# Subtle ways your innocent program can be Internet-facing

devblogs.microsoft.com/oldnewthing/20060509-30

May 9, 2006

Raymond Chen

Last time, we left off with a promise to discuss ways your program can be Internet-facing without your even realizing it, and probably the most common place for this is the command line. Thanks to CIFS, files can be shared across the Internet and accessed via UNC notation. This means that anybody can set up a CIFS server and create files like `\\server.example.com\some\file.ext`, and they will look to the world like a file on a file server somewhere (because that is, in fact, what it is). When you double-click it, you're launching the document. And that's where the command line attack comes from. Suppose your program is a handler for a file association. Say, your program is `litware.exe` and it is the registered handler for `.LIT` files. The attacker just has to create a file called `\\server.example.com\some\path\target.lit` and induce the user into double-clicking it. Once that's done, your program will be run with the command line you registered, which will probably be

```
"C:\Program Files\Litsoft\litware.exe"
\\server.example.com\some\path\target.lit
```

Notice that the attacker controls the path. This means that if you have a bug in your command line parser, the attacker can exploit it.

> Code injection via the command line is an elevation of privilege.

Note that this extends beyond merely extra-long file names. If you registered your verb incorrectly by forgetting to put quotation marks around the file name insertion `%1`, the attacker can hatch a file with an odd name like `\\server.example.com\strange -uninstall path.lit`. The resulting command line is therefore

```
"C:\Program Files\Litsoft\litware.exe" \\server.example.com\strange -
uninstall path.lit
```

Your parser then breaks the command line up into words and interprets this command line as having three parts:

- The file `\\server.example.com\strange`
- The command line switch `-uninstall`
- The file `path.lit`.

The program then tries to load the file `\\server.example.com\strange` and fails, possibly displaying an error message, then it uninstalls itself, and then tries (and fails) to load the file `path.lit`. End result: The user gets two strange error messages and the program is uninstalled.

Of course, the attacker also controls the contents of the file, so any vulnerabilities in your file parser can be exploited as well.

> Code injection via file contents is an elevation of privilege.

If you write a shell extension, your extension will run if the user activates it on the remote file. For example, if you have a context menu extension, it will be instantiated and initialized with the remote file as the data object. Many context menu extensions contain buffer overflow bugs in the way they mishandle the names of the files that the user right-clicked on. (Notice that I said "names"—plural. The user might multi-select files and right-click on them.) For example, a certain shareware file archival program responds to the `GCS_HELPTEXT` request by taking the names of all the files and combining them into the message "Add the files A, B, C, D, and E to the archive." Unfortunately, when the names A, B, C, D, and E are very long, an exploitable buffer overrun occurs.

> Code injection triggered by file name length is an elevation of privilege.

Just because your program doesn't contact the Internet explicitly doesn't mean it's safe from Internet-based attacks.

Raymond Chen

**Follow**