قسمت ۱۰ – 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

فصل نهم : دیباگر OllyDBG

این فصل روی دیباگر OllyDBG تمرکز خواهیم داشت که توسط Oleh Yuschuk برای معماری x86 طراحی و توسعه داده شده است. این دیباگر توانایی تحلیل باینری بدافزار را در هنگام اجرا به متخصصین ارائه می‌کند. همچنین قابل ذکر است، دیباگر OllyDBG به دلیل رایگان بودن و راحتی کار در استفاده از آن و همچنین قابلیت دریافت پلاگین، توسط بسیاری از تحلیلگران بدافزار و کسانی که مهندسین معکوس انجام می‌دهند و همچنین کرکرها در سطح وسیعی مورد استفاده قرار می‌گیرد.

دیباگر OllyDBG بیش از یک دهه است که ایجاد شده است و تاریخچه جالبی دارد. اولین بار این دیباگر پیش از این‌که برای تحلیل بدافزار مورد استفاده قرار گیرد، برای کرک برنامه‌های کاربردی به خدمت گرفته شد. دیباگر OllyDBG اولین انتخاب تحلیلگران بدافزار و توسعه‌دهندگان اکسپلویت همواره بوده است، تا این‌که توسط شرکت Immunity خریداری و به Immunity Debugger تغییر نام داده شد.

هدف Immunity بردن OllyDBG سمت ابزاری برای توسعه‌دهندگان اکسپلویت و رفع مشکلات نرم‌افزاری موجود در نسخه 1.1 دیباگر OllyDBG بود. دیباگر ImmDbg با زیباسازی رابط‌کاربری OllyDBG و اضافه کردن یک مفسر پایتون که از رابط‌های برنامه‌نویسی ویندوز[1] پشتیبانی می‌کرد، موجب شد بعضی از کاربران بجای استفاده از OllyDBG از دیباگر بهبود یافته ImmDbg استفاده کنند.

اگر شما دیباگر ImmDbg را به OllyDBG ترجیح می‌دهید، نگران هیچ چیزی نباشید. زیرا دیباگر ImmDbg همان OllyDBG 1.1 است و هر چیزی را که در مورد OllyDBG یاد می‌گیرید، می‌توانید روی دیگری نیز اجرا کنید. تنها نکته‌ای که باید بدانید این است که بسیاری از پلاگین‌های OllyDBG به صورت خودکار به دیباگر ImmDbg اضافه نمی‌شوند. بنابراین تا زمانی‌که آن‌ها برای دیباگر ImmDbg بازنویسی نشده باشند، نمی‌توانید از آن پلاگین‌ها برای دیباگر ImmDbg استفاده کنید.

دیباگر ImmDbg مزایا خاص خود را دارد، مانند قابلیت توسعه ویژگی‌های آن با استفاده از رابط‌های برنامه‌نویسی پایتون[2] که در قسمت دیباگ با اسکریپت‌نویسی بیشتر در این مورد بحث خواهیم کرد. همچنین

[1] Windows API
[2] Python API

شایان ذکر است، در سال ۲۰۱۰ نسخه دیگری برای OllyDBG منتشر شد، که همگان آن را با نسخه دو OllyDBG می‌شناسند. در این نسخه دیباگر OllyDBG از ابتدا بازنویسی شد و بسیاری از ضعف‌های آن مرتفع گردید اما بسیاری از افراد متخصص در زمینه تحلیلگری این نسخه را یک نسخه Beta می‌دانند و تا زمان نوشتن این کتاب توسط آن‌ها مورد استفاده وسیعی قرار نگرفته است. قابل تذکر است، ورژن دو OllyDBG دارای ویژگی‌های خاصی می‌باشد که در نسخه 1.1 آن پیدا نمی‌شود که در این فصل و فصول دیگر به آن اشاره خواهیم کرد.

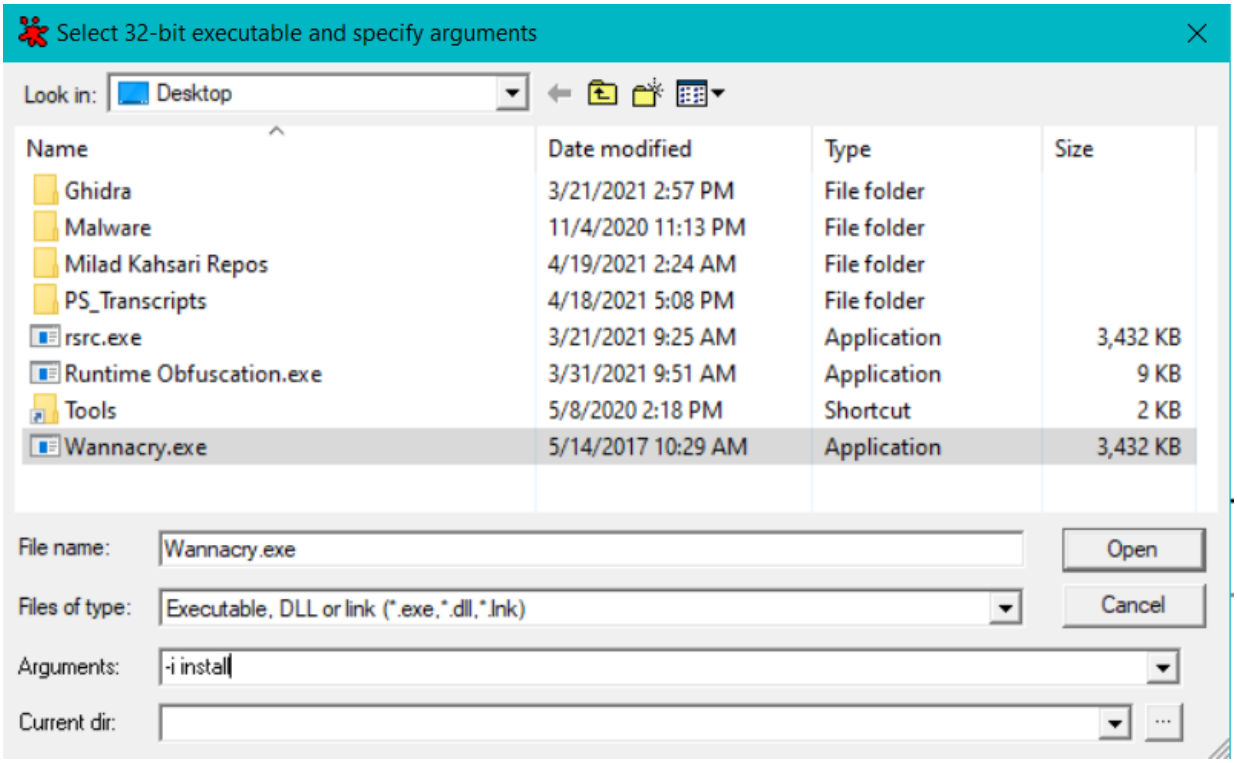
بارگذاری بدافزار

چندین روش برای آغاز دیباگ یک باینری با OllyDBG وجود دارد. شما می‌توانید فایل‌اجرایی و حتی کتابخانه‌های پیوندی پویا را به صورت مستقیم در OllyDBG بارگذاری کنید یا اگر بدافزار روی سامانه شما درحال اجرا باشد، می‌توانید دیباگر را به پروسه آن بدافزار پیوست[1] کرده و شروع به دیباگ پروسه اجرایی آن بدافزار کنید. دیباگر OllyDBG همچنین یک سامانه قابل انعطاف برای اجرای بدافزار از طریق گزینه‌های خط فرمان یا اجرای ویژگی خاصی در کتابخانه‌های پیوندی پویا (DLL) را به ما ارائه می‌دهد که در ادامه با آن‌ها آشنا خواهیم شد.

باز کردن یک فایل اجرایی

راحترین روش برای دیباگ بدافزارها، بارگذاری فایل‌اجرایی آن‌ها از طریق منوی Open و انتخاب گزینه File است. همچنین اگر بدافزار نیازمند پارامترهای مختلفی باشد، می‌توانید با تعیین آن پارامترها در بخش Arguments پنجره Open File دیباگر OllyDBG آن پارامترها را تنظیم کنید (همانند تصویر ۱).

[1] Attachment



تصویر ۱: باز کردن یک فایل‌اجرایی با گزینه‌های خط فرمان

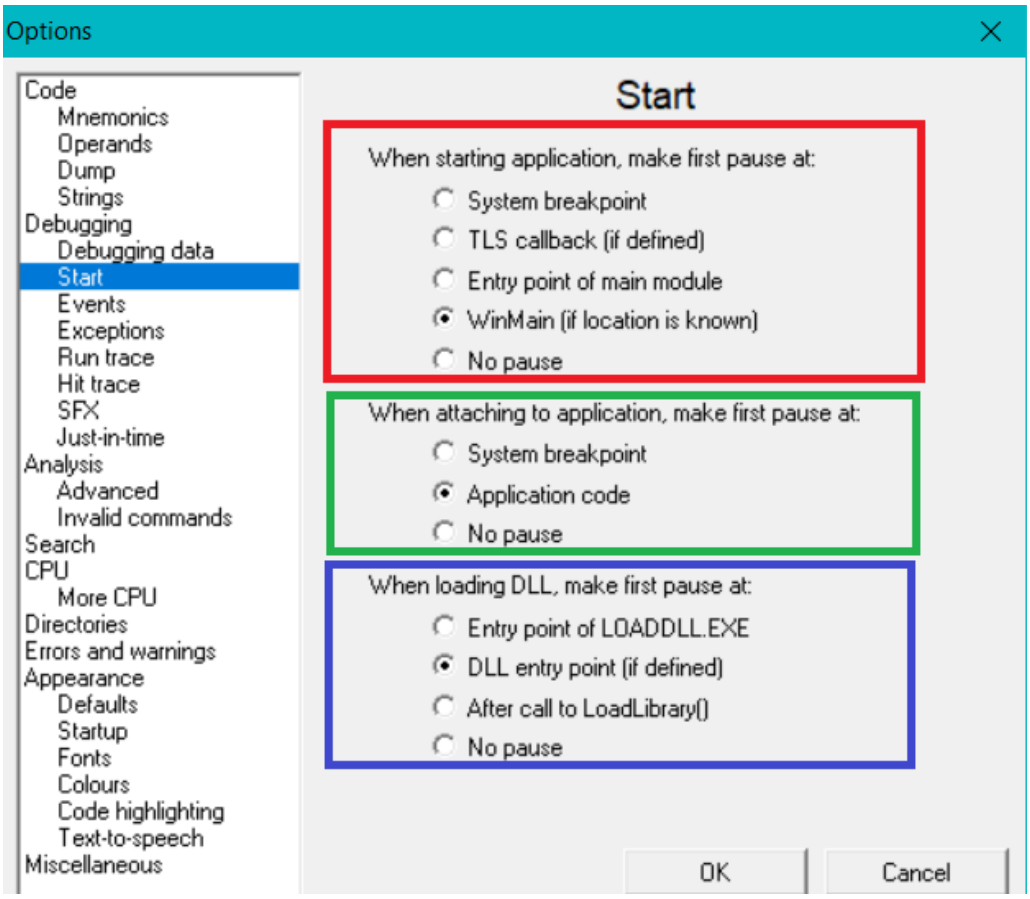
هنگامی‌که یک برنامه اجرایی را درون دیباگر بارگذاری می‌کنید، دیباگر OllyDBG کدهای باینری آن فایل اجرایی را با لودر خود، بارگذاری می‌کند. این عملیات مشابه روشی است که سامانه‌عامل برای بارگذاری یک فایل مورد استفاده قرار می‌دهد.

به صورت پیش‌فرض دیباگر OllyDBG یا حتی دیگر دیباگرها، موقتا اجرای یک برنامه را در نقطه ورودی[1] که به تابع WinMain (برای برنامه‌های گرافیکی نوشته شده با C/C++) یا تابع main (برای برنامه‌های تحت خط فرمان نوشته شده با C/C++) معروف است، متوقف می‌کند، البته در صورتی که بتواند این نقطه را شناسایی کند.

در غیر اینصورت دیباگر از نقطه آغازی که در هدر PE مشخص شده است (هدر EntryPoint)، برای متوقف‌سازی موقت اجرای برنامه استفاده می‌کند. همچنین می‌توانید با استفاده از منوی تنظیمات دیباگر (Options> Debugging Options) این نقطه شروع را تغییر دهید. در تصویر ۲ گزینه‌هایی که در

[1] EntryPoint

OllyDBG می‌توانید به عنوان نقطه توقف در هنگام آغاز دیباگ یک باینری استفاده کنید، در کادر قرمز رنگ نمایش داده شده است.



تصویر ۲: تنظیمات **OllyDBG** برای دیباگ باینری

به عنوان مثال، برای توقف برنامه پیش از اجرای هر نوع کدی توسط باینری که در حال دیباگ هستیم، گزینه System Breakpoint را باید انتخاب کنید یا اگر گزینه Entry point of main module را انتخاب کنیم، دیباگر قبل اینکه تابع main برنامه اجرا شود، اجرای باینری را در تابع mainCRTStartup (در برنامه‌های نوشته شده با زبان C/C++) متوقف خواهد کرد از همین روی ما شانس این را خواهیم داشت که قبل اجرای هر نوع کدی توسط باینری، تحلیل و دیباگ خود را ادامه بدهیم. اگر هم گزینه No pause انتخاب شود، باینری بدون اینکه متوقف شود در محیط دیباگر اجرا خواهد شد.

نکته: دیباگر OllyDBG ورژن ۲ قابلیت توقف‌سازی بیشتری نسبت به نسخه ۱.۱ دارا می‌باشد. به عنوان مثال OllyDBG می‌تواند طوری تنظیم شود که در نقطه

TLS Callback اجرای برنامه را موقتاً متوقف کند (همانطور که در تصویر ۲ مشاهده می‌کنید). ویژگی TLS Callback به بدافزار اجازه می‌دهد اجرا شود، پیش از آنکه OllyDBG بدافزار را به صورت موقت متوقف کند. در فصول آینده، در مورد این‌که چگونه می‌توان از TLS Callback برای عملیات‌های ضد دیباگ و همچنین محافظت از خود استفاده کنید، بحث خواهیم کرد.

پیوست به یک پروسه در حال اجرا[1]

علاوه بر باز کردن یک فایل‌اجرایی در دیباگر OllyDBG شما می‌توانید دیباگر OllyDBG را همچنین به یک پروسه در حال اجرا پیوست کنید. این ویژگی زمانی‌که یک بدافزار در حال اجرا را بخواهید دیباگ کنید، بسیار مفید می‌باشد. برای پیوست یک پروسه در حال اجرا به دیباگر OllyDBG به منوی File مراجعه کرده و گزینه Attach را انتخاب کنید.

با انتخاب این گزینه منوی دیگری باز شده و به شما اجازه می‌دهد از لیست پروسه‌های در حال اجرا روی سامانه، پروسه مورد نظر خود را انتخاب کرده و OllyDBG را به آن پیوست کنید (در صورتی که دو یا چند پروسه هم نام در لیست وجود داشته باشد، شما نیازمند داشتن شناسه آن پروسه خواهید بود). سپس پروسه مورد نظرتان را انتخاب کرده و روی دکمه Attach کلیک کنید. پس از کلیک روی دکمه Attach دیباگر OllyDBG برنامه و تمامی تردهای آن را به صورت موقت متوقف می‌کند.

زمانی‌که شما OllyDBG را به یک پروسه در حال اجرا پیوست می‌کنید، ترد جاری آن پروسه که در حال اجرای کد باینری ما است، متوقف خواهد شد و دستوراتی که ترد در حال اجرا است به ما نمایش داده خواهد شد. با این حال، ممکن است ما زمانی پروسه را به دیباگر پیوست کنیم که ترد اصلی آن در حال اجرای توابعی از کتابخانه های سیستمی ویندوز باشد.

