Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

Kyle Snihur
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

*GIAC GCWN Gold Certification*

Author: Kyle Snihur, ksnihur@protonmail.com
Advisor: Jonathan Risto

Accepted: March 31, 2020

Abstract

Security professionals are inundated with alerts, and analysts are suffering alert fatigue with no actionable intelligence (Miliard, 2019). Poor priorities and lack of resources put enterprises at risk (Wilson, 2015). In Windows domains, PowerShell can be used to aggregate data and provide actionable reports and alerts for security professionals continuously. This paper explores the viability of creating an Active Defense PowerShell framework for small to medium-sized organizations to improve security hygiene and posture. The benefits include providing actionable alerts and emails that security professionals can quickly address. Aggregated data can also be used to identify and prioritize holes in an organization's security posture.
1. Introduction

PowerShell can be used to increase security posture, hygiene, and compliance in Windows domains. Scripts can aggregate data and analysts can create actionable reports and alerts based on aggregated data. The benefits of using PowerShell over SCCM or OSquery to collect data are: PowerShell is built into the Windows operating system at no additional charge and not having to install any software agent or patch that could potentially introduce vulnerabilities to enterprise environments (MITRE, 2019). PowerShell is preferred over Windows Management Instrumentation (WMI) because of the ease of writing scripts in PowerShell compared to WMI. PowerShell can also query WMI data using the Get-CimInstance and GET-WmiObject PowerShell cmdlets (Warner, 2016).

The security hygiene and posture framework is used to find insecure Windows service binaries, systems missing installed mandatory security software, compulsory security software that is not running, scheduled tasks stored in a location with insecure NTFS permissions and more. It is possible to configure the framework for alerting automation, as actionable alerts can be generated with detailed information for security professionals to remediate efficiently.

The goal of this paper is to present how organizations can leverage PowerShell 5.1 to aid in the collection of data from all Windows systems within Windows domains. One scheduled task runs on each Windows system; this scheduled task triggers more than 20 PowerShell scripts while outputting the collected data into a network file share for secure aggregation and manipulation of data. The master script in this paper executes various scripts, including the retrieval of: running services, processes, processes by the owner, installed software, local group memberships, installed Linux subsystems, insecure service permissions, scheduled tasks, installed chocolatey packages, the status of mandatory security software, and more. The modularity and ease of adding new scripts create a scalable solution for small to medium-size organizations to improve their security posture and hygiene.
2. PowerShell

PowerShell is a command-line shell and scripting language built on .NET that helps system administrators and power-users rapidly automate tasks that manage operating systems and processes (Microsoft, 2019). Microsoft released PowerShell version 1.0 in November 2006 for Windows XP SP2, Windows Server 2003 SP1, and Windows Vista. In the summer of 2018, Microsoft released PowerShell Core 6.0, which is open-source and cross-platform. It can run on Windows, macOS, and Linux, thereby increasing its adoption (Microsoft, 2018).

Windows Server 2016, Windows Server 2019, and Windows 10 version 1607 and newer have Windows PowerShell 5.1 installed by default while Windows Server 2012 R2 has Windows PowerShell 4.0 installed and Windows Server 2012 has PowerShell version 3.0 (Microsoft, 2019). For the PowerShell framework, only the versions above are supported.

PowerShell is used for both offensive and defensive security in enterprises, and there are significant benefits of using PowerShell for defense (McGuire, 2018). The benefits of using PowerShell in enterprises for defense include: integration with the Windows operating system, ability to access the Win32 application programming interface classes, consistent ability to query for wanted data, and its object-oriented structure (Monogioudis, 2019). The ability to use tools built-in to the Microsoft operating system is preferential over introducing third-party applications that possibly increase the risk of outdated or insecure software being installed.

As PowerShell can query most aspects of a computer system, it can be used for data collection and threat hunting, which is the practice of proactively searching for cyber threats that are lurking undetected in a network (Crowdstrike, 2019). The ability to schedule scripts on all systems while collecting data that security teams care about makes PowerShell an ideal candidate to use the resultant data for domain hygiene and posture purposes while having the ability to threat hunt the resultant data.

During the development of the Security Hygiene and Posture Framework, Visual Studio Code and the PowerShell Integrated Scripting Environment (PowerShell ISE) were used to test

Hoelzer, Daveknihur@protonmail.com
and create scripts within the project. Git was used for version control and should be used to track script changes. All changes should be peer-reviewed before being deployed to production.

3. Active Defense Module

For the Active Defense PowerShell framework, prerequisites are required for the scheduled task to run on all systems and write the output of each script called by the master script to a network file share or a central scripting/log server. The master script runs and triggers multiple subscripts on all systems, and the data can be leveraged for threat hunting purposes to find suspicious activities or artifacts. With PowerShell scripting, one can easily automate processes and set criteria for when to receive alerts, which helps security professionals reduce noise and alert fatigue.

3.1. Prerequisites

3.1.1. Group Policy Object creation

Although there are different ways that the master script can be called, the more straightforward approach is to create a Group Policy Object. Within the Group Policy Object, a scheduled task that runs as NT AUTHORITY\System is created on all endpoints. The scheduled task runs every 15 minutes, and within the master script, there are provisions in place to ensure that the subscripts run only once per day unless forced. This is accomplished by creating a file within the machine's output folder with the date of the last successful completion. The script needs to run with the highest privileges to ensure that all relevant data from the system can be captured and written back to the network share (Figure 1.).
3.1.2. Script server share permissions.

Safeguarding scripts that run on all systems in a domain is needed to ensure integrity, and that unauthorized parties cannot alter scripts for malicious activity, data destruction, or theft. Ideal solutions include monitoring file modifications to the scripts folder using products like Varonis or writing rules in Security Information and Event Management (SIEM) platforms that create real-time alerting. Explicit NTFS permissions reduce the likelihood a malicious attacker can edit the scripts and force endpoints to execute a malicious script.

