Systemy operacyjne

Lista zadań nr 4

Na zajęcia 31 października 2024

Należy przygotować się do zajęć czytając następujące materiały: [1, rozdziały 3, 4 i 5], [2, rozdziały 4, 5, 18 i 44], [3, rozdziały 4.1, 4.2, 10.6], [4, rozdział 39].

UWAGA! W trakcie prezentacji należy być gotowym do zdefiniowania pojęć oznaczonych **wytłuszczoną** czcionką.

Zadanie 1. W bieżącej wersji biblioteki «libcsapp» znajdują się pliki «terminal.h» i «terminal.c». Przeczytaj [2, 62.1 i 62.2], a następnie zreferuj działanie procedury «tty_curpos» odczytującej pozycję kursora terminala. Do czego służy kod sterujący «CPR» opisany w CSI sequences[1]? Posiłkując się `tty_ioctl(4)` wytłumacz semantykę rozkazów «TCGETS» i «TCSETSW», wykorzystywanych odpowiednio przez `tcgetattr(3)` i `tcsetattr(3)`, oraz «TIOCINQ» i «TIOCSTI». Na podstawie `termios(4)` wyjaśnij jak flagi «ECHO», «ICANON», «CREAD» wpływają na działanie sterownika terminala.

Zadanie 2. Na podstawie [1, 19.2] wyjaśnij działanie programu `script(1)`. Nagraj interaktywną sesję z powłoką «dash» przy pomocy polecenia «`script -T timing -c dash`». Wykonaj kilka poleceń i zakończ powłokę przy pomocy polecenia «`exit 42`», po czym odtwórz sesję przy pomocy polecenia «`scriptreplay -t timing`». Następnie uruchom polecenie powyższe polecenie przy pomocy «`strace -f -e read,write -o script.log`» i na podstawie zawartości pliku «script.log» pokaż jak «script» używa **pseudoterminala** do komunikacji z programami działającymi pod kontrolą powłoki «dash». Pokaż, że sterownik terminala przepisuje znaki zgodnie z flagami «ICRNL» i «ONLCR» opisanymi w `termios(4)`.

Zadanie 3. Uruchom **potok** (ang. *pipeline*) «`ps -ef | grep sh | wc -l > cnt`» w powłoce utworzonej przy pomocy polecenia «`strace -o pipeline.log -f dash`». Na podstawie zawartości pliku «pipeline.log» opisz jak powłoka realizuje funkcje łączenia procesów **rurami** (ang. *pipe*) i wykonuje przekierowanie **standardowego wyjścia** do pliku. W szczególności wskaż które procesy i w jakiej kolejności będą wołały następujące wywołania systemowe: `openat(2)` z flagą «O_CREAT» (realizuje `creat(2)`), `dup2(2)`, `pipe(2)`, `close(2)`, `clone(2)` (realizuje `fork(2)`) i `execve(2)`. Zwróć szczególną uwagę na to kiedy powłoka tworzy rury i kiedy są zamykane ich poszczególne końce.

Zadanie 4. Przyjrzyjmy się raz jeszcze plikowi «pipeline.log» z poprzedniego zadania. Zauważ, że wszystkie procesy należące do potoku muszą zostać umieszczone w jednej grupie procesów. Wskaż kiedy powłoka tworzy nową grupę procesów i jak umieszcza tam procesy realizujące potok. Przeczytaj [2, 34.2] i wyjaśnij czemu dla każdego podprocesu wywołanie `setpgid(2)` jest robione zarówno w procesie powłoki jak i w procesie potomnym? Kiedy powłoka ustala grupę pierwszoplanową przy pomocy `ioctl(2)` (realizuje `tcsetpgrp(3)`)? Na jakiej podstawie powłoka wyznacza kod wyjścia potoku?

Zadanie 5. Czemu nie można czytać i modyfikować katalogów przy pomocy wywołań `read(2)` i `write(2)`? Jakim wywołaniem systemowym można wczytać **rekord katalogu** (ang. *directory entry*)? Dlaczego zawartość katalogu nie jest posortowana? Wyświetl **metadane** katalogu głównego «/» przy pomocy polecenia «stat», a następnie wyjaśnij z czego wynika podana liczba **dowiązań** (ang. *hard link*)?

