

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

گزارش پروژه درس مهندسی معکوس

موضوع پروژه: نحوه ایجاد یک Sniffer برای بخشی از حافظه

پیاده سازی شده به زبان: C++ (Character Mode)

پیاده ساز: محمد خانجانی

نام استاد: آقای دکتر سعید پارسا

تاریخ: ۱۲ فروردین ۱۳۹۳

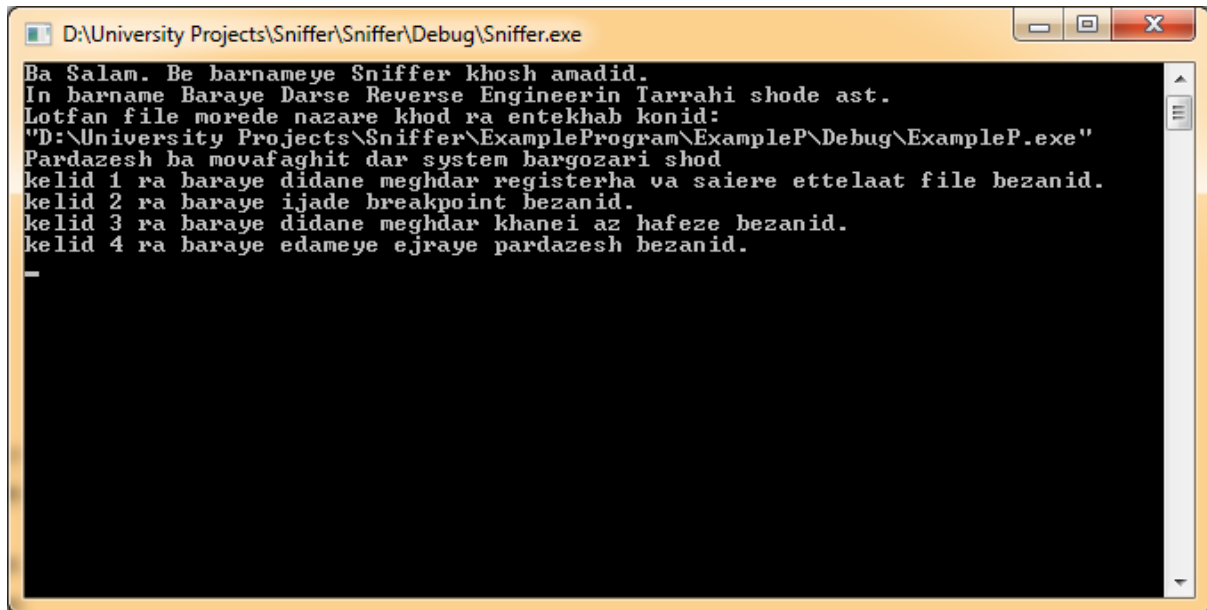
Sniffer چیست؟

Sniffer اصولاً برای کرک کردن برنامه ها، و برای پیدا کردن سریال یا اطلاعاتی خاص از یک برنامه استفاده می شود. فرض کنید یک برنامه داریم که رمز عبور آن یک عدد تصادفی است، و ما رمز عبور آن را نمی دانیم. برای پیدا کردن رمز عبور این برنامه، ما می توانیم یک Sniffer بنویسیم که برنامه را در خط خاصی از کدهایش متوقف کرده، و سپس اقدام به خواندن شماره سریال از آدرسی از حافظه اش می کند.

برای ساخت Sniffer استفاده از توابع API مربوط به دیباگر موجود در ویندوز الزامی است. لذا ما برای اینکه یک پروژه کلی تر داشته باشیم، به جای ساختن Sniffer برای یک نرم افزار خاص، یک Sniffer کلی نوشته ایم، که تمامی آدرس ها و کارهایی که باید انجام دهد را از ورودی دریافت می کند.

طریقه کار با Sniffer پیاده سازی شده

برای استفاده از این برنامه، ابتدا آن را اجرا کرده، و سپس فایل مورد نظر خود را به برنامه بدهید (Drag کنید)، و کلید Enter را بزنید. همانطور که در شکل یک می بینید، برنامه اجرا شده است، و شما می توانید کارهای مورد نظر خود را روی برنامه انجام دهید. مثلاً برای دیدن رجیسترها، و آدرس شروع آن در حافظه مجازی خودش، کافیت کلید یک را بزنید. برای اضافه کردن Breakpoint از کلید ۲، برای دیدن محتوای خانه ای از برنامه (محتوای ۴ بایت، یا یک Double Word را نمایش می دهد) کلید ۳، و برای ادامه اجرای برنامه از کلید ۴ استفاده کنید.



