Quand les bleus se prennent pour des chercheurs de vulnérabilités : Recherche des événements ETW

Sylvain Peyrefitte
sylvain.peyrefitte@airbus.com

Airbus

Résumé. Trouver un attaquant déjà établi dans un réseau depuis des mois ou des années n'est pas chose facile ! L'énorme quantité et la variété de *traces* et *artefacts* laissés sur le système d'exploitation effraient un grand nombre d'analystes forensiques. Face à ces problématiques, ils sont très souvent livrés à eux mêmes, avec leur expérience pour faire face à l'exercice de recherche de compromission d'un SI ou de réponse sur incident. Ainsi, l'objectif de cet article est de présenter les méthodes et les processus techniques (inspirés de cas réels) permettant à une équipe de réponse sur incident d'instrumenter le système Windows afin de détecter un comportement malveillant. Pour rendre le problème plus concret, nous prendrons pour exemple une chaîne d'exploitation existante (exploit *Bluekeep*), en explicitant des moyens concrets et les outils qu'un membre d'une équipe bleue peut mettre en place pour analyser, *disséquer* et générer des événements systèmes de Windows (les ETW), dans le but de détecter des comportements ou binaires suspects. Ce processus fait appel à des notions et outils proches de ceux utilisés par les chercheurs de vulnérabilités.

1 Introduction

Dans le monde de la sécurité, on oppose souvent, à tort, le travail des équipes bleues (de défense) et des équipes rouges (d'attaque).

On résume le travail des bleus à une simple analyse de logs. Pourtant, avec des systèmes devenant toujours plus complexes, offrant toujours plus d'opportunités aux attaquants, le travail des défenseurs nécessite de plus en plus d'expertise englobant des compétences mixtes.

Longtemps sous-estimé, Microsoft a désormais bien compris la problématique de visibilité, certainement dû au fait que Microsoft offre aujourd'hui des services de détection. Pour preuve, le nombre de sources de log entre Windows 7 et Windows 10, a augmenté de façon exponentielle :

— Windows 7 : 635 providers
— Windows 10 : 1102 providers + WPP + Tracelogging

Maintenant, la difficulté réside dans la recherche du bon chemin dans cet océan d'information. Il est vrai que nous possédons maintenant des SIEM, des outils puissants gérant de grands volumes de données, tel que Splunk.

Les SIEM jouent le rôle de témoin de l'histoire de notre parc. Si nous arrivions à trouver un événement déjà collecté, témoignant d'un comportement malveillant découvert que très récemment, nous pourrions mettre en lumière des attaques persistantes.

Certains événements peuvent être aussi considérés comme le témoignage d'un comportement caractéristique d'un groupe ennemi.

Mais les SIEM ont aussi un coût. Si nous pouvons optimiser notre collecte des événements ciblant précisément un comportement malveillant, cela peut nous permettre de faire des économies.

C'est dans ce contexte que nous avons décidé, dans cet article, de vous exposer un ensemble d'outils de mesure, permettant de trouver un événement, et d'en évaluer sa pertinence en laboratoire.

Pour rendre le problème plus concret et pragmatique, on présentera le processus en trois étapes :

— Analyse des événements malveillants : Comme pendant un pentest réseau ou une analyse de trace réseau (fichier PCAP), on commence par étudier les événements pour détecter un binaire suspect. Pour cela, nous avons développé Winshark (plugins Wireshark pour disséquer les événements systèmes de Windows)
— Remonter à la *source* et l'analyser : Analyse (statique via outils de décompilation (IDA) ou dynamique via debugger) du service, mettant en lumière le comportement malveillant.
— Etre proactif pour aller plus loin dans l'analyse et la détection système : une fois la détection et l'analyse réalisées, à nous de créer nos propres sources d'événements pour aller encore plus loin.

2 Event Tracing for Windows

Depuis Windows XP (2001), Microsoft implémente au coeur de son système d'exploitation un système de log puissant, modulaire et permettant une exploitation en temps réel se nommant ETW (Event Tracing for Windows). Sa modularité passe tout d'abord par une architecture divisée en trois parties :

— Les providers, en charge d'émettre des logs
— Les sessions, combinent et configurent des providers
— Les consumers, lisent les logs depuis une session

Un message est caractérisé par :
— un identifiant d'événement unique à chaque provider
— un niveau d'information (debug, info, warning, error),
— un keyword, concept aussi appelé *canal d'émission*.