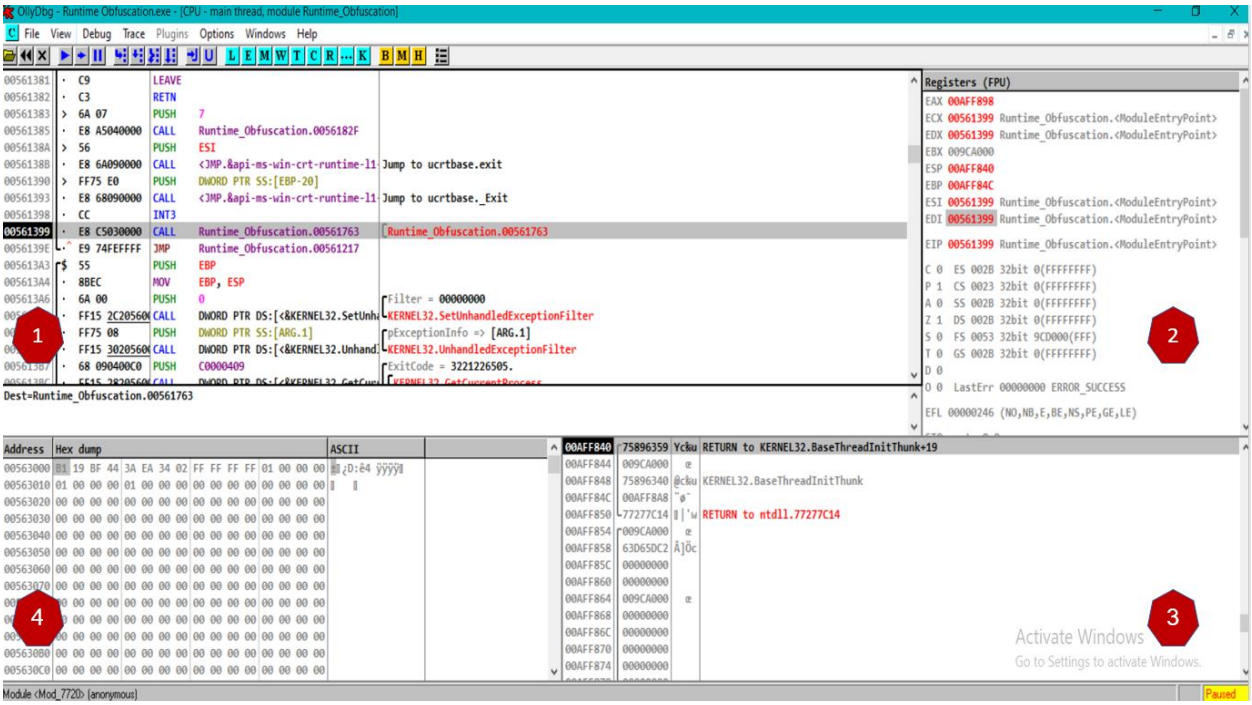
از همین روی، به منظور این‌که کتابخانه‌های ویندوز را دیباگ نکنید، راحت‌ترین راه برای گرفتن دسترسی به کد اصلی برنامه، ایجاد نقطه توقف زمان دسترسی[2] بر روی سکشن کد باینری بدافزار است. این کار باعث

[1] Attaching to a Running Process
[2] Breakpoint on access

می‌شود که برنامه اجرای خود را هنگامی‌که به سکشن کد خود دسترسی می‌گیرد، متوقف کند. تنظیم چنین نقاط توقفی را در این فصل توضیح خواهیم داد.

رابط کاربری دیباگر OllyDBG

هنگامی‌که برنامه‌ای را درون OllyDBG بارگذاری می‌کنید، چهار پنجره که با اطلاعات مفیدی به منظور تحلیل بدافزار کامل شده‌اند، رو به رو خواهید شد. (شکل ۳ را مشاهده کنید).



شکل ۳: رابط کاربری OllyDBG

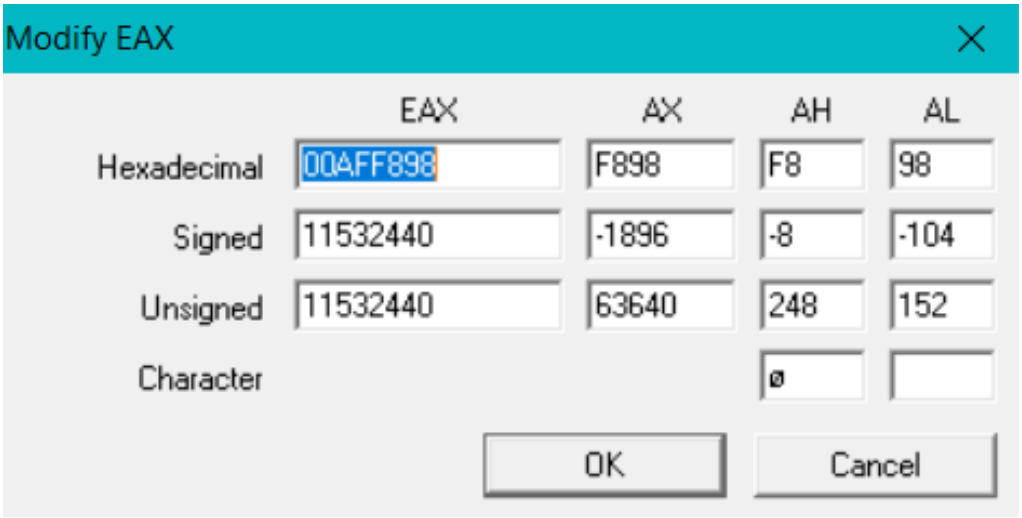
این پنجره‌ها اطلاعات زیر را به نمایش می‌گذارد:

- **پنجره دیزاسمبلر[1] (شماره ۱):** این پنجره کدهای دیزاسمبل شده برنامه را به همراه وضعیت فعلی اشاره‌گر دستورالعمل[2] و چندین دستورالعمل بعد و قبل از آن را نمایش می‌دهد. معمولا، دستورالعملی که قرار است اجرا شود در این پنجره با رنگ دیگری برجسته می‌شود. همچنین شما می‌توانید در این پنجره برای تغییر دستورالعمل‌ها یا داده‌ها (یا اضافه کردن یک دستورالعمل اسمبلی

[1] Disassembler window
[2] IP - Instruction Pointer

جدید)، کلید space را بفشارید و سپس در پنجره جدیدی که باز می‌شود فرمان خود را وارد و اعمال کنید.

- **پنجره ثبات‌ها[1] (شماره ۲):** این پنجره وضعیت فعلی ثبات‌های پردازنده را برای برنامه دیباگ شده نمایش می‌دهد. زمانی‌که دستورالعملی از برنامه اجرا شود و موجب شود ثبات‌ها تغییر یابند، رنگ آن ثبات‌ها از مشکی به قرمز در پنجره ثبات‌ها تغییر می‌کند. همچنین مانند پنجره دیزاسمبلر شما می‌توانید اطلاعات موجود در پنجره ثبات‌ها را زمانی‌که برنامه دیباگ می‌شود با کلیک راست روی ثبات‌ها و انتخاب گزینه Modify مقدار آن‌ها را تغییر بدهید. در تصویر ۴ پنجره تغییر دهنده مقدار ثبات‌ها نمایش داده شده است. شما می‌توانید با این پنجره مقادیر ثبات‌ها را عوض کنید. به عنوان مثال ما در این قسمت می‌خواهیم مقدار جاری ثبات EAX را تغییر بدهیم.



تصویر ۴: تغییر دادن یک ثبات

نکته: یکی از مواردی که شخص تحلیلگر باید روی آن مسلط باشد، مباحث مبتنی بر زبان اسمبلی است، در فصول ابتدایی این سلسله مقالات ما کتاب‌های نوشته شده شخص Kip Irvine و Richard Blum را به منظور مطالعه معرفی کردیم، اما به دلیل عدم تسلط برخی از افراد روی زبان انگلیسی، در اینجا لازم دیدیم کتاب‌هایی را به زبان پارسی برای یادگیری اسمبلی معرفی کنیم. کتاب‌های زبان

---

[1] Registers window

ماشین و اسمبلی و کاربرد آن در رایانه نوشته دکتر حسن سید رضی از انتشارات ناقوس و کتاب زبان اسمبلی پیشرفته ترجمه کیاوش و تکاوش بحرینی از انتشارات دیباگران، کتاب‌های مفیدی در زمینه یادگیری زبان اسمبلی هستند که شما می‌توانید از آن‌ها استفاده کنید. البته شایان ذکر است کتاب زبان اسمبلی پیشرفته، ترجمه کتاب Assembly Professional Programing است که به شما برنامه‌نویسی اسمبلی با گرامر دستوری At&t را آموزش می‌دهد. کتاب Assembly Professional Programing یکی از بهترین کتاب‌های موجود برای کسانی است به دنبال یادگیری زبان اسمبلی هستند. زیرا ترتیب و محتویات موجود در آن بسیار با ارزش می‌باشد.

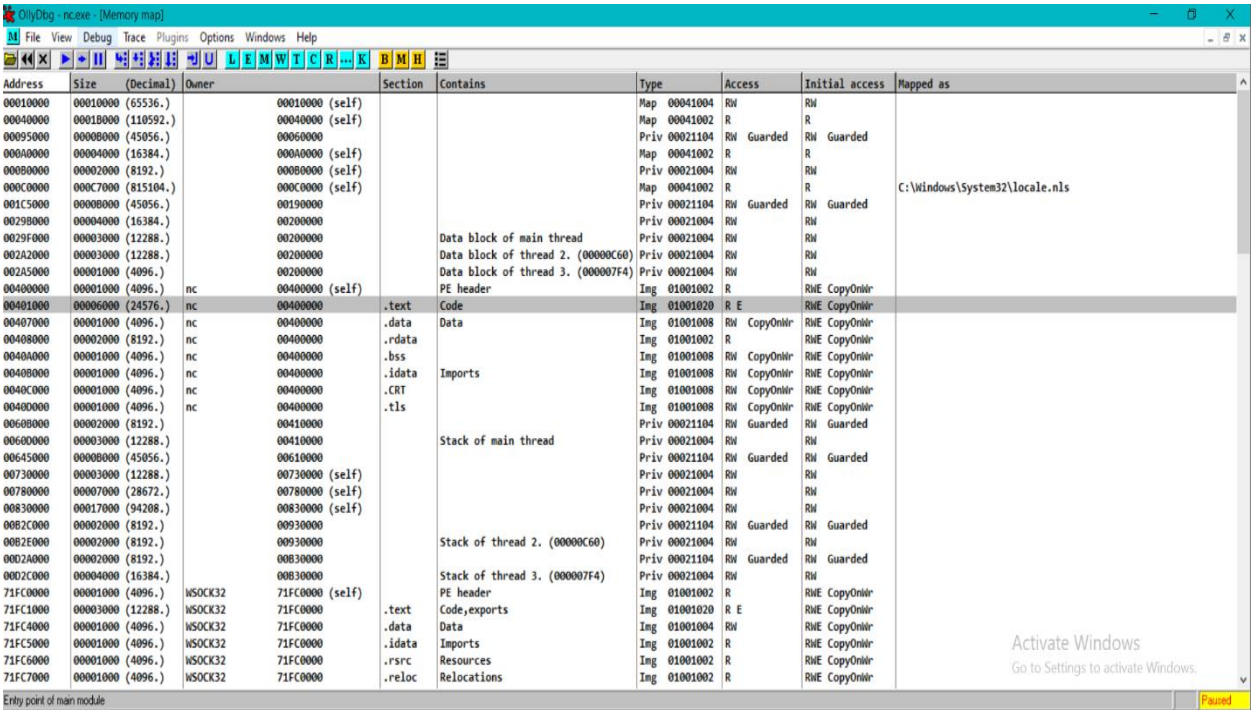
- **پنجره پشته[1] (شماره ۳):** این پنجره وضعیت جاری حافظه پشته را برای ترد دیباگ شده فعلی به نمایش می‌گذارد. این پنجره همیشه ابتدای پشته را برای ترد تحلیل شده نشان می‌دهد. شایان ذکر است، در این پنجره هم می‌توانید مانند پنجره‌های قبلی با کلیک راست روی یک آدرس پشته و انتخاب گزینه Modify محتوای پشته را دستکاری کنید. همچنین دیباگر OllyDBG یک مجموعه از توضیحات به مکان‌های پشته می‌افزاید که توضیح دهنده پارامترهایی می‌باشد که پیش از فراخوانی یک API در پشته قرار گرفته‌اند. این کار به تحلیل برنامه کمک زیادی می‌کند، زیرا دیگر نیاز ندارید ترتیب پشته و همچنین ترتیب پارامترهای API را بدانید.
- **پنجره دامپ از حافظه (شماره ٤):** این پنجره محتویات حافظه فعال پروسه دیباگ شده را نمایش می‌دهد. دکمه Ctrl+g را در این پنجره فشار دهید و آدرس حافظه‌ای را که می‌خواهید از آن دامپ بگیرید را در آن پنجره وارد کنید. (یا در پنجره دیزاسمبلر روی آدرس حافظه مد نظر خود کلیک راست کرده و گزینه Follow in Dump را برای مشاهده محتویات آن آدرس حافظه انتخاب کنید.). همچنین برای ویرایش مقادیر دامپ شده از حافظه در این پنجره می‌توانید روی آن مقدار

[1] Stack window

کلیک راست کرده و از منوی Binary گزینه Edit را انتخاب کنید. از این گزینه می‌توان برای تغییر در متغیرهای عمومی و سایر داده‌هایی که بدافزار در حافظه ذخیره می‌سازد، استفاده کنید.

نقشه حافظه[1]

پنجره نقشه حافظه (منوی View گزینه Memory) تمام خانه‌های حافظه را که به برنامه دیباگ شده اختصاص داده شده است، نمایش می‌دهد. تصویر ۵ نقشه حافظه را برای برنامه Netcat نمایش می‌دهد.



تصویر ۵: نقشه حافظه برای Netcat (nc.exe)

نقشه حافظه یک روش خوبی برای دیدن نحوه قرارگیری برنامه و ماژول‌های آن در حافظه است. همان‌طور که در تصویر ۵ مشاهده می‌کنید، برنامه اجرایی به همراه کد و سکشن داده‌های خودش مشخص شده است. همچنین تمامی کتابخانه‌های پیوندی پویا به همراه کدها و سکشن‌های دیگر آن‌ها نیز قابل مشاهده هستند. شما می‌توانید روی هر ردیفی در نقشه حافظه دو بار کلیک کنید تا بتوانید دامپ آن بخش از حافظه را مشاهده کنید. شایان ذکر است، در پنجره Memory Dump همچنین می‌توان با انتخاب سکشن کد (.text) هر ماژولی و انتخاب View in Disassembler دستورات دیزاسمبلی کد آن ماژول را مشاهده کرد.

[1] Memory Map

تغییر آدرس پایه[1]

نقشه حافظه می‌تواند به شما کمک می‌کند تا متوجه شوید یک فایل‌اجرایی چگونه در زمان اجرا آدرس پایه آن تغییر می‌کند و در محل پیش‌فرض خود بارگزاری نمی‌شود. تغییر آدرس پایه باینری زمانی رخ می‌دهد که یک ماژول در آدرس پایه‌ای که باید قرار می‌گرفت، جای نگرفته باشد.

آدرس‌های پایه[2]

همه فایل‌های PE ویندوز دارای یک آدرس پایه از پیش تعریف شده هستند که با عنوان Image Base در هدر PE شناخته می‌شود. آدرس در فیلد ImageBase لزوما آدرسی نیست که لودر سیستم‌عامل باینری بدافزار را در آن بارگذاری کند، گرچه معمولا این اتفاق رخ می‌دهد.

اکثر فایل‌های اجرایی برای بارگذاری در آدرس 0x00400000 طراحی می‌شوند که آدرس پیش‌فرض مورد استفاده بسیاری از کامپایلرها برای ویندوز است. با این حال، توسعه‌دهندگان نرم‌افزار می‌توانند آدرس پایه از پیش تعریف شده متفاوتی برای برنامه‌های خود انتخاب کنند. برنامه‌های اجرایی که از ویژگی امنیتی تصادفی‌سازی آدرس‌های حافظه[3] پشتیبانی می‌کنند، معمولا در حافظه تغییر موقعیت یا به عبارت دقیق‌تر Relocation می‌شوند. البته، تغییر آدرس کتابخانه‌های پیوندی پویا (DLL) رایج‌تر است.

