Secure Coding
Terminologie

Halim Djerroud

LISV
Laboratoire d'ingénierie
des systèmes de Versailles

UVSQ
université PARIS-SACLAY

révision : 0.1

Plan de cours

Programmation sécurisée (Secure Coding)

La programmation sécurisée (Secure Coding) consiste à prendre en compte la sécurité informatique à tous les moments de la conception, de la réalisation et de l'utilisation d'un programme informatique. Cela permet d'éviter au maximum les trous de sécurité et les bugs.

- Lors de la définition des besoins des logiciels
- Lors de la conception
- Lors de la réalisation (codage)
- Lors de l'exploitation (exécution)

La sécurité n'est pas quelque chose qu'on ajoute après coup...
...c'est quelque chose qui est intégré, au même titre que la qualité, l'évolutivité et les performances.

Conception

- Lors de la conception, il s'agit par exemple de concevoir le programme de façon modulaire afin de permettre une meilleur lisibilité et assurer la maintenabilité. (voire la norme : ISO/IEC 25010:2011)
- Attribuer le moins de droits utilisateurs possible

Réalisation

- Ensuite, lors de la réalisation, il faut penser à bien valider les données entrées par l'utilisateur pour éviter toutes les attaques du type dépassement de tampon (buffer overflow), les injections SQL, l'exploitation de mauvaises utilisations des chaînes de formatage (format string attacks), les dépassements d'entiers (integer overflow), etc.

Exécution

- Enfin lors de l'exécution, il faut penser par exemple à appliquer les différentes mises à jour de sécurité lorsqu'elles sortent. Pour ce faire, il peut être pratique de la part du concepteur de l'application de proposer un système de mise à jour simplifié de l'application.

Triade CIA

L'élément clé de la sécurité est la triade InfoSec, également appelée triade CIA (Confidentiality, Integrity, Availability). C'est un modèle de sécurité qui permet d'assurer la sécurité des données d'une organisation ou d'une structure professionnelle. Ces trois principes que sont :

- La **confidentialité** signifie qu'aucun utilisateur non autorisé ne peut lire des données.
- L'**intégrité** signifie qu'aucun utilisateur non autorisé ne peut écrire des données.
- Le système doit être **disponible** pour les utilisateurs autorisés à lire et à écrire des données.

Un système qui possède tous les éléments de la triade InfoSec en toutes circonstances est considéré comme sûr.

En pratique, pour la plupart des logiciels, il y aura souvent des circonstances particulières dans lesquelles un ou plusieurs des éléments ne seront pas respectés.

Les tests de sécurité

- Les tests de sécurité sont différents des du test logiciels classique
- Il faut imaginer des adversaires *intelligents*, *pervers*, utilisant des méthodes déloyales (trahit un accord)
- Tout est permit pour trouver une faille dans votre système

Les Challenges des tests de sécurité

Pour tester la sécurité de votre logiciel, vous devez :

- Réfléchir comme vos adversaires
- Pensez que votre système ne sera pas utilisé comme vous l'avez prévu
- Arrêtez de penser comme une honnête programmeur
- S'il y avait des vulnérabilités qui pourraient être trouvées par une utilisation normal du logiciel, elles ont très probablement déjà été découvertes et corrigées
- Vous vous avez corrigé une faille de sécurité il faut s'assurer que cette faille n'est pas déjà exploitée

Les standards de sécurité

- ISO/IEC 27000 (Généralité) / 27001 (Implémentation) / 27002 (Bonnes pratiques : Codage) : Information Security Management Systems
- British Standard 7799 Part 3 (Gestion des risques)
- COBIT - The Control Objectives for Information and related Technology (Framework de gouvernance)
- ISO/IEC 15408
- ITIL (or ISO/IEC 20000 series) : Bonnes pratiques

Attaques actives et passives

- Attaques actives
  - Une attaque active tente de modifier les ressources du système ou d'affecter leur fonctionnement
  - Cause toujours des dommages au système
  - Modification de l'information
  - La transmission est capturée en contrôlant physiquement la partie d'un lien
- Attaques passives
  - L'attaque passive tente de lire ou d'utiliser les informations du système mais n'influence pas les ressources du système.
  - Ne provoque aucun mal direct au système
  - Pas de modification de l'information
  - L'attaquant a juste besoin d'observer la transmission

Attaques : Disponibilité

- Interruptions (physique)
- Denial of service (DoS)

Attaques : Confidentialité

- Interception (accès aux données par des utilisateurs non autorisés)
- packet sniffing

Attaques : Intégrité

Il existe deux types :

- Modification de données
- Fabrication de données

Exploitation de la vulnérabilité

- L'exploitation de la vulnérabilité est une technique ou un mécanisme qui est utilisé pour compromettre l'un des éléments de la Triade (CIA) de la sécurité de l'information d'un système. Cela peut aller de la connaissance d'un mot de passe par défaut à des morceaux de logiciels complexes qui interagissent avec le système de manière connue pour provoquer un comportement indésirable.