Ces méta informations permettent de filtrer les événements provenant d'un provider afin de diminuer la quantité d'information à gérer.

Pour des raisons de sécurité et de stabilité, Microsoft ferme de plus en plus la porte aux développeurs noyau. En contrepartie, ce dernier offre de plus en plus de visibilité tant pour l'espace utilisateur que pour l'espace noyau, au travers justement d'ajout de *providers* ETW.

Un exemple significatif est la capacité de réaliser des captures de paquets réseau, via l'utilisation d'un provider, en espace utilisateur, sans avoir recours à la programmation d'un driver NDIS (Network Driver Interface Specification), ou encore la possibilité de réaliser un logiciel équivalent à Procmon sans avoir à programmer le moindre driver.

D'un point de vue développeur, il existe de nombreux outils permettant de créer facilement des providers ETW :
— TraceLogging API
— Winevt pour la manipulation de Manifest

Pour autant, l'utilisation des providers système et la compréhension de ces derniers se heurtent à :
— Format non documenté (Exemple Tracelogging)
— Outils lents ou obsolètes
— Définition peu explicite

Microsoft publie de nombreux outils autour des ETW, orientés essentiellement autour de l'analyse de performance système comme Windows Performance Analyzer, ou bien pour le monde .NET perfview, qui se base sur des providers disponibles dans la Common Language Runtime (clr.dll).

Concernant l'analyse de ces événements, plusieurs outils existent. On peut citer Windows Message Analyzer qui est certainement l'outil le plus complet. Cependant,ce dernier n'est plus supporté et n'est plus disponible en téléchargement depuis la fin de l'année 2019.

3 Analyse des événements

Pour rendre le problème plus concret et pragmatique, nous allons tenter de détecter l'exécution de l'exploit Bluekeep (CVE-2019-0708) sur Windows 10. Initialement, nous savons juste que la vulnérabilité et l'exploit Bluekeep se basent sur le mélange que l'implémentation RDP de Windows fait entre les canaux statiques et dynamiques, et un en particulier : le

canal statique ms_t120. Pour cela, l'exploit crée un canal dynamique se nommant ainsi pour déclencher le bug.

Nous allons, dans un laboratoire, tenter d'analyser l'ensemble des traces que produit le service en charge du RDP sur un poste Windows. Nous avons donc besoin, pour cela, d'un outils de capture, ou consumers dans la terminologie ETW.

Par défaut, Microsoft propose un certain nombre de consumers.

— En premier lieu, l'Event Log lui-même. Mais ce dernier est difficilement exploitable en pratique car sa configuration est complexe et il introduit une latence dans la capture, ce qui le rend difficilement exploitable pour une analyse temps réel.
— On peut aussi utiliser Logman (outil de performance/débogage proposé par Windows). Ce dernier permet de créer des sessions, et comme il est en ligne de commande, il peut être facilement scriptable. De plus il permet de réaliser de l'introspection par processus, c'est-à-dire lister les providers disponibles pour un PID donné. Malgré ces atouts, il possède une importante limitation : l'ensemble des résultats est uniquement capturé dans un fichier difficilement exploitable et surtout peu documenté !
— Enfin nous pouvons citer Microsoft Message Analyzer. Ce dernier est complet, car il permet de configurer précisément nos sessions, et de réaliser une capture temps réel de nos logs. Malheureusement son interface complexe et sa lenteur le rendent quasiment inexploitable pour des captures en temps réel. Enfin, les dissectors sont écrits en langage OPN (Open Protocol Notation), format peu documenté et complexe. Microsoft fournit de nombreuses implémentations pour des protocoles et structures propriétaires. De plus, Microsoft Message Analyzer a récemment été abandonné par Microsoft et n'est plus publié par ce dernier.

Afin de pallier aux problèmes ci-dessus, nous avons eu l'idée d'utiliser l'outil de prédilection pour analyser les traces : Wireshark. Nous avons étudié la possibilité de capturer et analyser des ETW directement depuis ce dernier. Son architecture interne étant très modulaire, nous avons pu développer aisément un ensemble de plugins nommé Winshark. Ce dernier repose sur l'implémentation d'un backend pour la libpcap supportant la capture d'événements ETW. Nous avons écrit un programme convertissant automatiquement les manifests ETW en dissector Lua. Nous avons aussi ajouté quelques dissectors un peu spécifique, afin de prendre en charge les événements Tracelogging, et les ETW émis depuis le provider NDIS

(capture réseau). Un schéma indiquant les différentes briques logicielles est présenté ci-dessous.

