معرفی بسته نرم‌افزاری Detours و کاربرد آن در جعل API

شهاب‌الدین نمازی‌خواه
namazikhah@cert.um.ac.ir

سوسن نادری
naderi@cert.um.ac.ir

آزمایشگاه تخصصی آپا در زمینه امنیت فناوری اطلاعات و ارتباطات

http://cert.um.ac.ir
cert@um.ac.ir

ویرایش اول – فروردین‌ماه ۱۳۹۳

شماره سند: APA_FUM_W_MAL_0115

چکیده

در این مقاله، کتابخانه Detours و چگونگی استفاده از آن شرح داده می‌شود. استفاده از ابزار Detours یکی از محبوب‌ترین روش‌ها برای جعل API ها است. همچنین نمونه‌هایی از روش کار کتابخانه‌ی Detours نیز به اختصار توضیح داده می‌شود.

واژه‌های کلیدی

جعل API، کتابخانه‌ی Detours، DLL، تابع Trampoline، تابع منحرف شده، تابع هدف.

۱- مقدمه

Detours کتابخانه‌ای برای ایجاد وقفه در توابع فایل‌های باینری است که بر روی ماشین‌های ARM، x86، x64، IA64 کار می‌کنند. Detours در اکثر موارد برای ایجاد وقفه در توابع win32 است که در برنامه‌های کاربردی مانند نرم‌افزارهای Debugging استفاده می‌شود. این وقفه‌ها به صورت پویا در زمان اجرا ایجاد می‌شود و Detours برای انجام این کار چند دستور ابتدایی تابع موردنظر را با یک دستور پرش غیرشرطی جا به جا می‌کند و باعث می‌شود که اجرای دستورات به سمت دستوراتی که کاربر نیاز دارد حرکت کند. این عمل باعث می‌شود که دستورات تابع موردنظر از بین برود، اما Detours برای جلوگیری از این موضوع دستورات حذف شده را در تابعی به نام Trampoline قرار می‌دهد و آدرس آن را در اشاره‌گری قرار می‌دهد. تابع جدید می‌تواند به جای تابع هدف قرار گیرد و یا با فراخوانی تابع هدف به عنوان یک زیرتابع از طریق آدرس اشاره‌گر به Trampoline امکانات آن را افزایش می‌دهد.

Detours در زمان اجرا عملیات خود را انجام می‌دهد. کد تابع هدف در حافظه تغییر می‌کند و نه در دیسک، بنابراین فعال کردن وقفه در تابع باینری در بهترین وجه از جزئیات صورت می‌گیرد. برای مثال، یک تابع در یک DLL ممکن است در اجرای یک برنامه مورد هدف قرار گیرد در صورتی که توابع اصلی در همان زمان در برنامه‌های اجرا شده‌ی دیگر مورد هدف قرار گرفته نشده باشند. برخلاف DLL relinking یا ارجاع ایستا، تکنیک‌های ایجاد وقفه‌ای که در کتابخانه‌ی Detours استفاده شده‌اند فارغ از روش‌های مورد استفاده در برنامه یا کدهای سیستمی برای هدف قرار دادن توابع عمل می‌کنند.

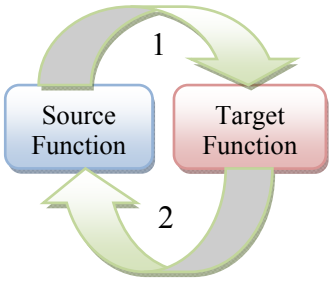
علاوه بر قابلیت‌های پایه‌ای توابع منحرف شده[1]، کتابخانه Detours شامل توابعی برای ویرایش جدول ورودی DLL هر فایل باینری است تا بتوانند هر نوع سگمنت داده‌ی دلخواه را به باینری موجود اضافه کند و DLL را در یک پردازه جدید بارگذاری کند. به محض این که فایل در یک پردازه بارگذاری شد، ابزار DLL می‌تواند تمام توابع موجود در پردازه را چه در برنامه و چه در کتابخانه سیستم مانند API های ویندوز مورد هدف قرار دهد.

[1] Detoured Function

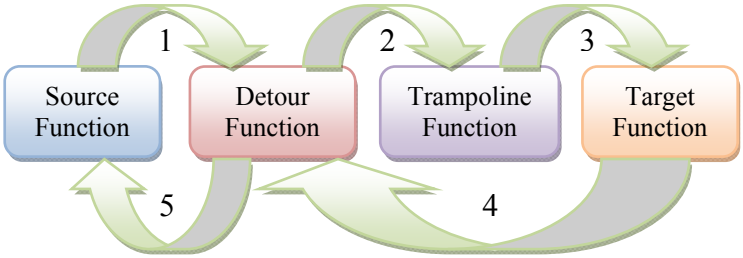
۲- ایجاد وقفه در توابع باینری

کتابخانه Detours ایجاد وقفه در فراخوانی توابع را ممکن می‌سازد. کد وقفه در زمان اجرا به صورت پویا اعمال می‌شود. Detours چند دستور ابتدایی تابع هدف[1] را با یک دستور پرش غیر شرطی به تابعی که کاربر تعریف کرده است جابه‌جا می‌کند. دستوراتی که از تابع هدف حذف شده‌اند در تابع Trampoline نگهداری می‌شوند. تابع Trampoline شامل دستورات حذف شده از تابع هدف و شاخه‌ی غیر شرطی به ادامه دستورات تابع هدف می‌باشد.

زمانی که اجرای برنامه به نقطه شروع تابع هدف می‌رسد کنترل‌ها مستقیم به تابعی که کاربر تعریف کرده است پرش می‌کنند. تابع منحرف شده هر وقفه را در پیش‌پردازه‌ی مناسب خود اجرا می‌کند. تابعی که توسط کاربر تغییر کرده است می‌تواند کنترل را به تابع اصلی برگرداند یا تابع Trampoline را فراخوانی کند که دستورات تابع هدف را بدون ایجاد وقفه در خود دارد. زمانی که اجرای تابع هدف به پایان رسید کنترل را به تابعی که تغییر کرده است برمی‌گرداند تا عملیات متناسب با پس‌پردازه‌ها را انجام دهد و پس از آن کنترل را به تابع اصلی برمی‌گرداند. شکل ۱ تصویری منطقی از جریان کنترل را برای فراخوانی توابع بدون ایجاد وقفه و شکل ۲ فراخوانی توابع با ایجاد وقفه را نشان می‌دهد.



شکل (۱): فراخوانی تابع بدون ایجاد وقفه

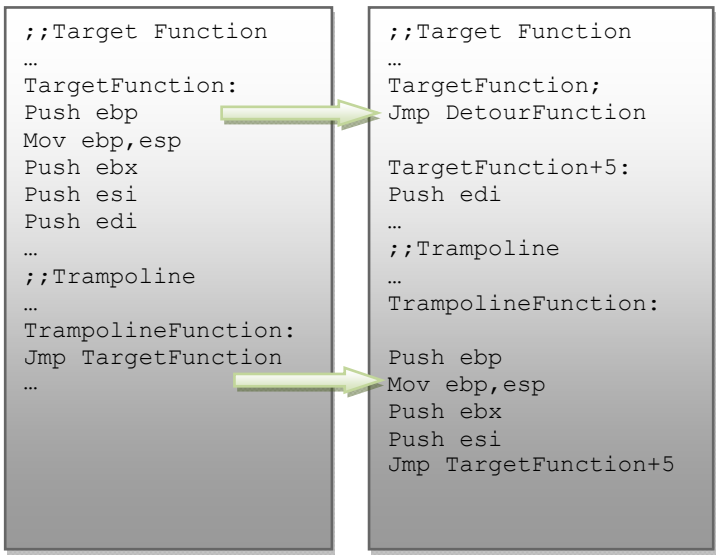


شکل (۲): جریان کنترل برای فراخوانی توابع با وقفه

[1] Target Function

کتابخانه‌ی Detours از طریق بازنویسی باینری در پردازه وقفه‌ها را در تابع هدف ایجاد می‌کند. کتابخانه Detours در حقیقت برای هر تابع هدف دو تابع را بازنویسی می‌کند که یکی از آن‌ها خود تابع هدف و دیگری تابع Trampoline متناسب با آن است و پس از آن اقدام به بازنویسی اشاره‌گر تابع هدف می‌کند. تابع Trampoline به صورت پویا توسط کتابخانه Detours تخصیص داده می‌شود. با توجه به درج تابع منحرف شده، تابع Trampoline تنها یک دستور پرش به تابع هدف دارد. بعد از درج آن، Trampoline دستورات ابتدایی تابع هدف و یک دستور پرش به ادامه تابع هدف را خواهد داشت.

