Développement sécurisé

•••

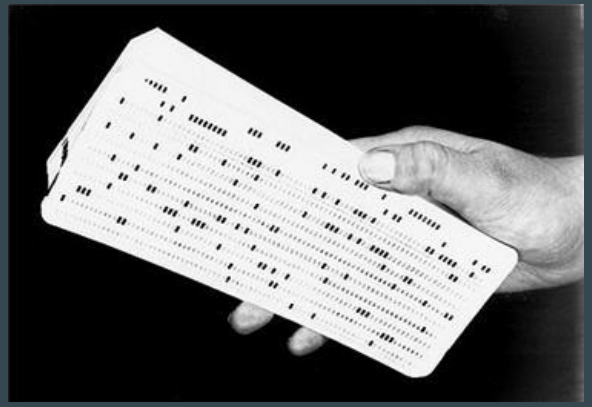
Il n’y a pas que le code dans la vie...

Plan

- Compilation
- Système d’exploitation

Compilation

- Base de l’informatique moderne
- Théorie des langages structurées et avancées
- Vecteur fondamentale de l’avancé de l’informatique moderne
  - Carte perforées
  - Langage ASSEMBLEUR (maintenant vous êtes des experts)
  - Langage haut niveau (langage C)
- Compilateur performant
  - GCC
  - LLVM
  - ... peut être même visual studio



Compilation

- Notion avancé en informatique théorique
- Théorie des automates
  - Machine de turing modèle le plus abstrait représentant la calculabilité d'un problème
  - Automates à pile permettant la représentation des langages non contextuelle
- Théorie de la complexité
  - En espace
  - Temporelle
- Preuve des programmes
  - Le langage ADA et la fameuse ligne 14 de métro parisien
  - Le langage B

Compilation

- Vérification statiques
  - Langage typés
  - Type Casting sécurisé
  - static_assert C++11
  - constexpr C++11
- Vérification dynamique
  - Canarie de pile

Compilation

Vérification statique

- Certainement le plus grosse plus value des compilateurs
- Arrivé des langages haut niveau dit fortement typé

```
float a = 0.0f;
int b = a + 2;
```

- Faire attention au phénomène de integer promotion et conversion explicite

```
int si = -1;
unsigned int ui = 1;
printf("%d\n", si < ui);
```

Compilation

Vérification statique

- Certainement le plus grosse plus value des compilateurs
- Arrivé des langages haut niveau dit fortement typé

```
float a = 0.0f;
int b = a + 2;
```

- Faire attention au phénomène de integer promotion et conversion explicite

```
int si = -1;
unsigned int ui = 1;
printf("%d\n", si < ui); // ui est gérée comme
un int et mit à INT_MAX
```

Compilation

Vérification statique

- Certainement le plus grosse plus value des compilateurs
- Arrivé des langages haut niveau dit fortement typé

```
float a = 0.0f;
int b = a + 2;
```

- Faire attention au phénomène de integer promotion et conversion explicite

```
int si = -1;
unsigned int ui = 1;
printf("%d\n", si < (int)ui);
```

Compilation

Vérification statique

- Bien comprendre ce qu’il se passe au niveau compilateur
- Connaîtres les options principales
- Suivre leurs évolution

Compilation

Vérification statique

Exemple de C++

- C++11 nouvelle norme
- Nouveau mot-clés du langage
  - static_assert assertion à la compilation
  - constexpr permet de réaliser des opérations simple (en fait de plus en plus complexe avec C++ 14, C++ 17...) à la compilation

Compilation

Vérification dynamique

Les canaries de pile

- Permet de se protéger des Buffer overflow sur la pile
- Ajoute une valeure aléatoire sur la pile au tout début de la fonction
- Avant de réaliser l’épilogue de la fonction on vérifie si cette valeure n’a pas été modifié lors de l'exécution de cette dernière

Compilation

Vérification dynamique

Les canaries de pile

- On exécute une instruction de comparaison à chaque appel de fonction
- Attention au performances...

