Trong video khai thác BoF (<https://whitehat.vn/threads/video-huong-dan-khai-thac-bof-co-ban.8391/>), chúng ta có dùng đến một con shellcode là binsh. Hôm nay mình sẽ hướng dẫn các bạn cách hoạt động của một con shellcode và cách viết nó.

Linh các file đính kèm (con shellcode binsh, file testShellcode.c):

Môi trường: Linux, viết shellcode 32bit.

Khi viết shellcode trên môi trường Linux, ta dùng đến các system call. Trên Windows thường không làm vậy, mà phải gọi tới các hàm trong thư viện dll, vì hệ thống system call của nó qua các đời Windows bị thay đổi, mà không phải việc gì cũng làm được. Còn trên Linux thì ngược lại, hỗ trợ mọi việc từ ghi file tới mở kết nối socket.

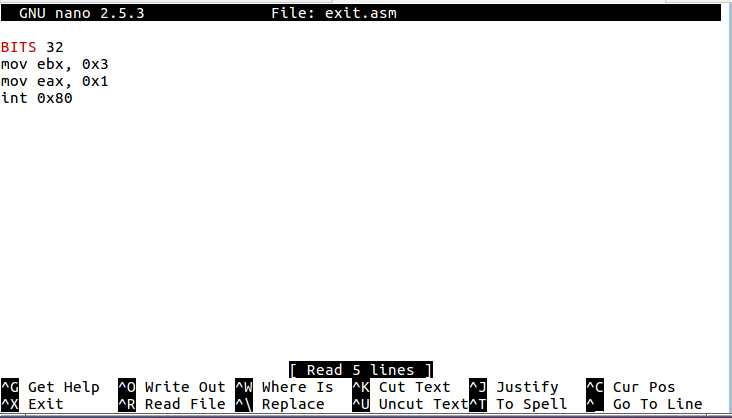
System call là các lời gọi tới các xử lý ngắt, mà các xử lý này được thực hiện trong kernel. Khi ta gọi một hàm trên hệ thống 32bit, thì ta phải truyền tham số vào stack và thực hiện lệnh call. Nhưng khi thực hiện gọi một system call, ta sẽ truyền các tham số vào các thanh ghi, và thực hiện lệnh int 0x80.

Ví dụ một con shellcode đơn giản nhất, chỉ thực hiện công việc tương tự hàm exit, qua lời gọi system call sys\_exit:

|  |
| --- |
| mov ebx, 0x3  mov eax, 1  int 0x80 |

Shellcode trên tương đương với lời gọi hàm exit(0x3). Trong Linux, hàm exit có system call tương ứng của nó, nhận 1 tham số. Tham số này được truyền qua thanh ghi ebx (mov ebx, 0x3). Còn system call exit có số hiệu 1, nên ta truyền eax bằng 1 (mov eax, 0x1). Cuối cùng gọi lệnh int 0x80 là xong.

Để biên dịch con shellcode trên, ta tạo một file exit.asm với nội dung:

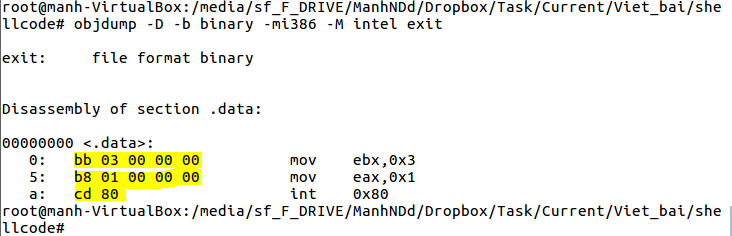


Và ta dùng nasm để biên dịch: nasm exit.asm -o exit

Dòng BITS 32 nhằm chỉ cho nasm biết là ta cần biên dịch mã 32bit.

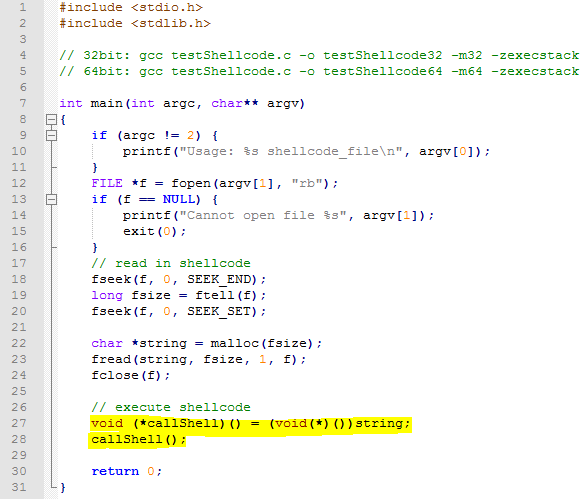
Để dump shellcode ra, ta dùng objdump:

**objdump -D -b binary -mi386 -M intel fileName**



Như hình trên ta thấy vùng bôi màu vàng chính là mã hex của shellcode sau khi được biên dịch.