اشاره‌گر به تابع هدف توسط کاربر مقداردهی می‌شود. بعد از این که تابع منحرف شده به تابع هدف پیوست شد، اشاره‌گر به تابع هدف دیگر به تابع Trampoline اشاره می‌کند و پس از این که تابع منحرف شده از تابع هدف جدا شد اشاره‌گر مقدار خود را برای اشاره به تابع اصلی تنظیم می‌کند.



شکل (۳): تابع موردنظر و Trampoline قبل و بعد از قرار دادن تابع منحرف شده

شکل ۳ اضافه کردن تابع منحرف شده را نشان می‌دهد. برای تغییر دادن تابع هدف کتابخانه Detours ابتدا حافظه‌ی تابع پویای Trampoline را اختصاص می‌دهد و سپس امکان دسترسی نوشتن را برای هر دو تابع هدف و Trampoline فراهم می‌کند. با شروع از دستورات اولیه Detours حداقل ۵ بایت از تابع هدف را در تابع Trampoline کپی می‌کند (این فضا برای یک پرش غیرشرطی کافی است) و در صورتی که تابع کمتر از ۵ بایت

داشته باشد Detours عملیات را متوقف کرده و یک کد خطا برمی‌گرداند.

برای کپی کردن دستورات، Detours از یک Disassembler جدول‌محور ساده استفاده می‌کند و سپس در انتهای تابع Trampoline یک دستور پرش به دستورات کپی نشده در تابع هدف اضافه می‌کند. Detours دستورات پرش غیر شرطی در تابع هدف را به تابع منحرف شده ایجاد می‌کند تا دستورات آن به عنوان دستورات اولیه تابع هدف در نظر گرفته شود. در پایان Detours مجوز صفحات اصلی را برای هر دو تابع هدف و Trampoline بازسازی می‌کند و آن بوسیله فراخوانی دستور Flush-InstructionCache در کش دستورات CPU را قرار می‌دهد.

۳- استفاده از Detours

برای ایجاد تغییر در تابع هدف دو مورد نیاز است: اشاره‌گر هدف[1] که آدرس تابع هدف را در خود دارد و تابع منحرف شده. برای ایجاد وقفه مناسب، تابع هدف، تابع منحرف شده و اشاره‌گر هدف باید دقیقاً دارای یک امضای فراخوانی باشند که شامل تعداد آرگومان‌ها و قرارداد یکسان است. استفاده از یک فراخوانی قرارداد این اطمینان را ایجاد می‌کند که رجیسترها به خوبی حفظ می‌شوند و پشته به صورت مناسب بین تابع جدید و تابع هدف همتراز می‌شود.

٤- نمونه‌های استفاده از کتابخانه Detours در جعل API

در این بخش چگونگی استفاده از کتابخانه Detours را در سه مورد متفاوت شرح می‌دهیم.

٤-١- استفاده از کتابخانه‌ی Detours در مواردی که متن برنامه[2] موجود است

برنامه‌ی Simple ساده‌ترین نمونه از DLL ای است که در آن از کتابخانه‌ی Detours استفاده می‌شود به گونه‌ای که API ویندوز را تغییر می‌دهد و قابلیت‌هایی را به آن اضافه می‌کند. در این مثال تابع Sleep به گونه‌ای تغییر داده شده است که تعداد تیک‌های ساعت گذرانده شده در زمان خواب بودن سیستم را ثبت می‌کند.

[1] Target Pointer
[2] Source Code

```
1       #include <windows.h>
2       #include <detours.h>
3
4       static LONG dwSlept = 0;
5
6       static VOID (WINAPI * TrueSleep)(DWORD dwMilliseconds) = Sleep;
7
8       VOID WINAPI TimedSleep(DWORD dwMilliseconds)
9       {
10      DWORD dwBeg = GetTickCount();
11      TrueSleep(dwMilliseconds);
12      DWORD dwEnd = GetTickCount();
13
14      InterlockedExchangeAdd(&dwSlept, dwEnd - dwBeg);
15  }
16
17  BOOL WINAPI DllMain(HINSTANCE hinst, DWORD dwReason, LPVOID reserved)
18  {
19      if (DetourIsHelperProcess())
20      {
21              return TRUE;
22      }
23
24      if (dwReason == DLL_PROCESS_ATTACH)
25      {
26              DetourRestoreAfterWith();
27              DetourTransactionBegin();
28              DetourUpdateThread(GetCurrentThread());
29              DetourAttach(&(PVOID&)TrueSleep, TimedSleep);
30              DetourTransactionCommit();
31      }
32      else if (dwReason == DLL_PROCESS_DETACH)
33      {
34              DetourTransactionBegin();
35              DetourUpdateThread(GetCurrentThread());
36              DetourDetach(&(PVOID&)TrueSleep, TimedSleep);
37              DetourTransactionCommit();
38      }
39      return TRUE;
40  }
```

در خط ۶ این برنامه اشاره‌گری به تابع هدف تعریف می‌شود که همان تابع Sleep است. سپس از خطوط ۸ تا ۱۵ تابع منحرف شده تعریف می‌شود که قرار است جایگزین تابع Sleep شود. در این تابع زمان سپری شده از آغاز سیستم قبل و بعد از اجرای تابع Sleep محاسبه می‌شود و سپس اختلاف این دو مقدار را در متغیری ذخیره می‌کند و به عنوان خروجی نشان می‌دهد.

ایجاد وقفه در تابع هدف از طریق فراخوانی DetourAttach در داخل تابع منحرف شده فعال می‌شود. تراکنش تابع منحرف شده توسط فراخوانی تابع DetourTransactionBegin و DetourTransactionCommit علامت‌گذاری می‌شود. تابع DetourAttach دو آرگومان می‌گیرد: آدرس اشاره‌گر تابع هدف و اشاره‌گر تابع منحرف

شده. تابع هدف به عنوان آرگومان ارسال نمی‌شود چرا که باید قبلاً در اشاره‌گر هدف ذخیره شده باشد.

تابع DetourUpdateThread فهرستی از نخ‌ها را در تراکنش فهرست می‌کند و با این کار زمانی که تراکنش صورت می‌گیرد اشاره‌گر دستورالعمل‌ها در زمان مناسب خود به روز رسانی می‌شود.

تابع DetourAttach، تابع Trampoline را برای فراخوانی تابع هدف مقداردهی کرده و آماده می‌کند. زمانی که تراکنش تابع منحرف شده صورت می‌گیرد، تابع هدف و تابع Trampoline بازنویسی می‌شوند و اشاره‌گر هدف برای اشاره کردن به تابع Trampoline به روز می‌شود.

زمانی که تابع هدف منحرف شد تمامی فراخوانی‌های صورت گرفته از آن به سمت تابع منحرف شده هدایت می‌شود. تابع منحرف شده وظیفه دارد تا در زمان فراخوانی تابع هدف از طریق Trampoline تمامی آرگومان‌ها را در تابع هدف کپی کند. طبیعی است که در این حالت تابع هدف زیرمجموعه‌ای از توابعی است که توسط تابع منحرف شده فراخوانی می‌شود.

