



Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Markus Hiltunen

PowerShellin kehitys ja historia

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tutkinto-ohjelman nimi

Insinöörityö

8.4.2019

Tekijä Otsikko Sivumäärä Aika	Markus Hiltunen PowerShellin kehitys ja historia 26 sivua 8.4.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikan koulutusohjelma
Ammatillinen pääaine	Tietoverkot
Ohjaajat	Lehtori Tapio Wikström
<p>Tämän insinöörityön tarkoituksena oli perehtyä PowerShellin historiaan ja avata tarkemmin kehitysten vaiheita sekä tulevaisuuden näkymiä. Ideana oli saada yleiskuva siitä, mitä mahdollisuuksia eri PowerShell-versiot tuovat käyttäjälle ja miten näitä voidaan käyttää helpottamaan järjestelmän hallintaa ja ylläpitoa. Tavoitteena oli myös havainnollistaa PowerShellin käyttöä tekemällä tarkoitukseen sopiva skripti.</p> <p>Työssä perehdyttiin Microsoftin verkkomateriaaliin ja käytiin läpi kaikki keskeisimmät muutokset versioittain. Muutoksia avattiin esimerkein ja korostettiin tärkeimpiä uusien versioiden tuomia ominaisuuksia. PowerShellin toimintaa havainnollistettiin tekemällä skripti, joka hakee erilaisia tarpeellisia tietoja käyttäjän laitteesta ja käy myös läpi erilaisia lokiin kirjatun virheilmoituksia sekä tallettaa ne tiedostoon käyttäjän osoittamaan polkuun jatkoselvitystä varten.</p> <p>Lopputuloksena syntyi kattava kuvaus PowerShellin muutoksista versioittain, jota voidaan käyttää apuna selvittäessä, mitä PowerShellin ominaisuuksia on käytettävissä eri käyttöjärjestelmille ja niiden versioille. Työssä myös vertailtiin eroavaisuuksia PowerShellin toiminnassa eri alustoilla ja annettiin esimerkkejä toiminnallisista eroista Windows PowerShellin ja PowerShell Coren välillä. Tämän lisäksi syntyi valmis skripti, jota voidaan käyttää laitteen ongelmanselvityksessä.</p>	
Avainsanat	Microsoft, PowerShell, Core, NET Framework

Author Title	Markus Hiltunen Development and History of PowerShell
Number of Pages Date	26 pages 8 April 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Communication Networks
Instructors	Tapio Wikström, Senior Lecturer
<p>The idea behind this thesis was to study the history of PowerShell and take a closer look at the different versions of PowerShell and how they have developed over time. The aim was to provide reader with a good overall understanding of what kind of possibilities different versions bring to the user and how one can use them to automatize daily administrative tasks. A PowerShell script was created to demonstrate the ways you can use PowerShell to your advantage.</p> <p>The thesis covers all the major versions of PowerShell in depth and focuses on the most important features each version brings. Some examples are given to help demonstrate these features and how these can be used. The script that was created can be used to gather useful information of the user's computer and find errors that have occurred in the system. These errors are then written into a file location given by the user and can later be used for problem solving.</p> <p>Thesis gives the reader a solid base for determining what kind of features are available for different operating systems and this can be used to help choose what version of the operating system is needed to provide all the necessary tools for the system administrator. The PowerShell's functionalities between Windows and Linux operating systems were also compared and especially the differences between Windows PowerShell and PowerShell Core versions.</p>	
Keywords	Microsoft, PowerShell, Core, NET Framework

Sisällys

Lyhenteet

1	Johdanto	1
2	PowerShellin synty ja käyttöönotto	1
2.1	Komentoliittymät Windows-laitteilla	1
2.1.1	COMMAND.COM	2
2.1.2	Cmd.exe	2
2.2	PowerShellin synty	2
3	PowerShellin kehitys ja versiot	4
3.1	PowerShell-versiot	4
3.1.1	Microsoft .NET Framework	4
3.1.2	Windows Management Framework	4
3.1.3	Vanhemmat PowerShell-versiot	5
3.2	PowerShell versioiden väliset erot	6
3.2.1	PowerShell 2.0	7
3.2.2	PowerShell 3.0	8
3.2.3	PowerShell 4.0	9
3.2.4	PowerShell 5.0	10
3.2.5	PowerShell 5.1	11
4	PowerShellin käyttö	12
4.1	Järjestelmän käyttöajan tarkistus	12
4.1.1	Käyttöaika toisella tapaa	13
4.1.2	Eventld -tutkintaa	13
4.2	Virhetapahtumien haku	14
4.3	Laitteen tietojen haku	15
4.3.1	WinRM QuickConfig -asetusten palautus	16
4.3.2	Get-CimInstance	16
4.3.3	Win32_ComputerSystem	17
4.3.4	Win32_OperatingSystem	17

4.3.5	Win32_NetworkAdapter ja Win32_NetworkAdapterConfiguration	18
4.3.6	Get-NetAdapter	19
4.3.7	Muita luokkia	19
4.4	Get-Volume	20
5	PowerShellin jatkokehitys	21
5.1	PowerShell Linuxille	21
5.2	Windows PowerShell ja PowerShell Core	22
5.3	PowerShell Coren tunnistetut ongelmat	23
5.4	PowerShell Core -versiot	23
5.4.1	PowerShell Core 6.0	23
5.4.2	PowerShell Core 6.1	25
	Lähteet	27

Lyhenteet

PS	PowerShell.
WPS	Windows PowerShell.
PSC	PowerShell Core.
C.C	COMMAND.COM = MS-DOS-käyttöjärjestelmälle kehitetty komentotulkki.
MS-DOS	Microsoft Disk Operating System = Microsoftin kehittämä käyttöjärjestelmä, josta puuttui graafinen käyttöliittymä.
GUI	Graphical user interface = Graafinen käyttöliittymä.
WMF	Windows Management Framework = Asennuspaketti, joka sisältää itse PS-sovelluksen sekä muut käyttöön tarvittavat sovellukset.
NF	.NET Framework = Microsoftin ohjelmistokomponenttikirjasto.
ISE	PowerShell Integrated Scripting Environment = PS:n graafinen käyttöliittymä.
DSC	Desired State Configuration = PS:n hallintaympäristö.
RTM	Release to manufacturing = Ohjelman versio, joka julkaistaan valmistajille ennen varsinaista julkaisua.
WinRm	Windows Remote Management = Windowsin etähallintatyökalu.
CIM	Common Information Model = Tietotekniikan standardi malli laitteiden ja sovellusten ominaisuuksien määrittämiseksi.
WMI	Windows Management Instrumentation = Windowsin malli laitteiden ja sovellusten ominaisuuksien määrittämiseksi.

DCOM	Distributed Component Object Model = Microsoftin luoma teknologia ohjelmistokomponenttien väliseen kommunikointiin.
BIOS	Basic Input-Output System = Sovellus, joka vastaa käyttöjärjestelmän käynnistyksestä.
NC	.NET Core = Avoimeen lähdekoodiin perustuva kehitysalusta.
HDD	Hard disk drive = Kiintolevy.
SSD	Solid-state drive = Puolijohdelevy.

1 Johdanto

Tämän insinööriyön tarkoituksena oli tutkia PowerShellin kehityksen eri vaiheita ja toimintaa sekä avata eri versioiden välisiä muutoksia. Selvitystyö aloitettiin vanhemmista komentoliittymistä ja kuvattiin näiden toimintaa ja kehitystä lyhyesti, mikä johti lopulta siihen, miten PowerShell sai alkunsa.

Tämän jälkeen avattiin Windowsin PowerShell-versioiden sisältöä yksityiskohtaisemmin. Työssä käytiin läpi kaikki merkittävät PS-versiot ja selvitettiin, mikä muuttui edellisiin versioihin nähden. Eri versioista käytiin läpi olennaisimmat muutokset ja uudet ominaisuudet sekä avattiin näiden toimintaa.

PowerShellin käyttöä demonstroitiin tekemällä skripti, jolla tutkitaan laitteen tilaa. Skriptillä voidaan hakea PS:n lokeilta virheilmoituksia halutulta aikaväliltä ja tulostaa nämä tekstitiedostoon käyttäjän syöttämään polkuun. Skripti myös hakee muun muassa olennaisimmat tiedot käytössä olevasta laitteesta virheiden jatkotutkimusta varten ja tarkistaa laitteen käyttöajan viimeisimmän uudelleenkäynnistyksen jälkeen.

Lopuksi kuvattiin PowerShellin jatkekehitystä ja avattiin etenkin siirtymistä Windows PS:stä kohti PS Corea. Luvussa kuvattiin uusimpien Core-versioiden muutoksia Windows-versioihin nähden ja myös hieman PS:n toimintaa UNIX-sukuisilla käyttöjärjestelmillä.

2 PowerShellin synty ja käyttöönotto

2.1 Komentoliittymät Windows-laitteilla

Kaikissa Windowsin käyttäjille suunnatuissa henkilökohtaisissa tietokoneissa on ollut mukana jokin komentoliittymä käyttöjärjestelmän hallintaa varten. Windowsin 1.0-käyttöjärjestelmästä alkaen oli käytössä COMMAND.COM, joka oli mukana myös seuraavien sukupolvien Windows 95-, Windows 98- sekä Windows ME -järjestelmissä. (Techopedia.)

2.1.1 COMMAND.COM

COMMAND.COM on MS-DOS-käyttöjärjestelmälle kehitetty komentotulkki, jolla pystyy ajamaan komentoja joko suoraan tai .BAT-tiedostojen kautta yksinkertaisina skripteinä. MS-DOS-järjestelmässä ei ollut graafista käyttöliittymää, joten sen käyttö tapahtui täysin C.C:n kautta. Cmd.exe, jota kutsutaan yleisesti myös komentoriviksi, on C.C:n seuraaja, joka on ollut käytössä kaikissa edeltäjänsä uudemmissa käyttöjärjestelmissä mukaan lukien Windows 10. Koska C.C-liittymä on suunniteltu 16-bittisiä sovelluksia varten, suositellaan sitä käytettäväksi vain vanhemmissa MS-DOS:iin perustuvissa käyttöjärjestelmissä. C.C- ja cmd.exe-käyttöliittymissä on hyvin paljon samankaltaisuuksia, kuten .BAT-tiedostojen käyttö skriptien kirjoittamisessa ja erilaisten komentojen syntaksi, mutta liittymien välillä on kuitenkin joitain merkittäviä eroavaisuuksia. (Computerhope 2018a, Microsoft 2018a, Computerhope 2018b.).

2.1.2 Cmd.exe

Cmd.exe kehitettiin käyttäjäläheisemmäksi tarkentamalla käyttäjälle näytettäviä virheilmoituksia ja siihen on valmiiksi integroituna joitain käyttöä helpottavia työkaluja, kuten DOSKEY-komennot, jotka lisäsivät historiatoiminnon ja makrojen hallintaa. COMMAND.COM:ssa nämä oli ladattava erikseen. Lisäksi cmd.exe:stä löytyy muita käyttöä nopeuttavia ja parantavia toimintoja, kuten automaattinen komennon täydennys, eikä vastaavia edeltäjästä löytyneitä vanhemman teknologian rajoitteita, esimerkiksi liian pitkiä tiedostojen nimiä, enää ole. Cmd.exe on paranneltu versio edeltäjästään, ja se suunniteltiin toimivammaksi myös muille alustoille, ja näin korvasi uudemmissa käyttöjärjestelmissä käytännössä täysin vanhemman komentoliittymän. MS-DOS:iin perustuvia käyttöjärjestelmiä uudemmissa järjestelmissä C.C suorittaa syötetyt komennot cmd.exe:ssä, joten näennäisesti nämä kaksi toimivat samoin Windows NT -järjestelmissä. (Microsoft 2018a., Computerhope 2017.)

