Introduction à la Programmation Sécurisée

Denis Ducamp <dducamp@sii.fr>

SII
Buropolis A - Bâtiment A
150 rue Nicolas Vauquelin
31100 TOULOUSE

www.sii.fr

La sécurité chez SII

- Les Missions :
  - Définition d'architectures sécurisées
  - Assistance à maîtrise d'ouvrage
  - Expertise en sécurité des S.I. embarqués
  - …
- Le Département Technique Sécurité SII:
  - Cellule d'experts en sécurité
    > capitalisation du savoir, formation, support
  - Laboratoire sécurité
    > recherche et veille technologique

SII France

- Société de Conseil et d'Ingénierie
- Création en 1979
- 9 agences et 9 bureaux
- 1.800 personnes (30/03/06)

SII Toulouse

- Création en 2000
- Effectif : 300 personnes
- Métiers :
  - Conception et développement de systèmes embarqués
  - Réseaux et sécurité informatique
  - Ingénierie logicielle

Plan : intro. à la prog. sécurisée

- Objectifs principaux de la sécurité
  - Confidentialité, intégrité, disponibilité
  - Notions : sûreté de fonctionnement & program° sécurisée
- Vulnérabilités (failles)
  - Développement :
    - débordement de tampon, chaine de format, XSS...
  - Conception : vol de session/d'identité...
- Contre-mesure
  - Principe du moindre privilège
  - Séparation des privilèges
- Contre-mesures externes
- Audits de code statique et dynamique

Objectifs principaux de la sécurité

- **Confidentialité** :
  - Une ressource ne peut pas être accédée par des tiers non autorisés
- **Intégrité** :
  - Une ressource ne peut pas être modifiée ou supprimée par des tiers non autorisés
- **Disponibilité** :
  - Une ressource (ou un service) peut être accédée dans un temps raisonnable quand l'utilisateur en a besoin

Notions

- **Sûreté de fonctionnement**
  safe programming
  - Le système doit :
    - Ne générer que des données valides
    - Fonctionner lors de la réception de données valides

- **Programmation sécurisée**
  - Le système doit être capable de traîter toutes les données reçues sans produire d'erreur ayant une conséquence au niveau de la sécurité

Vulnérabilités (failles)

- Erreur ayant une conséquence au niveau de la sécurité
- Se divisent en deux catégories principales
  - Problème dans le développement
    - Exemples : débordement de tampon, chaîne de format...
    - Peut être corrigé rapidement
    - Les problèmes d'interopérabilité sont quasiment nuls
  - Problème dans la conception
    - Exemples : dans un protocole, un automate d'états fini
    - Peut nécessiter beaucoup de travail
    - Les problèmes d'interopérabilité sont très courants

Failles classiques

- **Détournement d'exécution vers du code choisi par l'attaquant**
  - Failles les plus importantes : l'attaquant prend le contrôle du serveur
    - Débordements de tampons
    - Chaînes de format
- **Changement de sémantique**
  - L'attaquant change la sémantique d'une requête SQL ou d'une commande système
- **Vol de session / d'identité**
  - Usurpation de l'identifiant d'une session / d'un utilisateur
- **Dénis de service** :
  - Aux niveaux réseau, applicatif...

Failles classiques : Débordement de tampons

- Dans la pile d'exécution sont empilés :
  - Les paramètres de la fonction appelée
  - L'adresse de retour dans la fonction appelante
  - Les variables locales de la fonction appelée
- Quand une variable locale « déborde »
  - Par exemple recopie (avec *strcpy()*...) d'une longue chaîne dans un tampon trop petit
  - L'adresse de retour est modifiée
    - Habituellement : *Segmentation fault - core dump*
  - Sauf si l'attaquant redirige vers son propre code
    - Appelé *shell-code* car habituellement *exec("/bin/sh")*

Exemple : Débordement de tampons

- vulnerable.c

```
void main(int argc, char *argv[]) {
  char buffer[512];
  if (argc > 1)
    strcpy(buffer,argv[1]);
}
```

