P. Lafourcade, C. Olivier-Anclin, M. Puys

TP3 BoF et ROP

BoF & Reverse

Exercise 1 (BoF (3 points))
Créez un fichier nommé `exo1.c` avec le code contenu dans la Figure 1 et compilez le avec la commande `gcc exo1.c`

Ensuite exécuter le en tapant par exemple : `./a.out`

Et en saisissant votre chaîne de caractères sur la prochaine ligne.

Question : Comment modifier la variable `modified` qui vaut 0, seulement en executant le programme avec la bonne valeur? Expliquer votre raisonnement et votre approche.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv)
{
  int modified;
  char buffer[64];

  modified = 0;
  scanf("%s", buffer);

  if(modified != 0) {
      printf("you have changed the 'modified' variable\n");
  } else {
      printf("Try again?\n");
  }
  return 0;
}
```

Figure 1: Code de *exo1.c*

Exercise 2 (Integer Overflow (5 points))
Créez un fichier nommé `exo2.c` avec le code contenu dans la Figure 2 et compilez le avec la commande `gcc exo2.c`

Ensuite exécuter le en tapant par exemple : `./a.out str1 str2 len1 len2` avec `str1` et `str2` des chaines de caractères et `len1` et `len2` des entiers.

1. (1 point) Que fait ce programme, donnez un exemple de fonctionnement normal ?

2. (1 point) Quelle est la longueur d'un `unsigned char` et sa valeur maximale ?

3. (2 points) Comment modifier la variable `d.secret` qui vaut 0, seulement en executant le programme avec la bonne valeur ?

4. (1 point) comment faire pour que la variable `d.secret` prenne exactement la valeur `0x42`. Expliquer votre raisonnement et votre approche.

Exercise 3 (Reverse (5 points))
Installer le logiciel Ghidra[1], puis télécharger le binaire suivant qui a été compiler sur une architecure ARM : `https://sancy.iut.uca.fr/~lafourcade/SECWEB/civ.out`

Si vous avez une architecture ARM vous pourrez l'éxécuter, sinon voici une exécution de ce programme :

```
=== Break My ARM ===
Password: 12345678990

Nope. You need to feel the Force, young Padawan.
```

À l'aide de Ghidra, retrouver le programme C et comprenez comment il fontionne et tenter de trouver le bon password.

ROP (4 points)

Avant de commencer le TP, assurez vous de bien configurer votre machine à l'aide des étapes suivantes : Cloner le repository

```
git clone https://gitlab.limos.fr/palafour/tp-rop.git
```

Vous pouvez consulter la documentation sur l'outil 'ROP Gadget' qui pourra vous être utile durant le TP :

`https://github.com/JonathanSalwan/ROPgadget`

ROPGadget est un outil développé par le franąis Jonathan Salwan. Cet outil vous permet de rechercher vos gadgets sur vos binaires pour faciliter votre exploitation ROP. ROPgadget prend en charge les formats ELF/PE/Mach-O/Raw sur les architectures x86, x64, ARM, ARM64, PowerPC, SPARC, MIPS, RISC-V 64 et RISC-V Compressed.

Exercise 4 (ROPgadget (4 points))
Soit le fichier `vulnerable.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char ** argv) {
    char buff[128];
```

[1] `https://ghidra-sre.org/`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct data {
  char buffer[200];
  unsigned char secret;
  char out[200];
};

int copy(char * buf1, char * buf2, unsigned char len1, unsigned char len2) {
  struct data d;
  d.secret = 0;
  memset(d.buffer, 0, 200);  // Fills with 0.
  memset(d.out, 0, 200);     // Fills with 0.

  printf("[BEFORE] buffer = \"%s\"\n", d.buffer);
  printf("[BEFORE] secret = 0x%X\n", d.secret);
  printf("[BEFORE] out = \"%s\"\n", d.out);

  printf("[DEBUG] len1+len2 = %u\n", len1+len2);
  if((unsigned char)(len1 + len2) > 200) {
    printf("[ERROR] Size too long!\n");
    return -1;
  }

  memcpy(d.buffer, buf1, len1);
  memcpy(d.buffer + len1, buf2, len2);
  printf("[AFTER] buffer = \"%s\"\n", d.buffer);
  printf("[AFTER] secret = 0x%x\n",d.secret);
  printf("[AFTER] out = \"%s\"\n", d.out);

  return 1;
}

int main(int argc, char * argv[]) {
  copy(argv[1], argv[2], atoi(argv[3]), atoi(argv[4]));
}
```

Figure 2: Code de *exo2.c*

```
    gets(buff);

    char *password = "I am TP ROP PM 2 !";
```

```
    if (strcmp(buff, password)) {
        printf("You password is incorrect\n");
    } else {
        printf("Access GRANTED !\n");
    }

    return 0;
}
```

Utilisez `clang` pour compiler le fichier `vulnerable.c` :