وقفه‌ی ایجاد شده در تابع هدف توسط فراخوانی تابع DetourDetach در تراکنش تابع منحرف شده صورت می‌گیرد. درست مانند تابع DetourAttach تابع DetourDetach نیز دو آرگومان دارد: آدرس اشاره‌گر تابع هدف و اشاره‌گر به تابع منحرف شده. زمانی که تراکنش منحرف شده پایان می‌گیرد، تابع هدف بازنویسی شده و به حالت اصل خود بازمی‌گردد، که Trampoline حذف شده و اشاره‌گر تابع هدف بازسازی شده و به تابع اصلی اشاره می‌کند.

٤-٢- استفاده از کتابخانه‌ی Detours در مواردی که متن برنامه موجود نیست

در مواردی که توابع کتابخانه Detours باید در برنامه‌ای استفاده شوند که متن آن موجود نیست، توابع باید در یک DLL قرار بگیرند. سپس DLL می‌تواند در زمان اجرا با فراخوانی تابع DetourCreateProcessWithDllEx در یک پردازه‌ی جدید بارگذاری شود. در صورتی که DLL توسط این تابع وارد شد، تابع DllMain باید تابع DetourRestoreAfterWith را فراخوانی کند. اگر امکان اجرای برنامه در محیط‌های ۳۲ و ۶۴ بیتی وجود دارد، تابع DllMain باید تابع DetourIsHelperProcess را فراخوانی کند. DLL باید تابع DetourFinishHelperProcess را فراخوانی کند.

در ادامه برنامه‌ی دیگری به نام Withdll نشان داده شده است که چگونگی استفاده از تابع

DetourCreateProcessWithDllEx را برای بارگذاری DLL منحرف شده درون یک پردازه‌ی جدید بدون تغییر برنامه‌ی هدف نشان می‌دهد. این برنامه تابع CreateProcess را فراخوانی کرده و DLL مورد نظر را درون پردازه‌ی هدف بارگذاری می‌کند.

پردازه در حالت تعلیق با پرچم CREATE_SUSPENDED برای CreateProcess ایجاد می‌شود. Detours تصویر برنامه‌ی باینری را در پردازه جدید به گونه‌ای تغییر می‌دهد که DLL مشخص شده را به عنوان اولین ورودی آن در نظر بگیرد. سپس اجرا درون پردازه ادامه می‌یابد. هنگامی که اجرا از سر گرفته شد، بارکننده‌ی پردازه‌ی ویندوزی ابتدا DLL هدف و بعد از آن هر DLL دیگر درون جدول ورودی برنامه‌ی کاربردی را بارگذاری می‌کند.

