
# MÁSTER EN ANÁLISIS DE MALWARE, REVERSING Y BUG HUNTING

## MODULO 10. REVERSING DE SISTEMAS WINDOWS

## M10. TAREA COLABORATIVA.

**Autor**: IVÁN MERINO MESA

## *Introducción*

En este documento se pretende ejemplificar cómo el diseño vulnerable de un *driver* puede llevar a la escalada de privilegios en un equipo y las consecuencias que tienen este tipo de explotaciones por parte de un actor de amenazas.

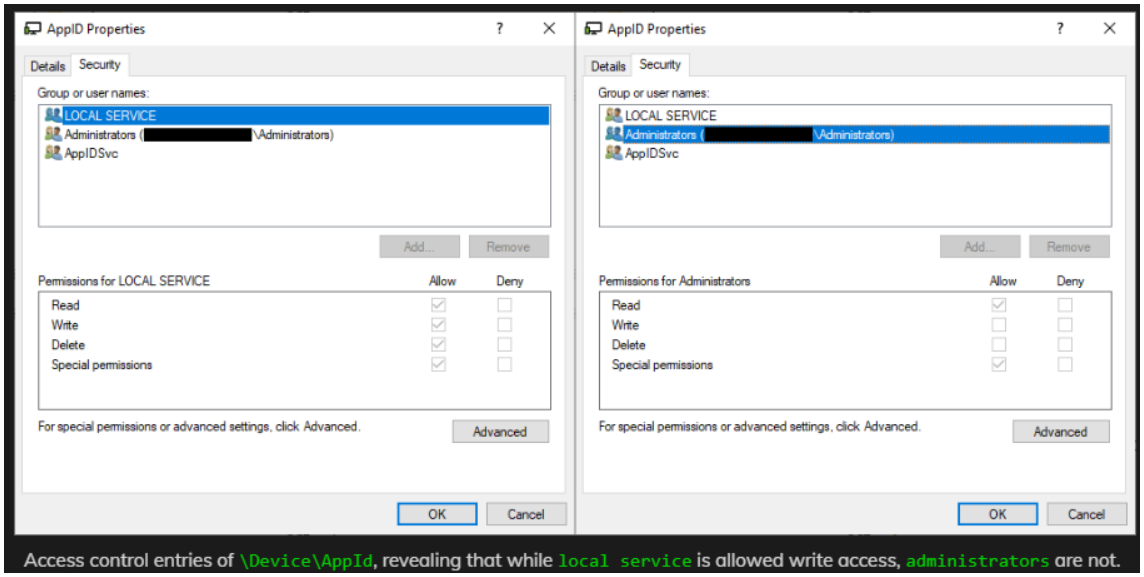
En este caso, ha sido escogido el Rootkit FudModule, que ha sido relacionado con el APT Lazarus. Este grupo se encuentra vinculado geopolíticamente con Corea del Norte y se relaciona con una gran cantidad de ataques sofisticados sobre organismos de todos los sectores.

## *Análisis*

Según se afirma en fuentes de inteligencia pública, pese a que el actor Lazarus acostumbra a utilizar la técnica BYOVD, en uno de los últimos incidentes estudiados por e equipo de Avast, se ha encontrado la explotación de la vulnerabilidad admin-to-kernel zero -day CVE-2024-213338 relacionada con el driver de AppLocker appid.sys. Cómo es obvio, el aprovechamiento de una vulnerabilidad Zero-day en contraposición de BYOVD ocasiona que la dificultad de detección de este tipo de ataques aumente.

Esta explotación conlleva la comunicación directa (read/write) con el kernel que ha sido aprovechada para la instalación de una nueva variante del rootkit FudModule. Una de las mejoras introducidas en este rootkit frente a veriones previas, es la posibilidad de deshabilitar algunas medidas de protección como son Microsoft Defender, CrowdStrike Falcon y Sophos Además, mediante técnicas DKOM se desactivan algunos mecanismos de defensa del kernel.

