Défense Linux – 32bits

Ce laboratoire fonctionne dans un environnement remote, des instructions / directives vous ont étés fournies pour accéder à l'infrastructure distante.

Introduction

Binaire : *chall4*

Ce binaire est largement repris de l'exercice *buffer2* vu en cours. Nous allons l'utiliser comme décrit dans le tutoriel https://www.0x0ff.info/2021/advanced- buffer-overflow-bypass-aslr-32-bits/ (dont le langage « fleuri » n'engage que son auteur). L'objectif consiste à implémenter la démarche décrite dans ce lien de manière à vérifier la faible entropie de la randomisation en 32 bits.

/ !\ Pour que les applications graphiques fonctionnent correctement, merci de « sourcer » le script clean.sh dans votre home directory une fois connecté sur la machine finale / !

source ./clean.sh

Laboratoire

Manipulation 1.1

Analysez *chall4* avec Ghidra et adaptez l'exploit présenté en cours pour *buffer2* sans l'ASLR. Utilisez le shellcode fourni dans *bash_shellcode32.c.*

Question 1.1 (1pts)

Quelle différence principale avez-vous notée entre les binaires *chall4* et *buffer2* ?

Question 1.2 (1pts)

Expliquez la modification que vous avez faite sur le payload prévu pour *buffer2* afin d'exploiter *chall4.*

Manipulation 1.2

A l'aide des informations fournies par le tutoriel, concevez un script capable de calculer « l'adresse **médiane** » du buffer vulnérable de *chall4.*

Question 2.1 (2pts)

Quelle valeur médiane avez-vous trouvé ? Justifiez rapidement cette valeur au vu de l'échantillon collecté.

Question 2.2 (2pts)

Présentez le/s script/s réalisé/s. Numérotez les lignes de votre code et indiquez en légende ce que font les lignes essentielles du script.

Manipulation 2.2

Écrivez maintenant un script qui exploite en boucle *chall4* en espérant que le buffer vulnérable finira bien par apparaître à une adresse proche de la médiane précédemment calculée, cette fois avec l'ASLR.

Aide : Votre payload doit minimiser les risques de tomber à une adresse trop éloignée.
Soyez un peu patient…

Question 2.3 (2pts)

Votre attaque a-t-elle finalement réussie ? Combien de fois et pendant combien de temps avez-vous dû envoyer votre payload.

Question 2.4 (2pts)

Comment avez-vous optimisé votre payload pour cette attaque ? Estimez empiriquement l'ordre de grandeur de réduction de l'espace à bruteforcer grâce à cette optimistation.

Manipulation 3.1

Utilisez maintenant l'astuce mentionnée (sans explication) dans le tutoriel pour mettre votre payload ailleurs en mémoire, là où vous serez plus libre de l'optimiser pour le bruteforce. Adaptez votre script qui exploite en boucle *chall4* afin de vérifier l'efficacité de l'astuce.

Aide : Les variables d'environnement sont adressables ainsi dans GDB :
(gdb) x/100s *((char **)environ)

Question 3.1 (2pts)

Votre attaque a-t-elle finalement réussie ? Combien de fois et pendant combien de temps avez-vous dû envoyer votre payload.

Question 3.2 (2pts)

Comment avez-vous optimisé votre payload pour cette attaque ? Estimez empiriquement l'ordre de grandeur de réduction de l'espace à bruteforcer.

Question 3.3 (2pts)

En conclusion de ce qui précède, dans quels cas jugez-vous la protection apportée par l'ASLR en 32 bits insuffisante et pourquoi ?