[1] https://en.wikipedia.org/wiki/ANSI_escape_code#CSI_(Control_Sequence_Introducer)_sequences

Zadanie 6. Intencją autora poniższego kodu było użycie plików jako blokad międzyprocesowych. Istnienie pliku o podanej nazwie w systemie plików oznacza, że blokada została założona. Brak tegoż pliku, że blokadę można założyć. Niestety w poniższym kodzie jest błąd TOCTTOU[2], który opisano również w [4, 39.17]. Zlokalizuj w poniższym kodzie wyścig i napraw go! Opowiedz jakie zagrożenia niesie ze sobą taki błąd.

```
1  #include "csapp.h"
2
3  bool f_lock(const char *path) {
4    if (access(path, F_OK) == 0)
5      return false;
6    (void)Open(path, O_CREAT|O_WRONLY, 0700);
7    return true;
8  }
9
10 void f_unlock(const char *path) {
11   Unlink(path);
12 }
```

Wskazówka: Przeczytaj komentarze do flagi «O_CREAT» w podręczniku do open(2).

Ściągnij ze strony przedmiotu archiwum «so21_lista_4.tar.gz», następnie rozpakuj i zapoznaj się z dostarczonymi plikami.
UWAGA! Można modyfikować tylko te fragmenty programów, które zostały oznaczone w komentarzu napisem «TODO».

Zadanie 7. Program «leaky» symuluje aplikację, która posiada dostęp do danych wrażliwych. Pod deskryptorem pliku o nieustalonym numerze kryje się otwarty plik «mypasswd». W wyniku normalnego działania «leaky» uruchamia zewnętrzny program «innocent» dostarczony przez złośliwego użytkownika.

Uzupełnij kod programu «innocent», aby przeszukał otwarte deskryptory plików, a następnie przepisał zawartość otwartych plików do pliku «/tmp/hacker». Zauważ, że pliki zwykłe posiadają **kursor**. Do pliku wyjściowego należy wpisać również numer deskryptora pliku i ścieżkę do pliku, tak jak na poniższym wydruku:

```
1 File descriptor 826 is '/home/cahir/lista_4/mypasswd' file!
2 cahir:...:0:0:Krystian Baclawski:/home/cahir:/bin/bash
```

Żeby odnaleźć nazwę pliku należy wykorzystać zawartość katalogu «/proc/self/fd» opisaną w procfs(5). Potrzebujesz odczytać plik docelowy odpowiedniego **dowiązania symbolicznego** przy pomocy readlink(2).

Następnie napraw program «leaky» – zakładamy, że nie może on zamknąć pliku z wrażliwymi danymi. Wykorzystaj fcntl(2) do ustawienia odpowiedniej flagi deskryptora wymienionej w open(2).

Zainstaluj pakiet «john» (John The Ripper[3]). Następnie złam hasło znajdujące się pliku, który wyciekł w wyniku podatności pozostawionej przez programistę, który nie przeczytał uważnie podręcznika do execve(2).

Wskazówka: Procedura «dprintf» drukuje korzystając z deskryptora pliku, a nie struktury «FILE».

Zadanie 8. Uruchom program «mkholes», a następnie odczytaj **metadane** pliku «holes.bin» przy pomocy polecenia stat(1). Wszystkie pola struktury «stat» są opisane w stat(2). Oblicz faktyczną objętość pliku na podstawie liczby używanych bloków «st_blocks» i rozmiaru pojedynczego bloku «st_blksize» systemu pliku. Czemu liczba używanych bloków jest mniejsza od tej wynikającej z objętości pliku z pola «st_size»? Czemu jest większa od liczby faktycznie używanych bloków zgłaszanych przez «mkholes»? Wyjaśnij to zjawisko na podstawie [1, 3.6].

[2] https://www.usenix.org/legacy/event/fast05/tech/full_papers/wei/wei.pdf
[3] https://www.openwall.com/john/

Literatura

[1] „*Advanced Programming in the UNIX Environment*”
W. Richard Stevens, Stephen A. Rago
Addison-Wesley Professional; 3rd edition; 2013

[2] „*The Linux Programming Interface: A Linux and UNIX System Programming Handbook*”
Michael Kerrisk
No Starch Press; 1st edition; 2010

[3] „*Systemy operacyjne*”
Andrew S. Tanenbaum, Herbert Bos
Helion; wydanie czwarte; 2015

[4] „*Operating Systems: Three Easy Pieces*”
Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
https://pages.cs.wisc.edu/~remzi/OSTEP/