2.2 PowerShellin synty

Windows PowerShell on Windows-laitteille tarkoitettu uuden sukupolven komentoliittymä, joka kehitettiin korvaamaan aiemmin käytössä olleita komentorivityökaluja. Se on rakennettu Microsoftin .NET Framework -ohjelmistokomponenttikirjaston päälle, jota

käytetään Microsoft Windows -käyttöjärjestelmissä. PS:n avulla käyttäjät pystyvät helposti automatisoimaan käyttöjärjestelmän hallinnan toimintoja sekä prosesseja ja kirjoittamaan monimutkaisempiakin skriptejä, jollaisia vanhemmilla komentoliittymillä ei pystynyt suoraan tekemään. (Microsoft 2018b.)

Vuonna 2002 Microsoft aloitti uuden komentoliittymän kehityksen, jota tuolloin kutsuttiin nimellä Monad. Jeffrey Snover (*PowerShell Chief Architect*) kirjoitti samana vuonna raportin "Monad Manifesto", jossa kuvataan uuden komentoliittymän kehityksen syitä ja tarpeita yksityiskohtaisesti. Päällimmäisenä ongelmana oli, että Microsoftilta puuttui työkalu, jolla ylläpitäjät pystyisivät helposti syöttämään komentoja sekä automatisoimaan ylläpitoa. Tekstissään hän myös huomauttaa, että suurin osa järjestelmien ylläpitäjistä on niin kutsuttuja "para-programmereja", eli heillä ei itsellään ole joko osaamista tai aikaa kehittää monimutkaisia skriptejä, ja PS:n kehityksessä huomioitiin tämä ongelma. Jeffrey Snover tunnetaan Windows PS:n keksijänä. (Snover 2002.)

Vuonna 2017 Heavybitin haastattelussa Jeffrey Snover avasi tarkemmin syitä PowerShellin syntymiselle ja sille, mikä inspiroi häntä sen kehityksessä. Ennen PS:n tuloa Microsoftilla keskityttiin graafisten käyttöliittymien kehitykseen, ja tarvetta uudelle komentoliittymälle ei tunnistettu. Snover alkoi kehittämään PS:ää, ja jonkin ajan kuluttua Microsoftilla alettiin ymmärtää graafisten käyttöliittymien puutteet ylläpidollisissa tehtävissä. Yksittäisten työasemien ylläpitoon graafinen käyttöliittymä toimi hyvin, mutta ylläpidettäessä palvelinkeskuksia, joissa sama prosessi saatettiin toistaa 500 kertaa, tuli automaattinen ylläpito komentoliittymän kautta pakolliseksi. (Snover 2017, Heavybitin haastattelu.)

Ensimmäinen julkinen Monad beta -versio julkaistiin vuoden 2005 kesällä ja heti perään syyskuussa beta 2 -versio. Tammikuussa 2006 julkaistiin viimeinen beta 3 -versio, jonka jälkeen huhtikuussa Microsoft ilmoitti, että Monad on uudelleennimetty Windows PowerShelliksi, ja sen ensimmäinen RC-1-versio on julkaistu ladattavaksi. Samalla ilmoitettiin, että Microsoftin seuraavat hankkeet tulevat hyödyntämään samaa PS-arkkitehtuuria. Ilmoitustilaisuudessa esitettiin myös konkreettisesti PS:n käyttömahdollisuuksia Exchange 2007 -järjestelmän ylläpidollisissa tehtävissä. (Cleverism, Microsoft 2018c.)

3 PowerShellin kehitys ja versiot

3.1 PowerShell-versiot

PowerShellista on saatavilla useita eri versioita, ja oikean version valinta riippuu käytössä olevasta Windowsin käyttöjärjestelmästä ja siihen saatavilla olevasta .NET Framework -versiosta. Microsoftilla on tyypillisesti ollut tapana julkaista uusi PS-versio aina uuden käyttöjärjestelmän julkaisun yhteydessä. Ainoina poikkeuksina tähän ovat Windows 10, joka julkaistiin 2015, ja PS-versio 5.0 seuraavan vuoden puolella 2016. Osalle vanhemmista järjestelmistä on mahdollista päivittää uusi versio julkaisun jälkeen, mutta yhteensopivuus tulee ensin tarkistaa. Yritettäessä asentaa yhteensopimatonta versiota, ilmoittaa asennusohjelma yhteensopivuusvirheestä. (Warner 2015.) Taulukosta 1 löytyy listaus eri PS-versioista sekä niiden yhteensopivuudesta eri Windows-käyttöjärjestelmien kanssa.

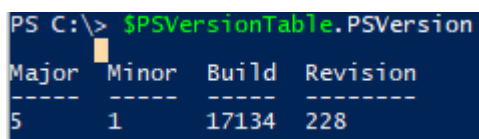
3.1.1 Microsoft .NET Framework

PowerShell -versiot ovat sidoksissa tiettyihin .NET Framework -versioihin, joille Microsoft jakaa saatavilla olevat päivitykset automaattisesti MS Updaten kautta. Microsoft .NET Framework on ajoympäristö erilaisille Windowsin komponenteille, ja sen ohjelmistorajapinta mahdollistaa pääsyn käytännössä kaikkialle Windows-ympäristössä, kuten käyttöjärjestelmään, tietokoneeseen, sovelluksiin ja sovellusten toiminnallisuuteen. PS on siis tehokas työkalu, jonka avulla hallitaan NF:ia. Se on rakennettu niin, ettei käyttäjän tarvitse täysin ymmärtää kaikkea taustalla tapahtuvaa toimintaa pystyäkseen käyttämään sitä tehokkaasti. Windows-työasemalle voi olla asennettuna samanaikaisesti useita eri NF-versioita. (Warner 2015.)

3.1.2 Windows Management Framework

Versiosta 3.0 alkaen päivitys onnistuu Microsoftin Windows Management Frameworkin kautta. WMF on paketti, joka sisältää itse PS-sovelluksen sekä muut käyttöön tarvittavat sovellukset. Ennen päivitystä tulee nykyinen versio tarkistaa. Tämä onnistuu helposti

PS-ikkunassa komennolla `PS C:\> $PSVersionTable.PSVersion`, joka palauttaa nykyisen käytössä olevan version. Itse asennus on varsin helppo, ja tarvittava asennuspaketti löytyy Microsoftin sivujen kautta. Asennus ei vaadi käyttäjältä muuta tietoa kuin onko käytössä oleva Windows käyttöjärjestelmä 32- vai 64-bittinen. (Warner 2015.) Jos käytössä on 64-bittinen Windows-versio, löytyy valikosta valmiiksi asennettuna myös 32-bittinen PS-versio testausta varten. Tämän tunnistaa valikossa lopun (x86)-liitteestä. Yleisesti ottaen 32- ja 64-bittiset versiot ovat keskenään yhteensopivia, ja ongelmiin törmää vain harvoin, mutta esimerkiksi työskenneltäessä 32-bittisten MS Office -sovellusten kanssa saattaa ongelmia ilmetä. Komennolla `[Environment]::Is64BitProcess` voidaan tarkistaa, onko käynnissä oleva PS-sessio 64-bittinen. (4sysops 2018.)



Major	Minor	Build	Revision
5	1	17134	228

Kuva 1. Tulos Windows 10 -käyttöjärjestelmässä PS-ikkunassa ajetusta komennosta `$PSVersionTable.PSVersion`. Kuvasta nähdään, että käytössä on PS versio 5.1, ja asennettuna on käyttöjärjestelmän versio 17134 ja revisio 228.

3.1.3 Vanhemmat PowerShell-versiot

PowerShellin kehityksessä on pyritty siihen, että uudemmat versiot olisivat suoraan yhteensopivia vanhempien versioiden kanssa, joten viimeisimpään versioon päivittäminen on yleisesti suotavaa, jos se on mahdollista. Poikkeuksena tähän on kuitenkin PowerShell Core 6.0, joka ei ole täysin yhteensopiva Windowsin PS:n kanssa. Uudempien toimintojen käyttö päivittämättömässä versiossa aiheuttaa skriptin epäonnistumisen. PS on mahdollista käynnistää versiossa 2.0, jos halutaan esimerkiksi testata, miten uudemmassa versiossa tehty skripti suoriutuu vanhoissa järjestelmissä. Tämä onnistuu komennolla `PS C:\> powershell -version 2.0`, jonka jälkeen voi edellä mainitulla komennolla varmistaa version vaihtumisen kyseisessä ikkunassa. (Warner 2015.)

Taulukko 1. Alla on listattuna taulukossa eri PowerShellin versiot, niiden julkaisupäivät, ja mille Windows-versioille ne ovat oletuksena asennettuina sekä Windowsin versiot, joille asennus on mahdollista. (4sysops 2018, Microsoft 2017a.)

PowerShell versio	Julkaisupäivä	Oletuksena Windows versioissa	Saatavilla Windows versioille
PowerShell 1.0	Marraskuu 2006	Windows Server 2008	Windows XP, Windows Server 2003, Windows Vista
PowerShell 2.0	Lokakuu 2009	Windows 7, Windows Server 2008 R2	Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008
PowerShell 3.0	Syyskuu 2012	Windows 8, Windows Server 2012	Windows 7, Windows Server 2008, Windows Server 2008 R2
PowerShell 4.0	Lokakuu 2013	Windows 8.1, Windows Server 2012 R2	Windows 7, Windows Server 2008 R2, Windows Server 2012
PowerShell 5.0	Helmikuu 2016	Windows 10	Windows 7, Windows 8.1, Windows Server 2012, Windows Server 2012 R2
PowerShell 5.1	Tammikuu 2017	Windows Server 2016	Windows 7, Windows 8.1, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2

3.2 PowerShell versioiden väliset erot

PowerShellin kehitys on ollut alusta asti Microsoftille tärkeää, ja se näkyy eri versioiden välisissä muutoksissa. Uusia ominaisuuksia lisätään ja vanhoja kehitetään aina uusien versioiden myötä, ja tämä mahdollistaa PowerShellin käytön yhä laajemmin järjestelmien hallinnassa ja automatisoinnissa. Seuraavissa luvuissa esitellään tarkemmin versioiden välisiä muutoksia ja merkittävimpiä uusia ominaisuuksia.

3.2.1 PowerShell 2.0

PowerShell 2.0 julkaistiin Windows 7 -käyttöjärjestelmän yhteydessä, ja se toi mukanaan monia merkittäviä uudistuksia edeltäjäänsä nähden. Versiossa mahdollistettiin komentojen ja skriptien ajaminen yhdeltä koneelta etänä usealle eri työasemalle, mikä oli huomattava parannus aiempien järjestelmien etähallinnan kannalta. Toiminto edellytti, että versio 2.0 oli asennettuna kaikkiin laitteisiin, joihin etähallinta kohdistui. Myös moduulit esiteltiin versiossa, joiden avulla skriptien kirjoittajat pystyivät kokoamaan skriptit omiksi uudelleenajettaviksi packageikseen. Moduulien kautta ajettava koodi suoritetaan omassa istunnossaan, eikä se vaikuta istunnon tilaan moduulin ulkopuolella. (Microsoft 2018d.) Moduuleja käytetään esimerkiksi funktiokirjastojen pakkaukseen ja jakoon. Moduuleja on myös mahdollista hyödyntää oman ympäristön muokkaamisessa esimerkiksi lisäämällä haluttuja cmdlet-komentoja tai funktioita. PS tulkitsee myös kaikki psm1-muotoon talletetut tiedostot moduuleiksi, joten yksinkertaisimmillaan yksittäinen skripti voidaan tulkita moduuliksi. (Microsoft 2016a.)

