CS 458 – A1 Milestone

Name: Dweep Shah
U.W. ID: dm2shah
Student Number: 20511868

Sploit1.c

This exploit uses the buffer overflow technique to overflow the salt (-s) option of the pwgen program. This exploit specifically leverages the vulnerability on line 278 to execute the buffer overflow threat.

```
strcpy(args.salt, optarg);
```

Inside the parse_args function the argument is taken from argv parameter, and copied into the args.salt buffer. Since strcpy doesn't check the length of the string being copied we are able to execute the buffer overflow attack to change the return address after the parse_args function completes.

First, to determine the length of the buffer needed we use gdb to determine the address of where args.salt is stored on the stack (print &args.salt). This address is determined to be 0xffbfd7df. Next, the address of where the return address of instruction pointer is stored on the stack is determined to be 0xffbfda0c. Since the difference between these addresses is 557 bytes, a buffer of 562 bytes is used, to have 4 bytes to override the return address and an extra byte for the null character to end the string.

Next, using the buffer of this size we construct a salt which is padded first with NOP, then the shellcode, and then address of where args.salt is stored on the stack (0xffbfd7df) is used. Since we get a segmentation error when we do this, we can use gdb to determine how to wrap the address, so the return address is determined to be 0xffbfd7df, and the vulnerability is exploited gaining access to the root shell.

To fix this vulnerability, we can check the length of the optarg argument being copied into args.salt so that a string greater than the maximum salt string length will not be copied over. An example of the fix is to replace line 278 with:

```
if (strlen(optarg) < SALT_SZ) {
  strcpy(args.salt, optarg);
else {
  printf("salt specified is too long");
  exit(0);
}
```

Sploit2.c