قسمت ۲ - تجزیه و تحلیل کاربردی بدافزارها

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

آزمایشگاه امنیت کی‌پاد

تحلیل استاتیک ساده

بررسی بدافزارها را با روش تحلیل استاتیک آغاز می‌کنیم که در حقیقت این روش اولین گام مطالعه روی بدافزار است. روش تحلیل استاتیک می‌تواند با پردازش جزییات و استراکچر کدهای یک نرم‌افزار عملکرد آن برنامه را تشریح کند. قابل ذکر است، در این روش بالعکس روش تحلیل دینامیک که باید بدافزار را اجرا کرد، در روش تحلیل استاتیک فایل اجرایی برنامه اجرا نمی‌شود. در این فصل از کتاب، تکنیک‌های گوناگونی را خواهید آموخت که می‌توانید با استفاده از آن‌ها اطلاعات نسبتا کاملی از بدافزار استخراج کنید. تکنیک‌های زیر در این قسمت بررسی خواهند شد:

- استفاده از ضدویروس‌ها به منظور اهراز هویت بدافزارها.
- استفاده از هش برنامه برای شناسایی بدافزار.
- استخراج اطلاعات از توابع، رشته‌ها و هدرهای یک فایل مخرب.

هر یک از این تکنیک‌ها در تحلیل بدافزار می‌توانند اطلاعات مختلف و گوناگونی به تحلیلگر ارائه بدهند. با این حال متخصصین تحلیلگر بدافزار باید مطابق با مقاصد خود از جمع‌آوری اطلاعات یکی از این تکنیک‌ها را به خدمت بگیرند. معمولا متخصصین از تکنیک‌های ترکیبی و مختلفی استفاده می‌کنند تا بتوانند نهایت اطلاعات ممکن را از هدف خود استخراج کنند.

اسکن با ضدویروس‌ها، اولین گام سودمند

هنگامی‌که در آینده بدافزاری را شروع به تحلیل می‌کنید، در اولین گام بهتر است آن را توسط ضدویروس‌های مختلف مورد بررسی قرار بدهید، هر کدام از ضدویروس‌ها ممکن است هویت نرم‌افزار مدنظرتان را شناسایی کنند و کار شما را در تحلیل آن تسهیل بخشند. با این حال مطمئناً ضدویروس‌ها خیلی عالی و کامل نیستند، زیرا آن‌ها به صورت گسترده‌ای متکی به یک پایگاه داده‌ای از کدهای‌آلوده[1] (سیگنیچرها[2]) و تحلیل تطبیق الگو و رفتار[3] (هیروستیک[4]) در تشخیص فایل‌های آلوده هستند. اما مسئله اصلی در استفاده از این نوع

[1] Suspicious Code Database
[2] Files Signatures
[3] Behavioral and Pattern-matching Analysis
[4] Heuristics

ضدویروس‌ها این است که نویسندگان بدافزار به راحتی می‌توانند استراکچر بدافزار خود را تغییر بدهند تا توسط هیچ ضدویروسی شناخته نشود. همین موضوع باعث می‌شود، این روش در تحلیل بدافزارهای پیچیده و حرفه‌ای نا کارآمد باشد.

همچنین اگر به عنوان بدافزارنویس، به‌گونه‌ای بدافزار خود را طراحی کنید که اطلاعات آن در هیچ ضدویروسی موجود نباشد، هیچ کدام از آن‌ها نمی‌توانند بدافزار شما را بدون تحلیل کامل شناسایی کنند.

همچنین قابل ذکرست، به ندرت بدافزاری تولید می‌شود که در لحظه اول فعالیت خود توسط ضدویروس‌ها شناسایی شوند، فقط فناوری هیروستیک در اغلب اوقات در شناسایی بدافزارها موفق است، اما این فناوری را هم می‌توان با استفاده از تکنیک‌های جدید و منحصربفرد دور زد.

اما این نکته را در نظر داشته باشید، ضدویروس‌های گوناگون دارای بانک اطلاعاتی و هیروستیک متفاوتی نسبت به یکدیگر هستند. با استناد به این موضوع، این مسئله بسیار مفید است که بدافزار را در چندین ضدویروس مختلف به صورت همزمان مورد بررسی و آزمایش قرار بدهیم.

خوشبختانه، برای این موضوع وب‌گاه‌هایی وجود دارند که این کار را برای ما انجام می‌دهند، از قبیل virtustotal.com یا hybrid-analysis.com یا any.run که به راحتی می‌توانید فایل خود را در آن به منظور بررسی توسط موتورهای ضدویروس‌های گوناگون در آن‌ها بارگذاری کنید و در آخر بعد از اعمال بررسی فایل مدنظرتان؛ گزارشی از جزییات آن برنامه به‌دست آورید. گزارش این وب‌سایت‌ها شامل اطلاعاتی در مورد ضدویروس‌هایی می‌شود که نرم‌افزار شما را مخرب شناسایی کرده‌اند. تصویر نمونه‌ای از این وب‌گاه در تصویر ۱-۱ نمایش داده شده است:

تصویر ۱-۱: وب‌گاه **VirusTotal**

هش، یک نشانه منحصربفرد برای شناسایی بدافزار[1]

هش یک روش رایج برای شناسایی بدافزارهای به صورت منحصربفرد است که تحلیلگران بدافزار از آن استفاده می‌کنند. در این روش نرم‌افزار مخرب در داخل یک برنامه محاسبه‌گر هش اجرا می‌گردد، سپس آن برنامه یک شناسه منحصربفرد در قالب یک هش برای باینری آن بدافزار در خروجی تولید می‌کند.

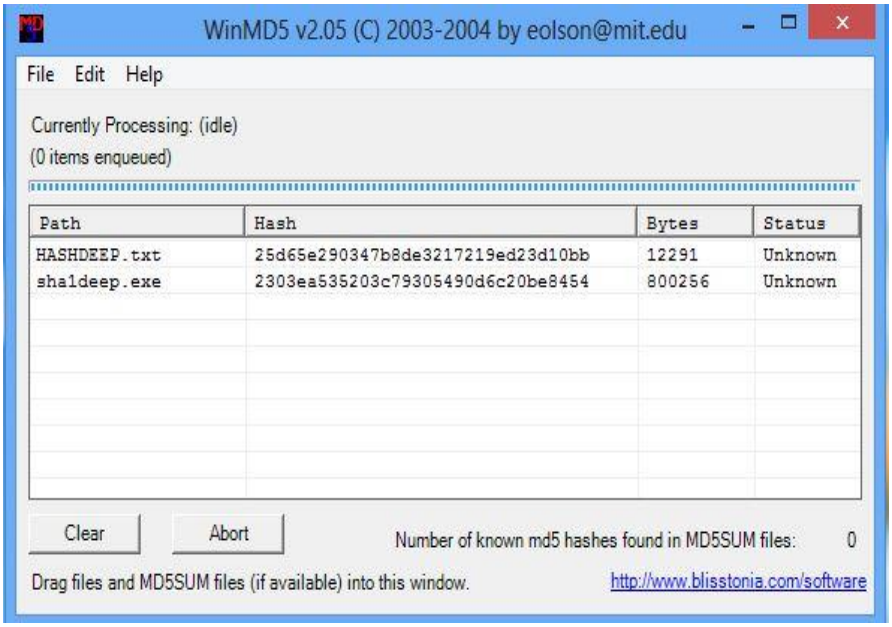
الگوریتم خلاصه پیام[2] که به اختصار MD5 خوانده می‌شود، به همراه الگوریتم هش ایمن[3] که به اختصار SHA-1 خوانده می‌شود، جزوء رایج‌ترین الگوریتم‌های تولید هش در شناسایی بدافزارها هستند. به عنوان مثال، با استفاده از برنامه رایگان md5deep که یک برنامه تحت خط فرمان است، شما به عنولن یک تحلیلگر بدافزار می‌توانید در ویندوز هش نرم‌افزارهای مخرب خود را محاسبه کنید. هنگامی‌که برنامه md5deep را به همراه فایل مخرب اجرا می‌کنید؛ خروجی به مانند زیر تولید می‌گردد.

لیست ۱-۱: هش برنامه **bin.exe**

```
C:\Users\clightning> md5deep c:\WINDOWS\system32\bin.exe
373e7a863a1a345c60edb9e20ec3231
c:\WINDOWS\system32\sol.exe
```

عبارت 373e7a863a1a345c60edb9e20ec3231، یک هش منحصربفرد برای برنامه bin.exe می‌باشد. علاوه بر برنامه md5deep که یک برنامه مبتنی بر خط فرمان به شمار می‌رود، برنامه WinMd5 هم وجود دارد که یک برنامه محاسبه‌گر هش با رابط گرافیکی است. این برنامه امکان محاسبه هش چندین برنامه را به صورت همزمان ارائه می‌دهد. در تصویر ۲-۱ محیط آن نمایش داده شده است.

[1] A Fingerprint for Malware
[2] Message-Digest 5
[3] Secure Hash Algorithm



تصویر ۲-۱: محیط برنامه **WinMD5.exe**

البته نرم‌افزارهای دیگری مانند CFFExplorer یا ExeInfo در ویندوز و همچنین BAP یا Cutter هم وجود دارند که علاوه بر نمایش شناسه منحصربفرد باینری یک بدافزار در قالب الگوریتم‌های هش MD5 و SHA1، اطلاعات کاربردی دیگری را هم ارائه می‌دهند که می‌توانند برای شما به عنوان یک تحلیلگر بدافزار سودمند واقع شوند. حتی کتابخانه‌هایی وجود دارند که به شما این توانایی را می‌دهند خود یک ابزار برای تحلیل ساختار فایل‌های PE یا ELF یا Mach-O پیاده‌سازی کنید. با این حال هنگامی‌که یک هش منحصربفرد برای یک بدافزار را دارید، از آن می‌توانید برای مقاصد زیر استفاده کنید.

- از آن هش به عنوان یک لیبل[1] برای شناسایی بدافزار استفاده کنید.
- آن هش را به صورت آنلاین برای شناسایی هویت مخرب آن در دیگر نقاط جستجو کنید.
- آن هش را با دیگر تحلیلگران به منظور تشخیص بدافزار در محیط‌های دیگر به اشتراک بگذارید.

شناسایی رشته‌ها

در یک برنامه می‌تواند دنباله‌ای از کاراکترهایی مانند "My name is Milad Kahsari Alhadi" وجود داشته باشد. به این دنباله از کاراکترها اصطلاحاً یک رشته یا String می‌گویند. یک برنامه شامل

[1] Label

رشته‌های گوناگونی است، از قبیل پیام‌های خاصی که در برنامه نشان داده می‌شوند، آدرس‌های اینترنتی که به آن متصل می‌گردند و یا یک مسیر خاص که یک فایل را در آن کپی می‌کنند و غیره. جستجو میان رشته‌ها می‌تواند یک راه ساده برای درک ویژگی‌های یک برنامه باشد. به عنوان مثال، اگر برنامه به یک URL دسترسی پیدا کند، آدرس URL دسترسی گرفته شده را به عنوان یک رشته ذخیره شده در برنامه مشاهده خواهید کرد. در گام بعد می‌توانید بررسی کنید که چرا یک برنامه با آن آدرس ارتباط برقرار می‌کند و همچنین بعد برقراری ارتباط چه اطلاعاتی بین برنامه کلاینت و آن سرور ارسال و دریافت می‌شود. خود همین مسئله، می‌تواند در موقعیت‌های گوناگونی تشریح کننده هویت یک فایل باینری باشد.

این نوع رشته‌های کاراکتری که عموماً دارای قالب استاندارد ASCII یا Unicode را با استفاده از برنامه String (این ابزار را می‌توانید از آدرس http://goo.gl/BreN4F دانلود کنید) یا برنامه CFFExplorer در یک فایل‌اجرایی استخراج و مشاهده کنید.

نکته: مایکروسافت از اصطلاح رشته کاراکتر گسترده[1] برای توصیف پیاده‌سازی رشته‌های کاراکتری فالب استاندارد Unicode استفاده می‌کند که مقداری از قالب Unicode استاندارد متفاوت است. همچنین به این نکته توجه داشته باشید، در سرتاسر این کتاب موقعی که به استاندارد Unicode اشاره خواهیم کرد، در اصل به پیاده‌سازی استاندارد آن اشاره داریم و به صورت مشخص به نوع پیاده‌سازی این استاندارد بر روی پلتفرم‌های گوناگون متمرکز نخواهیم شد.

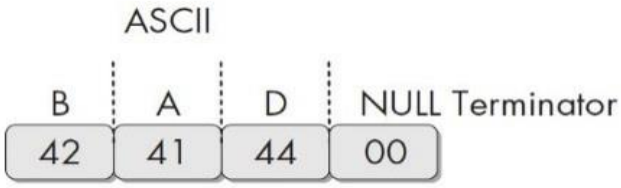
با این حال هر دو قالب‌های ذخیره‌سازی رشته‌های کاراکتری Unicode و ASCII رشته‌ها را به صورت ترتیبی از کاراکترها ذخیره کرده و در پایان رشته یک کاراکتر Null (در استاندارد ASCII) یا دو کاراکتر Null (در استاندارد Unicode) به آن می‌افزایند که گواه پایان رشته در این دو قالب ذخیره‌سازی است.

تنها تفاوتی‌که این دو قالب با همدیگر دارند، این است که قالب ASCII برای ذخیره‌سازی هر کاراکتر یک بایت در نظر می‌گیرد، در حالی‌که قالب Unicode برای ذخیره‌سازی هر کاراکتر دو بایت را در نظر می‌گیرد. زیرا قالب Unicode یک قالب رشته جهانی است و از تمامی زبان‌های رایج باید پشتیبانی کند. همان‌طور

---

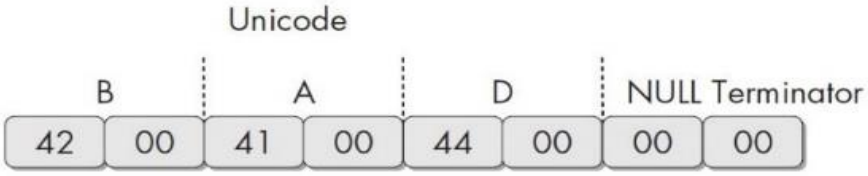
[1] Wide character string

که در تصویر ۳-۱ مشاهده می‌کنید، رشته Bad با قالب ASCII نمایش داده شده است. در این قالب‌های ذخیره‌سازی کاراکترها با کدهای هگزادسیمال خود ذخیره‌سازی می‌شوند. به عنوان مثال همان‌طور که در تصویر ۳-۱ مشاهده می‌کنید، رشته BAD با کدهای هگزادسیمال 0x42، 0x41، 0x44 و 0x00 که یک کاراکتر Null است به ترتیب در این قالب ذخیره‌سازی رشته قرار گرفته‌اند. در اینجا کد هگزادسیمال 0x42 کاراکتر حرف B، کد هگزادسیمال 0x41 حرف A و کد هگزادسیمال 0x44 حرف D را نمایش می‌دهد و همان‌طور هم که قبلاً ذکر شد، کد هگزادسیمال 0x00 نمایش‌دهنده کاراکتر Null است که پایان یک رشته را مشخص می‌کند.



تصویر ۳-۱: نمایش رشته **BAD** در قالب **ASCII**

تصویر ۴-۱ رشته BAD را در قالب Unicode نمایش می‌دهد. به سادگی می‌توانید با استناد به این دو تصویر به تفاوت موجود میان این دو روش ذخیره‌سازی رشته‌های کاراکتری پی ببرید. قالب Unicode کاراکترها را به صورت سلسله مراتبی 0x42، 0x00 سپس 0x41، 0x00 و... ذخیره‌سازی کرده است.



تصویر ٤-۱: نمایش رشته **BAD** در قالب **Unicode**

در این قالب ذخیره‌سازی رشته برخلاف قالب ASCII که از یک بایت برای نمایش یک کاراکتر استفاده می‌کرد از دو بایت استفاده شده است. به عنوان مثال در این قالب از کدهای هگزادسیمال 0x42 به همراه 0x00 برای نمایش کاراکتر B و از کد هگزادسیمال 0x41 به همراه 0x00 برای نمایش کاراکتر A و از کد هگزادسیمال 0x44 به همراه 0x00 برای نمایش کاراکتر D استفاده شده است و در پایان هم برای اعلام پایان رشته از دو کد هگزادسیمال 0x00 استفاده شده است.

نکته: هنگامیکه در مورد استاندارد Unicode صحبت می‌کنیم، باید به این مسئله توجه داشته باشیم که این استاندارد شامل گونه ۸ بیتی (UTF-8)، ۱۶ بیتی (UTF-16)، و ۳۲ بیتی (UTF-32) می‌شود که نوع پیاده‌سازی آن‌ها با یکدیگر متفاوت است. مثلا نوع استاندارد UTF-8، یک قالب کدگذاری است که برای کدگذاری هر کاراکتر از تعداد بایت‌های متغیری استفاده می‌کند، از همین روی آن را یک استاندارد Variable-length (در اصطلاح مایکروسافت Multibyte) معرفی می‌کنند، در حالیکه در استاندارد UTF-16 برای کدگذاری هر کاراکتر از دو بایت و در استاندارد UTF-32 برای کدگذاری هر کاراکتر از ٤ بایت استفاده می‌شود.

استانداردهای کدگذاری Fixed-Length و Variable-Length

به هر صورت، وقتی ما در مورد ساختارهای انکدینگ صحبت می‌کنیم، با دو خانواده از استانداردهای انکدینگ روبه‌رو هستیم. اولین خانواده را با عنوان Fixed-Length و دومین را با عنوان Variable Length می‌شناسیم. استانداردهای انکدینگ Fixed-length چه ۸ بیتی باشند (برای پشتیبانی از کاراکترهای لاتین)، چه ۱۶ بیتی باشند برای پشتیبانی از کاراکترهای لاتین و شبه لاتین، چه ۳۲ بیتی باشند برای پشتیبانی از تمامی فرم کاراکترها و ایموجی ها و ... یک مزیت و عیب بزرگی دارند.