The script called from the scheduled task is hosted on a file share where domain computers and domain controllers have read and execute permissions on the master script and all sub-scripts called from the master script. The scheduled task arguments below show the path of the master script and its location on a network share (Figure 2.).
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

Figure 2. – Scheduled Task command and arguments

Access to each domain computer output folder, as well as the script folder, should be restricted using NTFS permissions. The below image shows the restricted NTFS permissions on the Computer Results folder, which stores the output of all domain computers in their subfolder. Domain Computers and Domain Controllers can read, write, and execute on the parent folder, as each system must be able to create the computer folder and write script output data to their computer’s subfolder (Figure 3.). Each computer account can only read the data in their machine subfolder and no other subfolders.

<table>
<thead>
<tr>
<th>Name:</th>
<th>E:\Security\ComputerResults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner:</td>
<td>Administrators (LAB-SCRIPT-01\Administrators)</td>
</tr>
<tr>
<td>Permissions</td>
<td>Change</td>
</tr>
<tr>
<td>Access</td>
<td>Full control</td>
</tr>
<tr>
<td>Inherited from</td>
<td>None</td>
</tr>
<tr>
<td>Applies to</td>
<td>This folder, subfolders and files</td>
</tr>
</tbody>
</table>

Figure 3. – Scheduled Task output folder NTFS permissions

Only administrators on the scripting server and explicitly specified users can edit the scripts. Domain Computers and Domain Controllers can read and execute the scripts called within the master script, as each computers’ system account runs the scripts (Figure 4.). Git should be used for version control and as an auditing method for determining which user made

Hoelzer, Daveksnihur@protonmail.com
changes. It can also be used for testing purposes while applying changes to a test Organizational Unit with minimal machines.

![NTFS permissions table]

**Figure 4. – Script folder NTFS permissions**

### 3.1.3. Code signing certificate.

Guidelines in the article "NIST Security Considerations for Code Signing" should be followed in production environments, and all PowerShell scripts code signed to ensure best practices are followed (NIST, 2018). The lab environment Certificate Authority was used to generate the Code Signing Certificate instead of the preferred method of purchasing a code signing certificate from a certificate authority like DigiCert, Comodo, or GoDaddy.

The function Sign-PowerShellCodeScripts is used for code signing (Figure 5.). The pfx certificate path and the script path of the files to be code signed need to be provided. Below is the logic for the code signing function.

---

Hoelzer, Daveknihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

In Figure 6 below, the example shows running the code signing function to complete the initial signing of scripts with the user specifying their code signing certificate password when prompted.

Upon investigation of the Get-MandatorySoftwareStatus.ps1 script, the results show the script is code signed with a signature block at the bottom of the script (Figure 7.). The benefits of using code signing include providing an "identity of the source of the code as well as ensures detection of script modification" (Perez, 2013).

---

Hoelzer, Daveksnihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

3.1.4. Configuration of Settings

A global settings file makes the deployment, configuration, and maintenance of the framework simpler. Domain computers and domain controllers can read the settings file, and only administrators on the script hosting server can edit the settings file. The settings file is JSON-based and converted into a PowerShell object using the ConvertFrom-Json cmdlet (Figure 8.).

```
settings = gc "\lab-script-02\Scripts\GlobalSettings\settings.json" | ConvertFrom-Json
```

Figure 8. – Converting JSON into a PowerShell object

Customization of the settings file allows users to specify their share output folder, mandatory security software, and other global variables. As a result, users do not have to alter multiple variables on various scripts. The output file type variable can be either CSV or JSON and controls the data output of all scripts. The output folder and scripts folder must match the shares where the scripts are hosted as well as the location where all the script output is placed by each domain computer and domain controller. In Figure 9 below, the JSON settings file shows all variables that can be configured.

---

Hoelzer, Daveksnihur@protonmail.com
3.2. Master script walkthrough

There are various ways to run multiple scripts on a given system, but for ease of use and maintenance, a master script triggering subscripts is the preferred way. The following shows the JSON settings file being imported into a PowerShell object. Within the script, there is a check to see if the domain computer has a folder under the output folder. If the computer folder does not exist in the output folder, a subfolder named after the domain computer is created (Figure 10.).

```powershell
$outputFolder = [string]::Concat($settings.OutputFolder, ",", $env:COMPUTERNAME)
if(!((Test-Path $outputFolder))){
    New-Item -Path $outputFolder -ItemType Directory
}
```

Figure 10. – Checking if the computer’s output folder exists

Figure 11 shows the location of the created computer output folder and the folder structure. It is beneficial for security professionals to have all data in a centralized location so that they can quickly search the data and find anomalies with functions included in the project.

Hoelzer, Daveksnihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

![Folder structure](image)

Figure 11. – Output folder structure

The Create-Report function (Figure 12.) is used for formatting the information gathered from each subscript into CSV or JSON format. The filetype variables default is CSV format and can be changed globally by editing the settings file.

```powershell
function create-Report {
    param (
        [String] $filename,
        [ValidateSet("json", "csv")]
    )

    $fileType = "csv",
    $scriptvariable = ,
    $string,
    $outputFolder = ,

    if($fileType -eq "json"){
        invoke-expression scriptvariable | where-object {$_.} | convertto-json
            | Out-File -Force ([String]::Concat($outputFolder,"\",$filename,".",$fileType))
    }elseif($fileType -eq "csv"){
        invoke-expression scriptvariable | where-object {$_.}
            | Export-Csv -Path ([String]::Concat($outputFolder,"\",$filename,".",$fileType)) --NoTypeInformation -Force
    }
}

Figure 12. – Create-Report function used for generating formatted output

In the master script, there is a check to determine if all scripts have run in the last 24 hours. If the last run time file is missing or if the last run time file is greater than 24 hours old, the last run time file is created with the current time, and the master script continues running. If the last run time file is less than 24 hours old, the exit command closes the running Powershell

---

Hoelzer, Daveksnihar@protonmail.com
process. The scheduled task then runs every 15 minutes and exits until the last run time file is greater than 24 hours old (Figure 13.).

