ServerSide Vulnerabilities

PATH TRAVERSAL

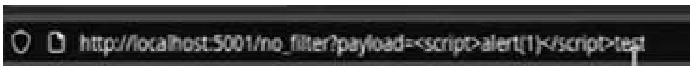
- Going to a place that was unintended
- When you can go outside the bounds of allowed files
- Best example is to include from server's root directory not from the applications
- .. means go up a directory
- ./ means what is the current directory
- You put in as many ../../../../../../../ as you like because eventually it will get you the root directory because the parent of the root directory is the root directory
- On its own it's not that useful because it records the commands on the main laptop and because we are just logged in as a regular user we don't have read write permissions
- This is a good way of finding out what is on the server
- We do know one thing that would be on a web server and that would be: app.py (if we know that its running a flask application because standard practice is to name it app.py
    - Will give you the source code for the application

LFI (local file inclusion)

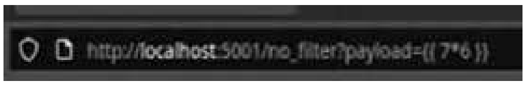
- There are 2 types here
    - Disclosure - reading the code
    - Inclusion - executing the code
- When you include files that aren't meant to be included or are not meant to be included at that time
- Best example is PHP because it will load anything
- Can be done across all languages
- If path traversal is being filtered (that is ../../../../../) is not appearing on the path traversal then you can try ….//….//
    - You start with what you want and just paste paste paste paste and then when you feel like that's enough put in /etc/passwd
    - This would get up the etc/passwd file for you
    - Is another local file disclosure here
- But if we were to include this in a php running web server then we can:
    - When there is no where to upload files but we want to have something run we can use log files: /var/log/apache2/access.log or /var/log/apache2/access.log
    - This will get a log file
    - Now that we have access to this we can edit it because everytime we make a request it logs that request in these logs
        - It logs it as URL encoding if you do it just from the url header
    - So in burp - send to repeater
    - And get rid of the user agent and here, we can put in a php payload <? Php phpinfo() ?> as a test
        - If they see that phpinfo file come up then we have remote code execution that works

        - So with this we can run shell commands in php
        - Can do this by going <?php system(‘whoami) ?>
    - Making this easier
        - <?php system($_GET[‘cmd’]) ?>
        - And then in your URL you can then add &cmd=ls
        - Revshells - a site that can help with reverse traversal commands you can use when doing these file commands
    - If the the mime type is not php it will not take and execute this stuff

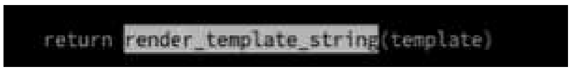
INSECURE FILE UPLOAD
- When you have a file upload vulnerability the first thing you want to do is check which file types that it does accept and then figure out how to bypass those exceptions so that we can upload what we want to upload
    - Check for what you are able to upload - if you can upload and then navigate to, and it's being included when you render it then you have basically gotten through

SSTI (template injection)
- First test that there is something there that you can inject into
- So can inject some html to it <h1>test</h1> - this will bold the input
- 

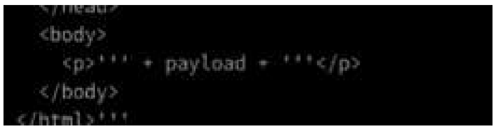
  ```
  http://localhost:5001/no_filter?payload=<script>alert(1)</script>test
  ```

  - this is reflected xss
    - Impact is low but the likelihood of doing this is that someone will click this link and some bad things may happen
- Once we have confirmed that we are able to inject things we can try and inject some {} if we know that or want to check that it is running flask
- 

  ```
  http://localhost:5001/no_filter?payload={{ 7*6 }}
  ```

  - so here it's doing math which means that jinja (the templating format for flask)
    - So what can we do with this?
        - {{ and {% %}
    - Is similar to python where things look like python but is just a template that flask uses so the commands in python may not necessarily work in jinja’s template
- The reason we can do this because
- 

