Dans un sous-repertoire nommé **vuln** à la racine du dépôt du projet.

Notions évaluées	Xp à gagner
xxxx	20

Fichier à rendre:

```
.
├── bin
│   ├── my_vm1
│   ├── my_vm2
│   ├── my_vm3
│   ├── my_vm4
│   ├── my_vm5
│   ├── my_vm6
│   ├── my_vm7
│   └── my_vm8
├── code2.insn
├── code.insn
├── gadget.o
├── gadget.s
├── Makefile
├── my_vm.c
├── my_vm.h
└── vuln.c

1 directory, 16 files
```

Sujet:

L’objectif de cette suite d’exercice est de découvrir les techniques basiques puis avancées d’attaque logiciel et notamment le ROP (Return Oriented Programming) dans le cadre d’un système avec pile non exécutable. Vous allez devoir compiler les différents niveaux d’un programme défaillant via le Makefile.

```
# pour compiler le niveau 1
user$ LEVEL=1 make clean all
# pour compiler TOUS les niveaux
user$ make all_levels
```

Pour chaque niveau, un programme my_vmN est généré. Il s’agit d’une mini machine virtuelle capable d’interpréter des instructions basiques. Comprendre ce que fait la machine est nécessaire car il s’agit de trouver ses failles et de les exploiter. La machine compile avec le “SetUserId Bit”, c’est à dire que le programme s’exécutera avec les droits du propriétaire du fichier. Pour mieux saisir, l’impact de l’exploitation d’un tel programme le Makefile changera le propriétaire du programme produit par root, et activera le SetUserId Bit. Pour ce faire le Makefile effectura un sudo, et vous demandera de saisir votre mot de passe et nécessite que vous soyez dans le groupe de sudoers. Nous vous demandons d’utiliser “Pwntools” et de nous rendre des scripts pythons fabriquant les payloads malveillantes nécessaire et lorsque nécessaire automatisant l’attaque afin d’obtenir un shell ROOT sur la machine.

Pour le valider le cours, vous devez atteindre le niveau 4. Les niveaux 5 à 8, vous permettes d'aborder d'autres techniques.

Contexte

La société CorpNet a développé un prototype d'interface Neuro-biologique Turing Complet interprétant dans le cerveau d'une grenouille des instructions d'une machine "Biologique". A cette fin et pour des raisons obscures le programme d'exécution des tests de ces instructions machines s'exécutent avec les droits root via les droits "SetUserId Bit". Normalement, le programme cherche à accéder à un périphérique obscure "/dev/frog". Ce périphérique étant directement connecté au cerveau de la grenouille. Vous avez obtenue le code source d'une version de test du programme finale.

1. Usage:

L'interprète mv_vmX est un programme qui prend en entrée une liste d'instruction de la machine "Frog" qu'il envoie directement dans le cerveau de la grenouille.

La machine virtuel donne accès à une liste d'instruction classique:

- ADD, SUB, MUL, DIV, MOD : Opération élémentaire entre les registres A, B, C. Du style A = B op C
- LOAD_B, LOAD_C : Charge une opérande de type entier respectivement dans B ou C
- MOV_B, MOV_C, MOV_SI, MOV_DI : Copie le registre A respectivement dans B, C, SI ou DI
- PUSH_A, PUSH_B, PUSH_C, PUSH_SI, PUSH_DI : Met sur la stack respectivement A, B, C, SI ou DI
- POP_A, POP_B, POP_C, POP_SI, POP_DI : Charge de la stack respectivement A, B, C, SI ou DI
- JMP et autre J* : Pour les instructions de saut (utilise la valeur du flags pour les sauts conditionnels)
- CMP_AB, CMP_BC, CMP_AC : Pour les instructions de comparaisons (met à jours les flags)
- CALL : Pour l'appel de fonction
- RET : Instruction de retour de fonction
- DUMP : Instruction qui affiche l'état du processeur Frog
- PRINTS : Instruction qui affiche une opérande de type chaîne de caractère sur la sortie standard
- PRINTD : Instruction qui affiche une chaîne de caractère stocké à l'adresse DI en mémoire sur la sortie standard
- MOVMEM : Copie A octet en mémoire de SI vers DI
- FETCH_BUF : Charge jusqu'à 128 octet d'opérande à l'index DI en mémoire
- SWAPDISI : Inverse DI et SI