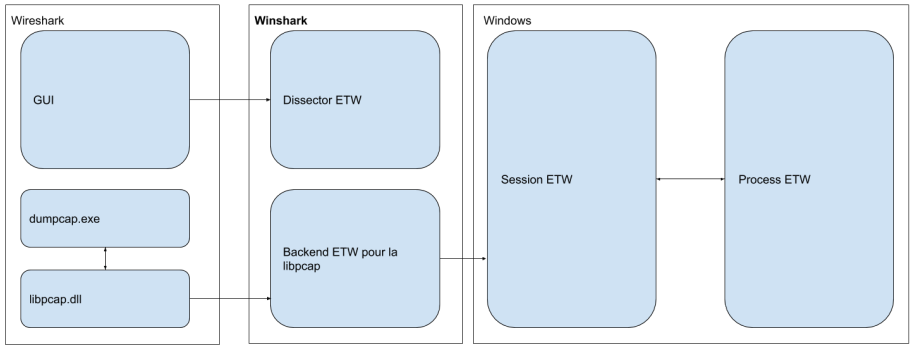


Fig. 1. Architecture de Winshark

Pour mieux illustrer l'intérêt de Winshark dans un cas réel (détection de l'exploitation de la vulnérabilité Bluekeep), nous vous proposons de présenter celui-ci dans un environnement Windows de test *grandeur nature*, comme DetectionLab.

Une fois DetectionLab lancé, nous allons dans un premier temps identifier le service vulnérable (RDP) et lister ses sources d'événements disponibles :

```
logman query providers -pid 123
```

Listing 1. Lister les providers enregistrés par le process 123

Ensuite, nous pouvons créer une session regroupant l'ensemble des providers émis par ce processus, toujours via logman :

```
logman start RDP -p "Microsoft-Windows-Remotedesktopservices-
    rdpcorets" -ets -rt
logman update RDP -p "Microsoft-Windows-NDIS-PacketCapture" -ets -rt
```

Listing 2. Créer une session regroupant deux providers

Et enfin grâce à Winshark, nous allons pouvoir facilement et simplement visualiser notre session comportant l'ensemble des providers RDP, ainsi que le provider en charge des traces réseau émises sur l'interface réseau, le tout en temps réel.

Winshark nous permet facilement d'identifier un événement particulier :

