Ostatnio dodawałem prostą funkcję do pewnej aplikacji webowej: umożliwienie ściągnięcia pliku z dysku. Były to pliki Excela i znajdowały się w katalogu ~/App_Data/reports.

Siłą rzeczy takie pliki nie mają ID. Mają nazwę. I po nazwie właśnie je się ściągało.

Kod otwierający plik do ściągnięcia można by napisać tak:

```
1  public Stream OpenFile(string fileName)
2  {
3      string reports_dir = HttpContext.Current.Server.MapPath("~/App_Data/reports"
4      string full_path = Path.Combine(reports_dir, fileName);
5
6      return File.OpenRead(full_path);
7  }
```

Użytkownik klika na jakiś link http://app.com/download?fileName=report1.xlsx i dostaje ładne okienko do ściągnięcia.

Piknie? Piknie!

Ale! Wejdźcie teraz na link

http://app.com/download?fileName=../../web.config . I co? Już nie tak piknie, prawda?

Takie zagrożenie siedzi u mnie w głowie od roku 2008, kiedy to na ówczesnym silniku mojego bloga (BlogEngine.NET) wykryto security issue zafixowany w tym commicie: http://blogengine.codeplex.com/SourceControl/changeset/41ac0cea1129. Nieźle, co? Setki blogów stojących na BE.NET miały hasła administratora plaintextem wpisane w web.config (bo nie było innej możliwości). A jakiś cwaniak zauważył, że gdzieś w silniku, w środku, siedzi sobie JavaScriptHandler.cs serwujący pliki JS na podstawie nazwy. Tyle że nie sprawdza czy zwraca faktycznie JSy i zwracał cokolwiek się chciało.

Dość prosto można ominąć taką lukę albo sprawdzając rozszerzenia żądanych plików (jak to zrobiono w BE.NET) albo bardziej... "pro", moim zdaniem. O tak:

```
1  public Stream OpenFile(string fileName)
2  {
3      string reports_dir = HttpContext.Current.Server.MapPath("~/App_Data/reports"
4
5      var reports_dir_info = new DirectoryInfo(reports_dir);
6      var fileInfo = reports_dir_info.EnumerateFiles(fileName).Single();
7      return fileInfo.OpenRead();
8  }
```

DirectoryInfo nie pozwoli wyjść "w górę" i udostępnia tylko pliki które faktycznie w danym katalogu się znajdują.