```powershell
$lastruntimeFile = [string]::Concat($outputFolder,"\",$settings.LastRunTimeFileName)
if{($Test-path $lastruntimeFile)) {
    New-Item -Path $lastruntimeFile -ItemType file -Force
}
else{
    $lastWriteTime = Get-ChildItem $lastruntimeFile | Select-Object LastWriteTime
    $24hrsAgo = (Get-Date).AddHours(-24)
    if($lastWriteTime.LastWriteTime -lt $24hrsAgo) {
        New-Item -Path $lastruntimeFile -ItemType file -Force
    }
    else{ exit; }
}
```

Figure 13. – Logic used for master script execution

If the last run time file is greater than 24 hours or does not exist, the master script continues and sets variables that are equal to the location for each script to be run (Figure 14.).

```powershell
$processes = (Get-ChildItem $scriptsFolder + "\Modules\Get-processes.ps1")
$services = (Get-ChildItem $scriptsFolder + "\Modules\Get-services.ps1")
$network = (Get-ChildItem $scriptsFolder + "\Modules\Get-Network.ps1")
$TcpConn = (Get-ChildItem $scriptsFolder + "\Modules\Get-TCPConnections.ps1")
$Sdp = (Get-ChildItem $scriptsFolder + "\Modules\Get-OPCConnections.ps1")
$InstalledSoftware = (Get-ChildItem $scriptsFolder + "\Modules\Get-InstalledSoftwareByRegistry.ps1")
$NetworkPath = (Get-ChildItem $scriptsFolder + "\Modules\Get-EnvironmentalVariablePaths.ps1")
$TickleTech2014_AccessibilityFeatures = (Get-ChildItem $scriptsFolder + "\Modules\Get-AccessibilityFeatures.ps1")
$TickleTech2013_AccessibilityFeatures2 = (Get-ChildItem $scriptsFolder + "\Modules\Get-AccessibilityFeatures2.ps1")
$TickleTech2014_PowerShellProfiles = (Get-ChildItem $scriptsFolder + "\Modules\Get-PowerShellProfiles.ps1")
$arp = (Get-ChildItem $scriptsFolder + "\Modules\Get-arp.ps1")
$hosts = (Get-ChildItem $scriptsFolder + "\Modules\Get-hosts.ps1")
$LocalGroupMembership = (Get-ChildItem $scriptsFolder + "\Modules\Get-LocalGroupMembership.ps1")
$InstalledFeatures = (Get-ChildItem $scriptsFolder + "\Modules\Get-installedFeatures.ps1")
$FireFoxExtensions = (Get-ChildItem $scriptsFolder + "\Modules\Get-FirefoxExtensions.ps1")
$ChromeExtensions = (Get-ChildItem $scriptsFolder + "\Modules\Get-ChromeExtensions.ps1")
$LinuxSubsystems = (Get-ChildItem $scriptsFolder + "\Modules\Get-LinuxSubsystems.ps1")
$ProcessOwner = (Get-ChildItem $scriptsFolder + "\Modules\Get-processes.ps1")
$ChocolettePackages = (Get-ChildItem $scriptsFolder + "\Modules\Get-ChocolateyPackages.ps1")
$ComputerSummary = (Get-ChildItem $scriptsFolder + "\Modules\Get-ComputerSummary.ps1")
$InsecurePermissions = (Get-ChildItem $scriptsFolder + "\Modules\Get-InsecurePermissions.ps1")
$RDPUers = (Get-ChildItem $scriptsFolder + "\Modules\Get-RDPUsers.ps1")
$CredentialGuardStatus = (Get-ChildItem $scriptsFolder + "\Modules\Get-CredentialGuardStatus.ps1")
$SecuritySupportProvider_T1101 = (Get-ChildItem $scriptsFolder + "\Modules\Get-SecuritySupportProvider_T1101.ps1")
$ScheduledTasks = (Get-ChildItem $scriptsFolder + "\Modules\Get-ScheduledTasks.ps1")
$MandatorySoftware = (Get-ChildItem $scriptsFolder + "\Modules\Get-MandatorySoftwareStatus.ps1")
$otfixes = (Get-ChildItem $scriptsFolder + "\Modules\Get-hotfixes.ps1")
```

Figure 14. – Setting variables to their script path

Once all the variables are set for each script to be run on each endpoint, the Create-Reports function is executed specifying the output filename, the created script variable, and the output folder. This is the step where the reports are generated and output into the network share, holding all the computer data for each domain computer and domain controller (Figure 15.).

---

Hoelzer, Daveksnihur@protonmail.com
Figure 15. – Create-Report function execution to generate and store the output

Figure 16 below shows the output of scripts triggered within the master script. All files are generated, but a 0kb file means there is no data in the report.

<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChocolateyPackages</td>
<td>2/28/2020 9:00 PM</td>
<td>CSV File</td>
<td>0 KB</td>
</tr>
<tr>
<td>ProcessByOwner</td>
<td>2/28/2020 9:00 PM</td>
<td>CSV File</td>
<td>4 KB</td>
</tr>
<tr>
<td>ChromeExtensions</td>
<td>2/28/2020 9:00 PM</td>
<td>CSV File</td>
<td>0 KB</td>
</tr>
<tr>
<td>FirefoxExtensions</td>
<td>2/28/2020 9:00 PM</td>
<td>CSV File</td>
<td>2 KB</td>
</tr>
<tr>
<td>getInstalledfeatures</td>
<td>2/28/2020 9:00 PM</td>
<td>CSV File</td>
<td>2 KB</td>
</tr>
<tr>
<td>LinuxSubsystems</td>
<td>2/28/2020 9:00 PM</td>
<td>CSV File</td>
<td>0 KB</td>
</tr>
<tr>
<td>NetBIOS</td>
<td>2/28/2020 9:00 PM</td>
<td>CSV File</td>
<td>1 KB</td>
</tr>
<tr>
<td>ArpA</td>
<td>2/28/2020 9:00 PM</td>
<td>CSV File</td>
<td>2 KB</td>
</tr>
</tbody>
</table>

Figure 16. – Folder output

3.2.1. Finding Insecure Service Permissions

The Check-InsecurePermissions function (Figure 17.) gets a list path, folder or file, and service start name. The function is run on every service obtained from Windows services using the Get-WmiObject cmdlet and specifying the win32_service class. All services are filtered to determine if the Authenticated Users, Domain Users, BUILTIN\Users, or Everyone groups have Full Control, Modify, or Write permissions. Services are determined to have insecure service permissions if any of the groups have the specified security principal on the parent folder or the executable of the service.

Hoelzer, Daveksnihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

Running the Get-InsecurePermissions script on machines outputs the insecure service permissions for the computer in the following format (Figure 18.). The output data shows information to help remediate the vulnerability and to stop abuse and privilege escalation.

![Figure 18. - Get-InsecurePermissions script](image)

Hoelzer, Daveknihur@protonmail.com
3.2.2. Check mandatory security software status

Various methods can be used to determine what security software is installed on systems, where the most efficient option (from a performance perspective) is searching through the registry (Dr.Scripto, 2013). The Get-Service PowerShell cmdlet can be used to gather running services on a system; the resultant services are then filtered for the service and service status. In the script, the PowerShell Where-Object is used to determine if the necessary security software is installed.

In the settings file, users can specify the mandatory security software to be monitored. The below example (Figure 19.) shows both Cylance and Carbon Black Defense setup as the necessary security software within the test environment.

The RegistryKeyWord and ServiceName variables are found initially using the helper function Get-RegistryandServiceValuesForMandatorySoftware. Without matches, the software will return as not installed, with no services running. The results below show the output when running the helper function (Figure 20.) for Cylance.
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

The Get-ItemProperty cmdlet can determine if required security software exists by querying the registry. The script then iterates through the mandatory security software set within the settings file and determines if each mandatory security software is installed or missing. The next step in the script is to obtain all services using the Get-Services cmdlet and then correlate them to the security software in the JSON settings file and output based on if the service is running or not (Figure 21.).