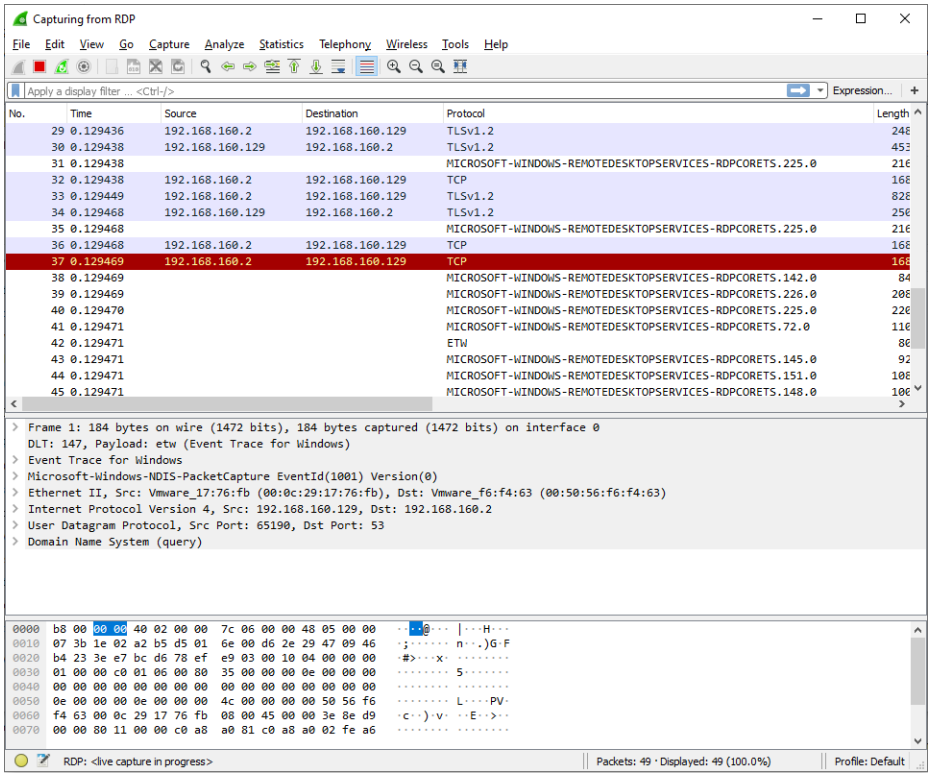


Fig. 2. Exemple de vue mixant ETW et capture réseau

— Il provient du provider Microsoft-Windows-RemoteDesktopServices-RdpCoreTS
— Il possède l'identifiant 148
— il possède un champ se nommant ChannelName possédant la valeur ms_t120

Nous avons pu mettre en évidence cet événement en comparant deux captures ; une provenant d'une session légitime et une seconde provenant du script d'exploiation de Bluekeep.

Pour nous assurer que cet événement caractérise bien Bluekeep, nous allons essayer de comprendre dans quel cas il est émis.

4 Remonter à la source de l'événement

Une fois l'événement suspect identifié, il faut comprendre pourquoi et comment cet événement est émis par le service Terminal Server, afin de savoir si ce dernier est pertinent.

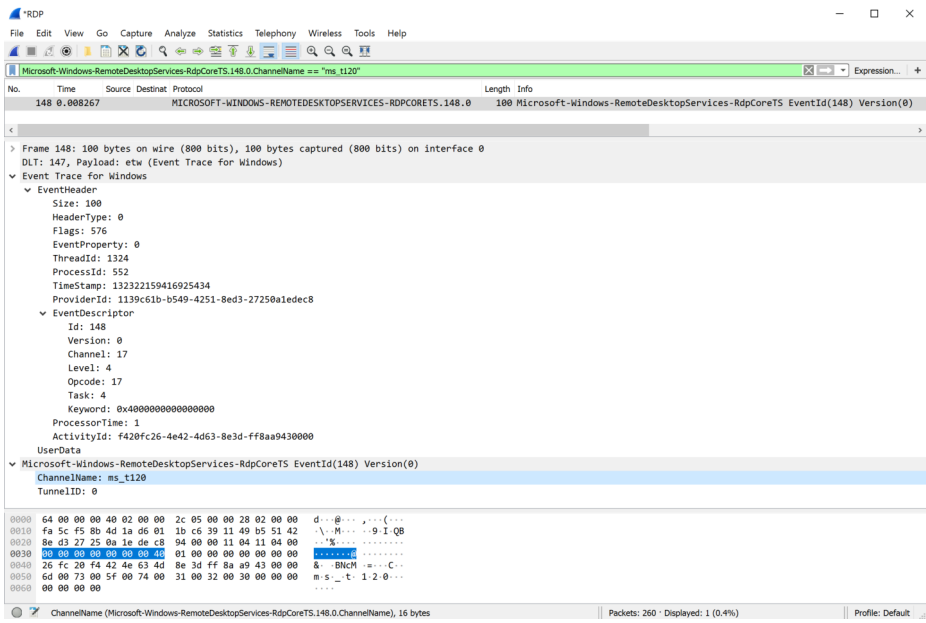


Fig. 3. Mise en évidence de l'événement

Pour cela nous allons commencer par réaliser une analyse statique de la DLL en charge de l'événement.

Il est très facile de réaliser le lien entre le module et un identifiant de provider, dans la mesure où ce dernier provient d'un enregistrement réalisé par winevt. Ce dernier maintient ce lien via une clé de registre :

