MRLAB

Başqa proses yaddaşına dinamik kitabxana faylının yeridilməsi

Azərbaycan Respublikası Xüsusi Rabitə və İnformasiya Təhlükəsizliyi Dövlət Xidməti – Kompüter İnsidentlərinə qarşı Mübarizə Mərkəzi - Malware Research Lab – S. Abasov - 4 Aprel 2022

Bu məqalədə sizlərə code injection metodlarından biri olan DLL injection haqqında qısa məlumat veriləcək. DLL Injection uzun zamandır zərərvericilər tərəfindən geniş istifadə edilən metodların başında gəlir. İnjection hər zaman üçün zərər vermək məqsədi güdməsə belə zərərvericilər üçün bu pis niyyətlərini başqa proseslər üzərindən həyata keçirmək üçün bir yoldur. Dll İnjection metodunu daha yaxşı başa düşməyiniz üçün kiçik bir ssenari üzərindən gedəcəyik. Lakin ilk öncə gəlin DLL nədir bu suala cavab verməyə çalışaq. DLL (Dynamic link library) shared library olaraqda bilinən paylaşılan bilən kitabxana fayllarıdır. Windows əməliyyat sistemi bu kitabxana fayllarına dynamic link library(dll) adı verib. Windows ƏS-də dll fayl formatı bildiyimiz çox kiçik dəyişikliklər ilə birlikdə PE32/64 (portable executable) formatıdır. Əsas üstünlükləri kodların başqa proseslər tərəfindən icra edilə bilməsi üçün paylaşılan bilən olmasıdır. Lakin bundan əlavə olaraqda memory efficient dediyimiz daha bir üstunlüyü movcudur. Windows əməliyyat sistemində sistem dll faylları olaraq bilinən kitabxana faylları yaddaşa yalnız bir dəfə yüklənir digər proseslər (dll yaddaşına məlumat yazmadığı müddət) bu kitabxanadan istifade edir və beləliklə yaddaş sistem tərəfindən olduqca effektiv istfadə edilir. Əlbətdə dll faylların üstünlükləri yalnız bunlardan ibarət deyil. Daha ətraflı məlumat üçün: https://docs.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library

Ümumilikdə 2 tip kitabxana konsepti mövuddur.

1. Static link library
2. Dynamic link library

Static link library

Static kitabxana konsepti başqa kitabxana içərisində olan bütün kodların olduğu kimi proqram içərisinə embeed edilməsidir. Bu kitabxana konseptində kodlar compile time proqram içərisinə yazılır və burada qalır. Əlbətdə bu paralel olaraq proqram həcminində artmasına səbəb olur.

Dynamic link library

Static kitabxana fayllarından fərqli olaraq bu kitabxana faylları runtime load edilə bilir. Yəni proqram təminatı istədiyi zaman dll faylı lazımı funksiyaların köməkliyi ilə yükləyə və içərisindən hər hansı bir funksiyanı icra edə bilər. Bundan əlavə olaraqda istədiyi an unload edə bilər. Static kitabxanadan fərqli olaraq dll həmçinin başqa proqramlaşdırma dillərinə api rolunuda oynaya bilir. A dilində yazılan bir kitabxana faylı B dilində load edilib istifadə edilə bilər. Bunun üçün kiçik bir örnək. C dilində yazılan bir toplama funksiyası daşıyan dll faylı python ilə load edib və həmin funksiyanı çağıraq.

```
#include <Windows.h>

#define DLL_EXPORT __declspec(dllexport)

DLL_EXPORT int topla(int a, int b)
{
	return a + b;
}
```

Yazılan funksiyanı python köməkliyi ilə çağırırıq.

```
>>> windll.TestApp
<WinDLL 'TestApp', handle 7ffc93140000 at 0x27362d7b760>
>>> netice = windll.TestApp.topla(5, 3)
>>> netice
8
```

Gördüyünüz kimi başqa bir proqramlaşdırma yazılan kodu bir başqa proqramlaşdırma dilindən icra etdik. *Bu arada qısa bir xatırlatma DLL bütün proqramlaşdırma dillərində yazıla bilən bir fayl deyil.* Burada ilk öncə **windll.TestApp** köməkliyi ilə dll faylımızı python yaddaşına load etdik.

```
>>> windll.TestApp
<WinDLL 'TestApp', handle 7ffc93140000 at 0x27362d7b760>
>>> netice = windll.TestApp.topla(5, 3)
>>> netice
8
```

python.exe (5584) Properties

General | Statistics | Performance | Threads | Token | Modules | Memory | Environment