```powershell
$settings = Get-Content "C:\ad-script-01\Scripts\GlobalSettings\settings.json" | ConvertFrom-Json
$listOfInstalledSoftware = Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\" -Property "DisplayAppName" -Force
$mandatorySoftware = Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\" -Property "DisplayAppName" -Force
$allServices = Get-Services
$installedServices = Get-MandatorySoftwareStatus.ps1

$masterList = @()
foreach ($service in $allServices)
{
    $installed = Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\" -Property $service.Name -Force
    if ($installedPROPERTY.VALUE -eq $true)
    {
        $masterList += $service
    }
}
```

Figure 21. – Get-MandatorySoftwareStatus.ps1

The Get-MandatorySoftwareStatus.ps1 script is executed from within the master script, forcing execution on all Windows systems in a domain. Figure 22 below shows the output of the script run on an individual server.

![ComputerName Mandatory Software Installed Service_Running](image)

Figure 22. – Output of Get-MandatorySoftwareStatus.ps1

The resulting data can be aggregated and emailed or run on an ad-hoc basis. Only systems where the missing mandatory software or the service for the mandatory software is not running, shows in the aggregated data. The resultant report helps ensure that mandatory security software is both installed and functioning as expected. This report can be automated and sent to the appropriate email distribution list for prompt remediation.

Hoelzer, Daveksnihur@protonmail.com
3.2.3. **Get users running email clients and browsers as administrator.**

Users running browsers and email clients with administrators or elevated permissions are common entry points of attack and need to be monitored. CIS Control 7 states to minimize the attack surface and the opportunities for attackers to manipulate human behaviour through their interaction with web browsers and email systems (CIS, 2019). Privileged accounts should not interact with web browsers or email systems, as user compromise can lead to an attacker obtaining a privileged session.

To determine what processes are running under which user or system account, the Get-CimInstance and Invoke-CimMethod cmdlets can be used (Dr.Scripto, 2015). The below logic shows obtaining all processes and accounts running the operation (Figure 23.).

```powershell
$Processes = Get-CimInstance Win32_Process
$List = @()
foreach($Process in $Processes){
    $Var = Invoke-CimMethod -InputObject $Process -MethodName GetOwner
    $createdObject = New-Object -TypeName psobject
    $createdObject | Add-Member -MemberType NoteProperty -Name ComputerName -Value $env:computername
    $createdObject | Add-Member -MemberType NoteProperty -Name Name -Value $Process.Name
    $createdObject | Add-Member -MemberType NoteProperty -Name Owner -Value $Var.User
    $createdObject | Add-Member -MemberType NoteProperty -Name ProcessId -Value $Process.ProcessId
    $createdObject | Add-Member -MemberType NoteProperty -Name ParentProcessId -Value $Process.ParentProcessId
    $createdObject | Add-Member -MemberType NoteProperty -Name Path -Value $Process.Path
    $List += $createdObject
}
$List
```

*Figure 23. –Get-UserProcesses.ps1*

When Get-UserProcesses.ps1 runs as an administrator or NT Authority\System on an individual system, the output shows all users and the processes which were run as administrator and other privileged accounts (Figure 24.). Although this is a snapshot, it enables security professionals to look for users who are possibly abusing their privileged accounts.
After the master run script has run on all systems, the Send-UnauthorizedAppsRunAsAdmin.ps1 can be used to view the aggregated data on which the user is running monitored applications using an administrator account. Within the script, the monitored application variable is set to all processes monitored. The filtered list returns all users running the specified applications, and if their account names match the naming convention specified in the settings file variable admin account suffix (Figure 25.).

