

Buffer Overflow (BOF) – Exploiting

[1] Vysvětlení pojmů

- **Buffer** – buffer je část paměti, kam jsou ukládána data aplikace (např. proměnné, načtená data, atp.). Zjednodušeně můžeme rozeznat dvojí alokaci bufferu. K tomu je třeba dodat, že v programu existuje jeden „hlavní“ buffer, který je alokován přímo PE Loaderem při načtení PE souboru – tzv. „stack“. Celkově můžeme rozeznat dva druhy bufferů:
 - a) **globální** – globální buffer je alokován přímo ve spustitelném souboru, případně je alokován ihned po spuštění a je přístupný veškerému kódu programu.
 - b) **lokální** – lokální buffer je alokován většinou přímo do stacku (zásobník) na požadavek nějakého kódu (funkce, procedury, atp.)
- **Buffer Overflow** – přetečení bufferu nastává tehdy, když je do něj zapsáno více dat, než pro jaké množství je daný buffer alokován.
- **Exploiting** – činnost, kdy je využita chyba pro napadení cíle
- **ESP** – registr procesoru, který obsahuje aktuální adresu stacku

[2] Teorie

Pokud budeme chtít pochopit, jak BOF dosáhnout, je nutné si říct základy interakce programu se stackem (zásobníkem). Zásobník je využíván jako dočasná paměť. Pokud funkce A zavolá funkci B, jsou do zásobníku uloženy parametry, se kterými byla funkce B zavolána a hlavně adresa, odkud byla zavolána (!) – po dokončení funkce program pokračuje touto uloženou adresou.

Nyní si tuto hypotetickou událost „zilustrujeme“:

```

Akce:      Funkce A volá funkci B:
ESP:      0010000
Zásobník
0010000:  Adresa funkce A      ;zpáteční adresa
000FFFC:  Parametr 1.        ;parametry
000FFF8:  Parametr 2.
...
                                ;další data
  
```

```

Akce:      inicializace funkce B, která používá buffer velký 8 bytů
ESP:      0010008
Zásobník
0010008:  00 00 00 00          ;alokovaný buffer
0010004:  00 00 00 00
0010000:  00 40 12 31          ;zpáteční adresa funkce A
000FFFC:  Parametr 1.
000FFF8:  Parametr 2.
...
  
```

Nyní si začneme „hrát“ s tímto lokálním bufferem. Například bude uživatel ve funkci B vyzván, aby zadal své jméno, které bude následně uloženo do bufferu.

```

Akce:      zápis zadaného jména „Jan Pták“ do zásobníku
ESP:      0010008
Zásobník
0010008:    4A 61 6E 20           ;Jan_
0010004:    50 74 E1 6B           ;Pták
0010000:    00 40 12 31           ;zpáteční adresa funkce A
000FFFC:    Parametr 1.
000FFF8:    Parametr 2.
...

```

Sami vidíte, že celý zásobník je zaplněn. Ale co když zapíšeme delší jméno?

```

Akce:      zápis zadaného jména „Jaromír Stránský“ do zásobníku
ESP:      0010008
Zásobník
0010008:    4A 61 72 6F           ;Jaro
0010004:    6D ED 72 20           ;mír_
0010000:    53 74 72 E1           ;Strá
000FFFC:    6E 73 6B FD           ;nský
000FFF8:    Parametr 2.
...

```

Zadáním delšího jména jsme způsobili přetečení zásobníku a přepsali jsme tak adresu, kam se program z funkce vrátí. Nyní program skočí na adresu 537472E1h. Pokud na této adrese není paměť určená pro „execuci“ (spuštění), dojde k známé chybě „Access violation“.

[3] Exploiting

Nechceme, aby program skončil chybou, ale aby zpracoval náš kód. To provedeme stejným způsobem, jako výše uvedený buffer overflow s jediným rozdílem. Data, kterými budeme přepisovat data zásobníku budou speciálně navržena.

Víme, že první dword za bufferem je adresa, kam program skočí zpátky. My nastrčíme falešnou adresu ukazující na náš kód – tzv. „shellcode“.

```

Akce:      zápis shellcodu
ESP:      0010008
Zásobník
0010008:    4A 61 72 6F           ;Jaro
0010004:    6D ED 72 20           ;mír_
0010000:    00 0F FF FC           ;adresa našeho shellcodu (!)
000FFFC:    ...                   ;kód, co dělá cokoliv
000FFF8:    ...                   ;co chceme ;)
...

```

[4] Závěr

Každá aplikace obsahuje takovéto buffery, do kterých ukládají svá data. Ať už je to přehraavač hudby, který si do bufferu uloží mp3 soubor, nebo FTP server, který si načte uživatelské jméno. Všechny tyto programy, pokud nejsou ochráněné proti přetečení zásobníku jsou napadnutelné.

Většina z dnešních vyšších jazyků (např. Delphi, Visual Basic) obsahují kontrolní algoritmy, které testují, zda-li nejsou data větší, než buffer, do kterého budou uloženy. Pokud jsou, je alokován nový, větší buffer. Oproti tomu, nízkourovňové jazyky (např. Asm, C) tyto algoritmy neobsahují a je jen na programátorech, zda-li tyto ochranné prvky do programu zahrnou, nebo ne.

Microsoft implementoval do Windows ochranu, která se snaží zabránit takovému zneužívání pomocí funkce DEP (Zabránění spuštění dat) – tato funkce blokuje spuštění kódu tam, kde je to více, než podezřelé.

[5] Autor

Lodus

Web: www.lodusweb.net

Email: lodus at seznam dot cz

Icq: 17_80_50_881

21. června 2007

© Lodus Software 2003 - 2007