مزئیت استفاده از استانداردهای Fixed-length این است که در هنگام پارز آن اشکالی وجود نخواهد داشت، و چون در این نمونه از دو بایت برای هر کاراکتر استفاده شده است، می‌توان طیف وسیعی از کاراکترها را با استفاده از آن کدگذاری و استفاده کرد، و در ادامه هم برای پارز آن مسئله خاصی (مانند همگام‌سازی و ...) نداشته باشد، چون هر دو بایت معرف یک کاراکتر هستند. برای پارز یک فایل با همچین الگویی کافی است ابتدای فایل را تشخیص داد و سپس دو بایت دو بایت آن را پارز کرد.

ولی یک عیب بزرگ دارد، آن هم این است که اگر با استفاده از این الگو برای کدگذاری کاراکترهای لاتین استفاده شود که برای نمایش آن‌ها فقط یک بایت نیاز است، میزان زیادی از حافظه را از دست خواهید داد چون بدون دلیل یک بایت با Null پر خواهد شد که لزومی ندارد همچین اتفاقی رخ بدهد.

برای مدیریت این مسئله می‌توان از استانداردی استفاده کرد که از یک بایت برای کدگذاری کاراکترهای لاتین استفاده می‌شود، مانند ASCII ولی این استاندارد موجب خواهد شد اگر از کاراکترهای فارسی استفاده کرده باشید، دیگر فایل پارز نشود و بعد از باز شدن کاراکتر ؟؟؟؟ نمایش داده شود. راه دوم برای حل این مسئله استفاده از انکدینگ‌های Variable-Length است که این استاندارد هم مجموعه‌ای از نقاط منفی و مثبت دارند. همانطور که پیش از این ذکر شد، در خانواده کدگذاری Variable Length، استاندارد UTF-8 وجود دارد که به صورت هوشمند مشخص می‌کند که برای کدگذاری هر کاراکتر از چند بایت استفاده شود، مثلا کاراکترهای لاتین ۱ بایت، کاراکترهای فارسی دو بایت، کاراکترهای چینی و ایموجی ۴ بایت حافظه استفاده خواهند کرد.

با این رویکرد می‌توان به صورت بهینه از حافظه استفاده کرد و تمامی کاراکترها از قبیل انگلیسی، فارسی، چینی و حتی ایموجی‌ها ... را انکد کرد، اما در پارز آن اتلاف توان زیادی رخ خواهد داد، چون به تصویری باید تشخیص بدهید که کاراکتر برای چه زبانی است و چه تعداد بایت متعلق به آن است. به همین دلیل، در پارزرهای استانداردهای Variable-Length آسیب‌پذیری بسیار زیاد وجود دارد، اگر به درستی پیاده‌سازی نشوند.

در تصویر ۴-۱ هم اگر مشاهده می‌کنید کاراکتر 00 تکرار شده‌اند به خاطر این است که استاندارد کدگذاری کاراکترهای از خانواده Fixed-Length است و برای کدگذاری هر کاراکتر از دو بایت استفاده شده است. تا اولا به صورت نسبتا بهینه‌ای از حافظه استفاده شود، دوماً خطر آسیب‌پذیری در پارزر آن کاهش پیدا کند و سوماً حجم وسیعی از کاراکترها را بتوان با استفاده از آن کدگذاری کرد. برای اطلاعات بیشتر درباره این مباحث در برنامه‌نویسی سیستمی می‌تواند به آدرس (https://b2n.ir/677333) رجوع کنید.

در نتیجه، هنگامی‌که برنامه Strings.exe یا CFFExplorer درون یک فایل‌اجرایی جستجو رشته‌های ASCII و Unicode را آغاز می‌کند، محتویات و قالب‌بندی استریم بایت‌ها را نادیده خواهند گرفت و از ابتدا فایل تا انتها را پارز خواهند کرد تا کاراکترهای نمایش را استخراج کنند و نمایش بدهند، بنابراین این ابزارها می‌توانند هر نوع فایلی را تحلیل کرده و تمامی رشته‌های آن را شناسایی کنند (همچنین این موضوع بدین معنا است که این برنامه ممکن است کاراکترهایی را به عنوان رشته شناسایی کند که در واقع اصلا رشته نیستند). این برنامه‌های قالب‌های ASCII و Unicode را برای رشته‌های سه حرفی یا دنباله‌ای بزرگتر از سه کاراکتر که به یک کاراکتر پایان دهنده خاتمه (NULL Byte) پیدا کرده است، جستجو می‌کنند.

گاهی اوقات برنامه strings.exe رشته‌هایی را شناسایی می‌کند که واقعا رشته نیستند. به عنوان مثال، اگر برنامه Strings.exe با یک سلسله مراتب از کدهای هگزادسیمال 0x56، 0x50، 0x33 و 0x00 مواجه شود آن را رشته VP3 تفسیر می‌کند. اما امکان دارد آن بایت‌ها یک رشته واقعی نباشند؛ آن‌ها می‌توانند یک آدرس حافظه، دستور ماشین یا داده‌های مورد استفاده برنامه باشند.

اما خوشبختانه اغلب رشته‌های غیرمعتبر قابل شناسایی هستند. چون آن‌ها یک متن غیردستوری را نمایش می‌دهند. برای درک این مطلب یک مثال را با هم بررسی می‌کنیم. در این قسمت یک فایل‌اجرایی (bp6.ex_) را از طریق strings.exe اجرا کرده و خروجی آن را مورد تحلیل قرار خواهیم داد.

لیست ۲-۱: رشته‌های برنامه **bp6.ex_**

```
C:>strings bp6.ex_
VP3
VW3
t$@
D$4
99.124.22.1 ❹
e-@
GetLayout ❶
GDI32.DLL ❸
SetLayout ❷
M}C
Mail system DLL is invalid.!Send Mail failed to send message. ❺
```

در این مثال، رشته‌هایی که برجسته شده‌اند، می‌توانند در نظر گرفته نشوند. همچنین با توجه به مثال آورده شده، می‌توانید رشته‌های کوتاه که دارای معنی خاصی نیستند را حذف کنید. رشته‌های GetLayout (شماره ۱) و SetLayout (شماره ۲) که در خروجی بالا نمایش داده شده‌اند دو تابع سامانه‌ای برای سامانه‌عامل ویندوز هستند که برای نمایش پنجره‌های گرافیکی مورد استفاده قرار می‌گیرند. ما می‌توانیم به سادگی آن‌ها را شناسایی کنیم، چون اسامی توابع ویندوزی معمولا با یک حرف بزرگ شروع می‌شوند.

در خروجی بالا، GDI32.Dll (شماره ۳) برای ما دارای معنا است، زیرا GDI32.Dll نام یک کتابخانه پیوندی پویا ویندوز (DLL) می‌باشد که توسط برنامه‌های گرافیکی مورد استفاده قرار می‌گیرد (فایل‌های DLL شامل کدهای اجرایی می‌شوند که میان برنامه‌های کاربردی به اشتراک گذاشته شده‌اند). همچنین در خروجی (شماره ۴) یک IP آدرس نمایش داده شده است.

در پایان (شماره ۵) یک پیام رشته‌ای با مضمون " Mail system DLL is invalid.!Send Mail failed to send message." مشاهده می‌شود، که یک پیام خطا در فایل باینری است. اغلب اوقات،

با ارزش‌ترین اطلاعات به‌دست آمده از اجرای بدافزارها درون برنامه Strings.exe یا برنامه‌های مشابه مانند آن مثل CFFExplorer همین پیام‌های خطایی است که پیدا می‌شوند. مثلا این پیام خاص دو چیز را به ما می‌رساند. یک هدف بدافزار ارسال پیام است (احتمالا از طریق ایمیل) و دو، بدافزار باید وابسته به یک تابع سیستمی برای ارسال ایمیل باشد. این اطلاعات به ما توصیه می‌کنند که ما باید رویدادهای ثبت شده با محوریت ارسال ایمیل برای یک ترافیک مشکوک و همچنین استفاده از کتابخانه‌های سیستمی را که برای ارسال ایمیل بدافزار با آن در ارتباط است، مورد بررسی قرار بدهیم. اما به این نکته هم توجه کنید، کتابخانه‌های پیوندی پویا به تنهایی خودشان برای سامانه مخرب نیستند، بلکه بدافزارها اغلب از آن‌ها برای اعمال مقاصد خودشان سوءاستفاده می‌کنند.

بدافزارهای مبهم‌سازی و پَک شده[1]

نویسندگان بدافزار اغلب برای این‌که شناسایی یا تحلیل فایل‌های خودشان را دشوار سازند از روش پکینگ یا مبهم‌سازی فایل‌ها استفاده می‌کنند. ولی شاید سوال پرسیده شود که مبهم‌ساز چیست؟ مبهم‌سازها برنامه‌هایی هستند که نویسندگان بدافزار برای مبهم‌سازی کد بدافزارهای خود از آن‌ها استفاده می‌کنند. پکرها هم یک زیر مجموعه از برنامه‌های مبهم‌ساز هستند که بدافزارها را فشرده می‌سازند و آن‌ها را غیر قابل تحلیل می‌کنند. هر دو این تکنیک‌ها به شدت تلاش‌های شما را در تحلیل استاتیک بدافزار محدود و دشوار می‌کنند.

در حالت معمول تمامی برنامه‌های کاربردی مشروع تقریبا شامل رشته‌های متعدد و بسیاری می‌شوند. ولی بدافزارهایی که مبهم‌سازی و یا پَک می‌شوند، شامل رشته‌های کمتری نسبت به برنامه‌های عادی خواهند شد. در نتیجه اگر پس از جستجو فایل‌اجرایی با strings.exe متوجه شدید که آن فایل‌اجرایی مقدار خیلی کمی رشته دارد، به احتمال خیلی زیاد آن مبهم‌سازی یا پَک شده است و می‌توان حدس زد که آن یک بدافزار باشد.

نکته: بدافزارهای مبهم‌سازی یا پَک شده حداقل شامل توابع LoadLibrary و GetProcAddress می‌شوند که از آن‌ها برای آنپک خود و دسترسی به توابع خارجی استفاده می‌کنند.

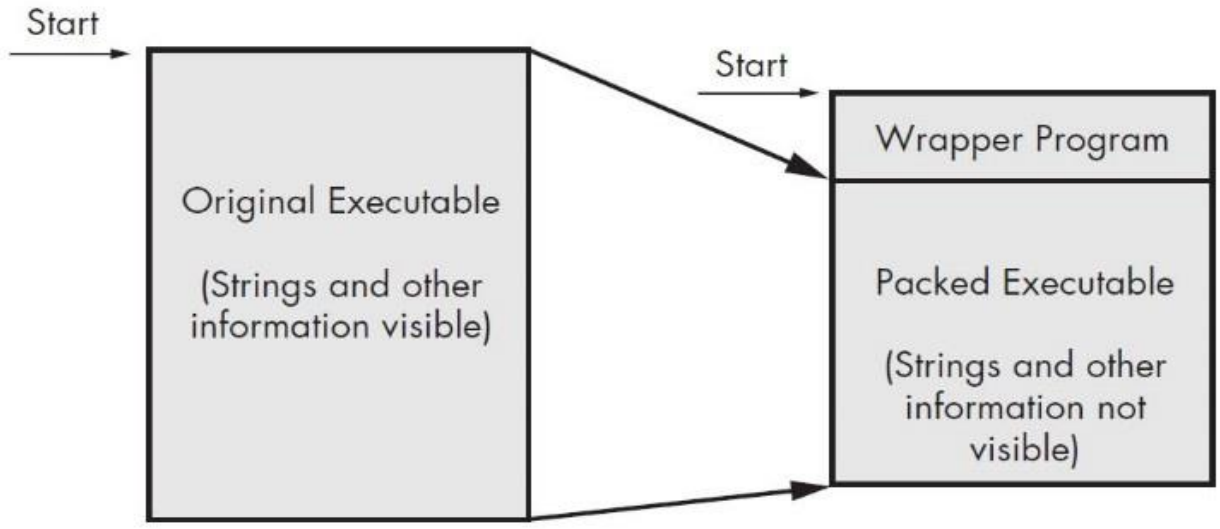
---

[1] Packed and Obfuscated Malware

فایل های پک شده[1]

همان‌طور که در تصویر ۵-۱ نمایش داده شده است، هنگامی‌که یک برنامه بسته‌بندی شده اجرا می‌گردد، در کنار آن یک برنامه کوچک Wrapper اضافی هم اجرا می‌شود تا آن را آنپَک کرده و در پایان فایلی که آنپَک شده را اجرا کند.

هنگامی‌که یک برنامه پَک شده با روش استاتیک تحلیل می‌شود، تنها آنپَکر را می‌توان کالبدشکافی کرد. (در مقالات بعدی در مورد پَکینگ و آنپَکینگ فایل‌ها با جزییات بیشتری بحث خواهیم کرد.)



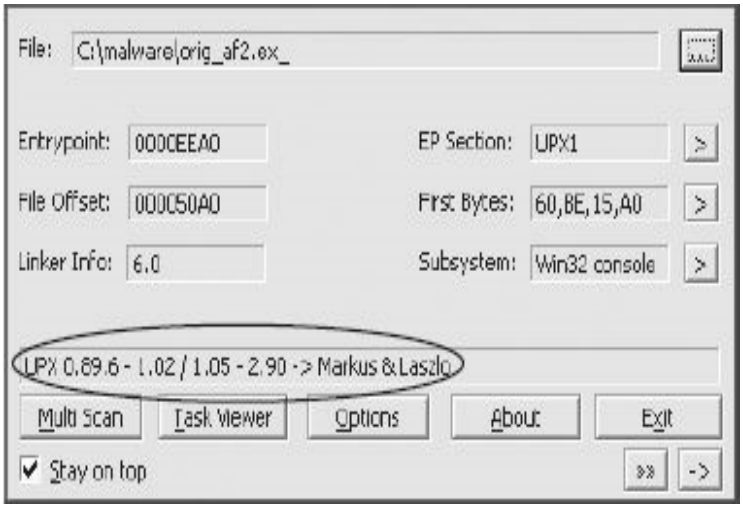
تصویر ۵-۱: سمت چپ فایل‌اجرایی اصلی برنامه با تمامی رشته‌ها، ورودی‌ها و دیگر اطلاعات قابل روئیت است. سمت راست فایل‌اجرایی پَک شده قرار دارد. تمامی رشته‌ها فایل پَک شده، ورودی‌ها و دیگر اطلاعات فشرده شده، توسط اکثریت ابزارهای تحلیل استاتیک قابل رویت نیستند.

شناسایی پَکر فایل‌اجرایی[2]

یکی از راه‌های شناسایی فایل پک شده، استفاده از برنامه PEiD یا ExeInfo در ویندوز و استفاده از ابزارهایی مانند ELF Parser یا BAP در لینوکس است. به عنوان مثال، متخصصین می‌توانند با استفاده از ابزار PEiD نوع برنامه پَکر و زبان به خدمت گرفته شده برای نوشتن آن برنامه را مورد شناسایی قرار دهند. برنامه PEiD

[1] Packing Files
[2] Detecting Packers

باعث می‌شود، تحلیل فایل پَک شده بسیار آسان شود. تصویر ۶-۱ اطلاعات فایل bin.exe را نمایش می‌دهد که توسط PEiD گزارش داده شده است.



تصویر ٦-١: محیط برنامه PeiD

نکته: متاسفانه از سال ۲۰۱۱ توسعه و پشتیبانی ابزار PEiD متوقف شده است. با این وجود این ابزار همواره یکی از بهترین ابزارهای شناسایی پکرها و نوع کامپایلرهای موجود بوده است.

همان‌طور که در تصویر ۶-۱ مشاهده می‌کنید، فایل مدنظر ما با استفاده از پَکر UPX نسخه ۰.۸۹.۶-۱.۰۲ یا ۱.۰۵-۲.۹۰ پَک یا فشرده شده است. مابقی اطلاعات نمایش داده شده در تصویر ۶-۱ را در نظر نگیرید. این برنامه در فصل ۱۸ با جزییات دقیق‌تری مورد بررسی قرار می‌گیرد.

هنگامی‌که یک برنامه پَک یا فشرده شده است، ابتدا باید آن را آنپَک کنید تا بتوانید هر نوع تحلیلی بر روی آن انجام بدهید. پروسه آنپَکینگ بسیار پیچیده است و ما آن را در مقالات بعدی این سلسله مقالات با توضیحات کامل مورد بررسی قرار خواهیم داد. ولی با این حال پکر UPX بسیار معروف است و استفاده از آن به منظور آنپَک بسیار ساده می‌باشد.

نکته: پکر UPX را می‌توانید به سادگی در سامانه‌عامل‌های لینوکس مبتنی بر Debian با دستور apt install upx نصب کنید و مورد استفاده قرار بدهید.

همچنین خود برنامه PeiD یک پلاگین با نام Upx Fileinfo دارد که می‌توانید از آن برای آنپَک استفاده کنید.

به عنوان مثال در این قسمت برای آنپَک فایل پک شده با UPX که توسط PEiD شناسایی شد، ما می‌توانیم با استفاده از فرمان upx -d را آنپَک کنیم. این پکر را همچنین می‌توانید از upx.sourceforge.net دانلود کنید.

هشدار: بیشتر پلاگین‌های PEiD، فایل‌اجرایی بدافزارها را بدون ایجاد هیچ هشداری اجرا می‌کنند. همچنین، مانند برنامه‌های دیگر، مخصوصا برنامه‌هایی که برای تحلیل بدافزار مورد استفاده قرار می‌گیرند، برنامه PEiD می‌تواند خود دارای آسیب‌پذیری باشد. به عنوان مثال، برنامه PEiD نسخه ۰.۹۲ شامل یک آسیب‌پذیری سرریز بافر بود که اجازه می‌داد هکر کدهای دلخواه خود را روی ماشین هدف اجرا کند. این موضوع می‌تواند به یک بدافزارنویس باهوش این امکان را بدهد، برنامه‌ای بنویسد که ماشین تحلیلگر بدافزار را اکسپلویت کند. به همین دلیل همیشه از نسخه‌های بروز شده این ابزارها استفاده کنید.