```powershell
$settings = gc "C:\lab-script-01\Scripts\GlobalSettings\settings.json" | ConvertFrom-Json
$OutputFileName = "ProcessByOwner"
$buildoutputPaths = Get-ChildItem ('string':Concat($settings.OutputFolder,"\","$outputFileName", ",",$settings.OutputFileType))
$monitoredApps = $settings.monitoredApps
$finallist = @() $buildoutputPaths | ForEach-Object { if($settings.OutputFileType -eq "csv") {
    $temp = Import-Csv "$_.FullName"
    $temp = $temp | Where-Object {$_._Name -in $monitoredApps } | Select-Object ComputerName,Name,Owner,Path | Where-Object {$_._Owner -like "*$_elevatedAccount*" } | Select-Object ComputerName,Name,Owner,Path | Where-Object {$_._Owner -like "*$_elevatedAccount*" } | Select-Object ComputerName,Name,Owner,Path | Where-Object {$_._Owner -like "*$_elevatedAccount*" }
    $finallist += $data | Where-Object {$_._Owner -like "*$_elevatedAccount*" }
    $finallist
}
```

The Send-UnauthorizedAppsRunAsAdmin script runs and shows which computers, processes, and users are running monitored applications with their privileged accounts (Figure 26.). Security professionals should have conversations with individuals regarding running programs with their privileged accounts and the damage that could potentially be caused if a user with a privileged account clicked on a phishing link or their account was compromised.

Figure 24. – Output of Get-UserProcesses.ps1

Figure 25. – Send-UnauthorizedAppsRunAsAdmin.ps1
3.2.4. Inventory Installed Software and Windows Features

The Center for Internet Security's number two control is to inventory and control software assets. CIS recommends to have a software inventory and method of tracking installed software, so corrections and uninstalls of unauthorized software can occur (Center for Internet Security, 2019). Security professionals should be aware of all software installed on systems and be able to quickly determine installed software, so they can quickly evaluate the security posture when new critical vulnerabilities are released. Determining installed software can be accomplished using the Get-ItemProperty PowerShell cmdlet and retrieving data from the registry on each system (Figure 27.).

Security professionals can use the aggregated data collected from all systems to determine and create a software map or library useful for finding unauthorized application installs. Using the Get-ItemProperty to query, the registry returns the installed software, seen below (Figure 28.).

```
$Software = Get-ItemProperty HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\* ` | Select-Object DisplayName,DisplayVersion,Publisher,InstallDate,UninstallString,InstallLocation
$Software += Get-ItemProperty HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\* ` | Select-Object DisplayName,DisplayVersion,Publisher,InstallDate,UninstallString,InstallLocation
$Software += $Software | Where-Object {([string]::IsNullOrWhiteSpace($_.DisplayName) -eq $false)} ` | Select-Object @([name]::computerName):expression{$_env:COMPUTERNAME}, # | Sort-Object DisplayName
$Software
```

Figure 27. – Get-InstalledSoftwareByRegistry.ps1

Hoelzer, Daveksnihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

The Get-WindowsFeature cmdlet retrieves all installed features on Windows servers, as it only gathers information about roles, role services, and features on Windows Servers (Khanse, 2015). For systems running Windows 10, the Get-WindowsFeature PowerShell cmdlet gathers all installed Windows features. The logic applied is determined by checking whether the system is a Windows Server or is running Windows 10 (Figure 29.).

```powershell
If ($WindowsVersion -notlike "Windows 10") {
    $list = Get-WindowsFeature -where @{installed} -Select @
    Select @
    Select @
    $List = Get-WindowsOptionalFeature -online -where-object @{state -eq "Enabled"}
    foreach($i in $List) {
        $List = Get-WindowsOptionalFeature -online -featurename $i.featurename
        Select @
        Select @
    }
} else {
    Get-WindowsOptionalFeature -online -where-object @{state -eq "Enabled"}
}
```

Figure 29. – Get-InstalledFeatures.ps1

Shown below (Figure 30.) is the output of Get-InstalledFeatures.ps1, which collects all installed Windows features on an individual system.

Figure 30. – The output of Get-InstalledFeatures.ps1

Hoelzer, Daveksnihur@protonmail.com
3.3. Addition of New Scripts and Modules

Users who want to add new scripts to gather output from all systems should follow the below process and test within either PowerShell ISE, Visual Studio Code, or their favourite Integrated Development Editor (IDE). As an example, a script will be created to gather all the hotfixes on the system and relevant data used to determine if patches were recently applied. The first step is to create a script that collects all the relevant data; in the example, we are obtaining all hotfixes installed on a specific system (Figure 31.). Gathering all hotfixes installed on a machine can be accomplished with the Get-Hotfix PowerShell cmdlet.

```powershell
$data = @()
$hotfixes = Get-Hotfix | Select-Object ClassName,Description,HotfixID,InstalledOn,InstalledBy,Caption
    | Sort-Object InstalledOn -Descending
```

![Figure 31. – Creating a new module](image)

The next step is to create a PowerShell object and then assign the collected data from the Get-Hotfix cmdlet into the PowerShell object and add each object to an empty list (Figure 32.). The purpose of creating objects is to have consistency in the output of all scripts for aggregation, hunting, and searching purposes. The first item in every object should be the computer name using the built-in PowerShell variable $env:COMPUTERNAME (Microsoft, 2020).

```powershell
$hotfixes | ForEach-Object { $ret = New-Object PSObject
    $ret | Add-Member -MemberType NoteProperty -Name "ComputerName" -Value $env:COMPUTERNAME
    $ret | Add-Member -MemberType NoteProperty -Name "Description" -Value $_.Description
    $ret | Add-Member -MemberType NoteProperty -Name "Caption" -Value $_.Caption
    $ret | Add-Member -MemberType NoteProperty -Name "HotfixID" -Value $_.HotfixID
    $ret | Add-Member -MemberType NoteProperty -Name "InstalledOn" -Value $_.InstalledOn
    $ret | Add-Member -MemberType NoteProperty -Name "InstalledBy" -Value $_.InstalledBy
    $data += $ret
}
$hotfixes
```

![Figure 32. – Creation of a PowerShell object](image)

Once saved in the modules folder, the newly created script can be added to the master script. It is advised to have a Group Policy Object assigned to testing computers and have a specific folder for alpha script testing. Within the master script, the administrator should create a variable and specify the path of the newly created script (Figure 33.). For example, the generated script was named get-hotfixes.ps1.

```powershell
$hotfixes = ($settings.ScriptsFolder + "\modules\get-hotfixes.ps1")
```

Hoelzer, Daveksnihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

As the newly created script path is specified, it now needs to be run on all systems. The report must then be generated. The Create-Report function outputs the report after specifying the output filename, output file type, output folder, and the script variable path set above.

