TP Sécurité Unix
Kali Linux 2022
Exploitation de *buffer overflow*

Mathieu Blanc

Avant-propos

Au travers de ce TP, vous allez étudier certains aspects de la sécurité d'un système Linux :
- Mécanismes de sécurité du système ;
- Failles de sécurité applicatives ;
- Outil d'exploitation de vulnérabilités.

Les cours et le sujet de TP sont accessibles sur `http://moutane.net/enseirb/se/`.

Le TP est principalement écrit pour fonctionner dans l'environnement Kali Linux 2022, mais en principe il peut être effectué sur n'importe quel système Linux, de préférence en machine virtuelle.

Conseils

- Utilisez `man` ;
- Pensez à sauvegarder votre travail sur une clé USB, c'est un Live DVD ;
- Google est votre ami aussi (pour la documentation, pas pour la vie) ;
- Pour avoir Kali Linux en clavier français, lancez le système avec l'option `keyboard-layouts=fr`. Pressez TAB au moment du menu de démarrage de Kali, et ensuite ajoutez l'option, en n'oubliant pas que le clavier est en QWERTY (a sur q, '-' sur ')'). Si vous avez raté le moment du démarrage, vous pouvez taper la commande `setxkbmap fr` dans un terminal.

Evaluation

Un compte-rendu vous est demandé pour les 2 séances de TP sur la sécurité Linux.
- En particulier, répondez à ce qui est demandé en italique dans le sujet ;
- Vous avez **une semaine** après la fin du second TP pour terminer et restituer ce compte-rendu par **e-mail** à `moutane@moutane.net`.

Le compte-rendu devra contenir les différentes commandes utilisées dans le TP, ainsi que les éventuels résultats demandés ; vous pouvez également mettre toute précision que vous jugez intéressante.
Le sujet de TP concerne la distribution Kali, mais vous pouvez l'appliquer à un autre système de votre choix.

Première partie
Analyse du programme vulnérable

Le sujet et les fichiers du TP seront disponibles sur mon laptop. Dès qu'il est connecté au réseau, vous pourrez récupérer le sujet sur l'adresse `http://srv/`. Si ça ne fonctionne pas, vous devez peut-être reconfigurer le réseau avec l'icône accessible en haut à droite.

Les fichiers utiles pour le déroulement du TP se trouvent également à l'adresse `http://moutane.net/enseirb/se/ressources_tp/`.

Récupérez le code source vulnérable `vuln1.c` et le sujet de TP. Nous allons maintenant étudier la raison pour laquelle il contient une vulnérabilité.

1 Regardons le code source

Ouvrez le fichier `vuln1.c`. Pour découvrir quelle est l'origine de la vulnérabilité, commencer par observer quelles sont les variables locales du programme, en notant leurs tailles. Puis cherchez quelles fonctions vont écrire des données dans ces variables locales.

Expliquez quelle est la fonction qui provoque un buffer overflow et pourquoi.

2 Compilation du programme sans protection contre les buffer overflow

La distribution Kali Linux 2022 Live ne permet pas de compiler le programme avec les options qui nous intéressent. Vous pouvez récupérer le binaire `vuln1` précompilé au même endroit que le sujet de TP dans le dossier `step1`.

Pour information, ou si vous disposez d'un autre système Linux que Kali Live, le programme est compilé avec la commande suivante :

```
# gcc -m32 -z execstack -fno-stack-protector -mpreferred-stack-boundary=2 -o vuln1 vuln1.c
```

Cette commande de compilation désactive plusieurs protections :

- `-z execstack` : le programme compilé aura une stack exécutable ;
- `-fno-stack-protector` : désactive la fonctionnalité de canary.

Normalement vous êtes sur un système Linux 64 bits, l'option `-m32` est ajoutée pour compiler le programme en architecture 32 bits. Le TP n'est pas faisable tel quel sur une architecture 64.