- non-vulnerable.c

```
void main(int argc, char *argv[]) {
  #define LG 512
  char buffer[LG+1];
  if (argc > 1)
    strncpy(buffer,argv[1],LG);
    buffer[LG]='\0';
}
```

Failles classiques : Chaînes de format

- Les fonctions de mise en forme de texte nécessitent une chaîne de format
  - Pour spécifier le type et le format d'affichage de chaque donnée
- Si le programmeur ne spécifie pas cette chaîne alors le premier paramètre est utilisé comme chaîne de format
  - Quand il s'agit d'une donnée externe alors
    - La pile d'exécution peut être scannée (*%<val>$x*)
    - Et peut être modifiée (*%n*) dont l'adresse de retour
      - L'attaquant redirige alors vers son propre code

Exemple : Chaînes de format

- vulnerable.c
  - `snprintf(buffer,LG,argv[1]);`
- non-vulnerable.c
  - `snprintf(buffer,LG,"%s",argv[1]);`
- Fonctions potentiellement vulnérables
  - *printf :
    - printf(), fprintf(), sprintf(), snprintf()
    - vprintf(), vfprintf(), vsprintf(), vsnprintf()
  - syslog(int priority, const char *format, ...);
    - vsyslog(int priority, const char *format, va_list ap);

Failles classiques : Failles web

- XSS (*Cross Site Scripting*) :
  - exécution de code sur un client tiers via un serveur de confiance
- Injection SQL / de commande :
  - changement de sémantique d'une requête SQL / d'une commande système
- Vol de session / d'identité :
  - usurpation de l'identifiant d'une session / d'un utilisateur
- *Directory traversal* :
  - accès hors de l'arborescence d'origine

Failles classiques : Dénis de service

- Au niveau réseau
  - Saturation de la bande passante d'une connexion
- Au niveau système
  - Saturation de la mémoire physique
  - Saturation des processeurs
  - Saturation de l'espace de stockage, temporaire ou pas
  - Saturation des journaux
- Au niveau applicatif
  - Plantage
  - Changement de la configuration
  - etc...

Contre-mesures

- **Moindre privilège** :
  - Un processus s'exécute avec le minimum de privilèges dont il a besoin pour mener à bien sa tâche
- **Séparation des privilèges** :
  - Plusieurs processus, avec des privilèges différents, coopèrent pour mener à bien une tâche
  - Un processus de moindre privilège peut demander à un processus plus privilégié d'effectuer des tâches pour lui, mais le 2nd ne fait pas confiance au 1er.

Contre-mesures externes

- Permettre de rendre impossible (très difficile) certaines exploitations de certaines failles
  - Pile non exécutable : pages écriture ou exécution
  - Adresses aléatoires : pile, tas, bibliothèques, exé.
  - Détection des débordements de tampons
    - Option /GS de Visual Studio, patch SSP à GCC d'IBM
- Limiter les conséquences (niveaux systèmes et réseaux) de l'exploitations de failles
  - Exécution dans une cage : chroot(), VM...
  - Système de confiance : privilèges, MAC...
  - Architecture de sécurité : DMZ, filtrage IP, relais...

Audits de code statique

- Détection de failles potentielles par analyse statique du code source
- Nombreux faux positifs/négatifs possibles
  - flawfinder
    - http://www.dwheeler.com/flawfinder/
  - rats
    - http://www.securesoftware.com/download_rats.htm
  - splint
    - http://lclint.cs.virginia.edu/
  - its4 (NON-COMMERCIAL LICENSE)
    - http://www.cigital.com/its4/

Audits de code dynamique

- Outils d'aide à l'audit dynamique
  - Problèmes de mémoire
    - valgrind http://valgrind.kde.org
  - Problèmes sur les arguments
    - bfbtester http://bfbtester.sourceforge.net/
    - fuzz http://fuzz.sourceforge.net/
  - Problèmes sur les entrées/sorties fichiers/réseau
    - zzuf http://sam.zoy.org/zzuf/
  - Fuzzer générique
    - spike / spike proxy (web) / sharefuzz (var. d'env.)
      - http://www.immunitysec.com/resources-freesoftware.shtml