```
Create-Report -filename hotfixes -fileType $settings.OutputFileType `-scriptvariable $hotfixes -outputFolder $outputFolder
```

Once the master script is saved and the scheduled task runs, the output will be stored in the network share for the computer, which triggered the get-hotfixes.ps1 script. The ideal approach is determining when the last security update was installed on systems and sending a list of outdated systems to the required teams for patching.

3.4. Functions

3.4.1. Forcing Master Script Execution

Analysts need to have the ability to force an endpoint to run the scripts when necessary. Removing the last runtime file for a specified machine will force the master script to execute entirely the next time the scheduled task runs. The scheduled task will execute between zero and 15 minutes, depending on when the file was removed and when the scheduled task runs. Once the LastRunTime file is removed the next time the scheduled task runs, it successfully passes the check that determines where the script was run in the last 24 hours. The below PowerShell function (Figure 35.) can be used to remove the last run file for all machines.

```
function Remove-LastRunForMachineRegex ([regex] $regex) {
  $settings = gc "{shareServer\Scripts\Global\settings\settings.json" | ConvertFrom-Json
  $listOfMachines = (get-childitem $settings.OutputFolder).Name | Where-Object {$_ -match $regex}
  $listOfMachines | ForEach-Object {
    $buildoutputPath = (string):concat($settings.outputFolder,"\$\",$settings.LastRunTimeFilename))
    if(Test-Path $buildoutputPath)
      Remove-Item -Path $buildoutputPath -Verbose
  }
}
```

Figure 35. – Remove-LastRunForAll function

Hoelzer, Daveksnihur@protonmail.com
The Remove-LastRunForMachineRegex function and a supplied regex of (04|05|99) removes the last run file from multiple machines at once, as shown below.

```powershell
PS C:\Security\Scripts\Modules> Remove-LastRunForMachineRegex -regex '(04|05|99)

VERBOSE: Performing the operation Remove File on target "\lab-script-01\ComputerResults\LAB-SCRIPT-04\LastRunTime".
VERBOSE: Performing the operation Remove File on target "\lab-script-01\ComputerResults\LAB-V-99\LastRunTime".
VERBOSE: Performing the operation Remove File on target "\lab-script-01\ComputerResults\WLO-004\LastRunTime".
VERBOSE: Performing the operation Remove File on target "\lab-script-01\ComputerResults\WLO-005\LastRunTime".
VERBOSE: Performing the operation Remove File on target "\lab-script-01\ComputerResults\WLO-099\LastRunTime".
```

Figure 36. – Executing the Remove-LastRunForMachineRegex function

### 3.4.2. Searching Collected Data

Analysts can quickly search all the aggregated data from all computers to find hygiene or posture-related issues in the domain. Security professionals can use the Search-InstalledSoftware function to search for any string within all installed software that was collected using the Get-ItemProperty cmdlet. The below shows a search looking for the dual-use software NMAP (Figure 37.).

![Search-InstalledSoftware searching for nmap](image)

Figure 37. – Search-InstalledSoftware searching for nmap

The ability to determine what features are installed on all computers quickly can help analysts discover if environmental drift is occurring, as well as ensure installed features are not reducing the security posture within the domain. After an environment baseline, the data can be used for metrics and highlight improvements or declines in posture. An analyst's ability to determine what machines have specific features enabled within domains will help quickly remediate vulnerabilities and ensure domain-wide compliance for features that should not be
installed and where organizational drift may be occurring. Using the reporting script Send-InstalledFeatures.ps1 and grouping by the display name shows the total installed feature count and helps analysts find anomalies quickly (Figure 38.).

![Figure 38. – Output using Send-InstalledFeatures and using Where-Object to search for specific features](image)

Using the above output from Send-InstalledFeatures, an analyst can then use the Search-InstalledFeatures.ps1 script to quickly find the machine that has SMBv1 enabled (Figure 39.).

![Figure 39. – Search-InstalledFeatures searching for SMBv1](image)

### 3.5. Automation of Reporting

Security professionals can create and send actionable alerts to a specific email address or distribution list by using the PowerShell cmdlet Send-MailMessage (Yung, 2019) and any of the pre-built reporting functions found in the PostureSettings.ps1 file. For example, if the goal is to automate the insecure service permissions report, a scheduled task can be created on the scripting server with a specified frequency and run time. The below-scheduled task would send an email to the specified address every Tuesday at 8:35 AM with all existing insecure service permissions (Figure 39.).

```xml
<Actions Context="Author">
  <Exec>
    <Command>C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe</Command>
    <Arguments>-executionPolicy RemoteSigned -f "C:\Security\Scripts\Reporting\Send-InsecureServicePermissions.ps1"</Arguments>
  </Exec>
</Actions>
```

Hoelzer, Daveksnihar@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

Figure 39. – Scheduled task to send insecure service permissions

The attached Excel document within the email shows all insecure Windows service permissions. It includes the path of the service, the insecure permissions, and the security principal that should not have any access to the service (Figure 40.).

Figure 40. – Send-InsecureServicePermissions.ps1 generated CSV

The created functions can retrieve the data or send an email. If a user wants to retrieve the data, they can view all functions in the posturefunctions.ps1 file. If a user wants to see all the insecure service permissions, the send-insecureservicepermissions function can be used while specifying the action as ReturnData.

Figure 41. – Send-InsecureServicePermissions function with ReturnData as the argument

If users want to send an automated email, the best approach is to make a report file in the reporting folder and explicitly call the Send-InsecureServicePermissions function while specifying the action as SendEmail, as shown below. Before sending an email, the SMTP from address and SMTP server has to be configured in the JSON settings file.

Hoelzer, Daveksnihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

4. MITRE ATT&CK Scenario

Using the security posture and hygiene framework to search for specific MITRE ATT&CK framework tools, techniques, and artifacts can improve the incident response times of security teams. Multiple tactics will be discussed, along with examples of objects found using the framework.

4.1. Persistence

4.1.1. Security Support Provider (T1101)

The Security Support Provider (SSP) is a set of distributed programming libraries used for authenticated communications (Microsoft, 2018). If attackers can add values to the SSP, they can create persistence. The below output (Figure 42.) shows abnormal registry values set for the Security Support Provider on two machines. If a security professional discovered the below artifact in their environment using the Send-SecuritySupportProvider.ps1 script, an Incident Response Plan should be triggered as the Mimikatz dll obtained credentials in plain text (Metcalf, 2015).

