در هوک

WH_CALLWNDPROC: شنود بر پیامهایی که با استفاده از تابع SendMessage() به پنجره‌ها ارسال می‌شوند. نوع تابع آن CallWndProc بوده و درست قبل از ارسال پیام ذکر شده اجرا می‌گردد.

WH_CALLWNDPROCRET: پس از پردازش پیامهای مذکور در برنامه پنجره مقصد، توسط این هوک می‌توان آنها را مشاهده کرد. نوع تابع آن CallWndProcRet است.

WH_CBT: نوع تابع خروجی CBTProc بوده و قبل از اعمالی مثل فعال شدن پنجره‌ها، ساخت، تخریب، تغییر مکان یا اندازه پنجره‌ها و یا تمرکز روی کنترلها (Focus) و به عبارتی قبل از پردازش هر عملی که توسط ماوس یا کیبورد بر روی پنجره‌ها انجام می‌شود، اجرا می‌گردد.

WH_DEBUG: نوع تابع آن DebugProc است و دقیقاً قبل از فراخوانی توابع هوک نصب شده در سیستم اجرا شده و اطلاعاتی در مورد هوک فعال شده دریافت می‌کند. به عبارتی به منظور دیباگ کردن روند اجرای دیگر هوکها از آن استفاده می‌شود.

WH_GETMESSAGE: نوع تابع آن GetMsgProc بوده و دقیقا زمانی اجرا می‌شود که برنامه‌ای با استفاده از توابع GetMessage() و یا PeekMessage() اقدام به خواندن یک پیام از صف پیامهای دریافتی خود می‌کند. قبل از پردازش پیام دریافت شده توسط برنامه، این هوک فعال میگردد.

WH_JOURNALPLAYBACK: از این هوک برای اجرای دنباله‌ای از رویدادهای ماوس و کیبورد که قبلاً توسط هوک دیگری با نام WH_JOURNALRECORD از سیستم ضبط شده‌اند، استفاده می‌گردد. پس از نصب این هوک، ورودیهای استاندارد ماوس و کیبورد غیر فعال شده و دنباله مذکور اجرا می‌شود. همچنین نوع تابع آن JournalPlaybackProc است.

WH_JOURNALRECORD: تابع آن JournalRecordProc بوده و برای ضبط دنباله ذکر شده مورد استفاده قرار می‌گیرد. کاربرد این دو هوک در برنامه‌های Macro Recorder بوده که در ابتدا اعمال انجام شده توسط ماوس و کیبورد را ضبط نموده و سپس در زمان مناسب دوباره اجرا می‌نمایند.

WH_KEYBOARD: نوع تابع آن KeyboardProc بوده و تمام رویدادهای صفحه کلید را شنود می‌کند. به عبارتی هر وقت که برنامه‌ای با استفاده از GetMessage() یا PeekMessage() اقدام به پردازش پیامی از نوع WM_KEYUP یا WM_KEYDOWN کند، هوک مذکور آن پیام را دریافت خواهد کرد.

WH_MOUSE: نوع تابع آن MouseProc بوده و تمام رویدادهای دستگاه ماوس را شنود می‌کند.

WH_MSGFILTER و WH_SYSMSGFILTER: نوع تابع آنها MessageProc بوده و تمام پیامهایی که توسط DialogBox، MessageBox، Scrollbar و منوها ارسال می‌گردند را دریافت می‌کنند.

WH_SHELL: پیامهای مربوط به برنامه‌های کنسولی را دریافت کرده و نوع تابع آن ShellProc است.

استفاده از تابع SetWindowsHookEx() به همراه پیامهای ذکر شده، روش استاندارد سیستم عامل ویندوز برای استفاده از هوکها و شنود می‌باشد. حال آنکه ممکن است بخواهیم بر روی پیامهای دیگر و یا بر روی توابع موجود در فایلهای DLL شنود نماییم. این مبحث در بخش بعدی تشریح خواهد شد.