2. Mode d’adressage et instruction de saut:

Les instructions de la Vm Frog sont données en mode “TEXTE” et donc ne sont pas représenté en mémoire sous forme d’octet. La Vm Frog travaille directement en adressage “instruction”. Ainsi pour les instructions de saut, l’opérande prise par les fonctions J* et CALL indique le nombre d’instruction (positif ou négatif) qu’il faut exécuter après l’instruction de saut. Par exemple, ce code affiche 12 fois coucou.

```
load_b 12
load_c -1
add
dump
prints coucou
mov_b
jne -5
```

3. CLI: Command Line Interface:

L'exécution du programme my_vmX lorsqu'aucun programme n'est passé en paramètre passe en mode CLI, et lit les commandes sur l'entrée standard. Vous pouvez directement saisir des commandes du processeur grenouille, ainsi que 2 commandes spécifiques de l'interface CLI:

- EXEC : Exécute toutes les commandes précédemment tapées
- INCLUDE File.Insn : Charge et exécute les commandes à partir du fichier "File.insn"

```
user$ ./my_vm1
Simple Virtual Machine helping you to understand BOF/ROP/etc...
Enter command: load_b 12
Enter command: load_c 4
Enter command: add
Enter command: dump
Enter command: exec
A:16 B:12 C:4 SI:0 DI:0 IP:3
Enter command:
```

3. 1. Niveaux

Niveau 1

Ce niveau est un niveau de prise en main. Le programme est seulement compilé avec une pile non exécutable… Vous devez comprendre le code de la Vm qui vous est fourni car il contient une faille qu'il va falloir exploiter. Il s'agit donc de pratiquer un buffer overflow de tel manière qu'une certaine fonction de vuln.c soit appelé. Il va donc falloir lire le code source et le comprendre ! Pour vous aider, répondez au formulaire SATA: my_vm (qui vous a été partagé). Votre payload peut être généré à froid une bonne fois pour toute car l'ASLR est désactivé, donc les adresses sont fixes.

Votre script './exploit1.py' génère un fichier xploit1.insn qu'on passe directement à l'interpréteur.

```
user$ ./exploit1.py
user$ ./my_vm1 ./xploit1.insn
$ whoami
root
```

Niveau 2

Similaire au niveau 1, il faut toutefois passer un argument à la fonction vers le bon paramètre pour invoquer execve avec /bin/sh. Regarder les astuces de Shellcode du précis.

Niveau 3

Similaire au niveau 2 mais avec un canary en plus et sans ASLR. Vous allez devoir via une technique interactive automatisé obtenir des fuites d'informations pour deviner le canary. A partir de ce niveau, votre exploit peut foirer de temps en temps. Pas de panique! Vous ne serez pas pénalisé sur ce point.

```
user$ ./exploit3.py
$ whoami
root
```

Niveau 4

L'ASLR est actif et un canary va vous barrer le chemin. Vous allez devoir deviner en plus du canary les adresses relatives au run du programme afin de forger une payload à chaud vous permettant d'exécuter pwn_func4 avec le bon paramètre.

Niveau 5

Vous devez forger un paramètre plus compliqué (char **) à envoyer à la fonction pwn.

Niveau 6

Vous n'avez plus d'aide pour trouver des gadgets.

Niveau 7

Vous n'avez plus d'aide pour trouver des gadgets, ni de fonction cible déjà présente dans le programme. Vous devrez réaliser le setuid(0) et execvp('/bin/sh') par vos propres moyen.

Niveau 8

Vous n'avez plus l'instruction fetch_mem. Bon courage!