Compilation

Vérification dynamique

Les canaries de pile

- Visual studio l'active par défaut, pour la désactiver il faut le préciser via l'options /GS-
- GCC lui ne l'active pas par défaut
  - --fstack-protector-all appliqué à toutes les fonctions
  - --fstack-protector applique aux fonctions possédant un tableau de char > 8
  - --fstack-protector-strong applique à toutes les fonction possédant un tableau sur la pile

Compilation

Vérification dynamique

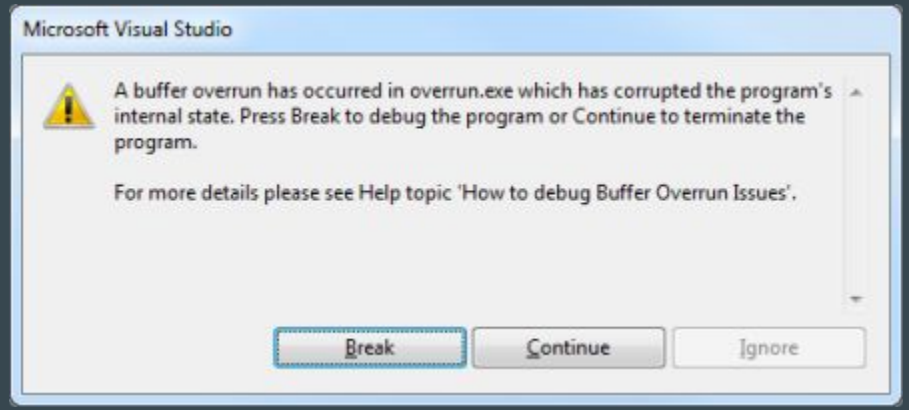
Les canaries de pile

```
void main()
{
    char buf[16];
    strcpy(buf, "This string is too long!"); //
Buffer overrun
    puts(buf);
}
```

Compilation

Vérification dynamique

Les canaries de pile



Système d’exploitation

... et les fabricants de processeurs

- Évolution importante depuis ces 20 dernières années
- Pierre angulaire de la sécurité informatique
- Complexe...
- ... mais passionnant

Système d’exploitation

... et les fabricants de processeurs

Deux évolution majeure concernant la sécurité informatique

- Modèle d'exécution en mémoire d’un processus
- Address Space Layout Randomization

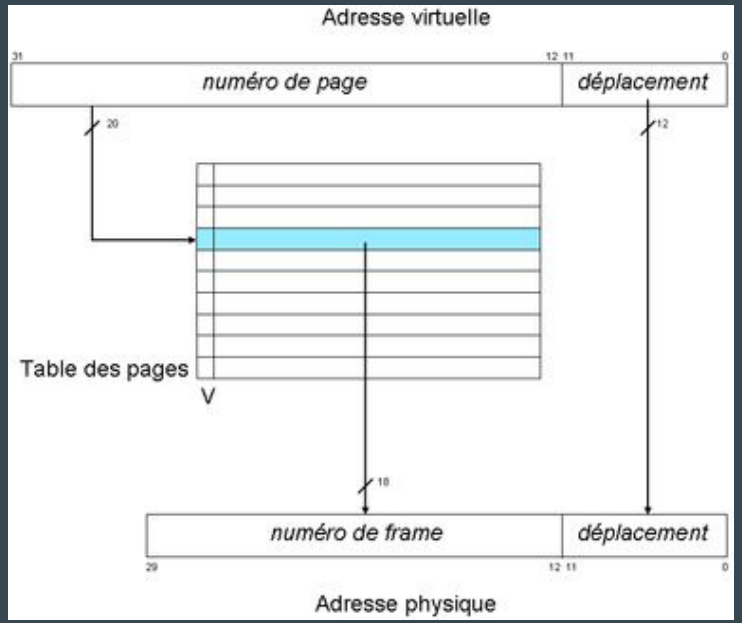
Système d’exploitation

Userland

- Tous les programmes sont exécuté en espace utilisateur
- Ring 3 -> notion plutôt orienté hardware
- Segmentation et pagination (segmentation fault)
- Adressage virtuel
- Cloisonnement des processus
- Un processus ne peut pas polluer l’espace d’adressage d’un autre processus...
- ... sauf si il existe un problème dans le kernel