Les outils : Malware

- Un logiciel malveillant (Malware) est un logiciel délibérément conçu pour avoir un effet indésirable sur un système informatique, généralement à l'insu de l'utilisateur autorisé du système.

Les outils : Exemple Malware

- **Bactéries :** programme qui consomme une quantité excessive de ressources système, occupant peut-être tous les descripteurs de fichiers ou l'espace disque.
- **bombe fork :** Un type spécial de bactérie qui se divise continuellement, entraînant l'utilisation de toutes les ressources du processeur, créant ainsi davantage de copies de la bombe fork.
- **Logic bomb (Bombe logique) :** code dans un programme qui exécute une fonction non autorisée, comme la suppression de toutes les données le premier jour du mois.
- **Trapdoor (porte dérobée) :** programme ou élément de programme qui fournit accès secret à un système ou une application.

Les outils : Exemple Malware

- **Cheval de Troie (Trojan Horse) :** logiciel qui prétend en être un autre afin d'inciter les utilisateurs à l'installer et à l'exécuter. Par exemple, un cheval de Troie peut déclarer qu'il contient différents curseurs de souris amusants, mais après l'avoir installé, il supprime tout ce qui se trouve sur votre disque dur.
- **Virus :** programme informatique, souvent petit, qui se réplique avec l'intervention humaine. Cette intervention peut consister par exemple à cliquer sur un lien ou à exécuter un programme qui vous a été envoyé en pièce jointe.
- **Ver (Worm) :** programme informatique, souvent petit, qui se réplique sans intervention humaine. Par exemple, une fois installé sur une machine, cette machine peut tenter de s'introduire dans d'autres machines et de copier le code du ver sur d'autres.

Les outils : Exemple Malware

- **Zombie :** Un ordinateur sur lequel un logiciel est installé qui permet à des utilisateurs non autorisés d'y accéder pour exécuter des fonctionnalités non autorisées. Par exemple, un système peut avoir un programme de messagerie intégré qui permettra à d'autres utilisateurs d'envoyer du spam depuis votre ordinateur, de sorte que les expéditeurs réels ne puissent pas être suivis.
- **Réseau de robots (Bot Net) :** une collection de zombies contrôlés par un maître.

Les outils : Exemple Malware

- **Logiciel espion :** Logiciel qui surveille furtivement les actions de l'utilisateur du système. Par exemple, un logiciel *keylogger*.
- **Outils DoS :** outils qui permettent des attaques par déni de service.
- **Ransomware :** Logiciel qui effectue une action indésirable (par exemple, le chiffrer vos fichiers) et demande de l'argent généralement en monnaie électronique (Monero par exemple) ou une autre compensation afin de récupérer les données.

Méthodes de sécurité

- **La sécurité par l'obscurité :** s'appuyer sur le fait que les attaquants ne savent pas que quelque chose est nécessaire pour vous nuire.
- Forcer les utilisateurs à se connecter à votre système avant d'effectuer des opérations sensibles
- Utilisez des protocoles sécurisés (https, etc.) pour empêcher le *sniffing*
- Forcer les utilisateurs à utiliser des mots de passe forts

Principe du moindre privilège

- Accorder juste assez d'autorité pour faire le travail (rien de plus !)
  - accès *root* uniquement pour les programmes utils
  - fermer les ports non utils ...

Élimination sélective

- Encodage (échappement par exemple) et filtrage des entrées utilisateur non fiables avant de les accepter dans un système fiable
  Exemple :
  - Assurez-vous que les données acceptées sont du bon type, du bon format, de la bonne longueur...
  - Interdire la saisie de données erronées dans un formulaire graphique.
  - Supprimez tout code SQL des noms d'utilisateurs soumis.
  - Encodez/filtrer le texte saisi qui est affiché à l'utilisateur

Vérifier que le code est sécurisé

- Avant que le code soit écrit:
  - prendre en compte la sécurité dans le processus de conception
- Pendant que le code est écrit:
  - révisions de code
  - audits de sécurité des codes
  - programmation en binôme
- Une fois le code écrit:
  - profilage / debuggage
  - audits de sécurité du système
  - tests de sécurité système/fonctionnels
  - tests d'intrusion

Audits de sécurité

Une série de contrôles et de questions pour évaluer la sécurité de votre système.

- Test d'intrusion : tentative ciblée (white-hat hackers) visant à compromettre la sécurité de votre système.
- Analyse des risques : évaluation des risques relatifs de ce qui peut mal se passer lorsque la sécurité est compromise.

