Cybersecurity Institute
Univ. Grenoble Alpes

Vulnérabilités Logicielles

Pourquoi ? Comment ?

Laurent Mounier

VERIMAG - PACSS

21 Février 2017

Le plan

Qu'est-ce qu'une vulnérabilité dans un programme ?

Comment cela fonctionne t-il ?

Pourquoi y en a t-il autant ?

Comment s'en prémunir ?

Quel matériel faut-il pour exécuter un programme ?

Plateforme d'exécution :

ordinateur, téléphone, "objet connecté", automate industriel, voiture, etc.

→ Fournit les ressources suivantes :

- une "unité de traitement et de calcul" (processeur, CPU)
- de la mémoire interne
- des interfaces de communications vers/depuis l'extérieur
  - clavier, caméra, micro, capteurs, etc
  - écran, haut-parleur, actionneurs,
  - réseau (Internet, ou autre)
  - mémoires "externe" (disques durs, clés USB, etc.)
  - ...

Ressources partagées entre plusieurs applications/utilisateurs ...

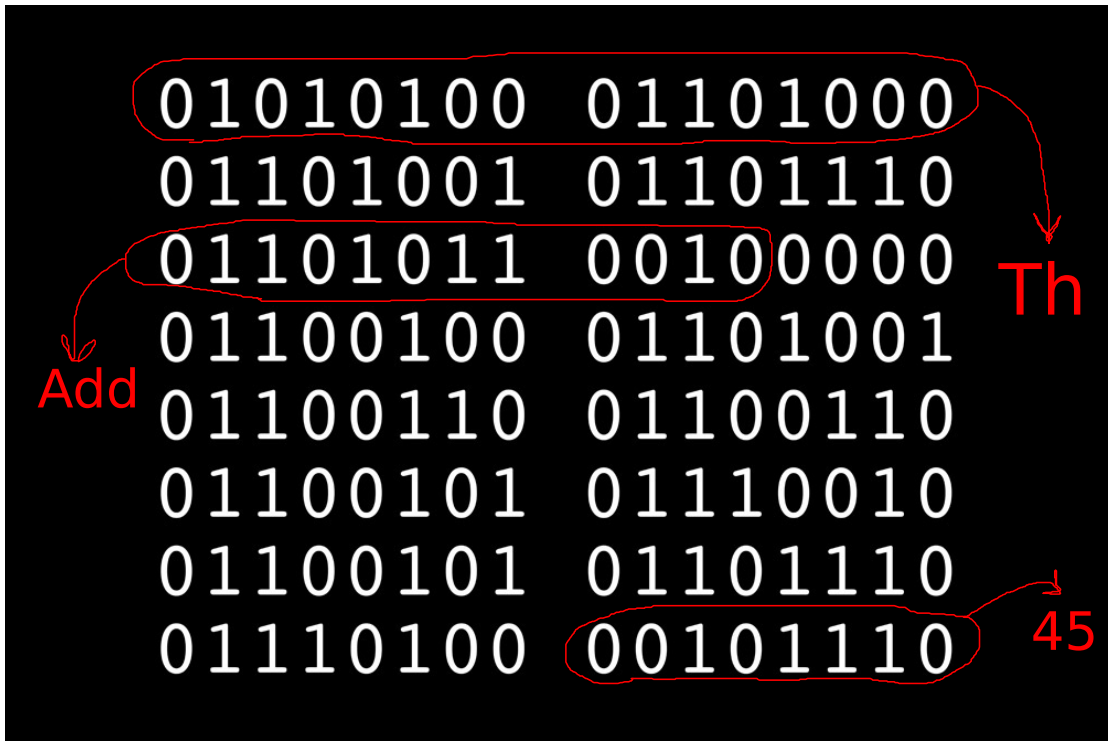
Comment s'exécute un programme ? (1/2)

La mémoire interne :

- contient les **instructions** et les **données** du programme
- mode de codage unique : la base 2
  un contenu mémoire $\Rightarrow$ **plusieurs interprétations possibles** !
  Exemple : un entier codé base 2
  $\hookrightarrow$ une instruction, un entier décimal, du texte , des pixels, ...