تغییر موقعیت آدرس‌ها لازم است زیرا امکان دارد یک برنامه کتابخانه‌های مختلفی را با یک آدرس پایه مشابه به خود وارد کند. لذا اگر دو کتابخانه DLL بارگذاری شوند و دارای آدرس بارگذاری 0x10000000 باشند، هر دوی آن‌ها نمی‌توانند در آنجا بارگذاری شوند. از همین روی، ویندوز یک کتابخانه را در آن آدرس و دیگری را در یک آدرس دیگر بارگذاری می‌کند.

اکثریت کتابخانه‌های پیوندی پویا که به همراه سامانه‌عامل ویندوز نصب می‌شوند، دارای آدرس پایه از پیش تعریف شده متفاوتی هستند و با هم تداخلی ندارند. اگرچه برنامه‌های ثالث ممکن است دارای آدرس از پیش تعریف شده مشابه باشند.

---

[1] Rebasing
[2] Base Addresses
[3] Address Space Layout Randomization (ASLR)

آدرس‌های نسبی در مقابل آدرس‌های مطلق[1]

فرآیند تغییر موقعیت آدرس بارگذاری یک باینری در یک آدرس دیگر پیچیده‌تر از یک تغییر موقعیت ساده است. بسیاری از دستورالعمل‌ها به آدرس‌های نسبی در حافظه اشاره می‌کنند در حالی‌که بعضی دیگر به آدرس مطلق ارجاع داده می‌شوند. به عنوان مثال لیست ۱ یک مجموعه از دستورالعمل‌های معمولی را نمایش می‌دهد.

```
00401203        mov eax, [ebp+var_8]
00401206        cmp [ebp+var_4], 0
0040120a        jnz loc_0040120
0040120c      ❶mov eax, dword_40CF60
```

لیست ۱: کد اسمبلی که نیازمند تغییر موقعیت است.

بسیاری از دستورالعمل‌ها به دلیل استفاده از آدرس‌های نسبی بدون توجه به این‌که در کجای حافظه بارگذاری شده‌اند، به درستی کار می‌کنند. اما، دستورالعمل دسترسی به داده (شماره ۱) کار نخواهد کرد، زیرا از یک آدرس مطلق برای دسترسی به یک بخش از حافظه استفاده می‌کند. تحت این شرایط، اگر باینری در موقعیتی غیر از موقعیت پایه از پیش تعریف شده درون حافظه بارگذاری شود، در نتیجه آدرس مطلق در لیست ۱ اشتباه خواهد بود. بسیاری از کتابخانه‌های پیوندی پویا دارای آدرس‌های متغیری هستند که در بخش reloc. از هدر PE قرار دارند.

نکته: در واقع کد موجود در (شماره ۱) مقداری را که در آدرس 40CF60 (در واقع آدرس 0040CF60) قرار دارد را به عنوان یک DWORD به ثبات EAX منتقل می‌کند. در اینجا به دلیل این‌که از یک آدرس مطلق استفاده شده است، با تغییر محیط آدرس‌دهی برنامه (از محیط آدرس‌دهی ۳۲ بیتی به محیط آدرس‌دهی ٦٤ بیتی) ممکن است آدرس 40CF60 برای برنامه در دسترس نباشد و در نتیجه شما با خطای دسترسی غیرمجاز به حافظه[2] مواجه شوید و در نتیجه ممکن است این خطا موجب شود اجرای برنامه به کلی خاتمه یابد.

---

[1] Absolute vs. Relative Addresses
[2] Memory Access Violation

کتابخانه‌های پیوندی پویا پس از اجرای باینری با یک ترتیب نامشخص بارگذاری می‌شوند. این به این معناست که اگر کتابخانه‌های پیوندی پویا آدرس پایه خود را عوض کنند، نمی‌توان پیش‌بینی کرد که در کجای حافظه قرار خواهند گرفت. کتابخانه‌های پیوندی پویا می‌توانند فاقد بخش تغییر موقعیت مجدد[1] باشند و اگر یک کتابخانه پیوندی پویا دارای این وضعیت باشد، اگر نتواند در آدرس دلخواه خود بارگذاری شود، موقعیت آدرس پایه آن تغییر پیدا نخواهد کرد و در نتیجه بارگذاری نخواهد شد.

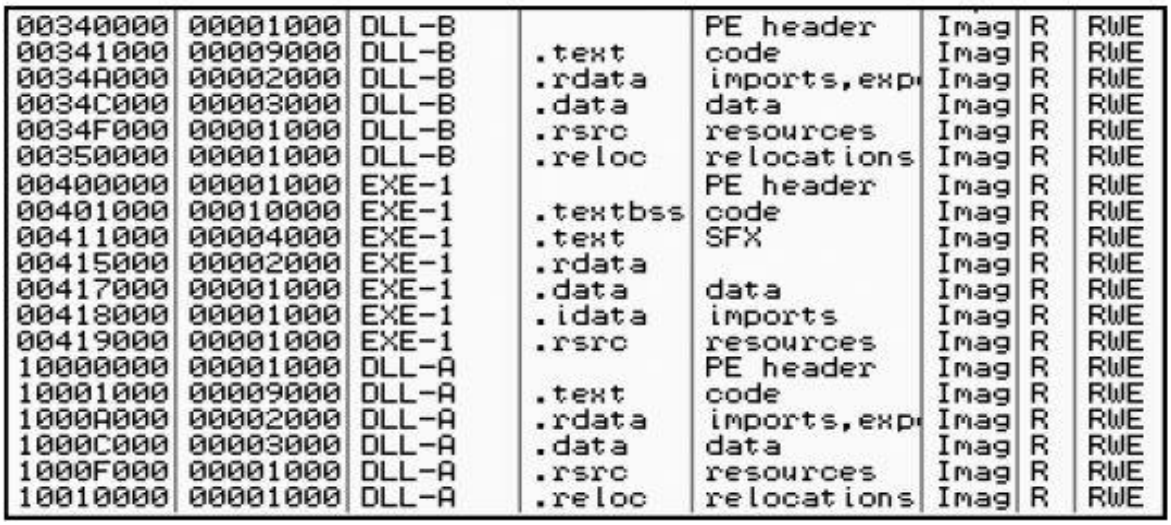
تغییر موقعیت مجدد آدرس کتابخانه‌های پیوندی پویا برای کارآیی سامانه خوب نبوده و زمان بارگذاری را افزایش خواهد داد. از همین روی کامپایلر از یک آدرس پایه پیش‌فرض برای تمامی کتابخانه‌های پیوندی پویا در زمان کامپایل استفاده می‌کند و عموما آدرس پایه پیش‌فرض برای تمامی کتابخانه‌های پیوندی پویا یکسان است.

به همین دلیل شانس تغییر موقعیت افزایش می‌یابد، چراکه تمامی کتابخانه‌های پیوندی پویا برای بارگذاری در یک آدرس یکسان طراحی شده‌اند. برنامه‌نویسان خبره از این مشکل آگاه بوده و خودشان آدرس پایه‌ای را برای کتابخانه پیوندی پویا خود در نظر می‌گیرند تا میزان تغییر موقعیت را کاهش دهند.

تصویر ۶ با استفاده از ویژگی نقشه حافظه در دیباگر OllyDBG تغییر موقعیت کتابخانه پیوندی پویایی را برای فایل EXE-1 نشان می‌دهد. همان‌طور که مشاهده می‌کنید، یک بخش اجرایی و دو کتابخانه پیوندی پویا داریم. DLL-A با آدرس بارگذاری از پیش تعریف شده 0x10000000 در حافظه قرار دارد و فایل EXE-1 دارای آدرس بارگذاری از پیش تعریف شده 0x00400000 است.

زمانی‌که Dll-B بارگذاری شده بود، دارای آدرس بارگذاری از پیش تعریف شده 0x10000000 بود و به دلیل تداخل با آدرس 0x00340000 تغییر موقعیت داده شده بود. تمامی آدرس‌های مطلق ارجاع شده در DLL-B تغییر داده شده‌اند تا بتوانند با این آدرس جدید کار کنند.

[1] Relocation Section



00340000	00001000	DLL-B		PE header	Imag	R	RWE
00341000	00009000	DLL-B	.text	code	Imag	R	RWE
0034A000	00002000	DLL-B	.rdata	imports,exp	Imag	R	RWE
0034C000	00003000	DLL-B	.data	data	Imag	R	RWE
0034F000	00001000	DLL-B	.rsrc	resources	Imag	R	RWE
00350000	00001000	DLL-B	.reloc	relocations	Imag	R	RWE
00400000	00001000	EXE-1		PE header	Imag	R	RWE
00401000	00010000	EXE-1	.textbss	code	Imag	R	RWE
00411000	00004000	EXE-1	.text	SFX	Imag	R	RWE
00415000	00002000	EXE-1	.rdata		Imag	R	RWE
00417000	00001000	EXE-1	.data	data	Imag	R	RWE
00418000	00001000	EXE-1	.idata	imports	Imag	R	RWE
00419000	00001000	EXE-1	.rsrc	resources	Imag	R	RWE
10000000	00001000	DLL-A		PE header	Imag	R	RWE
10001000	00009000	DLL-A	.text	code	Imag	R	RWE
1000A000	00002000	DLL-A	.rdata	imports,exp	Imag	R	RWE
1000C000	00003000	DLL-A	.data	data	Imag	R	RWE
1000F000	00001000	DLL-A	.rsrc	resources	Imag	R	RWE
10010000	00001000	DLL-A	.reloc	relocations	Imag	R	RWE

تصویر ۶: Dll-B از آدرس درخواستی خود به یک آدرس حافظه متفاوت دیگر جا به جا شده است.

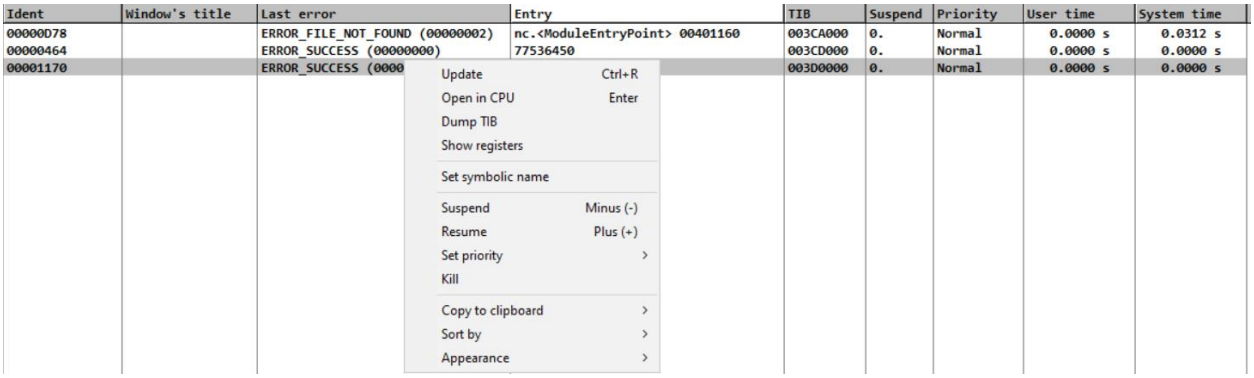
زمانی‌که برنامه را دیباگ می‌کنید و همزمان به Dll-B در IDA Pro نگاه کنید، مشاهده خواهید کرد که آدرس‌ها مشابه نخواهند بود، زیرا IDA Pro هیچ اطلاعی از شناسایی مجدد آدرس‌های پایه در زمان اجرا ندارد. شما باید مرتب این آدرس‌ها را تعیین کنید تا بتوانید این مشکل را حل کنید. همچنین می‌توانید از پروسه بارگذاری دستی (Manual Load) استفاده کنید.

مشاهده تردها و پشته‌ها[1]

بدافزارها اغلب از چندین ترد استفاده می‌کنند. شما می‌توانید با انتخاب View و سپس گزینه Threads، تردهای جاری برنامه را مشاهده کنید. این پنجره آدرس حافظه تردها و وضعیت فعلی آن‌ها (که شامل فعال بودن، توقف موقت و تعلیق می‌شود) را نشان می‌دهد.

از آنجایی که دیباگر OllyDBG یک ابزار تک تردی است، لازم است تمامی تردها به صورت موقت متوقف شده و یک نقطه توقف ایجاد شود و سپس اجرای برنامه را ادامه دهید تا بتوانید عملیات دیباگ را برای یک ترد خاص انجام دهید. به منظور متوقف کردن تمامی تردها می‌توانید روی دکمه Pause در نوار ابزار اصلی دیباگر OllyDBG کلیک کنید. تصویر ۷ یک نمونه از پنجره تردها را پس از توقف پنج ترد را نشان می‌دهد. همچنین شما می‌توانید با کلیک راست روی یک ترد خاص و انتخاب گزینه Kill آن ترد را از بین ببرید.

[1] Viewing Threads and Stacks



تصویر ۷: این تصویر پنجره تردها که در آن ۳ ترد متوقف شده است و همچنین منوی تنظیمات برای یک ترد را نمایش می‌دهد.

هر ترد در یک پروسه دارای پشته خاص خود می‌باشد و اطلاعات مهم آن در پشته ذخیره می‌شود. بدین منظور شما می‌توانید از گزینه Memory map در دیباگر OllyDBG به منظور مشاهده پشته حافظه استفاده کنید. به عنوان مثال، در تصویر ۵، می‌توانید مشاهده کنید که دیباگر OllyDBG ترد تابع اصلی برنامه را با عنوان "stack of main thread." نشان داده است.

اجرای کد

دانش کامل و توانایی اجرای کدها درون دیباگر در موفقیت عملیات دیباگ مهم است. همچنین شایان ذکر است، روش‌های بسیاری برای اجرای کدها در دیباگر OllyDBG وجود دارد که جدول ۱ لیست برخی از آن‌ها را نشان می‌دهد.

جدول ۱: گزینه‌های اجرای کد OllyDBG

Function	Menu	Hotkey	Button
Run/Play	Debug ▸ Run	F9	
Pause	Debug ▸ Pause	F12	
Run to selection	Breakpoint ▸ Run to Selection	F4	
Run until return	Debug ▸ Execute till Return	CTRL-F9	
Run until user code	Debug ▸ Execute till User Code	ALT-F9	
Single-step/step-into	Debug ▸ Step Into	F7	
Step-over	Debug ▸ Step Over	F8	

راحت‌ترین گزینه Run و Pause است که باعث اجرا و توقف برنامه در حال اجرا می‌شوند. با این حال Pause به ندرت استفاده می‌شود، زیرا باعث می‌شود برنامه در نقطه‌ای متوقف شود (به عنوان مثال در بخش کدهای کتابخانه‌ای) که چندان مفید نیست (در نسخه ۲ دیباگر Ollydbg حذف شده است). بجای استفاده از دکمه Pause معمولا باید نقاط توقف خود را در آدرس‌های دلخواه قرار دهید که این موضوع در بخش بعد مورد بررسی قرار خواهد گرفت.

گزینه Run بیشتر برای اجرای مجدد یا ادامه اجرای یک پروسه متوقف شده استفاده می‌شود، البته پس از آن‌که در نقطه از پیش تعیین شده (Breakpoint) برنامه متوقف شد. گزینه Run to Selection کد مورد نظر را تا پیش از دستورالعملی که در دیباگر انتخاب شده است، اجرا می‌کند. در صورتی که دستورالعمل مشخص شده هیچ وقت اجرا نشود، برنامه به اجرا شدن خود ادامه می‌دهد.