Versiossa lisättiin noin 100 uutta cmdlet-komentoa, joiden joukossa oli muun muassa Job-komennot tehtävien suorittamiseksi taustalla. Start-Job -komennolla käynnistetään haluttu tehtävä, joka siirtyy taustalle, ja käyttäjä voi tehtävän suorituksen aikana jatkaa muita töitään. Get-Job-komennolla saadaan haettua tehtävän tila, ja kun tehtävä on valmis, voidaan sen sisältö tulostaa ruudulle Receive-Job-komennolla. Tehtäviä on myös mahdollista käynnistää etänä taustalle -AsJob-parametrin avulla. Esimerkiksi komennolla *Start-Job -ScriptBlock {Get-EventLog Application -EntryType error}* voidaan käynnistää taustalle haku, joka käy läpi kaikki koneelta löytyvät sovellusten kirjaamat virheet. Myös tapahtumien seuranta parannettiin ja lisättiin komentoja niiden seuranta ja hallintaa varten. (Microsoft a.)

Skriptien kirjoituksessa edelleen paljon käytetty PowerShell Integrated Scripting Environment esiteltiin tässä versiossa, ja se toi käyttöön graafisen käyttöliittymän, jossa käyttäjät pystyvät ajamaan ja editoimaan sekä debugaamaan skriptejään. Ohjelman ominaisuuksiin kuuluu muun muassa syntaksien korostus ja Unicode-merkistöstandardin tuki. ISE mahdollisti myös testauksen ja virheenkorjauksen kannalta tärkeän ominaisuuden, jossa ainoastaan maalattu skriptin osa pystyttiin suorittamaan istunnossa. (Microsoft 2018d.)

3.2.2 PowerShell 3.0

PowerShell 3.0 oli ensimmäinen versio, joka jaeltiin Windows Management Framework -paketin mukana. Päivityksessä tuotiin parannuksia useisiin vanhoihin ominaisuuksiin, parannettiin käytettävyyttä ja lisättiin yli tuhat uutta cmdlet-komentoa. Yksi merkittävä muutos tehtiin PSSession-etäyhteyksiin, jotka aiemmassa versiossa katkesivat suljettaessa sessio, jossa yhteys oli avattu. Versiosta 3.0 alkaen *New-PSSession* komennolla avattu yhteys ei enää katkennut suljettaessa sessio, jossa se avattiin, vaan jäi auki ja siihen pystyi palaamaan myöhemmin jatkamaan toimia. Tämä tarkoitti myös sitä, että sessioon pystyi palata myöhemmin halutessaan toiselta työasemalta, mikä ei aiemmin ollut mahdollista. (Honeycutt 2012: 33-34). *Get-PSSession* komennolla voidaan hakea avatut etäyhteydet ja *Remove-PSSession* yhteydellä katkaista tarpeettomat. Jos yhteyttä ei katkaista, se pysyy auki, kunnes se katkeaa aikakatkaisuun. *PSSessionOption* cmdlet-komennolla voidaan muuttaa oletuksena uudelle PSSessiolle asetettavia arvoja, kuten aikakatkaisun pituutta parametrilla *-IdleTimeout*. (Microsoft 2018e.)

Windows PowerShell Workflow -ominaisuus mahdollisti workflow-tyyppisten skriptien kirjoittamisen PS:llä. Workflow'n etu tavallisiin skripteihin verrattuna on sen ominaisuus ajaa useita toimintoja rinnakkain, ja tämä on etenkin hyödyllistä suoritettaessa paljon aikaa vieviä tehtäviä useilla laitteilla. (Microsoft 2012.). Esimerkiksi usealla virtuaalikoneella voi suorittaa jonkin prosessin samanaikaisesti, mikä normaalin skriptin kautta ajettaessa suoritettaisiin koneille peräkkäin yksi kerrallaan. Workflow:ssa pystyy myös automatisoimaan toimia virhetilanteiden varalle asettamalla tallennuspisteitä etenkin sellaisiin työn vaiheisiin, jotka ovat alttiita virheille. Virhetilanteen sattuessa voidaan palata edelliseen tallennuspisteeseen ja jatkaa eteenpäin sekä tarvittaessa suorittaa epäonnistunut vaihe uudelleen. Hyvä esimerkki on virtuaalikoneen luonti, joka epäonnistuessaan tulisi tehdä uudelleen. Onnistuessaan virtuaalikonetta ei kuitenkaan turhaan kannata luoda uudelleen, jos myöhemmin tehtävän edetessä tapahtuu virhe, joten tästä syystä tallennuspiste tulisi asettaa juuri ennen virtuaalikoneen luontia ja heti sen jälkeen. (Microsoft 2018f.)

Muita uudistuksia olivat muun muassa ajastetut tehtävät, joissa PowerShellilla pystyttiin ajastamaan käyttöjärjestelmässä suoritettavia tehtäviä käyttäen apuna Windowsin tehtävien ajoitusohjelmaa. Ajastukseen pystyi lisäämään tehtävän suoritettavaksi tasaisin

väliajoin tai esimerkiksi silloin, kun jokin määrätty tapahtuma kirjataan lokiin. Myös ohjeistusta parannettiin lisäämällä komento *Update-Help*, joka hakee verkosta suoraan uusimmat ohjeet ja päivittää ne käynnissä olevaan PS-sessioon. Myös *Get-Help*-komentoon lisättiin parametri *-Online*, jonka avulla voidaan yksittäisen cmdlet-komennon tarkempi ohjeistus ladata verkosta. (Honeycutt 2012: 33-34.)

ISE sai myös päivityksiä käytettävyyteen uuden IntelliSense-ominaisuuden kautta. Kirjoitettaessa komennon alkuosa tulee ISE:ssä näkyviin pudotusvalikko mahdollisista komennoista, joita käyttäjä voi syöttää, ja nuolilla voidaan valita haluttu komento ja täydentää se tab-näppäimellä. Käyttäjän ei siis itse tarvitse muistaa kuin osa komentoa, ja ISE näyttää kaikki saatavilla olevat mahdolliset vaihtoehdot. Versiossa lisättiin myös erillinen komentoikkuna, jossa on listattuna kaikki saatavilla olevat komennot käytössä olevista PS-moduuleista. Ikkunasta pystyy myös ajamaan komentoja syöttämällä ensin vaadittavat parametrit. (Honeycutt 2012: 34-35.)

3.2.3 PowerShell 4.0

Uusi PowerShell-versio on tyypillisesti julkaistu aina uuden käyttöjärjestelmän yhteydessä, sama pätee myös versioon 4.0. Erona aiempaan on kuitenkin käyttöjärjestelmien Windows 8 ja Windows 8.1 välinen lyhyt ajankohta julkaisujen välillä: 8.1 julkaistiin jo vuoden kuluttua edeltäjästään. Tämä näkyy myös PS-versioiden 3.0 ja 4.0 välisissä muutoksissa, jotka ovat aiempaan nähden vähäisempiä, ja muutosta voidaankin pitää enemmän päivityksenä kuin merkittävänä versiouudistuksena. Versiossa lisättiin debugaus-mahdollisuus etänä ajettaviin sekä workflow-skripteihin. Aiemmissa versioissa skriptien debugaus oli mahdollista ainoastaan paikallisesti ajettaessa. Ominaisuus lisättiin sekä PS-konsoliin että ISE:een, ja skriptien debugaus etänä vaati toimiakseen version 4.0 molemmille laitteille. Debugaus tapahtuu molemmissa tapauksissa komennolla *Set-PSBreakpoint*, jolla asetetaan halutulle riville pysäytyspiste, johon skripti ajettaessa pysähtyy, ja käyttäjä voi tarkistaa skriptin tilan. Versiossa korjattiin aiemmasta löydettyjä bugeja sekä parannettiin suorituskykyä. (Microsoft 2015.)

Versiossa lisättiin kuitenkin yksi merkittävä uudistus; Desired State Configuration, joka mahdollistaa deklaratiivisten skriptien kirjoituksen. Nämä eroavat normaaleista skripteistä siten, että kirjoitettaessa koodia kerrotaan, miltä lopputuloksen tulisi näyttää,

esimerkiksi mikä luotavan tiedoston nimeksi halutaan, missä se sijaitsee ja minkä tyyppinen tiedosto on. Tämän jälkeen PowerShell tekee taustalla tarvittavat toimet ja luo kyseisen tiedoston. Skriptiin ei siis lisätä osiota, joka suorittaa kyseisen tiedoston luonnin, vaan ainoastaan kerrotaan, millainen tiedosto halutaan. DSC:tä pystyy soveltamaan käytännössä kaikkeen, mitä PS-moduulien kautta pystytään muokkaamaan. (PowerShell.org 2013.)

3.2.4 PowerShell 5.0

PowerShell 5.0 julkaistiin helmikuussa 2016 ja se päivitettiin oletuksena Windows 10 -käyttöjärjestelmään, joka oli julkaistu vuoden 2015 puolella. WMF 5.0 RTM -paketti julkaistiin jo aiemmin. Se sisälsi myös PS 5.0-version, mutta paketista löytyi virhe, jonka johdosta version lopullinen julkaisu viivästyi. Virhe liittyi PSModulePath-muuttujaan, joka pitää sisällään levyille asennettujen PS-moduulien sijainnit. PS hakee muuttujasta moduulien sijainnit, jos käyttäjä ei ole niitä syöttänyt erikseen. Version virhe aiheutti sen, että asennettaessa alkuperäinen WMF 5.0 RTM -paketti päivitettiin kyseisen muuttujan arvot oletusarvoiksi, jolloin käyttäjän lisäämät viittaukset moduuleihin hävisivät. (Microsoft 2016b.)

Päivitys toi mukanaan merkittäviä uudistuksia, kuten PackageManagement-moduulin, jota aiemmin kutsuttiin nimellä OneGet, mutta nimi muutettiin julkaistuun versioon. Moduuli mahdollistaa sovellusten asennuksen suoraan PS:n komentoliittymästä samaan tapaan kuin Linuxin *apt-get install*-komento. Moduulin komennolla *Find-PackageProvider* voidaan hakea lista saatavilla olevista ohjelmistotarjoajista, ja komennolla *Install-PackageProvider* asennetaan haluttu tarjoaja. Tämän jälkeen voidaan hakea ohjelmia kyseiseltä tarjoajalta *Find-Package*-komennolla halutuilla rajausehdoilla ja asentaa ohjelma komennolla *Install-Package*. Myös ohjelmien poisto onnistuu komennolla *Uninstall-Package*. (Woshub 2017.)