Name	Base address	Size	Description
KernelBase.dll	0x7ffce22e0000	2,78 MB	Windows NT BASE API Cli
KernelBase.dll.mui	0x27363100000	1,25 MB	Windows NT BASE API Cli
libffi-7.dll	0x7ffcc50b0000	44 kB	
locale.nls	0x27362b20000	804 kB	
msvcp_win.dll	0x7ffce28d0000	628 kB	Microsoft® C Runtime Lib
msvcrt.dll	0x7ffce3f60000	632 kB	Windows NT CRT DLL
ntdll.dll	0x7ffce4a10000	1,96 MB	NT Layer DLL
ole32.dll	0x7ffce4830000	1,16 MB	Microsoft OLE for Windov
oleaut32.dll	0x7ffce39a0000	820 kB	OLEAUT32.DLL
python3.dll	0x7ffcda7b0000	60 kB	Python Core
python39.dll	0x7ffc505d0000	4,42 MB	Python Core
rpcrt4.dll	0x7ffce2e20000	1,14 MB	Remote Procedure Call R
rsaenh.dll	0x7ffce1200000	208 kB	Microsoft Enhanced Cryp
sechost.dll	0x7ffce3a70000	624 kB	Host for SCM/SDDL/LSA L
SortDefault.nls	0x27362dc0000	3,22 MB	
TestApp.dll	0x7ffc93140000	148 kB	

topla funksiyasına lazımı parametləri ötürüb nəticəni əldə etdik. Burada topla funksiyasını çağıra bilməyimizin səbəbi topla funksiyasının export edilməsidir. `__declspec(dllexport)` direktivi funksiyanın public edilməsinə və başqaları tərəfindən görülməsinə imkan yaradır. Windows əməliyyat sistemi bunu PE export tablosu üzərindən həyata keçirir. **_declspec(dllexport)** direktivi ilə yaradılan funksiyalar PE export tablosuna yazılır. Funksiya çağrıldığı zaman **export** tablosunda funksiya adresi götürülür və həmin adres də yerləşən funksiya icra edilir.

```
<testapp.topla>   8D0411   lea eax,qword ptr ds:[rcx+rdx]
                  C3       ret
```

Gördüyünüz kimi əslində olduqca yaxşı məqsədlər üçün yaradılan bu mexanizm zərərvericilər tərəfindən pis məqsədlər üçün istifadə edilir.

Zərərvericilər bəzi kritik əməliyyatları legitim proseslər tərəfindən icra edilməsi üçün zərərli dll faylını legitim prosesin yaddaşına yükləyib orada icra edirlər.

Remote DLL Loading

Hər hansı bir proses DLL faylını load edə bilmək üçün kernel32.dll tərəfindən export edilən LoadLibrary funksiyasını istifadə edir.

```
HMODULE LoadLibraryA( [in] LPCSTR lpLibFileName);
```

Funksiya parameter olaraq DLL faylının full pathını qəbul edir.

Misal: `LoadLibrary("c:\\users\\test \\filename.dll");`

Normal bir prosesin dll yükləmə metodu belədir. Əgər proses bir başqa prosesin yaddaşına dll yükləmək istəyirsə bu zaman bir neçə sistem funksiyasından istifadə etməlidir. Bunlardan biri CreateRemoteThread funksiyasıdır. Bu funksiya başqa proses üzərində yeni thread yaratmağa imkan verir.

```
HANDLE CreateRemoteThread(
      [in]  HANDLE                 hProcess,
      [in]  LPSECURITY_ATTRIBUTES  lpThreadAttributes,
      [in]  SIZE_T                 dwStackSize,
      [in]  LPTHREAD_START_ROUTINE lpStartAddress,
      [in]  LPVOID                 lpParameter,
      [in]  DWORD                  dwCreationFlags,
      [out] LPDWORD                lpThreadId
);
```