گزینه Run utill Return در منوی دیباگ اجرای برنامه را پیش از بازگشت تابع فعلی متوقف می‌کند. این ویژگی زمانی‌که بخواهیم یک برنامه را به محض پایان یافتن یک تابع متوقف سازیم، استفاده می‌شود. در صورتی که تابع پایان نیابد برنامه به اجرای خود ادامه می‌دهد. گزینه Run untill User Code هنگامی‌که در زمان تحلیل بدافزار کد کتابخانه‌ای در حین دیباگ گم شود، مفید می‌باشد. وقتیکه در بخش کدهای کتابخانه‌ای متوقف هستید از منوی Debug گزینه Run untill User Code را انتخاب کنید که باعث می‌شود برنامه تا جایی که پروسه اجرا به کد کامپایل شده بدافزار (عموماً بخش .text) بر می‌گردد اجرا شود.

دیباگر OllyDBG روش‌های مختلفی را برای گام برداشتن در کد ارائه می‌دهد. همان‌طور که در فصل قبل بحث کردیم، گام برداری به معنای اجرای یک دستورالعمل و سپس موقتاً متوقف ساختن اجرا به محض اجرای دستورالعمل می‌باشد و به شما اجازه می‌دهد که وضعیت اجرای برنامه را گام به گام با دستورالعمل‌های اجرایی برنامه طی کنید.

دیباگر OllyDBG دو نوع گام برداری را پیشنهاد می‌کند که در فصل قبل در مورد آن‌ها صحبت کردیم، که شامل گام برداشتن به داخل (Stepping into) و گام برداشتن از روی (Stepping Over) هستند. برای اجرای گام به گام دستورالعمل‌های کد برنامه، کلید F7 و برای گام برداشتن از روی یک تابع کلید F8 را می فشاریم.

همان‌طور که قبلا گفته‌ایم، مرحله به مرحله گام برداشتن (Single Step) ساده‌ترین روش گام‌برداری در دیباگر بوده است. در این روش، OllyDBG یک دستورالعمل از برنامه را اجرا و سپس به صورت موقت توقف می‌کند فارق از اینکه نوع دستورالعمل چه خواهد بود. به عنوان مثال، اگر شما با روش اجرای گام به گام دستورالعمل call 01007568 را اجرا کنید، دیباگر OllyDBG در آدرس 01007568 به صورت موقت برنامه را متوقف می‌کند (زیرا دستورالعمل call ثبات EIP را با آدرس 01007568 تنظیم کرده است). از لحاظ مفهومی گام برداشتن از روی (Stepping Over) تقریبا به سادگی مرحله به مرحله گام برداشتن می‌باشد. دستورالعمل‌های زیر را در نظر بگیرید.

```
010073a4    call    01007568
010073a9    xor     ebx, ebx
```

اگر از روی دستورالعمل Call 01007568 به روش گام برداشتن از روی (Step over) پرش کنید. دیباگر OllyDBG در آدرس 010073a9 موقتا متوقف می‌شود و وارد تابع نخواهد شد. این ویژگی زمانی مفید می‌باشد که شما نخواهید وارد سابروتین موجود در آدرس 01007568 شوید.

اگرچه گام برداشتن از روی (Step Over) از لحاظ مفهومی ساده می‌باشد، اما در لایه‌های زیرین آن مفاهیم پیچیده‌تری قرار دارد. در واقع دیباگر OllyDBG یک نقطه توقف را در آدرس 010073a9 قرار داده و سپس اجرا را ادامه می‌دهد (مانند این‌که دکمه Run را دیباگر فشار داده باشید) و سپس وقتی سابروتین دستورالعمل ret (بازگشت) را اجرا می‌کند دیباگر OllyDBG برنامه را در 010073a9 متوقف می‌کند.

هشدار: تقریبا در تمامی موارد، گام برداشتن از روی (Stepping Over) کار می‌کند. اما در بعضی موارد نادر، ممکن است برای یک کد مبهم‌سازی شده یا مخرب از این پروسه سوءاستفاده شود. به عنوان مثال، سابروتین موجود در آدرس 01007568 ممکن است هیچگاه دستورالعمل ret (بازگشت) را اجرا نکرده و یا دستورالعمل get-EIP را اجرا کند که همین موضوع باعث می‌شود آدرس بازگشتی از پشته خارج شود. در این موارد خاص، گام برداشتن از روی

(Stepping Over) می‌تواند منجرب به ادامه اجرای برنامه بدون توقف موقت آن شود. لذا از این نکته آگاه بوده و با احتیاط از این ویژگی استفاده کنید.

نقاط توقف

همان‌طور که در فصل قبل بحث کردیم، انواع مختلفی از نقاط توقف وجود دارند که دیباگر OllyDBG از همه آن‌ها پشتیبانی می‌کند. به صورت پیش‌فرض OllyDBG از نقاط توقف نرم‌افزاری پشتیبانی می‌کند، اما شما می‌توانید علاوه بر آن از نقاط توقف سخت‌افزاری هم استفاده کنید. همچنین می‌توانید از نقاط توقف شرطی و نقاط توقف بر روی حافظه نیز استفاده کنید.

شما می‌توانید نقطه توقف را در پنجره دیزاسمبلر و با انتخاب دستورالعمل مورد نظر و سپس فشار دادن کلید F2 فعال یا غیر فعال کنید. همچنین می‌توانید نقاط توقف فعال در برنامه را با انتخاب منوی View و سپس گزینه Breakpoints و یا کلیک روی آیکون B در نوار ابزار دیباگر OllyDBG مشاهده کنید.

بعد از این‌که برنامه دیباگ شده را متوقف می‌کنید یا آن را می‌بندید، دیباگر OllyDBG نقاط توقفی را که شما پیش از این فعال کردید را ذخیره کرده و به شما این امکان را می‌دهد که برنامه را مجدداً در نقاط توقف یکسان دیباگ کنید (بدین معنا که دیگر لازم نیست نقاط توقف را اعمال کنید). جدول ۲ یک لیست کامل از نقاط توقف دیباگر OllyDBG را نمایش می‌دهد.

جدول ۲: گزینه‌های نقاط توقف دیباگر OllyDBG

Function	Right-click menu selection	Hotkey
Software breakpoint	Breakpoint ▸ Toggle	F2
Conditional breakpoint	Breakpoint ▸ Conditional	SHIFT-F2
Hardware breakpoint	Breakpoint ▸ Hardware, on Execution	
Memory breakpoint on access (read, write, or execute)	Breakpoint ▸ Memory, on Access	F2 (select memory)
Memory breakpoint on write	Breakpoint ▸ Memory, on Write	

نکته: در نسخه ۲ دیباگر OllyDBG، کلیات تغییری نکرده است اما به جای گزینه Memory, on Access ما فقط دارای گزینه Memory هستیم. برای نقاط توقف سخت‌افزاری همچنین ما فقط دارای گزینه Hardware هستم که از این دو گزینه می‌توان برای تنظیم نقاط توقف مورد نظر خود که در بالا تشریح کردیم، استفاده بهینه کنیم.

نقاط توقف نرم‌افزاری[1]

نقاط توقف نرم‌افزاری هنگام دیباگ یک تابع رمزگشای یک عبارت رشته‌ای[2] می‌توانند بسیار کاربردی باشند. از فصل اول به یاد آورید که رشته‌ها می‌توانستند برای درک نحوه کار یک برنامه مفید واقع گردند، به همین دلیل نویسندگان بدافزار معمولا تلاش می‌کنند رشته‌های درون بدافزار را مبهم‌سازی کنند. زمانی‌که نویسندگان بدافزار چنین کاری انجام می‌دهند، معمولا از یک تابع رمزگشای برای عبارت رشته‌ای مبهم‌سازی شده خود استفاده می‌کنند که پیش از استفاده از آن رشته فراخوانی می‌شود. لیست ۲ نمونه‌ای از فراخوانی تابع String_Decoder را پس از آنکه داده مبهم‌سازی شده وارد پشته می‌شود، نمایش می‌دهد.

```
push offset "4NNpTNHLKIXoPm7iBhUAjvRKNaUVBlr"
call String_Decoder
...
push offset "ugKLdNlLT6emldCeZi72mUjieuBqdfZ"
call String_Decoder
...
```

لیست ۲: یک نقطه توقف رمزگشای رشته

این تابع، داده مبهم‌سازی شده را معمولا به صورت یک رشته در پشته رمزگشایی می‌کند، پس تنها راه ممکن رسیدن به رشته، مشاهده پشته پس از کامل شدن عملیات رمزگشایی است. بنابراین بهترین محل برای قرار دادن نقطه توقف و مشاهده کامل رشته، انتهای روتین رمزگشایی است. بدین وسیله هر زمان که شما دکمه Play را در دیباگر OllyDBG انتخاب می‌کنید. برنامه اجرا شده و در نقطه‌ای که رشته رمزگشایی شده است،

[1] Software Breakpoints
[2] String Decoder

متوقف می‌شود (به دلیل ایجاد نقطه توقف). این روش تنها رشته‌هایی را شناسایی می‌کند که در خود برنامه در آن لحظه استفاده می‌شوند، ادامه این فصل در مورد چگونگی تغییر دستورالعمل‌ها برای رمزگشایی تمام رشته‌ها صحبت خواهیم کرد.

نقطه های توقف شرطی

همان‌طور که در فصل قبل یاد گرفتید، نقاط توقف شرطی، نقاط توقف نرم‌افزاری هستند که تنها زمانی‌که شرایط مشخصی رخ دهد، فعال می‌شوند. دیباگر OllyDBG به شما اجازه می‌دهد نقاط توقف شرطی را با استفاده از عبارات شرطی ایجاد کنید؛ هر زمانی‌که دیباگر OllyDBG به نقطه توقف شرطی شما می‌رسد عبارت شرطی بررسی می‌شود و در صورتی که نتیجه عبارت شرطی غیر صفر باشد، اجرای برنامه موقتا متوقف می‌شود.

هشدار : در زمان استفاده از نقاط توقف شرطی مراقب باشید. ایجاد آن‌ها ممکن است باعث شود برنامه شما بسیار کند اجرا شود و اگر شما شرط خود را درست ایجاد نکنید، ممکن است اجرای برنامه هیچ وقت متوقف نشود.

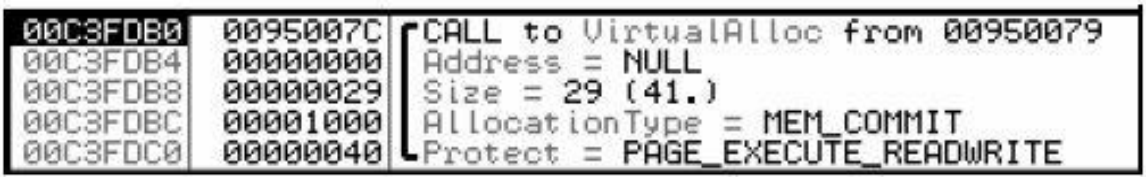
نقاط توقف شرطی می‌توانند زمانی‌که شما می‌خواهید در وقت صرفه‌جویی کنید و اجرای برنامه را هنگامی‌که پارامتر خاصی به رابطهای برنامه‌نویسی رایج ارسال می‌شوند متوقف کنید، برای شما مفید واقع شوند. نمونه‌ای از این کار را می‌توانید در ادامه مشاهده کنید. حتی می‌توانید از نقاط توقف شرطی برای شناسایی تخصیص حافظه بیش از اندازه استفاده کنید.

به عنوان مثال، در پشتی Poison Ivy که فرامین خود را از سرور کنترل و فرماندهی تحت مدیریت نفوذگر دریافت می‌کند، در نظر بگیرید (Poison Ivy در گروه بات‌نت‌ها قرار می‌گیرد). فرامین در شلکد این ابزار پیاده‌سازی شده‌اند و Poison ivy برای نگهداری این شلکدها حافظه‌ای را به آن‌ها اختصاص می‌دهد. اگرچه بیشتر تخصیص حافظه‌ها در Poison Ivy کوچک هستند. مگر زمانی‌که سرور کنترل و فرماندهی در حجم زیادی شلکد برای اجرا ارسال کند.

بهترین راه حل برای کشف اختصاص حافظه به شلکد در Poison Ivy ایجاد نقطه توقف شرطی در تابع VirtualAlloc متعلق به کتابخانه Kernel32.dll می‌باشد. VirtualAlloc یک API است که

Poison Ivy برای تخصیص حافظه پویا از آن استفاده می‌کند، بنابراین در صورتی که شما نقطه توقف شرطی را با شرط اختصاص حافظه بیش از ۱۰۰ بایت تعریف کنید، برنامه زمانی‌که حافظه کمتری از این مقدار به شلکد تخصیص دهد، متوقف نمی‌شود.

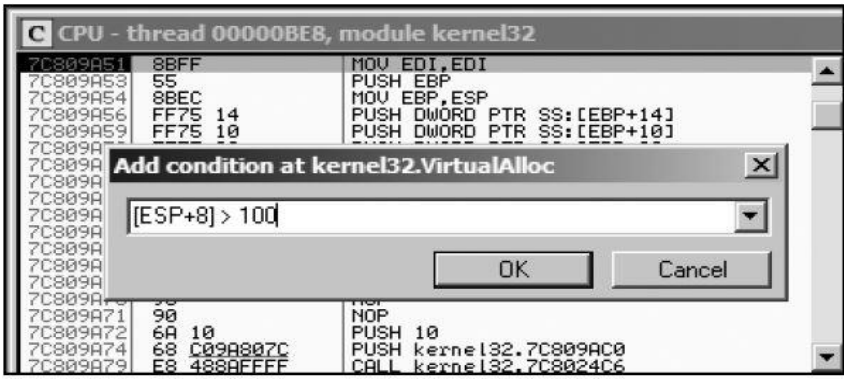
برای ایجاد این تله، می‌توانیم یک نقطه توقف عادی را در ابتدای تابع VirtualAlloc قرار دهیم. برنامه تا زمان رسیدن به این تابع اجرا می‌شود. تصویر ۸ پنجره پشته را زمانی‌که نقطه توقف در ابتدای تابع VirtualAlloc فعال می‌شود، نمایش می‌دهد.



تصویر ۸: پنجره پشته در ابتدای **VirtualAlloc**

شکل بالا پنج آیتم بالای پشته را نمایش می‌دهد. آدرس بازگشتی اولین آیتم است و چهار پارامتر دیگر که پس از آن آمده‌اند (Address، Size، AllocationType و Protect) برای تابع VirtualAlloc هستند. پارامترها در کنار مقادیر و آدرسشان در پشته برچسب خورده‌اند. در این مثال، 29 بایت فضا به شلکد اختصاص داده شده است.

از آنجایی‌که توسط ثبات ESP به ابتدای پشته اشاره می‌شود، برای دسترسی به فیلد اندازه، ما باید به آن در حافظه بدین گونه [ESP+8] اشاره کنیم. تصویر ۹ پنجره دیزاسمبلر را هنگامی‌که نقطه توقف به شروع VirtualAlloc می‌رسد، نمایش می‌دهد. سپس ما یک نقطه توقف شرطی با شرط [ESP+8]>100 تنظیم می‌کنیم، تا زمانی‌که Poison Ivy مقدار بیش از اندازه‌ای شلکد دریافت می‌کند، متوقف شود. به منظور تنظیم نقطه توقف شرطی گام‌های آورده شده در زیر را دنبال کنید.

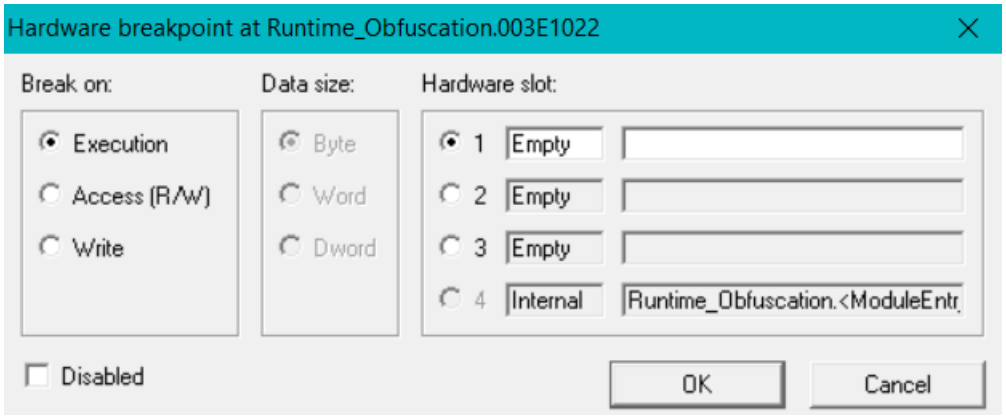


تصویر ۹: تنظیم یک نقطه توقف شرطی در پنجره دیزاسمبلر

۱. در پنجره دیزاسمبلر روی اولین دستورالعمل تابع کلیک راست کرده و سپس از منوی Breakpoint گزینه Conditional را انتخاب کنید. پس از انتخاب این گزینه یک پنجره محاوره‌ای نمایش داده می‌شود که می‌توانید در آن شرط خود را اعمال کنید.
۲. عبارت شرطی خود را وارد کنید و سپس روی دکمه Ok کلیک کنید. در این مثال ما از عبارت شرطی [ESP+8]>100 استفاده می‌کنیم.
۳. پس از اعمال نقطه توقف شرطی روی دکمه Play کلیک کنید و صبر کنید پروسه اجرای برنامه به نقطه توقف برسد.