```
clang -o rop vulnerable.c -m64 -fno-stack-protector -Wl,-z,relro,-z,now,-z,noexecstack -static
```

Explication: Voici le sens des différentes parties figurant dans le commandement précédent.

- clang : Le compilateur Clang est utilisé pour compiler le code source.
- -o rop : Spécifie le nom du fichier de sortie, "rop" dans ce cas.
- vulnerable.c : Le fichier source C à compiler, "vulnerable.c" ici.
- -m64 : Indique que l'on souhaite compiler en mode 64 bits.
- -fno-stack-protector : Désactive la protection du stack, ce qui peut être nécessaire pour les exploits.
- -Wl,-z,relro,-z,now,-z,noexecstack : Options passées au linker (ld) pour spécifier diverses options de sécurité :
- -z,relro : Active la relocalisation en lecture seule, renforcant la sécurité des structures de données.
- -z,now : Demande au linker de résoudre toutes les références symboliques immédiatement, plutôt qu'à la demande, renforçant ainsi la sécurité.
- -z,noexecstack : Empêche l'exécution de code à partir de la pile.
- -static : Indique au linker de lier statiquement les bibliothèques, plutôt que dynamiquement. Cela signifie que toutes les bibliothèques seront incluses dans le binaire final.

1. Lorsque vous compilez, quelle fonction est annoncée comme dangereuse et présente la première faille pour effectuer l'attaque ROP ?

   On remarque donc la présence d'un 'buffer overflow', nous allons désormais passer à ce binaire un gros buffer à l'aide de la commande suivante :

   ```
   perl -e 'print "A"x500' | ./rop
   ```

2. Combien de caractères sont nécessaires pour causer cette erreur ?

L'outil ROPgadget. Pour installer ROPgadget, il faut lancer un `venv` et faire les commandes suivantes :

```
python3 -m venv .
source bin/activate
python3 -m pip install ROPgadget
python3 -m pip install pwn
ROPgadget --help
```

Pour lancer `ROPgadget` sur un binaire utilisez l'option `--binary`

Pour cela, nous vous avons fourni le fichier `pyscript.py`. Ce fichier est un template pour constituer la fameuse **chaine de gadgets**.

```
#!/usr/bin/env python3
# execve generated by ROPgadget

from pwn import *
from struct import pack

p = b''

r = process('./rop')

# A COMPLETER
# A COMPLETER
# ...

r.sendline(p)
r.interactive()
```

Voici une explication du contenu du fichier fourni :

- La bibliothèque "pwn" est couramment utilisée dans le domaine de l'exploitation de vulnérabilités pour simplifier la création de chaînes ROP (Return-Oriented Programming) et la manipulation de processus. Elle offre des outils et des fonctions qui rendent plus facile la construction d'une ROPchain.
- La variable p est initialement définie comme une chaîne de caractères vide. Nous allons concatener les gadgets dessus grâce a pwn.
- Le script créé un processus en exécutant notre exécutable "rop" (le binaire du programme c vulnérable),avec la ligne `r = process('./rop')`. C'est la préparation de l'attaque contre ce programme.
- Une fois la RopChain concaténer à `p`, nous allons la fournir au programme via le processus `r`: avec `r.sendline(p)`.
- Ensuite, nous rentrons en mode interactif avec `r.interactive()`, ce qui nous permettra d'intéragir avec le shell

Il faut le completer en ajoutant le buffer overflow puis le code donné par l'outil ROPgadget pour pouvoir ouvrir un `shell`.

Des mécanismes de défense Il existe deux mechanisme de défense parmis les plus classiques: DEP et ASLR. Nous allons regarder si ces derniers sont bien actifs.

DEP est une mesure de sécurité qui vise à empècher l'exécution de code malveillant dans des régions de la mémoire réservées aux données.

En utilisant la commande qui suit, prouvez que l'exécution du code malveillant est bien empèchée :

```
readelf -l rop
```

1. Quelle est la preuve que l'exécution du code malveillant est bien empéchée ?

ASLR est une technique qui vise à rendre plus difficile l'exploitation de vulnérabilités logicielles en modifiant aléatoirement la disposition des composants clés de l'espace d'adressage d'un processus lors de son démarrage.

Vérifiez que l'ASLR est bien activée sur votre machine à l'aide de la commande suivante qui doit vous retourner `2` :

```
cat /proc/sys/kernel/randomize_va_space
```

Exercise 5 (BoF le retour (3 points))
Compiler le code de la figure 3 à l'IUT et déterminer quelle valeur peremet d'afficher "you have correctly got the variable to the right value"? Expliquer votre raisonnement et votre approche.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
  volatile int modified;
  char buffer[64];

  if(argc == 1) {
      printf("please specify an argument\n");
      return 1;
  }

  modified = 0;
  strcpy(buffer, argv[1]);

  if(modified == 0x61626364) {
      printf("you have correctly got the variable to the right value\n");
  } else {
      printf("Try again, you got %d\n", modified);
  }
}
```

Figure 3: Code de BoF le retour