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers
```

Ensuite, le schéma des événements est inclus dans une ressource du PE nommé WEVT_TEMPLATE.

Nous avons donc réalisé un plugin IDA, ETWBreaker, permettant de parcourir les ressources WEVT_TEMPLATE, les analyser, et ainsi pouvoir lister l'ensemble des événements pouvant être émis par un module.

Nous allons analyser le module rdpcorets.dll en charge du provider Microsoft-Windows-RemoteDesktopServices-RdpCoreTS.

ETWBreaker réalise une recherche de symbole basée sur la signature de l'événement. L'événement 148 est associé au symbole RdpCoreTS_Event_ChannelClose, ce qui constitue un premier indice sur l'utilisation de cet événement.

En plus des événements basés sur un manifest, Microsoft a généralisé l'utilisation de Tracelogging. Tracelogging est une bibliothèque, basée sur

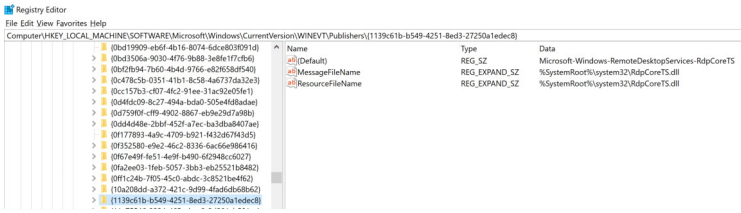
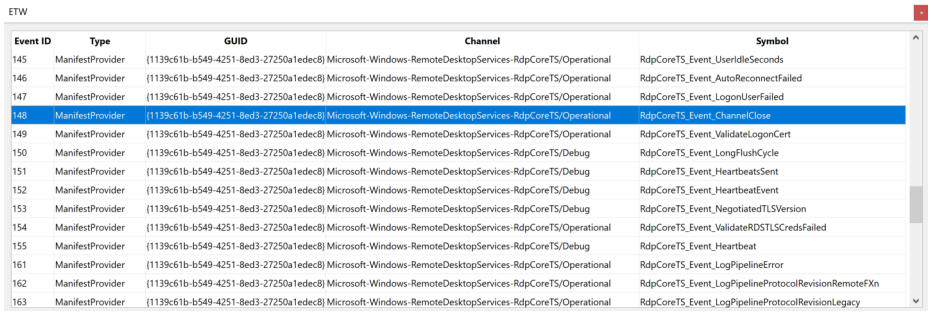


Fig. 4. Recherche d'un module en charge d'un provider



Event ID	Type	GUID	Channel	Symbol
145	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Operational	RdpCoreTS_Event_UserIdleSeconds
146	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Operational	RdpCoreTS_Event_AutoReconnectFailed
147	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Operational	RdpCoreTS_Event_LogonUserFailed
148	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Operational	RdpCoreTS_Event_ChannelClose
149	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Operational	RdpCoreTS_Event_ValidateLogonCert
150	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Debug	RdpCoreTS_Event_LongFlushCycle
151	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Debug	RdpCoreTS_Event_HeartbeatsSent
152	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Debug	RdpCoreTS_Event_HeartbeatEvent
153	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Debug	RdpCoreTS_Event_NegotiatedTLSVersion
154	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Operational	RdpCoreTS_Event_ValidateRDSTLSCredsFailed
155	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Debug	RdpCoreTS_Event_Heartbeat
161	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Operational	RdpCoreTS_Event_LogPipelineError
162	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Operational	RdpCoreTS_Event_LogPipelineProtocolRevisionRemoteFXn
163	ManifestProvider	{1139c61b-b549-4251-8ed3-27250a1edec8}	Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Operational	RdpCoreTS_Event_LogPipelineProtocolRevisionLegacy

Fig. 5. Listing des événements disponibles dans ce module

des macros complexes, reposant sur ETW. Le schéma de ces événements n'est pas publié mais inclus directement dans les méta informations de l'événement. Ce schéma étant statique, il est écrit dans un des segments .rdata. Afin d'empêcher l'exploitation de Tracelogging comme source potentielle de fuite d'information, une vérification est faite en conçervant une trace de l'adresse de début et de fin du segment utilisé pour sauvegarder les données de Tracelogging. Ce qui implique que les schémas sont contigue en mémoire, et va simplifier la recherche des providers Tracelogging utilisés par un module.

Nous continuons notre recherche par l'analyse dynamique du module responsable de l'événement.

Dans notre exemple de BlueKeep, nous voulons identifier quand l'événement 148 provenant du provider Microsoft-Windows-RemoteDesktopServices-RdpCoreTS est émis.

Pour envoyer un message, les providers ETW utilisent la fonction EventWrite disponible via l'APISET api-ms-win-eventing-provider-l1-1-0.dll. Cette dernière est redirigée par le schéma apisetschema.dll vers la bibliothèque système Advapi32.dll. Enfin, Advapi32.dll utilise une fonction du PE qui permet de créer un export redirigé vers une autre dll, et de faire correspondre l'export EventWrite vers l'export EtwEventWrite de

la bibliothèque système ntdll.dll. C'est donc bien cette dernière qui sera notre point d'arrêt.

La signature de EtwEventWrite est la suivante :