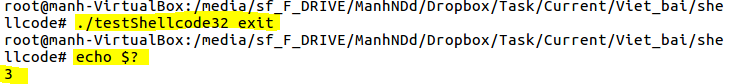
Để chạy shellcode này, ta dùng một chương trình gọi chạy shellcode như sau:



Chương trình trên nhận tham số thứ nhất là file shellcode; đọc shellcode vào một string, và thực thi vùng shellcode ấy. Vì shellcode 32bit nên ta biên dịch cần tham số -m32. Còn -execstack là để cho phép thực thi dữ liệu, như vậy shellcode mới có quyền chạy (theo như kiến thức phần NX - <https://whitehat.vn/threads/gioi-thieu-nx-va-aslr.8446/>)

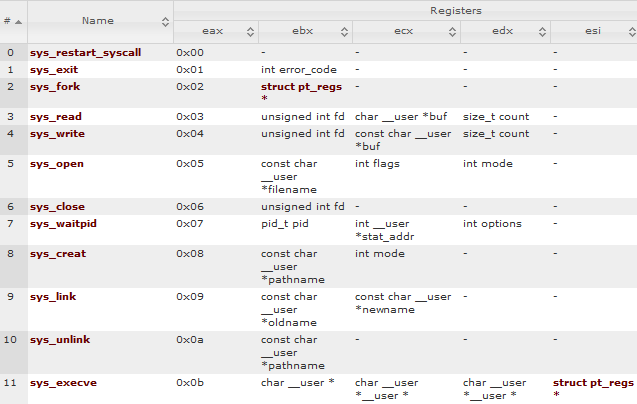
gcc testShellcode.c -o testShellcode32 -m32 -zexecstack

Ta gọi chạy fileshellcode exit, và in ra giá trị trả về:



Nhận thấy giá trị trả về là 3 => shellcode đã thực thi thành công!

Để tra số hiệu của các system call, có thể xem tại <http://syscalls.kernelgrok.com/>.



Ví dụ, sys\_read có số hiệu 3. Hay để gọi sys\_write, ta phải truyền eax = 0x4.

Trong bảng trên, các tham số cần truyền vào cho ebx, ecx,… không được chuẩn, ví dụ đối với sys\_execve, các tham số thể hiện như trên là sai. Để biết phải truyền các tham số cho sys\_execve như thế nào, ta sử dụng định nghĩa hàm:

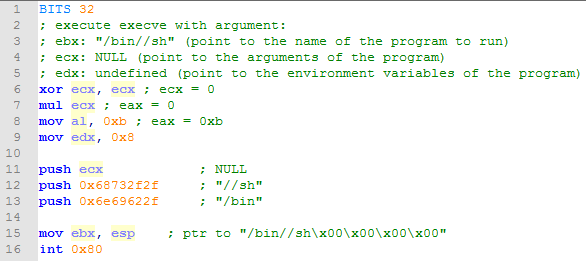
* Tra số hiệu được eax = 0xb
* Thứ tự truyền các tham số lần lượt là ebx, ecx, edx, esi, edi
* Xem định nghĩa hàm bằng man execve



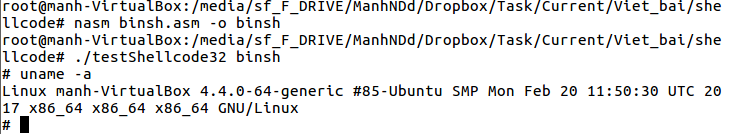
Vậy cần truyền ebx = filename, ecx = argv, edx = envp.

Con shellcode binsh chỉ đơn giản gọi đến hàm execve với các tham số như sau:

* filename = “/bin//sh\x00\x00\x00\x00”
* argv = NULL
* envp không xác định (không truyền gì vào cho edx)



Người ta truyền các tham số không tuân quy tắc như trên là để làm sao cho shellcode ngắn nhất có thể, và không có byte 0x00. Con shellcode binsh tạo ra chỉ dài 21 byte. Test chạy thử shellcode:



Tổng kết lại, ta có:

* Shellcode thường được gọi lên các system call. Sử dụng system call table để tra danh sách system call.
* Sử dụng nasm để viết shellcode.
* Dùng testShellcode để kiểm tra shellcode có chạy hay không (debug bằng gdb nếu cần).