INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

INSA de Lyon
Département Télécommunications,
Services et Usages

TP Buffer Overflow

François Lesueur

Ce TP sera réalisé dans l'environnement virtuel "TP-SEC-Infra". Pour le démarrer :

```
/machines_virtuelles/TP-SEC-Infra-Firewall/nlaunch.sh
/machines_virtuelles/TP-SEC-Infra-Hacker/nlaunch.sh
/machines_virtuelles/TP-SEC-Infra-Filer/nlaunch.sh
```

Avant de démarrer, connexion root sur Firewall (login/mdp : root/root) et faire :
`iptables -F FORWARD ; iptables -P FORWARD ACCEPT`

Une fois l'environnement démarré, la seule machine à utiliser est celle du hacker (login/mdp : root/root). Une fois connecté en console, vous pouvez si vous le souhaitez lancer une session graphique X11 avec la commande `startx`. Un clic droit sur le bureau vous donne accès aux applications installées.

1 Machine cible

La machine cible *Filer* héberge 2 versions différentes du service à exploiter :
— une version verbeuse (debug) sur le port 1234 ;
— une version normale (prod) sur le port 12345.
Ces 2 services sont strictement similaires, à la sortie près. La version prod est légèrement plus difficile et nécessite un peu plus l'utilisation de `gdb`. À vous de choisir !

1.1 Fonctionnement du service

Le service est un programme ligne de commande qui écoute sur une socket. Plus exactement, il est encapsulé dans un netcat, les adresses mémoire sont donc strictement identiques à la version que vous pourrez lancer de votre côté, en ligne de commande avec lecture sur *stdin*. Pour vous y connecter depuis la machine du hacker : `nc 192.168.0.3 <Port>`.

La machine serveur est configurée par défaut, le bit NX et l'ASLR sont activés. Le programme est compilé avec les options de `gcc` par défaut (pas de PIE, pas de canaris ici).

2 Exploitation du service par *buffer overflow*

Le service visé possède une succession de vulnérabilités que vous allez pouvoir exploiter pour :
— faire planter le programme ;
— valider le mot de passe dans la fonction `prepare()` ;
— faire retourner `launch()` vers la fonction `armageddon()` lorsque le mot de passe est invalide ;
— exécuter du code arbitraire sur le serveur pour par exemple récupérer des fichiers dans `/root`.

À chaque étape, testez l'exploit en local puis envoyez le vers le serveur. En local : `./service < payload` ; à distance : `nc IP Port < payload`. Il ne reste plus qu'à générer les payloads...

2.1 Analyse de l'exécutable

Sur la machine du hacker, vous disposez de service.c et service12345.c. Compilez avec l'option -g pour pouvoir utiliser gdb dessus. À l'aide de gdb, des affichages de debug ou de quelques lignes de codes ajoutées (attention, ajouter du code peut modifier les adresses), faîtes un petit plan de la pile à l'entrée de la fonction `prepare()` (buffer, mdp, @retour).

L'exploitation consiste à déborder du buffer alloué. Il est prévu pour un code PIN de 4 caractères mais aucune vérification n'est faite par `scanf` : il est ainsi possible d'écrire une chaîne arbitrairement longue.

2.2 Faire planter le programme

À la fin de la fonction `prepare()`, si l'authentification échoue, la fonction retourne à @retour. En exploitant buffer via une payload adéquate, faîtes planter le service localement puis sur le serveur.

> Un simple plantage peut-il avoir des conséquences ? Ou suffit-il de redémarrer automatiquement le service ?

2.3 Valider le premier mot de passe

À l'aide de votre plan de la pile, proposez une payload permettant de valider un mot de passe quelconque. Utilisez là localement puis à distance.

2.4 Faire retourner vers armageddon

L'astuce utilisée précédemment ne fonctionne pas dans la fonction `launch()`. Pourquoi ?

Trouvez une payload permettant de faire retourner `launch` vers `armageddon`, au lieu de son adresse de retour originale.

2.5 Exécution de code arbitraire

Pour aller encore plus loin, nous allons exécuter un code arbitraire, par exemple un shell (le service tourne en root, ce qui est une mauvaise idée en cas d'attaque...). Une possibilité est de combiner astucieusement les deux fonctions de la libc `read()` et `system()`. Avec le schéma proposé,

cette payload doit être exécutée sur le service avec les affichages de debug (pour l'autre version, le niveau de complexité augmente mais reste réalisable).

Quelques rappels (ou pas) :

— Avec le bit NX, il n'est pas possible d'exécuter du code sur la pile (ancienne approche du BO). Nous allons faire du *ret2libc* (retour vers la libc) pour exécuter du code de la libc avec des paramètres que nous allons manipuler.
— À l'entrée dans une fonction, schéma de la pile : @$ret|param1|param2|$
— `man 2 read` et `man 3 system` pour avoir les prototypes de ces fonctions
— Il est possible d'écrire à l'adresse 0xbf848800, ainsi que dans le BSS (objdump -h ./service ou readelf -a ./service)

3 Aide-mémoire

Les registres qui vont le plus nous intéresser sont esp et ebp. esp pointe vers le sommet de pile, ebp vers le début de la frame. L'adresse de retour est stockée à l'adresse ebp+4.

Utilisation de gdb :

— Mettre un breakpoint sur la fonction `prepare()` : b prepare
— Exécuter avec un paramètre passfile et une payload au lieu de stdin : run passfile < payload
— Afficher une plage mémoire : x/10x 0x1234ABCD, x/10w 0x1234ABCD, x/10x $esp, x/10x (buffer+5)
— Afficher une valeur : print nom-de-la-variable
— Aller à l'instruction suivante : next
— Rentrer dans un appel vers une fonction : step

Création d'une payload en hexadécimal avec bash : `printf '\x01\x23\x45\x67' >> payload` (exemple sur la machine du hacker).

L'architecture x86 est *little-endian*, les entiers stockés sur 4 octets (typiquement, des adresses mémoire) sont inversés. Par exemple, la valeur 0x1234ABCD est obtenue par la succession des octets 0xCD, 0xAB, 0x34, 0x12.