فرمت فایل اجرایی[1]

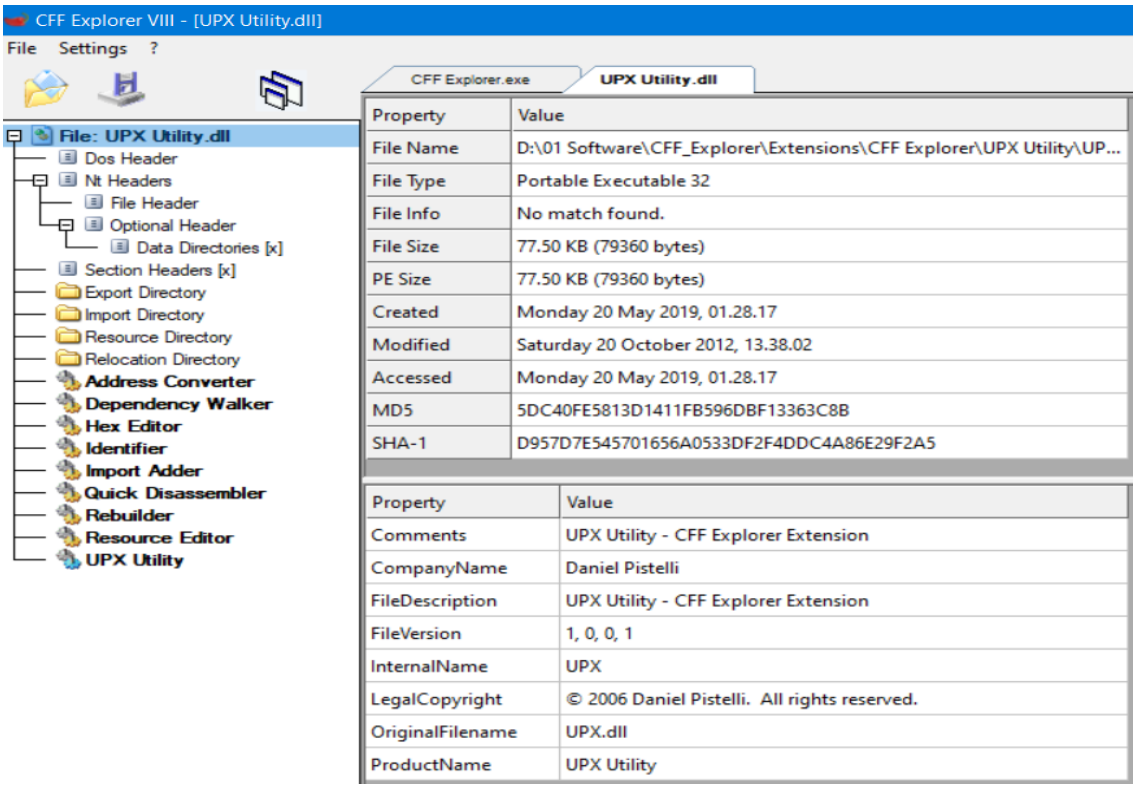
با توجه به این‌که دانستن فرمت فایل اجرایی می‌تواند اطلاعاتی در مورد عملکرد باینری به ما ارائه بدهد، اما تا به حال با ابزارها و تکنیک‌هایی آشنا شدیم که فایل را بدون توجه به قالب آن مورد اسکن قرار می‌دادند. فرمت فایل‌های اجرایی مثلا فایل اجرایی قابل حمل[2] یا به اختصار PE فرمت فایل‌های ویندوزی از قبیل فایل های اجرایی (EXE) و کتابخانه‌های پیوندی پویا (DLL) و ... را می‌دهد.

فرمت فایل اجرایی قابل حمل یا PE یک دیتا استراکچر است که شامل اطلاعاتی از قبیل تعداد سکشن ها، آدرس بارگزاری فایل اجرایی در حافظه، آدرس توابع و ... می شود که در نهایت لودر سامانه‌عامل ویندوز از آن

---

[1] Executable File Format
[2] Portable Executable File Format

اطلاعات می تواند برای مدیریت فایل اجرایی استفاده کند. تقریبا می‌توان گفت هر فایل‌اجرایی که ویندوز لود می‌کند، در قالب PE قرار می‌گیرد، اگرچه برخی قالب‌های فایل قدیمی در برخی بدافزارها مشاهده شده‌اند.

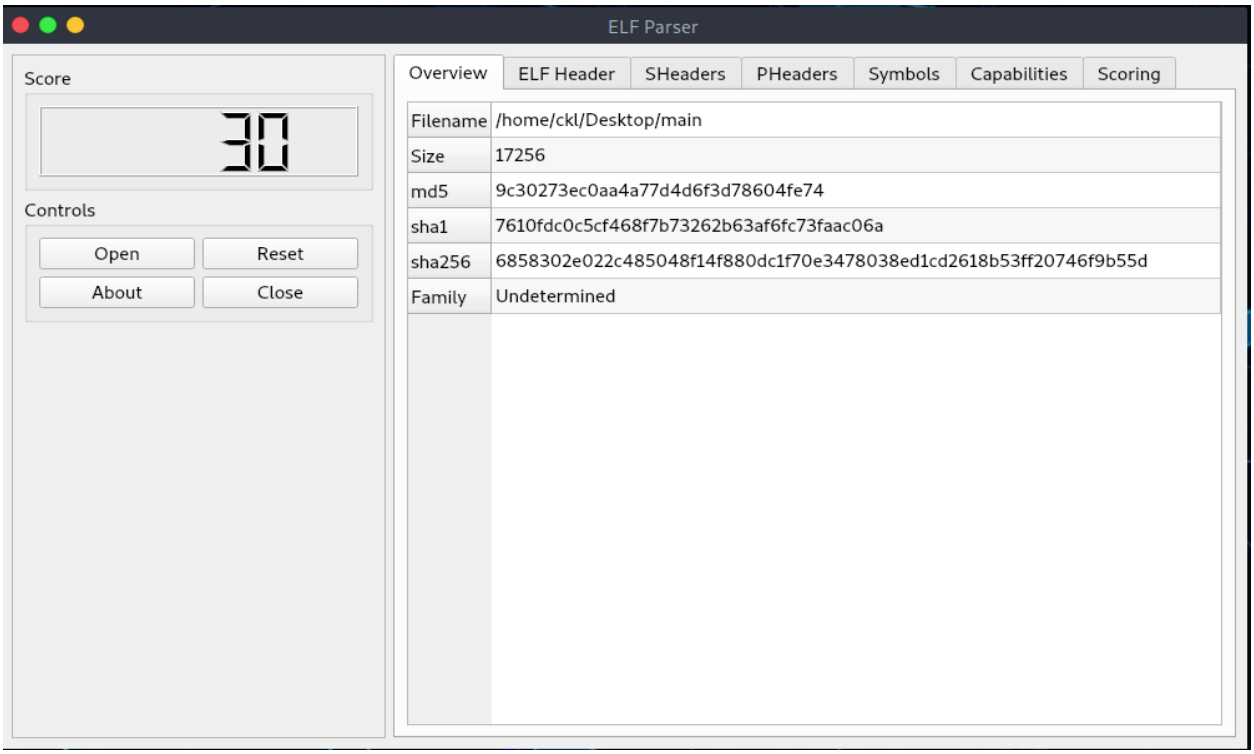


تصویر ۷-۱: محیط **CFFExplorer** و ارائه جزئیات درباره فایل **UPX Utility.dll**

فایل‌های اجرایی قابل حمل (PE) ویندوز شامل چندین هدر می‌شوند که این هدرها اطلاعاتی درباره نوع باینری، نوع معماری آن، توابع و تاریخ کامپایل و ... ارائه می‌دهند که این اطلاعات می‌توانند در تحلیل بدافزار به متخصص کمک‌های بسیاری کنند. در تصویر ۷-۱ جزئیات یک کتابخانه DLL در ویندوز نمایش داده شده است که تحلیلگران بدافزار از این اطلاعات می‌توانند برای تحلیل و استخراج عملکرد آن استفاده بهینه کنند. همانطور که در این تصویر مشاهده می‌کنید، هر بخش از قبیل NT Headers و DOS Header و Import Directory و ... شامل اطلاعات مهم و کاربردی می‌شوند که لودر سیستم عامل بدون این اطلاعات نمی‌تواند فایل مذکور را مورد استفاده قرار بدهد و آن را در حافظه ترسیم یا Map کند.

در لینوکس همین مسئله صادق است، فقط تنها تفاوت این است که لینوکس از فرمت فایل ELF به جای PE برای فایل های آبجکت خود استفاده می‌کند. این فایل فرمت تفاوت‌های اساسی با PE دارد، اما به هر صورت هر دو این فرمت‌ها دیتا استراکچری هستند که لودر سیستم عامل از اطلاعات درون آن‌ها برای اجرای

فایل‌های اجرایی و ترسیم کتابخانه‌ها در حافظه هنگام Runtime استفاده می‌کنند. در تصویر ۸-۱ محیط نرم افزار ELF Parser نمایش داده شده است که اطلاعات فرمت ELF از یک فایل باینری برای لینوکس استخراج و نمایش داده شده است:



تصویر ۸-۱: استخراج و نمایش اطلاعات یک فایل **ELF**

کتابخانه‌های دینامیک[1]

یکی از سودمندترین اطلاعاتی که ما می‌توانیم از یک فایل‌اجرایی استخراج کنیم، لیست توابعی است که به آن برنامه ایمپورت[2] شده است. توابع ایمپورت شده، توابعی هستند که توسط یک برنامه اجرایی (در ویندوز EXE و در لینوکس ELF) مورد استفاده قرار می‌گیرند، درحالی‌که در یک آبجکت اجرایی متفاوتی مانند کتابخانه‌ها پیوندی پویا (DLL) و کتابخانه ایستا (LIB) در ویندوز یا آبجکت‌های اشتراکی (SO) یا آرشیو (Ar)

[1] Dynamic Libraries
[2] Imports

در لینوکس قرار دارند. به منظور استفاده از توابع موجود در این نوع کتابخانه‌ها، آن‌ها باید به برنامه اجرایی اصلی لینک[1] شوند.

برنامه‌نویس‌ها با ایمپورت این کتابخانه‌ها به برنامه‌های خود می‌توانند بدون این‌که یک تابع را دوباره پیاده‌سازی کنند از توابع موجود در آن کتابخانه‌ها برای انجام عملیات‌های خود در پروسه متفاوتی بهره‌مند شوند. این نوع کتابخانه‌ها می‌توانند در زمان اجرا[2]، به صورت دینامیک[3] یا در زمان بارگزاری[4] به صورت استاتیک[5] به برنامه لینک شوند.

همچنین در درک ویژگی‌های باینری بدافزار دانستن چگونگی لینک کتابخانه‌ها جانبی به برنامه بسیار مهم است، زیرا اطلاعاتی که ما از یک فایل PE یا ELF پیدا می‌کنیم، مطابق با چگونگی لینک کد کتابخانه‌های جانبی یا سیستمی به برنامه اجرایی (فایل باینری) است. در این فصل از کتاب درباره چندین ابزار برای دیدن توابع ایمپورت شده به یک باینری یا یک فایل‌اجرایی PE یا ELF گفتگو خواهیم کرد.

لینک استاتیک، دینامیک و زمان اجرا[6]

لینک استاتیک کتابخانه‌ها در محیط ویندوز کمتر رایج است و بیشتر در برنامه‌های لینوکسی و یونیکسی ممکن است به عنوان یک تحلیلگر بدافزار با چنین موردی رو به رو شوید. هنگامی‌که یک کتابخانه به روش استاتیک به یک برنامه اجرایی لینک می‌شود، تمامی کد آن کتابخانه به کد اصلی برنامه در زمان کامپیال و ایجاد (Build) باینری نهایی افزوده خواهد شد که در نتیجه این عملیات حجم باینری نهایی به شکل قابل توجه‌ای افزایش می‌یابد. هنگامی‌که کد فایل‌اجرایی بدافزار را شروع به تحلیل می‌کنید، تشخیص تفاوت کد اصلی بدافزار از کدهای لینک شده به آن بسیار سخت است. زیرا هیچ چیزی در هدر فایل PE وجود ندارد که این تفاوت را نمایش دهد.

روش لینک کد کتابخانه‌ها به صورت استاتیک در برنامه‌های کاربردی رایج نیست در حالی‌که، روش لینک زمان اجرا رایج‌ترین روش لینک کد کتابخانه‌ها به یک بدافزار است. الخصوص اگر یَک یا مبهم‌سازی شده

---

[1] Linking
[2] Runtime
[3] Dynamically
[4] Loadtime
[5] Statically
[6] Static, Runtime, and Dynamic Linking

باشد. در این روش برنامه اجرایی در صورت نیاز به یک تابع، به کتابخانه آن لینک می‌شود و آن تابع را مورد استفاده قرار می‌دهد.

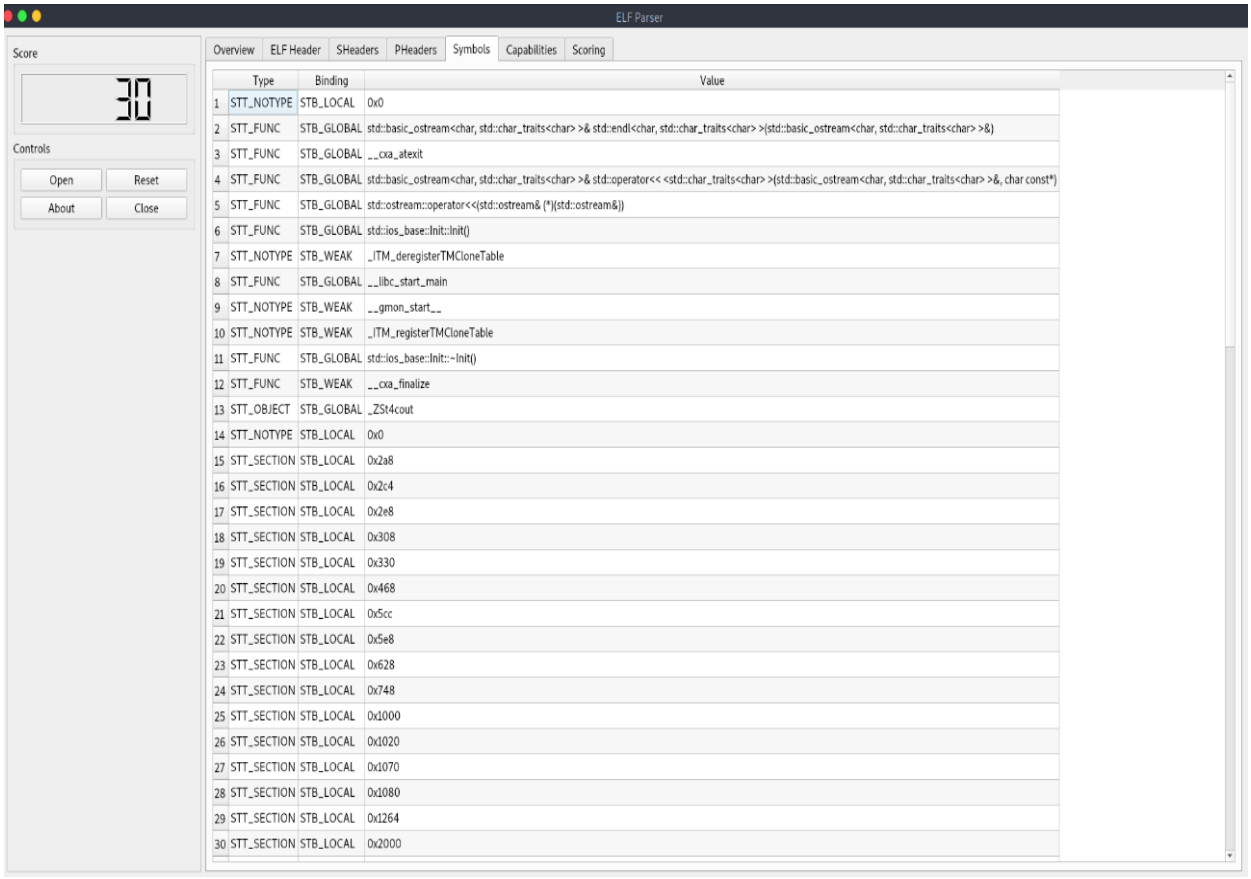
روش لینک دینامیک کتابخانه‌ها به درون پروسه‌های اجرایی بالعکس روش استاتیک می‌باشد، زیرا تمام کد کتابخانه را وارد کد اصلی برنامه نمی‌کند تا از بالا رفتن حجم برنامه ممانعت به عمل آورد. در این رویکرد، کتابخانه به صورت مجزا در حافظه بارگزاری می‌شوند و در نهایت پروسه‌ها به صورت مجزا می‌توانند آن فایل را در فضای آدرس خود ترسیم کنند و توابع درون آن‌ها را مورد استفاده قرار بدهند.

شایان ذکر است، برنامه‌نویسان می‌توانند به صورت دستی توابع درون کتابخانه‌های اشتراکی (Dll) را مورد استفاده قرار بدهند. به عبارتی برنامه‌نویس فقط با دسترسی به کتابخانه و همچنین مشاهده اطلاعات دیرکتوری توابع اکسپورت شده آن می‌تواند با استفاده از LoadLibrary و GetProcAddress توابع اکسپورت شده توسط کتابخانه اشتراکی را در پروسه برنامه خود مورد استفاده قرار بدهد. این توابع اجازه می‌دهند یک برنامه به هر تابعی در هر کتابخانه‌ای دسترسی پیدا کند. همچنین از دو تابع LdrGetProcAddress و LdrLoadDll هم بدین مقاصد استفاده می‌شود. به هر حال هنگامی‌که گفته می‌شود دو تابع GetProcAddress و LoadLibrary این اجازه را می‌دهند که هر تابعی از هر کتابخانه‌ای را فراخوانی کنید، این بدین معنی است، هنگامی‌که این توابع مورد استفاده قرار می‌گیرند، شما نمی‌توانید بصورت استاتیک بگویید کدام توابع به فایل آلوده لینک شده‌اند. شایان ذکر است، در لینوکس از تابع dlopen که معادل LoadLibrary و تابع dlsym که معادل GetProcAddress است می‌توان با این محوریت به منظور دسترسی به کدهای درون یک آبجکت مانند یک کتابخانه اشتراکی (SO) استفاده کرد.