Burada 2 önəmli parametr var. lpStartAddress və lpParameter. *lpStartAddress* parametri başqa prosesdə icra olunacaq funksiyanın adresini istəyir. *lpParametr* isə çağrılan funksiyaya göndəriləcək parametri istəyir. Yuxarıda qeyd etdiyim kimi proses dll load üçün LoadLibrary funksiyasını icra etməlidir və parametr olaraq dll faylın full pathını istəyir. Biz *CreateRemoteThread* funksiyasını çağırarkən **lpStartAddress** parametrini **LoadLibrary** funksiyasının adresini veririk. Parametr olaraq isə zərərli dll faylın pathını. Beləliklə başqa prosesdə yeni thread işə salınır və bu thread **LoadLibrary** funksiyasını çağırır və başqa proses dll faylını load edir.

lpParametr bizdən parametrin (dll fayl path) olduğu ünvan istəyir. Deməli biz zərərli dll faylın pathını prosesin yaddaşına yazıb daha sonra bu məlumatın olduğu ayaddaş adresini **lpParametr** olaraq **CreateRemoteThread** funksiyasına göndərməliyik.

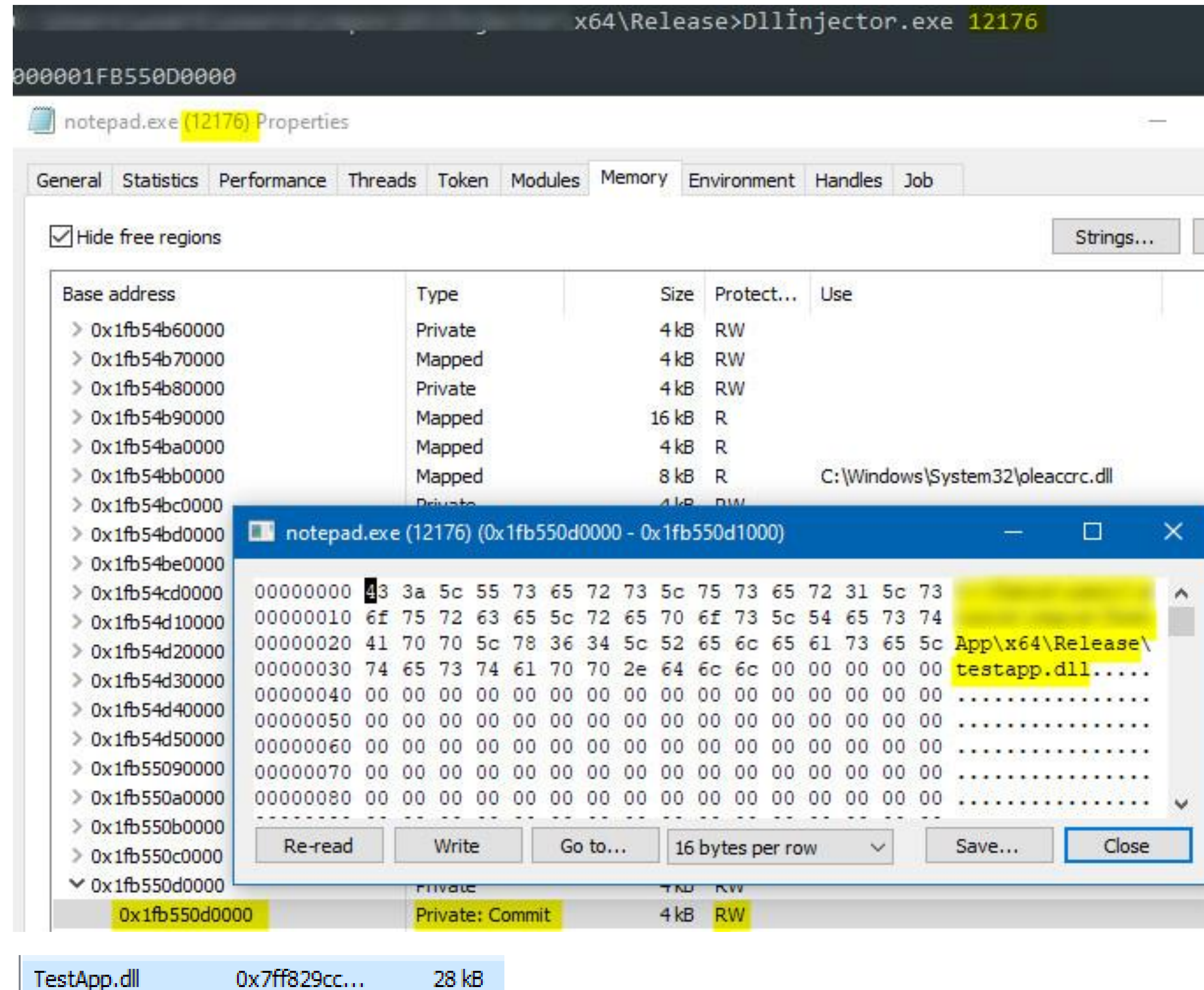
Bunun üçün isə **WriteProcessMemory** funksiyasından istifadə edəcəyik.

```
BOOL WriteProcessMemory(
      [in]  HANDLE  hProcess,
      [in]  LPVOID  lpBaseAddress,
      [in]  LPCVOID lpBuffer,
      [in]  SIZE_T  nSize,
      [out] SIZE_T* lpNumberOfBytesWritten
);

HANDLE hProcess = NULL;
DWORD dwPid = atol(argv[1]);
PCHAR DLLPATH = "c:\\Users\\testuser\\Desktop\\test.dll";
PVOID AllocatedMemory = NULL;
SIZE_T DLLPATH_Len = strlen(dllpath);
SIZE_T dwWritten = 0;
LPVOID ProcAddr = NULL;


ProcAddr = GetProcAddress(GetModuleHandle(L"kernel32.dll"), "LoadLibraryA");
hProcess = OpenProcess( PROCESS_CREATE_THREAD | PROCESS_VM_WRITE, FALSE, dwPid );
if (hProcess == NULL) { return 0; }

AllocatedMemory = VirtualAllocEx(hProcess, NULL, DLLPATH + 1, MEM_COMMIT, PAGE_READWRITE);
if (AllocatedMemory == NULL) { return 0; }

if (!WriteProcessMemory(HpROCESS, AllocatedMemory, DLLPATH, DLLPATH_Len, &dwWritten))
{
      return 0;
}

CreateRemoteThread(hProcess, NULL, 0, ProcAddr, AllocatedMemory, 0, NULL);
```