Questions d'audit de sécurité

- Votre système nécessite-t-il une authentification sécurisée avec des mots de passe ?
- Les mots de passe sont-ils difficiles à déchiffrer ?
- Existe-t-il des listes de contrôle d'accès (ACL) en place sur les périphériques réseau ?
- Existe-t-il des journaux d'audit pour enregistrer qui accède aux données ?
- Les journaux d'audit sont-ils examinés ?
- Les paramètres de sécurité de votre système d'exploitation sont-ils conformes aux niveaux acceptés par l'industrie ?
- Toutes les applications et services inutiles ont-ils été éliminés ?
- Tous les systèmes d'exploitation et applications sont-ils mis à jour aux niveaux actuels ?
- Comment les supports de sauvegarde sont-ils stockés ? Qui y a accès ? Est-ce que c'est à jour?
- Existe-t-il un plan de reprise après sinistre ? A-t-il déjà été répété ?
- Existe-t-il de bons outils cryptographiques pour régir le cryptage des données ?
- Les applications personnalisées ont-elles été écrites dans un souci de sécurité ?
- Comment ces applications personnalisées ont-elles été testées pour détecter les failles de sécurité ?

A propos des données

- Ces informations sont-elles de nature personnelle ou sensible ?
- Que fait mon application avec ces informations ?
- Où et sous quel format est-il enregistré ?
- Est-il envoyé sur le réseau ?
- (pour tout ce qui précède) Est-ce nécessaire ? Puis-je réduire ma consommation ?

Ou sont stockées les données

- Où votre application stocke-t-elle ses données et pourquoi ?
- Est-ce sécurisé ?

Quelques exemples relatif au WEB

- DoS
- Packet sniffing
- Password cracking
- Élévation de prévilèges
- Injection SQL
- Phishing : Il est difficile de tester que l'ingénierie sociale ne fonctionnera pas sur un système

Exemple : Buffer Overflow (débordement de tampon)

- Le langage C est un langage de programmation de haut niveau, mais il suppose que le programmeur est responsable de l'intégrité des données.

Exemple Buffer Overflow

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
  int value = 5; char buf1[8], buf2[8];
  strcpy(buf1, "one"); /* Put "one" into buf1 */
  strcpy(buf2, "two"); /* Put "two" into buf2 */
  printf("[AVANT] buf2 @ %p contains \'%s\'\n", buf2, buf2);
  printf("[AVANT] buf1 @ %p contains \'%s\'\n", buf1, buf1);
  printf("[AVANT] value @ %p is %d (0x%08x)\n", &value, value, value);
  printf("\n[STRCPY] copying %d bytes into buf2\n\n", strlen(argv[1]));
  strcpy(buf2, argv[1]); /* Copy first argument into buf2. */
  printf("[APRES] buf2 @ %p contains \'%s\'\n", buf2, buf2);
  printf("[APRES] buf1 @ %p contains \'%s\'\n", buf1, buf1);
  printf("[APRES] value @ %p is %d (0x%08x)\n", &value, value, value);
}
```

LISV Laboratoire d'Ingénierie des systèmes de Versailles — UVSQ université PARIS-SACLAY

Résultat 1

```
[AVANT] buf2 @ 0x7fff192c484c contains 'two'
[AVANT] buf1 @ 0x7fff192c4854 contains 'one'
[AVANT] value @ 0x7fff192c485c is 5 (0x00000005)

[STRCPY] copying 3 bytes into buf2

[APRES] buf2 @ 0x7fff192c484c contains 'abc'
[APRES] buf1 @ 0x7fff192c4854 contains 'one'
[APRES] value @ 0x7fff192c485c is 5 (0x00000005)
```

Résultat 2

```
$./a.out abcdefghijklmn
[AVANT] buf2 @ 0x7fff3932a37c contains 'two'
[AVANT] buf1 @ 0x7fff3932a384 contains 'one'
[AVANT] value @ 0x7fff3932a38c is 5 (0x00000005)

[STRCPY] copying 14 bytes into buf2

[APRES] buf2 @ 0x7fff3932a37c contains 'abcdefghijklmn'
[APRES] buf1 @ 0x7fff3932a384 contains 'ijklmn'
[APRES] value @ 0x7fff3932a38c is 5 (0x00000005)
```

Résultat 3

```
$ ./a.out  AAAAAAAAAAAAAAAAAAAAAAAAAAAA
[AVANT] buf2 @ 0x7ffdca89421c contains 'two'
[AVANT] buf1 @ 0x7ffdca894224 contains 'one'
[AVANT] value @ 0x7ffdca89422c is 5 (0x00000005)

[STRCPY] copying 29 bytes into buf2

[AFTER] buf2 @ 0x7ffdca89421c contains 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
[AFTER] buf1 @ 0x7ffdca894224 contains 'AAAAAAAAAAAAAAAAAAAA'
[AFTER] value @ 0x7ffdca89422c is 1094795585 (0x41414141)
Erreur de segmentation
```