شکل ۱: تصویر برنامه Sniffer

ما یک برنامه ExampleP طراحی کرده ایم، که کد آن به صورت زیر می باشد:

```
#include <iostream>
#include <time.h>

using namespace std;

void main()
{
    int entered_password,password;
    srand(time(NULL));
    password=rand()%1000+1;
    cout << "Please Enter Password:(Beetween 1-1000):" << endl;
    cin >> entered_password;
    if(password==entered_password)
        cout << "Password is true";
    else
        cout << "Password is false";
    cin.get();cin.get();
}
```

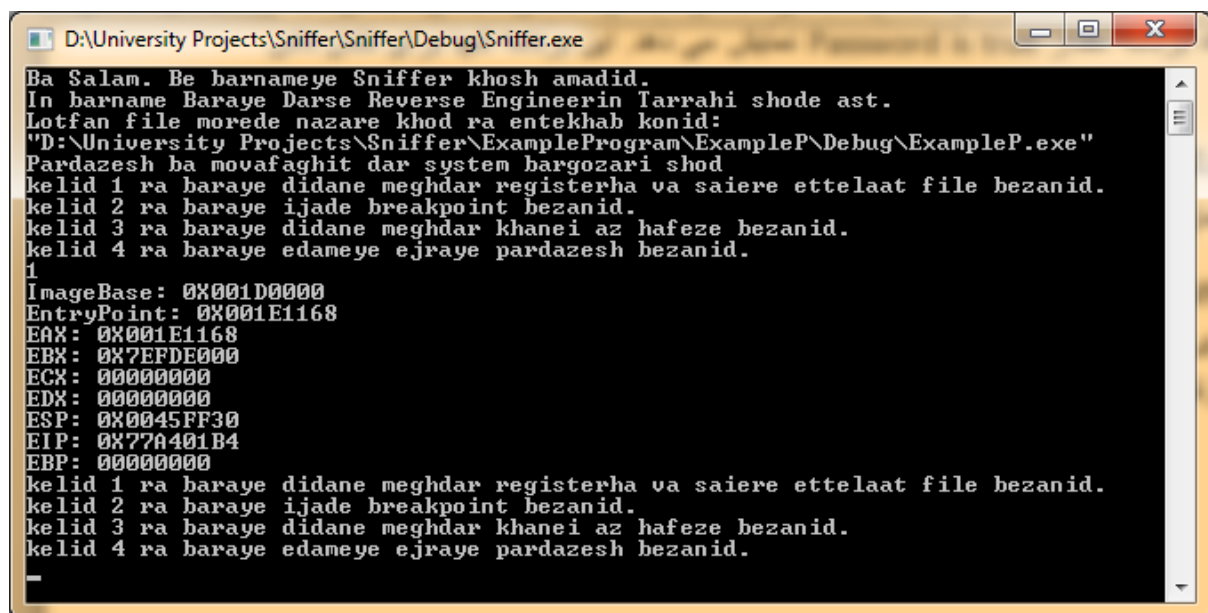
همانطور که می بینید کدهای برنامه مثال ما بسیار واضح است. یک عدد تصادفی تولید شده، سپس یک ورودی از ما گرفته می شود، اگر این دو مقدار برابر باشند، آنگاه برنامه مقدار Password is true نمایش می دهد. این برنامه تنها برای نشان دادن عملکرد Sniffer طراحی شده است.

اگر این برنامه را با OllyDBG یا هر دیباگر دیگر مورد بررسی قرار دهید، می بینید که این برنامه وقتی به آفست 0X143E5 می رسد، مقدار تصادفی ایجاد شده (که همان رمز عبور صحیح است) را در رجیستر EDX ذخیره می کند.

لذا ما بعد از اجرای برنامه در داخل Sniffer خودمان، ابتدا با زدن کلید ۱، آدرس شروع فایل را می بینیم. در ویندوز Vista به بعد، به خاطر افزایش امنیت، از تکنولوژی ASLR استفاده می شود. به این ترتیب با هر بار اجرای برنامه، محل بارگذاری برنامه در حافظه مجازی تغییر می کند، به این ترتیب انجام حملات Buffer Overflow بر روی نرم افزار تا حد زیادی سخت می شود.

با زدن کلید ۱، خروجی موجود در شکل ۲، بر روی صفحه نمایش کامپیوتر ما ظاهر شده است. همانطور که می بینید این بار برنامه در آدرس 0X001D0000 قرار گرفته است. پس ما باید بر روی آدرس $0X001E43E5 = 0X001D0000 + 0X143E5$ یک Breakpoint قرار دهیم. برای این کار، از کلید ۲ استفاده می کنیم. سپس با کلید ۴ برنامه را اجرا می کنیم.