La vulnerabilidad del driver radica principalmente en la forma en la gestión de los IOCTL dentro del objeto \Device\AppId, terminando en la ejecución arbitraria de código con permisos de administrador. En este caso particular, es necesario en caso de desear realizar acciones de escritura, realizarlas bajo los permisos de LOCAL SERVICE, ya que Administrators sólo posee permisos de lectura.



Asimismo, además del proceso de explotación disponible en el siguiente enlace, Avast ofrece el código facilitado a Microsoft para que pueda parchear y gestionar de manera adecuada el manejo de IOCTL por este driver.

```
    case 0x22A018u:
      if ( WPP_GLOBAL_Control != &WPP_GLOBAL_Control && (HIDWORD(WPP_GLOBAL_Control->Timer) & 2) != 0 )
        WPP_SF_(WPP_GLOBAL_Control->AttachedDevice, 27u, &WPP_Traceguids);
      if ( Feature_2959575357__private_IsEnabled() && ExGetPreviousMode() )
        goto invalid_device_request;
      if ( CurrentStackLocation->Parameters.DeviceIoControl.InputBufferLength == 32 )
      {
        status = AipSmartHashImageFile(a2->AssociatedIrp.SystemBuffer, 0i64, 0i64, 0i64);
ret_status:
        ret_status = status;
        break;
      }
status_invalid_parameter:
      ret_status = STATUS_INVALID_PARAMETER;
      break;
```

The patched IOCTL handler. If feature 2959575357 is enabled, attempts to call the IOCTL with PreviousMode==UserMode should immediately result in STATUS_INVALID_DEVICE_REQUEST, failing to even reach AipSmartHashImageFile.

Para más información acerca del comportamiento observado en esta campaña, se recomienda observar el enlace adjunto en el párrafo anterior.

## *Análisis y erradicación*

Poniendo el enfoque en el lado más defensivo de la ciberseguridad, se han buscado algunos recursos acerca de cómo defenderse de este tipo de ataques sobre *drivers* vulnerables.

Lo más parecido que se ha encontrado, además de los múltiples mecanismos implementados Microsoft para la protección de su Sistema Operativo y que deben estar habilitados, es un estudio de la comunidad de Microsoft, en el que se realizan algunas *queries* de *Threat Hunting* para la búsqueda proactiva de este tipo de amenazas. Cabe destacar que este estudio habla de consultas a realizar sobre la plataforma *Microsoft Defender ATP*. Algunas.

- Búsqueda avanzada para el descubrimiento de drivers (ficheros .sys) dentro de una infraestructura, extrayendo información acerca del propio servicio.

```
DeviceFileEvents
| where ActionType == "FileCreated"
| where FileName endswith ".sys"
| invoke FileProfile(SHA1,10000)
| where GlobalPrevalence <= 500
| join kind=leftouter
    (
    DeviceFileCertificateInfo ) on SHA1
| project FileName, FolderPath, GlobalPrevalence, GlobalFirstSeen,
GlobalLastSeen,
Signer, Signer1, Issuer, Issuer1, CertificateCreationTime,
CertificateExpirationTime, CertificateSerialNumber
```

- Búsqueda de servicios a nivel de kernel instalados en el equipo.

```
DeviceEvents
| where ActionType contains "ServiceInstalled"
| extend ParsedFields=parse_json(AdditionalFields)
| extend AttributeList = ParsedFields.ServiceType
| where AttributeList == 1
```