۳- شنود بر توابع API

تا کنون روشهایی متفاوتی برای شنود بر توابع API توسط برنامه‌نویسان مختلف ارائه شده است که برخی از آنها در این بخش تشریح می‌شوند [3]. اما قبل از ادامه این بحث بهتر است به توضیح مختصری در مورد مهمترین فایلهای کتابخانه‌ای ویندوز، توجه نمایید:

- **Kernel32.dll**: حاوی توابع غیر GUI مانند توابع مدیریت I/O و فایلها، مدیریت حافظه، مدیریت اشیاء، پروسه‌ها و نخها (Threads) می‌باشد. همچنین رابطی بین توابع سطح پایین موجود در NTDll.dll و سرویسهای سیستم عامل است.

- **GDI32.dll**: حاوی تمام توابع مربوط به اعمال گرافیکی مثل ترسیم خطوط، قلمها، فایلهای Bitmap و کار با رنگها می‌باشد. البته تمام توابع بصری سطح پایین سیستم در فایل Win32k.sys قرار داشته و توابع موجود در GDI در حین اجرا، آنها را فراخوانی می‌کنند.

- **User32.dll**: تمام توابع سطح بالای مرتبط با GUI مانند توابع کار با فرمها، پنجره‌ها، منوها و کنترلها در این کتابخانه قرار دارند. البته این کتابخانه برای انجام اعمال بصری خود از توابع GDI استفاده می‌کند.

۳-۱- استفاده از رجیستری

یکی از روشها تزریق کد به برنامه‌ها، استفاده از کلید زیر در رجیستری است:

```
HKLM\Software\Microsoft\WindowsNT\CurrentVersion
\Windows\AppInit_DLLs
```

تمام DLL هایی که نام آنها در این کلید ذکر شده باشد به عنوان بخشی از کتابخانه User32.dll محسوب شده و با استفاده از تابع LoadLibrary() به درون حافظه تمام پروسه‌ها بارگزاری می‌گردند. البته این روش هم مشکلاتی دارد از جمله اینکه فقط در مورد برنامه‌هایی می‌تواند به کار برود که از کتابخانه ذکر شده استفاده می‌کنند و به عبارتی برنامه‌های کنسولی را شامل نمی‌شود، زیرا این برنامه‌ها معمولا هیچ یک از توابع User32 را Import نمی‌کنند.
مشکل دیگر این است که برای غیر فعال کردن این روش می‌بایست کلید مربوطه در رجیستری پاک شده و سیستم Restart گردد. همچنین اضافه شدن DLL مربوطه به حافظه تمام پروسه‌ها، سرباری بوجود می‌آورد که باعث اشغال حافظه سیستم و کند شدن پردازش می‌شود.

۳-۲- استفاده از DLL واسطه

مثلاً فرض کنید که می‌خواهیم بر فراخوانیها و پارامترهایی شنود کنیم که توسط برنامه IE به فایل کتابخانه‌ای Wsock32.dll ارسال می‌گردند. این DLL حاوی توابع ویندوز جهت کار با WindowsSocket و ارسال و دریافت داده‌ها در شبکه است. به عبارتی تمام برنامه‌هایی که نیاز به ارتباط با شبکه از طریق WindowsSocket داشته باشند، از توابع موجود در این فایل استفاده می‌کنند. حال می‌بایست فایلی DLL مشابه این فایل درست کرده و تمامی توابع مورد نظر را در آن قرار داده و فایل مذکور را در فهرست حاوی برنامه مورد نظر قرار دهیم.
از این پس هر وقت برنامه مورد نظر اجرا شود، در صورت نیاز به فراخوانی توابع موجود در DLL، اولویت اول با DLL ای است که در دایرکتوری خود برنامه قرار دارد. در صورتی که این فایل موجود نباشد مسیرهای System، System32 و غیره جستجو خواهند شد. بدین شکل تمام فراخوانیهای برنامه به این DLL رسیده و می‌توان اقدام به شنود یا تغییرات دلخواه در پاسخ به برنامه نمود.
البته نقطه ضعف بسیار بزرگ این روش این است که می‌بایست تمام توابع موجود در DLL اصلی، عیناً در DLL واسطه نیز وجود داشته باشند، تا امکان استفاده از آنها توسط برنامه وجود داشته باشد. در نتیجه ممکن است برای مانیتور کردن یک تابع مجبور شویم بیش از صدها تابع دیگر را نیز تعریف نماییم.
برای رفع این مشکل می‌توان با بررسی دقیق برنامه مربوطه، تمام فراخوانیهای مربوط به DLL مربوطه را پیدا کرده و فقط توابع Import شده را بازسازی کنیم که خود این مسئله هم ممکن است بسیار وقتگیر و مشکل باشد.