Versiossa lisättiin myös useita muita hyödyllisiä moduuleita, kuten NetworkSwitch-moduuli, joka sisältää cmdlet-komennot kytkinten porttien konfigurointiin ja ylläpitoon Windows Server 2012 R2-sertifioituille laitteille. Microsoft.PowerShell.Archive-moduuli taas toi komennot tiedostojen pakkaamiseen zip-tiedostoiksi sekä näiden purkamiseksi. PowerShellGet-moduuli mahdollisti uusien moduulien ja DSC-resurssien haun, julkaisun,

asennuksen sekä päivittämisen PowerShell Galleryyn tai sisäiseen moduuliluetteloon. (Microsoft 2017b.). Myös omien PS-luokkien luonti tuli mahdolliseksi versiossa. Käyttäjät voivat tehdä omia luokkia ja antaa niille haluttuja ominaisuuksia ja metodeja, joita käytetään uusien objektien luontiin. Luokkien lisääminen laajensi PS:n käyttömahdollisuuksia erilaisten ongelmien ratkaisussa. (Microsoft 2018g.)

3.2.5 PowerShell 5.1

Uusin Windows PowerShell 5.1 -versio julkaistiin WMF 5.1 -paketin mukana tammi-kuussa 2017. Julkaisussa tuotiin lähinnä parannuksia ja lisäyksiä aiemmassa versiossa 5.0 lisättyihin ominaisuuksiin, mutta myös joitain uusia toimintoja lisättiin, kuten *Get-ComputerInfo* cmdlet -komento, jonka avulla saadaan helposti haettua erilaisia käytössä olevan järjestelmän sekä käyttöjärjestelmän tietoja. *Catalog* cmdlet -komennolla *New-FileCatalog* voidaan luoda uusia .cat-päätteisiä catalog-tiedostoja, jotka pitävät sisällään viitteen kaikkiin tiettyssä kansiorakenteessa sijaitseviin tiedostoihin. Komennolla *Test-FileCatalog* taas voidaan testata tuota luotua tiedostoa, jolloin verrataan tiedostosta löytyviä viitteitä levyältä löytyvään todelliseen tilanteeseen. Komento palauttaa arvon *Valid*, jos eroavaisuuksia ei löydy, mutta jos löytyy, palautetaan arvo *ValidationFailed*. Parametrilla *-Detailed* voidaan tarkastella muuttuneita tietoja. Komentoja voidaan käyttää esimerkiksi jonkin kansiorakenteen alta löytyvien tiedostojen muutosten seurantaan. (Microsoft 2018h.)

Yksi merkittävin osa julkaisua oli kuitenkin PS:n jako kahteen eri editioon, jotka olivat työpöytä- ja Core-editio. Työpöytä-editio oli nimensä mukaisesti tarkoitettu Windowsin työpöytäkoneille, se on rakennettu .NET Framework -ohjelmistokomponenttikirjaston päälle, kuten aiemmat PS-versiot. Core-editio taas oli suunniteltu kevyemmille Windowsin versioille, kuten Nano Server ja Windows IoT, se on rakennettu .NET Core -ohjelmistokomponenttikirjaston päälle. (Microsoft 2018h.)

4 PowerShellin käyttö

Tässä luvussa esitellään PowerShellin käyttöä laitteen tutkimisessa luomalla skripti viivanselvityksen avuksi. Skriptillä tutkitaan tietokoneen event-lokeja, joiden avulla selvitetään tapahtuneita virheitä. Vikojen jatkoselvitystä varten haetaan tarpeellisia tietoja käyttäjän laitteesta ja tulostetaan virheilmoitukset erilliseen tiedostoon.

4.1 Järjestelmän käyttöajan tarkistus

Yhtenä osana skriptiä halusin tarkistaa Windowsin käyttöajan eli kuinka kauan järjestelmä on ollut käynnissä edellisen uudelleenkäynnistyksen jälkeen. Tarkoituksena oli asettaa raja, jonka ylittymisen jälkeen käyttäjää kehoitetaan uudelleenkäynnistämään laite. Rajaksi valitsin kaksi viikkoa. Tutkiessani eri tapoja selvittää järjestelmän käyttöaikaa törmäsin erilaisiin eventlog-tapahtumiin, ja näistä selkein oli eventid 6013, joka kertoo järjestelmän käyttöajan sekunteina edellisen uudelleenkäynnistyksen jälkeen. Tämä vaikutti tarkoitukseen sopivalta, mutta ongelmana oli se, että käyttöaika ilmoitettiin tuloksessa viestin sisällä tähän tapaan: ”Järjestelmän toiminta-aika on 581909 sekuntia”, eikä omana arvonaan. Saadakseni ainoastaan tuon käytetyn ajan talletettua omaksi arvokseen käytin substring-metodia katkaistakseni tekstin alkamaan aikamääreen kohdalta ja talletin tuloksen muuttujaan. Metodilla pystyy myös suoraan määrittämään, kuinka monta merkkiä merkkijonosta poimitaan, mutta tuosta ei ollut tässä tapauksessa apua, koska sekuntimäärä vaihteli radikaalisti tulosteiden välillä. Käytin muokkaukseen vielä replace-metodia, jonka avulla sain poistettua lopusta ylimääräisen tekstin ja korvattua sen tyhjällä, jolloin sain pelkän sekuntimäärän talletettua muuttujaan. Muutin vielä ilmoitetut sekunnit tunneiksi, jotta saadaan näytettyä käyttäjälle parempi kuva edellisestä uudelleenkäynnistyksestä kuluneesta ajasta.

Lisäsin skriptiin komennon, joka hakee tuoreimman eventid 6013:n tiedot, muokkaa ne edellä ilmoitetulla tavalla ja näyttää käyttäjälle, kuinka pitkään järjestelmä on ollut käynnissä ilman uudelleenkäynnistystä. Jos järjestelmä on ollut käynnissä kaksi viikkoa, suositellaan käyttäjälle tekemään uudelleenkäynnistys. Tämä toimi ensimmäisissä testauksissa hyvin, mutta myöhemmin testattaessa huomasin, että eventid 6013 päivitetään melko harvoin. Selvitellessäni asiaa havaitsin, että Microsoftin omilta sivuilta oli vaikeaa

löytää tarkempaa tietoa yksittäisistä eventid:stä ja niiden tarkoituksesta, mutta löysin kuitenkin muilta sivuilta tietoa, joiden perusteella tuo event kirjataan noin 24 tunnin välein. Tarkastellessani omalta koneelta lokia pidemmältä aikaväliltä havaitsin, että tuo kirjaus tosiaan tapahtuu oletuksena kello 12:00 tai heti koneen käynnistyksen yhteydessä tuon kellonajan jälkeen. Jotta kahden viikon aikaraja tarkistuksessa ei pääsisi ylittymään, las-kin tarkistuksessa käytettävän lukuarvon 312 tuntiin, mikä vastaa 13 päivän tuntimäärää. Käytin aluksi vertailussa tuota arvoa tunteina, jonka olin aiemmin muuntanut sekun-neista, mutta tämä ei testeissä toiminut toivotulla tavalla. Vertailtaessa eventid 6013:n kautta saatua lukua lukuun 312, tulkittiin esimerkiksi luku 161,64 suuremmaksi kuin luku 312. Tämä johtui siitä, että käännettäessä sekuntimäärä tunneiksi jäi lukuun desimaa-leja, jolloin vertailussa käytettiin tässä tapauksessa desimaalipilkun jälkeistä arvoa 640, joka on suurempi kuin vertailuluku. Korjauksena ongelmaan päätin tehdä vertailun alku-peräisellä sekuntiarvolla ja vertasin tuota arvoa lukuun 1123200, joka on sekuntimäärä 13 päivän ajalta. Eventid:ltä luetun käyttöajan ollessa yli 13 päivää kehoitetaan laite käyn-nistämään uudelleen, ja koska kahden viikon aikaraja ei oltu kiveen hakattu, vaan tarkoi-tus oli kehottaa käyttäjää tekemään uudelleenkäynnistys silloin, kun järjestelmä on ollut pitkään käynnissä, ei tuo muutos 14 päivästä 13:een aiheuttanut ongelmaa.

4.1.1 Käyttöaika toisella tapaa

Myöhemmin tutkiessani jatkoselvitystä varten tarpeellisia käyttöjärjestelmän tietoja ha-vaitsin, että CimInstance-luokasta win32_OperatingSystem löytyi tieto LastBootUpTime. Käyttämällä komentoa *New-TimeSpan* pystytään tuota arvoa vertaamaan komennolla *Get-Date* saatavaan reaaliaikaan, ja näiden erotuksesta saadaan todellinen kulunut aika edellisestä uudelleenkäynnistyksestä. *New-TimeSpan* ilmoittaa suoraan kuluneen ajan eri aikamuodoissa, kuten kokonaismäärän päivinä tai tunteina, joten tuo ei vaadi erillistä muutosta, jolla saadaan haluttu aikayksikkö.

4.1.2 EventId -tutkintaa

Erilaisten EventLogin eventid -ilmoitusten kautta pystytään selvittämään tarkemmin ta-pahtumia, kuten mahdollisia syitä järjestelmän sammutuksille. Vertaamalla eventid:iä 6006, joka kirjataan käyttäjän painaessa sammuta tai käynnistä uudelleen, eventid:hen 6005, joka taas kirjataan käynnistyksessä, saadaan tarkempaa tietoa, kuinka pitkään

järjestelmä oli alhaalla tai kuinka nopeasti se käynnistyi. Eventid 6013 on informatiivinen tieto, joka kertoo järjestelmän käyttöajan 24 tunnin tarkkuudella, mutta jos halutaan ainoastaan tietää reaaliajassa, kuinka pitkään järjestelmä on ollut käynnissä edellisestä uudelleenkäynnistyksestä, on tuo New-TimeSpan-käyttö LastBootUpTime kanssa parempi tapa ilmaista asia. Pidin kuitenkin skriptissäni alkuperäisen tarkistustavan, sillä työn tarkoituksena on esitellä PowerShellin käyttöä eri tavoilla.

4.2 Virhetapahtumien haku

Yksi skriptin olennaisin osa oli selvittää erilaisia laitteen virheitä käyttäen PowerShellin lokeja. Päädyin käyttämään tähän EventLogia ja tarkemmin System- ja Application-lokeja. Käyttäjältä kysytään haluttu aloitus- ja lopetuspäivämäärä, joilla saadaan rajattua ajankohta tutkimuksia varten käyttäen -After- ja -Before-parametreja. Tarkoituksena oli hakea lokeja ainoastaan päivärajauksella, ja testauksessa havaitsin, että syötettäessä aloitus- ja lopetuspäivämääräksi sama päivä, haetaan lokilta kaikki tapahtumat. Tämä johtui siitä, että syötettäessä molemmille parametreille ainoastaan päivämäärä ilman kellonaikaa, myös -Before-parametrille käytetään oletuksena kellonaikaa 00:00, kuten -After-parametrissa. Tällöin raja on ”ennen ja jälkeen saman ajankohdan”, eli huomioidaan tapahtumat koko lokin ajalta. Korjauksena lisäsin -Before-parametrille käyttäjän syöttämän päivämäärän jälkeen kellonajan 23:59, jolloin myös yksittäisen päivän virheiden haku onnistui. Aikarajauksen lisäksi pyydetään vielä syöttämään haluttu kansio, johon lokit talletetaan, jonka jälkeen etsitään annetulta aikaväliltä varoituksia ja virheitä käyttäen parametria -EntryType error, warning. Tiedot haetaan kahdessa osassa: ensin System-lokilta ja sitten Application-lokilta, jonka jälkeen talletetaan tiedot käyttäjän syöttämän polun alle tekstitiedostoon käyttäen *Out-File* -komentoa.

