Sploit1: Buffer Overflow

A buffer overflow exploit targets areas in the code where the data copied into a buffer is larger than the length of the buffer. The excess data runs up the stack past the end of the buffer, overwriting data in the higher addresses of the stack. The goal of the overflow is to overwrite the location of the saved return address with a different address, controlled by the exploiter.

In the submit program, this exploit is performed at line 369, where the text that describes the usage of the program is printed out. The snprintf on this line enforces a length of at most 640 to be copied into the txt buffer, but the size of the txt buffer is only 171 bytes. Any additional bytes will run off the end of the buffer and overwrite the stack.

The format string inserts the first argument to the program into the buffer, which it expects to be the name of the program as passed from execve, but it is in fact the string that will cause the exploit.

An environment variable containing the code to execute /bin/sh is sent to the program and located at 0xffbfdfb8.

The difference between the address of the start of the txt buffer and the address of the saved return address is 176 bytes. Since 7 bytes are already consumed by the snprintf with "Usage: ", the overflow from the txt buffer exploit must be 169 bytes of padding, followed by 5 bytes for the address of the environment variable, including the null terminator.

Then when the print_usage method returns, it will return to the overwritten return address, which is the address of the location of the environment variable, and /bin/sh will be run, providing root access.

This exploit can be fixed by changing the size limit in the snprintf on line 369 to 171, the size of the txt buffer. That way no data can be written outside the buffer, and the higher addresses of the stack cannot be modified.

Sploit2: Format String Vulnerability

A format string vulnerability is present when a print method such as printf is only given one argument, such as

printf(text);

instead of format arguments like below:

printf("%s", text);

This vulnerability allows malicious users to create text string such as "%08x %08x", which expect arguments to be given to printf to replace the format specifiers, but when none are provided, the specifiers are evaluated with the values of addresses moving up the stack from low to high addresses.

This allows users to examine the stack, and even change it. The format string maintains a pointer to the stack location of the current format parameter. Since none are provided, this pointer walks up the stack as more specifiers are evaluated. Once the pointer has been adjusted to the location of the saved return address, the specifier %n will take the value at the current address and dereference it, writing the number of bytes output so far in the format string to that address.

By writing to the environment variable address like in the previous exploit, root access can be attained by running /bin/sh.

In the submit program, line 396 is targetted because it calls print(source) directly with a user supplied string passed from execve.

After inspection, it was determined that there are 64 bytes, or 16 words, between the start of the format function pointer and the address on the stack where the format string is stored.

The format string begins with alternating 4-byte dumbie strings and 4 strings containing the address where the saved return address is stored, each offset by one byte. It then contains the correct amount of padding, 16 words, to advance the format pointer to the location of the format string on the stack. Finally, it contains the four alternating %iu and %n combinations, where i is the correct amount of padding to make the following %n write each byte of the address of the environment variable to the value at the address of the saved return address.

After using %n four times with the correct paddings, the saved return address now contains the address of the environment variable, and when the format string frame is popped, execution will return to the environment variable, which runs /bin/sh and gives the user root access.

The simple fix for this exploit is to replace the printf(source) line on 369 with printf("%s", source), as then the format string specifiers will have matching arguments and the stack cannot be read or written to. Any specifiers in the source will not be evaluated, since they are inserted into the format string, rather than a part of the format string.