

Tartu Ülikool

Referaat aines

Andmeturve

CVE ID: CAN-2004-0234

Autor: Laas Toom
Kursus: infotehnoloogia magister II

Tartu 2005

Sissejuhatus

CAN-2004-0234 nimeline turvahoiautus annab teada, et pakkimisprogramm LHa sisaldab pinu-põhist puhvri ületäitumist. Seda ära kasutades saab ründaja käivitada oma programmikoodi kasutaja õigustes, kes LHa käima pani.

Turvaaugu kirjeldus

LHa versioonid kuni lha-114i sisaldavad failis header.c järgnevat koodilõiku:

```
650         if (dir_length) {
651             strcat(dirname, hdr->name);
652             strcpy(hdr->name, dirname);
653             name_length += dir_length;
654         }
```

Ülaltoodud kood on pärit funktsionist **get_header()**, mis esmalt loeb arhiivifailist failide või kataloogide nimede pikkused ning seejärel loeb niimitu baiti arhiivist puhvrise. Puhvrise kirjutamiseks kasutatakse funktsiooni **strcpy()**, kuid ei kontrollita, kas loetav info puhvrise ka ära mahub. Tulemuseks ongi puhvri ületäitumise võimalikkus ning seeläbi puhvri järel asuva mäluosa hävitamine (ning võimalik võõra koodi käivitamine).

Rünne

SecuritiFocus'e leheküljelt¹ on võimalik alla laadida turvaaugu demonstreerimiseks koostatud C-keelne fail ning ka spetsiaalselt konstrueeritud LHa arhiiv (overflow.lha), mis lahti pakkimisel põhjustabki puhvri ületäitumise. Kuna need mõlemad failid on viidatud veebilehelt vabalt alla laetavad, siis siinkohal ei hakka nende koodi ära tooma, vaid piirdun ainult nende funktsionaalsuse kirjeldamisega.

Antud ekploit kasutas nn keskkonnamuutuja rünnet, milles rünnatav programm käivitatakse eksploidi poolt, kusjuures käivitamisel antakse ette üks keskkonnamuutuja, kuhu on paigutatud kestakood.

Selle ründemeetodi idee on järgmine – programmi pinu on järjestatud algusest selliselt:

- pinu algusaadress 0xBFFFFFFF
- neli NULL baiti
- käivitatava programmi nimi
- N-s keskkonnamuutuja

¹ <http://www.securityfocus.com/bid/10243/info/>

- ...
- 1. keskkonnamuutuja
- ...

Kõik see kokku tähendab, et kui programmile on ette antud üks keskkonnamuutuja, siis on võimalik selle asukoht mälus alati **täpselt** välja arvutada:

$$\&env_N = 0xBFFFFFFFA - \text{strlen}(\text{prog_name}) - \text{strlen}(env_N)$$

Seega kui programmi käivitamisel paigutada kestakood ainukesse ehk viimasesse (N-ndasse) keskkonnamuutujasse, siis on täpselt teada, kus ta mälus paikneb. Jääb üle ainult puhver täita selle aadressiga ja loota, et ka tagastusaadressi asukoht sai üle kirjutatud (seejuures tuleb jälgida, et ei oleks baidinihet – et tagastusaadressi alguskoht on kohakuti meie kirjutatud väärtuse alguskohaga). Samasugust lähenemist kasutas ka kõnealune eksplloit – kestakoodi paigutas keskkonnamuutujasse ning arvutas selle baasil välja tagastusaadressi, mille eksplloidi kasutaja peab paigutama overflow.lha faili õige kohta peale.

Eksplloidi kestakood oli aga pisut kummaline – kasutades **setreuid(0,0)** ja **setgid(0)** funktsione üritas ta ilmselt roodu õigusi saada, kuid kuna vaevalt lha on kuskil **setuid root** installitud, siis see loomulikult ei õnnestu (kui õnnestuks, siis poleks ju vaja üldse mingeid turvaauke otsida, vaid võiks lihtsalt oma suvalise programiga omale roodu õigused võtta). Seetõttu võiks eksplloidis toodud kestakoodi asendada ka palju lühemaga, mis tegeleb ainult /bin/sh käivitamisega või näiteks kirjutab mingisse faili mõne lause.

Kohe peale alla laadimist ei andnud testimiseks kasutatavas arvutis (linux-2.6.9) antud eksplloit kahjuks oodatud tulemusi vaid lahtipakkimisprogramm kuvas veateate ning sai opsüsteemilt SIGSEGV, mis viitab asjaolule, et üritati kasutada vale mäluaadressi.

Uurimisel selgus kaks probleemi:

1. Arhiivifaili sissekirjutatud mäluaadress, kust peale puhvri ületäitumist oleks pidanud koodikäivitus jätkuma, ilmselt ei olnud õige. Eksplloidi käivitamisel trükitakse kasutajale ekraanile ka korrektne tagastusaadress, mis tuleks overflow.lha failis õigesse kohta kirjutada. Siit aga probleem – aadressi üks bait (0x94) ei oma nõ trükitavat sümbolit, mistõttu ekraanil oli näha vaid 3 sümbolit. Selle asemel, et hakata vaev nägema, kuidas mittetrükitavat sümbolit failis õige koha peale sisestada, läksin kergema vastupanu teed: tegin kindlaks esimese suurema väärtusega sümboli, mis on trükitav – selleks osutus 0xA1, mis on 13 võrra suurem kui 0x94, seega lisasin kestakoodile piisavalt NOP käske etteotsa, et tagastusaadress võiks panna mulle sobivama väärtuse.
2. Kuna ka sellest polnud abi, siis GDB abil tegin kindlaks et protsessori register **EIP** sisaldab vajaliku tagastusaadressi asemel hoopis baite **0x55555555** ehk **UUUU**. Tulenevalt overflow.lha sisust (sisaldas pikka U-de jada kataloogi nime asemel) oli selge, et overflow.lha kirjutab tagastusaadressi liiga kaugemale mällu (tegelik asukoht kaeti U-dega). Seega tuli järgnevalt leida failis õige asukoht. Selleks hakkasin lha failis U-sid asendama teiste tähtedega (näiteks ABCD) ning GDB abil jälgima, mis väärtusi registrid omandavad. Niiviisi leidsin failis õige koha, kuhu tagastusaadress kirjutada, nii et see hiljem EIP registrisse laetakse.