نقطه‌های توقف سخت‌افزاری

دیباگر OllyDBG به ما ویژگی قرار دادن نقطه توقف سخت‌افزاری با استفاده از ثبات‌های مخصوصی را ارائه می‌دهد که در فصول گذشته توضیح داده شده است. نقاط توقف سخت‌افزاری بسیار قدرتمند هستند، زیرا آن‌ها کد خود، پشته و دیگر منابع هدف را تغییر نمی‌دهند و همچنین باعث کاهش سرعت اجرای پروسه دیباگ نمی‌شوند. همان‌طور که در فصول گذشته ذکر کردیم، تنها مشکل استفاده از نقاط توقف سخت‌افزاری این است که فقط می‌توانید چهارتا از نقاط توقف سخت‌افزاری را در هر لحظه تنظیم کرده و مورد استفاده قرار بدهید. در تصویر ۱۰، این مسئله نمایش داده شده است.



تصویر ۱۰: تنظیم نقاط توقف سخت‌افزاری

برای تنظیم یک نقطه توقف سخت‌افزاری، روی یک دستورالعمل کلیک راست کرده و از منوی Breakpoint گزینه Hardware, on Execution را انتخاب کنید (در دیباگر OllyDbg نسخه ۲ گزینه Hardware را باید انتخاب کنید). همچنین می‌توانید به دیباگر OllyDBG بگوید که به صورت

پیش‌فرض از نقاط توقف سخت‌افزاری بجای نقاط توقف نرم‌افزاری استفاده کند. بدین منظور کافی است به منوی Options بروید و در آنجا گزینه Debugging Options را انتخاب کنید تا پنجره تنظیمات باز شود. سپس در این پنجره می‌توانید از زبانه Debug تنظیمات مد نظر خود را اعمال کنید.

با این حال گاهی اوقات نیاز است که شما از این گزینه بهره‌مند شوید تا بتوانید در مقابل بعضی از روش‌های ضد دیباگ مانند پویش نقاط توقف نرم‌افزاری (Software Breakpoint Scanning) مقابله کنید. این مورد در فصل‌های آینده مورد بررسی قرار خواهد گرفت.

نقطه‌های توقف در حافظه

دیباگر OllyDBG از قرار دادن نقاط توقف در حافظه پشتیبانی می‌کند، که اجازه می‌دهد روی یک قسمت از حافظه نقطه توقف اعمال کنید تا اگر برنامه در حین اجرا به آن قسمت از حافظه دسترسی گرفت، متوقف شود. دیباگر OllyDBG همچنین از نقاط توقف نرم‌افزاری و نقاط توقف سخت‌افزاری حافظه هم پشتیبانی می‌کند.

برای تنظیم یک نقطه توقف در حافظه یک تکه قسمت از حافظه را در پنجره Memory Dump انتخاب کرده و روی آن کلیک راست کنید، سپس از منوی Breakpoint گزینه Memory, on Access را انتخاب کنید (در نسخه ۲ دیباگر، گزینه Memory را انتخاب کنید). همچنین به این نکته دقت کنید، در هر بار فقط می‌توانید یک نقطه توقف در حافظه قرار دهید، اگر نقطه توقف جدیدی در حافظه تنظیم کنید، نقطه توقف قبلی به صورت خودکار حذف می‌شود.

شایان ذکر است، دیباگر OllyDBG با تغییر خواص بلاک حافظه کد انتخابی شما نقطه توقف حافظه نرم‌افزاری را پیاده‌سازی می‌کند. با این حال، این روش همیشه پایدار نیست و می‌تواند سربار اضافی قابل توجه‌ای ایجاد کند. بنابراین بهتر است از نقاط توقف حافظه به مقدار کمی استفاده کنید.

در حین تجزیه و تحلیل بدافزار، هنگامی‌که می‌خواهید بفهمید چه موقع یک کتابخانه پیوندی پویا بارگذاری شده و مورد استفاده قرار می‌گیرد، نقاط توقف حافظه بسیار مفید هستند. در این شرایط شما می‌توانید به محض این‌که کد درون کتابخانه پیوندی پویا اجرا شد از یک نقطه توقف حافظه برای موقتا متوقف ساختن اجرای برنامه استفاده کنید. برای انجام این عملیات گام‌های زیر را دنبال کنید.

۱. به پنجره Memory Map بروید (View > Memory) و روی قسمت text. فایل Dll کلیک راست کنید. (این قسمت شامل کد اجرایی برنامه است) سپس روی گزینه Set Memory Breakpoint on Access کلیک کنید.

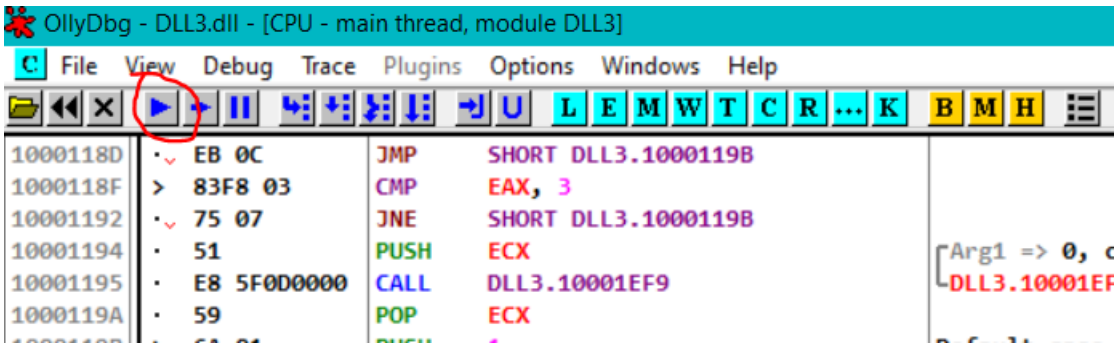
۲. در پایان دکمه F9 یا دکمه Play را بفشارید تا اجرای برنامه ادامه پیدا کند.

اجرای برنامه باید هنگامی‌که کد درون Dll اجرا می‌شود، متوقف گردد.

بارگذاری کتابخانه‌های پیوندی پویا

علاوه بر این‌که ما قادر هستیم فایل‌های اجرایی را به دیباگر OllyDBG به منظور دیباگ بارگذاری و پیوست کنیم، همچنین می‌توانید فایل‌های کتابخانه‌ای (Dll) را هم دیباگ کنید. از آنجایکه کتابخانه‌های پیوندی پویا نمی‌توانند به صورت مستقیم اجرا شوند، دیباگر OllyDBG از یک برنامه مصنوعی که loaddll.exe خوانده می‌شود برای بارگذاری آن‌ها استفاده می‌کند. این روش بسیار مفید است، زیرا بدافزارها اغلب به عنوان کتابخانه‌های پیوندی پویا انتشار پیدا می‌کنند که بیشتر کد آن‌ها در تابع DllMain قرار دارد (هنگامی‌که کتابخانه پیوندی پویا به درون پروسه بارگذاری می‌شود، این تابع راه‌انداز اولیه فراخوانی می‌شود). در حالت پیش‌فرض، هنگامی‌که کتابخانه پیوندی پویا بارگذاری می‌شود، دیباگر OllyDBG در نقطه ورود به کتابخانه پیوندی پویا (DllMain) متوقف می‌شود.

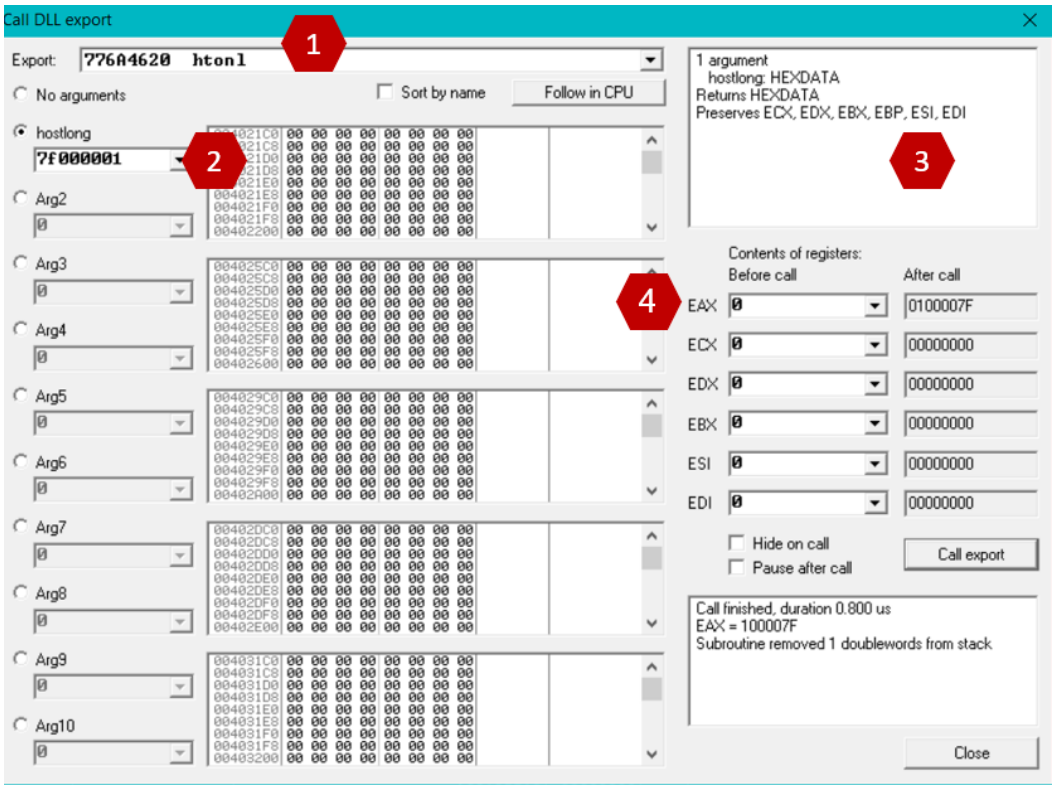
به منظور فراخوانی توابع اکسپورت شده با پارامترهایشان درون کتابخانه پیوندی پویا دیباگ شده، شما ابتدا نیاز دارید کتابخانه پیوندی پویا را به درون دیباگر OllyDBG بارگذاری کنید. سپس هنگامی‌که در نقطه ورود به کتابخانه پیوندی پویا متوقف شد، روی دکمه Play (تصویر ۱۱) کلیک کنید تا DllMain و دیگر راه اندازهای اولیه کتابخانه پیوندی پویا اجرا شوند.



تصویر ۱۱: دکمه Play دیباگر OllyDBG

سپس دیباگر OllyDBG متوقف خواهد شد و شما می‌توانید توابع اکسپورت شده مشخصی را با پارامترهایشان فراخوانی کرده و آن‌ها را با انتخاب گزینه Call Dll Export از منوی Debug در منوی اصلی دیباگر OllyDBG دیباگ کنید.

به عنوان مثال، در تصویر ۱۲ ما کتابخانه ws2_32.dll را درون دیباگر OllyDBG بارگذاری کرده و تابع ntohl را (شماره ۱) انتخاب کرده‌ایم که یک عدد ۳۲ بیتی شبکه (Network Byte Order) را به الگو عددی میزبان (Host Byte Order) تبدیل می‌کند. همانطور که در سمت راست پنجره Call Dll Export مشاهده می‌کنید (شماره ۳)، این تابع فقط یک پارامتر به عنوان ورودی دریافت می‌کند. به همین دلیل در سمت چپ، ما یک مقدار هگزادسیمال 0x7F000001 معادل 127.0.0.1 به hostlong اضافه می‌کنیم (شماره ۲). همچنین شایان ذکر است، پارامترهای سمت چپ تنها زمانی‌که تابع تعداد بیشتری پارامتر دریافت کند، فعال می‌شوند.



تصویر ۱۲: دیباگ و فراخوانی توابع اکسپورت شده توسط ws2_32.dll

شما می‌توانید به سرعت دستورالعمل‌های اسمبلی تابع ntohl را با کلیک روی دکمه Follow in Disassembler مشاهده کنید (در نسخه ۲ دیباگر OllyDBG نام این گزینه Follow in CPU است).

گزینه Hide on call در سمت راست دکمه Call می‌تواند برای مخفی کردن این پنجره پس از این‌که یک فراخوانی انجام می‌دهید، استفاده شود. کادر علامت Pause after call برای توقف سریع اجرای برنامه پس از این‌که تابع صادر شده فراخوانی شد، مورد استفاده قرار گیرد که می‌تواند یک جایگزین مناسب برای نقطه‌های توقف باشد.

هنگامی‌که پارامترها و ثبات‌ها را تنظیم کردید، روی دکمه Call کلیک کنید. سپس پنجره دیباگر OllyDBG باید مقادیر همه ثبات‌ها را قبل و بعد از فراخوانی نمایش دهد. برای دیباگ این تابع اکسپورت شده، مطمئن شوید هیچ نقطه توقفی قبل از کلیک روی Call تنظیم نکرده‌اید، یا کادر علامت Pause after call را تنظیم نکرده‌اید. در تصویر ۱۲ مشاهده می‌کنید که نتیجه اجرای تابع در ثبات EAX ذخیره شده است که مقدار آن 127.0.0.1 (0x0100007F) است (شماره ۴).

ردیابی[1]

ردیابی یک روش قدرتمند دیباگ به شمار می‌رود که جزییات اطلاعات اجرایی برای شما به منظور بررسی ذخیره می‌شود. همچنین شایان ذکر است، دیباگر OllyDBG از انواع ویژگی‌های ردیابی از جمله ردیابی بازگشتی استاندارد[2]، ردیابی فراخوانی پشته[3] و ردیابی اجرایی[4] پشتیبانی می‌کند.

ردیابی بازگشتی استاندارد

هر زمان که در پنجره دیزاسمبلر دیباگر OllyDBG در بین دستورالعمل‌ها با گام برداشتن به داخل (Step Into) یا گام برداشتن از روی (Step Over) حرکت می‌کنید، دیباگر OllyDBG به صورت خودکار گام‌های برداشته شده شما را ضبط می‌کند. با این حال شما می‌توانید با فشردن دکمه (–) در صفحه کلید به مرحله قبل بازگردید و دستورالعملی را که در مرحله قبل اجرا شده است، مشاهده کنید. همچنین کلید (+) در صفحه کلید شما را یک گام به جلو حرکت می‌دهد. اگر از گزینه گام برداشتن به داخل (Step Into) استفاده

[1] Tracing
[2] Standard Back Trace
[3] Call Stack Trace
[4] Run Trace

می‌کنید، می‌توانید هر یک از گام‌ها را ردیابی کنید. اگر از گام برداشتن از روی (Step Over) استفاده کنید، می‌توانید فقط تنها به ناحیه‌ای که به آن گام برداشتید بازگردید.

فراخوانی پشته

شما می‌توانید در دیباگر OllyDBG از طریق ردیابی فراخوانی پشته، مسیر اجرایی تابع را مشاهده کنید. برای مشاهده پشته فراخوانی‌ها، از منوی اصلی دیباگر گزینه View سپس Call Stack را انتخاب کنید. پس از انتخاب این گزینه پنجره‌ای را مشاهده خواهید کرد که سلسله مراتبی از فراخوانی‌های انجام شده که شما را به مکان جاریتان رسانده است، نمایش می‌دهد.