Important : lancez la commande suivante en root pour désactiver globalement la fonctionnalité d'ASLR :

```
# echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

Deuxième partie
Ecriture d'un code d'exploitation

3 Crash du programme

Suivant votre analyse du code source programme, essayez de provoquer une erreur de type *segmentation fault* dans le programme `vuln1` en effectuant un buffer overflow.

Vous pouvez utiliser la commande `nc` pour envoyer des données au programme par le réseau.

Vous pouvez aussi vous inspirer du script Python suivant (voir aussi les scripts Pythonà disposition sur mon serveur) :

```python
#!/bin/python

import socket

TCP_IP = '127.0.0.1'
TCP_PORT = 11222

MESSAGE = 'A'*100

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE)
print "Buffer sent."
print ':'.join(x.encode('hex') for x in MESSAGE)
s.close()
```

A l'aide de la commande `dmesg`, observez les détails concernant le crash du programme, normalement une erreur de type *segfault*. En particulier, les valeurs qui nous intéressent sont les contenus des registres `ip` et `sp`.

Quelle est la taille minimale de données à envoyer pour provoquer un plantage du programme ? Pouvez-vous établir un lien entre les données que vous avec injectées et la valeur de `ip` *?*

4 Optionnel : obtention et analyse d'un core dump

Le système Kali Linux 2022 Live ne contient pas a priori de commande pour analyser un *core dump*, cette partie du TP ne peut être réalisée que sur un autre système.

Configurez bash pour qu'il crée un *core dump* avec la commande `ulimit -c` (*man bash*).

Avec le programme `gdb`, analysez le fichier core. (Indice : `man gdb`) En particulier la commande intéressante de gdb est `info register` ou `i r` en version abrégée.

Ce qui nous intéresse ici, ce sont les valeurs des registres `%eip` et `%esp` au moment du crash.

Une fois que vous maîtrisez le fait d'utiliser `gdb` pour visualiser les informations nécessaires, vous pouvez les retrouver plus rapidement en visualisant les logs du noyau Linux à l'aide de la commande `dmesg` et en repérant les messages correspondant aux crash du programme vulnérable (en particulier les valeurs de `ip` et `sp`).

5 Analyse du crash du programme

Trouver la valeur du registre `sp` au moment du crash du programme.

Ecrivez un script qui va injecter dans le programme un tableau d'octets adéquat pour que le flot d'exécution soit détourné sur la pile. (Indice : concaténer une chaîne de caractères qui va remplir le buffer + quelques octets + l'adresse recupérée dans le registre sp).

Concrètement, l'objectif est que le programme charge dans le registre `ip` la valeur du registre `sp` au moment du crash. Dans le message du crash, vous devriez observer des valeurs similaires pour les 2 registres.

Vous pouvez étendre le script donné précédemment (concaténation de chaîne de caractère avec + en Python), et vous inspirer des scripts Python disponibles sur mon serveur.

Expliquez le format des données que vous envoyez pour faire en sorte que le programme tente d'exécuter le code à l'adresse sp.

6 Technique automatique pour déterminer l'espace avant l'adresse de retour

Le projet Metasploit contient de nombreux outils pour faciliter l'écriture d'un code d'exploitation. En particulier, deux outils permettent de déterminer l'espace présent entre le début du *buffer* et l'adresse qu'on peut manipuler pour la charger dans le registre `ip` : `msf-pattern_create` sert à créer une chaine de caractère de mesure, et `msf-pattern_offset` sert à déterminer la position dans la chaine précédemment créée avec 4 octets de données récupérés dans le registre `ip` au moment du crash.

Avec le premier programme, créez une chaîne de caractères suffisamment longue à communiquer au programme vulnérable. Ensuite récupérez l'adresse présente dans le registre `ip` au moment du crash, et passez cette adresse en argument au second programme.

Rappels : vous pouvez utiliser le programme `nc` pour envoyer la chaîne de caractères au programme vulnérable, et `dmesg` pour récupérer la valeur de `ip` au moment du crash.

Expliquez comment utiliser ces outils pour déterminer le format des données à envoyer au programme vulnérable.

7 Ajout d'un *payload*

Un *payload* est une suite d'instructions visant à changer le comportement du programme où il est injecté. Généralement il a pour objectif de provoquer l'exécution d'un shell.

Utilisez la commandes `msfvenom` pour générer un payload de type `linux/x86/shell_bind_tcp` au format Python.