علیرغم همه متدهای لینک کدهای کتابخانه‌ای، لینک دینامیکی رایج‌ترین و جالب‌ترین روش مورد استفاده تحلیلگران بدافزار است. هنگامی‌که کتابخانه‌ای به صورت دینامیک به یک برنامه لینک می‌شود، سامانه‌عامل میزبان (Host OS) هنگام لود برنامه کتابخانه‌های مورد نیاز آن برنامه را لینک کرده و امکان استفاده برنامه از آن کتابخانه‌ها را برای آن برنامه فراهم می‌کند.

فرمت فایل اجرایی هر سیستم عامل (بخش Import Directory در فایل‌های PE و بخش Dynamic Section برای فایل‌های ELF) تمامی اطلاعات کتابخانه‌هایی که باید توسط فایل اجرایی بارگذاری شوند و هر تابعی را که مورد استفاده برنامه است، ذخیره می‌کنند. کتابخانه‌های مورد استفاده و توابع فراخوانی شده اغلب مهم‌ترین قسمت‌های یک برنامه هستند و فهمیدن و شناختن آن‌ها برای تحلیلگران حیاتی است. چون

به ما اجازه می‌دهند که بتوانیم عملکرد برنامه را حدس بزنیم. در تصویر ۹-۱ مشاهده می‌کنید با ابزار ELFParser لیست سیمبول‌هایی (توابع) که فایل main (یک برنامه ساده که فقط پیام Milad را نمایش می‌دهد.) به آن ها نیازمند است، نمایش داده شده است.



	Type	Binding	Value
1	STT_NOTYPE	STB_LOCAL	0x0
2	STT_FUNC	STB_GLOBAL	std::basic_ostream<char, std::char_traits<char> >& std::endl<char, std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&)
3	STT_FUNC	STB_GLOBAL	__cxa_atexit
4	STT_FUNC	STB_GLOBAL	std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&, char const*)
5	STT_FUNC	STB_GLOBAL	std::ostream::operator<<(std::ostream& (*)(std::ostream&))
6	STT_FUNC	STB_GLOBAL	std::ios_base::Init::Init()
7	STT_NOTYPE	STB_WEAK	_ITM_deregisterTMCloneTable
8	STT_FUNC	STB_GLOBAL	__libc_start_main
9	STT_NOTYPE	STB_WEAK	__gmon_start__
10	STT_NOTYPE	STB_WEAK	_ITM_registerTMCloneTable
11	STT_FUNC	STB_GLOBAL	std::ios_base::Init::~Init()
12	STT_FUNC	STB_WEAK	__cxa_finalize
13	STT_OBJECT	STB_GLOBAL	_ZSt4cout
14	STT_NOTYPE	STB_LOCAL	0x0
15	STT_SECTION	STB_LOCAL	0x2a8
16	STT_SECTION	STB_LOCAL	0x2c4
17	STT_SECTION	STB_LOCAL	0x2e8
18	STT_SECTION	STB_LOCAL	0x308
19	STT_SECTION	STB_LOCAL	0x330
20	STT_SECTION	STB_LOCAL	0x468
21	STT_SECTION	STB_LOCAL	0x5cc
22	STT_SECTION	STB_LOCAL	0x5e8
23	STT_SECTION	STB_LOCAL	0x628
24	STT_SECTION	STB_LOCAL	0x748
25	STT_SECTION	STB_LOCAL	0x1000
26	STT_SECTION	STB_LOCAL	0x1020
27	STT_SECTION	STB_LOCAL	0x1070
28	STT_SECTION	STB_LOCAL	0x1080
29	STT_SECTION	STB_LOCAL	0x1264
30	STT_SECTION	STB_LOCAL	0x2000

تصویر ۹-۱: سیمبول‌های ایمپورت شده فایل ELF

به هر صورت، این اطلاعات برای ما اهمیت فراوان دارند. به عنوان مثال، اگر در محیط ویندوز فایل باینری را تحلیل کنیم که تابع URLDownloadToFile را شامل می‌شود، می‌توان حدس زد که آن برنامه به اینترنت متصل می‌شود و برخی از محتویات مورد نیاز خود را دانلود کرده و در سامانه محلی ذخیره می‌کند.

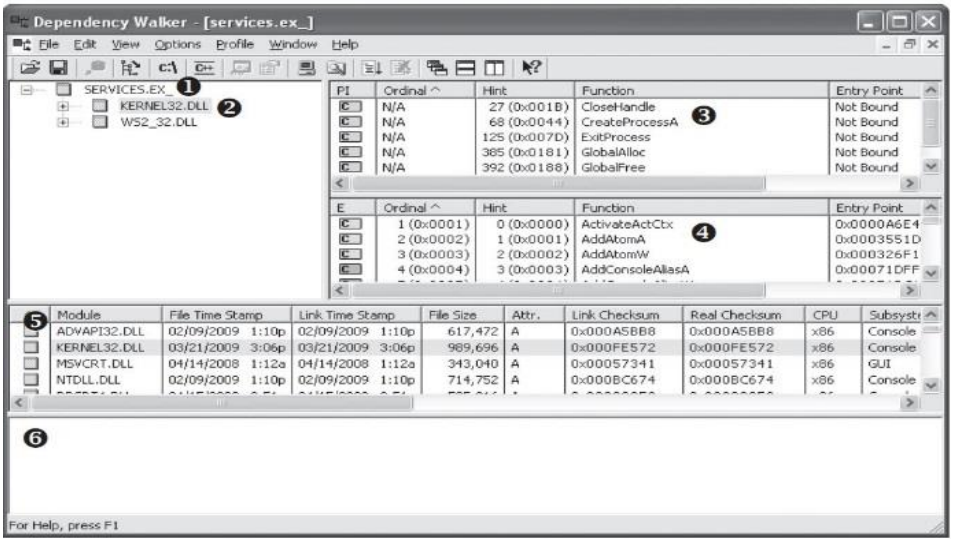
کاوش فایل‌های اجرایی با ابزارهای جانبی

به منظور کاوش و استخراج اطلاعات فایل‌های باینری با فرمت PE یا ELF ابزارهای بسیار زیادی اکنون وجود دارند. برخی از آن‌ها Cross-platform هستند، و برخی دیگر هم فقط بر روی یک پلتفرم خاص

مورد استفاده قرار می‌گیرند. در این قسمت، برخی از این ابزارهای بسیار مهم و همچنین کاربردی را مورد بررسی قرار خواهیم داد که از آن‌ها می‌توان برای استخراج اطلاعات از فایل‌های باینری استفاده کرد.

برنامه Dependency Walker (این برنامه را می‌توانید از آدرس http://goo.gl/cVL6sz دانلود کنید) توسط ویژوال‌بیسیک و یا سایر بسته‌های برنامه‌نویسی مایکروسافت توزیع می‌شود که فقط می‌تواند کتابخانه‌های یک فایل‌اجرایی را در خروجی برای شما لیست کند. مشابه این ابزار، CFF Explorer است که پیش از این معرفی شد و تقریبا عملکرد مشابه با یکدیگر دارند.

در تصویر ۱۰-۱ محیط برنامه Dependency Walker نمایش داده شده است که فایل SERVICES.EX_ را مورد تحلیل قرار داده است (شماره ۱). در سمت چپ پنجره (شماره ۲) برنامه Dependency Walker تمامی کتابخانه‌های که در برنامه مورد استفاده قرار گرفته‌اند (Kernel32.dll و WS_32.dll) را لیست کرده است.



تصویر ۱۰-۱: برنامه Dependency Walker

اگر روی KERNEL32.DLL در پنجره سمت چپ کلیک کنید، تمامی توابعی که از کتابخانه Kernel32.dll به این برنامه وارد شده‌اند در قسمت بالای سمت راست برنامه (شماره ۳) به نمایش گذاشته خواهند شد، با این حال در این قسمت چندین تابع مشاهده می‌کنیم، که جالبترین آن‌ها در این قسمت تابع CreateProcessA است. این تابع باعث می‌شود برنامه یک پروسه جدید دیگری ایجاد کند. این موضوع این را به ما می‌فهماند وقتیکه این برنامه اجرا می‌شود، باید برنامه‌های اضافی دیگری هم به همراه آن اجرا گردند.

پنجره میانی برنامه (شماره ۴) تمامی توابع کتابخانه Kernel32.dll را که می‌توانند به برنامه ایمپورت شوند را لیست کرده است. این اطلاعات برای ما در حالت معمولی مفید نیست. حال به جدول Ordinal پانل (شماره ۳ و ۴) دقت کنید. فایل‌اجرایی می‌تواند توابع ایمپورت شده را به صورت آدرس‌های عددی بجای نام بگیرد. در این حالت هیچگاه نام تابع در برنامه اجرایی نمایش داده نمی‌شود و این موضوع می‌تواند کار تحلیل نرم‌افزار را در شناسایی توابع ایمپورت شده به برنامه سخت کند. هنگامی‌که یک برنامه یک تابع را به صورت آدرس عددی ایمپورت می‌کند، می‌توانید با جستجوی مقدار عددی در پانل (شماره ۴) بفهمید چه توابعی به برنامه وارد شده است.

دو پانل پایینی (شماره‌های ۵ و ۶) اطلاعات اضافی درباره نسخه‌های Dll و خطاهای گزارش شده را نمایش می‌دهند. کتابخانه‌های وارد شده به یک برنامه می‌تواند به شما اطلاعات بسیاری در مورد ویژگی‌های برنامه بدهند. به عنوان مثال، در جدول ۱-۳ لیست کتابخانه‌های DLL رایج و نوع ویژگی‌هایی که ارائه می‌دهند آورده شده است.

جدول ۳-۱: کتابخانه‌های پویای پیوندی رایج

نام کتابخانه	توضیحات
Kernel32.dll	Kernel32 یک کتابخانه رایج مورد استفاده برنامه‌های سامانه‌عامل ویندوز است که ویژگی‌های سطح پایین از قبیل دسترسی و ایجاد تغییر در حافظه، فایل‌ها و سخت‌افزار را ارائه می‌دهد.
Advapi32.dll	این کتابخانه یک راه دسترسی به مولفه‌های پیشرفته ویندوز از قبیل مدیریت سرویس‌ها و رجیستری را ارائه می‌دهد.
User32.dll	این کتابخانه شامل همه مولفه‌های رابط کاربری برنامه‌ها، از قبیل دکمه‌ها، اسکرول‌بارها و مولفه‌های کنترل و پاسخگویی به عملیات‌های کاربر می‌شود.
Gdi32.dll	این کتابخانه شامل توابع گرافیکی برای نمایش و تغییرات در منوهای گرافیکی می‌شود.
Ntdll.dll	این کتابخانه یک رابط به کرنل سامانه‌عامل ویندوز است. برنامه‌های اجرایی عموما این فایل را به صورت مستقیم مورد استفاده قرار نمی‌دهند. گرچه این کتابخانه همیشه غیر مستقیم توسط Kernel32.dll وارد برنامه می‌شود. با این حال اگر یک برنامه اجرایی این کتابخانه را مورد

	استفاده قرار دهد، بدین معناست که برنامه‌نویس آن قصد داشته است از ویژگی‌هایی استفاده کند که به صورت پیشفرض توسط ویندوز ارائه نمی‌شوند. به عنوان مثال؛ برای انجام برخی از عملیات‌ها، از قبیل مخفی‌سازی ویژگی‌ها و تغییر پروسه‌ها باید از این رابط استفاده شود.
WSock32.dll وWs2_32.dll	این‌ها کتابخانه‌های مرتبط با شبکه هستند. یک برنامه که به شبکه متصل شود یا وظایفی مرتبط با شبکه را قصد دارد انجام دهد از توابع درون این کتابخانه‌ها استفاده می‌کند.
Wininet.dll	این کتابخانه شامل توابع پیاده‌سازی مفاهیم سطح بالا شبکه مانند پروتکل‌های Ftp, http و Ntp می‌شود.

قراردادهای نام‌گذاری توابع

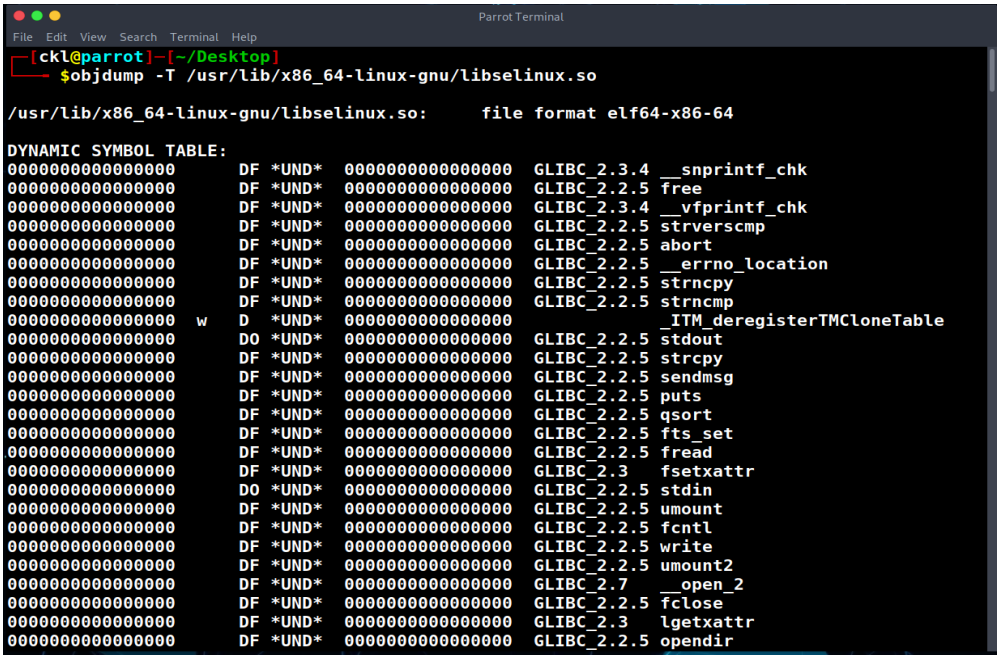
در هنگام ارزیابی توابع ویندوز، مقدار خیلی کمی از قراردادهای نام‌گذاری توابع وجود دارند که می‌توانند برای ما با ارزش باشند. آن‌ها اغلب ظاهر می‌شوند و ممکن است در صورت تشخیص اشتباه، تحلیلگر را گیج کنند. برای مثال، وقتی با نام توابعی با پسوند Ex از قبیل CreateWidnowsEx رو به رو می شوید، احتمال دارد کمی سردرگم شوید. به هر حال، وقتی مایکروسافت یک تابع را به‌روزرسانی می‌کند و تابع جدید با نوع قبلی ناسازگار است، مایکروسافت پشتیبانی خود را از تابع قدیمی همچنان ادامه می‌دهد. اما تابع جدید، نام تابع قدیمی را همراه با یک پسوند Ex می‌گیرد. برخی توابع خاص که نیازمند به‌روزرسانی دوباره هستند ۲ تا Ex می‌گیرند.

همچنین تعداد زیادی از توابع که رشته‌ها را به عنوان ورودی دریافت می‌کنند، در پایان نام خود شامل یک کاراکتر مانند A یا W هستند. مانند تابع CreateDirectoryW که در پایان W گرفته است. این حروف در مستندات توابع نمایش داده نمی‌شوند، بلکه این حروف نشان می‌دهند که تابع یک پارامتر رشته‌ای قبول می‌کند و ۲ نسخه مختلف از تابع وجود دارد. یکی برای رشته‌های اسکی (ASCII) و یکی برای رشته‌های کاراکتر عمومی (UNICODE) مورد استفاده قرار می‌گیرد. به یاد داشته باشید، جستجو برای توابعی با انتها A یا W در مستندات مایکروسافت بیهوده است.

توابع اکسپورت شده[1]

همانند توابع ایمپورت شده توسط فایل‌های اجرایی EXE و ELF، کتابخانه‌ها برای ارائه سرویس به فایل‌های اجرایی توابع را اکسپورت می‌کنند که روندی برخلاف فایل‌های اجرایی است. به عنوان مثال، معمولا یک کتابخانه (DLL) یا یک آبجکت اشتراکی (SO)، یک یا بیش از یک تابع دارد که برای استفاده توسط پروسه‌های اجرایی دیگر آن‌ها را اکسپورت می‌کنند. در نهایت، پروسه‌های اجرایی می‌توانند آن توابع اکسپورت شده را به خود ایمپورت کنند و مورد استفاده قرار بدهند.

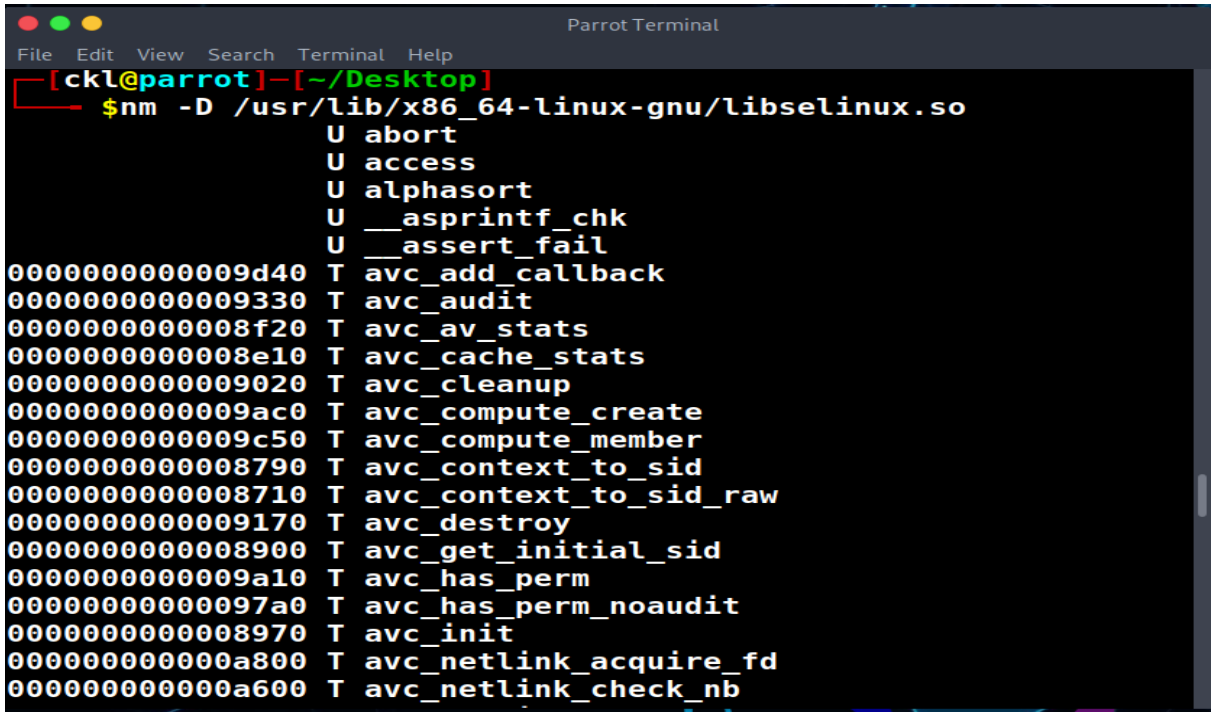
شایان ذکر است، همان‌طور که در قسمت‌های قبل مورد بررسی قرار گرفت؛ متخصصین تحلیلگر بدافزار می‌توانند اطلاعات اکسپورت شده توسط یک فایل‌اجرایی را با استفاده از برنامه Dependency Walker یا CFF Explorer در ویندوز و برنامه ELF Parser یا objdump در لینوکس به‌دست آورند. بدین منظور کافی است، روی نام فایلی که می خواهید اطلاعات آن را ببینید، کلیک کرده و به قسمت پانل شماره چهار که در تصویر ۱۰-۱ نمایش داده شده است، با دقت بنگرید. در این قسمت از برنامه Dependency Walker تمامی توابع اکسپورت شده لیست می‌شوند.