ردیابی اجرایی

ردیابی اجرا به شما اجازه می‌دهد کد برنامه مد نظرتان را اجرا کنید و دیباگر OllyDBG هر دستورالعملی را که اجرا می‌شود به همراه تغییراتی که در ثبات‌ها و پرچم‌ها اعمال می‌شود را ذخیره‌سازی کند. راه‌های مختلفی برای فعال ساختن ردیابی اجرایی وجود دارد، در زیر دو روش را مورد بررسی قرار می‌دهیم:

- ✓ کدی که قصد دارید ردیابی اجرایی کنید را در پنجره دیزاسمبلر برجسته یا انتخاب کنید و سپس روی آن کلیک راست کرده و از منوی Run Trace گزینه Add Selection را انتخاب کنید. سپس بعد از اجرا شدن آن قطعه کد، از منوی View گزینه Run Trace را برای مشاهده دستورالعمل‌هایی که اجرا شده‌اند، انتخاب کنید. همچنین می‌توانید در این پنجره با استفاده از دکمه‌های (+) و (–) ما بین دستورالعمل‌ها بالا و پایین بروید. با این روش می‌توانید حتی تغییراتی که دستورالعمل‌ها روی ثبات‌ها اعمال کرده‌اند را هم مشاهده کنید.
- ✓ همچنین در روشی دیگر می‌توانید از گزینه‌های Trace Into و Trace Over استفاده کنید. استفاده از این گزینه‌ها نسبت به استفاده از Add Selection ساده‌تر است، زیرا شما نیاز ندارید کدی که می‌خواهید ردیابی کنید را انتخاب کنید. Trace Into گام برداشتن به داخل (Step Into) خواهد کرد و تمامی دستورالعمل‌ها را تا زمانی‌که با یک نقطه توقف برخورد کند ضبط می‌کند اما Trace Over تنها دستورالعمل‌هایی را ظبط می‌کند که در تابع جاری شما اجرا شده‌اند.

نکته : توجه داشته باشید، اگر در هنگام استفاده از گزینه‌های Trace Into و Trace Over در کد نقطه توقفی قرار ندهید، دیباگر OllyDBG به صورت خودکار کل برنامه را دیباگ خواهد کرد، که این عمل موجب می‌شود زمان و حافظه بسیار زیادی صرف شود.

- ✓ در روش بعد، از منوی Debug گزینه Set Condition را انتخاب کنید. شما می‌توانید با این گزینه یک برنامه را تا زمانی‌که یک شرط خاص برقرار شود، مورد ردیابی قرار بدهید. این گزینه زمانی مفید است که شخص تحلیلگر می‌خواهد برنامه را تا زمانی‌که یک شرط برقرار نیست مورد ردیابی قرار بدهد. برقرار شدن شرط موجب می‌شود برنامه موقتا متوقف شود و با استناد به این موضوع سپس می‌توانید با بازگشت به دستورالعمل‌های اجرا شده قبلی (با روش ردیابی بازگشتی) دلیل برقراری آن شرط را مورد بررسی قرار بدهید. در ادامه این قسمت شما یک مثال از این مبحث مشاهده خواهید کرد.

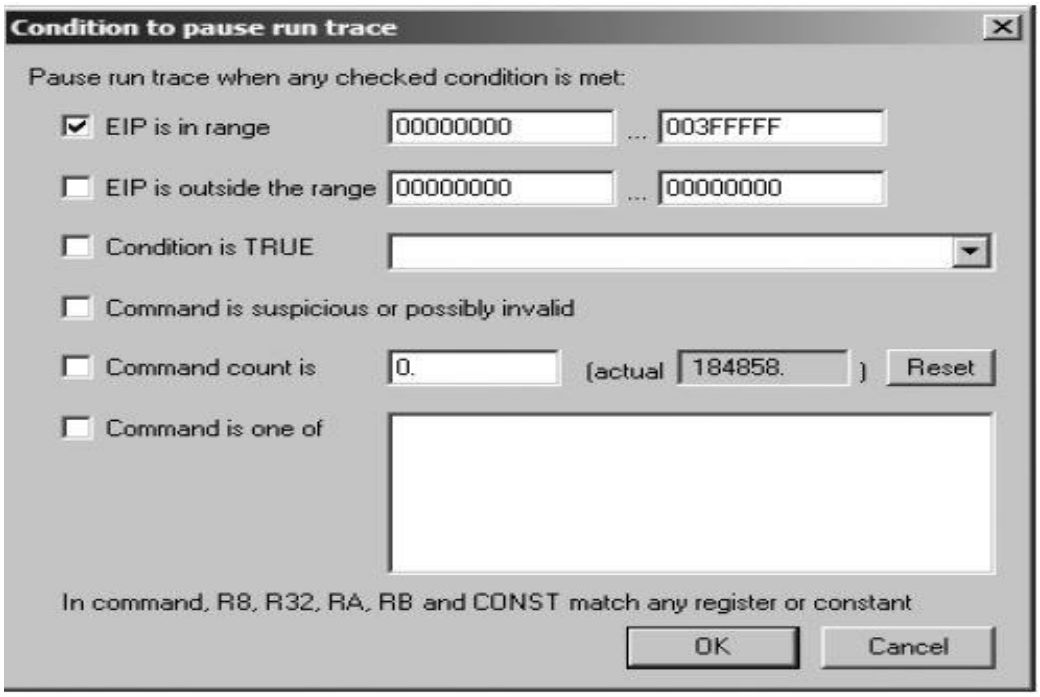
ردیابی Poison Ivy

از بحثی که به تازگی کردیم به یاد دارید که درب‌پشتی Poison Ivy اغلب به شلکدی که از سرور کنترل و فرماندهی خود دریافت می‌کرد، حافظه‌ای اختصاص می‌داد. درب‌پشتی Poison Ivy شلکد را دانلود می‌کرد و آن را در مکان حافظه پویا کپی کرده و در پایان آن را اجرا می‌کرد. در برخی حالات، شما می‌توانید با استفاده از ردیابی، هنگامی‌که ثبات EIP در حافظه Heap می‌باشد شلکد اجرایی را به دست آورید. ردیابی می‌تواند به شما نشان دهد که چگونه شلکد شروع به کار می‌کند.

نکته: به این نکته دقت کنید، منظور از حافظه پویا در متن بالا همان حافظه هیپ در رایانه است. ما در مفاهیم سامانه‌ای، مطلبی داریم که به آن Dynamic memory allocation یا تخصیص حافظه پویا می‌گویند، این مکانیزم معمولا به عنوان اسمی به نام هیپ شناخته می‌شود. اولین نکته‌ای که درباره هیپ باید

بدانید این است که هیپ خود یک حافظه است. حافظه‌ای که هنگام اجرای یک برنامه به آن اختصاص می‌یابد.

تصویر ۱۳ شرطی را که برای گرفتن شلکد اجرایی اعمال کرده‌ایم، نمایش می‌دهد. شایان ذکر است، ما می‌توانیم دیباگر OllyDBG را به‌گونه‌ای تنظیم کنیم که اگر ثبات EIP از محدوده آدرس برنامه خارج شد، اجرای برنامه را موقتا متوقف کند. (در برنامه‌های ساده زیر آدرس 0x400000 معمولا پشته، هیپ و دیگر حافظه‌های اختصاص داده شده پویا قرار دارند). آدرس درون ثبات EIP نباید در یک برنامه عادی در این ناحیه‌ها قرار گیرد. سپس Trace Into را انتخاب کرده و تمام برنامه را ردیابی کنید تا شلکد اجرا شود. در این قسمت، برنامه هنگامی‌که ثبات EIP با 0x142A88 تنظیم شود، متوقف می‌گردد، چون این آدرس ابتدای شلکد است. سپس ما می‌توانیم به سادگی از دستورات (+) و (–) برای مرور دستورات شلکد استفاده کنیم.



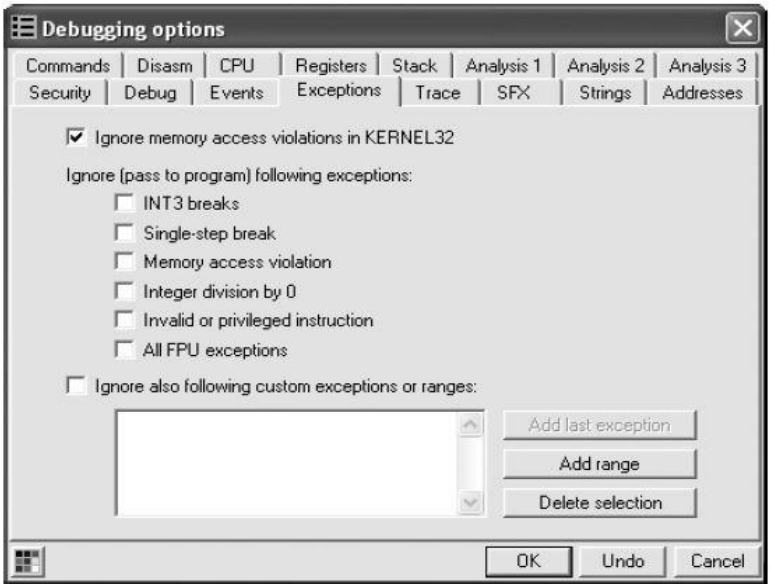
تصویر ۱۳: ردیابی شرطی

در حالت پیش‌فرض، هنگامی‌که دیباگر OllyDBG به یک پروسه پیوست شده است و یک استثناء رخ می‌دهد، اجرای برنامه متوقف می‌شود و دیباگر کنترل آن را به دست می‌گیرد. دیباگر می‌تواند استثناء را کنترل

کند یا آن را به برنامه عبور دهد. دیباگر OllyDBG اجرای برنامه را زمانی‌که استثنا رخ دهد متوقف می‌کند و شما می‌توانید تصمیم بگیرید که با کدام یک از روش‌های زیر آن را به برنامه عبور دهید.

- ✓ SHIFT-F7 به درون استثناء گام برخواهد (Step Into) داشت.
- ✓ SHIFT-F8 از روی استثناء گام بر خواهد (Step Over) داشت.
- ✓ SHIFT-F9 کنترل کننده استثناء را اجرا خواهد کرد.

دیباگر OllyDBG گزینه‌هایی برای کنترل استثناها دارد که در تصویر ۱۴ نمایش داده شده است. این گزینه‌ها می‌توانند به دیباگر بگویند برخی از استثناها را نادیده بگیرد و مستقیما آن‌ها را به برنامه عبور دهد. (این گزینه اغلب در حین تجزیه و تحلیل بدافزار مفید است، زیرا می‌خواهیم بدافزار را تحلیل کنیم نه این‌که مشکلات آن را رفع کنیم.)



تصویر ۱۴: گزینه‌های کنترل استثناء در OllyDBG

مهار یا اصلاح کردن[1]

دیباگر OllyDBG ایجاد تغییر در هر داده فعالی مانند ثبات‌ها و پرچم‌ها را تا حد امکان ساده کرده است. این دیباگر همچنین شما را قادر می‌سازد مستقیما یک برنامه را اسمبل و اصلاح کنید. در دیباگر OllyDBG

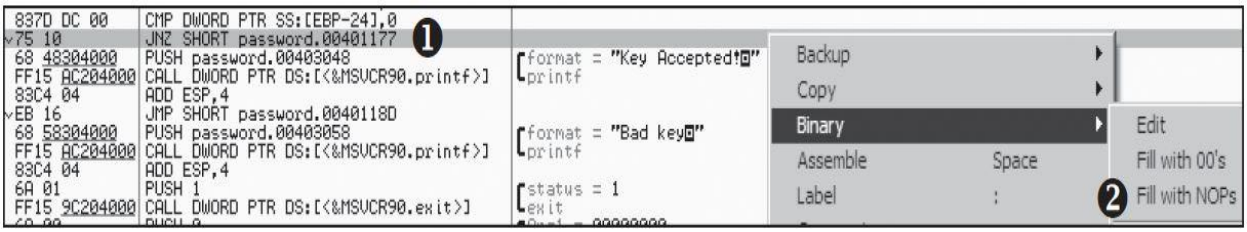
[1] Patching

می‌توانید ناحیه‌ای از دستورالعمل‌ها را انتخاب کنید و با کلیک راست روی آن محدوده و انتخاب گزینه Edit از منوی Binary در دستورالعمل‌ها و حافظه تغییر ایجاد کنید.

پس از کلیک روی Edit پنجره‌ای باز خواهد شد که در آن می‌توانید هر گونه Opcode یا داده‌ای را وارد کنید. (دیباگر OllyDBG همچنین توابع خاصی برای پر کردن محدوده‌ای از حافظه را با مقدار ۰۰ یا دستورالعمل NOP دارد.) تصویر ۱۵ یک قسمت از کد بدافزار که با یک کلمه عبور محافظت شده است را نمایش می‌دهد، بدافزار باید یک کلمه عبور از ورودی بگیرد تا اجرا شود.

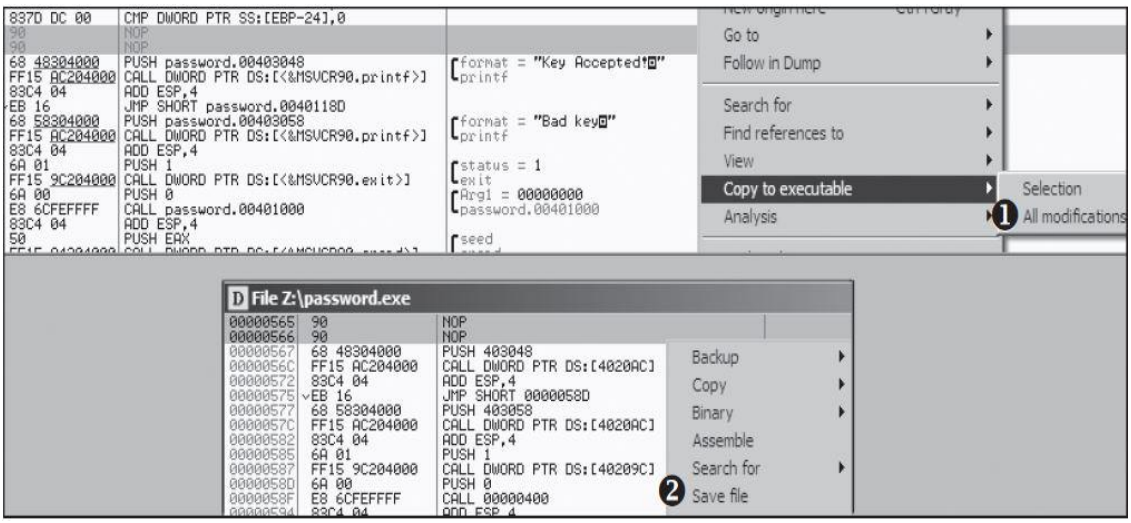
در این برنامه یک پرش شرطی مهم (JNZ) در (شماره ۱) وجود دارد که تصمیم می‌گیرد کلید وارد شده به برنامه درست است یا خیر. اگر دستورالعمل پرش اجرا شود رشته Bad Key در خروجی چاپ می‌شود و اگر اجرا نشود پیام Key Accepted در خروجی چاپ خواهد شد.

یک راه ساده برای این‌که برنامه را مجبور کنیم که به مسیر Key Accepted برود این است که یک اصلاحیه روی آن اعمال کنیم. همان‌طور که در تصویر ۱۵ نمایش داده شده است، روی دستورالعمل شرطی که مشخص شده است کلیک راست کرده و از منوی Binary گزینه Fill with Nops را انتخاب (شماره ۲) کنید. این گزینه دستورالعمل JNZ را به دستورالعمل‌ها NOP تغییر می‌دهد و برنامه پس از آن فکر می‌کند هر کلیدی که وارد می‌شود درست است و پیام Key Accepted را در خروجی چاپ می‌کند.



