[RElog] Pwnable (sheadovas/artykuly/relog/pwnable/)

Kwi 02, 2016 / RElog (sheadovas/category/artykuly/relog/)

Pseudo-writeup zadania „random”.

Witam Was w kolejnym (tym razem dość krótkim) RElogu, czyli luźnej serii powiązanej inżynierią wsteczną. W tym wpisie chciałbym Wam pokazać serwis na którym można w dość przyjemny sposób rozwinąć swoje umiejętności „hakerskie”.

Pwn? Ctf?

W skrócie: [CTF (https://ctftime.org/ctf-wtf/)] (capture the flag) to zadania/wyzwania polegające na znalezienie i odczytanie flagi.

Z kolei pwn oznacza przejęcie kontroli nad np innym komputerem, stroną, aplikacją, ... [Wiki (https://en.wikipedia.org/wiki/Pwn)].

Serwis *Pwnable* udostępnia zestaw zadań, które przypominają te z CTF'ów, ale są o wiele prostsze, dzięki czemu początkujący (w tym ja) mogą pobawić się w rozwiązywanie zadań tego typu samodzielnie. Po więcej informacji zapraszam na oficjalną stronę [Pwnable (http://pwnable.kr/index.php)], gdzie można znaleźć m.in. regulamin.

Writeup

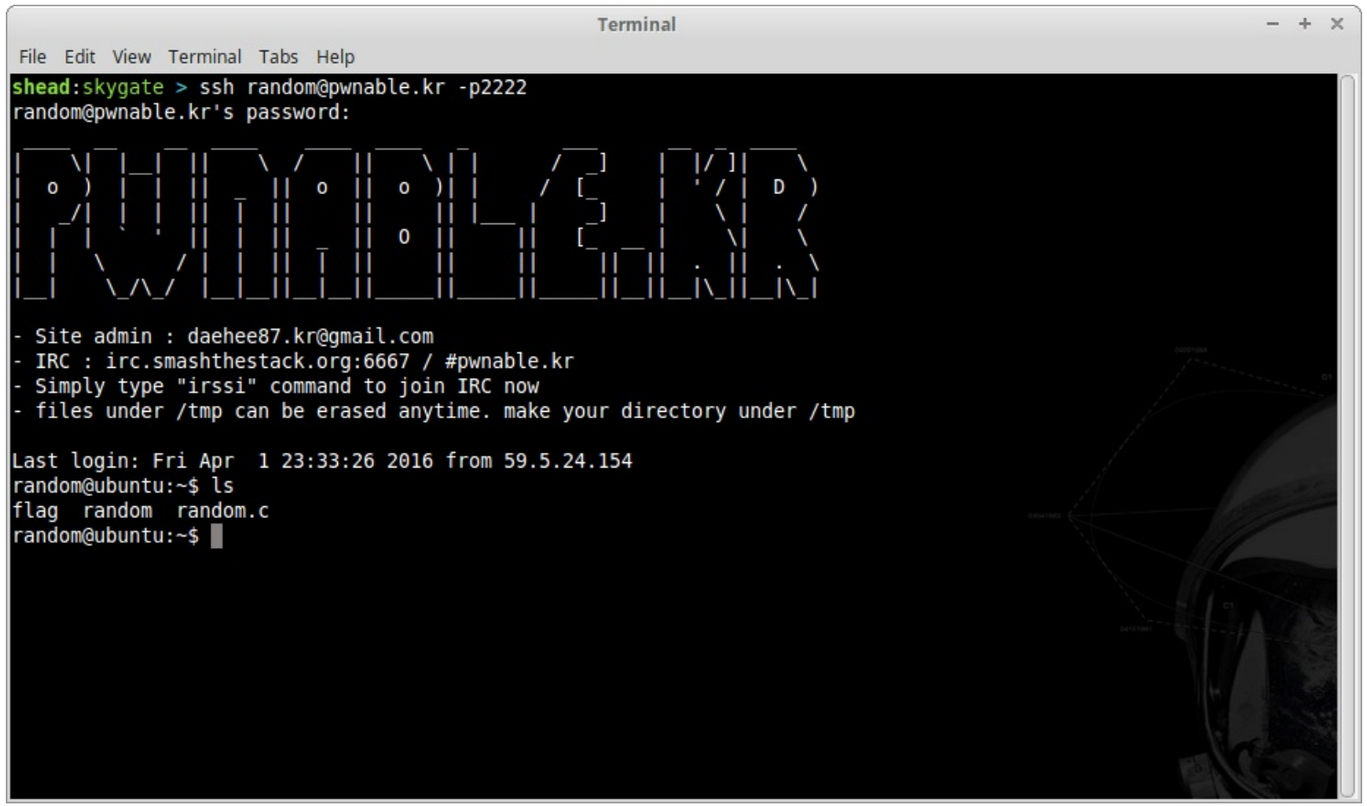
Chciałbym Wam przedstawić rozwiązanie chyba najprostszego zadania (nie chcę Wam psuć zabawy); wszystkie potrzebne informacje (dane do połączenia przez ssh) są podawane w treści zadania, podobnie sama treść niesie ze sobą małe wskazówki odnośnie tego gdzie należy szukać rozwiązania.

