Nish Patel

Part 2 Documentation

Methodology and Thought Process:

Buffer Overflow: To successfully exploit the buffer overflow vulnerability – `strcpy()` in `test()` – I used the `gdb` debugger to examine the binary and determine the proper payload to overwrite the return address of `test()`. Using a breakpoint at the end of `test()`, I checked the memory around the local array `test[17]` to see how large the buffer and gap were for writing the necessary `A`'s in the input. Eventually, with comparing to the usual saved return address, I deduced the number of A's and appended the string with the address of `log_result_advanced()`, `0x08048EA0`.

The general approach was to use a bunch of letters (in this case, `A`'s) to ultimately overflow the buffer, since `strcpy()` would just 'dump' the whole input into test regardless of the expected size, and ultimately replace the saved return address `($ebp+4)` with the address of `log_result_advanced()`…

```
0xffffdd9f:    0x41414141    0x41414141    0x41414141    0x41414141
0xffffddaf:    0x41414141    0x41414141    0x41414141    0x08048ea0
```

(Rough snippet; picture may not exactly reflect stack frame when working on the project)

Unlike part 1, the input argument for the `log_result_advanced(print)` function needed to be overwritten as well, given the if-condition that argument `(print == 0xefbeadde)`. I used additional `A`'s to overwrite through the gap after the saved return address toward the parameters – which seemed to be at `($ebp+8)` – and overwrote it with `0xDEADBEEF`.

---

*Input Provided to Binary (**Exploit Payload**)*:

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAA$'\xA0\x8E\x04\x08'AAAA$'\xDE\xAD\xBE\xEF'
```