تصویر ۱۵: گزینه‌های اصلاح‌کننده در OllyDBG

توجه کنید این اصلاحیه فقط روی این مثال کار می‌کند و اگر برنامه را راه‌اندازی مجدد یا Restart کنید این اصلاحیه از بین خواهد رفت. لذا برای این‌که بتوانید فایل اصلاح شده خود را ذخیره کنید دو گام آورده شده در تصویر ۱۶ را انجام بدهید.



تصویر ۱۶: دو گام اعمال وصله روی فایل و ذخیره‌سازی آن روی دیسک

برای اعمال این اصلاحیه، روی قسمتی که اصلاحیه را اعمال کردید کلیک راست کرده و از منوی Copy To Executable گزینه All Modifications (شماره ۱) را انتخاب کنید. این گزینه تمامی تغییراتی که شما روی حافظه فعال اعمال کرده‌اید را ذخیره ساخته و یک پنجره جدید باز می‌کند. در پنجره جدیدی که باز می‌شود کلیک راست کنید و از گزینه Save File (شماره ۲) را انتخاب کنید و فایل را روی دیسک ذخیره کنید.

نکته : به این نکته توجه کنید، کد موجود در تصویر ۱۶ مشابه کدی است که در تصویر ۱۵ آمده است. با این تفاوت‌که بجای دستورالعمل JNZ برای همیشه دستورالعمل NOP که مخفف No Operation است، قرار می‌گیرد و در پایان فایل اصلاح شده در سامانه ذخیره می‌شود. شایان ذکر است با قرار دادن دستورالعمل NOP به جای JNZ عملا دستور بررسی کننده (شرطی) حذف می‌شود و دیگر هر رشته‌ای به برنامه وارد کنید، برنامه آن رشته را درست فرض کرده و خروجی اصلی را به شما ارائه می‌دهد.

تجزیه و تحلیل شلکد[1]

دیباگر OllyDBG یک راه ساده برای تحلیل شلکد ارائه می‌دهد. برای استفاده از این روش، گام‌های زیر را دنبال کنید.

- ✓ شلکد را از یک ویرایشگر کدهای هگزادسیمال به clipboard کپی کنید.
- ✓ درون پنجره Memory map یک ناحیه از حافظه که نوع آن Priv باشد، انتخاب کنید. (این حافظه خصوصی است که به یک پروسه اختصاص داده می‌شود، برخلاف فایل‌های اجرایی فقط خواندنی که میان چندین پروسه اشتراک گذاشته می‌شوند.)
- ✓ روی سطر آن دو بار کلیک کنید تا پنجره ویرایشگر Hex باز شود. بعد از باز شدن آن پنجره می‌توانید محتویات آن را بررسی کنید. این ناحیه باید با چند صد بایت ۰۰ مقداردهی شده باشند.
- ✓ بر روی ناحیه انتخابی خود در پنجره Memory Map کلیک راست کرده و سپس از منوی Set Access گزینه Full Access را انتخاب کنید تا به آن ناحیه سطح دسترسی read، write و execute ارائه شود.
- ✓ سپس به پنجره Memory dump بازگردید. یک ناحیه بزرگ که با صفر کامل شده‌اند، انتخاب کرده و روی آن ناحیه انتخابی کلیک راست کنید و از منوی Binary گزینه Binary Paste را انتخاب کنید. پس از انتخاب این گزینه شلکد در آن ناحیه انتخابی قرار خواهد گرفت.
- ✓ ثبات EIP را با محل حافظه که آن را تغییر داده‌اید به‌روزرسانی کنید. (شما می‌توانید به راحتی با کلیک راست روی یک دستورالعمل و انتخاب گزینه New Origin Here ثبات EIP را با آن دستورالعمل مقداردهی کنید.)

حال می‌توانید شلکد را مانند یک برنامه عادی اجرا و دیباگ کنید.

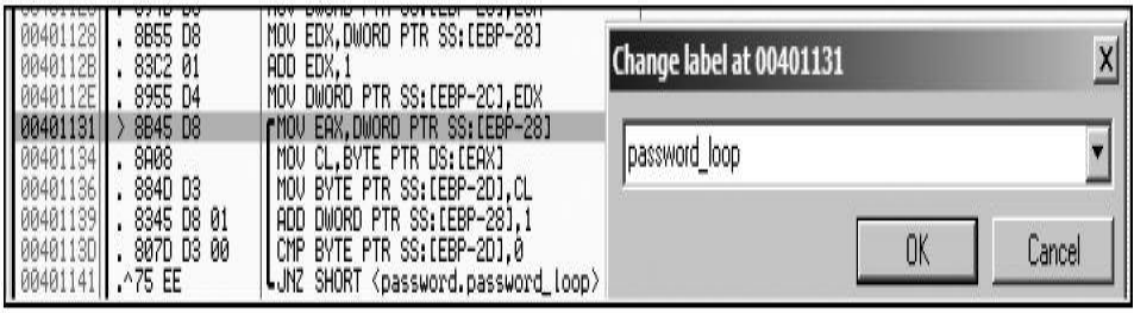
[1] Analyzing Shellcode

ویژگی‌های کمکی[1]

دیباگر OllyDBG مکانیزم‌های بسیاری برای کمک به تحلیل ارائه می‌دهد که برخی از آن‌ها در اینجا تشریح شده‌اند.

- ✓ **گزارش یا Logging :** دیباگر OllyDBG همواره گزارشی از رویدادهای رخ داده برای شما در دسترس نگه می‌دارد. برای دسترسی به آنها می‌توانید از منوی View گزینه Log را انخاب کنید. این گزینه گزارش ماژول‌های اجرایی که به برنامه بارگذاری شده‌اند، نقاط توقفی که اجرا شده‌اند و دیگر اطلاعات مفید را نمایش می‌دهد. همچنین گزینه Log می‌تواند در حین تجزیه و تحلیل برای کشف برخی از گام‌هایی که شما برای انجام تحلیل برداشته‌اید، بسیار مفید واقع شود.
- ✓ **پنجره مشاهدات یا Watches Window:** دیباگر OllyDBG به کمک پنجره Watches به شما اجازه می‌دهد مقادیر یک عبارت را که تولید کرده‌اید، مشاهده کنید. این عبارت به طور مداوم در این پنجره به روز می‌شود. شما می‌توانید از منوی View گزینه Watches به این ویژگی دسترسی بگیرید.
- ✓ **برچسب‌گذاری یا Labeling :** همانند IDA Pro در برنامه OllyDBG می‌توانید حلقه‌ها و سابروتین‌ها را برچسب‌گذاری کنید. یک برچسب نام سمبولیک ساده‌ای است که به یک آدرس از برنامه دیباگ شده اختصاص داده می‌شود. برای قرار دادن یک برچسب در پنجره دیزاسمبلر، روی یک آدرس کلیک راست کرده و گزینه Label را انتخاب کنید. با انتخاب این گزینه پنجره‌ای نمایش داده خواهد شد که در آن می‌توانید برچسب مد نظر خود را وارد کنید. تمامی مراجعات به این مکان از حافظه از این برچسب بجای آدرس استفاده خواهند کرد. تصویر ۹–۱۵ یک مثال از اضافه کردن برچسب password_loop به یک آدرس را نمایش می‌دهد.

[1] Assistance Features



تصویر ۱۷: گذاشتن یک برچسب در **OllyDBG**

پلاگین‌های OllyDBG

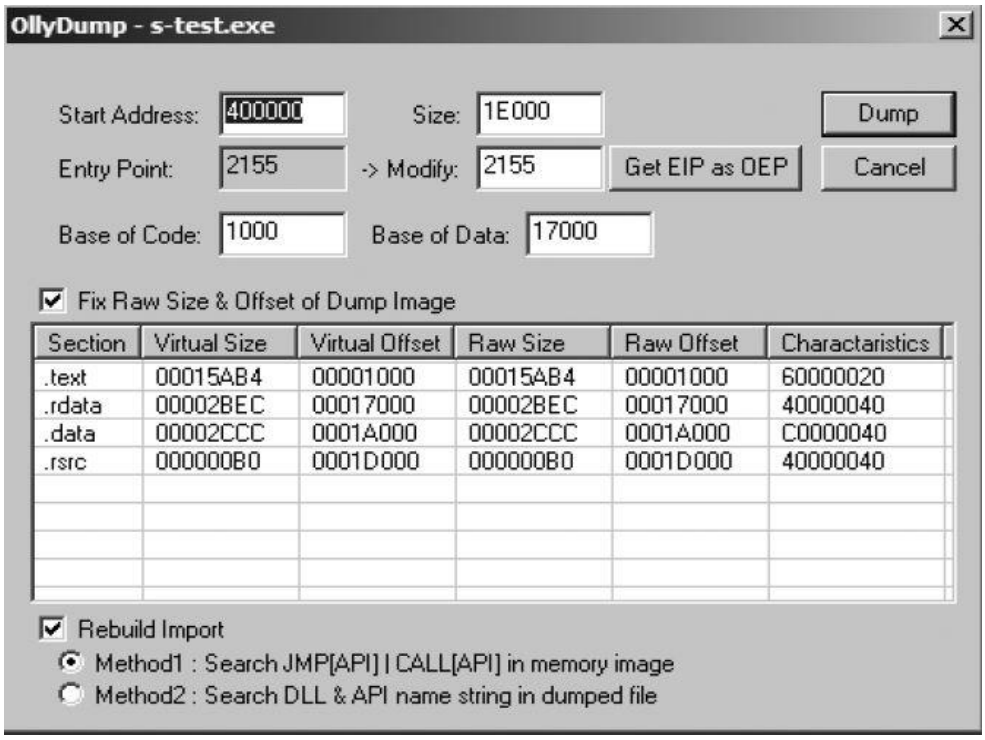
دیباگر OllyDBG پلاگین‌های بسیاری دارد که برخی از آن‌ها استاندارد هستند و در آن به صورت پیش‌فرض وجود دارند، اما برخی دیگر از آن‌ها که توسط اشخاص گوناگون نوشته می‌شوند را می‌توان از روی محیط اینترنت دانلود کرد. با این حال، می‌توانید یک مجموعه از بهترین پلاگین‌های OllyDBG که برای تحلیل بدافزارها می‌توانند بسیار مفید واقع شوند را در آدرس (http://goo.gl/a2CkVn) پیدا کنید.

نکته: نوشتن پلاگین در OllyDBG می‌تواند یک پروسه بسیار خسته کننده باشد. با این حال اگر علاقمند هستید که ویژگی‌های دیباگر OllyDBG را توسعه دهید، پیشنهاد می‌کنیم اسکریپت را با پایتون بنویسید، در قسمت (توسعه دیباگر با قابلیت اسکریپت‌نویسی) این موضوع را تشریح خواهیم کرد.

پلاگین OllyDump

پلاگین OllyDump در دیباگر OllyDBG بطور رایجی استفاده می‌شود، زیرا اجازه می‌دهد از یک پروسه در حال دیباگ، در قالب یک یک فایل PE نمونه‌برداری کنیم (آن را ذخیره کنیم). پلاگین OllyDump تلاش می‌کند پروسه‌ای که لودر سیستم‌عامل انجام می‌دهد، معکوس کند. به هر حال، این پلاگین از حالات جاری بخش‌های مختلف (code، data و غیره) که در حافظه وجود دارند، استفاده می‌کند. (این پلاگین معمولا برای پروسه Unpacking استفاده می‌شود که در فصل‌های بعدی در این مورد بحث خواهیم کرد.) تصویر ۱۸ پنجره OllyDump را نمایش می‌دهد. هنگام نمونه‌برداری، می‌توانید به صورت دستی نقطه ورود

و آفست‌های بخش‌ها را تنظیم کنید، گرچه ما پیشنهاد می‌کنیم بگذارید خود دیباگر OllyDBG به صورت خودکار این کار را انجام بدهد.



تصویر ۱۸: پنجره پلاگین **OllyDump**

پلاگین Hide Debugger

پلاگین Hide Debugger تعدادی روش برای مخفی ساختن OllyDBG از شناسایی دیباگرهای مورد استفاده قرار می‌دهد. بیشتر تجزیه و تحلیل کنندگان بدافزار این پلاگین را بارها اجرا می‌کنند مخصوصا زمانی‌که بدافزار از روش‌های ضد دیباگ استفاده می‌کند. این پلاگین شما را در برابر بررسی IsDebuggerPresent، بررسی FindWindow، ترفندهای استثناهای کنترل نشده و اکسپلویت OuputDebugString از شما در دیباگر OllyDBG محافظت می‌کند (در مورد روش‌های ضد دیباگ در فصل هفدهم بحث خواهیم کرد.).

پلاگین خط فرمان

پلاگین خط فرمان اجازه می‌دهد یک دسترسی خط فرمان به دیباگر OllyDBG داشته باشید. همانند دیباگر WinDBG که یک دسترسی خط فرمان به شخص دیباگ کننده ارائه می‌دهد. با این حال بیشتر کاربران

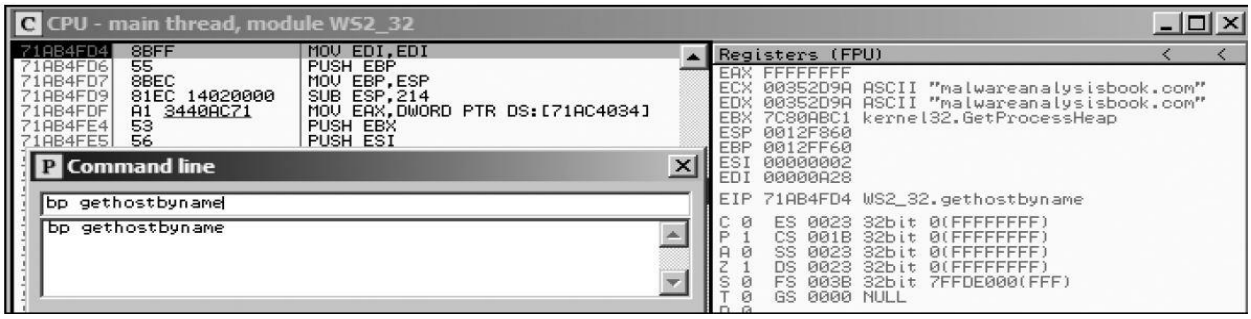
دیباگر OllyDBG از این مزئیت استفاده نمی‌کنند. (دیباگر WinDBG در فصل بعد مورد بررسی قرار خواهد گرفت.) برای فعال ساختن پنجره Command Line از منوی Plugins سپس Command Line گزینه Command Line را انتخاب کنید. جدول ۹-۳ لیست فرامین رایج مورد استفاده در این پلاگین را نمایش می‌دهد. علاوه بر این فرامین، فرمان‌های دیگری وجود دارند که می‌توانید در فایل راهنمای این پلاگین آن‌ها را به دست آورید.

جدول ۳-۹: فرمان‌هایی برای استفاده در خط فرمان دیباگر OllyDBG

Command	Function
BP *expression* [,*condition*]	Set software breakpoint
BC *expression*	Remove breakpoint
HW *expression*	Set hardware breakpoint on execution
BPX *label*	Set breakpoint on each call to *label*
STOP or PAUSE	Pause execution
RUN	Run program
G [*expression*]	Run until address
S	Step into
SO	Step over
D *expression*	Dump memory

هنگام دیباگ یک برنامه، در خیلی از مواقع می‌خواهید هنگامی‌که اجرای برنامه به تابع خاصی رسید به صورت خودکار متوقف شود تا شما بتوانید پارامترهای ورودی به آن تابع را بررسی کنید. در این موقعیت، می‌توانید با استفاده از پلاگین خط فرمان OllyDBG به سادگی یک نقطه توقف در ابتدای تابع مد نظر خودتان تنظیم کنید تا برنامه پس از رسیدن به آن نقطه به صورت خودکار متوقف شود.