تصویر ۱۱-۱: توابع اکسپورت شده توسط یک فایل **SO** در لینوکس

[1] Exported Functions

در لینوکس هم مسائل تقریبا مشابه ویندوز است البته با اندکی تفاوت که طبیعی است. به عنوان مثال، در تصویر ۱۱-۱ توابع اکسپورت شده توسط کتابخانه libselinux.so با استفاده از فرمان objdump نمایش داده شده است، اگرچه ابزارهای گرافیکی هم با این محوریت وجود دارند که می‌توان از آن‌ها استفاده کرد. همچنین از فرمان nm هم می‌توان برای مشاهده این لیست استفاده کرد. در تصویر ۱۲-۱ استفاده از nm در لینوکس نمایش داده شده است.

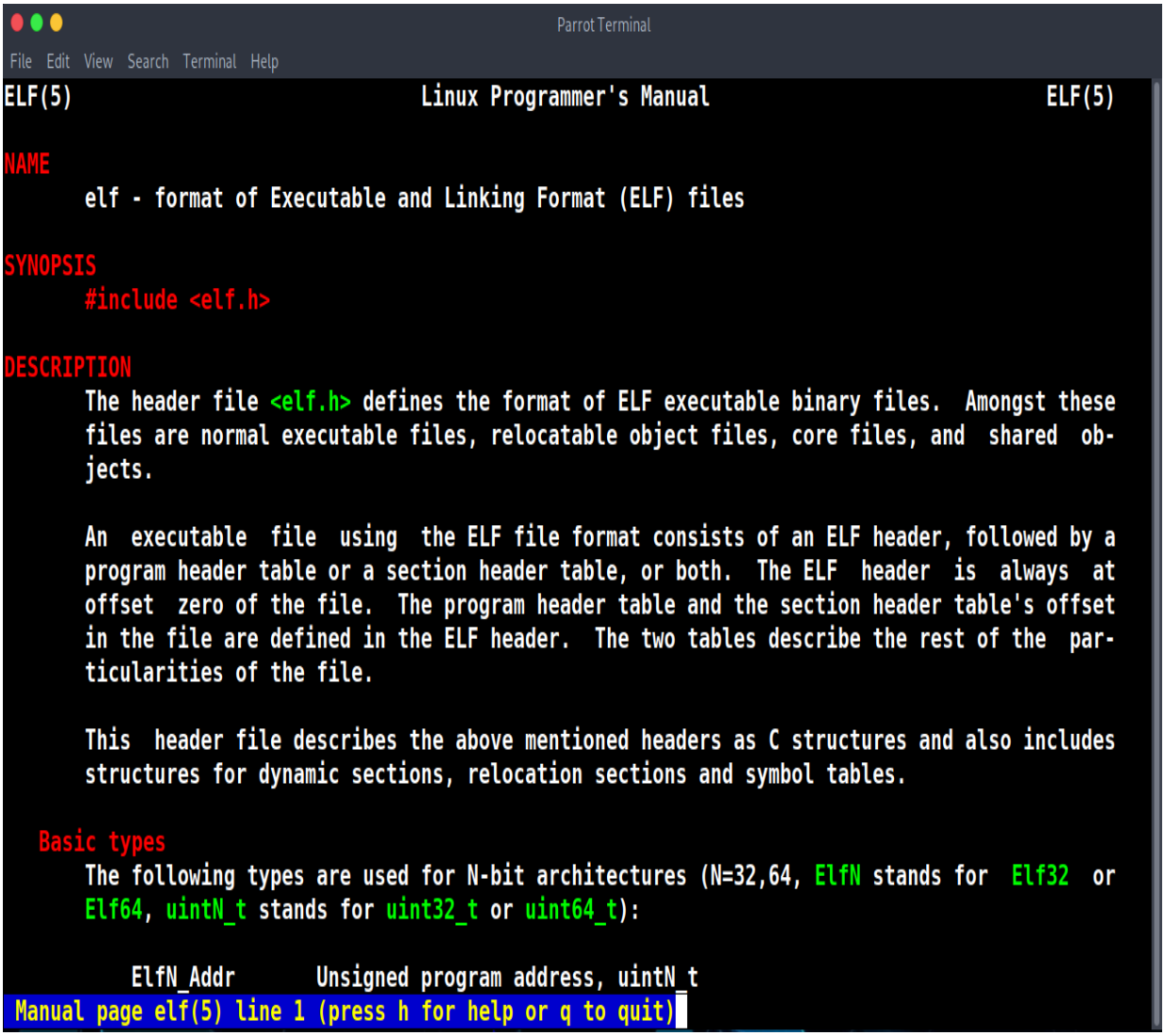


تصویر ۱۲-۱: توابع اکسپورت شده توسط یک فایل so در لینوکس

در تصویر ۱۲-۱، توابعی که با تگ T مشخص شده‌اند، در حقیقت سیمبول‌های اکسپورت شده کتابخانه اشتراکی libselinux.so در لینوکس هستند، و توابعی که با U تگ‌گذاری شده‌اند، سیمبول‌های مورد نیازی هستند که باید از آبجکت‌های اشتراکی دیگری بارگزاری شوند.

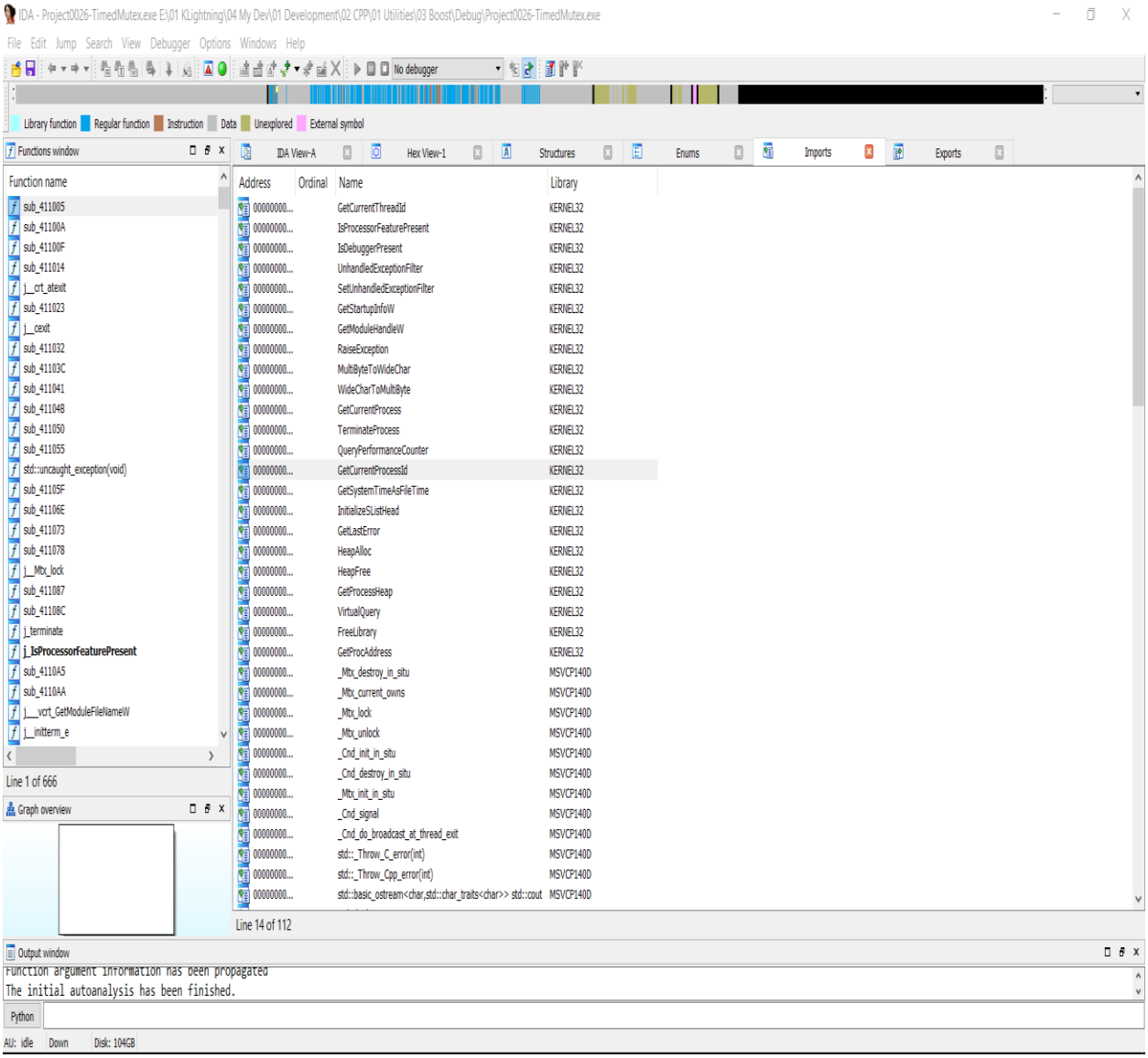
در لینوکس برای اطلاعات بیشتر در مورد nm و objdump می‌توانید در man مطالعه کنید. کافی است فرمان man objdump را اجرا کنید، تا راهنمای این برنامه و دیگر برنامه‌ها برای شما نمایش داده شود. همچنین برای مطالعه درباره هر کدام از توابع سیستمی هم می‌توانید از man استفاده کنید، به عنوان مثال برای مطالعه در مورد تابع abort می‌توانید با اجرای دستور man abort درباره آن مطالعه کنید. حتی

درباره فرمت فایل ELF که مخصوص لینوکس است، می‌توانید با اجرای دستور man elf راهنمای کامل آن را مشاهده و مطالعه کنید. در تصویر ۱۳-۱ صفحه راهنمای elf در لینوکس نمایش داده شده است.



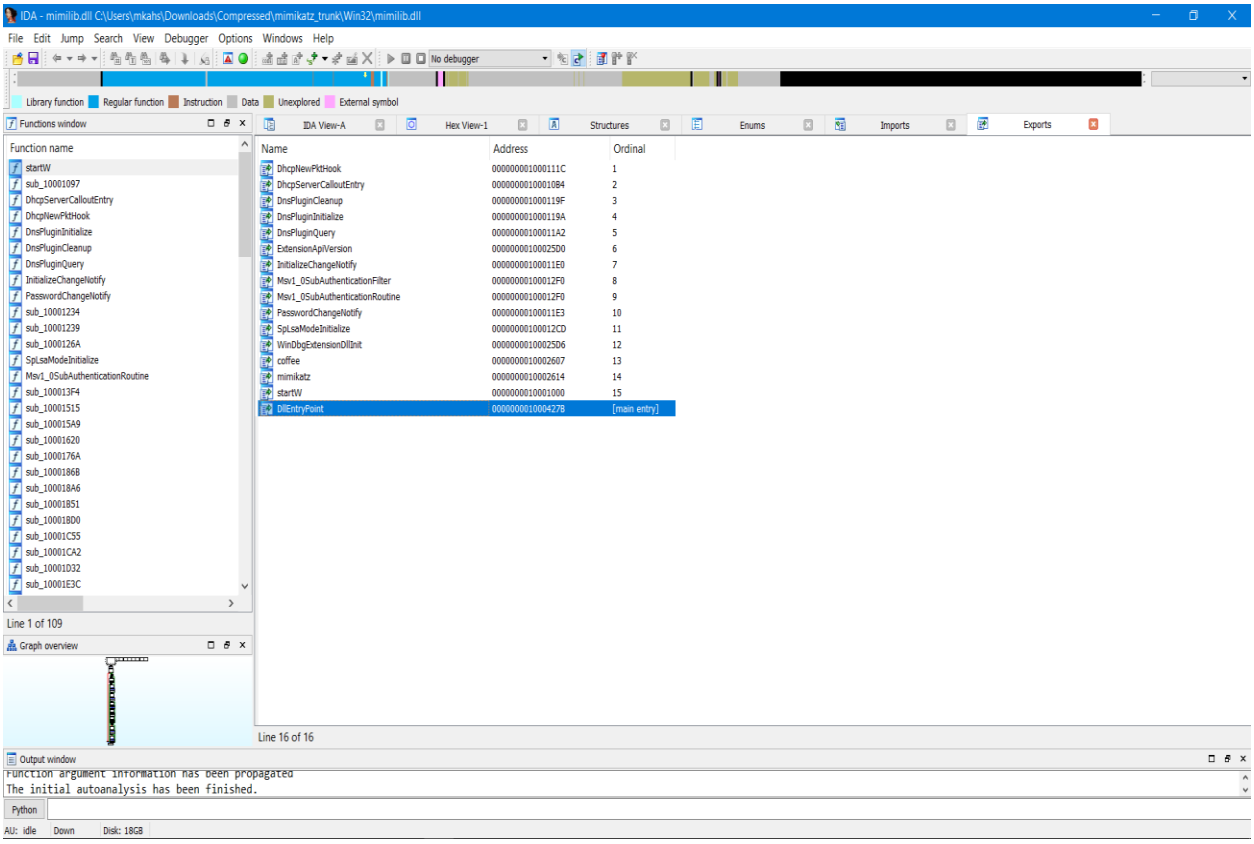
تصویر ۱۳-۱: راهنمای **ELF** در لینوکس

علاوه بر ابزارهایی که تاکنون معرفی شدند، در ویندوز و لینوکس و حتی مکینتاش می توانید از ابزارهایی مانند Ghidra یا IDA Pro یا Hopper Disassembler یا Radare2 برای مشاهده این دست از اطلاعات استفاده کنید. به عنوان مثال، در تصویر ۱۴-۱، برنامه IDA Pro را مشاهده می‌کنید که در پانل Imports لیست توابعی که توسط فایل TimedMutex.exe مورد استفاده قرار گرفته‌اند، نمایش داده است.



تصویر ۱۴-۱: توابع ایمپورت شده در IDA Pro

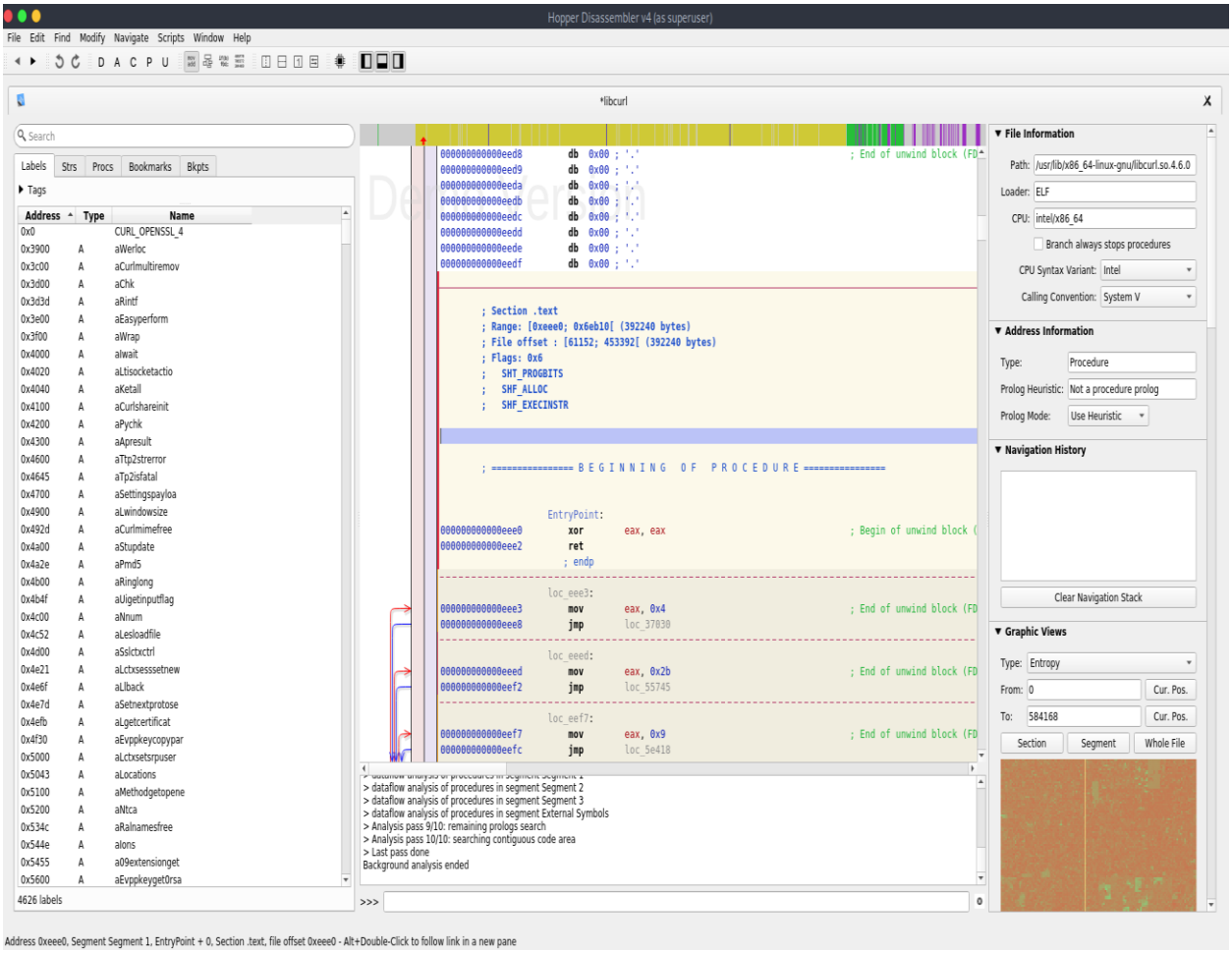
یا اگر یک کتابخانه Dll را در IDA Pro باز کنید و به پانل Exports بروید، لیست تمامی توابعی را که توسط Dll مذکور ارائه شده‌اند، خواهید دید. لیست توابع مذکور در روند تحلیل یک فایل باینری کمک شایانی خواهد کرد. در تصویر ۱۵-۱ لیست توابع اکسپورت شده توسط کتابخانه mimilib.dll نمایش داده شده است.



