Exercise (secure coding)

Consider the following piece of C code that senselessly copies the first command-line argument into a local C-string:

```
int main(int argc, char* argv[]) {
   char opt1[128];
   if (argc < 2) {
      /* Handle error */
   }
   strcpy(opt1, argv[1]);
   return 0;
}
```

Does the code contain any vulnerability? If yes, give an example of how an attacker can exploit it. How should the code be corrected?

SOLUTION

All the "argv[*]" arguments are assured to contain a string terminators. However, the length of "argv[1]" could go beyond the capacity of "opt1", causing a buffer overflow. A solution is to use the strncpy() standard function in place of strcpy(), which lets the programmer specify the maximum number of characters to copy. The strncpy() function has the peculiarity that, if the maximum number of copied characters is copied, the destination string is left without terminator. This could lead to further vulnerabilities. To avoid non-terminated strings it is always needed to "manually" put a terminator at the end of the destination buffer. The corrected code is the following:

```
int main(int argc, char* argv[]) {
  char opt1[128];
  if (argc < 2) {
    /* Handle error */
  }
  strncpy(opt1, argv[1], 128);
  opt1[127] = '\0';
  return 0;
}
```