Tämän lisäksi halusin vielä hakea käyttäjälle tietoa järjestelmän vakavammista virheistä omaan lokiinsa. EventLogiin talletettavat virheilmoitukset ovat error-tyyppisiä, ja -EntryType-parametrilla ei suoraan saa haettua eritasoisia virheitä. Selvitin erilaisia virheiden lähteitä ja päädyin lisäämään erillisen haun EventLogin System -lokille, jossa virheiden lähde on BugCheck. Tällä saadaan haettua virheitä, jotka ovat aiheuttaneet bluescreenin eli järjestelmän kaatumisen. Syitä tällaisille virheille voivat olla esimerkiksi laitevika, kuten viallinen virtalähde tai järjestelmän ylikuumeneminen, virhe laitteen ajureissa tai oh-

jelmistovika. Tämän lisäksi halusin vielä erillisen haun, jolla haetaan mahdollisia levyviikaan viittaavia virheilmoituksia, ja tämä onnistui käyttämällä virheiden lähteenä -Source Disk-parametria, eli haetaan levyyn liittyviä ilmoituksia. Näiden lisäksi otin mukaan vielä yhden haun, jolla etsitään EventLog:lta instanceid 41 -tyyppisiä ilmoituksia. Windows-järjestelmän käynnistyksessä tarkistetaan aina, oliko edellinen sammutus niin sanottu "Clean shutdown", eli sammutettiinkö järjestelmä hallitusti. Jos näin ei tapahtunut, eli sammutus johtui esimerkiksi virtakatkosta tai käyttäjä teki sammutuksen virtapainikkeesta, kirjataan tästä instanceid 41 -tyyppinen ilmoitus. Nämä vakavammat virheet haetaan ja tulostetaan yhteen omaan tiedostoon, eli skriptin suorittaminen hakee tietoja yhteensä kolmeen eri tiedostoon: yhteen Application-lokin virheet, yhteen System-lokin virheet ja yhteen vakavammat virheet. Näiden avulla käyttäjä voi tutkia ongelmien syitä ja lähteitä.

4.3 Laitteen tietojen haku

Virheiden selvityksen kannalta olennaisena osana on käytössä olevan järjestelmän ja laitteen tiedot. Halusin lisätä skriptiin osion, joka hakee käyttäjälle nämä perustiedot ongelmien jatkoselvitystä varten. Tutkin aluksi eri tapoja, joilla PowerShellissa pystytään hakemaan näitä tietoja, jolloin törmäsin jo aiemmin mainittuun PS-versiossa 5.1 lisättyyn cmdlet-komentoon *Get-ComputerInfo*, jolla saa helposti haettua esimerkiksi tietoja laitteesta, käyttöjärjestelmästä ja BIOS:sta. Testatessani tämän toimintaa omalla koneellani havaitsin kuitenkin, että vain pienelle osalle komennon hakemista tiedoista palautettiin jokin arvo, ja suurin osa jäi tyhjäksi. Testasin toimintaa myös toisella koneellani, ja sama toistui. Aloin selvittämään asiaa ja selvisi, että myös muilla käyttäjillä on vastaavaa ongelmaa tuon cmdlet-komennon käytössä, ja käydessäni eri artikkeleita läpi, löysin ongelmaan lopulta ratkaisun. Toimiakseen tuo komento vaatii, että WinRm-palvelun tulee olla päällä ja sen asetukset tulee määrittää, jotta etäyhteydet laitteeseen toimivat.

Komento *Get-ComputerInfo* itsessään ei liity millään tavalla etäyhteyksiin, vaan tutkii ainoastaan tietoja paikalliselta koneelta. Komentoon on kuitenkin määritetty computer-name-arvoksi localhost, jonka on tarkoituksella suunniteltu käyttävän tuota WinRm-palvelua. Jos arvo olisi null, toimisi cmdlet luultavasti oikein, mutta tuota ei pysty parametrilla muuttamaan. Jotta komennolla saadaan näkyviin tarvittavia arvoja, voi WinRm-palvelun

ottaa käyttöön ja määrittää asetukset helposti komennolla *WinRm QuickConfig*. Tuo komento käynnistää WinRm-palvelun, muuttaa palvelun käynnistystyyppin automaattiseksi ja luo kuuntelijan (Listener), joka vastaanottaa koneen IP-osoitteisiin HTTP-protokollan kautta lähetettyjä viestejä. Komento myös luo poikkeuksen palomuurin asetuksiin sallien WinRm-palvelun ja muuttaa rekisterin asetuksia niin, että otettaessa etäyhteys tähän koneeseen järjestelmänvalvojan oikeuksilla, jäävät nuo oikeudet käyttöön. Oletuksena arvo on asetettu siten, että järjestelmänvalvojan oikeuksilla otettaessa etäyhteys laitteeseen poistetaan kyseiseltä käyttäjältä järjestelmänvalvojan oikeudet etälaitteeseen. Kun tämä on tehty, näyttää edellä mainittu cmdlet-komento kaikki siihen määritellyt saatavilla olevat tiedot laitteesta.

4.3.1 WinRM QuickConfig -asetusten palautus

Tämän testauksen jälkeen havaitsin, että PowerShellissa on tarjolla vain tuo komento asetusten määrittämiseen ja kääntämiseen päälle, mutta vastaavaa komentoa ei löydy, joka muuttaisi asetukset takaisin alkuperäisiin arvoihin. Tämä piti siis tehdä manuaalisesti. Komennolla *Set-Service -Name winrm -StartupType Disabled*, saatiin tuo käynnistystyyppi disabloitua ja kuuntelija poistettua komennolla *winrm delete winrm/config/Listener?Address=*&Transport=http*. Myös palomuuriasetuksen pystyi poistamaan helposti PowerShellilla ajamalla komennon *Get-NetFirewallRule | ? {\$_.Displayname -eq "Windowsin etähallinta (saapuva HTTP)"} | Set-NetFirewallRule -Enabled "False"*. Tässä tulee huomioida Windowsin kieliasetus ja hakea asetusta oikealla nimellä järjestelmään asetetun kielen mukaan. Lopuksi muutetaan vielä tuo rekisteriasetus komennolla *Set-ItemProperty -Path "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System" -Name LocalAccountTokenFilterPolicy -Value 0*.

4.3.2 Get-CimInstance

Koska *Get-ComputerInfo*-komennon käyttö osoittautui oletettua vaikeammaksi tuon WinRm-virheen johdosta, päätin olla käyttämättä tuota ja kokosin kaikki tiedot itse. PS:stä löytyy tähän tarkoitukseen muutamia erilaisia cmdletteja, jotka ovat Common Information Model eli CIM-cmdletit ja Windows Management Instrumentation eli WMI-cmdletit. Hieman tutkittuani asiaa oli selkeää, että näistä kannattaa käyttää mieluummin CIM-cmdlet komentoja, sillä jopa *Get-WmiObject*:in help-osiosta löytyy kommentti, että

alkaen PowerShell-versiosta 3.0 tämä cmdlet on korvattu komennolla *Get-CimInstance*. Testimielessä kokeilin kuitenkin tietojen hakua molemmilla tavoilla enkä havainnut juurikaan ongelmia tai poikkeuksia näiden käytössä tämän skriptin kannalta. Vastaavat tiedot löytyivät molemmilla komennoilla hieman eri nimillä. Merkittävin ero näiden välillä vaikuttaisi kuitenkin olevan ajettaessa komentoja laitteille etäyhteyden yli, jolloin WMI cmdletit käyttävät etäyhteydessä DCOM-protokollaa, jonka kanssa saattaa tulla ongelmia palomuurin kohdalla. Jotta cmdletit toimivat, laitteen asetukset saattavat joutua muuttamaan manuaalisesti, mikä voi olla todella työlästä, jos kyseessä on useita eri laitteita pitkien yhteyksien välillä. CIM cmdletit eivät oletuksena käytä tätä protokollaa, näiden käyttö on lähtökohtaisestikin suunniteltu käyttäjäystävällisemmiksi toimittaessa etäyhteyksien yli.

4.3.3 Win32_ComputerSystem

Aloitin selvittämisen siitä, mitä tietoja laitteesta yleisesti tarvitaan ongelmanselvitystä varten. Tutkin CimInstance-luokkia ja niiden sisältöä ja kokosin näistä mukaan tarpeelliset. Ensimmäisenä hain tiedot luokasta Win32_ComputerSystem, joka pitää sisällään tietoja itse laitteesta, kuten laitteen nimen, pääkäyttäjän, laitteen valmistajan, mallin ja käytössä olevan kokonaismuistin määrän. Muistin määrä on ilmoitettu tavuina, niin kuin PS:ssä usein on tapana, joten tuo käännettiin vielä gigatavuiksi antamaan parempi kuva todellisesta muistin määrästä. Myöhemmin selatessani Microsoftin sivuilta Win32_ComputerSystem-osiota havaitsin kuitenkin, että tässä luokassa ilmoitettu määrä ei aina ole tarkka arvo, esimerkiksi silloin, kun BIOS käyttää osaa fyysisestä muistista. Sivulla kehoitettiin sen sijaan käyttämään oikean arvon saamiseksi toista luokkaa, joka on Win32_PhysicalMemory, ja tarkemmin tämän luokan ominaisuutta Capacity. Tein testiksi haut molemmista luokista ja huomasin, että Capacity antoi tosiaan hieman korkeamman arvon kuin Win32_ComputerSystem-luokan ominaisuus TotalPhysicalMemory, joka on todellinen omasta tietokoneestani löytyvä muistin määrä. Tein siis skriptiin muutoksen, jossa tuo kokonaismuistin määrä haetaan luokasta Win32_PhysicalMemory.

4.3.4 Win32_OperatingSystem

Seuraavaksi tarkastelin luokkaa Win32_OperatingSystem, joka nimensä mukaisesti sisältää tietoa laitteen käyttöjärjestelmästä. Sieltä saatiin haettua tarkempaa tietoa muun muassa käytössä olevasta muistin määrästä, käyttöjärjestelmän versiosta, Service pack

-versiosta, valmistajasta ja käyttöjärjestelmän arkkitehtuurista. Lisäsin tarvittavat tiedot, mutta testatessani skriptiä havaitsin, että ilmoitetut muistin määrät eivät näyttäneet arvoja oikein. FreePhysicalMemory sekä FreeVirtualMemory näyttivät molemmat arvokseen 0, kun yritin kääntää luvut gigatavuiksi, ja katsoessani PS:ltä saatuja arvoja tosiaan havaitsin, että luvut ovat oletettua pienempiä. Tutkin asiaa, ja Microsoftin sivuilta selvisi, että luokassa Win32_OperatingSystem ovat muistiarvot ilmoitettukin kilotavuina, ja tämän johdosta ilmoitetut luvut ovat pienempiä kuin muissa tutkimissani luokissa. Korjauksena jaoin luvut megatavuilla gigatavujen sijaan, jolloin saatiin oikeat lukemat näkyviin. Tämän lisäksi halusin vielä tiedon siitä, kuinka suuri osa muistista on vapaana prosentteina skriptin ajohetkellä. Tämän sai selvitettyä jakamalla vapaana olevan muistin määrän kokonaismäärällä, ja nämä molemmat ominaisuudet löytyivät luokasta Win32_OperatingSystem. Tein jaon ja lisäsin vielä loppuun metodin ToString("P"), jolla luku saadaan suoraan käännettyä prosenttiluvuksi. Vastaavalla tavalla lisäsin skriptiin ensin fyysisen muistin ja perään vielä virtuaalisen muistin määrät prosentteina.

