Reverse - PredictMyRandom - 75 points

Waïl TAHMAOUI, Kévin DUVERGER

Table des matières

1 Résolution PredictMyRandom :

1.1 Avec des outils en ligne :

On peut tenter d'envoyer notre programme dans `https://dogbolt.org/`. En explorant les différents décompilateurs, celui qui semble donner le code le plus lisible est `BinaryNinja` :

```
int32_t _computeRandom() {
    _.bss = (_.bss * 0x41c64e6d + 0x3039) & 0x7fffffff;
    return _.bss;
}

int32_t _main() {
    _.bss = time(nullptr);
    printf("Please give the number : ");
    fflush(_.data(1));
    char var_80[0x64];
    fgets(&var_80, 0x64, _.data(0));
    uint32_t eax_5 = strtoul(&var_80, nullptr, 0xa);
    int32_t eax_6 = _computeRandom();
    if (eax_5 != eax_6) {
        printf("Oh no, you did not find the random value (it was %d) !\n", eax_6);
    } else {
        puts("Yay ! You won !");
        FILE* _Stream = fopen("flag.txt", "r");
        fgets(&var_80, 0x64, _Stream);
        fclose(_Stream);
        printf("Here is the flag : %s\n", &var_80);
    }
    return 0;
}
```

En analysant un peu le code, vous pouvez voir qu'un nombre est demandé à l'utilisateur (via le `fgets`), puis cela est mis dans le buffer `var_80`. Le programme produit ensuite un nombre depuis le buffer, puis calcule un autre nombre via `computeRandom` (qui opère sur la variable `_.bss` initialisée au timestamp actuel).
Comme vous pouvze le voir on a un générateur d'aléatoire dont la formule est $s_{i+1} = (s_i \times 1103515245 + 12345) \pmod{2^{31}}$ et la valeur aléatoire utilisée est s_{i+1}. En regardant un peu plus le code, vous voyez qu'il faut qu'on arrive à envoyer le même aléatoire que celui calculé par le programme, c'est à dire $(\text{time} \times 1103515245 + 12345) \pmod{2^{31}}$.

La première solution consiste à prendre un timestamp assez proche de l'heure à laquelle vous êtes (à la date de rédaction de ce document, le timestamp est `1759589629`), puis de calculer la valeur attendue et de spammer le service avec cette valeur.

Une autre façon de faire consiste à coder un programme qui va lui aussi récupérer le timestamp actuel (avec peut être un petit delta car votre horloge n'est pas exactement la même que celle du serveur et car le paquet prend du temps à traverser le réseau) puis va faire le calcul :

```
import os, sys, socket
import time

server = "146.59.227.136"
port = 23401

ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    ma_socket.connect((server, port))
except Exception as e:
    print("Problème de connexion", e.args)
    sys.exit(1)

toSend = (int(time.time() + 1) * 1103515245 + 12345) % (2 ** 31)
ligne = ma_socket.recv(4096).decode("utf-8")
ma_socket.sendall(str(toSend).encode("utf-8") + b"\n")
print(ma_socket.recv(4096).decode("utf-8"))

ma_socket.close()
```

1.2 Avec Ghidra :

En envoyant le programme dans Ghidra, voici ce que l'on peut obtenir :

```
int __cdecl _main(int _Argc,char **_Argv,char **_Env) {
    FILE *pFVar1;
    time_t tVar2;
    char acStack_80 [100];
```

```
    FILE *pFStack_1c;
    uint uStack_18;
    ulong uStack_14;

    ___main();
    tVar2 = _time((time_t *)0x0);
    _currentGeneratorState = (undefined4)tVar2;
    _printf("Please give the number : ");
    pFVar1 = (FILE *)(*(code *)__imp____acrt_iob_func)(1);
    _fflush(pFVar1);
    pFVar1 = (FILE *)(*(code *)__imp____acrt_iob_func)(0);
    _fgets(acStack_80,100,pFVar1);
    uStack_14 = _strtoul(acStack_80,(char **)0x0,10);
    uStack_18 = _computeRandom();
    if (uStack_14 == uStack_18) {
        _puts("Yay ! You won !");
        pFStack_1c = _fopen("flag.txt","r");
        _fgets(acStack_80,100,pFStack_1c);
        _fclose(pFStack_1c);
        _printf("Here is the flag : %s\n",acStack_80);
    }
    else {
        _printf("Oh no, you did not find the random value (it was %d) !\n",uStack_18);
    }
    return 0;
}
```

En étudiant le code et en le simplifiant, nous pouvons arriver au pseudo-code suivant :

```
currentGeneratorState = time(NULL)
print("Please give the number : ")
buffer[100]
fgets(buffer, 100, stdin)
nombreUser    = strtoul(buffer, NULL, 10)
nombreMachine = computeRandom()
Si nombreUser == nombreMachine :
    Afficher le flag
Sinon :
    Afficher perdu
```

Grâce à cela, nous pouvons voir que pour obtenir le flag, il faut prédire la prochaine valeur aléatoire d'un générateur d'aléatoire représenté par `computeRandom()` et initialisé au timestamp actuel. Une recherche de cette fonction dans le code décompilé nous donne :

```
uint __cdecl computeRandom(void) {
    _currentGeneratorState = _currentGeneratorState * 0x41c64e6d + 0x3039 & 0x7fffffff;
    return _currentGeneratorState;
}
```

Ici, il suffit donc de coder la même fonction et de communiquer avec le serveur (automatiquement) la valeur aléatoire calculée. Attention, il se peut que l'horloge du serveur ne soit pas exactement au même timestamp que vous, et que le paquet prenne du temps à passer sur le réseau, donc il faudra peut-être rajouter un petit δ à la valeur du timestamp actuel. Ici, avec Python, si on ne veux pas se casser la tête avec une communication réseau avec le serveur, on peut copier-coller à la main le prochain état de notre générateur :

```
import time

def compute_random():
    now = int(time.time()) + 2
    return (now * 0x41c64e6d + 0x3039) & 0x7fffffff

print(compute_random())
```

Le +2 est là pour laisser 2 secondes à l'attaquant pour prédire le prochain nombre pseudo-aléatoire généré. Un simple ctrl+c, ctrl+v prends environ ce temps, et donc on s'économise bien du temps de programmation (pour l'instant).