تصویر ۱۵-۱: توابع اکسپورت شده در IDA Pro

در محیط لینوکس از ابزار Hopper می‌توان استفاده کرد که مانند IDA Pro یک دیزاسمبلر قدرتمند و تجاری است. این ابزار تقریبا تمامی ویژگی‌هایی که IDA Pro ارائه می‌دهد، شامل می‌شود. در محیط لینوکس استفاده از این ابزار می‌تواند کار را برای تحلیل یک باینری و استخراج اطلاعات از آن بسیار ساده‌تر کند.

علاوه بر اینکه این ابزارها ساختار فرمت فایل اجرایی را پارز می‌کنند و اطلاعات هر بخش را نمایش می‌دهند، این امکان را هم فراهم می‌کنند که بتوانید کدهای سکشن .text را دیزاسمبل و تحلیل کنید. درباره مبحث دیزاسمبلی در آینده بیشتر صحبت خواهیم کرد. در تصویر ۱۶-۱ محیط ابزار Hopper نمایش داده شده است که یک آبجکت اشتراکی با عنوان libcurl را پارز کرده است و در قسمت Labels، سیمبول هایی که توسط این کتابخانه اشتراک گذاشته شده است، نمایش داده شده است.

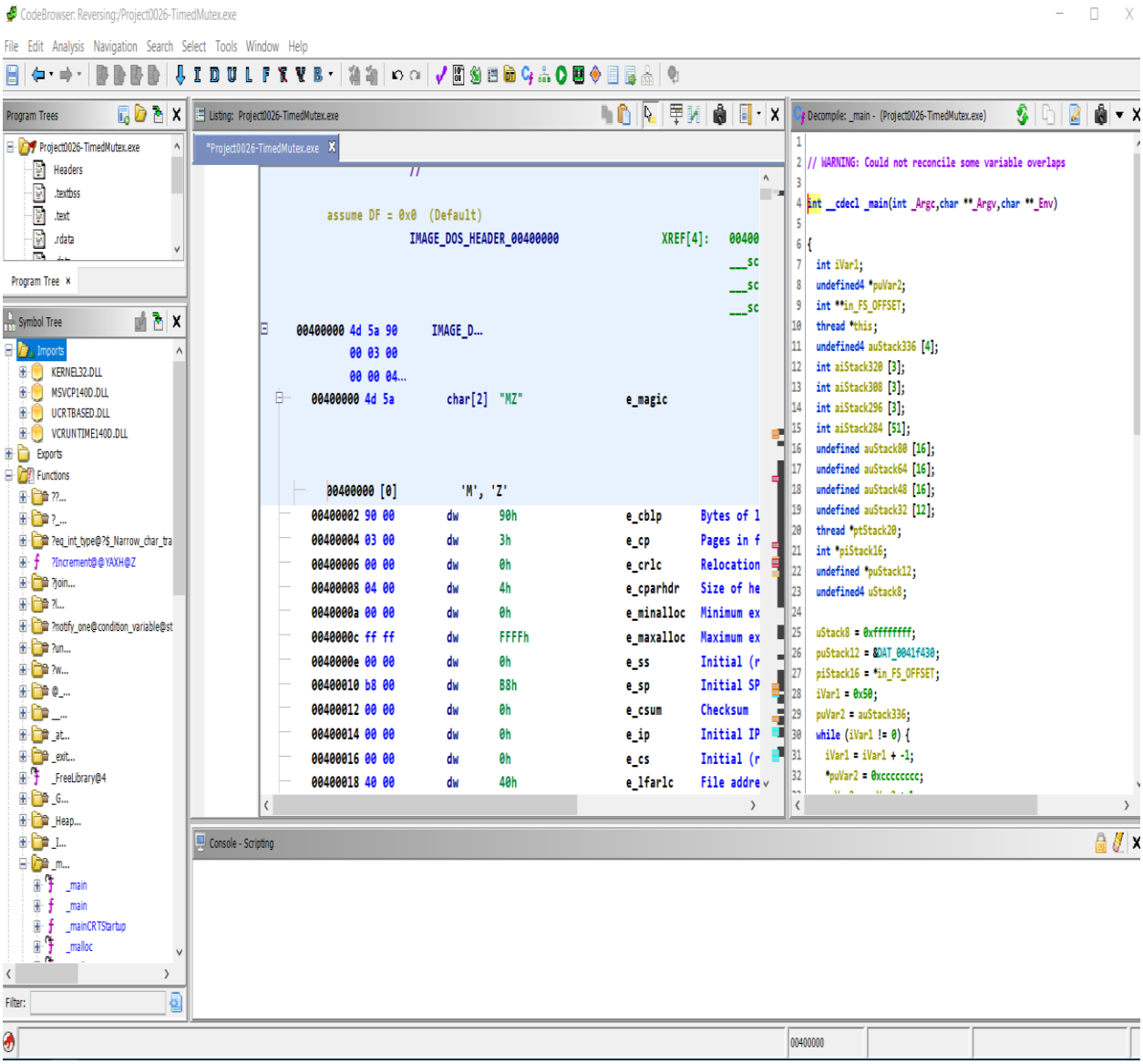


تصویر ۱۶-۱: نمایش سیمبول‌های اکسپورت شده توسط **Hopper**

یکی دیگر از ابزارهای بسیار قدرتمند که در ادامه این سلسله مقالات به صورت کامل آن را مورد بررسی و استفاده قرار خواهیم داد، دیزاسمبلر و دیکامپایلر Ghidra است. این ابزار که توسط آژانس امنیت داخلی (NSA) ایالات متحده آمریکا توسعه داده شده است، علاوه بر اینکه دیزاسمبلر قدرتمندی است، دارای یک دیکامپایلر هم هست.

همچنین این ابزار را می‌توانید بر روی پلتفرم ویندوز و لینوکس مورد استفاده قرار بدهید، چون کاملا قابل حمل طراحی شده است. در تصویر ۱۷-۱، محیط این نرم‌افزار نمایش داده شده است. این ابزار اطلاعات بسیار گسترده‌ای درباره یک باینری به ما ارائه می‌دهد. به عنوان مثال، اگر به پانل Symbol Tree نگاه کنید، مشاهده خواهید کرد که این نرم‌افزار تمامی توابع ایمپورت شده و توابع تعریف شده در خود برنامه و ... را لیست کرده است.

در قسمت بالای پانل Symbol Tree پانل دیگری با عنوان Program Trees وجود دارد که سکشن‌های درون یک باینری را نمایش می‌دهد. هر سکشن اطلاعات منحصربفردی را شامل می‌شود که در ادامه به صورت مشخص این سکشن‌ها در محیط ویندوز معرفی خواهیم کرد.



تصویر ۱۷-۱: محیط برنامه Ghidra

در ادامه این مقالات، به صورت کامل و عملی تمامی این ابزارها را در موقعیت‌های گوناگون مورد بررسی قرار خواهیم داد تا با عملکرد هر یک از آنها بهتر آشنا شویم.

تحلیل عملی استاتیک

حال که درباره تحلیل استاتیک اطلاعات مقدماتی دارید، اجازه بدهید به صورت عملی بعضی از بدافزارهای واقعی را مورد آزمایش قرار بدهیم. در این قسمت ما یک کیلاگر و سپس یک بدافزار پک شده را مورد تحلیل قرار خواهیم داد.

PotentialKeylogger.exe یک فایل‌اجرایی پک نشده[1]

جدول ۴-۱ یک لیست از توابع را نشان می‌دهد که به برنامه PotentialKeylogger.exe ایمپورت شده‌اند. این لیست از طریق برنامه Dependency Walker به‌دست آمده است. با این حال به دلیل این‌که ما در خروجی تولیده شده توسط Dependency Walker تعداد زیادی توابع ایمپورت شده مشاهده می‌کنیم، می‌توانیم به سرعت نتیجه بگیریم که این فایل فشرده نشده است.

جدول ٤-١: خلاصه‌ای از فهرست کتابخانه‌ها و توابعی که به برنامه **PotentialKeylogger.exe** وارد شده‌اند.

Kernel32.dll	**User32.dll**	User32.dll (continued)
CreateDirectoryW	BeginDeferWindowPos	ShowWindow
CreateFileW	CallNextHookEx	ToUnicodeEx
CreateThread	CreateDialogParamW	TrackPopupMenu
DeleteFileW	CreateWindowExW	TrackPopupMenuEx
ExitProcess	DefWindowProcW	TranslateMessage
FindClose	DialogBoxParamW	UnhookWindowsHookEx
FindFirstFileW	EndDialog	UnregisterClassW
FindNextFileW	GetMessageW	UnregisterHotKey
GetCommandLineW	GetSystemMetrics	GDI32.dll
GetCurrentProcess	GetWindowLongW	GetStockObject
GetCurrentThread	GetWindowRect	SetBkMode
GetFileSize	GetWindowTextW	SetTextColor
GetModuleHandleW	InvalidateRect	Shell32.dll
GetProcessHeap	IsDlgButtonChecked	CommandLineToArgvW
GetShortPathNameW	IsWindowEnabled	SHChangeNotify
HeapAlloc	LoadCursorW	SHGetFolderPathW

[1] PotentialKeylogger.exe: An Unpacked Executable

HeapFree	LoadIconW	ShellExecuteExW
IsDebuggerPresent	LoadMenuW	ShellExecuteW
MapViewOfFile	MapVirtualKeyW	Advapi32.dll
OpenProcess	MapWindowPoints	RegCloseKey
ReadFile	MessageBoxW	RegDeleteValueW
SetFilePointer	RegisterClassExW	RegOpenCurrentUser
WriteFile	RegisterHotKey	RegOpenKeyExW
	SendMessageA	RegQueryValueExW
	SetClipboardData	RegSetValueExW
	SetDlgItemTextW	
	SetWindowTextW	
	SetWindowsHookExW	

مانند تمامی برنامه‌های بزرگ و متوسط، این فایل‌اجرایی هم شامل تعداد زیادی از توابع ایمپورت شده است. متاسفانه، فقط برخی از این توابع در تحلیل بدافزار با ارزش هستند. به همین دلیل در طول این مقالات، ما فقط توابع ایپورت شده به بدافزار که از دید یک تحلیلگر با ارزش هستند را نشان خواهیم داد و روی آن‌ها متمرکز خواهیم شد.

هنگامی‌که یک تحلیلگر مطمئن نیست یک تابع چه کاری در برنامه انجام می‌دهد، باید وقت زیادی را صرف کند تا عملکرد آن را متوجه شود. البته می‌تواند به مستندات مایکروسافت رجوع کند و اطلاعات آن تابع را مورد مطالعه قرار دهد، اما باید این نکته را در نظر گرفت که تمامی توابع API در MSDN مایکروسافت یا مستندسازی نشده‌اند. اگرچه این مسئله در لینوکس به صورت دیگری است و تمامی توابع سیستمی در راهنمای لینوکس (Manual Page) مستندسازی شده‌اند.

شایان ذکر است، به عنوان یک تحلیلگر تازه کار بدافزار، وقت زیادی را صرف توابعی خواهید کرد که دارای ارزش خاصی نیستند. اما به زودی خواهید آموخت که چه توابعی مهم هستند و چه توابعی در تحلیل بدافزار به کار شما نمی‌آیند. مثلا در این بدافزار، ما تعداد بسیار زیادی از توابع ایمپورت شده را مشاهده می‌کنیم که برای ما ارزش خاصی ندارند.

همچنین معمولا، ما نمی‌دانیم که بدافزار مد نظر ما (PotentialKeylogger.exe) یک کیلاگر است. به همین دلیل نیاز داریم که به دنبال توابعی بگردیم که اطلاعاتی در مورد عملکرد این فایل اجرایی به ما ارائه

بدهد. به همین دلیل فقط روی توابعی متمرکز خواهیم شد که ویژگی‌های عملیاتی نرم‌افزار را به ما ارائه می‌دهند.

در این مثال، توابع ایمپورت شده از کتابخانه Kernel32.dll به نرم‌افزار که در جدول ۴-۱ لیست شده‌اند می‌گویند که این نرم‌افزار می‌تواند با استفاده از توابعی مانند OpenProcess، GetCurrentProcess و GetProcessHeap پروسه‌هایی را باز کرده و در آن‌ها تغییرات ایجاد کند. همچنین این نرم‌افزار می‌تواند روی فایل‌ها مشابه همین عملیات‌ها را با استفاده از توابع ReadFile، CreateFile و WriteFile انجام بدهد. توابع FindFirstFile و FindNextFile هم برای ما با ارزش هستند، زیرا از این دو تابع برای جستجو در دیرکتوری‌های سامانه‌عامل استفاده می‌شوند.

در این قسمت توابع ایمپورت شده از کتابخانه User32.dll جالب‌تر هستند. همان‌طور که مشاهده می‌کنید تعداد زیادی از توابع دستکاری کننده رابط‌های گرافیکی از قبیل RegisterClassEx، SetWindowsText و ShowWindows در برنامه مورد استفاده قرار گرفته‌اند که گواه این است که برنامه مدنظر ما یک رابط گرافیکی هم دارد (اگرچه رابط گرافیکی ضرورتی ندارد به کاربر نمایش داده شود.).

تابع SetWindowsHookEx یکی از توابعی است که توسط جاسوس‌افزارها به منظور دزدیدن کلیدهای فشرده شده صفحه کلید توسط قربانی به صورت رایجی مورد استفاده قرار می‌گیرد. این تابع برخی کاربردهای مشروع دارد، اما اگر به نرم‌افزاری مشکوک شدید و مشاهده کردید که این تابع در آن مورد استفاده قرار گرفته است، می‌توانید به این نتیجه برسید که آن برنامه دارای خاصیت دزدین کلیدهای فشرده است.

تابع RegisterHotKey یکی دیگر از توابع جالب در این قسمت است. این تابع یک میانبر (مانند Ctrl + Shift + P) در سامانه‌عامل ویندوز ثبت می‌کند که هرگاه کاربر این ترکیب از کلیدها را فشرد برنامه کاربردی مطلع گردد و عملیات خاصی را انجام بدهد. مهم نیست که برنامه فعال باشد یا خیر، کلید میانبر کاربر را به این برنامه کاربردی خواهد آورد.

اما توابع ایمپورت شده از کتابخانه GDI32.dll توابع مرتبط با گرافیک هستند که به سادگی اثبات می‌کنند برنامه دارای یک رابط گرافیکی است. توابع ایمپورت شده از کتابخانه Shell32.dll به ما می‌گویند؛ این برنامه می‌تواند برنامه‌های دیگر را اجرا کند، که این یک مزیت برای برنامه‌های مخرب و برنامه‌های عادی یا مشروع است.

توابع ایمپورت شده از کتابخانه Advapi32.dll به ما می‌گویند که این برنامه از رجیستری استفاده می‌کند. به همین دلیل باید در برنامه به دنبال رشته‌هایی باشیم که شبیه به کلیدهای رجیستری هستند. رشته‌های رجیستری مانند یک آدرس دایرکتوری هستند. در این برنامه، ما رشته‌ای به تصویر Software\Microsoft\Windows\CurrentVersion\Run پیدا خواهیم کرد که یک کلید ریجیستری است. این کلید ریجیستری برای کنترل برنامه‌هایی می‌باشد که باید به صورت خودکار زمانی‌که سامانه‌عامل بارگذاری می‌شود، برنامه را اجرا کند. این کلید رجیستری بطور خیلی رایجی توسط بدافزارها مورد استفاده قرار می‌گیرد تا بدافزار بتواند حتی پس از راه‌اندازی مجدد سامانه‌عامل دوباره فعال شود و عملیات خود را از سر گیرد.

همچنین این فایل‌اجرایی دارای توابع اکسپورت شده LowLevelKeyboardProc و LowLevelMouseProc است. در مستندات مایکروسافت آورده شده است، روتین هوک LowLevelKeyboardProc یک تابع بازگشتی تعریف شده در یک کتابخانه یا برنامه کاربردی است که از تابع SetWindowsHookEx استفاده می‌کند تا مشخص سازند کدام توابع به منظور ایجاد یک رویداد مشخص فراخوانی می‌شوند، در این برنامه یک رویداد سطح پایین از صفحه کلید مد نظر قرار دارد. با این حال تابع SetWindowsHookEx هنگامی‌که رویداد سطح پاینی برای صفحه کلید رخ می‌دهد، فراخوانی می‌گردد.

مستندات مایکروسافت و برنامه‌نویس‌ها از نام LowLevelKeyboardProc در این حالت استفاده می‌کنند. ما قادر خواهیم بود در این مورد اطلاعات با ارزشی استخراج کنیم. با استفاده از اطلاعات توابع ایمپورت شده و اکسپورت شده ما می‌توانیم برخی نتیجه‌گیری‌های مشخص و برخی جزییات را درباره بدافزار دریافت کنیم. مثلا، به نظر می‌رسد فایل یک کیلاگر محلی باشد که از SetWindowsHookEx به منظور ذخیره کلیدهای فشرده شده صفحه کلید قربانی استفاده می‌کند. همچنین ما می‌توانیم دریافت کنیم که این برنامه دارای یک رابط گرافیکی است که به یک کاربر خاص نمایش داده می‌شود.