La mémoire interne :

- contient les **instructions** et les **données** du programme
- mode de codage unique : la base 2
  un contenu mémoire $\Rightarrow$ **plusieurs interprétations possibles** !
  Exemple : un entier codé base 2
  $\hookrightarrow$ une instruction, un entier décimal, du texte , des pixels, ...



Comment s'exécute un programme ? (2/2)

Le processeur/CPU

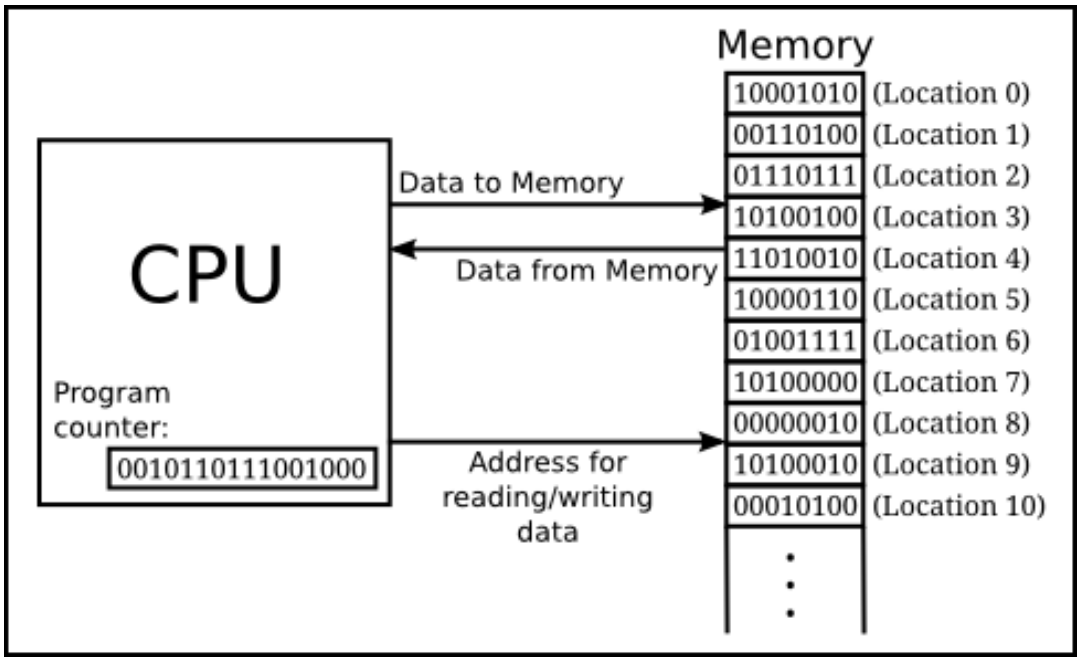
1. lit une instruction en mémoire (ex : additionner 2 entiers)
2. exécute cette intruction (avec lecture/écriture en mémoire)
3. choisit l'instruction suivante à exécuter ...

   ... en fonction du contenu courant de la mémoire !

Le processeur/CPU

1. lit une instruction en mémoire (ex : additionner 2 entiers)
2. exécute cette intruction (avec lecture/écriture en mémoire)
3. choisit l'instruction suivante à exécuter ...

   ...en fonction du contenu courant de la mémoire !



L'attaquant, ses objectifs, ses moyens . . .

Sécurité = confidentialité, intégrité, disponibilité, . . .

attaques directes sur la mémoire = lire et/ou modifier une donnéee
(ex : *write-what-where condition*)
→ déni de service, fuite d'information, exécution arbitraire, etc

L'attaquant, ses objectifs, ses moyens . . .

Sécurité = confidentialité, intégrité, disponibilité, . . .

attaques directes sur la mémoire = lire et/ou modifier une donnéee
(ex : *write-what-where condition*)
→ déni de service, fuite d'information, exécution arbitraire, etc

Différents modèles d'attaquant

- utilisateur "ordinaire" . . . mais malintentionné !
  → utilise les entrées/sorties standards du pgm

L'attaquant, ses objectifs, ses moyens . . .