- Búsqueda de *drivers* vulnerables basada en listas de Microsoft, así como recursos de la comunidad.

```
# creating a database of files from the OpenSource list
let LolDriverSHA1 = externaldata(SHA1:
string)[@"https://raw.githubusercontent.com/magicsword-
io/LOLDrivers/main/detections/hashes/samples.sha1"] with (format="txt",
ignoreFirstRecord=False);


# creating a database of files from the Microsoft Vulnerable Driver List
let indicatorsFromMsft = materialize(
            externaldata(data:string)["https://learn.microsoft.com/en-
us/windows/security/application-security/application-control/windows-
defender-application-control/design/microsoft-recommended-driver-block-
rules#vulnerable-driver-blocklist-xml"] with(format="raw")
            | extend tempExtractedData = extract('(?sU)<pre><code
class="lang-xml">(.*)</code></pre>', 1 , tostring(data))
            | extend fix_xml = replace_strings(tempExtractedData,
                                               dynamic(['&lt;','&gt;',
'&quot;']),
                                               dynamic(['<','>','"'])
                                              )
            | project-away data, tempExtractedData
            | extend parsed_xml = parse_xml(fix_xml)
            | project-away fix_xml
            | mv-apply x=['parsed_xml']['SiPolicy']['FileRules']['Deny'] on
              (
                  extend _FriendlyName = x.['@FriendlyName']
                  | extend _Hash = x.['@Hash']
                  | extend _FileName1 = x.['@FileName']
                  | summarize VulnDrivers_Sha1 = make_set_if(_Hash,
_FriendlyName endswith 'Hash Sha1'),
                              VulnDrivers_Sha256 = make_set_if(_Hash,
_FriendlyName endswith 'Hash Sha256'),
                              VulnDrivers_FileNames1 = make_set(_FileName1)
```

```
                )
            | mv-apply
y=['parsed_xml']['SiPolicy']['FileRules']['FileAttrib'] on
                (
                    extend _FileName2 = y.['@FileName']
                    | summarize VulnDrivers_FileNames2 = make_set(_FileName2)
                )
            | extend VulnDrivers_FileNames =
array_sort_asc(array_concat(VulnDrivers_FileNames1, VulnDrivers_FileNames2))
            | project-away VulnDrivers_FileNames1, VulnDrivers_FileNames2,
parsed_xml
      );
      //let VulnDrivers_FileNames = toscalar(indicatorsFromMsft | project
VulnDrivers_FileNames);
      let VulnDrivers_Sha1 = toscalar(indicatorsFromMsft | project
VulnDrivers_Sha1);
      //let VulnDrivers_Sha256 = toscalar(indicatorsFromMsft | project
VulnDrivers_Sha256);
   DeviceFileEvents | where SHA1 in~ (VulnDrivers_Sha1) or SHA1 in~
(LolDriverSHA1)
```

Asimismo, otra de las ideas encontradas en la comunidad de cara a defenderse de ataques BYOVD, es modificar el concepto de lista negra con *drivers* vulnerables, pasando a aplicarse una lista blanca de aquellos controladores que sí pueden ser introducidos en el sistema (*whitelisting*). Esta tecnología se llama WDAC y fue introducida por primera vez en Windows 10. Más información al respecto puede consultarse en:

*https://github.com/HotCakeX/Harden-Windows-Security/wiki/WDAC-policy-for-BYOVD-Kernel-mode-only-protection*

## *Bibliografía*


https://securityaffairs.com/159728/apt/lazarus-exploited-zero-day-windows-applocker-driver.html

https://msrc.microsoft.com/update-guide/vulnerability/CVE-2024-21338

https://decoded.avast.io/janvojtesek/lazarus-and-the-fudmodule-rootkit-beyond-byovd-with-an-admin-to-kernel-zero-day/

https://github.com/HotCakeX/Harden-Windows-Security/wiki/WDAC-policy-for-BYOVD-Kernel-mode-only-protection

https://www.virusbulletin.com/uploads/pdf/conference/vb2022/papers/VB2022-Lazarus-and-BYOVD-evil-to-the-Windows-core.pdf

https://techcommunity.microsoft.com/t5/microsoft-security-experts-blog/strategies-to-monitor-and-prevent-vulnerable-driver-attacks/ba-p/4103985