```
PS C:\Windows\system32> E:\Security\Scripts\Reporting\Send-SecuritySupportProvider.ps1

ComputerName PossiblePersistenceAuth ReferenceLink
------------------------- ------------------------ ------------------------
LAB-APP-25 mimilib       https://adsecurity.org/?p=1760
LAB-SCRIPT-01 mimilib    https://adsecurity.org/?p=1760
```

Figure 42. – Send-SecuritySupportProvider.ps1 output

Hoelzer, Daveksnihur@protonmail.com
4.1.2. **Browser Extension (T1176): Auditing installed Google Chrome, Mozilla Firefox, and Internet Explorer browser extensions.**

Collection and monitoring of all browser extensions should be automated, and each app id validated, to ensure they are legitimate applications and not malicious extensions masquerading as legitimate ones. Using the Send-InstalledFirefoxExtensions.ps1 reporting script will show the installed firefox extensions in the test domain (Figure 43.), with the Cisco Webex extension version 1.0.11 having a remote code execution vulnerability (CISCO, 2017). Whitelisting allowed extensions in the script would allow analysts to determine suspicious extensions in less time.

![Figure 43. – Send-InstalledFirefoxExtensions.ps1 output](image)

### 4.2. Privilege Escalation - File System Permissions Weakness (T1044)

Within Windows domains, security professionals frequently see individuals installing business software for work-related purposes. Some software installations improperly set the permissions on the folder and binary executables that can lead to SYSTEM level privilege escalation. Security professionals can proactively determine installed software and where the executable has incorrect permissions (Figure 44.). Analysts can remediate by removing the security principal on the specified folder or file, thereby removing the possible privilege escalation vector.
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

Figure 44. – Send-InsecureServicePermissions with the ReturnData argument

4.3. Defense Evasion - Disabling Security Tools (T1089)

Adversaries and threat actors may disable security tools to avoid possible detection of their tools and activities (Disabling Security Tools, 2017). If security software or logging solutions are disabled along with other methods, interference with security alerting and notifications can occur. The Mandatory Security Software Installed and Running report shows which machines are missing mandatory security software and which devices have the software installed but not running (Figure 45.). Knowing where mandatory security software is not functioning can help security teams ensure they have 100% coverage for compulsory software across their domain on Windows machines. From the data, investigation on all four devices should take place to determine why the software is not running or installed.

Hoelzer, Daveksnihar@protonmail.com
4.4. Command and Control Remote Access Tools (T1219)

Adversaries use legitimate desktop support and remote access tools such as TeamViewer, Go2Assist, LogMeIn, AmmyyAdmin, to establish command and control channels to targeted systems within networks (MITRE, 2019). The ability to search and automate reports targeting specific applications can create efficiencies for security professionals to proactively identify potential dual-use, outdated, or malicious software and create actionable alerts. Cobalt Group has used TeamViewer for maintaining persistence in case persistence was lost while using a Cobalt Strike module (MITRE ATT&CK, 2019). Finding TeamViewer installed in an environment can be accomplished using the Search-InstalledSoftware function (Figure 46.).

Any software can be set up as a daily, weekly, monthly, or quarterly reports using the above function.

---

Hoelzer, Daveknihur@protonmail.com
5. Conclusion

Using PowerShell to increase security posture, hygiene, and compliance in Windows domains is ideal because of the integration and access to most aspects of the Windows operating system. The flexibility and ease of obtaining data can create efficiencies for security professionals while reducing the attack surface compared to introducing third-party solutions and applications.

The ability to gather and collect data on all systems while creating actionable alerts can help organizations improve security hygiene and posture. Analysts can use the aggregated data to threat hunt. Having scheduled reports and alerts can reduce the long term workload for security analysts. The modular approach of the security and hygiene framework gives organizations the flexibility to prioritize where they have determined they are lacking while collecting the relevant data. Key Performance Indicators can be measured, tracked and provided to management and C-Level executives.
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

References


---

Hoelzer, Daveksnihar@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture


---

Hoelzer, Daveksnihar@protonmail.com

6. Appendices (Scripts developed)

6.1. Security Hygiene Master Script

The following script is the SecurityHygiene master script that is triggered by the scheduled task on all domain computers and domain controllers. All scripts will be available at https://github.com/snowbirdflies/Whitepaper_ActiveDefense.

```powershell
$settings = gc '\\lab\script-01\Global\GlobalSettings\settings.json' | ConvertFrom-Json
$outputfolder = [string]:(\settings\OutputFolder\"\$.settings.COMPUTERNAME"

if((\settings\OutputFolder\"\$.settings.LastRunTimeFileName)

$LastrunTimeFile = [string]:(\settings\OutputFolder\"\$.settings.LastRunTimeFileName)

$NewItem -Path $LastrunTimeFile -ItemType File -Forc

$scriptsFolder = (\settings\OutputFolder\"\$.settings.LastRunTimeFileName)

$NewItem -Path $LastrunTimeFile -ItemType File -Forc

$scriptsFolder = (\settings\OutputFolder\"\$.settings.LastRunTimeFileName)

$NewItem -Path $LastrunTimeFile -ItemType File -Forc

$processes = (\settings\scriptsFolder + \"\Modules\Get-Process.ps1\")

$services = (\settings\scriptsFolder + \"\Modules\Get-Service.ps1\")

$dsnCache = (\settings\scriptsFolder + \"\Modules\Get-DNSCache.ps1\")

$tcpConn = (\settings\scriptsFolder + \"\Modules\Get-TCPPort.ps1\")

$udpConn = (\settings\scriptsFolder + \"\Modules\Get-UDPPort.ps1\")

$installedSoftware = (\settings\scriptsFolder + \"\Modules\Get-InstalledSoftwareByRegistry.ps1\")

$envPath = (\settings\scriptsFolder + \"\Modules\Get-EnvironmentVariablePaths.ps1\")

$fireextensions = (\settings\scriptsFolder + \"\Modules\Get-FirefoxExtensions_T1176.ps1\")

$chromeExtensions = (\settings\scriptsFolder + \"\Modules\Get-ChromeExtensions_T1176.ps1\")

$linuxSubsystems = (\settings\scriptsFolder + \"\Modules\Get-LinuxSubsystems.ps1\")