Sécurité = confidentialité, intégrité, disponibilité, . . .

attaques directes sur la mémoire = lire et/ou modifier une donnéee
(ex : *write-what-where condition*)
→ déni de service, fuite d'information, exécution arbitraire, etc

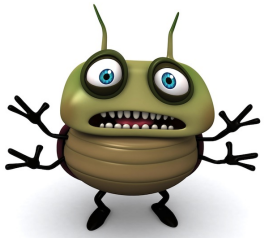
Différents modèles d'attaquant

- utilisateur "ordinaire" . . . mais malintentionné !
  → utilise les entrées/sorties standards du pgm
- utilisateur plus "avancé" :
  - observer des **canaux cachés** :
    consommation, temps d'execution, mémoires partagées
  - perturber l'exécution par **injection de fautes**
    laser, champ magnétique, etc.

→ l'attaquant peut donc aussi être la plateforme d'exécution . . .

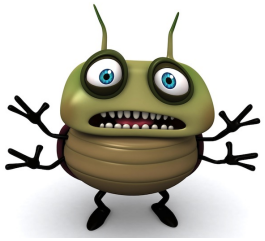
Exemple classique de vulnérabilité logicielle

Tout part d'un bug !



Exemple classique de vulnérabilité logicielle

Tout part d'un bug !



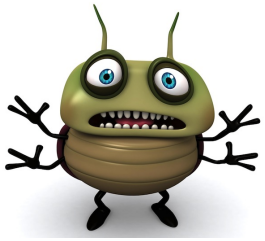
. . . avec pour **conséquences :**

- des exécutions non prévues par le programmeur
- des accès incorrects à la mémoire

$\Rightarrow$ fuite/altération de données ; exécution arbitraire de code

Exemple classique de vulnérabilité logicielle

Tout part d'un bug !



. . . avec pour **conséquences :**

- des exécutions non prévues par le programmeur
- des accès incorrects à la mémoire

$\Rightarrow$ fuite/altération de données ; exécution arbitraire de code

Exemple

1. lire une entrée x au clavier (ex : age de l'utilisateur)
2. allouer un zone mémoire M de taille x
3. utiliser M pour écrire des données utilisateur

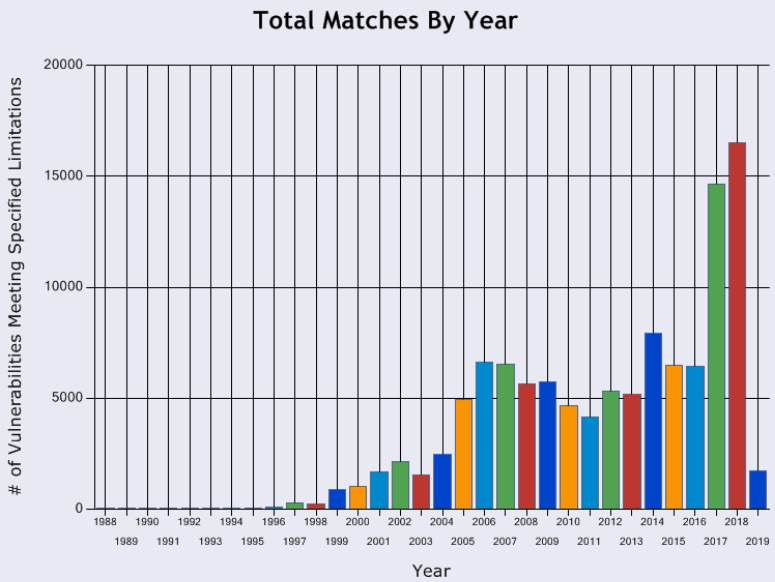
Bug possible : ne pas vérifier la pertinence de x (ex : $1 \leq x \leq 150$)
Si $x < 0$? si x très grand ?
$\rightarrow$ accès à M peut permettre une attaque sur la mémoire . . .