4.3.5 Win32_NetworkAdapter ja Win32_NetworkAdapterConfiguration

Laitteen verkkokortin tietojen selvitys skriptiä varten osoittautui hieman hankalammaksi kuin muiden tietojen. Ensinnäkin tarvittavat tiedot on jaettu kahteen eri luokkaan, jotka ovat Win32_NetworkAdapter ja Win32_NetworkAdapterConfiguration. Jälkimmäisestä luokasta löytyy tiedot verkkokortin konfiguraatioista, muttei itse verkkokortin tietoja, jotka taas löytyvät tuosta ensimmäisestä. Lisäksi ajettaessa näitä komentoja, palautetaan paljon tietoa eri adaptereista, jotka eivät ole laitteen fyysisiä verkkokortteja, joita skriptin tarkoitusta varten yritetään selvittää. Selviteltyäni asiaa jonkin aikaa löysin tavan, jolla käytössä oleva verkkokortti saadaan rajattua hakuun. Tuo onnistuu asettamalla filterin - *Filter 'ipenabled= "true"'* hakuun, jolloin palautetaan ainoastaan verkkokortti, jolla on IP/TCP-protokolla sallittuna eli käytössä oleva verkkokortti. Tässäkin tuli kuitenkin vastaan yksi ongelma: tuo ominaisuus löytyy ainoastaan luokasta Win32_NetworkAdapterConfiguration, mutta ei toisesta verkkokorttiluokasta. Tutkittuani luokkia huomasin, että näiden palauttamissa tiedoissa oli yhdistävänä tekijänä index-ominaisuus. Korjasin ongelman niin, että hain skriptiin ensin tiedot konfiguraatioluokasta käyttäen edellä mainittua filteriä ja talletin arvot muuttujaan. Tämän jälkeen asetin verkkokorttiluokan hakuun rajauksen, jolla haetaan luokan arvot ainoastaan niiden laitteiden osalta, joille index-arvo

löytyy tiedoista, jotka on talletettu tuohon konfiguraatio luokan muuttujaan. Näin sain rajattua mukaan vain käytössä olevan verkkokortin tiedot, joista kerättiin mukaan verkkokortin nimi, adapterin tyyppi ja valmistaja. Konfiguraatioluokasta taas kerättiin DNS domain -tiedot, IP-osoitteet, oletusyhdykäytävä, aliverkon osoite sekä MAC-osoite.

4.3.6 Get-NetAdapter

Edellä mainitun kahden luokan lisäksi törmäsin selvityksen aikana vielä komenttoon *Get-NetAdapter*, jolla voidaan myös hakea tietoa verkkokorteista. Tästä komennosta löytyi kätevä vipu *-Physical*, jonka avulla saadaan suoraan haettua laitteen fyysiset verkkokortit. Tämän lisäksi löytyy vielä status ominaisuus, joka kertoo, onko verkkokortti käytössä. Komennolla löytyy vastaavia tietoja kuin aiemmin esitellyistä verkkokorttien CIM-luokista, mutta näiden välillä on myös joitain eroavaisuuksia. Laitteen ja reitittimen välisen yhteyden nopeus selvisi suoraan tällä komennolla ja tuota tietoa ei löytynyt ainakaan näistä kahdesta CIM-luokasta. Lisäsin skriptiin mukaan nopeustiedon käyttäen tätä komentoa.

4.3.7 Muita luokkia

Näiden yleisimpien ja itselle tuttujen luokkien lisäksi selvittelin erilaisia skriptin tarkoitukseen sopivia luokkia, joiden kautta sain selvitykseen olennaisia tietoja laitteesta. Osa tuli vastaan selvitellessäni muita asioita, mutta kätevä apu selvityksessä oli komento *Get-CimClass*, jonka avulla pystyi helposti hakemaan erilaisia CIM-luokkia syöttämällä osan nimeä, esimerkiksi komennon *Get-CimClass -ClassName *video** avulla löytyi luokka *Win32_VideoController*, josta saatiin näytönohjaimen tietoja. Muita tarpeellisia tietoja löytyi muun muassa luokista *Win32_BIOS*, *Win32_Processor* ja *Win32_LogicalDisk*. BIOS:in osalta haettiin tietoihin mukaan nimi, valmistaja, BIOS:n versio sekä sarjanumero. *Win32_Processor*-luokasta saatiin tarvittavat tiedot laitteen prosessorista, kuten prosessorin merkki ja malli, prosessorin tila, ydinten määrä, revisio sekä prosessorin tämänhetkinen nopeus megahertseinä. Prosessorin tila ilmoittaa normaalitilassa viestin OK, mutta esimerkiksi virhetilassa saadaan Error-ilmoitus. Luokasta *Win32_VideoController* otettiin vielä näytönohjaimen tiedot, kuten merkki, malli, versio sekä valmistaja.

Luokista löytyi paljon muutakin tietoa, jota skriptiin olisi voitu ottaa mukaan, mutta tarkoitus oli antaa käyttäjälle selkeä raportti käytössä olevasta laitteesta, sen osista ja käyttöjärjestelmästä. Tietojen määrää rajattiin vain olennaisimpiin, jotta raportti olisi mahdollisimman selkeä eikä vaatisi käyttäjän osalta tarkempaa tietämystä eri luokkien ominaisuuksista ja niiden arvojen merkityksestä. Skriptiä voi helposti jatkossa työstää lisäämällä siihen muita tarvittavia ominaisuuksia CIM-luokista.

4.4 Get-Volume

Lähdin aluksi selvittämään levyjen tietoja käyttäen CIMInstance-luokkaa Win32_LogicalDisk. Luokasta löytyi varsin hyvin tietoa, mutta tutkiessani tarkemmin luokan sisältämiä tietoja ja mitä halusin skriptiini lisätä, törmäsin komentoon *Get-Volume*. Tällä komennolla saa kätevästi haettua tietoa joko kaikista laitteen levyistä tai yksittäisistä syöttämällä levyn kirjaintunnisteen. Tämä vaikutti skriptiä varten sopivammalta, koska halusin antaa käyttäjälle mahdollisuuden valita, tarkistetaanko tiedot kaikkien levyjen osalta vai ainoastaan jonkin tietyn levyn. Lisäsin skriptiin erillisen osion, jossa kysytään käyttäjältä minkä levyn tietoja hän haluaa tarkistella ja syöttämällä enter haetaan kaikki levyt. Käyttäjän syöttäessä levyn tunnisteen, jollaista ei koneesta löydy, kysytään tunnistetta uudelleen niin kauan, kunnes käyttäjä syöttää jonkin oikean tunnisteen. *Get-Volume* palauttaa tietoja muun muassa levyn nimestä, levyn tiedostojärjestelmä tyypistä, levyn tyypistä sekä tilatiedon levyn kunnosta, joka normaalitilassa on "Healthy" eli hyväkuntoinen. Toinen levyn kunnosta informoiva tieto on "Operational status", joka on normaalitilassa OK, mutta tila kertoo suoraan, jos kyseisen levyn kohdalla havaitaan jotain ongelmaa. Esimerkiksi tilan ollessa "Dead", voi levy olla hajonnut tai se voi olla poistettu käyttäjän toimesta. Näiden lisäksi löytyy vielä tiedot levyn koosta sekä siitä, kuinka paljon levytilaa on jäljellä tavuina. Halusin kuitenkin vielä lisätä tiedon, kuinka paljon levytilaa on jäljellä prosentteina, ja tämä onnistui luomalla kustomoitu ominaisuus, joka laskee koon prosentteina käyttäen komennolla saatavia arvoja levyn koosta ja vapaana olevasta levytilasta. Tulos käännetään vielä prosenteiksi käyttäen jälleen ToString-metodia parametrilla "P". Lisättyäni tuon huomasin kuitenkin, että valittaessa select-komennolla näytettävät arvot käännetään tila ja vapaa tila -arvot tavuiksi, jolloin käyttäjälle näkyvät arvot eivät ole enää helposti luettavassa muodossa. *Get-Volume*-komentoon on ilmeisesti jo sisäl-

lytetty tuo käännös tavuiksi, joka kuitenkin select-komentoa käytettäessä häviää. Korjauksena tein vielä näille molemmille arvoille oman ominaisuutensa, jossa tehdään tuo käännös gigatavuiksi.

Edellä mainitun Win32_LogicalDisk-CIM-luokan lisäksi törmäsin muutamaa muuhun luokkaan, kuten Win32_DiskDriveen sekä erilaisiin komentoihin, joilla saadaan haettua levyjen tietoja järjestelmästä. Komennolla *Get-Disk* löytyy tietoa kaikista käyttöjärjestelmälle näkyvistä levyistä. Komennolla saadaan haettua muun muassa levyjen mallit sekä sarjanumerot, mutta tietoja ei ole yksilöity levyn tunnisteeseen tai levyn nimen mukaan vaan ainoastaan juoksevalla numeroinnilla. Toinen hyvä komento on *Get-PhysicalDisk*, jolla saadaan suoraan rajattua ainoastaan järjestelmästä löytyvät fyysiset levyt. Komennolla löytyy myös muuta käytännöllistä tietoa, kuten ominaisuus "MediaType", joka kertoo, onko käytössä oleva levy tyypiltään HDD vai SSD. Tälläkään komennolla ei ole tietoja yksilöity levyn mukaan, vaan tiedot on yksilöity DeviceId-ominaisuudella, joka toimii myös juoksevalla numeroinnilla. Koska *Get-Volume*-komennolla saadaan suoraan haettua tarpeellisimmat tiedot levyistä, kuten levyjen kunto, päätin käyttää ainoastaan tuota skriptissä. Lisäksi komento mahdollistaa tietojen haun levyn kirjaintunnisteella, mikä on käyttäjän kannalta yksinkertaisin tapa hakea tietoja ja mahdollistaa myös tietojen haun vain yksittäisen levyn osalta.

5 PowerShellin jatkokehitys

5.1 PowerShell Linuxille

PowerShellin käyttö on ollut alusta alkaen sidoksissa Windows-käyttöjärjestelmiin johtuen .NET Framework:stä, jota ei ole saatavilla muille käyttöjärjestelmille. Tähän tuli kuitenkin muutos vuonna 2014, kun Microsoftin toimitusjohtajaksi valittiin Satya Nadella, joka alkoi ajamaan yhtiön suuntaa yhä enemmän Linux-ystävällisemmäksi. Tarkoituksena oli mahdollistaa Microsoftin työkalujen käyttö laajemmin eri käyttöjärjestelmissä ja siirtyä kohti avointa lähdekoodia, jolla saadaan koko yhteisö mukaan kehitykseen. Tämän uuden ajattelutavan pohjalta syntyi lopulta .NET Core. NC on avoimeen lähdekoodiin perustuva kehitysalusta, jota ylläpidetään yhteistyössä Microsoftin ja .NET-yhteisön toimesta. Toinen merkittävä ero NF:ään nähden on sen riippumattomuus alustasta, joka

mahdollistaa käytön esimerkiksi Linux- ja macOS-käyttöjärjestelmissä. NC mahdollisti viimein PowerShellin kehityksen aloittamisen myös muille käyttöjärjestelmille ja tästä alkoi PowerShell Core 6.0 -version kehitys. (Microsoft 2016c, Microsoft 2018i.)