همچنین این فایل با استفاده از RegisterHotKey کلیدهای ترکیبی وارد شده را ذخیره می‌کند و رابط گرافیکی و کلیدهای ذخیره شده دسترسی ارائه می‌دهد. همچنین ما با استناد به Software\Microsoft\Windows\CurrentVersion\Run نتیجه می‌گیریم که برنامه در هنگام بارگذاری سامانه‌عامل اجرا می‌شود.

PackedProgram.exe : بن‌بست

جدول ۵-۱ لیست کامل توابع ایمپورت شده به نرم‌افزار مخرب دوم یعنی PackedProgram.exe را نشان داده است. مختصر بودن لیست توابع نمایش داده شده در جدول ۴-۱ به ما می‌گوید که برنامه از تکنیک‌های مبهم‌سازی یا پکینگ استفاده می‌کند.

برای تایید این‌که برنامه واقعا مبهم‌سازی یا پک شده است، این موضوع کافی است که در برنامه هیچ رشته خواندنی وجود ندارد زیرا کامپایلرهای ویندوزی یک برنامه را با چنین لیست کمی از توابع ایجاد نمی‌کنند، حتی برنامه Hello world که ساده‌ترین نوع برنامه است، شامل توابع بیشتری نسبت به توابع این برنامه می‌شوند. با این حال شواهد حکم می‌کند که این برنامه مبهم‌سازی شده است.

جدول ۵-۱: کتابخانه‌های پیوندی پویا و توابع وارد شده به برنامه **PackedProgram.exe**

Kernel32.dll	User32.dll
GetModuleHandleA	MessageBoxA
LoadLibraryA	
GetProcAddress	
ExitProcess	
VirtualAlloc	
VirtualFree	

اطلاع از اینکه این برنامه پک و مبهم سازی شده است، ارزشمند می‌باشد اما در این شرایط نمی‌توانیم چیز زیادی در مورد ساختار اصلی برنامه با استفاده از روش تحلیل استاتیک ساده متوجه شویم و در آخر به بن‌بست می‌رسیم. در این نوع موارد ما نیاز داریم که از تکنیک‌های تحلیل پیشرفته مانند تحلیل دینامیک پیشرفته یا تکنیک‌های آنپکینگ (Unpacking) استفاده کنیم.

بخش‌ها و هدرهای فرمت فایل اجرایی

هدرهای فایل PE و ELF می‌توانند اطلاعات قابل ملاحظه‌ای را نسبت به توابع ایمپورت شده به ما ارائه دهند. فایل فرمت PE و ELF شامل یک هدر با دنباله‌ای از سکشن‌ها می‌شود. هدر شامل متادیتا درباره خود فایل است. به دنبال هدر سکشن‌های واقعی فایل قرار می‌گیرند و هر قسمت شامل اطلاعات کاربردی و مفیدی است.

هر چه در مطالعه این مقالات بیشتر پیشروی کنیم، درباره استراتژی‌های گوناگون به منظور مشاهده اطلاعات هر کدام از این قسمت‌ها در یک فایل فرمت بیشتر بحث و گفتگو خواهیم کرد. بخش‌های آورده شده در زیر مهم‌ترین و جالب‌ترین قسمت‌های یک فایل‌اجرایی قابل حمل PE و ELF هستند.

جدول ٦-١: تشریح سکشن‌های یک باینری

بخش‌ها	توضیحات
.text	بخش text شامل اینستراکشن‌ها می‌شود که پردازنده آن‌ها را اجرا می‌کند. مابقی بخش‌ها فقط داده‌ها را ذخیره‌سازی و از اطلاعات پشتیبانی می‌کنند. معمولا این قسمت تنها بخشی است که اجرا می‌شود و به همین دلیل این قسمت تنها بخشی است که باید دستورالعمل‌های برنامه را شامل شود.
.rdata	بخش rdata معمولا شامل اطلاعات ایمپورت و اکسپورت شده می‌شود که در حقیقت شامل همان اطلاعاتی است که در برنامه‌های Dependency Walker و PEview مشاهده می‌کنید. این بخش همچنین می‌تواند داده‌های فقط خواندنی استفاده شده در برنامه را هم ذخیره‌سازی کند. گاهی اوقات یک فایل شامل بخش‌های .idata و .edata خواهد شد که اطلاعات ایمپورت و اکسپورت شده را در خود ذخیره می‌کنند. (جدول ۱-۴ را ببینید)
.data	بخش data شامل داده‌های عمومی برنامه می‌شود که از هرجایی در برنامه قابل دسترس هستند. در این بخش یا هرجایی از فایل PE داده‌های محلی نمی‌توانند ذخیره شوند.
.rsrc	بخش rsrc شامل منابع مورد استفاده برنامه اجرایی می‌شود، مانند آیکون‌ها، منوها، تصاویر و رشته‌ها که نمی‌توان آن‌ها را یک بخش از برنامه اصلی در نظر گرفت. شایان ذکر است، رشته‌ها می‌توانند در بخش Rsrc یا در برنامه اصلی ذخیره شوند، ولی اغلب اوقات در برنامه‌های چند زبانه رشته‌ها در قسمت .rsrc ذخیره می‌شوند.

نام بخش‌ها در اغلب اوقات مطابق با نوع کامپایلر است و می‌تواند در هر کامپایلر متفاوت باشد. به عنوان مثال، Visual Studio از نام text برای قسمت اجرایی برنامه استفاده می‌کند، اما Borland Delphi از نام CODE برای این قسمت استفاده می‌کند.

با این حال ویندوز و لینوکس هیچ اهمیتی به این نام‌ها نمی‌دهد، زیرا از سایر بخش‌های مختلف خود فرمت فایل اجرایی پلتفرم خود استفاده می‌کنند تا مشخص سازند هر بخش چه کاری انجام می‌دهد. علاوه بر این‌ها، اسامی بخش‌ها برخی اوقات مبهم‌سازی می‌شوند تا تحلیل را سخت کنند. اما خوشبختانه در اغلب اوقات از نام‌های پیشفرض که در نظر گرفته شده استفاده می‌شود که در جدول ۵-۱ به همراه توضیح مختصری آورده شده است.

جدول ۷-۱: توضیحات بخش‌های مختلف فایل PE

بخش‌ها	توضیحات
.text	شامل کد اجرایی برنامه می‌شود.
.rdata	داده‌های فقط خواندنی را نگه می‌دارد که به صورت عمومی از همه جای برنامه قابل دسترس هستند.
.data	داده‌های عمومی که از سرتاسر برنامه قابل دسترس هستند را نگه می‌دارد.
.idata	گاهی اوقات ظاهر می‌شود و اطلاعات توابع ایمپورت شده را ذخیره می‌کند؛ اگر این بخش نمایش داده نشود اطلاعات توابع ایمپورت شده در بخش rdata ذخیره می‌شود.
.edata	گاهی اوقات ظاهر می‌شود و اطلاعات توابع اکسپورت شده را ذخیره می‌کند؛ اگر این بخش نمایش داده نشود اطلاعات توابع اکسپورت شده در بخش rdata ذخیره می‌شود.
.pdata	در برنامه‌های اجرایی ۶۴ بیتی فقط نمایش داده می‌شود و اطلاعات هندل اکسپشن‌ها[11] را ذخیره می‌کند.
.rsrc	در این بخش منابع مورد نیاز برنامه اجرایی ذخیره می‌شود.
.reloc	این بخش شامل اطلاعات فایل‌های کتابخانه‌ای جابه‌جا شده می‌شود.

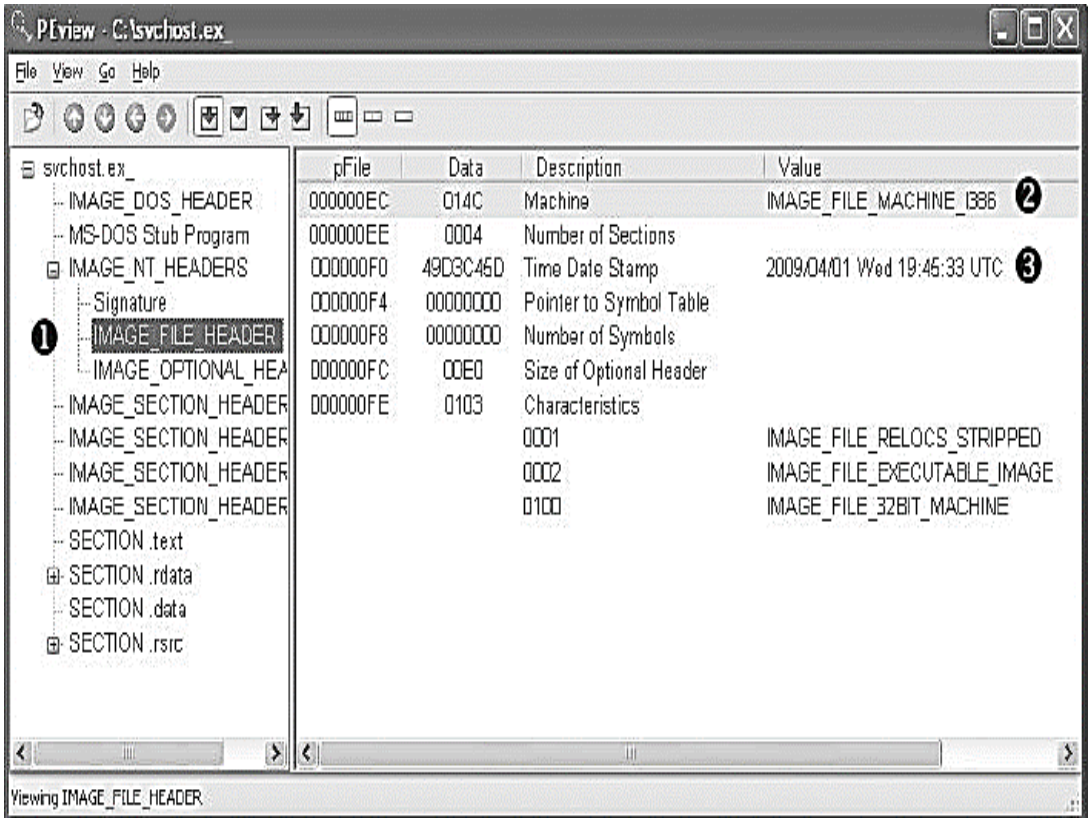
[11] Exception handling

بررسی فایل‌های PE

فایل فرمت‌اجرایی قابل حمل (PE) اطلاعات جالبی را در هدر خود ذخیره‌سازی می‌کند. بدین ترتیب ما می‌توانیم از برنامه PEview یا CFF Explorer به منظور جستجوی این اطلاعات استفاده کنیم. محیط این برنامه در تصویر ۱۸-۱ نمایش داده شده است.

در تصویر ۱۸-۱، پانل سمت چپ (شماره ۱) قسمت‌های اصلی هدر فایل PE را نشان می‌دهد. همان‌طور هم که در تصویر مشاهده می‌کنید دو قسمت از فایل هدر PE یعنی IMAGE_DOS_HEADER و MS-DOS Stub Program نادیده گرفته شده‌اند، زیرا اطلاعات جالبی به ما ارائه نمی‌دهند و فقط نشان‌دهنده تاریخ هستند. بخش بعدی از هدر فایل PE، یعنی IMAGE_NT_HEADERS هدرهای NT را نمایش می‌دهد. در این قسمت گزینه Signature همواره مشابه بوده و می‌تواند نادیده گرفته شود.

ورودی IMAGE_FILE_HEADER برجسته و انتخاب شده و محتویات آن در پانل سمت راست (شماره ۲) ظاهر گشته است. این گزینه شامل اطلاعات اساسی درباره فایل می‌شود. مثلا، گزینه Time Date Stamp (شماره ۳) به ما می‌گوید این برنامه در چه زمان و تاریخی کامپایل شده است، این مورد در تحلیل بدافزار و پاسخگویی به حوادث غیر قابل پیش‌بینی بسیار ضروری و مفید است. به عنوان مثال، یک زمان کامپایل قدیمی به ما می‌تواند این را بفهماند که این یک حمله قدیمی بوده است و ضدویروس‌ها ممکن است شامل یک سیگنیچر برای ممانعت کردن از انتشار این بدافزار باشند. اما یک زمان کامپایل جدید گواه یک حمله جدید است و می‌تواند بالعکس این موضوع بر آن صادق باشد. یعنی احتمال دارد که سامانه‌های امنیتی (ضدویروس‌ها، فایروال) شامل هیچ اطلاعاتی به منظور ممانعت کردن از آن بدافزار نباشند.



تصویر ۱۸-۱: مشاهده IMAGE_FILE_HEADER در برنامه PEview

گاهی اوقات زمان و تاریخ کامپایل برنامه مسئله ساز است. مثلا تمامی برنامه‌های دلفی از تاریخ June 19, 1992 برای تاریخ کامپایل برنامه خود استفاده می‌کنند. به همین دلیل اگر این زمان را در قسمت Time Date Stamp مشاهده کردید، می‌توانید بگوئید که احتمالا برنامه قربانی با دلفی نوشته شده است.

متاسفانه در این حالت نخواهید دانست که برنامه دقیقا در چه زمانی کامپایل و عملیاتی شده است. علاوه بر این موضوع، همچنین نویسندگان بدافزار می‌توانند به آسانی زمان کامپایل برنامه را با یک مقدار تقلبی تنظیم کنند. با این حال اگر با یک زمان کامپایل بی معنا رو به رو شدید می‌توانید بگوئید که آن زمان تقلبی بوده است.

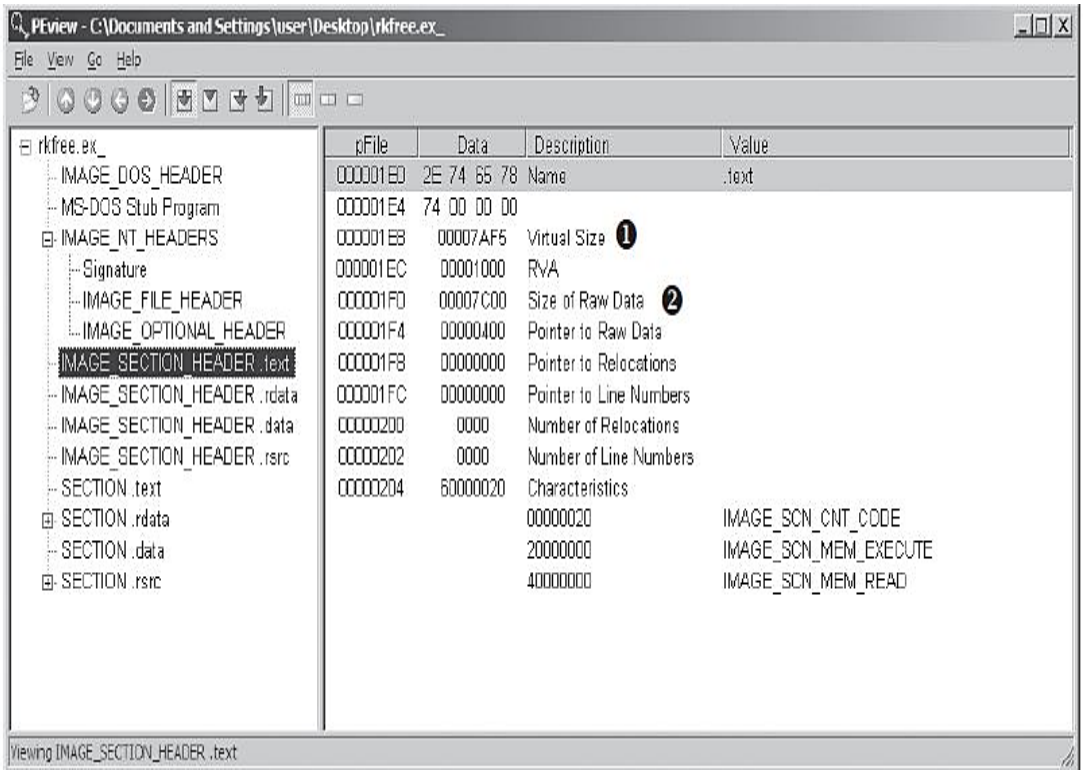
گزینه بعدی در پانل سمت چپ، گزینه IMAGE_OPTIONAL_HEADER است که شامل اطلاعات حساس مختلف دیگری می‌شود. هنگامی‌که شما روی این گزینه کلیک کنید، محتویات آن در صفحه سمت راست نمایش داده می‌شود.

در جدول Description پارامتر Subsystem مشخص می‌شود برنامه مبتنی بر خط فرمان است یا یک برنامه گرافیکی می‌باشد. مثلا برنامه اگر تحت خط فرمان (Console) باشد پارامتر Subsystem آن با مقدار IMAGE_SUBSYSTEM_WINDOWS_CUI مقداردهی می‌شود و اگر برنامه گرافیکی باشد، این پارامتر با IMAGE_SUBSYSTEM_WINDOWS_GUI مقداردهی می‌شود. همچنین در این قسمت‌ها زیرسامانه‌های غیررایج مانند Native یا Xbox استفاده می‌شود.

اما بیشترین اطلاعات جالب از هدرهای قسمت IMAGE_SECTION_HEADER به دست می آیند، که در تصویر ۱۹-۱ نمایش داده شده است. هر یک از این هدرها یک بخش از فایل PE را تشریح می‌کنند. عموما کامپایلرها نام بخش‌های یک فایل‌اجرایی را تولید می‌کنند و کاربر کنترل بسیار محدودی روی این نام‌ها دارد. در نتیجه، بخش‌ها از یک فایل‌اجرایی تا فایل‌اجرایی دیگر یک‌نواخت هستند و هر گونه تغییری مشکوک خواهد بود.

