Blind SSRF exploitation

Blind SSRF exploitation - @bo0om, Wallarm.

- Published: 2020-02-11
- Original: <https://lab.wallarm.com/blind-ssrf-exploitation/>
- Preserved from: https://lab.wallarm.com/blind-ssrf-exploitation/ (live) on 2026-08-10
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Author @bo0om

There is such a thing as SSRF. There's lots of information about it, but here is my quick summary.

Let's say you go to a website, fill out your profile, and get to the “Upload Profile Picture” step. And you have a choice: upload a file or specify a link.

In this case, we are interested in the latter option. If we specify a link to a resource that contains an image, then the web application will:

- Download it
- Check that it is indeed a picture
- Check its size, because it may not fit
- Display the picture to the user (for shrinking)

Well then... If the website does not check where it is trying to download the image from, this is a vulnerability. Moreover, the attack vectors on such a small feature as loading images are so extensive that a night at the bar won't be enough to go through all the options.

[image: SSRF]

I was once asked, "What can I do with an http(s) link only? It's nothing."

Ok, listen.

[image: image]

The easiest option is to try to identify the internal services. If we are talking about the picture, you can try to access the standard paths like favicon.ico, logos, or the icons directory, assuming that Apache is used. By sending requests, we can iterate over the common local addresses, as well as the website's subdomains that work in the internal infrastructure. It's stuff like bamboo, jira, gitlab, and other things used by all companies.

Why do we need this? Because knowledge is power. After all, even blindly, you can leverage various vulnerabilities and exploits. If you know the vendor or version of the web server or the service used, you narrow down the range of applicable attacks. Not necessarily now, but in the future, knowing the technical information about the internal infrastructure will help exploiting other vulnerabilities.

Alright, we are not allowed to enter IP addresses. Now, suppose that we have some important resource inside the infrastructure and its IP address is 192.168.1.1.

First of all, we mentally create a domain to which assign this IP. Let it be my-test-site.com. In real life, you should create subdomains with addresses directing to the IPs we need, but more on that later.

Password Brute Force

Let's imagine that we have a router inside. The /admin/ directory is under Basic auth. By changing the link, we can try guesses for the router's login and password. But here it's actually quite simple, we just form a link as follows

```
https://login:password@my-test-site.com/admin/
```

Thus, the value before the colon is the login, and after is the password. The @ symbol is followed by the domain where this data will be sent to. Keep in mind that this doesn't work with things like social networks where you need to fill out a form. It works only with the Basic authentication: a pop-up window with a 401 response from the server.

If we were lucky and the website would return AT LEAST the response code (200, 401, 503), it would be much easier. Then we could clearly observe the process and see our win:

```
https://admin:password@my-test-site.com/admin/ - 401
https://admin:admin@my-test-site.com/admin/ - 401
https://admin:123456@my-test-site.com/admin/ - 200
```

By sending a dozen or two of such requests, you can try to crack the password for this router. And then you can address the router's DNS updating scenario or even the /reboot.cgi

And if there is no answer or it is always the same?

In this case, timings will come to the rescue.

Timings

Everything takes time. Just like I take your time to read this article, services take time to respond.

The thing is that we can try to access internal resources, while measuring their timings, to answer the question: is there a service or not?

Sending multiple requests you can blindly sort through internal services, ports, and even directories and files, relying on anomalies of responses.

```
302 ms - https://test.company.com
1307 ms - https://db.company.com
5483 ms - https://jira.company.com
1410 ms - https://docs.company.com
1366 ms - https://kafka.company.com
```

The jira subdomain responded much longer than others—most likely it is inside. The difference is noticeable because the web server tried to load the page and then realized that this was not an image. And what matters to us is not “Who answered the longest?”, but “Who answered not like everyone else?” Because the anomaly can be either a delay (for example, if you find a large file or script that takes a long time to execute) or a quick response.

```
1302 ms - https://test.company.com
1307 ms - https://db.company.com
423 ms - https://jira.company.com
1410 ms - https://docs.company.com
1366 ms - https://kafka.company.com
```

In this case, the response says that it is either 401 or a redirect that is not processed by the image parser. Or maybe that the link is accessible, but our web application rejected it before downloading the page completely after checking the first bytes or content-type. Other websites in this example refused to connect (or the web application couldn't resolve the host name).

Checking IP Address

Many websites have introduced verification process to make sure the IP address is not internal. However, the logic may be wrong and sometimes you can still make the web application connect to the internal IP address. For example, a website verification is divided into two logical steps. First, it checks to see if the specified host directs to an external IP. As long as the first check is successful, the website establishes a connection. And if it doesn't cache the IP address from the previous step, you can go for a funny trick.

At the first request of the domain my-test-site.com, reply with an external IP, for example, 123.123.123.123.

And as soon as it gets validated, start to send an internal IP to the same domain.

Emil Lerner introduced a cool service for this: 1u.ms!

The 1u.ms domain answers with the IP addresses you specified.