L'importance du phénomène . . . sur le plan quantitatif

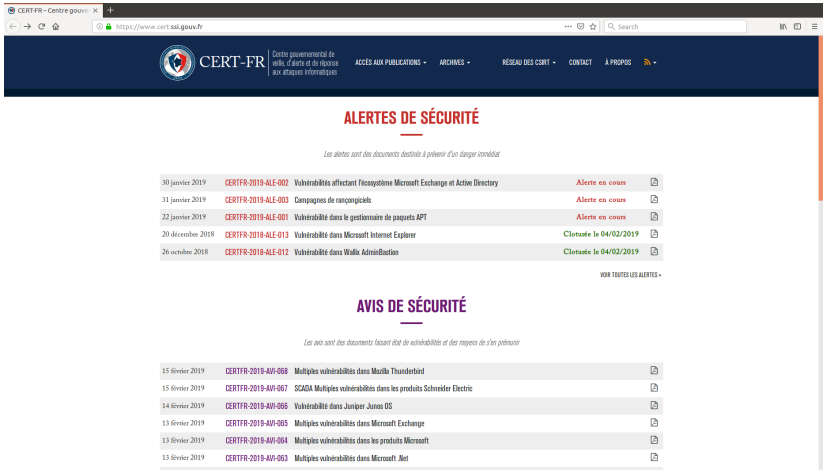
Veracode "State of Software Security Report 2018"

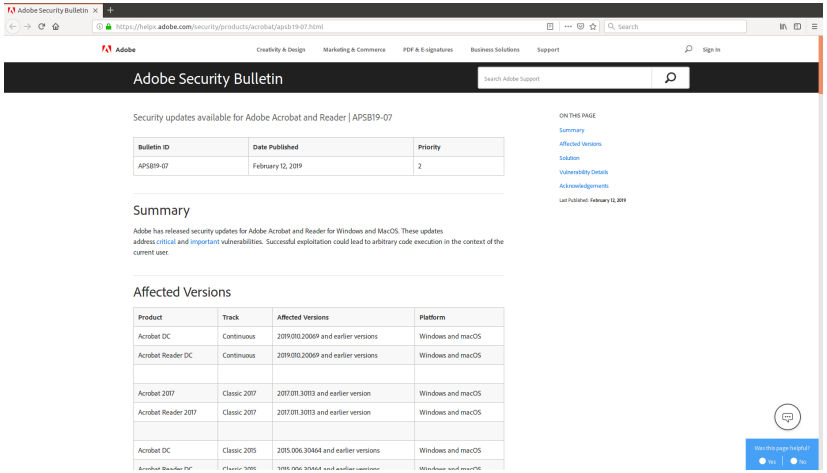
- More than 85% of all applications have at least one vulnerability in them; more than 13% have at least one critical severity flaw
- flaw persistence analysis:
  75% after 21 days ; 50% after 121 days, 25% after 472 days