DetourCreateProcessWithDllEx جدول ورودی فایل PE هدف که درون حافظه قرار دارد را در پردازه‌ی جدیدی که ایجاد کرده است تغییر می‌دهد. جدول ورودی به روز شده شامل اشاره‌گری به اولین تابعی است که از DLL هدف صادر می‌شود. اگر پردازه‌ی هدف ۳۲ بیتی و پردازه‌ی پدر ۶۴ بیتی باشد یا پردازه‌ی هدف ۶۴ بیتی و پردازه‌ی پدر ۳۲ بیتی باشد، تابع DetourCreateProcessWithDllEx از rundll32.exe برای بارگذاری DLL درون پردازه‌ی helper استفاده می‌کند که پردازه‌ی هدف را به صورت موقت به منظور به روز رسانی جدول ورودی پردازه‌ی هدف با DLL درست تطبیق می‌دهد.

```
1.    #include <stdio.h>
2.    #include <windows.h>
3.    #include <detours.h>
4.
5.    void PrintUsage(void)
6.    {
7.          printf("Usage:\n"
8.                "    withdll.exe [options] [command line]\n"
9.                "Options:\n"
10.               "    /d:file.dll    : Start the process with file.dll.\n"
11.               "    /v             : Verbose, display memory at start.\n"
12.               "    /?             : This help screen.\n");
13.   }
14.
15.   struct ExportContext
16.   {
17.         BOOL    fHasOrdinal1;
18.         ULONG   nExports;
19.   };
20.
21.   static BOOL CALLBACK ExportCallback(PVOID pContext,ULONG nOrdinal,
22.         PCHAR pszSymbol,PVOID pbTarget)
23.   {
24.         (void)pContext;
```

```
25.         (void)pbTarget;
26.         (void)pszSymbol;
27.
28.         ExportContext *pec = (ExportContext *)pContext;
29.
30.         if (nOrdinal == 1)
31.         {
32.               pec->fHasOrdinal1 = TRUE;
33.         }
34.         pec->nExports++;
35.
36.         return TRUE;
37.   }
38.
39. //////////////////////////////////////////////////////////////////////////////
40. //
41.
42.   void TypeToString(DWORD Type, char *pszBuffer, size_t cBuffer)
43.   {
44.         if (Type == MEM_IMAGE)
45.         {
46.               sprintf_s(pszBuffer, cBuffer, "img");
47.         }
48.         else if (Type == MEM_MAPPED)
49.         {
50.               sprintf_s(pszBuffer, cBuffer, "map");
51.         }
52.         else if (Type == MEM_PRIVATE)
53.         {
54.               sprintf_s(pszBuffer, cBuffer, "pri");
55.         }
56.         else
57.         {
58.               sprintf_s(pszBuffer, cBuffer, "%x", Type);
59.         }
60.   }
61.
62.   void StateToString(DWORD State, char *pszBuffer, size_t cBuffer)
63.   {
64.         if (State == MEM_COMMIT)
65.         {
66.               sprintf_s(pszBuffer, cBuffer, "com");
67.         }
68.         else if (State == MEM_FREE)
69.         {
70.               sprintf_s(pszBuffer, cBuffer, "fre");
71.         }
72.         else if (State == MEM_RESERVE)
73.         {
74.               sprintf_s(pszBuffer, cBuffer, "res");
75.         }
76.         else
77.         {
78.               sprintf_s(pszBuffer, cBuffer, "%x", State);
79.         }
80.   }
81.
82.   void ProtectToString(DWORD Protect, char *pszBuffer, size_t cBuffer)
83.   {
84.         if (Protect == 0)
85.         {
86.               sprintf_s(pszBuffer, cBuffer, "");
87.         }
88.         else if (Protect == PAGE_EXECUTE)
89.         {
```

```
90.             sprintf_s(pszBuffer, cBuffer, "--x");
91.         }
92.         else if (Protect == PAGE_EXECUTE_READ)
93.         {
94.             sprintf_s(pszBuffer, cBuffer, "r-x");
95.         }
96.         else if (Protect == PAGE_EXECUTE_READWRITE)
97.         {
98.             sprintf_s(pszBuffer, cBuffer, "rwx");
99.         }
100.        else if (Protect == PAGE_EXECUTE_WRITECOPY)
101.        {
102.            sprintf_s(pszBuffer, cBuffer, "rcx");
103.        }
104.        else if (Protect == PAGE_NOACCESS)
105.        {
106.            sprintf_s(pszBuffer, cBuffer, "---");
107.        }
108.        else if (Protect == PAGE_READONLY)
109.        {
110.            sprintf_s(pszBuffer, cBuffer, "r--");
111.        }
112.        else if (Protect == PAGE_READWRITE)
113.        {
114.            sprintf_s(pszBuffer, cBuffer, "rw-");
115.        }
116.        else if (Protect == PAGE_WRITECOPY)
117.        {
118.            sprintf_s(pszBuffer, cBuffer, "rc-");
119.        }
120.        else if (Protect == (PAGE_GUARD | PAGE_EXECUTE))
121.        {
122.            sprintf_s(pszBuffer, cBuffer, "g--x");
123.        }
124.        else if (Protect == (PAGE_GUARD | PAGE_EXECUTE_READ))
125.        {
126.            sprintf_s(pszBuffer, cBuffer, "gr-x");
127.        }
128.        else if (Protect == (PAGE_GUARD | PAGE_EXECUTE_READWRITE))
129.        {
130.            sprintf_s(pszBuffer, cBuffer, "grwx");
131.        }
132.        else if (Protect == (PAGE_GUARD | PAGE_EXECUTE_WRITECOPY))
133.        {
134.            sprintf_s(pszBuffer, cBuffer, "grcx");
135.        }
136.        else if (Protect == (PAGE_GUARD | PAGE_NOACCESS))
137.        {
138.            sprintf_s(pszBuffer, cBuffer, "g---");
139.        }
140.        else if (Protect == (PAGE_GUARD | PAGE_READONLY))
141.        {
142.            sprintf_s(pszBuffer, cBuffer, "gr--");
143.        }
144.        else if (Protect == (PAGE_GUARD | PAGE_READWRITE))
145.        {
146.            sprintf_s(pszBuffer, cBuffer, "grw-");
147.        }
148.        else if (Protect == (PAGE_GUARD | PAGE_WRITECOPY))
149.        {
150.            sprintf_s(pszBuffer, cBuffer, "grc-");
151.        }
152.        else
153.        {
154.            sprintf_s(pszBuffer, cBuffer, "%x", Protect);
```

```
155.        }
156.  }
157.
158.        static BYTE buffer[65536];
159.
160.        typedef union
161.        {
162.              struct
163.              {
164.                    DWORD Signature;
165.                    IMAGE_FILE_HEADER FileHeader;
166.              } ih;
167.
168.              IMAGE_NT_HEADERS32 ih32;
169.              IMAGE_NT_HEADERS64 ih64;
170.              } IMAGE_NT_HEADER;
171.
172.        struct SECTIONS
173.        {
174.              PBYTE   pbBeg;
175.              PBYTE   pbEnd;
176.              CHAR    szName[16];
177.        } Sections[256];
178.        DWORD SectionCount = 0;
179.        DWORD Bitness = 0;
180.
181.        PCHAR FindSectionName(PBYTE pbBase, PBYTE& pbEnd)
182.        {
183.              for (DWORD n = 0; n < SectionCount; n++)
184.              {
185.                    if (Sections[n].pbBeg == pbBase)
186.                    {
187.                          pbEnd = Sections[n].pbEnd;
188.                          return Sections[n].szName;
189.                    }
190.              }
191.              pbEnd = NULL;
192.              return NULL;
193.        }
194.
195.        ULONG PadToPage(ULONG Size)
196.        {
197.              return (Size & 0xfff)? Size + 0x1000 - (Size & 0xfff): Size;
198.        }
199.
200.        BOOL GetSections(HANDLE hp, PBYTE pbBase)
201.        {
202.              DWORD beg = 0;
203.              DWORD cnt = 0;
204.              SIZE_T done;
205.              IMAGE_DOS_HEADER idh;
206.
207.              if (!ReadProcessMemory(hp, pbBase, &idh, sizeof(idh), &done) || done
208.                    != sizeof(idh))
209.              {
210.                    return FALSE;
211.              }
212.
213.              if (idh.e_magic != IMAGE_DOS_SIGNATURE)
214.              {
215.                    return FALSE;
216.              }
217.
218.              IMAGE_NT_HEADER inh;
219.              if (!ReadProcessMemory(hp, pbBase + idh.e_lfanew, &inh, sizeof(inh),
```

```
220.                &done) || done != sizeof(inh))
221.           {
222.                printf("No Read\n");
223.                return FALSE;
224.           }
225.
226.           if (inh.ih.Signature != IMAGE_NT_SIGNATURE)
227.           {
228.                printf("No NT\n");
229.                return FALSE;
230.           }
231.
232.           beg = idh.e_lfanew+ FIELD_OFFSET( IMAGE_NT_HEADERS, OptionalHeader )
233.                + inh.ih.FileHeader.SizeOfOptionalHeader;
234.           cnt = inh.ih.FileHeader.NumberOfSections;
235.           Bitness = (inh.ih32.OptionalHeader.Magic ==
236.           IMAGE_NT_OPTIONAL_HDR32_MAGIC) ? 32 : 64;
237.           #if 0
238.                printf("%d %d count=%d\n", beg, Bitness, cnt);
239.           #endif
240.
241.           IMAGE_SECTION_HEADER ish;
242.           for (DWORD n = 0; n < cnt; n++)
243.           {
244.                if (!ReadProcessMemory(hp, pbBase + beg + n * sizeof(ish),
245.                     &ish, sizeof(ish), &done) || done != sizeof(ish))
246.                {
247.                     printf("No Read\n");
248.                     return FALSE;
249.                }
250.                Sections[n].pbBeg = pbBase + ish.VirtualAddress;
251.                Sections[n].pbEnd = pbBase + ish.VirtualAddress +
252.                     PadToPage(ish.Misc.VirtualSize);
253.                memcpy(Sections[n].szName, ish.Name, sizeof(ish.Name));
254.                Sections[n].szName[sizeof(ish.Name)] = '\0';
255.                #if 0
256.                     printf("--- %p %s\n", Sections[n].pbBeg,
257.                          Sections[n].szName);
258.                #endif
259.           }
260.           SectionCount = cnt;
261.
262.           return TRUE;
263.      }
264.
265.      BOOL DumpProcess(HANDLE hp)
266.      {
267.           ULONG64 base;
268.           ULONG64 next;
269.
270.           MEMORY_BASIC_INFORMATION mbi;
271.
272.           printf("  %12s %8s %8s: %3s %3s %4s %3s : %8s\n", "Address",
273.                "Offset", "Size", "Typ", "Sta", "Prot", "Ini", "Contents");
274.           printf("  %12s %8s %8s: %3s %3s %4s %3s : %8s\n", "------------",
275.                "--------", "--------", "---", "---", "----", "---", "--------
276.                     ---------");
277.
278.           for (next = 0;;)
279.           {
280.                base = next;
281.                ZeroMemory(&mbi, sizeof(mbi));
282.                if (VirtualQueryEx(hp, (PVOID)base, &mbi, sizeof(mbi)) == 0)
283.                {
284.                     break;
```

```
285.                    }
286.                    if ((mbi.RegionSize & 0xfff) == 0xfff)
287.                    {
288.                         break;
289.                    }
290.
291.                    next = (ULONG64)mbi.BaseAddress + mbi.RegionSize;
292.                    if (mbi.State == MEM_FREE)
293.                    {
294.                         continue;
295.                    }
296.
297.                    CHAR szType[16];
298.                    TypeToString(mbi.Type, szType, ARRAYSIZE(szType));
299.                    CHAR szState[16];
300.                    StateToString(mbi.State, szState, ARRAYSIZE(szState));
301.                    CHAR szProtect[16];
302.                    ProtectToString(mbi.Protect, szProtect, ARRAYSIZE(szProtect));
303.                    CHAR szAllocProtect[16];
304.                    ProtectToString(mbi.AllocationProtect, szAllocProtect,
305.                         ARRAYSIZE(szAllocProtect));
306.
307.                    CHAR szFile[MAX_PATH];
308.                    szFile[0] = '\0';
309.                    DWORD cb = 0;
310.                    PCHAR pszFile = szFile;
311.
312.                    if (base == (ULONG64)mbi.AllocationBase)
313.                    {
314.                         #if 0
315.                              cb = pfGetMappedFileName(hp,
316.                              (PVOID)mbi.AllocationBase, szFile,
317.                                   ARRAYSIZE(szFile));
318.                         #endif
319.                         if (GetSections(hp, (PBYTE)mbi.AllocationBase))
320.                         {
321.                              next = base + 0x1000;
322.                              sprintf_s(szFile, ARRAYSIZE(szFile), "%d-bit PE",
323.                                   Bitness);
324.                         }
325.                    }
326.                    if (cb > 0)
327.                    {
328.                         for (DWORD c = 0; c < cb; c++)
329.                         {
330.                              szFile[c] = (szFile[c] >= 'a' && szFile[c] <= 'z')
331.                                   ? szFile[c] - 'a' + 'A' : szFile[c];
332.                         }
333.                         szFile[cb] = '\0';
334.                    }
335.
336.                    if ((pszFile = strrchr(szFile, '\\')) == NULL)
337.                    {
338.                         pszFile = szFile;
339.                    }
340.                    else
341.                    {
342.                         pszFile++;
343.                    }
344.
345.                    PBYTE pbEnd;
346.                    PCHAR pszSect = FindSectionName((PBYTE)base, pbEnd);
347.                    if (pszSect != NULL)
348.                    {
349.                         pszFile = pszSect;
```

```
350.                     if (next > (ULONG64)pbEnd)
351.                     {
352.                          next = (ULONG64)pbEnd;
353.                     }
354.                }
355.
356.                CHAR szDesc[128];
357.                ZeroMemory(&szDesc, ARRAYSIZE(szDesc));
358.                if (base == (ULONG64)mbi.AllocationBase)
359.                {
360.                     sprintf_s(szDesc, ARRAYSIZE(szDesc), "  %12I64x %8I64x
361.                          %8I64x: %3s %3s %4s %3s : %s",
362.                     (ULONG64)base,(ULONG64)base -
363.                     (ULONG64)mbi.AllocationBase,
364.                     (ULONG64)next - (ULONG64)base,szType,szState,szProtect,
365.                     szAllocProtect,pszFile);
366.
367.                }
368.                else
369.                {
370.                     sprintf_s(szDesc, ARRAYSIZE(szDesc), "  %12s %8I64x
371.                          %8I64x: %3s %3s %4s %3s : %s","-",
372.                          (ULONG64)base - (ULONG64)mbi.AllocationBase,
373.                          (ULONG64)next - (ULONG64)base,szType,szState,
374.                          szProtect,szAllocProtect,pszFile);
375.                }
376.                printf("%s\n", szDesc);
377.           }
378.           return TRUE;
379.      }
380.
381. ////////////////////////////////////////////////////////////////////////// main.
382. //
383.      int CDECL main(int argc, char **argv)
384.      {
385.           BOOLEAN fNeedHelp = FALSE;
386.           BOOLEAN fVerbose = FALSE;
387.           PCHAR pszDllPath = NULL;
388.
389.           int arg = 1;
390.           for (; arg < argc && (argv[arg][0] == '-' || argv[arg][0] == '/');
391.                arg++)
392.           {
393.
394.                CHAR *argn = argv[arg] + 1;
395.                CHAR *argp = argn;
396.                while (*argp && *argp != ':' && *argp != '=')
397.                     argp++;
398.                if (*argp == ':' || *argp == '=')
399.                     *argp++ = '\0';
400.
401.                switch (argn[0])
402.                {
403.                     case 'd':                    // Set DLL Name
404.                     case 'D':
405.                               pszDllPath = argp;
406.                               break;
407.
408.                     case 'v':                    // Verbose
409.                     case 'V':
410.                               fVerbose = TRUE;
411.                               break;
412.
413.                     case '?':                    // Help
414.                               fNeedHelp = TRUE;
```

```
                                    break;

                    default:
                                    fNeedHelp = TRUE;
                                    printf("withdll.exe: Bad argument: %s\n",
                                        argv[arg]);
                                    break;
                }
            }

            if (arg >= argc)
            {
                fNeedHelp = TRUE;
            }

            if (pszDllPath == NULL)
            {
                fNeedHelp = TRUE;
            }

            if (fNeedHelp)
            {
                PrintUsage();
                return 9001;
            }

////////////////////////////////////////////////////////// Validate DLLs.
//
            CHAR szDllPath[1024];
            PCHAR pszFilePart = NULL;

            if (!GetFullPathName(pszDllPath, ARRAYSIZE(szDllPath), szDllPath,
                &pszFilePart))
            {
                printf("withdll.exe: Error: %s is not a valid path name..\n",
                    pszDllPath);
                return 9002;
            }

            HMODULE hDll = LoadLibraryEx(pszDllPath, NULL,
                DONT_RESOLVE_DLL_REFERENCES);
            if (hDll == NULL)
            {
                printf("withdll.exe: Error: %s failed to load (error %d).\n",
                    pszDllPath,GetLastError());
                return 9003;
            }

            ExportContext ec;
            ec.fHasOrdinal1 = FALSE;
            ec.nExports = 0;
            DetourEnumerateExports(hDll, &ec, ExportCallback);
            FreeLibrary(hDll);

            if (!ec.fHasOrdinal1)
            {
                printf("withdll.exe: Error: %s does not export ordinal #1.\n",
                    pszDllPath);
                printf("See help entry DetourCreateProcessWithDllEx in
                    Detours.chm.\n");
                return 9004;
            }

////////////////////////////////////////////////////////////////////////////
            STARTUPINFO si;
```

```
480.            PROCESS_INFORMATION pi;
481.            CHAR szCommand[2048];
482.            CHAR szExe[1024];
483.            CHAR szFullExe[1024] = "\0";
484.            PCHAR pszFileExe = NULL;
485.
486.            ZeroMemory(&si, sizeof(si));
487.            ZeroMemory(&pi, sizeof(pi));
488.            si.cb = sizeof(si);
489.
490.            szCommand[0] = L'\0';
491.
492.            #ifdef _CRT_INSECURE_DEPRECATE
493.                 strcpy_s(szExe, sizeof(szExe), argv[arg]);
494.            #else
495.                 strcpy(szExe, argv[arg]);
496.            #endif
497.            for (; arg < argc; arg++)
498.            {
499.                 if (strchr(argv[arg], ' ') != NULL || strchr(argv[arg], '\t')
500.                      != NULL)
501.                 {
502.                      #ifdef _CRT_INSECURE_DEPRECATE
503.                           strcat_s(szCommand, sizeof(szCommand), "\"");
504.                           strcat_s(szCommand, sizeof(szCommand), argv[arg]);
505.                           strcat_s(szCommand, sizeof(szCommand), "\"");
506.                      #else
507.                           strcat(szCommand, "\"");
508.                           strcat(szCommand, argv[arg]);
509.                           strcat(szCommand, "\"");
510.                      #endif
511.                 }
512.                 else
513.                 {
514.                      #ifdef _CRT_INSECURE_DEPRECATE
515.                           strcat_s(szCommand, sizeof(szCommand), argv[arg]);
516.                      #else
517.                           strcat(szCommand, argv[arg]);
518.                      #endif
519.                 }
520.
521.                 if (arg + 1 < argc)
522.                 {
523.                      #ifdef _CRT_INSECURE_DEPRECATE
524.                           strcat_s(szCommand, sizeof(szCommand), " ");
525.                      #else
526.                           strcat(szCommand, " ");
527.                      #endif
528.                 }
529.            }
530.            printf("withdll.exe: Starting: `%s'\n", szCommand);
531.            printf("withdll.exe:   with `%s'\n", szDllPath);
532.            fflush(stdout);
533.
534.            DWORD dwFlags = CREATE_DEFAULT_ERROR_MODE | CREATE_SUSPENDED;
535.
536.            SetLastError(0);
537.            SearchPath(NULL, szExe, ".exe", ARRAYSIZE(szFullExe), szFullExe,
538.                 &pszFileExe);
539.            if (!DetourCreateProcessWithDllEx(szFullExe[0] ? szFullExe : NULL,
540.                 szCommand,NULL, NULL, TRUE, dwFlags, NULL, NULL,
541.                 &si, &pi, szDllPath, NULL))
542.            {
543.                 DWORD dwError = GetLastError();
544.                 printf("withdll.exe: DetourCreateProcessWithDllEx failed:
```

```
545.                         %d\n", dwError);
546.                   if (dwError == ERROR_INVALID_HANDLE)
547.                   {
548.                         #if DETOURS_64BIT
549.                               printf("withdll.exe: Can't detour a 32-bit target
550.                                     process from a 64-bit parent process.\n");
551.                         #else
552.                               printf("withdll.exe: Can't detour a 64-bit target
553.                                     process from a 32-bit parent process.\n");
554.                         #endif
555.                   }
556.                   ExitProcess(9009);
557.             }
558.
559.             if (fVerbose)
560.             {
561.                   DumpProcess(pi.hProcess);
562.             }
563.
564.             ResumeThread(pi.hThread);
565.
566.             WaitForSingleObject(pi.hProcess, INFINITE);
567.
568.             DWORD dwResult = 0;
569.             if (!GetExitCodeProcess(pi.hProcess, &dwResult))
570.             {
571.                   printf("withdll.exe: GetExitCodeProcess failed: %d\n",
572.                         GetLastError());
573.                   return 9010;
574.             }
575.
576.             return dwResult;
577.       }
578. ////////////////////////////////////////////////////////////// End of File.
```

