Počítačové viry a bezpečnost počítačových systémů
Protokol z předmětu (4b)

Tématická oblast: DLL injection

Přednášející: prof. Ing. Ivan Zelinka, Ph.D.; Ing. Jan Plucar

Jméno a číslo studenta: Daniel Trnka, TRN0038
Datum vypracování: 5. 4. 2019

Zadání:

1) Seznamte se s technikou DLL injection a užívanými technikami.
2) Vyberte si jednu z metod DLL injection (krom Appinit_DLLs registry) a naprogramujte aplikaci umožňující DLL injection dle Vámi vybrané metody.

   BODOVÁNÍ:
   Metoda Windows hooks – minimum bodů
   Metoda "LoadLibrary" a "WriteProcessMemory" – plný počet bodů
   Jiná metoda – dle obtížnosti

3) Vytvořte DLL knihovnu, která bude obsahovat libovolnou (smysluplnou) funkci, jež se provede po úspěšném zavedení. Můžete využít kód malware z minulých cvičení.
4) Přes Vaši aplikaci proveďte DLL injection vytvořené knihovny do zamýšleného (běžícího) procesu.

Závěr:

Diskutujte následující témata:

1) Co je DLL injection? Diskutujte možná využití v praxi.
2) Detailněji popište, jak funguje Vámi vybraná metoda DLL injection a dle vybrané metody odpovězte na následující otázky:
   a) Co se stane po ukončení obslužné (útočníkově) aplikaci, běží-li nadále původní (napadená) aplikace? Bude knihovna stále vykonávat svoji funkci?
   b) Jak se DLL injection projeví v task manageru?

1 Zadání

1.1 Techniky DLL injection

DLL injection umožňuje vložit do jiného legitimního procesu dynamickou knihovnu a vykonávat ji pod daným procesem. Lze tak skrýt případný malware.

Existuje několik možnosti jak provést DLL injection:

[1]

Appinit_DLL injektuje DLL nastavenou v registrech do všech procesů využívající knihovnu `user32.dll`. Nastavení musí provést administrátor a systém se musí zavést s vypnutým SecureBoot.

1.2 DLL injection pomocí `LoadLibrary` a `WriteProcessMemory`

Windows API umožňuje pomocí funkce `CreateRemoteThread` spustit vlákno v jiném procesu. Pro spuštění vlákna potřebujeme `HANDLE` na daný proces a adresu funkce či obecně první instrukce. Dále je možné do této funkce předat jediný číselný argument.

V adresáři `c/function_call` je ukázka, jak z `inject.c` zavolat funkci v `victim.c`. Programy je nutné zkompilovat pomocí překladače `x86_64-w64-mingw32-gcc` či v překladači MSVCC s vypnutou volbou `/DYNAMICBASE`. V opačném případě se obraz procesu nahrává na náhodnou adresu v rámci adresního prostoru procesu, aby se zkomplikovali možné útoky, kdy například útočník zná adresu funkce. V rámci ukázek se volá funkce bez parametru, s číselný parametrem, ale také se předává ukazatel na pole či se volá funkce s více parametry.

Pro předání pole do funkce je nutné v cizím procesu alokovat pole potřebné velikosti a překopírovat sem obsah. Poté se ve funkci `CreateRemoteThread` předá jako první parametr adresa alokované paměti v cizím procesu.

Zavolání funkce s více parametry vyžaduje do procesu vložit instrukce. Konvence pro předávání argumentu na x86_64 vyžaduje, aby první čtyři parametry byly předány pomocí registrů `RCX`, `RDX`, `R8` a `R9`[2]. Pro zavolání funkce se třemi číselnými argumenty `1`, `2` a `3` je nutné vložit instrukce, které do registrů vloží hodnoty. Protože funkce vyžaduje 3x `int`, tak je možné provést zápis argumentů do 32bitových registrů, čímž se získá kratší opcodes. Zavolání funkce se pak provede skokem na absolutní adresu. Instrukce skoku

[1] `https://resources.infosecinstitute.com/using-setwindowshookex-for-dll-injection-on-windows`

[2] `https://en.wikipedia.org/wiki/X86_calling_conventions#Microsoft_x64_calling_convention`

`jmp` neumožňuje provést skok na 64bitovou adresu a je tedy nutné tuto adresu uložit do registru a následně provést skok na hodnotu uloženou v registru:

```
mov ecx, 1
mov edx, 2
mov r8d, 3
mov rax, remote_add
jmp rax
```

Binární reprezentaci instrukci lze manuálně získat například pomocí online nástroje[3]. Tuto reprezentaci je nutné vložit do alokované paměti v cizím procesu a následně zavolat `CreateRemoteThread` s adresou na dané instrukce. Vlákno provede nastavení registrů a skok do funkce `add`, která vytiskne argumenty a jejich součet vrátí. Vrácení hodnoty `int` probíhá pomocí registru `EAX`. Protože se tato funkce volala pomocí skoku `jmp`, tak se jedná o funkci po jejímž návratu je vlákno ukončeno a návratová hodnota je předana předkovi jako `EXIT CODE`.

Nahrání DLL do procesu je možné pomocí funkce:

```
void* LoadLibraryA(char* path)
```

Funkce vyžaduje jediný argument a to řetězec obsahující cestu k dynamické knihovně. Jedinou komplikaci je, že argumentem je řetězec a je tedy nutné v cizím procesu alokovat paměť pro tento řetězec. Další komplikaci by mohlo být získání adresy funkce `LoadLibraryA`. Tato funkce je ale součástí knihovny `kernel32.dll`, která je do všech procesů namapovaná na stejné místo. Tento problém tedy odpadá.

Při načtení dynamické knihovny se zavolá funkce `DllMain` z dané knihovny. Pokud funkce vrátí `1`, tak se vlákno s funkci `LoadLibraryA` ukončí a předek může získat adresu načtené knihovny z `EXIT CODE`. Pro zavolání vlastní funkce v dynamické knihovně je nutné k této adrese přičíst offset od začátku knihovny.

1.3 DLL knihovna

Dynamická knihovna `c/inject_dll/powershell.c` definuje dvě funkce. První funkce `DllMain` se zavolá při načtení/odebrání knihovny do procesu a vypíše PID daného procesu. Druhá funkce `run` vytvoří nový PowerShell proces a vykoná zakódovaný BASE64 kód z minulého cvičení.

[3] `https://defuse.ca/online-x86-assembler.htm`

1.4 Injection

Program `c/inject/inject.c` provádí výše popsaný postup DLL injection.

Pro vložení DLL do procesu `cmd.exe` lze zavolat:

```
$ inject.exe cmd.exe E:/5/c/inject_dll/powershell.dll
```

2 Závěr

1.