NIST statistics on software vulnerabilities

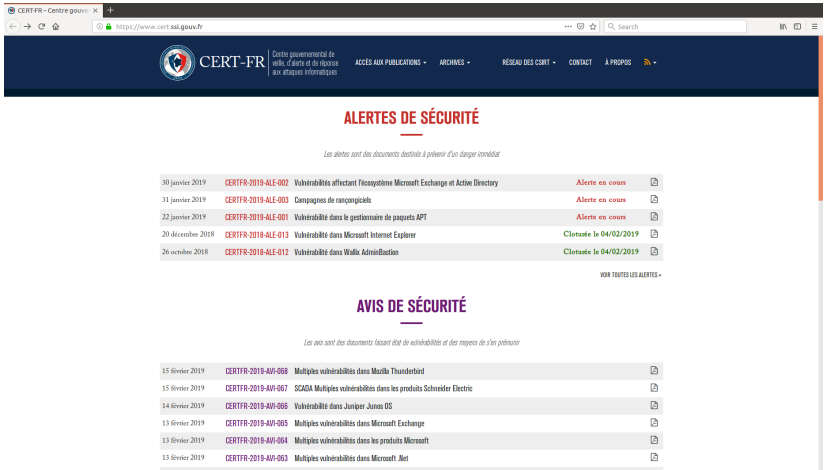


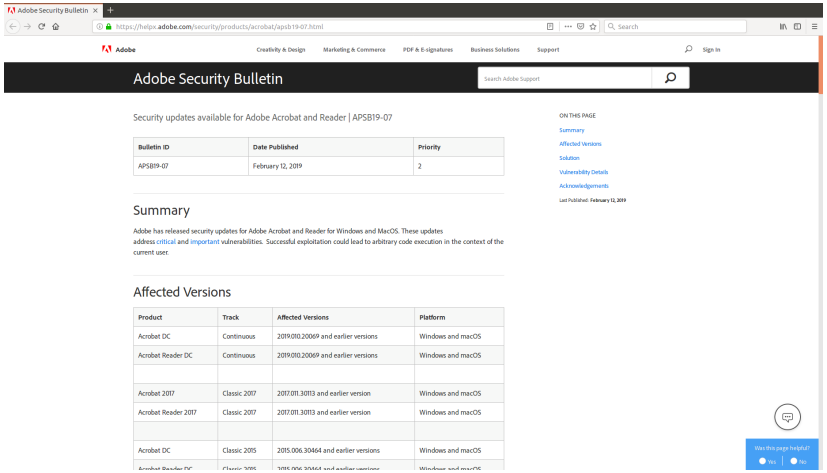
L'importance du phénomène . . . sur le plan qualitatif





L'importance du phénomène . . . sur le plan qualitatif





Deux catégories d'attaques

- attaquer un logiciel "de sécurité" :
  chiffrement, pare-feux réseau, authentification, etc
- attaquer un logiciel "général", non critique :
  navigateur, lecteur PDF, éditeur de texte, etc.

  $\hookrightarrow$ élévation de privilèges sur le plateforme :
  installer un malware, récupérer un fichier d'adresses, ...

  $\Rightarrow$ ratio surface d'attaque /gain bien plus élevé !

Pourquoi une telle situation ?

Persitance de **bugs** dans le programmes

- limites théoriques : analyse de code est indécidable
- programmation = conception d'un système **discret**
- poids du code existant, complexité des applications récentes

Sécurité logicielle = préoccupation récente !

- peu de prise de conscience des développeurs (time to market) ?
- faiblesses de certains langages de programmation
  expressivité vs contrôle vs performance
- pas (encore ?) de chaîne de développement sécurisé "standard"

Les moyens de défense

Collaboration internationnale

- Recensement des vulnérabilités : CERT, CVE et CWE
- Recherche "éthique" de vulnérabilité
  responsible disclosure vs *bug bounties*

Contre-mesures → protection du logiciel

- langages de programmation, règles de codage
- compilateur, outils d'analyse de code
- plateforme d'exécution
  système d'exploitation et/ou matériel "durci"

+ certification dans certains domaines (ex : cartes à puces)

Quelques pistes de recherche (Vérimag/PACSS)

Analyse de code pour la securité

- analyse de code “bas niveau”
- modèle d’attaquant
- passage à l’échelle, qualité des verdicts

exploitabilité des vulnérabilités

- quels bugs sont dangereux ?
- quantifier le coût d’une attaque ?

robustesse à l’injection de fautes

- aide au développeur : où durcir le code ?
- aide à l’évaluateur : quelles attaques possibles en multi-faute ?

Projets et personnes impliquées

Quelques projets en cours

IRT NanoElec (CLAPS, NanoTrust)
ANR Sacade (Sécurité des Systèmes Industriels)
SecureIoT-2

Collaboration CEA-List

Personnes impliquées dans Vérimag

Saddek Bensalem, Etienne Boespflug, Sylvain Boulmé, Pierre Corbineau, Cristian Ene, Claire Maiza, David Monniaux, Laurent Mounier, Marie-Laure Potet, Jean-Louis Roch, Valentin Touzeau