Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
PREMIER MINISTRE

S . G . D . S . N
Agence nationale de la sécurité des systèmes d'information
CERT-FR

Paris, le 12 décembre 2016
N° CERTFR-2016-ACT-050

Affaire suivie par :
CERT-FR

BULLETIN D'ACTUALITÉ

Objet : Bulletin d'actualité CERTFR-2016-ACT-050

1 - Durcissement logiciel sur système Linux

Introduction

Dans ce bulletin, nous allons tout d'abord aborder les méthodes de durcissement à la compilation des logiciels écrits en langage C/C++ dans un environnement Linux. Ensuite, nous décrirons quelques outils qui permettent de tester et d'analyser du code de façon statique et dynamique.

Nous assumons l'utilisation du compilateur *GCC* dans le reste de ce document.

La chaîne de compilation

La chaîne de compilation classique d'un projet en C est composée de trois étapes.

1. Le compilateur va compiler un fichier source en fichier objet. Lors de cette étape, le code C est transformé en code machine, mais n'est pas exécutable. Chacun des fichiers source est compilé dans un fichier objet différent.
2. L'éditeur de liens va générer le fichier exécutable final en combinant tous les fichiers objet.
3. Finalement, une étape optionnelle de post-traitement permet de modifier certains attributs du fichier exécutable avant de le déployer ou de le partager.

Étape de la compilation

Plusieurs actions importantes du durcissement ont lieu lors de l'étape de la compilation.

***Address Space Layout Randomization* (ASLR), première partie** La *randomization* de l'espace d'adressage (en anglais *ASLR*) est une technique qui permet de changer l'emplacement des ressources d'un programme lors de son chargement en mémoire.

Cela a pour effet de rendre plus difficile l'exploitation de certaines vulnérabilités. En effet, l'exploitation de vulnérabilités d'un logiciel est facilitée si celui-ci place toutes ses ressources à des endroits fixes, connus et documentés.

Pour fonctionner, l'ASLR nécessite un support à la fois du système d'exploitation et du logiciel chargé en mémoire. Pour pouvoir profiter de l'ASLR, votre programme doit être compilé avec l'option de compilation `-fPIC`. Les fichiers objets ainsi compilés sont dits relogeables, c'est-à-dire qu'ils peuvent être chargés n'importe où en mémoire.

Durcissement de la glibc La *glibc* est une des bibliothèques de base les plus utilisées. Ainsi, elle propose certaines options de durcissement.

Pour pouvoir en profiter, il faut compiler avec l'option `-D_FORTIFY_SOURCE=2`. Cette option va remplacer l'implémentation d'un ensemble de fonctions considérées comme attaquables et rajouter des vérifications supplémentaires.

Protection de la pile La pile est une des zones mémoire exposées aux attaques. Régulièrement mise en cause dans les exploitations de vulnérabilités en raison des dépassements de tampons, il est nécessaire d'agir pour la protéger.

La protection consiste au rajout d'une valeur spéciale appelée un "canari" dans la pile. Si la valeur "canari" a été modifiée entre le début et la fin de la fonction, un dépassement de tampon a eu lieu.

L'option de compilation `-fstack-protector` commande au compilateur de générer le "canari" dans les fonctions qu'il considère comme étant potentiellement vulnérables. Il existe aussi -fstack-protector-all et -fstack-protector-strong. L'option `-fstack-protector-strong` protège plus de fonctions que `-fstack-protector`, et `-fstack-protector-all` protège toutes les fonctions.

Il est conseillé d'utiliser l'option `-fstack-protector-strong`, car elle représente un bon compromis entre la protection et le surcoût en taille/performance engendrée par la protection.

Étape de l'édition des liens

L'édition des liens est l'étape finale de génération d'un programme exécutable. Ici encore, il convient de faire les bons choix pour assurer un maximum de sécurité.

Randomisation de l'espace d'adressage (ASLR), seconde partie Pour disposer d'un exécutable correctement protégé et compatible avec ASLR, il faut passer l'option `-pie` lors de l'édition des liens.

Il est primordial de ne pas oublier cette étape, sans quoi une quantité importante des ressources du logiciel compilé ne seront pas considérées relogeables, et ne profiteront pas de ASLR.

Protection des ressources relogeables Les ressources relogeables sont placées aléatoirement en mémoire lors du chargement du programme. Ces chargements ont lieu à plusieurs reprises tout au long de l'exécution d'un programme lorsque celui-ci charge une bibliothèque dynamique. En conséquence, les zones mémoire où sont positionnées les ressources ne peuvent être en lecture seule, comme c'est le cas pour un programme non relogeable.

Pour durcir cela il existe deux options à passer à l'éditeur de liens, `-Wl,-z,relro` et `-Wl,-z,now`. Ces deux options forcent le chargement de toutes les bibliothèques dynamiques au démarrage du logiciel et passe en lecture seule une majeure partie des zones mémoire ainsi chargées. Cette protection augmente le temps de chargement du logiciel.

Protections supplémentaires

Dans le cas de logiciels sensibles qui doivent être distribués à des partenaires ou acteurs tiers, il est envisageable de cacher un maximum d'information pour rendre les études ou analyses plus compliquées en cas de fuite ou de perte du logiciel.