در مثال آورده شده در تصویر ۱۷-۹، ما یک قطعه کد از بدافزار با رشته‌های مبهم‌سازی شده داریم که به آن تابع gethostbyname وارد شده است. همان‌طور که در تصویر نمایش داده شده است، فرمان bp gethostbyname را در قسمت خط فرمان دیباگر اجرا کرده‌ایم که یک نقطه توقف در ابتدای آن تابع قرار بدهد. پس از متوقف شدن برنامه می‌توانید به پارامترهای آن نگاه کنید، در این مثال ما نام میزبان را مشاهده می‌کنیم که قصد دارد resolve شود. (در این مثال نام میزبان malwareanalysisbook.com است.)



تصویر ۱۹: استفاده از خط فرمان برای قرار دادن یک نقطه توقف سریع

نشانه‌گذاری[1]

پلاگین نشانه‌گذاری یا به اصطلاح Bookmark به صورت پیش‌فرض در OllyDBG وجود دارد. این پلاگین شما را قادر می‌سازد روی محل‌های حافظه نشانه‌گذاری کنید. با انجام این کار می‌توانید به سادگی به محل‌های مختلف حافظه بدون آنکه آدرس‌های آن را حفظ کنید، دسترسی بگیرید. برای افزودن یک نشانه به یک آدرس از حافظه کافی است در پنجره دیزاسمبلر روی آدرس مد نظر خود راست کلیک کرده و منوی Bookmark گزینه Insert Bookmark را انتخاب کنید. پس از انتخاب این گزینه آدرس مورد نظر شما نشانه‌گذاری می‌شود و شما می‌توانید با رفتن به منوی Plugins و سپس Bookmarks و انتخاب گزینه Bookmarks نشانه‌هایی که روی حافظه گذاشته‌اید را مشاهده کنید.

نکته: در این قسمت کلمه Bookmark را به نشانه‌گذاری برگردانده‌ایم که با اصل معنی آن بسیار متفاوت است. معنی کلمه Bookmark در فرهنگ لغت به معنای بین کتاب ترجمه شده است، که برگرداندن این جمله لاتین به همچین معنی می‌تواند بسیار محتوای ترجمه شده را گنگ و نامفهوم کند. لذا برای این‌که بتوانیم کاربرد این ویژگی را به شخص خواننده به درستی برسانیم از واژه نشانه‌گذاری استفاده کرده‌ایم.

---

[1] Bookmark

توسعه دیباگر با قابلیت اسکریپت‌نویسی

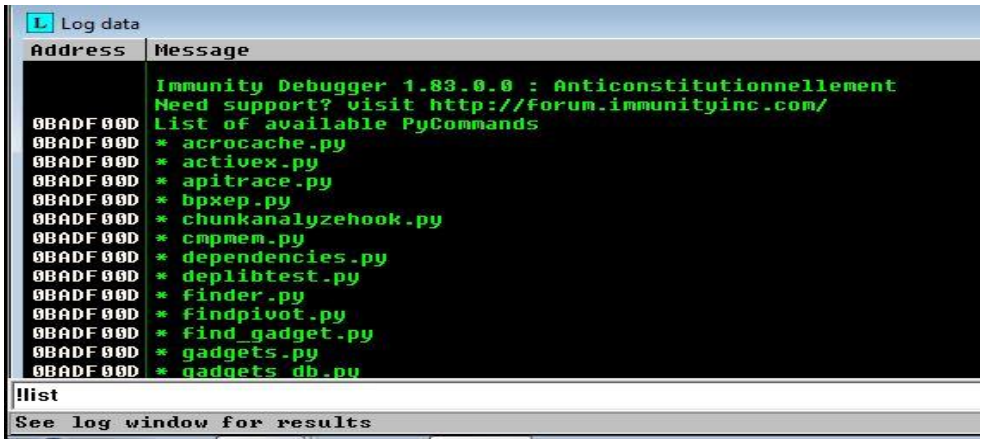
از آنجایکه پلاگین‌های دیباگر OllyDBG به کتابخانه‌های پیوندی پویا کامپایل شده‌اند، ایجاد و یا اصلاح یک پلاگین به یک پروسه پیچیده تبدیل شده است. بنابراین، هنگام توسعه ویژگی‌های دیباگر، ما ImmDBG را به خدمت می‌گیرم که از پایتون استفاده می‌کند و یک مجموعه API ساده برای توسعه مقاصدمان به ما ارائه می‌دهد. رابطهای برنامه‌نویسی کاربردی[1] پایتون ImmDBG شامل بسیاری از امکانات و توابع مفیدی برای ما می‌شود. به عنوان مثال، شما می‌توانید اسکریپت خود را به منظور ایجاد جدول‌های سفارشی، گراف‌ها و رابطها در دیباگر OllyDBG توسعه دهید.

با این حال، دلایل مهم نوشتن اسکریپت برای تجزیه و تحلیل بدافزار شامل مهار روش‌های ضد دیباگ، هوک خطی توابع و مشاهده پارامترهای ورودی به تابع است. دلایل بیشتر را می‌توانید در اینترنت پیدا کنید. رایج‌ترین نوع اسکریپت نوشته شده با پایتون برای ImmDBG با نام pyCommands شناخته می‌شود. این ابزار، یک نوع اسکریپت با زبان پایتون است که در محلی که ImmDBG نصب شده است در پوشه pyCommands\ قرار دارد. پس از این‌که شما یک اسکریپت نوشتید، باید آن را در این پوشه کپی کنید و از قسمت Command bar به همراه دستورالعمل (!) آن را اجرا کنید. برای مشاهده یک لیست موجود pyCommands دستورالعمل !list را وارد کنید و enter را بفشارید. در تصویر آورده شده در زیر خروجی این دستورالعمل نمایش داده شده است.

نکته: در این قسمت منظور ما از ImmDBG همان دیباگر Immunity است که در فصول گذشته در مورد آن بحث کردیم و گفتیم که این دیباگر نسخه بهبود یافته OllyDBG است که دارای ساختار پیشرفته و ظاهری کاربر پسند است.

---

[1] API



تصویر ۲۰: اسکریپت‌های پایتون ImmDBG

اسکریپت‌های pyCommands شامل ساختار زیر می‌شود. یعنی شما برای نگارش یک اسکریپت به زبان پایتون که بتوان آن را در دیباگر ImmDBG مورد استفاده قرار داد باید از ساختار زیر تبعیت کنید.

- ✓ در ابتدای کد اسکریپت (همانند دیگر اسکریپت‌های پایتون) می‌توانید با استفاده از دستور import یک تعداد ماژول پایتون را به اسکریپت خود وارد کنید، همچنین برای دسترسی به ویژگی‌ها و قابلیت‌های ارائه شده دیباگر ImmDBGماژول‌های immlib یا immutils را می‌توانید به اسکریپت خود وارد کنید.
- ✓ همچنین این اسکریپت همانند دیگر کدها دارای یک تابع اصلی (Main Function) است که پارامترهای خط فرمان را می‌خواند و آن‌ها را به اسکریپت عبور می‌دهد.
- ✓ در پایان کد هر تابع اسکریپت یک دستورالعمل return قرار می‌گیرد که یک پیام را به خروجی ارسال می‌کند. زمانی‌که هر تابع اسکریپت عملیاتی خود را به اتمام می‌رساند، نوار وضعیت دیباگر اصلی با آن رشته به روز رسانی می‌شود.

کد موجود در لیست ۳ یک اسکریپت ساده PyCommand را نمایش می‌دهد. از این اسکریپت می‌توان برای جلوگیری از پاک شدن فایلی توسط بدافزار از روی سامانه استفاده کرد.

```
import immlib

def Patch_DeleteFileA(imm): ❷
    delfileAddress = imm.getAddress("kernel32.DeleteFileA")
    if (delfileAddress <= 0):
        imm.log("No DeleteFile to patch")
        return
    imm.log("Patching DeleteFileA")
    patch = imm.assemble("XOR EAX, EAX \n Ret 4") ❸
    imm.writeMemory(delfileAddress, patch)

def main(args): ❶
    imm = immlib.Debugger()
    Patch_DeleteFileA(imm)
    return "DeleteFileA is patched..."
```

لیست ۳: اسکریپت **PyCommand** برای جلوگیری از پاک شدن فایل

بدافزارها اغلب برای حذف فایل‌ها از روی سامانه تابع DeleteFile را فراخوانی می‌کنند. بنابراین اگر شما این اسکریپت را با نام myscript.py در مسیر PyCommand ذخیره کنید و با دستور !myscript آن را در دیباگر OllyDBG اجرا کنید، اسکریپت تابع DeleteFileA را مهار خواهد کرد و آن را غیرقابل استفاده می‌کند. متد اصلی تعریف شده در (شماره ۱) تابع Patch_DeleteFileA را فراخوانی می‌کند. این تابع در (شماره ۲) تعریف شده است که آدرس تابع DeleteFileA را با فراخوانی تابع getAddress بازگشت می‌دهد. هنگامی‌که ما آدرس آن محل را داشته باشیم، می‌توانیم تابع را با آدرس خود بازنویسی کنیم. با این حال، ما آن را با کد آورده شده در (شماره ۳) مهار می‌کنیم. این کد ثبات EAX را با ۰ تنظیم می‌کند و از فراخوانی DeleteFile باز می‌گردد. این وصله موجب می‌شود تابع DeleteFile همیشه در پاک کردن فایل‌ها از روی سامانه با شکست رو به رو شود. در نتیجه باعث می‌شود بدافزار نتواند فایلی را از سامانه حذف کند.

نکته : برای اطلاعات بیشتر درباره نوشتن اسکریپت‌های پایتون می‌توانید کتاب Gray Hat Python نوشته Justin Seitz را مورد مطالعه قرار بدهید.

نتیجه گیری

دیباگر OllyDBG معروفترین دیباگر در سطح کاربر برای تحلیل بدافزار می‌باشد. این دیباگر دارای ویژگی‌های بسیاری است که در انجام تجزیه و تحلیل دینامیک به شخص تحلیلگر کمک‌های شایانی می‌کند. همان‌طور

هم که در این فصل مشاهده کردید، دیباگر OllyDBG دارای یک رابط کاربری بسیار غنی است که در آن اطلاعات بسیاری درباره بدافزار به تحلیلگر ارائه می‌دهد.

به عنوان مثال، پنجره Memory Map یک راه عالی برای مشاهده چگونه قرار گیری برنامه در حافظه و مشاهده تمامی بخش‌های بدافزار به شمار می‌رود. همچنین انواع مختلف نقاط توقف موجود در دیباگر OllyDBG، از جمله نقاط توقف شرطی که می‌توانند برای متوقف کردن برنامه هنگامی‌که به یک ناحیه خاصی از حافظه دسترسی می‌گیرد یا پارامتر خاصی از تابع را فراخوانی می‌کند، بسیار مفید واقع شوند.

دیباگر OllyDBG می‌تواند دستورالعمل‌های یک فایل باینری در حال اجرا را به منظور نشان دادن وضعیتی که در حالت معمولی ممکن است، رخ ندهد اصلاح کند. همچنین شما می‌توانید تمامی تغییراتی اعمالی خودتان روی فایل باینری را در قالب یک فایل اصلاح شده جدید روی دیسک سخت ذخیره کنید. همچنین می‌توانید با استفاده از نوشتن اسکریپت‌ها و پلاگین‌ها ویژگی‌های دیباگر را توسعه دهید که همین به خودی خود ویژگی بسیار قدرتمندی است.

با این‌که دیباگر OllyDBG مشهورترین دیباگر سطح کاربر است، اما ما در قسمت بعدی به یکی دیگر از دیباگرهای مشهور سطح کرنل که WinDBG نام دارد، خواهیم پرداخت. از آنجایی که دیباگر OllyDBG نمی‌تواند بدافزارهای در حالت کرنل از قبیل روت‌کیت‌ها و درایورهای راه‌انداز را دیباگ کند، شما برای تجزیه و تحلیل دینامیک آن‌ها باید با دیباگر WinDBG آشنا شوید.

آزمایشگاه ششم

آزمایش ۹-۱

فایل Lab 09-01.exe را با استفاده از OllyDBG و IDA Pro به منظور پاسخ به سوال‌های آورده شده در زیر تحلیل کنید. این بدافزار در فصل سه با روش تجزیه و تحلیل استاتیک ساده و پویا تحلیل شده بود.

سوالات :

۱. چه کاری باید انجام دهیم تا بدافزار خودش را نصب کند؟
۲. گزینه‌های خط فرمان این برنامه چیست و چه کلمه عبوری نیاز دارد؟
۳. چه کاری می‌توان برای اصلاح این بدافزار در OllyDBG انجام داد، به طوری که نیاز به هیچ کلمه عبوری نداشته باشد.
۴. نشانه‌های مبتنی بر میزبان این بدافزار چیست؟
۵. اقدامات مختلفی که از طریق شبکه می‌توان به این بدافزار، دستور انجام آن‌ها را داد چیست؟
۶. آیا هیچ نشانه مبتنی بر شبکه برای این بدافزار وجود دارد؟

آزمایش ۹-۲

فایل Lab09-02.exe را با استفاده از OllyDBG برای پاسخ به سوالات زیر تحلیل کنید.

سوالات:

۱. چه رشته‌هایی را به صورت استاتیک در فایل باینری مشاهده می‌کنید؟
۲. چه اتفاقی رخ می‌دهد هنگامی‌که این بدافزار را اجرا می‌کنید؟
۳. چه کاری باید انجام دهیم تا این فایل پیلود مخرب خودش را اجرا کند؟

۴. در آدرس 0x00401133 چه اتفاقی رخ می‌دهد؟

۵. چه پارامترهایی به سابروتین موجود در آدرس 0x00401089 عبور داده می‌شود؟

۶. چه نام دامنه‌ای این بدافزار مورد استفاده قرار می‌دهد؟

۷. چه روتین رمزنگاری برای مبهم‌سازی نام دامنه استفاده شده است؟

۸. اهمیت فراخوانی تابع CreateProccessA در آدرس 0x0040106E چیست؟

آزمایش ۳-۹

فایل Lab09-03.exe را با استفاده از OllyDBG و IDA Pro تحلیل کنید. این بدافزار سه کتابخانه پیوندی پویا (DLL1.dll، DLL2.dll و DLL3.dll) را بارگذاری می‌کند که همه آن‌ها برای پاسخگویی به یک محل مشابه در حافظه بارگذاری می‌شوند. بنابراین، هنگام مشاهده این کتابخانه‌های پیوندی پویا در OllyDBG در مقابل IDA Pro، ممکن است کد در حافظه‌های مختلف نمایش داده شود. هدف از این آزمایشگاه این است، هنگامی‌که در دیباگر OllyDBG به دنبال کدی هستید بتوانید محل درست آن را در IDA Pro پیدا کنید.

سوالات

۱. چه کتابخانه‌های پیوندی پویایی به فایل Lab09-03.exe وارد شده‌اند؟

۲. آدرس پایه درخواست شده توسط DLL1.dll ، DLL2.dll و DLL3.dll چه هستند؟

۳. هنگامی‌که از OllyDBG برای دیباگ Lab09-03.exeاستفاده می‌کنید، چه آدرس‌های پایه‌ای به DLL1.dll، DLL2.dll وDLL3.dll اختصاص داده می‌شود؟

۴. هنگامی‌که فایل Lab09-03.exe تابعی را از DLL1.dll فراخوانی می‌کند، این تابع ورودی چه کاری انجام می‌دهد؟

۵. هنگامی‌که Lab09-03.exe تابع WriteFile را فراخوانی می‌کند، نام فایلی که این تابع روی دیسک سخت می‌نویسد چیست؟

۶. هنگامی‌که Lab09-03.exe یک Job را با استفاده از NetScheduleJobAdd ایجاد می‌کند، از کجا برای پارامتر دوم داده دریافت می‌کند؟

۷. هنگام اجرا یا دیباگ برنامه، مشاهده خواهید کرد برنامه سه قطعه داده سری در خروجی چاپ می‌کنند. آن‌ها DLL 1 mystery data 1، DLL 2 mystery data 2 و DLL 3 mystery data 3 چه هستند؟

۸. چگونه می‌توانید DLL2.dll را به درون دیزاسمبلر IDA Pro بارگذاری کنید، بطوریکه آدرس بارگذاری توسط OllyDBG با آن مطابقت کند؟