Uncontrolled data used in OS command

Name: Uncontrolled data used in OS command

Description: Using user-supplied data in an OS command, without neutralizing special elements, can make code vulnerable to command injection.

ID: cpp/command-line-injection

Kind: problem

Severity: error

Precision: low

The code passes user input as part of a call to `system` or `popen` without escaping special elements. It generates a command line using `sprintf`, with the user-supplied data directly passed as a formatting argument. This leaves the code vulnerable to attack by command injection.

Recommendation

Use a library routine to escape characters in the user-supplied string before passing it to a command shell.

Example

The following example runs an external command in two ways. The first way uses `sprintf` to build a command directly out of a user-supplied argument. As such, it is vulnerable to command injection. The second way quotes the user-provided value before embedding it in the command; assuming the `encodeShellString` utility is correct, this code should be safe against command injection.

```
1  int main(int argc, char** argv) {
2    char *userName = argv[2];
3
4    {
5      // BAD: a string from the user is injected directly into
6      // a command line.
7      char command1[1000] = {0};
8      sprintf(command1, "userinfo -v \"%s\"", userName);
9      system(command1);
10   }
11
12   {
13     // GOOD: the user string is encoded by a library routine.
14     char userNameQuoted[1000] = {0};
15     encodeShellString(userNameQuoted, 1000, userName);
16     char command2[1000] = {0};
17     sprintf(command2, "userinfo -v %s", userNameQuoted);
18     system(command2);
19   }
20 }
```

References

- CERT C Coding Standard: STR02-C. Sanitize data passed to complex subsystems.
- OWASP: Command Injection.
- Common Weakness Enumeration: CWE-78.
- Common Weakness Enumeration: CWE-88.