همانطور که در شکل ۳ می بینید، با اجرای برنامه، بلافاصله به خطی که روی آن Breakpoint گذاشتیم، رسیدیم. حال با زدن کلید ۱، مقدار رجیستر EDX را می خوانیم. در کامپیوتر ما، مقدار رجیستر EDX برابر 0X000001A7 (برابر ۴۲۳ در مبنای ۱۰) است. پس کافیت برنامه را با کلید ۴ اجرا کنیم، و مقدار ۴۲۳ را وارد کنیم، تا برنامه به ما بگوید رمز عبور صحیح است!



```
D:\University Projects\Sniffer\Sniffer\Debug\Sniffer.exe
Ba Salam. Be barnameye Sniffer khosh amadid.
In barname Baraye Darse Reverse Engineerin Tarrahi shode ast.
Lotfan file morede nazare khod ra entekhab konid:
"D:\University Projects\Sniffer\ExampleProgram\ExampleP\Debug\ExampleP.exe"
Pardazesh ba movafaghit dar system bargozari shod
kelid 1 ra baraye didane meghdar registerha va saiere ettelaat file bezanid.
kelid 2 ra baraye ijade breakpoint bezanid.
kelid 3 ra baraye didane meghdar khanei az hafeze bezanid.
kelid 4 ra baraye edameye ejraye pardazesh bezanid.
1
ImageBase: 0X001D0000
EntryPoint: 0X001E1168
EAX: 0X001E1168
EBX: 0X7EFDE000
ECX: 00000000
EDX: 00000000
ESP: 0X0045FF30
EIP: 0X77A401B4
EBP: 00000000
kelid 1 ra baraye didane meghdar registerha va saiere ettelaat file bezanid.
kelid 2 ra baraye ijade breakpoint bezanid.
kelid 3 ra baraye didane meghdar khanei az hafeze bezanid.
kelid 4 ra baraye edameye ejraye pardazesh bezanid.
```

شکل ۲

شکل ۳

نگاهی به کدهای برنامه Sniffer

کدهای برنامه Sniffer بسیار ساده است. فقط ۳ تابع دارد. تابع Main که بسیار ساده هست، و نیازی به توضیح ندارد. از تابع GetInput برای دریافت ورودی های کاربر (یعنی کلیدهای ۱ و ۲ و ۳ و ۴ استفاده می شود) روال اصلی برنامه، در داخل تابع DebugProcess اجرا می شود. کدهای این تابع در زیر نشان داده شده است:

```
ZeroMemory( &si, sizeof(si) );
si.cb = sizeof(si);
ZeroMemory( &pi, sizeof(pi) );
if(!CreateProcess(prog_name,NULL,NULL,NULL,FALSE,DEBUG_PROCESS+ DEBUG_ONLY_THIS_PROCESS +
NORMAL_PRIORITY_CLASS, NULL,NULL,&si,&pi))
{
    lastErr=GetLastError();
    return;
}
while(WaitForDebugEvent(&dbgr, INFINITE))
{
    if(dbgr.dwDebugEventCode==CREATE_PROCESS_DEBUG_EVENT)
    {
        printf("Pardazesh ba movafaghit dar system bargozari shod\n");
        GetInput();
    }
    else if(dbgr.dwDebugEventCode==EXCEPTION_DEBUG_EVENT)
    {
        if(dbgr.u.Exception.ExceptionRecord.ExceptionCode==EXCEPTION_BREAKPOINT &&
dbgr.u.Exception.ExceptionRecord.ExceptionAddress==(void*)old_address)
        {
            printf("Yek Breakpoint dar adrese %#.8X rokh dade ast.\n",old_address);
            WriteProcessMemory(pi.hProcess,(LPVOID)old_address,(LPVOID)&oldByte,1,NULL);
            FlushInstructionCache(pi.hProcess,(void*)old_address,1);
            GetThreadContext(pi.hThread,&t);
            t.Eip--;//eip ra yek adad aghab mikeshim.
            SetThreadContext(pi.hThread,&t);
            GetInput();
        }
    }
}
```

```

        ContinueDebugEvent(pi.dwProcessId,pi.dwThreadId,DBG_CONTINUE);
        continue;
    }
}
else if(dbgr.dwDebugEventCode==EXIT_PROCESS_DEBUG_EVENT)
{
    printf("Pardazesh baste shod.\n");
    return;
}
ContinueDebugEvent(pi.dwProcessId,pi.dwThreadId,DBG_EXCEPTION_NOT_HANDLED);
}
return;