در خطوط ۵ تا ۱۳ این برنامه تابع PrintUsage روش استفاده از Withdll.exe را نشان می‌دهد و می‌توان از آن به عنوان کمک استفاده کرد. خطوط ۱۵ تا ۳۷ این برنامه مشخص می‌کند که DLL موردنظر به درستی تنظیم شده است که درون پردازه‌ی هدف قرار بگیرد و DLL باید اولین تابع را صادر کند. در خطوط ۴۲ تا ۱۵۶ این برنامه توابع TypeToString و StateToString و ProtectToString به ترتیب نوع، حالت و نحوه‌ی حفاظت هر فایل را به صورت رشته درمی‌آورند. در خطوط ۱۵۸ تا ۲۶۳ این برنامه با استفاده از توابع تعریف شده بخش‌های مختلف فایل PE از DLL موردنظر را به دست می‌آورد. در خطوط ۲۶۵ تا ۳۷۹ این برنامه تابع DumpProcess از تمام بخش‌های پردازه‌ی موردنظر که قرار است DLL ای درون آن تزریق شود را با استفاده از توابع گفته شده در بالا روبرداری[1] می‌کند. در خطوط ۳۸۳ تا ۵۷۷ این برنامه در قسمت main ابتدا مسیر DLL موردنظر را پیدا کرده سپس آن را با استفاده از تابع DetourCreateProcessWithDllEx که به طور مفصل در مورد آن توضیح داده شد درون پردازه‌ی