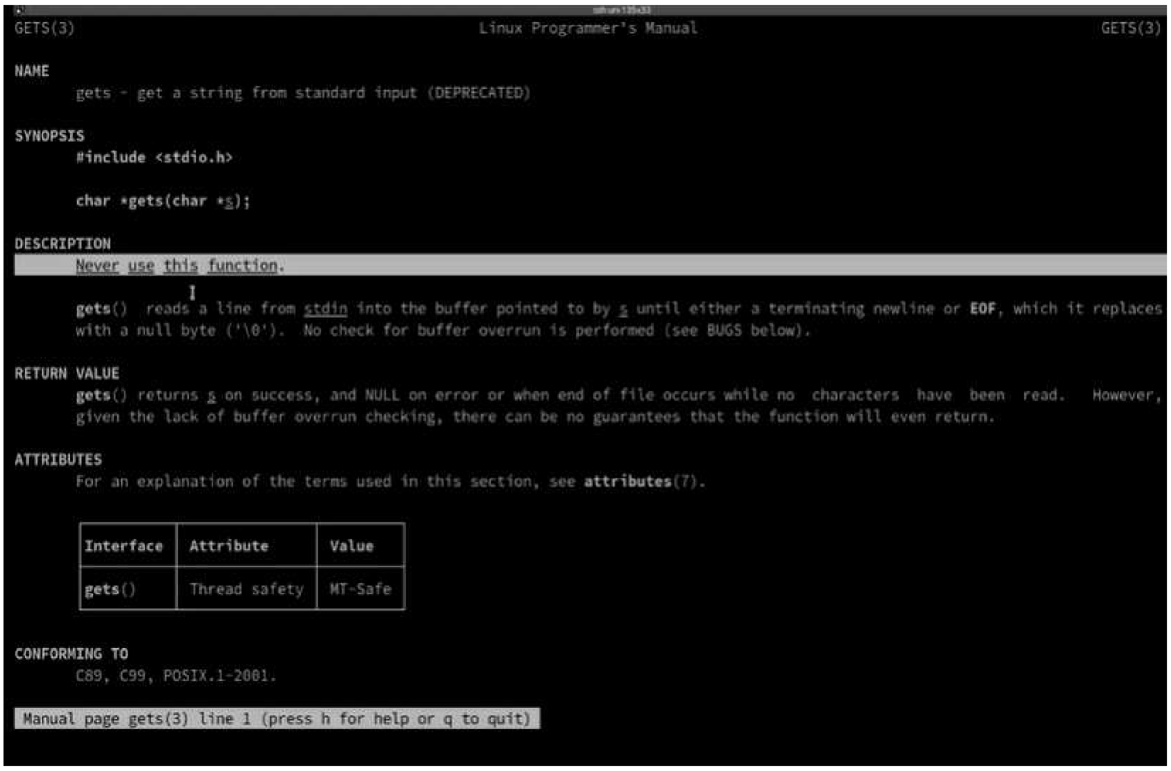
  ```
  return render_template_string(template)
  ```

    - Render_template would solve this
    - So instead of render_template_string just use render_template in your code
- 

  ```
  </head>
  <body>
    <p>''' + payload + '''</p>
  </body>
  </html>'''
  ```

- And this one within the same file that may contribute to the issue
    - The render template one will just render whatever we give it as a string form and this results in the output that we see when we format a payload based on the features of the template we have found

- Some docs are really good at helping us with not making dumb mistakes



```
GETS(3)                          Linux Programmer's Manual                          GETS(3)

NAME
       gets - get a string from standard input (DEPRECATED)

SYNOPSIS
       #include <stdio.h>

       char *gets(char *s);

DESCRIPTION
       Never use this function.

       gets()  reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF, which it replaces
       with a null byte ('\0').  No check for buffer overrun is performed (see BUGS below).

RETURN VALUE
       gets() returns s on success, and NULL on error or when end of file occurs while no  characters  have  been  read.   However,
       given the lack of buffer overrun checking, there can be no guarantees that the function will even return.

ATTRIBUTES
       For an explanation of the terms used in this section, see attributes(7).

        ┌──────────┬───────────────┬─────────┐
        │Interface │ Attribute     │ Value   │
        ├──────────┼───────────────┼─────────┤
        │gets()    │ Thread safety │ MT-Safe │
        └──────────┴───────────────┴─────────┘

CONFORMING TO
       C89, C99, POSIX.1-2001.

 Manual page gets(3) line 1 (press h for help or q to quit)
```

Requestbin - can create an application that lists out all the requests that get made to it
- So if you have url post insertions for upload you can do it this way
- So you can now make the server make requests on the server side