Şəkil 1Yuxarıda qeyd edilən əməliyyatları sınamaq üçün kiçik bir kod

Ilk öncə **OpenProcess** fuksiyası ilə prosesə handle alırıq. Daha sonra **VirtualAllocEx** funksiyası ilə hədəf proses yaddaşında yeni region yaradırıq. Daha sonra **WriteProcessMemory** ilə ayrılan regiona dll fayl pathını yazırıq. Bundan sonra isə **CreateRemoteThreadEx** funksiyası call edilir.

```
CreateRemoteThread(hProcess, NULL, 0, ProcAddr, AllocatedMemory, 0, NULL);
```



Hədəf process bizim istədiyimiz dll faylını istəmədən yaddaşına yükədi. Lakin burada kiçik bir məsələ var. Hədəf process dll kitabxana faylını yükləsə belə bizim istədiyimiz funksiyanı işə salmaq üçün bu funksiyanı çağırmalıdır. Çün ki, funksiya çağrılmasa remote dll inyeksiyanın heç bir mənası qalmır. Geriyə məqalənin əvvəlində qeyd edilən ssenariyə qayıdaq. Firewall zərərli proqram təminatının internetə çıxışını bloklayıb. Zərərverici isə internet üzərindən daha bir zərərli faylı yükləyib icra etmək istəyir. Belə olan halda əlbətdə ən uyğun strategiya faylı yükləyən bir dll faylın legitim bir prosesin yaddaşına yükləyib icra etməkdir. Yuxarıda qeyd edildiyi kimi burada əsas məsələ dll inyeksiya edildikdən sonra lazım olan funksiyanın icra edilməsidir. Hədəf proses bunu bizim üçün etməyəcəyindən iş yenə dll faylın özünə qalır. Burada **DllEntrypoint** funksiyası problemi həll edir. **DLLEntryPoint** funksiyası dll load edilən zaman avtomatik olaraq icra edilən fuksiyadır. Buna "main" funksiyası kimi baxa bilərsiniz. Əslində dllentrypoint funksiyası dll load edilərkən öncədən sazlamalar edilməsi üçün istifadə edilir. Məsələn **TLS** ilə bağlı etmək istədiyiniz sazlamalar varsa bunu dllentrypoint içərisində etmək daha məqsədə uyğundur.
Microsoft bu haqda:
https://learn.microsoft.com/en-us/windows/win32/dlls/dllmain
https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-entry-point-function
Əgər remote prosesin yaddaşına zərərverici dll inyeksiya etdikdən sonra icra olunmasını istədiyimiz funksiya varsa bunu **DllMain** funksiya içərisində yazırıq. Belə olan halda dll prosesin yaddaşına yükləndiyi an bizim funksiya icra olunacaqdır.