۳-۳- تغییر در خود تابع API

برای اینکار روشهای مختلفی وجود دارد که یکی از آنها دیباگ DLL مربوطه در حافظه و تنظیم بایت اولیه API با فرمان Breakpoint (وقفه Int3) می‌باشد. بدین شکل هر فراخوانی از تابع مربوطه، منجر به تولید یک استثناء (Exception) شده که به برنامه دیباگر ما ارسال می‌گردد.
مشکلات این روش، کند شدن سیستم عامل به خاطر درگیر شدن با سیستم مدیریت استثناء در ویندوز بوده و دیگری مشکلی است که

زمان خاتمه پروسه دیباگر پیش می‌آید. همانطور که می‌دانید پس از پایان پروسه یک دیباگر، تمام برنامه‌های تحت کنترل آن دیباگر نیز بسته خواهند شد.

راهکار دیگر تغییر کنترل اجرای تابع با استفاده از فرمانهای پرش JMP یا CALL به قسمت شنودگر است. بدین شکل که می‌توان تابع مربوطه را Dissassemble کرده و در ابتدای آن، فرمان پرش را قرار دهیم. مشکلی که در این زمینه ممکن است پیش بیاید کمبود فضا برای درج فرمان پرش است. زیرا ممکن است روتین API مربوطه حجم بسیار کمی مثلاً در حدود ۴ بایت داشته باشد، حال آنکه خود فرامین پرش، پس از کامپایل به ۵ بایت تبدیل می‌شوند.

مشکل دیگر دشواری حذف روتین شنود است. زیرا برای اینکار می‌بایست کد برنامه در حافظه و به طور پویا رونویسی شده و فرمان پرش از بین برود.

۳-۴- تغییر فراخوانیها

برای شنود بر فراخوانیهای یک برنامه، نقطه دیگری که می‌توان بر آن متمرکز شد خود فراخوانیها در فایل برنامه مذکور هستند. اینکار را می‌توان با تغییر در فایل اجرایی برنامه و یا پس از اجرای برنامه و با رونویسی مکانهای اجرای فراخوانی در حافظه انجام داد. قسمت مشکل اینکار پیدا کردن یک به یک این نقاط در کد برنامه می‌باشد که کار دشواری است.

۳-۵- تغییر در IAT (Import Table Address)

این روش بر این اصل بنا نهاده شده که فایلها اجرایی ویندوز و DLLها همگی دارای فرمت PE هستند. در این فرمت، فایلهای اجرایی از بخشهای منطقی مجزایی به نام سکشن تشکیل شده‌اند. هر کدام از سکشنها حاوی نوع خاصی از داده‌ها می‌باشند. مثلاً سکشن .text حاوی کد اجرایی کامپایل شده برنامه بوده و یا سکشن .rsrc حاوی ریسورسهای مورد استفاده برنامه مثل تصاویر، اصوات یا فرمها است.

در میان تمام آنها، سکشن .idata حاوی جدول آدرسهای وارد شده (IAT) می‌باشد. این جدول حاوی آدرس نسبی تمام توابع Import شده به فایل اجرایی است.

پس از اجرای یک برنامه، سیستم عامل آن را درون حافظه بارگزاری کرده و این آدرسها را به آدرس صحیح توابع مذکور در حافظه تغییر می‌دهد. دلیل وجود این جدول این است که فایلهای اجرایی همیشه در مکان ثابتی از حافظه بارگزاری نمی‌شوند.

در نتیجه پس از هر بار اجرای برنامه، می‌بایست در تمام فراخوانیهای موجود، آدرس نسبی تصحیح گشته که این عمل بسیار مشکل است. این مشکل با مجتمع کردن تمام آدرسهای وارد شده در مکانی از فایل به نام IAT حل می‌شود.

بدین منظور کافی است که هر یک از فراخوانیها را به طور نسبی (محلی) نسبت به IAT آدرس‌دهی کنیم و آدرسهای اصلی را در خود IAT نگهداریم. بدین شکل سیستم عامل به سادگی می‌تواند یک بار آدرس مربوطه را در IAT تغییر دهد که به ازای آن تمام فراخوانیها از طریق همین آدرس تصحیح شده انجام خواهند شد [4].