5.2 Windows PowerShell ja PowerShell Core

18.8.2016 julkaistiin PowerShell Core 6.0 alpha -versio, joka oli saatavilla käyttöjärjestelmille Red Hat, Ubuntu, Centos ja Mac OS X. Versio oli yhteisön tukema ja Microsoft ilmoitti julkaisun yhteydessä tuovansa virallisen Microsoftin PowerShell Core -version myöhemmin, ja sen julkaisuajankohta tulisi riippumaan yhteisön panoksesta ja liiketoiminnan tarpeesta Linux-versiolle. (Microsoft 2016c.)

Alpha-version julkaisun yhteydessä ei kuitenkaan vielä mainittu, mikä suunnitelma Microsoftilla on versioiden kehityksen osalta, ja aiotaanko tulevaisuudessa jatkaa kahden eri PS-version kehitystä. Microsoft julkaisi 14.7.2017 tiedotteen, jossa avattiin tulevaisuuden näkymiä ja etenemissuunnitelmaa PS:n osalta. Tavoitteena oli kehittää PowerShell Coresta mahdollisimman yhteensopiva Windows PS:n kanssa ja varmistaa, että PS:n kieli sekä sisäänrakennetut moduulit pysyisivät mahdollisimman samankaltaisina aiempiin versioihin nähden. Tuohon aikaan oli julkaistu myös beta-versio PSC:stä, johon ei kuitenkaan ollut vielä portattu kaikkia merkittävimpiä moduuleita aiemmista versioista. Tästä huolimatta jo tuolloin mainittiin, että suuren osan näistä moduuleista pitäisi toimia normaalisti suoraan PSC:llä ilman mitään suurempia muutoksia. Julkaisussa myös ilmoitettiin, että PSC toimii täysin rinnakkain WPS:n kanssa eikä sen asennus aiheuta ongelmia omien vanhojen skriptien toiminnassa. Merkittävin tieto tuossa julkaisussa liittyi kuitenkin WPS:n tulevaisuuteen. Julkaisussa mainittiin, että versio 5.1 tulee jatkossakin olemaan osa Windows 10- ja Windows Server 2016 -käyttöjärjestelmiä, ja sen tuki tulee jatkumaan normaalisti. Versiolle ei kuitenkaan luultavasti olisi enää jatkossa tulossa mitään suuria muutoksia, kuten uusia ominaisuuksia eikä myöskään merkityksellimpien bugien korjauksia. PSC:n osalta taas tullaan aktiivisesti selvittämään ja korjaamaan aiemmista versioista löytyneitä bugeja. Julkaisu aiheutti paljon kysymyksiä liittyen etenkin WPS:n tulevaisuuteen, sillä julkaisussa ei mainittu mitään uudesta Windows PS 6.0-versiosta. Microsoftilta pahoiteltiin epäselvää julkaisua ja kerrottiin, ettei

heillä ole mitään suunnitelmia uudelle WPS-versiolle, ja PSC-version kehityksessä pyritään siihen, ettei uudelle WPS-versiolle myöskään olisi jatkossa enää tarvetta. (Microsoft 2017c.)

5.3 PowerShell Coren tunnistetut ongelmat

Microsoft teki beta-version julkaisun jälkeen listan alpha- ja beta-versioiden tunnetuista ongelmista ja pyysi myös yhteisöltä kommentteja, miten erilaisia ongelmia haluttaisiin ratkaista. Yksi tällainen ongelma, johon kaivattiin yhteisöltä mielipiteitä, oli Linux/macOS-järjestelmien peruskomentojen aliakset, jotka oli poistettu käytöstä. Yhteisöltä haluttiin mielipiteitä, pitäisikö nämä komennot palauttaa vai ei. Yksi merkittävä ero Windowsin ja muiden käyttöjärjestelmien toiminnassa oli merkkikokoriippuvuus. Windows ei ole riippuvainen merkkien koosta, ja tämä on periytynyt myös PS:ään. UNIX-sukuiset käyttöjärjestelmät kuitenkin ovat, joten PS:n on tästä syystä suunniteltu noudattavan käyttöjärjestelmän toimintaa. Tämä aiheuttaa eron PSC-versioiden välillä käytettäessä eri käyttöjärjestelmiä. UNIX-sukuisissa järjestelmissä skriptin ajo epäonnistuu skriptin yrittäessä ladata moduulia, jonka nimen kirjankokoja ei ole syötetty oikein. Tämä voi aiheuttaa ongelmaa Windows-järjestelmässä tuotetuille skripteille, joita yritetään käyttää muissa käyttöjärjestelmissä. Lisäksi tab-täydennyksen käyttö ei onnistu, jos esimerkiksi moduulin ensimmäinen kirjain on syötetty pienenä, vaikka sen tulisi olla isolla. Myös etäyhteyksien käytössä oli eroja eri käyttöjärjestelmien välillä. Tuolloin mainittiin, että SSH-protokollan kautta toimivat etäyhteydet ovat toiminnassa eri alustoilla, mutta protokolla ei kuitenkaan vielä ollut virallisesti tuettu. Julkaisussa ilmoitettiin myös lista erilaisista yleisistä cmdlet-komennoista, jotka eivät vielä toimi Linux/macOS-järjestelmissä, kuten *-Service cmdletit. (Microsoft 2018j.)

5.4 PowerShell Core -versiot

5.4.1 PowerShell Core 6.0

Microsoft julkaisi 10.1.2018 ensimmäisen virallisen Core-version PowerShell Core 6.0. Suunta oli edelleen sama kuin heinäkuun julkaisun aikaan eli Windows PowerShellin

osalta ilmoitettiin vain, että versiosta 2.0 eteenpäin tuki jatkuu, mutta uusien ominaisuuksien tuomiseksi ei ole mitään suunnitelmia. PSC 6.0 oli virallisesti tuettu useille eri käyttöjärjestelmille, jotka Windowsin osalta olivat Windows 7, 8.1 ja 10 sekä Windows Server 2008 R2, 2012 R2 ja 2016. Muita tuettuja käyttöjärjestelmiä olivat Red Hat Enterprise Linux 7, Ubuntu 14.04, 16.04, ja 17.04, Debian 8.7+ ja 9, CentOS 7, OpenSUSE 42.2, Fedora 25 ja 26 sekä macOS 10.12+. Näiden lisäksi oli yhteisön toimesta tuotettu päivityksiä muutamille muille käyttöjärjestelmille, mutta nämä eivät kuuluneet virallisesti Microsoftin tuen piiriin. (Microsoft 2018k.)

Kuten aiempien PowerShell-versioiden kohdalla, oli PowerShell Coren suunnittelussa pyritty siihen, että versio olisi mahdollisimman yhteensopiva vanhempien Windows PS-versioiden kanssa. Uusi versio oli kuitenkin rakennettu .NET Coren päälle, joten tästä johtuen kaikkia ominaisuuksia ei pystytty enää tukemaan. WPS-versiossa 3.0 lisätty workflow-ominaisuus päätettiin jättää pois, koska NC:stä ei löytynyt tälle tarpeeksi vahvaa tukea. Suunnitelmissa oli kuitenkin lisätä PSC:hen tulevaisuudessa vastaavanlainen ominaisuus, joka mahdollistaa tehtävien samanaikaisen ajon. PS Snap-in ominaisuutta ei myöskään lisätty coreen, koska ominaisuus oli jo pitkälti korvattu moduuleilla, eikä PS-yhteisössä ollut enää juurikaan käyttöä tälle. Toinen vastavanlainen vanhempi poisjätetty ominaisuus oli WMI-cmdlet-komennot. Nämä jätettiin pois, koska kahden WMI-moduulin ylläpito koettiin haastavaksi ja uudemmilla CIM-cmdlet-komennoilla pystyy tekemään samoja asioita kuin WMI-cmdleteilla. Tällä on kuitenkin merkittävä vaikutus käyttäjien olemassa oleviin skripteihin, joissa monessa on vielä käytössä WMI-cmdletteja, jotka eivät enää PSC-versiossa 6.0 toimi. (Microsoft 2018k, Microsoft 2018l.)

Uuteen versioon tehtiin paljon muutoksia, jotta PS toimisi paremmin myös ei-Windows-käyttöjärjestelmillä. Kuitenkin osa näistä muutoksista oli sellaisia, jotka vaikuttavat PS:n toimintaan myös Windows-käyttöjärjestelmässä. Powershell.exe uudelleennimettiin pwsh.exe:ksi, jotta erotetaan WPS- ja PSC-versiot paremmin. Joidenkin yksittäisten muuttujien toiminta muuttui hieman, kuten \$ErrorActionPreference-muuttujan. Toimintaa muutettiin niin, että -Verbose- ja -Debug-parametrien kohdalla ei enää ylikirjoiteta muuttujalla asetettua virheidenkäsittelyn preferenssiä, vaan toimitaan kuten muuttujalla on asetettu toimittavan virhetilanteissa. Parametri -ComputerName poistettiin kokonaan *-Computer ja *-Service cmdleteista. Muutos tehtiin, koska RPC-etäyhteyksien toiminnassa oli ongelmia Core-versiossa, ja vaikka nämä Windows-versiossa toimivat, haluttiin

uudesta versiosta tehdä mahdollisimman yhtenäinen kaikille alustoille. Vastaavanlaisia pieniä kielen, cmdlettien ja ohjelmistorajapinnan muutoksia oli uudessa versiossa melko paljon johtuen juuri siitä, että uusi versio haluttiin saada mahdollisimman yhtenäiseksi huolimatta käytössä olevasta laitteesta ja järjestelmästä. (Microsoft 2018l.)

Pääosin versio sisälsi aiemmista versioista löytyneiden moduulien porttausta uuteen versioon sekä muutoksia, jotka parantavat PS:n toimintaa kaikilla alustoilla. Versioon tehtiin kuitenkin myös joitain lisäyksiä, kuten PS-etäyhteys protokollaan, joka aiemmin toimi WinRm:n kautta. Etäyhteyksiin lisättiin tuki autentikoitaessa SSH-protokollan kautta. Toimiakseen tämä vaatii, että SSH on asennettuna kaikille käytettäville laitteille, ja lisäksi SSH-palvelin tulee määritellä luomaan SSH-alijärjestelmä, joka ylläpitää PS-prosessia etälaitteella. Versiossa lisättiin myös kolme uutta cmdlettia, jotka ovat Get-Uptime, Remove-Alias ja Remove-Service. Lisäksi moniin olemassa oleviin cmdlet-komentoihin lisättiin parametreja helpottamaan toimintaa. Tiedostojärjestelmään tehtiin muutoksia, jotka mahdollistavat Linux- ja macOS-järjestelmissä tunnetun merkistön käyttötavan, jota ei Windows-järjestelmissä ole tyypillisesti tuettu. Esimerkiksi tiedostojen ja kansioden nimessä pystyi nyt käyttämään kaksoispistettä ja skriptien nimissä pilkkua. (Microsoft 2018n, Microsoft 2018m.)

5.4.2 PowerShell Core 6.1

Viimeisin PS-versio 6.1 julkaistiin 13.9.2018. Edeltäjänsä tavoin oli tämäkin versio pyhitetty pääosin Windows PS:n moduulien toimivuuden parantamiseksi Core-versiossa eikä niinkään uusien ominaisuuksien lisäämiselle. Versiossa lisättiin jopa yli 1900 cmdlet-komentoa, jotka aiemmin eivät PSC-versiossa toimineet. Käyttöjärjestelmien osalta tukea lisättiin jo aiemmin tuettujen lisäksi järjestelmille Ubuntu 18.04, OpenSUSE 42.3 ja Fedora 27/28. (Microsoft 2018o.)