```
ULONG EtwEventWrite( REGHANDLE RegHandle,  PCEVENT\textunderscore
    DESCRIPTOR EventDescriptor, ULONG UserDataCount, PEVENT\
    textunderscore DATA\textunderscore DESCRIPTOR UserData );
```

Listing 3. Signature de la fonction EtwEventWrite

EVENT_DESCRIPTOR est une structure bien connue, et elle est présente dans les symboles publics de Windows. Elle nous permet d'ajouter l'identifiant de l'événement comme condition dynamique de notre point d'arrêt en utilisant la valeur du registre rdx. Mais ceci n'est pas suffisant car nous allons alors nous arrêter sur tous les événements correspondant à l'identifiant 148 sans connaissance de son provider. Pour cela, nous pouvons utiliser le premier paramètre, RegHandle, présent dans le regitre rcx. RegHandle est un identifiant opaque, dont les 12 derniers octets correspondent à l'adresse d'une structure mémoire pouvant nous renseigner sur le provider. Cette structure étant en espace utilisateur, son adresse ne pourra jamais être plus grande. A l'offset 0x20 de cette dernière, nous retrouvons le GUID de notre provider, et nous pouvons donc placer notre point d'arrêt en lui ajoutant des conditions sur la valeur de l'identifiant de l'événement ainsi que sur la source de ce dernier. Dans le cas qui nous intéresse, nous voulons filtrer les événements 148 (0x94) sur le provider Microsoft-Windows-RemoteDesktopServices-RdpCoreTS qui possède le GUID 1139C61B-B549-4251-8ED3-27250A1EDEC8. Ce qui peut s'exprimer en utilisant WinDBG :