Concaténez le payload à la chaîne de caractères produite à la question précédente. Envoyez cette chaîne au programme et vérifier la bonne exécution du payload (ouverture d'un port TCP supplémentaire).

Mettez en évidence la réussite de l'exécution du payload à l'aide des commandes `netstat`*,* `nc` *et* `ps`*.*

Troisième partie
Automatisation : ajout d'un module dans Metasploit

Adaptez le module d'exemple que je vous fournis pour le faire fonctionner sur votre machine (étape 1 avec `vuln1_bof_crash.rb` et étape 2 avec `vuln1_bof_generic.rb`).

Pour qu'il soit chargé automatiquement, vous pouvez placer votre module personnalisé dans le dossier `.msf4/modules/exploits` de votre dossier HOME. En général vous devrez créer ce dossier.

Pour tester votre module, vous allez devoir utiliser l'interface console de Metasploit. Elle est accessible avec la commande `msfconsole`, ou dans les menus de Kali. Si vous avez lancé l'interface de Metasploit avant d'avoir copié votre module dans le bon dossier, vous pouvez utiliser la commande `reload_all` pour recharger tous les modules.

Ensuite les commandes les plus courantes sont :

- `use exploit/vuln1...`
- `set PAYLOAD ...`, `set RHOST ...`
- `exploit` ou `run`
- `reload` pour recharger votre module ou `rexploit` pour recharger et relancer la tentative d'exploitation ;
- `ctrl+z` pour mettre en tâche de fond une session Metasploit, et `sessions` pour restaurer une session ;
- et évidemment les commandes `help` et `info`.

En cas de problèmes lors de l'exploitation de la vulnérabilité, positionnez la variable `ENCODER` à la valeur `generic/none`, cela peut résoudre certains problèmes liés à l'encodage du payload.

Vous disposez d'une complétion automatique avec la touche `TAB`.

Expliquez sans trop de détails les étapes principales à réaliser pour créer le module puis effectuer une attaque.

Quatrième partie

Mécanismes de protection

Maintenant que nous avons étudié l'attaque par *buffer overflow*, nous allons mettre en évidence l'effet des mécanismes de protection que nous avons préalablement désactivés.

8 Pile non exécutable et (bonus) attaque `return into libc`

Si vous êtes sur le système Kali Linux Live, utilisez le programme vuln1 dans le dossier step2.

Le programme vuln1 utilisé dans cette partie est obtenu en retirant l'option de compilation `-z execstack`.

Tentez d'exploiter la vulnérabilité dans la nouvelle version du programme, avec un script Python fonctionnel ou avec Metasploit.

Quel est le résultat ? Comment l'expliquez-vous ?

On peut utiliser une autre technique d'exploitation ici, la technique nommée "`return into libc`". Le principe est de passer dans le registre `ip` l'adresse d'une fonction intéressante de la libc comme `system`. Pour trouver l'adresse d'une fonction, vous pouvez charger le programme vuln1 dans gdb, puis après avoir stoppé son exécution, utiliser la commande `info address <symbol>`.

Egalement, il faut placer les arguments de cette fonction dans la pile, en respectant le schéma normal d'une trame de la pile.

Malheureusement, cette technique d'attaque ne peut pas être utilisé avec le programme précompilé pour Kali Linux Live, car il est compilé avec l'option `-static`. Il est nécessaire d'avoir un exécutable qui charge dynamiquement ses bibliothèques pour exploiter cette technique. Vous pouvez tenter cette question si vous avez un autre système Linux.

9 Stack protector

Si vous êtes sur le système Kali Linux Live, utilisez le programme vuln1 dans le dossier step3.

Ici le programme vuln1 est compilé en transformant l'option `-fno-stack-protector` en `-fstack-protector`. Tentez d'exploiter cette nouvelle version du programme.

Expliquez le comportement observé.

10 Bonus : ASLR

Nous allons réactiver la dernière protection désactivée au début du TP, la fonctionnalité d'*Address Space Layout Randomisation*. En utilisant le programme vuln1 obtenu à l'étape précédente, tentez à nouveau d'exploiter la vulnérabilité présente, et mettez en évidence le fait que la pile du programme se situe à une adresse différente à chaque exécution.

Vous pouvez maintenant recompiler vuln1 en reprenant toutes les options du début du TP, et tenter d'exploiter le programme en observant l'adresse où démarre la pile. Pour trouver cette adresse, vous pouvez observer le contenu du fichier `/proc/<PID_du_programme_vuln1>/maps`. Il vous faudra calculer l'écart entre le début de la pile et l'endroit où se situe l'adresse de retour à écraser, puis utiliser cet écart en connaissant la nouvelle adresse de début de pile, pour calculer l'adresse à passer dans votre *shellcode*.