به عنوان مثال، در تصویر ۱۹-۱ پارامتر Virtual Size (شماره ۱) به ما می‌گوید چه مقدار فضا به یک بخش از برنامه در هنگام بارگذاری آن اختصاص داده شده است. پارامتر بعدی Size of Raw Data (شماره ۲) نشان‌دهنده میزان بزرگی بخش روی دیسک می‌باشد. این دو مقدار باید معمولا با هم برابر باشند، چون فایل هر مقداری که روی دیسک فضا اخذ کند، باید به همان اندازه نیز روی حافظه فضا بگیرد. با این حال تفاوت‌های اندک عادی است که حاکی از تفاوت‌های میان اختصاص حافظه و دیسک است.

اما این موضوع چه کمکی می‌تواند به ما بکند؟ می‌توان گفت که بخش اندازه‌ها می‌تواند در شناسایی بسته‌های اجرایی بسته‌بندی شده مفید باشد. به عنوان مثال، اگر مقدار Virtual Size بیشتر از مقدار Size of Raw Data باشد، خواهیم فهمید که این بخش مقدار بیشتری از حافظه را نسبت به فضای دیسک می‌گیرد و همین موضوع اغلب نشان‌دهنده بسته‌بندی شدن فایل‌اجرایی است، مخصوصاً اگر بخش text در حافظه از دیسک بیشتر باشد.



تصویر ۱۹-۱: محتوای بخش **IMAGE_SECTION_HEADER .text** را در برنامه **Peview** نمایش می‌دهد.

جدول ۸-۱ بخش‌های برنامه PotentialKeylogger.exe را نشان می‌دهد. همان‌طور که مشاهده می‌کنید، بخش‌های text، rdata و rsrc هر یک دارای یک مقدار تقریبا مشابه برای پارامترهای Virtual Size و Size of Raw هستند. اما بخش data به نظر مشکوک می‌رسد، زیرا پارامتر Virtual Size آن دارای مقدار بیشتری نسبت به Raw Data Size است، ولی این موضوع در برنامه‌های ویندوز برای قسمت data یک مورد طبیعی است. اما توجه کنید این اطلاعات نمی‌تواند به تنهایی به ما بگوید که برنامه مخرب نیست. این اطلاعات فقط به سادگی به ما می‌گویند که برنامه بسته‌بندی نشده است و هدر فایل PE توسط یک کامپایلر تولید شده است.

جدول ۸-۱: اطلاعات بخش‌های فایل **PotentialKeylogger.exe**

Section	Virtual size	Size of raw data
.text	7AF5	7C00
.data	17A0	0200
.rdata	1AF5	1C00
.rsrc	72B8	7400

جدول ۹-۱ بخش‌های برنامه PackedProgram.exe را نمایش می‌دهد. بخش‌ها این فایل دارای مقداری آنومالی هستند: نام بخش‌ها Dijfpds، sdfuok. و Kijijl است که برای ما غیرعادی هستند و بخش‌های text، data و rdata مشکوک به نظر می رسند. همان‌طور که مشاهده می‌کنید، بخش text برای Raw Data دارای مقدار ۰ است، این بدین معنی است که برنامه هیچ حافظه‌ای از دیسک اشغال نمی‌کند و مقدار Virtual Size آن A000 است، این بدین معنی است که برنامه مقدار A000 را به بخش text اختصاص داده است. در حالت کلی این اطلاعات به ما می‌گویند که یک برنامه باید این کد بسته‌بندی شده را از حالت بسته‌بندی خارج کند.

جدول ۹-۱: اطلاعات بخش‌های فایل **PackedProgram.exe**

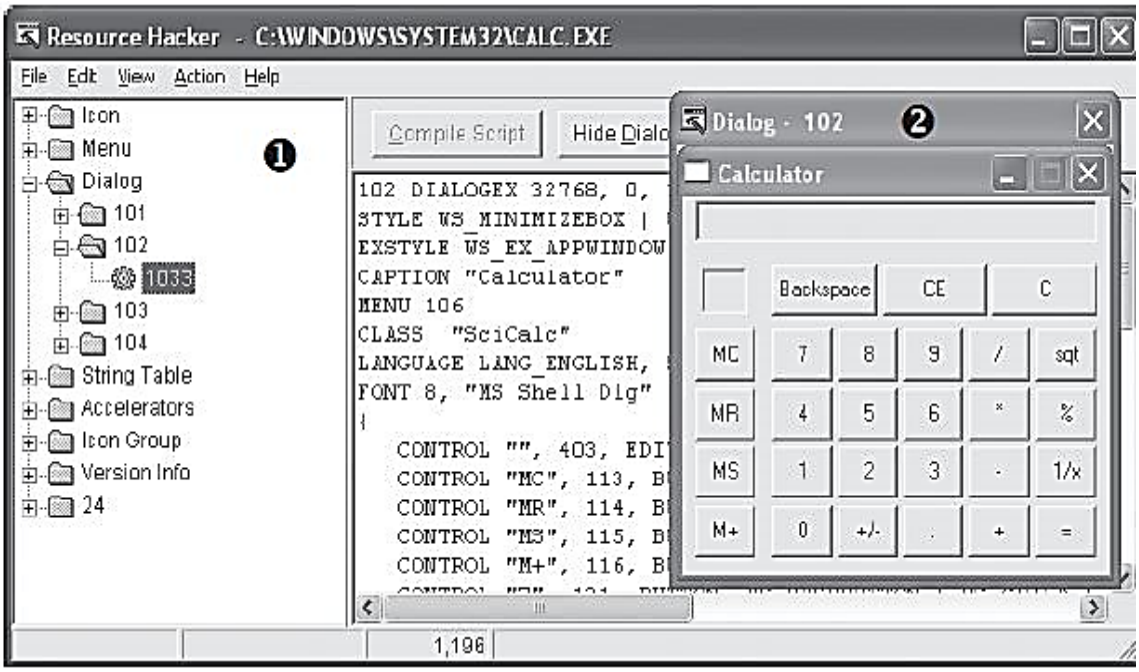
Name	Virtual size	Size of raw data
.text	A000	**0000**
.data	3000	**0000**
.rdata	4000	**0000**
.rsrc	19000	**3400**
Dijfpds	20000	**0000**
.sdfuok	34000	**3313F**
Kijijl	1000	**0200**

دیدن منابع بخش‌ها با استفاده از Resource Hacker

حال که مشاهده هدرهای فایل PE را به اتمام رسانده‌ایم، می‌توانیم به برخی از بخش‌های یک فایل PE نگاهی بیندازیم. تنها بخشی که می‌توانیم آن را بدون دانش اضافه از فصول بعدی بررسی کنیم، بخش منابع است. بدین منظور می‌توانید از ابزار Resource Hacker برای مرور محتویات بخش rsrc استفاده کنید.

این برنامه رایگان است و می‌توانید آن را از سایت http://www.angusj.com دانلود کنید. هنگامی‌که روی آیتم‌های درون این برنامه کلیک کنید، به سرعت رشته‌ها، آیکون‌ها و منوهای درون برنامه مدنظر خود را خواهید دید که می‌توانید با این برنامه در آن‌ها تغییرات اعمال کنید. قابل ذکر است، منوهای نمایش داده شده در این برنامه مطابق با منوهایی است که در برنامه اصلی تعبیه شده است. در هر حال در تصویر ۲۰-۱

برنامه Resource Hacker نمایش داده شده است که برنامه Caculator ویندوز را مورد بررسی قرار داده است.



تصویر ۲۰-۱: برنامه Resource Hacker در حال بررسی calc.exe

همان‌طور که در تصویر بالا مشاهده می‌کنید، در پانل سمت چپ تمامی منابع فایل‌اجرایی نمایش داده شده است. هر پوشه ریشه (Root Folder) که در پانل سمت چپ نشان داده شده است (شماره ۱) منابع مختلفی از برنامه را ذخیره می‌کند. با این حال فقط برخی از این بخش‌های نمایش داده شده در تصویر ۱۲-۱ برای ما در تحلیل بدافزار مهم هستند که در جدول ۱۰-۱ مختصرا توضیح داده شده‌اند.

جدول ۱۰-۱ : اطلاعات پوشه های ریشه برنامه‌ها در Resource Hacker

پوشه ریشه	توضیحات
Icon	این پوشه تمامی آیکون‌هایی که در برنامه هنگام اجرا نمایش داده می‌شوند را شامل می‌گردد.
Menu	این قسمت شامل تمامی منوهای درون برنامه می‌شود که در قسمت‌های گوناگون آن مورد نمایش قرار می‌گیرند. از قبیل منو File ، Edit و View که در تمامی برنامه‌های ویندوز وجود دارند. علاوه بر این، این بخش شامل نام همه منوها و همچنین

	متن نمایش داده شده برای هر یک از آن‌ها می‌شود. نام هر منو می‌تواند در مورد ویژگی‌های آن به شما اطلاعات بدهد.
Dialog	بخش دیالوگ برنامه شامل منوهای محاوره‌ای می‌شود. به عنوان مثال منو محاوره‌ای که در تصویر ۱–۸ (شماره ۲) نمایش داده شده است، نشان می‌دهد کاربر چه خواهد دید وقتیکه فایل calc.exe را اجرا کند. اگر ما اطلاعاتی در مورد برنامه calc.exe نداشته باشیم، به راحتی می‌توانیم با نگاه به منوی محاوره‌ای برنامه تشخیص دهیم که آن یک ماشین حساب است.
String	این قسمت اطلاعات رشته‌های برنامه را ذخیره می‌کند.
Version	این بخش شامل اطلاعات گزارشی برنامه مانند نسخه برنامه و اغلب نام شرکت سازنده برنامه و قوانین کپی رایت می‌شود.

بخش rsrc نشان داده شده در تصویر ۲۰–۱ نمونه ای از یک برنامه کاربردی ویندوز است و می‌تواند شامل هر آنچه که یک برنامه‌نویس می خواهد شود.

استفاده از دیگر ابزارهای PE

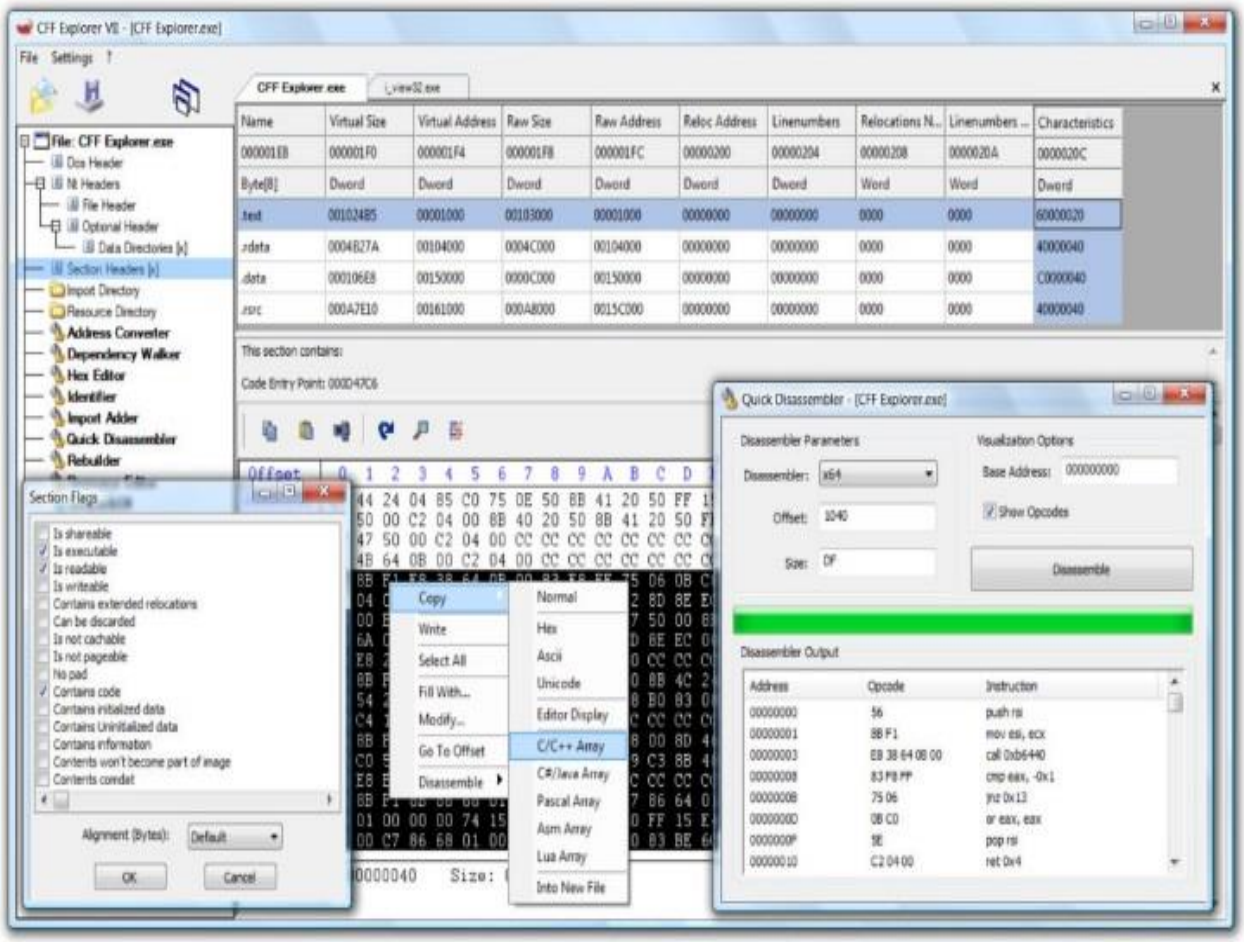
تعداد بسیار زیاد دیگری برنامه برای مرور بخش‌های فایل PE وجود دارد که دو تا از برترین آن‌ها در این قسمت معرفی می‌شوند. اولین آن‌ها ابزار PEBrowse Professional است، این برنامه بسیار از نظر ظاهر شبیه برنامه PEView می‌باشد. این برنامه به شما اجازه می‌دهد، به تمامی بایت‌های بخش یک فایل PE دسترسی داشته باشید. علاوه بر این ها ابزار PEBrowser Professional در نمایش اطلاعات بخش Rsrc که مخفف resource است، بسیار بهتر نسبت به دیگر ابزار ها عمل می نماید. از پیوند (http://goo.gl/y9nUGB) برای دانلود این برنامه استفاده کنید.

برنامه بعدی PE Explorer است. این برنامه دارای یک رابط کاربری فوق العاده کاربر پسند به منظور مرور کردن قسمت‌های گوناگون یک فایل PE است. با این برنامه می‌توانید برخی از قسمت‌های یک فایل PE را ویرایش کنید. این برنامه همچنین شامل یک ویرایشگر کدهای منبع می‌باشد که برای مرور کردن و ویرایش منابع یک فایل بسیار عالی است. تنها نقطه ضعف این برنامه این است که رایگان نیست و شما باید نسخه

کامل آن را خریداری کنید. برای دانلود نسخه ۳۰ روز آن به آدرس (http://goo.gl/4RgLh5) رجوع کنید.

نرم‌افزار CFF Explorer

نرم افزار CFF Explorer یک ابزار مشهور دیگر در زمینه مشاهده محتویات فایل‌های PE است که توسط Daniel Pistelli ایجاد شده است. این ابزار شامل ویژگی‌های کامل به منظور ویرایش بخش‌های منابع فایل‌های PE و انجام پویش سیگنیچرهای است. نرم افزار CFF Explorer از سامانه‌های x64 و x86 پشتیبانی می‌کند و می‌تواند فایل‌های .NET را بدون نصب .NET Framework کنترل کند. این نرم افزار فوق العاده کاربردی را می‌توانید از آدرس (http://goo.gl/X1gbie) دانلود کنید. در تصویر ۲۱-۱ محیط این نرم‌افزار کاربردی نمایش داده شده است.



تصویر ۲۱-۱ : تصویر محیط نرم‌افزار CFF Explorer

خلاصه هدر PE

هدر PE شامل اطلاعات مفیدی برای تحلیل کنندگان بدافزار است و در ادامه فصل‌های این کتاب بیشتر مورد بررسی قرار خواهد گرفت. جدول ۱۱-۱ اطلاعات کلیدی که می‌تواند از یک هدر PE دریافت شود را مجددا مورد بررسی قرار داده است.

جدول ۱۱-۱ : اطلاعات درون هدر PE

فیلد	اطلاعات نمایش دهنده
Imports	این فیلد شامل توابع دیگر کتابخانه‌ها می‌شود که توسط بدافزار مورد استفاده قرار می‌گیرد.
Exports	این فیلد شامل اطلاعات توابع در بدافزار است که توسط دیگر برنامه‌ها یا کتابخانه فراخوانی می‌شوند.
Time Date Stamp	شامل اطلاعات زمان و تاریخی می‌شود که بدافزار کامپایل شده است.
Sections	شامل نام بخش‌ها یک فایل و اندازه آن‌ها روی دیسک و حافظه می‌شود.
Subsystem	این فیلد مشخص می‌کند که برنامه تحت خط فرمان است یا دارای یک رابط گرافیکی می‌باشد.
Resources	این فیلد شامل اطلاعاتی از قبیل رشته، آیکون‌ها، تصاویر و... استفاده شده در فایل می‌شود.

نتیجه‌گیری

در این فصل آموختیم که چگونه می‌توانیم با استفاده از ابزارهای ساده، تحلیل استاتیک روی بدافزار انجام داده و به اندازه مشخصی از ویژگی‌ها و نوع کاربردهای آن اطلاعات جمع‌آموری کنیم. به هر صورت، تحلیل استاتیک بدافزار اولین گام در تحلیل بدافزار است و تحلیل‌های بیشتری در ادامه نیاز است. به همین دلیل ما برای تحلیل‌های قوی‌تر و دریافت اطلاعات کامل‌تر از بدافزارها یا جنگ‌افزارهای پیچیده نیازمند هستیم از تکنیک‌های تحلیل دینامیک استفاده کنیم. همان‌طور هم که در این فصل گفته شد، در روش تحلیل دینامیک بدافزار باید فایل‌اجرایی آن اجرا گردد. در گام بعدی ما یک محیط ایمن راه‌اندازی خواهیم کرد تا بتوانیم بدافزارها را بدونه آن‌که به سامانه اصلی ما آسیب برسانند، اجرا کرده و روی آنان با روش تحلیل دینامیک مطالعه خود را انجام دهیم. در فصل بعد این موضوع را خواهید آموخت.