```

همانطور که می بینید، ابتدا با استفاده از تابع `CreateProcess` پردازش را ایجاد می کنیم. سپس با استفاده از تابع `WaitForDebugEvent` یک حلقه بی نهایت می سازیم، که هر بار با ایجاد یک اتفاق در پردازش در حال دیباگ، وارد این حلقه می شویم. و کارهای مورد نظر را انجام می دهیم. در انتهای حلقه با اجرای تابع `ContinueDebugEvent` اجازه ادامه اجرا را به برنامه در حال دیباگ می دهیم.

همانطور که می دانید برای نوشتن در حافظه برنامه از دستور `WriteProcessMemory` و برای خواندن از حافظه برنامه، از دستور `ReadProcessMemory` استفاده می شود. همچنین دستور `INT 3` یا معادل هکس آن که برابر `0xCC` است، برای ایجاد `Breakpoint` استفاده می شود. لذا ما برای قرار دادن `Breakpoint` در داخل برنامه در حال دیباگ، کافست در آدرس مورد نظر یک بایت `0xCC` قرار دهیم، تا برنامه هر وقت به آنجا رسید، متوقف شود، و اختیار به دست دیباگر (که در اینجا برنامه `Sniffer` است)، بیفتد. (توجه داشته باشید که بهتر است بعد از تابع `WriteProcessMemory`، برای بروزرسانی حافظه کش دستورات، از تابع `FlushInstructionCache` استفاده کرد).

همچنین زمانی که یک `Breakpoint` اتفاق بیفتد، مقدار متغیر `dbgr.u.Exception.ExceptionRecord.ExceptionCode` برابر `EXCEPTION_BREAKPOINT` می شود، لذا ما در داخل یک حلقه `if`، زمانی که این شرط برقرار بود، ابتدا با استفاده از تابع `WriteProcessMemory` بایت قبلی را جایگزین می کنیم، تا `Breakpoint` حذف شود. و سپس با استفاده از توابع `GetThreadContext` (برای دریافت مقدار رجیسترهای برنامه در حال دیباگ) و `SetThreadContext` (برای تغییر رجیسترهای برنامه در حال دیباگ) مقدار رجیستر `EIP` را یکی کم می کنیم، تا برنامه به دستور قبلی برسد. (چون طول دستور `INT 3` یک بایت است، باید از مقدار `EIP` (که به خاطر اجرای دستور `INT 3` یکی اضافه شده) یکی کم شود، تا خللی به روال اجرای برنامه وارد نشود).

شاید اگر نگاهی مختصر به توابع `API` استفاده شده در برنامه بیندازیم، بهتر از طریقه کار برنامه سر در بیاوریم.

نگاهی به توابع API استفاده شده در برنامه Sniffer

نام تابع API	کاربرد تابع
CreateProcess	برای ساخت یک پردازش جدید (اگر از فلگ <code>DEBUG_PROCESS</code> در این تابع استفاده کنیم، آنگاه می توانیم پردازش ساخته شده را دیباگ کنیم).
ContinueDebugEvent	این تابع اجرای برنامه در حال دیباگ که به خاطر ارائه گزارشی به دیباگر متوقف شده بود، را ادامه می دهد.
FlushInstructionCache	حافظه کش دستورات را برای پردازش مشخص شده، بروزرسانی می کند.

مقدار رجیسترها و سایر اطلاعات یک پردازش را می گیرد، و در ساختمانی از نوع CONTEXT قرار می دهد.	GetThreadContext
کد آخرین خطای اتفاق افتاده در توابع API فراخوانی شده در پردازش فعلی را برمی گرداند.	GetLastError
آدرس خاصی از حافظه پردازش در حال دیباگ را می خواند.	ReadProcessMemory
مقدار رجیسترها و سایر اطلاعات یک پردازش را تغییر می دهد.	SetThreadContext
منتظر ایجاد اتفاقی در داخل پردازش در داخل دیباگ باقی می ماند. این تابع، تا زمانی که اتفاقی در داخل پردازش در حال دیباگ نیفتد، دیباگر را در حالت مسدود نگه می دارد. (البته می توان یک زمان-time-out نیز تعریف کرد.)	WaitForDebugEvent
در آدرس خاصی از حافظه پردازش در حال دیباگ، می نویسد.	WriteProcessMemory
بلاکی از حافظه را صفر می کند. (مقدار صفر در آنجا می نویسد.)	ZeroMemory