Merkittävä muutos versioon 6.0 nähden oli PS:n suorituskyvyn parantaminen. Jo versiossa 6.0 tehtiin joitain suorituskyvyn parannuksia versioon 5.1 nähden, mutta myös joidenkin toimintojen suoritus aika piteni siirryttäessä uudempaan versioon. Nyt kuitenkin 6.1-versiossa useiden eri toimintojen suoritus aika lyheni merkittävästi. Esimerkiksi Group-Object käyttö nopeutui versioiden 5.1 ja 6.0 välillä noin 22 % ja nopeutui vielä entisestään 6.0- ja 6.1-versioiden välillä jopa 66 %. Yhtenä esimerkkinä annettiin myös

Import-Csv ja testauksessa importattiin csv-tiedosto, jossa oli noin 27 000 riviä. Testattaessa WPS-versiolla 5.1 meni tuon importtaukseen aikaa 0.441 sekuntia, mutta PSC-versiolla 6.0 kesti importtaus 1.069 sekuntia. Kuitenkin versiossa 6.1 suoritus aika saatiin pudotettua 0.125 sekuntiin, mikä oli merkittävä parannus 6.0-versioon ja myös hyvä parannus 5.1-versioon nähden. (Microsoft 2018p.)

Erilaisten uusien ominaisuuksien testauksen helpottamista varten lisättiin ”Experimental Feature support”, jonka avulla vielä kehitysvaiheessa olevat ominaisuudet voidaan ottaa käyttöön samanaikaisesti toimivien ominaisuuksien kanssa. Tämä ei oletuksena ole käytössä, mutta se voidaan halutessa ottaa käyttöön muokkaamalla konfiguraatitiedostoa. Ominaisuudelle oli paljon kysyntää yhteisössä ja sen uskottiin tehostavan erilaisten uusien ominaisuuksien testausta merkittävästi etenkin kehityksen alkuvaiheissa. (GitHub 2018.) Toinen uusi lisäys versiossa oli tuki käytettävien sovellusten Whitelistaamiseksi. Sovellusten whitelistaus on ominaisuus, jonka avulla voidaan kontrolloida, mitä ohjelmia käyttäjä pystyy suorittamaan. Ominaisuus toimii ”PowerShell Constrained Language” tilassa, joka on PS:n tila, jolla voidaan kontrolloida pääsyä virheherkkiin elementteihin. Ominaisuus oli jo käytössä versiossa 5.1, mutta ei vielä versiossa 6.0. Näiden muutosten lisäksi versiossa tuli paljon muutoksia muun muassa moduuleihin sekä erilaisten komentojen käyttäytymiseen. (Microsoft 2018p.)

Lähteet

- 1 Techopedia. Verkkodokumentti. <<https://www.techopedia.com/definition/1360/commandcom>> Luettu 15.10.2018.
- 2 Computerhope 2018a. Microsoft DOS cmd and command. Verkkodokumentti. <<https://www.computerhope.com/cmd.htm>> Luettu 15.10.2018.
- 3 Computerhope 2018b. Batch File Help and Support. Verkkodokumentti. <<https://www.computerhope.com/batch.htm>> Luettu 15.10.2018.
- 4 Microsoft 2018a. The Windows NT Command Shell. Verkkodokumentti. <<https://technet.microsoft.com/library/cc750982.aspx#XSLTsection126121120120>> Luettu 15.10.2018.
- 5 Computerhope 2017. COMMAND.COM vs. CMD.EXE. Verkkodokumentti. <<https://www.computerhope.com/issues/ch000395.htm>> Luettu 15.10.2018.
- 6 Microsoft 2018b. PowerShell. Verkkodokumentti. <<https://docs.microsoft.com/fi-fi/powershell/scripting/powershell-scripting?view=powershell-6>> Luettu 15.10.2018.
- 7 Snover, Jeffrey 2002. Monad Manifesto. Verkkodokumentti. <<http://www.jsnover.com/Docs/MonadManifesto.pdf>> Luettu 15.10.2018.
- 8 Cleverism. Verkkodokumentti. <<https://www.cleverism.com/skills-and-tools/powershell/>> Luettu 15.10.2018.
- 9 The Man Behind Windows PowerShell 2017. Haastattelu. <<https://www.heavybit.com/library/podcasts/to-be-continuous/ep-37-the-man-behind-windows-powershell/>> Luettu 16.10.2018.
- 10 Microsoft 2018c. Windows PowerShell (Monad) Has Arrived. Verkkodokumentti. <<https://blogs.msdn.microsoft.com/powershell/2006/04/25/windows-powershell-monad-has-arrived/>> Luettu 16.10.2018.
- 11 Warner, Timothy 2015. Understanding the Windows PowerShell Release Cycle. Verkkodokumentti. <<http://www.informit.com/articles/article.aspx?p=2324463>> Luettu 16.10.2018.
- 12 4sysops 2018. Differences between PowerShell versions. Verkkodokumentti. <<https://4sysops.com/wiki/differences-between-powershell-versions/>> Luettu 17.10.2018.

- 13 Microsoft 2017a. Windows PowerShell System Requirements. Verkkodokumentti. <<https://docs.microsoft.com/en-us/powershell/scripting/setup/windows-powershell-system-requirements?view=powershell-6>> Luettu 17.10.2018
- 14 Microsoft 2018d. Windows Management Framework (Windows PowerShell 2.0, WinRM 2.0, and BITS 4.0). Verkkodokumentti. <<https://support.microsoft.com/fi-fi/help/968929/windows-management-framework-windows-powershell-2-0-winrm-2-0-and-bits>> Luettu 17.10.2018.
- 15 Microsoft 2016a. Writing a Windows PowerShell Module. Verkkodokumentti. <<https://docs.microsoft.com/fi-fi/powershell/developer/module/writing-a-windows-powershell-module>> Luettu 17.10.2018.
- 16 Microsoft a. Start-Job. Verkkodokumentti. <<https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/start-job?view=powershell-6>> Luettu 18.10.2018
- 17 Honeycutt, Jerry 2012. Introduction of Windows 8 Introducing Windows 8: An Overview for IT Professionals. E-kirja 33-34.
- 18 Microsoft 2018e. About PSSessions. Verkkodokumentti. <https://docs.microsoft.com/fi-fi/powershell/module/microsoft.powershell.core/about/about_pssessions?view=powershell-6&viewFallbackFrom=powershell-Microsoft.PowerShell.Core> Luettu 18.10.2018.
- 19 Microsoft 2012. Windows Management Framework 3.0 Verkkodokumentti. <<https://www.microsoft.com/en-us/download/details.aspx?id=34595> > Luettu 19.10.2018.
- 20 Microsoft 2018f. Windows PowerShell Workflow Concepts. Verkkodokumentti. <<https://docs.microsoft.com/en-us/system-center/sma/overview-powershell-workflows?view=sc-sma-1807>> Luettu 19.10.2018.
- 21 Microsoft 2015. Windows Management Framework 4.0. Verkkodokumentti. <<https://www.microsoft.com/en-us/download/details.aspx?id=40855>> Luettu 19.10.2018.
- 22 Powershell.org 2013. Microsoft announces PowerShell v4, DSC. Verkkodokumentti. <<https://powershell.org/2013/06/04/microsoft-announces-powershell-v4-dsc/>> Luettu 19.10.2018.
- 23 Microsoft 2016b. Windows Management Framework (WMF) 5.0 RTM packages has been republished. Verkkodokumentti. <<https://blogs.msdn.microsoft.com/powershell/2016/02/24/windows-management-framework-wmf-5-0-rtm-packages-has-been-republished/>> Luettu 20.10.2018.

- 24 Woshub 2017. Using PowerShell PackageManagement In Windows 10. Verkkodokumentti. <<http://woshub.com/using-powershell-packagemanagement-in-windows-10-2016/>> Luettu 20.10.2018.
- 25 Microsoft 2017b. What's New in Windows PowerShell 5.0. Verkkodokumentti. <<https://docs.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-in-windows-powershell-50?view=powershell-6>> Luettu 20.10.2018.
- 26 Microsoft 2018g. About Classes. Verkkodokumentti. <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_classes?view=powershell-6> Luettu 21.10.2018.
- 27 Microsoft 2018h. about_Windows_PowerShell_5.1. Verkkodokumentti. <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_windows_powershell_5.1?view=powershell-5.1> Luettu 5.11.2018.
- 28 Microsoft 2016c. PowerShell is open sourced and is available on Linux. Verkkodokumentti. <<https://azure.microsoft.com/en-us/blog/powershell-is-open-sourced-and-is-available-on-linux/>> Luettu 19.11.2018.
- 29 Microsoft 2018i. .NET Core Guide. Verkkodokumentti. <<https://docs.microsoft.com/en-us/dotnet/core/>> Luettu 19.11.2018.
- 30 Microsoft 2017c. PowerShell 6.0 Roadmap: CoreCLR, Backwards Compatibility, and More! Verkkodokumentti. <<https://blogs.msdn.microsoft.com/powershell/2017/07/14/powershell-6-0-roadmap-coreclr-backwards-compatibility-and-more/>> Luettu 19.11.2018.
- 31 Microsoft 2018j. Known Issues for PowerShell 6.0. Verkkodokumentti. <<https://docs.microsoft.com/en-us/powershell/scripting/whats-new/known-issues-ps6?view=powershell-6>> Luettu 20.11.2018.
- 32 Microsoft 2018k. PowerShell Core 6.0: Generally Available (GA) and Supported! Verkkodokumentti. <<https://blogs.msdn.microsoft.com/powershell/2018/01/10/powershell-core-6-0-generally-available-ga-and-supported/>> Luettu 20.11.2018.
- 33 Microsoft 2018l. Breaking Changes for PowerShell 6.0. Verkkodokumentti. <<https://docs.microsoft.com/en-us/powershell/scripting/whats-new/breaking-changes-ps6?view=powershell-6>> Luettu 21.11.2018.
- 34 Microsoft 2018m. PowerShell Remoting Over SSH. Verkkodokumentti. <<https://docs.microsoft.com/en-us/powershell/scripting/core-powershell/ssh-remoting-in-powershell-core?view=powershell-6>> Luettu 22.11.2018.

- 35 Microsoft 2018n. What's New in PowerShell Core 6.0. Verkkodokumentti. <<https://docs.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-in-powershell-core-60?view=powershell-6>> Luettu 22.11.2018.
- 36 Microsoft 2018o. Announcing PowerShell Core 6.1. Verkkodokumentti. <<https://blogs.msdn.microsoft.com/powershell/2018/09/13/announcing-powershell-core-6-1/>> Luettu 23.11.2018.
- 37 Microsoft 2018p. What's New in PowerShell Core 6.1. Verkkodokumentti. <<https://docs.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-in-powershell-core-61?view=powershell-6>> Luettu 23.11.2018.
- 38 GitHub 2018. Support Experimental Features via Configuration. Verkkodokumentti. <<https://github.com/PowerShell/PowerShell-RFC/blob/master/5-Final/RFC0029-Support-Experimental-Features.md>> Luettu 23.11.2018.