با توجه به مطالب ذکر شده، در این روش می‌توانیم کد اجرایی شنودگر را در فضای پروسه مربوطه در حافظه (Memory context) بارگزاری کرده و سپس آدرس تابع مورد نظر را در IAT، به آدرس شنودگر تغییر داده و در خود شنودگر تابع اصلی را فراخوانی کنیم.

بدین شکل قبل از اینکه تابع اصلی فراخوانی شود، محتوای پارامترهای ورودی آن به شنودگر ارسال می‌شوند که می‌توان تغییرات دلخواه را روی آنها اعمال نمود. تنها مشکلی که در این زمینه پیش می‌آید نحوه بارگزاری کد اجرایی شنودگر در فضای پروسه مربوطه در حافظه است.

۴- نگاهی کوتاه به ساختار فایل PE

شرکت مایکروسافت در زمان عرضه ویندوز NT، فرمت جدیدی را برای فایلهای اجرایی ویندوز با نام PE (Portable Executable) معرفی نمود. کلمه PE به معنی اجرایی قابل حمل می‌باشد که دلیل آن این است که در میان تمام سیستم عاملهای ۳۲ بیتی مایکروسافت قابل استفاده بوده و توسط آنها پشتیبانی می‌شود [5].

این فرمت بر اساس COFF (Common Object File Format) که فرمت فایل رایج و مورد استفاده در سیستم عامل یونیکس است طراحی شده بود. فرمت PE در سال ۱۹۹۳ توسط کمیته‌ای به نام TIS (Tool Interface Standard) متشکل از شرکتهای اینتل، مایکروسافت، بورلند و آی‌بی‌ام به استانداردی همگانی تبدیل شد.

فایل PE به بلاکهایی به نام سکشن تقسیم شده که حاوی داده‌هایی با خصلتهای مختلف مثل داده/کد اجرایی/خواندنی/نوشتنی هستند. این فایل را می‌توان مشابه یک دیسک منطقی فرض کرد که هدر (Header) فایل PE مثل بوت‌سکتور آن بوده و سکشنها همان فایلهای نوشته شده روی دیسک می‌باشند.

پس از هدر فایل، قسمتی به نام جدول سکشنها (Section table) موجود است که مانند جدول FAT در دیسک می‌باشد. این جدول آرایه‌ای از هدر تمام سکشنهای موجود در فایل است. به عبارتی به تعداد سکشنهای موجود در فایل، در این جدول هدر وجود دارد. در هر کدام از این هدرها اطلاعاتی مثل خصلت سکشن مربوطه، آفست و سایز آن در فایل و نحوه بارگزاری آن در حافظه موجود است.

در نهایت پس از این جدول به خود سکشنها می‌رسیم که حاوی تمام داده‌های موجود در فایل، با توجه به نوع سکشن می‌باشند.

پس به طور کلی یک فایل PE یک جدول بزرگ حاوی هدرها و سکشنها می‌باشد. خلاصه‌ای از مهمترین قسمتهای فرمت PE در جدول ۷ ذکر شده‌اند.

۴-۱- روند اجرای فایلهای PE

پس از اجرای یک فایل PE، سیستم بارگزار ویندوز فضایی را در حافظه به آن پروسه اختصاص داده و اقدام به بارگزاری فایل اجرایی در آن فضا می‌کند. بدین منظور جداول هدر را در آدرس پایه پیش‌فرض قرار داده و به همین ترتیب سکشنها نیز با محاسبه آدرس نسبیشان (RVA) نسبت به آدرس پایه، در حافظه قرار می‌گیرند.

در همین حین خصلتهای هر سکشن نیز با توجه به تعاریف موجود در هدر آن تنظیم می‌شوند. پس از اینکه تمام سکشنها در حافظه قرار گرفتند، جدول توابع ورودی (Import table) بررسی شده و تمام فایلهای کتابخانه‌ای مورد نیاز، در فضای رزرو شده توسط این پروسه بارگزاری می‌شوند.

