CS392_MidtermTest

The name and photo associated with your Google account will be recorded when you upload files and submit this form.

Not **maheeth2000@gmail.com**? Switch account

* Required

Email address *

maheeth2000@gmail.com

Rollno *

1801CS31

Name *

Maheeth Reddy

Q1) Amar's problem

The answer is NO.
After executing the command "sudo chmod 4755 prog", the prog file becomes a SET-UID program. But, after executing the command, "sudo chown root prog", the SET-UID bit is reset.
If we execute the commands in reverse order, then the goal can be achieved.

Q2) Kanchan's problem

Stack frame layout mentioned in supporing file.

Q3) Shubham's problem

Role of ebp register in Buffer Flow Attack
The following happens when a caller function calls a callee function: The caller's epilogue is performed followed by the callee's prologue. In the epilogue of caller function, a new stack frame is created for the callee function and the current frame pointer is pushed into the new stack frame. The return address of the callee function is also pushed to stack frame, which points back to the caller function. These are used in the epilogue of the callee function to return to the caller function.

During a buffer overflow attack, the ebp pointer and then the return address within the newly created stack frame are overwritten. So, when the epilogue of the callee function is executed, we expect the ebp to go back to the cllaer frame and esp to the location where the caller called callee.

But now, the ebp value will point to an arbitrary location and esp shall point to the malicious code injected by the attacker. If the malicious code is a shell code and can be executed, then ebp value won't affect the buffer overflow attack as there is no requirement to jump to another frame from the malicious code. But in case the attack fails then during the second epilogue we may get error.

Q4) Deepak's problem

Actual content of the input file is obtained by executing the following command:

```
echo $(printf
"\xa8\xcd\xbb\xaa@@@@\xa6\xcd\xbb\xaa")_%.8x_%.8x_%.8x_%.8x_%.43658x%hn_%.8737
x%hn$(printf "malicious code") > input
```

Justification:
To execute Deepak's code, he needs to place the address 0xAABBCCDD into the return address (0xAABBCDA6).

So, we break 0xAABBCDA6 into two contiguous 2-byte memory locations : 0xAABBCDA6 and 0xAABBCDA8.
Store 0xCCDD into 0xAABBCDA6 and 0xAABB into 0xAABBCDA8

Due to the above command, Number of characters printed before first %hn =
12 + (4x8) + 5 + 43658 = 43707 (0xAABB).

After first %hn, 8737 + 1 =8738 characters are printed

43707 + 8738 = 0xCCDD and hence 0xAABBCCDD is printed on 0xAABBCDA6

Q5) Lalit's problem

To crash the program we need minimum of 5 format specifiers.

Reason:
We can use: printf("\x22\x33\x66\x99_%08x.%08x.%08x.%08x.|%s|")

Stack content: 0xAABBCCDD, 0xAABBDDFF, 0x22334455, 0x00000000, 0x99663322 (address low to high)

The four format specifiers %08x move the printf()'s argument pointer from 0xAABBCCDD to the address 0x99663322 that we stored in the format string. Then, we use %s to print the contents of address 0x99663322. Because of format-string vulnerability, printf() considers them as arguments to match with the %x in the format string. After printf()'s argument pointer points to address 0x99663322, %s acesses the data stored at the address pointed by value at 0x99663322. This causes the program to crash as that address space would be protected.

Supporting pdf file if any (max size 10 MB)

1801CS31-midse...

A copy of your responses will be emailed to the address you provided.

Submit