random (1pt)

Treść zadania

```
Daddy, teach me how to use random value in programming!

ssh random@pwnable.kr -p2222 (pw:guest)
```



(https://i0.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/04/Screenshot_2016-04-02_15-50-20.png)

Jest to chyba najprostsze dostępne zadanie. Z jego nazwy możemy wywnioskować, że będzie dotyczyła losowości, a raczej pseudo-losowości funkcji *rand()*. Po zalogowaniu się przez ssh zobaczymy w głównym katalogu 3 pliki: *flag*, *random.c*, *random*. Flaga znajduje się w pliku *flag* jednak nie możemy go odczytać, bo nie mamy odpowiednich uprawnień. Jak łatwo możemy zauważyć, to może to zrobić program *random* (ma odpowiednia uprawnienia). Rzućmy okiem na jego kod źródłowy.

random.c C

```c
#include <stdio.h>

int main(){
 unsigned int random;
 random = rand(); // random value!

 unsigned int key=0;
 scanf("%d", &key);

 if( (key ^ random) == 0xdeadbeef ){
 printf("Good!\n");
 system("/bin/cat flag");
 return 0;
 }

 printf("Wrong, maybe you should try 2^32 cases.\n");
 return 0;
}
```

Jesteśmy w stanie zauważyć podstawowy błąd: funkcja rand() o ile nie ma ustawionego ziarna na podstawie np. czasu to sama z siebie przy każdym uruchomieniu programu zwróci zawsze identyczną wartość, dalej widzimy, że musimy podać jakąś liczbę typu *uint*, która następnie jest xor'owana z liczbą „losową" i wynik tej operacji musi zwrócić *0xdeadbeef*.

Wiemy, że odwrotną operacją do xor jest xor, a więc jeżeli poznamy wartość pod zmienną random, to okazuje się że jedyną niewiadomą pozostaje klucz, który musimy podać, bo *key^random=0xdeadbeef => random^0xdeadbeef=key*. Jak widzimy po tej operacji powinniśmy móc odczytać flagę.

Zobaczmy co znajduje się pod zmienną random, do tego posłuży nam gdb.

```
dump main
```

```
random@ubuntu:~$ gdb random -q
Reading symbols from /home/random/random...(no debugging symbols found)...done.
(gdb) set disassembly-flavor intel
(gdb) disas main
Dump of assembler code for function main:
   0x00000000004005f4 <+0>: push   rbp
   0x00000000004005f5 <+1>: mov    rbp,rsp
   0x00000000004005f8 <+4>: sub    rsp,0x10
   0x00000000004005fc <+8>: mov    eax,0x0
   0x0000000000400601 <+13>: call   0x400500 <rand@plt>
   0x0000000000400606 <+18>: mov    DWORD PTR [rbp-0x4],eax
   0x0000000000400609 <+21>: mov    DWORD PTR [rbp-0x8],0x0
   0x0000000000400610 <+28>: mov    eax,0x400760
   0x0000000000400615 <+33>: lea    rdx,[rbp-0x8]
   0x0000000000400619 <+37>: mov    rsi,rdx
   0x000000000040061c <+40>: mov    rdi,rax
   0x000000000040061f <+43>: mov    eax,0x0
   0x0000000000400624 <+48>: call   0x4004f0 <__isoc99_scanf@plt>
   0x0000000000400629 <+53>: mov    eax,DWORD PTR [rbp-0x8]
   0x000000000040062c <+56>: xor    eax,DWORD PTR [rbp-0x4]
   0x000000000040062f <+59>: cmp    eax,0xdeadbeef
   0x0000000000400634 <+64>: jne    0x400656 <main+98>
   0x0000000000400636 <+66>: mov    edi,0x400763
   0x000000000040063b <+71>: call   0x4004c0 <puts@plt>
   0x0000000000400640 <+76>: mov    edi,0x400769
   0x0000000000400645 <+81>: mov    eax,0x0
   0x000000000040064a <+86>: call   0x4004d0 <system@plt>
   0x000000000040064f <+91>: mov    eax,0x0
   0x0000000000400654 <+96>: jmp    0x400665 <main+113>
   0x0000000000400656 <+98>: mov    edi,0x400778
   0x000000000040065b <+103>: call   0x4004c0 <puts@plt>
   0x0000000000400660 <+108>: mov    eax,0x0
   0x0000000000400665 <+113>: leave
```