Système d’exploitation

Modèle d'exécution en mémoire d’un processus



Système d’exploitation

Modèle d'exécution en mémoire d’un processus

- Gestion des droit pour chaque page physique
- La notion de page existe aussi au niveau processeur
- DPE pour Data Protection Execution
- Gestion de droit par page
  - Read
  - Write
  - Execute

Système d’exploitation

Modèle d'exécution en mémoire d’un processus

- Segment de pile non exécutable (RW)
- Segment de code exécutable (RE)
- Segment de tas (RW)
- Les constantes (comme les chaîne de caractère)

Permet de se prémunir des injection de code malveillant (shellcode)

Système d’exploitation

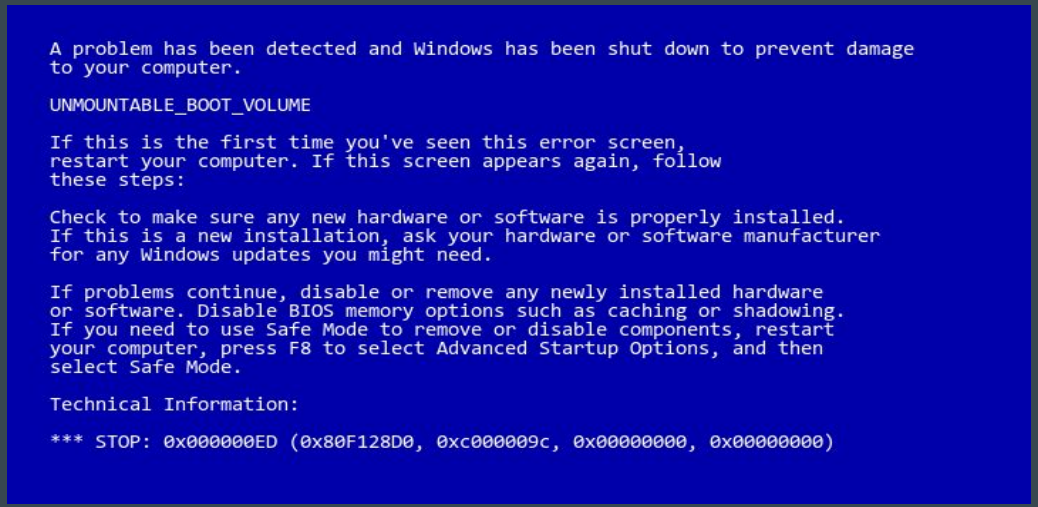
Kerneland

- Programmation driver
- Ici tout est permis
- plus de sécurité
- La moindre erreur est fatale

Système d’exploitation

Kerneland

- Blue Screen Of Death
- BSODER (crasher pour les programmeurs driver)



Système d’exploitation

Address Space Layout Randomization

- Les shellcodes était la technique des année 2000
- Beaucoup plus difficile
- Utiliser le code contenue dans l’application vulnérable
  - Modification du flux
  - Return Oriented Programming
- Techniques reposant sur l’aspect statique de l’adressage du processus

Système d’exploitation

Address Space Layout Randomization

- Une des principales sécurité offerte par les systèmes durant ces 10 dernières années
- Charge les librairies à des adresses aléatoire
- Voire le segment de code principale
- Chaque exécution du programme n’aura pas le même modèle d’adressage mémoire

Système d’exploitation

Address Space Layout Randomization

- Windows l'implémente en partie mais à longtemps introduit un biais
  - Librairie windows toujours chargé à la même adresse dans chaque processus
- Linux utilise un mécanisme lors de la compilation pour la partie librairie
  - options de compilation -fPIC (Position Independant Code) pour une librairie
  - options de compilation -fPIE (Position Independant Executable) pour un programme principal