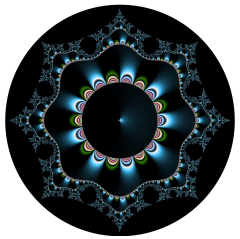
Sécurité des applications

Gestion des ressources

Thibaut et Corinne HENIN



www.arsouyes.org
@arsouyes

INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
CENTRE VAL DE LOIRE

Sommaire

- **Accès concurrents**
  - Base de donnée
  - Système de fichiers
  - Gestionnaire de signaux
  - Mutexes
- **Épuisement des ressources**
  - Complexité
  - Divers

Accès concurrents

Situation de compétition

Principe

- Accès concurrents
    - Ordre des opérations change le résultat
- « TOCTTO »
    - Time Of Check To Time Of Use

Base de donnée

Code vulnérable

```
<?php

$a     = User::retrieve($_GET["a"]) ;
$b     = User::retrieve($_GET["b"]) ;
$price = intval($_GET["price"]) ;

if (   $price    <= 0
     || $a->money <  $price
     || $a->id    == $b->id) {
     throw new \Exception("Not allowed") ;
}

$a->money -= $price ;
$b->money += $price ;

$a->persists() ;
$b->persists() ;
```

Protection

- Exclusion mutuelle :
    - Utiliser des « transactions »
    - Utiliser des « locks »

- Attention :
    - Dépend des bases
    - Impact sur les performances

- DBA, c’est un métier

Fichiers

Algorithme vulnérable

```php
<?php

if (is_file($filename)) {
    $content = file_get_contents($filename) ;
}
```

```php
<?php

while (true) {
    touch($filename) ;
    unlink($filename) ;
    symlink("/etc/passwd", $filename) ;
    unlink($filename)
}
```

Protection

- Préférer les « file handlers » aux noms de fichiers
    - *i.e. (f)*open, fstat, fchmod, …
- Verrouiller les fichiers
    - i.e. open(O_EXCL), flock()
- Fonctions atomiques
    - *i.e.* mv

Sig Handlers

Non atomicité

```
#include <stdio.h>
#include <signal.h>

void sighndlr() {
    printf("UID: %d\n", getuid());
    /* other cleanup code... */
}

int main() {
    int origuid = getuid();
    signal(SIGINT, sighndlr);
    setuid(0);
    sleep(5);
    setuid(origuid);
    return(0);
}
```

Code non réentrant

```
#include <stdio.h>
#include <signal.h>

char * buffer = NULL ;

void sighndlr() {
  free(buffer) ;
  sleep(10) ;
  buffer = NULL ;
}

int main() {
    int origuid = getuid();
    signal(SIGINT, sighndlr);
    buffer = (char*) malloc(512) ;
    sleep(10);
    return(0);
}
```

Protection

- Uniquement des fonction ré-entrantes
- Bloquer les signaux dans le gestionnaire
- Bloquer les signaux dans les phases critiques

Portée des variables

Singletons

```
public class GuestBook extends HttpServlet {
  String name ;

  protected void doPost(
    HttpServletRequest req,
    HttpServletResponse res) {
      name = req.getParameter("name") ;
      // ...
      out.println(
        name + ", thanks for visiting !") ;
  }
}
```

Sessions

SomeView.php

```
<?php

session_start() ;

$_SESSION["current"] =
    $_SERVER["REQUEST_URI"] ;
```

SomeForm.php

```
<?php

session_start() ;

doStuff($_GET, $_POST) ;

header(
   "Location: "
   . $_SESSION["current"]) ;
```

Sessions bis

```
Login() {
    Session["Username"] = Username.Text ;
    Session["Password"] = Password.Text ;
    If CheckLogin() {
        Session["Authed"]=TRUE;
    } else {
        Session["Username"] = "";
        Session["Password"] = "";
    }
}
```

```
LoadUserData() {
    If !(Session["Authed"]=TRUE)
        return FALSE;

    GetUserDataFromDB(
       Session["Username"]
       );

    //Display user data
    Return TRUE;
}
```

Source : https://www.owasp.org/images/8/8e/OWASP_NZDay_2011_BrettMoore_ConcurrencyVulnerabilities.pdf

Protection

- Maîtriser la portée des variables
    - « Et si deux threads sont en parallèles ? »
- Principe des Singletons :
    - *« Coupable jusqu'à preuve du contraire »*
- Et si vraiment …
    - Thread safety : *mutexes*