پس پایان بارگزاری، جدول خروجی (Export table) تک‌تک DLLهای مذکور بررسی شده و آدرس توابع مورد نظر آنها در IAT پروسه ثبت می‌گردد. در نتیجه IAT حاوی اشاره‌گر تمام توابع مورد نیاز پروسه در حافظه می‌باشد. با استفاده از این مکانیسم دیگر نیازی به تغییر آدرس در خود فراخوانیهای تابع در کد برنامه نبوده و تمام آدرس‌دهی‌ها به طور نسبی و نسبت به IAT انجام می‌شوند.

۵- نتیجه

همانطور که ذکر شد، استفاده از هوکها، جزئی اجتناب ناپذیر از برنامه‌های سیستمی می‌باشد.

در این زمینه برنامه‌نویسان زیادی اقدام به انتشار کامپوننتها و کتابخانه‌هایی از توابع، جهت تسهیل روند هوک کردن توابع سیستمی کرده‌اند.

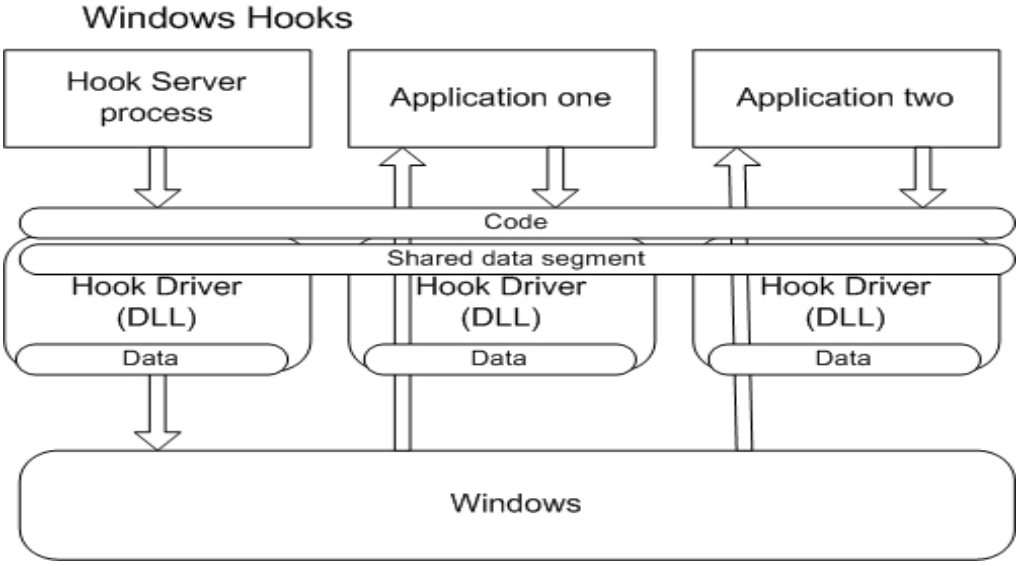
یکی از کتابخانه‌های معروف در این زمینه MadshiSDK برای پیاده‌سازی در زبان دلفی و دیگری مجموعه HookToolSDK از شرکت InfoProcess به منظور استفاده در برنامه‌های داتنت می‌باشد.

تمامی این کتابخانه‌ها به دفعات در ویروسهای کامپیوتری مورد سوء استفاده قرار گرفته و خسارات بسیاری به بار آورده‌اند. به عنوان مثال فرض کنید که ویروس مورد نظر با هوک کردن تابع CreateProcess() کنترل اجرای برنامه‌ها را بدست گرفته و یا با هوک تابعی مثل WriteProcessMemory() روند مدیریت حافظه را مختل کند.

در چنین مواردی، آنتی‌ویروسها نیز با استفاده از روشهای مشابهی اقدام به مقابله با آنها می‌نمایند.

نهایتاً کاربرد هوک بسیار وسیعتر از موارد ذکر شده بوده و در اکثر برنامه‌های مورد استفاده کاربران، مانند فایروالها، برنامه‌های گرافیکی، نرم‌افزارهای اداری، محیطهای برنامه‌سازی، دیباگرها و غیره مورد استفاده قرار می‌گیرد.

از این رو کسب اطلاعات و دانشی نسبی در این زمینه، امری ضروری برای تمام برنامه‌نویسان محسوب می‌شود.