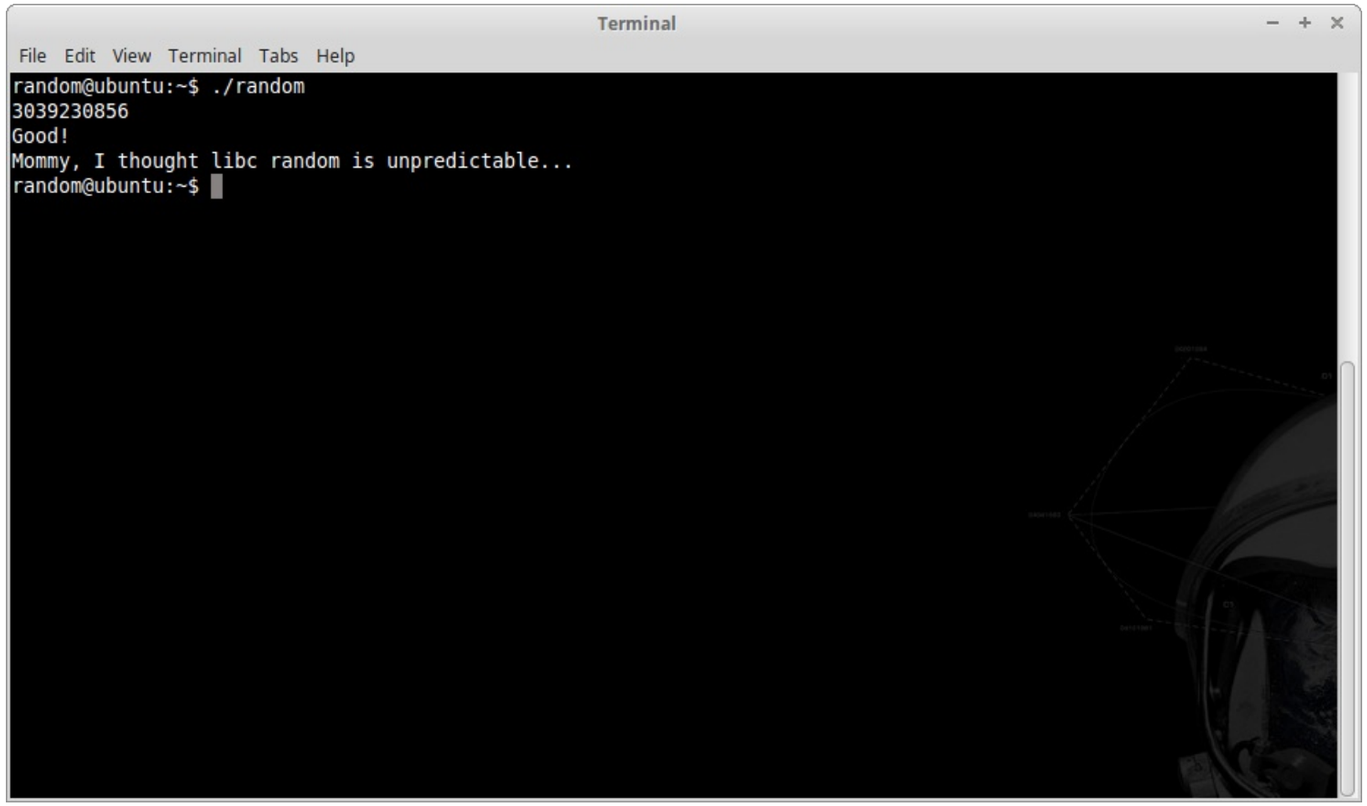
Jak widzimy zmienna random ukrywa się pod [rbp-0x4], jeżeli ustawimy break point na np [0x0000000000400610], po czym uruchomimy program to powinniśmy móc odczytać wartość tej zmiennej:

```
(gdb) x $rbp-0x4
0x7fff82d49eec: 0x6b8b4567
```

Teraz możemy już zamknąć gdb i obliczyć klucz, który sprawi że program odczyta flagę, można to zrobić dość szybko np w interpreterze Pythona

```
>>> 0x6b8b4567^0xdeadbeef
3039230856
```

Pozostało nam już teraz jedynie uruchomić program i podać wyliczony klucz po czym przekazać znalezioną flagę do *pwnable* aby uzyskać punkty za rozwiązanie zadania.



(https://i2.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/04/Screenshot_2016-04-02_16-21-09.png)

Jak widzimy zadanie było bajecznie proste i nie wymagało specjalistycznej wiedzy, nie wymagało także zbytniego myślenia; zachęcam do spróbowania własnych sił na tym serwisie.

Materiały dodatkowe

- [pwnable (http://pwnable.kr/)] – serwis przedstawiony w tym wpisie
- [ctftime (https://ctftime.org/)] – miejsce z informacjami o wszystkich ctf'ach
- [Hacking, by Gyn (http://gynvael.coldwind.pl/?id=366)] – artykuł zawierający informacje o hackingu oraz listę stron w stylu pwnable

Już w kolejnym *REIogu* porozmawiamy sobie o ukrywaniu danych i obfuskacji kodu maszynowego.

Code ON!