[1] Dump

جدید تزریق می‌شود و با استفاده از تابع DumpProcess اطلاعات مربوط به پردازه‌ی هدف در خروجی چاپ می‌شود.

خروجی برنامه به صورت شکل ۴ است.

```
withdll.exe: Starting: `..\..\bin.X86\sleepold.exe'
withdll.exe:   with `C:\Documents and Settings\Administrator\Desktop\Detours
ress 3.0\bin.X86\slept32.dll'
       Address    Offset     Size: Typ Sta Prot Ini : Contents
  ------------ -------- --------: --- --- ---- --- : ----------------
         10000        0     2000: pri com  rw- rw- :
         20000        0     1000: pri com  rw- rw- :
         30000        0    fe000: pri res      rw- :
             -   fe000     1000: pri com grw- rw- :
             -   ff000     1000: pri com  rw- rw- :
        130000        0     3000: map com  r-- r-- :
        140000        0     1000: pri com  rw- rw- : 32-bit PE
        400000        0     1000: img com  r-- rcx : 32-bit PE
             -    1000    1a000: img com  r-x rcx : .text
             -   1b000     6000: img com  r-- rcx : .rdata
             -   21000     3000: img com  rc- rcx : .data
             -   24000     2000: img com  r-- rcx : .reloc
        430000        0     1000: pri com  rw- rw- :
       7c900000        0     1000: img com  r-- rcx : 32-bit PE
             -    1000    7d000: img com  r-x rcx : .text
             -   7e000     5000: img com  rc- rcx : .data
             -   83000    2c000: img com  r-- rcx : .rsrc
             -   af000     3000: img com  r-- rcx : .reloc
       7ffb0000        0    24000: map com  r-- r-- :
       7ffde000        0     1000: pri com  rw- rw- :
       7ffdf000        0     1000: pri com  rw- rw- :
       7ffe0000        0     1000: pri com  r-- r-- :
             -    1000     f000: pri res  --- r-- :
```

شکل (٤) : خروجی برنامه withdll.exe

در آخر برنامه‌ی Simple را به صورت dll درآورده و با استفاده از برنامه‌ی دیگری به نام Withdll.exe درون پردازه‌ای تزریق می‌کند. این پردازه به نام Sleep5.exe به صورت زیر سیستم را برای ۵ میلی ثانیه به خواب می‌برد.

```
1.    #include <windows.h>
2.    #include <stdio.h>
3.
4.    int __cdecl main(int argc, char ** argv)
5.    {
6.          if (argc == 2)
7.          {
8.                Sleep(atoi(argv[1]) * 1000);
9.          }
10.         else
11.         {
12.               printf("sleep5.exe: Starting.\n");
13.
14.               Sleep(5000);
15.
16.               printf("sleep5.exe: Done sleeping.\n");
17.         }
18.         return 0;
19.   }
```

دستوری که عملیات گفته شده را انجام می‌دهد به صورت withdll.exe –d simple32.dll sleep5.exe است.

خروجی این عملیات به صورت شکل ۵ است.

```
C:\Documents and Settings\Administrator\Desktop\Detours Express 3.0\samples\simp
le> ..\..\bin.X86\withdll.exe -d:..\..\bin.X86\simple32.dll ..\..\bin.X86\sleep5
.exe
withdll.exe: Starting: '..\..\bin.X86\sleep5.exe'
withdll.exe:   with 'C:\Documents and Settings\Administrator\Desktop\Detours Exp
ress 3.0\bin.X86\simple32.dll'
simple32.dll: Starting.
simple32.dll: Detoured SleepEx().
sleep5.exe: Starting.
sleep5.exe: Done sleeping.
simple32.dll: Removed SleepEx() (result=0), slept 5000 ticks.
```

شکل (۵) : خروجی برنامه تزریق simple32.dll درون برنامه sleep5.exe

٤-۳- نمونه‌ای از جعل API در حالت User

در این نمونه با استفاده از کتابخانه Detours تابع NtOpenFile را جعل کرده و درون پردازه‌ی تزریق می‌کنیم. ابتدا تابع NtOpenFile را تعریف کرده و تابع موردنظر که قرار است جایگزین تابع اصلی شود را دقیقاً با همان آرگومان‌ها و امضای تابع اصلی تعریف می‌کنیم سپس با استفاده از تابع LoadLibrary و GetProcAddress آدرس تابع NtOpenFile را از Ntdll بدست می‌آوریم و اشاره‌گر به آن و همچنین اشاره‌گر به تابع منحرف شده که در اینجا MyNtOpenFile است را در جایی ذخیره می‌کنیم. سپس با استفاده از تابع DetourAttach و DetourDetach مسیر را از تابع NtOpenFile به تابع MyNtOpenFile تغییر می‌دهیم و بعد از انجام عملیات موردنظر به تابع اصلی برمی‌گردد. در این تابع عملیات موردنظر چاپ پارامترهای تابع NtOpenFile و نام فایل مورد استفاده‌ی آن در فایل متنی است که در C:\log.txt ذخیره می‌شود.