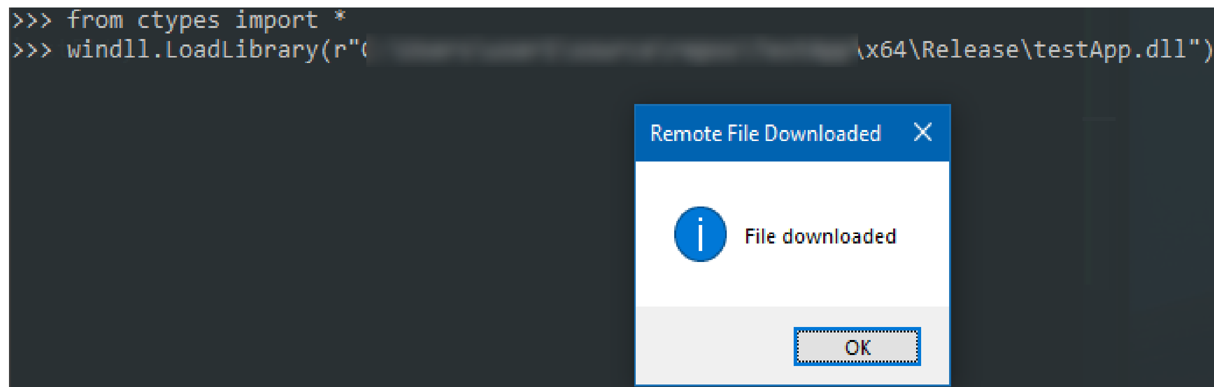
```
BOOL WINAPI DllMain(
    HINSTANCE hinstDLL,  // handle to DLL module
    DWORD fdwReason,     // reason for calling function
    LPVOID lpvReserved)  // reserved
{

    // Perform actions based on the reason for calling.
    switch (fdwReason)
    {
    case DLL_PROCESS_ATTACH:
        // Initialize once for each new process.
        // Return FALSE to fail DLL load.
        break;

    case DLL_THREAD_ATTACH:
        // Do thread-specific initialization.
        break;
```

```
    case DLL_THREAD_DETACH:
        // Do thread-specific cleanup.
        break;

    case DLL_PROCESS_DETACH:

        if (lpvReserved != nullptr)
        {
            break; // do not do cleanup if process termination scenario
        }

        // Perform any necessary cleanup.
        break;
    }

    return TRUE;  // Successful DLL_PROCESS_ATTACH.
}
```

Bəhs edilənlər üzərindən zərərli dll faylımız üzərində bəzi dəyişiklikləri edirik. Göründüyü kimi artıq export edilən bir funksiya yoxdur. **DownloadExecute** funksiyası bir başa **DllMain** içərisindən `DLL_PROCESS_ATTACH` zamanı icra edilir. Python üzərindən DLL faylı load edirik.

```
void DownloadExecute(void)
{
     MessageBoxW(NULL, L"File Downloaded", L"Remote File Downloaded", 64);
}

BOOL WINAPI DllMain(
    HINSTANCE hinstDLL,  // handle to DLL module
    DWORD fdwReason,     // reason for calling function
    LPVOID lpvReserved)  // reserved
{
```

```
    // Perform actions based on the reason for calling.
    switch (fdwReason)
    {
    case DLL_PROCESS_ATTACH:
        DownloadExecute();
    case DLL_THREAD_ATTACH:
        break;
    case DLL_THREAD_DETACH:
        break;
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```



Gördüyünüz kimi artıq heç bir funksiya çağırmağımıza ehtiyyac olmadan load zamanı istədiyimiz funksiyanı icra edə bildik. Bu metodun əsas istifadə məntiqidə məhz budur. Zərərlilər DLL Injection metodu ilə eyni şəkildə dll faylını başqa prosesin yaddaşına inject edib istədikləri əməliyyatları san ki, başqa proses icra edirmiş kimi bir görüntü yaradırlar. Beləliklə bir çox təhükəsizlik proqramlarınıda bu şəkildə bypass edə bilirlər.

İstinadlar

https://docs.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library
https://attack.mitre.org/techniques/T1055/001/
https://en.wikipedia.org/wiki/DLL_injection