The format of the domain should be as follows:

`make-%IP%-rebind-%IP-rr.1u.ms`

For example,

`make-123.123.123.123-rebind-127.0.0.1-rr.1u.ms`

will answer the first request with the 123.123.123.123 address, and the second one with 127.0.0.1 (if it follows within 5 seconds).

By the way, the IP address can be written without dots in case you need a subdomain:

`make-123-123-123-123-rebind-127-0-0-1-rr.1u.ms`

By the way, before "make" and after "rr", you can write random words in order to prevent the use of cached data:

`bo0om-make-123-123-123-123-rebind-127-0-0.1-rr-test.1u.ms`

And to view the log, you can use the analogue of tail -f:

```
curl https://1u.ms:8080/log
```

(or the same link in the browser)

Port Scanning Using DNS

In fact, you can try to scan ports by managing DNS records. A little trick will help us here.

Suppose we have a domain my-test-site.com.

Usually, it contains at least one A record for the resource to open.

Let's say our site's IP is 172.217.20.46 (taken from google.com). But we can specify several such entries! Imagine that we created them and our DNS records look like this:

`my-test-site.com 172.217.20.46 my-test-site.com 192.168.1.1`

Will our resource open? Yes, it will. And all because the first record goes first.

Now let us change the DNS records like this:

`my-test-site.com 192.168.1.1 my-test-site.com 172.217.20.46`

Will our resource open? And again, it will

And all because the resource will initially try to load from the first IP specified. Only if there are any problems with it, it will go to the second one.

`curl "my-test-site.com" -v

- Trying 192.168.1.1...
- TCP_NODELAY set

- Immediate connect fail for 192.168.1.1: Host is down
- Trying 172.217.20.46...
- TCP_NODELAY set
- Connected to my-test-site.com (172.217.20.46) port 80 (#0)

> GET / HTTP/1.1 Host: my-test-site.com User-Agent: curl/7.64.1 Accept: /

< HTTP/1.1 404 Not Found < Content-Type: text/html; charset=UTF-8 < Referrer-Policy: no-referrer < Content-Length: 1561 < Date: Tue, 21 Jan 2020 16:35:08 GMT`

Using this feature, you can find out which ports are open and which are not. Indeed, many (but not all) libraries will try to start with the first resource—listed first in the DNS records—and then to the second one.

Here we have indicated the link to our resource in the "Profile Picture Upload" field.

By specifying our domain and changing the port, we try to use the exception method to determine which port is available.

```
https://my-test-site.com:22 - request came for 172.217.20.46:22
https://my-test-site.com:80 - request came for 172.217.20.46:80
https://my-test-site.com:8080 - request did not come
https://my-test-site.com:9200 - request came for 172.217.20.46:9200
https://my-test-site.com:3306 - request did not come
```

Since we specified 192.168.1.1 as the first record, we conclude that this address answered the library at 192.168.1.1 (ports 8080 and 3306). Even if the answer was incorrect, it was still the answer. So these ports are open and there are services on them. In this case, the library will not switch to the second address.

1u.ms service can also help here, in this case we will have the following config:

`make-%IP%-and-%IP%rr.1u.ms`

or, for our example,

`make-192.168.1.1-and-172.217.20.46rr.1u.ms`

It will return two entries; all other features are the same as in the previous paragraph.

By the way, this approach can be used for faster DNS rebinding, forcing the browser to switch from a “hung” server to a working one. I think this will be faster than waiting a minute to update the DNS cache. However, this trick will not work with Chrome, for example, since it takes a random IP.

Another issue is that the DNS server used to resolve the domain addresses can use round robin mechanism and change the order of records, thereby evenly distributing the load on all servers. For example, 8.8.8.8 disabled this feature, but it is used at 1.1.1.1.

Redirects

Try redirects! Well, firstly, redirects can reduce the number of requests to a web application, for example, when scanning ports using DNS. If the answer reaches you, redirect requests to another port or domain. If not, maybe it stumbled on an open port.

But the best thing, of course, is to try changing the protocol using redirection. In my practice, there were cases when a redirect to file:///etc/passwd worked and showed the file's contents. Working with blind ssrf, you can try changing the protocol to gopher (while it still exists), and you will be able to send emails using commands to SMTP and do other magic.

Denial of Service

Will the loader hang up if I feed it a file of 10 GB in size? Or a 225,000 x 225,000 pixel image, occupying 141.4 GB of RAM. This may affect the website’s performance. However, a crashed server will not give us any fun, so just keep that in mind.

And a Whole Bunch of Everything

This is probably all I can tell you now. This is not taking into account the vulnerabilities associated with the upload (where it is loaded to, how it is saved, what is checked during the process) and 3rd-party things (recall the imagetragick and gifoeb).

I cannot help but give you the SSRF Bible that describes most of the cases on this attack. And there is also a page with SSRF on well-known repository PayloadsAllTheThings, which is actively supported by the community.

By the way, many attacks are applicable to both the server and the client side.