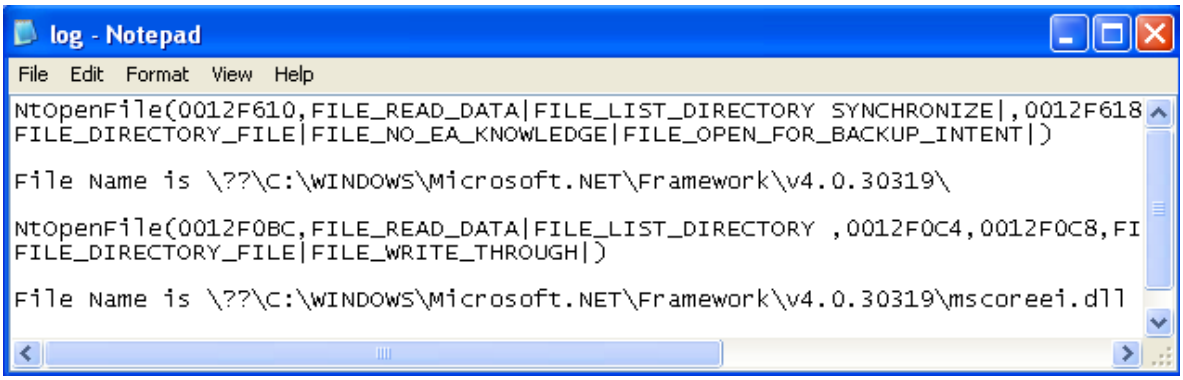
```
1.    #pragma comment(lib, "detours.lib")
2.
3.    #include <iostream>
4.    #include <fstream>
5.    #include <Windows.h>
6.
7.    #include < detours.h>
8.
```

```
using namespace std;

NTSTATUS (WINAPI *pNtOpenFile)(
      _Out_   PHANDLE FileHandle,
      _In_    ACCESS_MASK DesiredAccess,
      _In_    POBJECT_ATTRIBUTES ObjectAttributes,
      _Out_   PIO_STATUS_BLOCK IoStatusBlock,
      _In_    ULONG ShareAccess,
      _In_    ULONG OpenOptions
      );

NTSTATUS WINAPI MyNtOpenFile(PHANDLE FileHandle,ACCESS_MASK
            DesiredAccess,POBJECT_ATTRIBUTES ObjectAttributes,
            PIO_STATUS_BLOCK IoStatusBlock, ULONG ShareAccess,ULONG
            OpenOptions);

HMODULE hModule = LoadLibrary (L"Ntdll.dll");

pNtOpenFile = (NTSTATUS (WINAPI*)
      (PHANDLE,ACCESS_MASK,POBJECT_ATTRIBUTES,PIO_STATUS_BLOCK,
            ULONG,ULONG))GetProcAddress(hModule, "NtOpenFile");

INT APIENTRY DllMain(HMODULE hDLL, DWORD Reason, LPVOID Reserved)
{
      initial();
      switch(Reason)
      {
            case DLL_PROCESS_ATTACH:
                  DisableThreadLibraryCalls(hDLL);
                  DetourTransactionBegin();
                  DetourUpdateThread(GetCurrentThread());
                  DetourAttach(&(PVOID&)pNtOpenFile, MyNtOpenFile);
                  if(DetourTransactionCommit() == NO_ERROR)
                        OutputDebugString(L"NtClose() detoured successfully");
            case DLL_PROCESS_DETACH:
                  DetourTransactionBegin();
                  DetourUpdateThread(GetCurrentThread());
                  DetourDetach(&(PVOID&)pNtOpenFile, MyNtOpenFile);
                  DetourTransactionCommit();
            case DLL_THREAD_ATTACH:
            case DLL_THREAD_DETACH:
                  break;
      }
      return TRUE;
}

NTSTATUS WINAPI MyNtOpenFile(PHANDLE FileHandle,ACCESS_MASK
            DesiredAccess,POBJECT_ATTRIBUTES ObjectAttributes,
            PIO_STATUS_BLOCK IoStatusBlock, ULONG ShareAccess,ULONG OpenOptions)
{
      ofstream cout ("C://log.txt",ios_base::app);
      cout<<"NtOpenFile(";
      cout<<&FileHandle<<",";
      if(buffer[0]=='1')
            cout<<"FILE_READ_DATA|FILE_LIST_DIRECTORY ";
      if(buffer[1]=='1')
            cout<<"FILE_WRITE_DATA |FILE_ADD_FILE ";
      if(buffer[2]=='1')
            cout<<"FILE_APPEND_DATA | FILE_ADD_SUBDIRECTORY
                  |FILE_CREATE_PIPE_INSTANCE ";
      if(buffer[3]=='1')
            cout <<"FILE_READ_EA|";
      if(buffer[4]=='1')
            cout<<"FILE_WRITE_EA|";
      if(buffer[5]=='1')
```

```
74.             cout<<"FILE_EXECUTE|FILE_TRAVERSE |";
75.         if(buffer[6]=='1')
76.             cout<<"FILE_DELETE_CHILD|";
77.         if(buffer[7]=='1')
78.             cout<<"FILE_READ_ATTRIBUTES |";
79.         if(buffer[8]=='1')
80.             cout<<"FILE_WRITE_ATTRIBUTES|";
81.         if(buffer[16]=='1')
82.             cout<<"DELETE|";
83.         if(buffer[17]=='1')
84.             cout<<"READ_CONTROL|";
85.         if(buffer[18]=='1')
86.             cout<<"WRITE_DAC|";
87.         if(buffer[19]=='1')
88.             cout<<"WRITE_OWNER|";
89.         if(buffer[20]=='1')
90.             cout<<"SYNCHRONIZE|";
91.         if(buffer[31]=='1')
92.             cout<<"FILE_READ_DATA, FILE_READ_ATTRIBUTES,
93.                 FILE_READ_EA,SYNCHRONIZE|";
94.         if(buffer[30]=='1')
95.             cout<<"FILE_WRITE_DATA, FILE_WRITE_ATTRIBUTES, FILE_WRITE_EA,
96.                 FILE_APPEND_DATA,SYNCHRONIZE|";
97.         if(buffer[29]=='1')
98.             cout<<"FILE_EXECUTE, FILE_READ_ATTRIBUTES,SYNCHRONIZE|";
99.         if(buffer[28]=='1')
100.                cout<<"FILE_ALL_ACCESS|";
101.        if(buffer[24]=='1')
102.            cout<<"ACCESS_SYSTEM_SECURITY|";
103.        cout<<",";
104.        cout<<&ObjectAttributes<<",";
105.        cout<<&IoStatusBlock<<",";
106.
107.        if(buffer1[0]=='1')
108.            cout<<"FILE_SHARE_READ|";
109.        if(buffer1[1]=='1')
110.            cout<<"FILE_SHARE_WRITE|";
111.        if(buffer1[2]=='1')
112.            cout<<"FILE_SHARE_DELETE|";
113.        cout<<endl;
114.        if(buffer2[0]=='1')
115.            cout<<"FILE_DIRECTORY_FILE|";
116.        if(buffer2[1]=='1')
117.            cout<<"FILE_WRITE_THROUGH|";
118.        if(buffer2[2]=='1')
119.            cout<<"FILE_SEQUENTIAL_ONLY|";
120.        if(buffer2[3]=='1')
121.            cout<<"FILE_NO_INTERMEDIATE_BUFFERING|";
122.        if(buffer2[4]=='1')
123.            cout<<"FILE_SYNCHRONOUS_IO_ALERT|";
124.        if(buffer2[5]=='1')
125.            cout<<"FILE_SYNCHRONOUS_IO_NONALERT|";
126.        if(buffer2[6]=='1')
127.            cout<<"FILE_NON_DIRECTORY_FILE|";
128.        if(buffer2[7]=='1')
129.            cout<<"FILE_CREATE_TREE_CONNECTION|";
130.        if(buffer2[8]=='1')
131.            cout<<"FILE_COMPLETE_IF_OPLOCKED|";
132.        if(buffer2[9]=='1')
133.            cout<<"FILE_NO_EA_KNOWLEDGE|";
134.        if(buffer2[11]=='1')
135.            cout<<"FILE_RANDOM_ACCESS|";
136.        if(buffer2[12]=='1')
137.            cout<<"FILE_DELETE_ON_CLOSE|";
138.        if(buffer2[13]=='1')
```

```
139.            cout<<"FILE_OPEN_BY_FILE_ID|";
140.        if(buffer2[14]=='1')
141.            cout<<"FILE_OPEN_FOR_BACKUP_INTENT|";
142.        if(buffer2[15]=='1')
143.            cout<<"FILE_NO_COMPRESSION|";
144.        cout<<")";
145.        cout<<endl;
146.        ANSI_STRING name;
147.        pRtlUnicodeStringToAnsiString(&name, ObjectAttributes->ObjectName, TRUE);
148.        cout<<"File Name is "<<name.Buffer<<endl;
149.        return pNtOpenFile(FileHandle, DesiredAccess, ObjectAttributes,
150.            IoStatusBlock, ShareAccess, OpenOptions);
151.
152.  }
```