Dénuder un exécutable Dénuder un exécutable consiste à retirer un maximum d'information inutile à l'exécution, telle que des informations de débogage ou des informations de versions. Ce n'est pas une protection en soi, car elle ne rend pas l'exploitation d'une vulnérabilité plus difficile, mais elle en complique l'étude.

L'outil en ligne de commande `strip` ou l'option de compilation à passer à l'éditeur de liens `-Wl,-strip-all` permettent de dénuder un logiciel.

Obscurcir un exécutable Obscurcir un exécutable consiste à modifier le code machine d'un programme pour le rendre incompréhensible par un être humain ou par une analyse, mais sans en modifier le comportement. Une fois de plus, ce n'est pas une protection en soi car toute vulnérabilité reste présente, mais cela en complique encore plus l'analyse par un tiers.

Étape de réalisation des tests

Dans tous les cas, durcir un exécutable est une première étape, mais il ne faut pas négliger les outils d'analyse statique et dynamique pour trouver les bogues.

Analyse statique Il existe de nombreux outils d'analyse statique de code. Leurs façons de procéder diffèrent, mais leur but reste le même : lire le code source d'une application pour en trouver les bogues. Il n'existe pas de méthode fiable à 100 % capable de garantir que tous les bogues ont été détectés, c'est pour cela qu'il existe autant de logiciels d'analyse statique différents et que leurs résultats peuvent varier.

Analyse dynamique Les outils d'analyse dynamique permettent d'exécuter des logiciels dans un environnement instrumenté dans le but de détecter des comportements anormaux directement lors de l'exécution.

- **Valgrind**
  Valgrind est sûrement le cadriciel d'analyse dynamique le plus connu. Les outils de *Valgrind* permettent de détecter les erreurs et les fuites mémoire, les accès concurrentiels, les interbloquages et bien d'autres. *Valgrind* simule sur un processeur synthétique les instructions en code machine une par une. Les outils de *Valgrind* les plus coûteux peuvent donc engendrer des temps d'exécution dix à cinquante fois plus élevés que la normale.
- **Sanitizers**
  Il existe plusieurs outils d'analyse dynamique. Parmi ceux-ci, l'un des plus en vogue est le projet *Sanitizer*. Le but des *sanitizers* est d'instrumenter un logiciel grâce à du code généré lors de la compilation. Le projet *Sanitizer* est découpé en plusieurs parties :
  - *AddressSanitizer* détecte les erreurs d'utilisation de la mémoire allouée par un programme, telles que l'accès à de la mémoire déjà libérée ou l'accès via un pointeur nul ;
  - *LeakSanitizer* liste la mémoire allouée et non libérée à la fin de l'exécution d'un programme ;
  - *ThreadSanitizer* détecte les interbloquages et situations de compétitions ;
  - *MemorySanitizer* détecte les lectures de mémoire allouées, mais non initialisées.

  Les Sanitizers ajoutent eux aussi une pénalité en temps d'exécution et en taille en mémoire, généralement moindre que *Valgrind*, mais tout de même non négligeable.
- **Fuzzer**
  Les programmes dits *Fuzzer* vont fournir à votre logiciel des entrées générées automatiquement afin de faire exécuter un maximum de code et d'explorer les états d'un logiciel le plus précisément possible. Il en existe de tout genre et de toute forme, en source ouverte ou non.
  La génération d'une nouvelle entrée et l'exécution du logiciel *fuzzé* sont souvent consommatrice de temps. On ne peut donc pas utiliser un *fuzzer* pendant cinq minutes et conclure qu'il n'y a pas de bogues. Un *fuzzing* efficace doit être fait en continu pour lui laisser un maximum de temps pour analyser les résultats et générer de nouvelles entrées.
  Il existe des *fuzzers* en source ouverte tels que *zzuff*, *LibFuzzer* ou *AFL (Americain FuzzyLop)*.

Conclusion

Dans ce bulletin, nous avons présenté succinctement les différentes étapes du durcissement d'un logiciel sous Linux. Comme toujours, il faut garder à l'esprit que les problématiques de sécurité doivent être prises en compte dans toutes les étapes du développement, de la conception au test, en passant par la compilation.

2 - Rappel des avis émis

Dans la période du 05 au 11 décembre 2016, le CERT-FR a émis les publications suivantes :

- CERTFR-2016-AVI-399 : Vulnérabilité dans Nagios
- CERTFR-2016-AVI-400 : Vulnérabilité dans Cisco IOS et IOS XE
- CERTFR-2016-AVI-401 : Multiples vulnérabilités dans le noyau Linux de Suse
- CERTFR-2016-AVI-402 : Multiples vulnérabilités dans les produits BlueCoat
- CERTFR-2016-AVI-403 : Multiples vulnérabilités dans Asterisk
- CERTFR-2016-AVI-404 : Multiples vulnérabilités dans le noyau Linux de Suse

– CERTFR-2016-AVI-405 : Multiples vulnérabilités dans SCADA Siemens SIMATIC

Durant la même période, les publications suivantes ont été mises à jour :
– CERTFR-2016-AVI-395 : Multiples vulnérabilités dans le noyau Linux d'Ubuntu (version initiale.)

Gestion détaillée du document

12 décembre 2016 version initiale.