IS 3 - T.P. Système
Gestion de la mémoire

Sources : Tanguy Risset, Antoine Fraboulet, Cédric Bastoul

Les fichiers mentionnés en Section 2 et 3 sont disponibles dans `~jforget/Cours/Syst/TP/Memoire`. Recopiez les chez vous.

1 Structure mémoire d'un programme C

1.1 Préliminaire

On commence par réaliser un programme allouant différents types de variables. Le type et la valeur de ces variables importe peu, l'objectif est d'observer leurs adresses.

1. Ecrivez un programme C contenant :
   — Une variable globale `globVar` (déclarée au début du fichier, à l'extérieur de toute fonction) ;
   — Une variable `locVar` locale à la fonction `main` ;
   — Un tableau alloué dynamiquement `myTab`.
2. Ajoutez les instructions suivantes :
   — Affichez le PID du processus à l'aide de la fonction `getpid`. Une valeur de type `pid_t` peut s'afficher à l'aide d'un `%d`, comme pour un entier ;
   — Pour chacune des variables ci-dessus, affichez leur adresse. On pourra utiliser le format `%p` de `printf` pour afficher une adresse dans un format lisible (mais `%p` ne fait pas tout seul l'extraction d'adresse) ;
   — Affichez également les adresses des fonctions `main` et `malloc`. Vous pouvez afficher l'adresse d'une fonction via `printf`. Par exemple, `printf(''%p'',f)` affiche l'adresse de la fonction `f` ;
   — Afin d'éviter que le programme ne s'arrête immédiatement, insérez un point d'arrêt juste derrière l'affichage. Par exemple, demandez à lire un caractère (le programme reste bloqué tant que vous ne saisissez rien).

1.2 Observation des adresses

1. A l'aide de la commande `readelf -s` appliquée à votre exécutable, recherchez les adresses des *symboles* suivants. Expliquez pourquoi certaines adresses ne sont pas disponibles [1]. Si besoin, utilisez `grep` pour faire du tri. Recherchez :
   — Les variables de l'énoncé ;
   — La fonction `main` ;
   — La fonction `malloc`.
2. Exécutez votre programme. Lorsqu'il est en pause, affichez le contenu du fichier `maps` associé au processus (cf le répertoire `/proc/mon_pid` vu dans le TP sur les processus). A l'aide de ce contenu et des informations affichées par le `printf` de votre programme, déterminez dans quelles zones de la mémoire virtuelle (décrites dans le fichier `maps`) sont positionnées chacune des variables ou fonctions dont l'adresse est affichée par votre programme. Vous devriez retrouver les sections mémoires mentionnées en cours. Observez également les différences de droits d'accès entre ces sections et expliquez.

1. Notez que `readelf` se base uniquement sur les informations produites par le compilateur, il n'exécute pas le programme.

2 Overflow

1. Etudiez le code de la fonction `main` du programme `buf.c` (la fonction `rot13` est une fonction simpliste de chiffrage, qu'il n'est pas besoin d'analyser ici) ;
2. Compilez de la manière suivante (ignorez les warnings pour l'instant) :
   `gcc -g buf.c -o buf -fno-stack-protector`
3. Testez plusieurs cas d'exécution :
   (a) Entrez le bon mot de passe ;
   (b) Entrez `iloveos` ;
   (c) Entrez `jaimebeaucouplesos` ;

   Vous accédez au secret aussi dans le dernier cas !
4. Lancez le débuggage de votre programme (`gdb ./buf`). Dans `gdb`, ajoutez un breakpoint juste devant la ligne `if(myToken.pass)` (`break XXX`, où XXX est le numéro de ligne). Lancez l'exécution (`run`) et affichez les variables locales (`info locals`). Testez avec : le mot de passe correct, le mot de passe court incorrect, le mot de passe long incorrect. Expliquez pourquoi vous pouvez accéder au secret avec un mot de passe incorrect (pensez à la manière dont les champs d'un `struct` sont stockés en mémoire) ;
5. Testez avec un très long mot de passe (au moins deux fois le mot de passe long précédent). Expliquez pourquoi vous obtenez une erreur ;
6. Modifiez la manière dont les variables/types sont déclarés afin d'éviter d'afficher le secret quand le mot de passe est incorrect ;
7. Eliminez le problème d'overflow en utilisant `fgets` à la place de `gets`.

3 Consommation mémoire (Bonus)

1. Lisez le programme `vmem1.c` et expliquez ce qu'il fait ;
2. Compilez en ajoutant l'option `-lm`, puis exécutez ;
3. Lancez un moniteur système pour surveiller l'exécution de ce programme (par exemple : `gnome-system-monitor`, puis onglet Ressources) ;
4. Sans fermer le moniteur système, lancez une nouvelle exécution de ce programme en parallèle avec la première. Observez la consommation mémoire et lancez des exécutions supplémentaires, jusqu'à saturer la mémoire. **Attention** à ne pas abuser, au risque de rendre votre machine inopérante. Expliquez le ralentissement observé lorsque la mémoire sature : observez l'évolution des différentes courbes et surtout leurs corrélations.