Piltlikult kujutatuna näeb see olukord välja järgnevalt:

```
<----- [NNNNNNNNNNSSSSSSSSSS] [0xDE] [0xDE] [0xDE] [0xDE] [0xDE]
          ^                               |
          |_____|
```

Pinu lõpp

Pinu algus (mälu lõpp)

Kasutades jällegi GDB'd ning modifitseerides oma overflow.lha faili, on võimalik umbkaudu arvata ära tagastusaadressi asukoht sõltuvalt puhvriss loetud 'failinimest' (kuipalju sellest failinimest paikneb tagapool hetke ESP registri väärtusest). See ESP väärtus annab ka hinnangu tagastusaadressi asukohale pinus. Neid teadmisi kasutades tuleb konstrueerida uus overflow.lha arhiiv, kus on arhiivi sisse peidetud kestakood ning vajalik tagastusaadress.

Taoline kestakood peaks olema võimalikult lühike, sest LHa faili päise formaat seab piiranguid, mida on tarvis jälgida, et lahtipakkimisel jõutaks üldse selle haavatava puhvrini. Samuti pole võimalik ka lihtsalt niisama failis suvalisse kohta (ka mitte igale poole U-de asemele) mäluaadresse kirjutada, kuna faili teine bait on kontrollsumma, mis annab LHa-le koheselt teada, et faili päis on vigane.

Kuna LHa faili formaadi alusel vastavate programmide koostamine, mis sellele vastavat arhiivi koos sissepeidetud kestakoodiga suudaks tekitada, oleks liiga kaua aega võtnud, siis siinkohal jäi ülalkirjeldatud rünne praktikas katsetamata. Siiski võib oletada, et see rünne on teostatav, kuna juba GDB abil saadud info abil on võimalik päris hästi (konkreetse arvuti) jaoks ära arvata tagastusaadressi väärtus ning asukoht mälus, mis tähendab, et keegi pahalane saab kasutada endale kättesaadavat postiserverit anonüümselt näiteks spämmi levitamiseks, valmistades esmalt selle ründe oma kasutajakonto abil ette ning hiljem juba kuskilt mujalt saadetud kirja abil läbi viia.

Turvaaugu paikamine

SecuritiFocus ning mitmed teised pakkusid esimesteks paikadeks välja:

```
    if (dir_length) {
+         if ((dir_length + name_length) > sizeof(dirname)) {
+             fprintf(stderr, "Insufficient buffer size\n");
+             exit(112);
+         }
        strcat(dirname, hdr->name);
        strcpy(hdr->name, dirname);
+
+         if ((dir_length + name_length) > sizeof(hdr->name)) {
+             fprintf(stderr, "Insufficient buffer size\n");
+             exit(112);
+         }
+         strncpy(hdr->name, dirname, sizeof(hdr->name));
+         name_length += dir_length;
    }
```

Ülaltoodud paik muudab get_header() funktsiooni selliselt, et kui arhiivist loetav kataloogi või faili nime pikkus ületab puhvrise suurusi, siis väljutakse veateatega. Lisaks on asendatud strcpy() funktsioon strncpy() funktsiooniga, mis arvestab ka puhvri suurust.

Testarvutis installeeritud SuSE 9.1 update installeeris aga lha-114i versiooni asemele lha-114g versiooni, milles on selle ründe vastu võitlemiseks lisatud järgnevalt toodud ainult lisatud read):

```

@@ -538,6 +538,10 @@
        /*
        * filename
        */
+       if (header_size >= 256) {
+       fprintf(stderr, "Possible buffer overflow hack attack, type #1\n");
+       exit(109);
+       }
@@ -547,6 +551,10 @@
        /*
        * directory
        */
+       if (header_size >= FILENAME_LENGTH) {
+       fprintf(stderr, "Possible buffer overflow hack attack, type #2\n");
+       exit(110);
+       }

```

See paik lisab samasse funktsiooni pisut varasemasse kohta (enne kui üldse hakatakse failist midagi lugema) kontrolli päise suuruse üle ning kui see ületab lubatu, siis annab vastava veateate. Praktikas näeb asi välja järgnev:

```

|-----
[+] RET ADDRESS = 0xbffffd94
[!] Paste These ASCII 4 bytes Ret Adress to the XXXX in the file overflow.lha
[!] ASCII RET ADDR = ýÿ¿
[+] Exploiting the buffer..
Possible buffer overflow hack attack, type #1

```

Nagu näha, on kirjeldatud exploit #1 tüüpi ülaltoodud paiga mõttes ning lha on suuteline seda rünnakut efektiivselt tuvastama.

Kasutatud kirjandus

1. SecurityFocus, <http://www.securityfocus.com/bid/10243/info/>
2. CVE, <http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0234>
3. Buffer Overflows Demystified, <http://www.enderunix.org/docs/eng/bof-eng.txt>