```
bp ntdll!EtwEventWrite ".if (@@c++((*(_EVENT_DESCRIPTOR*)@rdx).Id)
    ==94 && (*(_GUID*)((@rcx & 0xFFFFFFFFFFFF) + 0x20)).Data1 ==
    1139C61B &&
(*(_GUID*)((@rcx & 0xFFFFFFFFFFFF) + 0x20)).Data2 == B549 &&
(*(_GUID*)((@rcx & 0xFFFFFFFFFFFF) + 0x20)).Data3 == 4251 &&
(*(_GUID*)((@rcx & 0xFFFFFFFFFFFF) + 0x20)).Data4[0] == 8E &&
(*(_GUID*)((@rcx & 0xFFFFFFFFFFFF) + 0x20)).Data4[1] == D3 &&
(*(_GUID*)((@rcx & 0xFFFFFFFFFFFF) + 0x20)).Data4[2] == 27 &&
(*(_GUID*)((@rcx & 0xFFFFFFFFFFFF) + 0x20)).Data4[3] == 25 &&
(*(_GUID*)((@rcx & 0xFFFFFFFFFFFF) + 0x20)).Data4[4] == 0A &&
(*(_GUID*)((@rcx & 0xFFFFFFFFFFFF) + 0x20)).Data4[5] == 1E &&
(*(_GUID*)((@rcx & 0xFFFFFFFFFFFF) + 0x20)).Data4[6] == DE &&
(*(_GUID*)((@rcx & 0xFFFFFFFFFFFF) + 0x20)).Data4[7] == C8) {} .else
     {gc}"
```

Listing 4. Point d'arrêt conditionnel

Cette condition est automatiquement générée par notre plugin IDA, et donc disponible autant pour les événements basés sur un manifest, que les

événement émis depuis l'API Tracelogging. Il suffit donc de selectionner l'événement dans la liste en double cliquant, et de s'attacher au processus en charge du service RDP.

Une fois le point d'arrêt atteint, nous pouvons identifier très rapidement le but de cet événement simplement en regardant la pile d'appel et grâce aux symboles fournis par Microsoft :

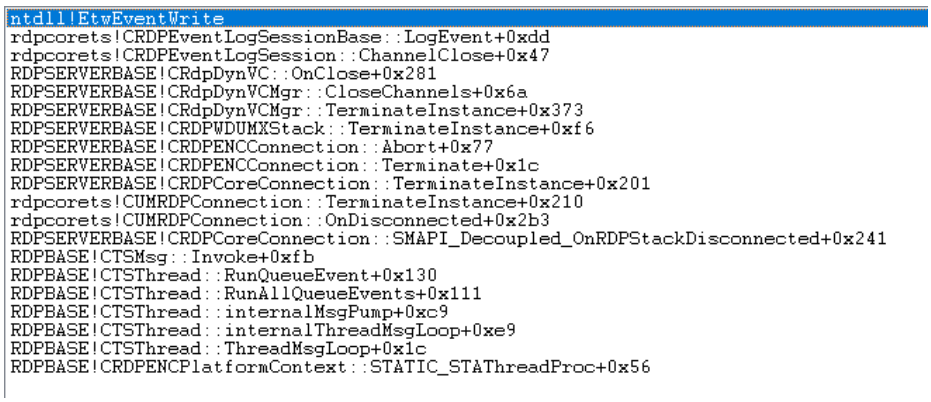


Fig. 6. Pile d'appel une fois le point d'arrêt atteint

On devine ainsi que cet événement se situe dans la gestion des canaux dynamiques (CRdpDynVC : Dynamic Virtual Channel), plus précisément lors de la fermeture d'un de ces canaux. Nous savons que ms_t120 ne doit pas être un canal dynamique, et qu'il ne doit jamais être fermé. On peut alors être sûr que la surveillance de cet événement pourra être utilisée pour détecter une exploitation de Bluekeep.

Nous avons trouvé un évènement dont la présence avec une certaine valeur, suffit à prouver, avec un degré suffisant, une tentative d'exploitation d'une vulnérabilité. Si ce provider faisait déjà partie de notre collecte, nous pouvons donc vérifier dans l'historique de notre SIEM la présence d'une trace d'exploitation. Ceci peut aussi s'avérer un moyen de créer un cas d'usage nous permettant de nous protéger de futures exploitations.

5 Être proactif : Créer sa propre source d'événement

Une fois la vulnérabilité exploitée, le schéma d'attaque principal consiste d'en la pérennisation de l'accès. Pour ce faire, les attaquants utilisent de plus en plus d'outils « légitimes », présents par défaut sur le système Windows. Ils vont s'employer à utiliser des outils d'administration

tels que Powershell, Visual Basic Script, Javascript etc. dans le but d'être le plus discret possible. Sauf pour Powershell (qui possède des logs), les autres outils cités ci-dessus laissent très peu de traces ou de logs.

Ceci représente un énorme avantage pour les attaquants, qui n'hésitent pas à utiliser des langages de script tels que le Visual Basic ou Javascript pour le déploiement du premier stage d'infection.

Pour pallier à cela, Microsoft nous offre toujours plus de moyens d'instrumenter son système d'exploitation.

Avec Windows 10, Microsoft introduit l'AMSI (Anti Malware Script Interface), permettant de réaliser de l'introspection de scripts, et dans un futur proche d'Assembly .Net, afin que les antivirus arrêtent de s'accrocher n'importe où dans le noyau avec des techniques dignes d'un rootkit. L'AMSI permet de valider chaque script avant exécution. Il est donc possible de réaliser un filtre "blanc" qui permettra d'émettre un ETW contenant le script à exécuter à chaque fois. Microsoft fournit un tel provider dans son dépôt GitHub d'exemple : https ://github.com/microsoft/Windows-classic-samples/tree/master/Samples/AmsiProvider

6 Conclusion

Microsoft nous permet, à nous équipe bleue, d'identifier des événements mettant en évidence le comportement recherché d'équipe rouge ou d'attaquants. Bien qu'il soit nécessaire d'avoir des connaissances en bas niveau du système, et que Microsoft n'ai pas choisi la voie de la documentation, les ETW sont la bonne et la meilleure méthode pour analyser et détecter des événements suspects sur les systèmes Windows. Avec un peu d'outillage et de développement (Winshark : plugins Wireshark pour disséquer les événements système de Windows), on peut facilement remonter au processus source des événements suspects pour l'analyser statiquement ou dynamiquement (avec nos amis IDA et WinDBG).

Une fois l'analyse finie, les ETW nous permettent de créer nous-même nos propres sources d'événements pour aller encore plus loin dans l'analyse et la détection système. Ainsi, on peut continuer d'imaginer des cas d'utilisation toujours plus précis, comme l'instrumentation du Common Language Runtime (CLR) à la mode de perfview. Cela pourra être pertinent dans la détection de malware tel que Covenant qui se repose exclusivement sur la CLR.