Server-side request forget (SSRF)

Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing.

Essentially, it exploits a trust relationship and can sometimes allow you to access unauthorized information.

Basic SSRF against the local server

```
In the request for a stock check, there is a stockapi input that
requests info from a separate website
We can exploit this by trying differnet payloads, the simplest being
http://localhost/admin
This assumes localhost is reserved/usable and there is an admin panel.

When we send the request, it will load the admin panel in the bottom of
the screen.
However, if we try to press delete or take any actions, it will deny us
as we are not longer sending the request from
localhost We can circumvent this by copying the URL the page sends, and
appending it to our payload.

In this case, /admin/dlete?username=carlos

giving us a paylaod of http://localhost/admin/delete?username=carlos

Put this into the stockapi, and it will run the command.

The machine implicitly trusts commands sent from the localhost, if we
enter taht in our regular URL bar it will not work.
```

SSRF attacks against other back-end systems

```
In this example the Stock API requests from a host of 192.168.0.1:8080
We can search through ports to see if there is an admin panel on any of
them, by sending
the request to intruder and imply iterating from 192.168.0.1-255
```

SSRF with blacklist-based input filter

```
Similar to directory travereasl blacklist problems, sometimes
developers will filter for specific values

alternatives to try:
127.1
017700000001
2130706433
url encode letters of admin
Register your own domain that resolves to 127.0.0.1 using spoofed.
burpcollaborator.net
```

SSRF with filter bypass via open redirection vulnerability

```
This allows us to bypass any filter placed on the SSRF vuln location.
When looking for an open redirection vulnerability, we need to locate
anywhere a path is
forwarded as a request. In this example, when pressing the next page
button there is a
path=/fff/ffff section, which points to the possiblity of us being able
to inject our malicious link

However, if we simply place our payload here it will not work. We need
to combine it our stockapi page load to craft
a malicious package.

The request that we will be exploiting is /product/nextProduct?path=
/product/something
we can edit this to point to our own link, /product/nextProduct?
path=http://192.168.0.12:8080/admin/delete?username=carlos

We then place this into the stockapi and run.
```