Gestion des mutex

Éviter le problème

- **Objets immuables**
  - Ne changent pas de valeur
  - Manipuler des copies

- **Fonctions et objets « thread safe »**
  - Peuvent être appelées en parallèle

Mutexes

- **Niveau « threads »** (même processus)
    - *E.g.* pthread_mutex, pthread_cond
- **Niveau processus** (même système)
    - Généralement associé à la ressource
    - *E.g.* les fichiers via « flock »
- **Niveau réseau**
    - Fournis par la gestion des messages entre agents
    - *i.e.* Elasticsearch (lol)

Dead Lock 1/2

```
<?php

function someCriticalStuff() {
   $file = fopen("/tmp/someFile", "c") ;

   while (! flock($file, LOCK_EXCL)) {
      ;
   }

   // Do critical stuffs
}
```

Dead Lock 2/2 (with RAII)

```php
<?php

class MyLock {
    private $fp ;

    public function __construct($filename) {
        $this->fp = fopen("/tmp/$name", "c") ;

        flock($this->fp, LOCK_EXCL) ;
    }

    public function __destruct() {
        fclose($this->fp) ;
    }
}
```

```php
<?php

function critical_one() {
    $first  = new MyLock("first") ;
    $second = new MyLock("second") ;

    // do stuffs
}

function critical_two() {
    $second = new MyLock("second") ;
    $first  = new MyLock("first") ;

    // do stuffs
}
```

Épuisement des ressources

Complexité

Complexité des algorithmes

Classe	Problème
O(1)	Opération de base, Accès tableau
O(log n)	Recherche dichotomique, accès dans un indexe
O(n)	Parcours d'une liste (naïve)
O(n log n)	Tri fusion (ksort)
$O(n^2)$	la plupart des tris, jointure naïve
$O(n^3)$	Multiplication de matrices
$O(2^{poly(n)})$	Sac a dos (force brute)
O(n!)	Voyageur de commerce (naïve)

Source : https://fr.wikipedia.org/wiki/Analyse_de_la_complexit%C3%A9_des_algorithmes

Complexité et sécurité

- Structure + algorithme
    - complexité
- Maîtrise des entrées
    - Déni de service

ReDoS
Regular **e**xpression **D**eny **o**f **S**ervice

- Principe :
  - Certaines expression ont une complexité exponentielle
    - (a+)+, (a|aa)+, …
  - Si l'application utilise une expression de ce genre
    - *E.g.* OWASP, java classname : ^**(([a-z])+.)+**[A-Z]([a-z])+$
  - Des entrées choisies peuvent ralentir fortement l'application

Source : https://en.wikipedia.org/wiki/ReDoS

Divers

Mémoire - Malloc

- Cas classique
    - Oubli de libération => fuite

- Protection
    - Ulimit –v (protéger les autres)
    - Valgrind (pour détecter les fuites)
        - Ne pas oublier le « long terme »

File Handlers

- Similaire à la mémoire
    - Oubli de fermeture des descripteurs

- Protection :
    - Ulimit –n
    - /proc/sys/fs/file-max

Processus – fork bomb

- Naïf :
  - Bash « `:(){ :|:& };:` »
  - Mais il y a généralement mieux à faire ;-)

- Bug :
  - Fork + pipe/dup + execve
    - Oubli du « exit »
  - Commande erronée => fork bomb
    - Il y a souvent mieux à faire aussi ;-)

Processus – Bad Design

- Principe :
  - Serveur avec N threads
  - Chaque thread gère une connexion

- Attaque « slowloris »
  - Établir des connexions HTTP partielles
  - Les threads attendent la suite (qui vient lentement)

- Protection :
  - Un thread ne traite qu'une tâche prête
  - Sinon … système de timeout, de nettoyage, …

Zip Bomb

- Archive de petite taille
    - Décompression de très grande taille

- 42.zip
    - 42,374 bytes
    - Contient 5 niveau de compression
    - Chaque niveau contient 16 fichiers
    - Le fichier de base fait 4,3 GB
    - Total décompressé : 4,5 PB

Divers

- Débit réseau
- Espace disque
  - E.g. fichiers journaux, base de données
- Identifiants
  - SQL int max = 2^31 – 1
  - IPv4 ;-)