Audits de code dynamique Exemple 1/2

- bfbtester http://bfbtester.sourceforge.net/

```
$ ls -lL vulnerable
-rwsr-sr-x 1 root daemon 13785 Dec 19  2002 vulnerable
$ ./bfbtester -s vulnerable
=> /home/ducamp/bfbtester-2.0.1/src/bfbt/vulnerable
 (setuid: 0)
 (setgid: 2)
   * Single argument testing
Cleaning up...might take a few seconds
*** Crash </home/ducamp/bfbtester-2.0.1/src/bfbt/vulnerable> ***
args:          [51200]
envs:
Signal:        11 ( Segmentation fault )
Core?          No
$ ./vulnerable `perl -e 'print "A"x51200'`
Segmentation fault
```

Audits de code dynamique Exemple 2/2

- zzuf http://sam.zoy.org/zzuf/

```
$ ./zzuf -qv -s0:1000 -r0.001:0.1 -S -c -T 6 -C 3 mplayer -- -nosound -vo
   null -benchmark Monkeywithadeathwish.wmv
zzuf[s=0,r=0.001:0.1]: signal 11 (SIGSEGV)
zzuf[s=13,r=0.001:0.1]: signal 6 (SIGABRT)
zzuf[s=16,r=0.001:0.1]: signal 11 (SIGSEGV)
$ ./zzuf -qv -s16 -r0.001:0.1 -S -c -T 6 -C 3 < Monkeywithadeathwish.wmv
   > Monkeywithadeathwish-zzuf.wmv
zzuf[s=16,r=0.001:0.1]: reading from stdin
$ mplayer -nosound -vo null -benchmark Monkeywithadeathwish-zzuf.wmv
MPlayer 1.0rc1-4.1.1 (C) 2000-2006 MPlayer Team
Selected video codec: [ffwmv1] vfm: ffmpeg (FFmpeg M$ WMV1/WMV7)
[...]
[wmv1 @ 0x8829db8]concealing 432 DC, 432 AC, 432 MV errors
V:  13.9 219/219  0%  0%  0.0% 0 0
MPlayer interrupted by signal 11 in module: video_read_frame
```

Conseils

- Documenter les données en entrée et sortie
- Être « lâche » avec ce qui est lu
  - Et être « strict » avec ce qui est écrit
- Vérifier les données en entrée avant utilisation :
  - En type : ensemble de caractères autorisés
  - Et en longueur
- Échapper les données en sortie
  - Si elles peuvent être interprétées par le client :
    - Métacaractères, mots clés...

Conseils

- Gérer ses ressources
  - Notamment le nombre de connexions simultanées et par laps de temps
- Ne pas produire de messages d'erreurs trop explicites vers l'utilisateur
  - Gérer les erreurs d'applications tierces pour retourner des messages génériques à l'utilisateur
- Journaliser
  - Tous les événements importants
  - Les messages d'erreur détaillés

Références

- Secure Programming for Linux and Unix HOWTO -- Creating Secure Software - David A. Wheeler
  - http://www.dwheeler.com/secure-programs/
  - + liens depuis cette page et la page principale
- OWASP : find and fight the causes of insecure software
  - http://www.owasp.org/
- Programmation Sécurisée - Frédéric Raynal
  - http://www.security-labs.org/
- Comment concevoir des applications sécurisées basées sur la séparation des privilèges
  - http://www.hsc.fr/ressources/presentations/privsep/index.html.fr
- Secure Programming Mailing List – Archive et FAQ :
  - http://www.securityfocus.com/archive/98
  - http://www.securityfocus.com/archive/98/description
- Writing Secure Code – Howard Leblanc
  - Microsoft Press