Depending on the functionality of the web application, it will require some sort of interaction with the underlying host system this is usually done by passing in raw system commands or input to a command shell(either directly or through some sort of library). Examples of when web applications interact with host systems involve:

- Checking monitoring statistics e.g. RAM being used, free disk space
- File conversion processes e.g. the web application would receive an image file that it wants to convert to a different image type
- Leaving debug functionality open; some frameworks have optional debug functionality that involve interaction with the underlying file system

Any input that is controlled by a user shouldn't be trusted by the server. User input could be manipulated. In the case of when a web application uses system commands, a user could manipulate input to execute arbitrary system commands. This type of an attack is called a command injection attack.

What Do You Do When You Have Command Injection

Command Injection attacks are considered extremely dangerous; they essentially allow an attacker to execute commands on a system. The most common thing to do when discovering you have a command injection attack is getting a reverse shell. Reverse shells can be thought of as backdoors. When an attacker creates this reverse shell, the target server acts as a client, executes command sent by an attacker, and sometimes sends the output of the command back to the attacker. An attacker would usually use system resources on the target to create a shell:

- Netcat
- Python
- Bash

http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet - this is a good resource to use different programs to create a reverse shell.

N.B. Reverse shells tend to be extreme(and sometimes destructive). To demonstrate a proof of concept, you can usually execute the following commands:

- id - gives the ID of the machine
- cat /etc/passwd - prints out a list of current users

- Hostname - shows the hostname of the computer

Variants of Command Injection

Like we mentioned above, an application can interact with the underlying system in different ways:

- The application may just ask users to input commands
    - This is the best case scenario and tends to be very rare. All you would have to do is just enter a command and the web application will execute it on the underlying system
- The application filters allowed commands:
    - This tends to be more common than the previous issue. For example, an application may only accept the ping command. You would have to enumerate what commands are allowed and try to use the allowed command to exfiltrate data
- The application takes input to a command:
    - This is even more common. An example is that an application takes input to the ping command(an IP address) and passes this input to the ping command on the backend.
    - If the input isn't encoded or filtered, you can actually uses this to run other commands.
        - The && operator is used with more than one command e.g. *ls && pwd.* The second command only executes if the first command **and** the second command is successful. You can pass an input containing && *other-command* and the backend would successfully execute it if both commands ran successfully.
        - The | command is used to pass output from one command to another and can also be used to execute commands on the server.
        - The ; character also works well. In most shells, it signifies that a command is complete. You can use this to chain commands so if the input is *command.* Then you can provide *command;other-command*.

Where Would You Find Command Injection

In the following places:

- Text boxes that take in input
- Hidden URLs that take input
    - E.g. /execute/*command-name*
    - Or through queries e.g. /location?parameter=command
    - *When using URLs, remember to URL encode the characters that aren't accepted*
- Hidden ports:
    - Some frameworks open debug ports that take in arbitrary commands