شکل (۱): بارگزاری درایور در حافظه پروسه‌های دیگر

نام پارامتر	نوع	توضیحات
idHook	Integer	نوع پیامی که هوک می‌شود را مشخص می‌نماید. مثلا WH_KEYBOARD
lpfn	TFNHookProc	آدرس تابع هوک در حافظه
hMod	Hinst	هندل مربوط به DLLای که هوک درون آن قرار دارد. برای هوکهای محلی از صفر استفاده می‌شود
dwThreadID	Cardinal	thread id مربوط به پروسه‌ای که قرار است هوک شود. برای هوکهای سراسری از صفر استفاده می‌شود
مقدار بازگشتی	HHOOK	در صورت موفقیت هندل تابع هوک در زنجیره و در غیر اینصورت Null را بازمی‌گرداند

جدول (۱): مشخصات تابع SetWindowsHookEx()

نام پارامتر	نوع	توضیحات
hhk	HHOOK	هندل تابع هوک در زنجیره
مقدار بازگشتی	BOOL	در صورت موفقیت مقداری غیر صفر و در غیر اینصورت صفر را بازمی‌گرداند

جدول (۲): مشخصات تابع UnhookWindowsHookEx()

نام پارامتر	نوع	توضیحات
hhk	HHOOK	هندل تابع هوک در زنجیره
Code	Integer	طریقه برخورد تابع با دو پارامتر بعدی را مشخص می‌کند
wParam	Word	پارامتر ارسالی اول به تابع هوک با توجه به نوع آن
lParam	Longword	پارامتر ارسالی دوم به تابع هوک با توجه به نوع آن
مقدار بازگشتی	LRESULT	در صورت موفقیت مقداری غیر صفر است که تابع بعدی در زنجیره هوک بازگردانده است

جدول (۳): مشخصات تابع CallNextHookEx()

نام پارامتر	نوع	توضیحات
lpFileName	LPCTSTR	اشاره‌گر به رشته‌ای حاوی نام ماژول مورد نظر
مقدار بازگشتی	HMODULE	در صورت موفقیت هندل ماژول و در غیر اینصورت Null را بازمی‌گرداند

جدول (۴): مشخصات تابع LoadLibrary()

نام پارامتر	نوع	توضیحات
hModule	HMODULE	هندل ماژول مورد نظر
مقدار بازگشتی	BOOL	در صورت موفقیت مقداری غیر صفر و در غیر اینصورت صفر را بازمی‌گرداند

جدول (۵): مشخصات تابع FreeLibrary()

نام پارامتر	نوع	توضیحات
hModule	HMODULE	هندل ماژول مورد نظر
lpProcName	LPCTSTR	اشاره‌گر به رشته‌ای حاوی نام تابع مورد نظر
مقدار بازگشتی	BOOL	در صورت موفقیت آدرس تابع مورد نظر و در غیر اینصورت Null را بازمی‌گرداند

جدول (۶): مشخصات تابع GetProcAddress()

MS-DOS information	IMAGE_DOS_HEADER	DOS EXE Signature DOS_ChkSum PE Pointer ...
	MS-DOS Stub Program	This program cannot be run in DOS mode
Windows NT information IMAGE_NT_HEADERS	IMAGE_FILE_HEADER	PE signature (PE) NumberOfSections TimeDateStamp SizeOfOptionalHeader Characteristics ...
	IMAGE_OPTIONAL_HEADER	MagicNumber SizeOfCode AddressOfEntryPoint ImageBase SizeOfImage SizeOfHeaders CheckSum SizeOfStackReserve SizeOfHeapReserve IMAGE_DATA_DIRECTORY[] [Export Table] [Import Table] [Resource Table] ...
Section table	IMAGE_SECTION_HEADER[0] . . . IMAGE_SECTION_HEADER[N]	Name VirtualSize VirtualAddress SizeOfRawData PointerToRawData Characteristics ...
Section Images	SECTION[0] . . . SECTION[N]	Binary data of sections: ".text" ".data" ".idata" ".edata" ".reloc" ".rsrc"

جدول (۷): ساختار فایل PE

مراجع

[1] Microsoft© Developer Network - October 2003
[2] "An Introduction to Hook Procedures" by Zarko Gajic
[3] "API Spying Techniques for Windows" by Yariv Kaplan, May 2001
[4] "Peering Inside the PE: A Tour of the Win32 Portable Executable File Format" by Matt Pietrek, March 1994
[5] "An In-Depth Look into the Win32 PE file format" by Matt Pietrek, February 2002