این برنامه را به صورت DLL درآورده و با استفاده از برنامه‌ی Withdll.exe و تابع DetourCreateProcessWithDllEx به پردازه‌ی مورد نظر که در اینجا notepad.exe است تزریق می‌کنیم و خروجی موردنظر را در فایل log.txt به صورت به شکل ۶ می‌بینیم.



شکل (٦) : خروجی برنامه جعل API

۵- استفاده از کتابخانه‌ی Detours جهت استفاده در Payloadها و ویرایش جدول ورودی

علاوه بر توابعی که می‌توان به برنامه پیوست کرد، کتابخانه‌ی Detours شامل توابعی است که می‌توان توسط آنها سگمنت‌های داده‌ی دلخواه را که با عنوان Payload شناخته می‌شود را به فایل اجرایی ویندوز پیوست کرد و یا بوسیله آن جدول ورودی DLL را ویرایش کرد. توابعی که برای ویرایش فایل‌های باینری در کتابخانه Detours وجود دارد کاملاً قابل برگشت است. چراکه Detours اطلاعات بازیابی را در فایل باینری نگه می‌دارد تا هر زمان بتواند فایل را به حالت ابتدایی و اصلی خود بازگرداند.

DOS Header
PE (W/COFF)
.text Section Program Code
.data Section Initialized Data
.idata Section Import Table
.edata Section Export Table
Debug Symbols

شکل (۷) : قالب فایل اجرایی قابل حمل در ویندوز

شکل ۷ ساختار اصلی فایل PE ویندوز را نشان می‌دهد. قالب این فایل در حقیقت قالب گسترش یافته قالب COFF (Common Object File Format) است. فایل باینری ویندوز شامل سرآیند سازگار با DOS، سرآیند PE، قسمت Text که حاوی کد برنامه است، قسمت داده که شامل داده‌های اولیه، جدول ورودی‌ها که لیستی از توابع و DLL های فراخوانی شده را در خود دارد، جدول خروجی شامل توابعی که توسط برنامه قابل دسترس است و debug symbols ها می‌باشد. به استثناء دو سرآیند سایر قسمت‌های برنامه اختیاری هستند و ممکن است در باینری وجود نداشته باشند.

DOS Header
PE (W/COFF)
.text Section Program Code
.data Section Initialized Data
.idata Section Unused Import Table
.edata Section Export Table
.Detours Section Detour header Original PE header New import table User payloads
Debug Symbols

شکل (۸): قالب فایل منحرف شده توسط Detours

برای تغییر در باینری ویندوز Detours یک قسمت جدید بین جدول خروجی و Debug symbol به نام Detours. ایجاد می‌کند که در شکل ۸ نشان داده شده است. دقت کنید که debug symbol حتماً باید در آخرین قسمت فایل باینری قرار بگیرد. این قسمت جدید شامل سرآیند Detours و یک کپی از سرآیند اصلی فایل PE است. در صورتی که جدول ورودی تغییر کند، Detours یک جدول ورودی جدید ایجاد می‌کند و آن را به ادامه‌ی سرآیند فایل PE اضافه می‌کند سپس سرآیند اصلی فایل PE را به گونه‌ای تغییر می‌دهد که به جدول ورودی جدید اشاره کند. در نهایت Detours در ادامه Payload کاربر را اضافه کرده و در پایان آن Debug symbol را قرار می‌دهد تا فایل بسته شود. Detours می‌تواند با بازگردانی اشاره‌گر سرآیند فایل PE و حذف قسمت Detours و فایل را به حالت ابتدایی درآورد. شکل ۸ قالب فایل منحرف شده را توسط Detours را نشان می‌دهد.

ایجاد جدول ورودی جدید دو هدف را دنبال می‌کند. ابتدا جدول ورودی اصلی تغییر نمی‌کند تا هر زمان که برنامه‌نویس نیاز به بازگردانی فایل به حالت اول را داشت، این امر ممکن باشد. دومین دلیل آن این است که با این کار می‌توان در قسمت جدول ورودی جدید از توابعی که نامشان تغییر کرده و یا DLL های جدید استفاده کرد. به عنوان مثال برنامه‌ی Setdll.exe موجود در بسته‌ی Detours، یک ورودی جدید به DLL ای که کاربر معرفی کرده است را در برنامه قرار داده است. با توجه به این که مقدار ورودی جدول ورودی برنامه DLL کاربر است، اولین DLL ای که برنامه در فضای آدرس خود اجرا می‌کند است. Detours توابعی را فراهم می‌کند که بوسیله آن می‌توان جدول ورودی را ویرایش کرد. (DetourBinaryEditImport)، Payload ای را اضافه کرد (DetourBinarySetPayload)، Payload ها را شمارش کرد (DetourBinaryEnumeratePayload) و Payload ها را حذف کرد (DetourBinaryPurgePayload). همچنین Detours توابعی را فراهم می‌کند که می‌تواند فایل‌های باینری نگاشت شده در فضای آدرس (DetourEnumerateModule) و Payload های قرار گرفته توسط آن‌ها را شمارش کند.

هر payload با یک شناسه‌ی سراسری و یکتای ۱۲۸ بیتی شناخته می‌شود. Payload ها می‌توانند به صورت پیوست به داده‌های تنظیمات هر برنامه مورد استفاده قرار گیرند. Payload ها می‌توانند به صورت مستقیم به کمک تابع DetourCopyPayloadToProcess در یک پردازه کپی شوند.

٦- مراجع

1. Z.Bosnic and J.Berdajs, Extending applications using an advanced approach to DLL injection and API hooking.. 2010.
2. G. Hunt and D. Brubacher, Detours: Binary Interception of Win32 Functions.
3. Alex Abramov. API Hooking with MS Detours. CodeProject. [Online] 8 14, 2008. http://www.codeproject.com/Articles/30140/API-Hooking-with-MS-Detours.
4. Shankar, Kamal. Circumventing Windows Group Policies using Detours. CodeProject. [Online] http://www.codeproject.com/Articles/6552/Circumventing-Windows-Group-Policies-using-Detours.
5. Ivanov, Ivo. API hooking revealed. CodeProject. [Online] http://www.codeproject.com/Articles/2082/API-hooking-revealed.