$psUser = (\settings\scriptsFolder + \"\Modules\Get-Process.ps1\")

$chocolateyPackages = (\settings\scriptsFolder + \"\Modules\Get-ChocolateyPackages.ps1\")

$systemFiles = (\settings\scriptsFolder + \"\Modules\Get-SystemFiles.ps1\")

$securityPermiss = (\settings\scriptsFolder + \"\Modules\Get-SecurityPermiss.ps1\")

$imposed = (\settings\scriptsFolder + \"\Modules\Get-Imposed.ps1\")

$mandatorySoftware = (\settings\scriptsFolder + \"\Modules\Get-MandatorySoftware.ps1\")

$hotfixes = (\settings\scriptsFolder + \"\Modules\Get-Hotfixes.ps1\")

Create-Report -filenm processes -filetype $settings.outputFiletype -scriptvariable $processes -outputfolder $outputfolder
Create-Report -filenm services -filetype $settings.outputFiletype -scriptvariable $services -outputfolder $outputfolder
Create-Report -filenm tcpconnections -filetype $settings.outputFiletype -scriptvariable $tcpConn -outputfolder $outputfolder
Create-Report -filenm udpconnections -filetype $settings.outputFiletype -scriptvariable $udpConn -outputfolder $outputfolder
Create-Report -filenm dsnCache -filetype $settings.outputFiletype -scriptvariable $dsnCache -outputfolder $outputfolder
```

Hoelzer, Daveksnihur@protonmail.com

© 2020 The SANS Institute

Author retains full rights.
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

6.2 Functions

The below images are of PostureFunctions.ps1, which contains multiple functions to remove duplicate code within all scripts.

```powershell
$shareServer = "\$shareserver\scripts\GlobalSettings\settings.json" | ConvertFrom-Json

function generateReport ($filename, [string]$fileType, [string]$scriptvariable, [string]$outputFolder)
{
  if($fileType -eq "json")
  {
    Invoke-Expression $scriptvariable | Where-Object {$_.Json} | ConvertTo-Json
    )
  else if($fileType -eq "csv")
  {
    Invoke-Expression $scriptvariable | Where-Object {$_.Csv} | Export-Csv -Path ([string]::Format("\$shareServer\scripts\GlobalSettings\settings.csv", $scriptvariable))
  }
}

function get-ActiveDefenseGlobalSettings
{
  $settings = Get-Content -Path "$shareServer\scripts\alpha\settings.json" | ConvertFrom-Json
  return $settings
}

function get-ActiveDefenseGlobalSettings($key)
{
  $settings = Get-Content -Path "$shareServer\scripts\alpha\settings.json" | ConvertFrom-Json
  return $settings.$key
}
```

Hoelzer, Daveksnihar@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

Hoelzer, Daveksnihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

Hoelzer, Daveksnihr@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

Hoelzer, Daveksnihur@protonmail.com

function search-installed-software([string] $searchstring)
{
    $settings = gc "C:\\lab-script-01\Scripts\GlobalSettings\settings.js" | ConvertFrom-Json
    $outputfile = "$settings.outputPath", "\", $outputfilename, ",", $settings.outputFileType
    $data = @()
    $buildoutputpaths = Get-ChildItem ($settings.outputPath), "\", $outputfilename, ",", $settings.outputFileType
    if($settings.outputFileType -eq "csv")
    {
        $data -eq Import-Csv $settings.include
        $data = $data
    }
    return $data | where-object {$_.match $"searchstring"
}
}

function search-installed-features([string] $searchstring)
{
    $settings = gc "C:\\lab-script-01\Scripts\GlobalSettings\settings.js" | ConvertFrom-Json
    $outputfile = "$settings.outputPath", "\", $outputfilename, ",", $settings.outputFileType
    $data = @()
    $buildoutputpaths = Get-ChildItem ($settings.outputPath), "\", $outputfilename, ",", $settings.outputFileType
    if($settings.outputFileType -eq "csv")
    {
        $data -eq Import-Csv $settings.include
        $data = $data
    }
    return $data | where-object {$_.match $"searchstring"
}
}

function get-installed-features
{
    $settings = gc "C:\\lab-script-01\Scripts\GlobalSettings\settings.js" | ConvertFrom-Json
    $outputfile = "$settings.outputPath", "\", $outputfilename, ",", $settings.outputFileType
    $data = @()
    $buildoutputpaths = Get-ChildItem ($settings.outputPath), "\", $outputfilename, ",", $settings.outputFileType
    if($settings.outputFileType -eq "csv")
    {
        $data -eq Import-Csv $settings.include
        $data = $data
    }
    return $data | where-object {$_.match $"searchstring"
}
}

function search-all-reports([string] $searchstring)
{
    $settings = gc "C:\\lab-script-01\Scripts\GlobalSettings\settings.js" | ConvertFrom-Json
    $outputfile = "$settings.outputPath", "\", $outputfilename, ",", $settings.outputFileType
    $data = @()
    $buildoutputpaths = Get-ChildItem ($settings.outputPath), "\", $outputfilename, ",", $settings.outputFileType
    if($settings.outputFileType -eq "csv")
    {
        $data -eq Import-Csv $settings.include
        $data = $data
    }
    return $data | where-object {$_.match $"searchstring"
}
}

function search-chrome-extensions
{
    $settings = gc "C:\\lab-script-01\Scripts\GlobalSettings\settings.js" | ConvertFrom-Json
    $outputfile = "$settings.outputPath", "\", $outputfilename, ",", $settings.outputFileType
    $data = @()
    $buildoutputpaths = Get-ChildItem ($settings.outputPath), "\", $outputfilename, ",", $settings.outputFileType
    if($settings.outputFileType -eq "csv")
    {
        $data -eq Import-Csv $settings.include
        $data = $data
    }
    return $data | where-object {$_.match $"searchstring"
}
}

function search-firefox-extensions
{
    $settings = gc "C:\\lab-script-01\Scripts\GlobalSettings\settings.js" | ConvertFrom-Json
    $outputfile = "$settings.outputPath", "\", $outputfilename, ",", $settings.outputFileType
    $data = @()
    $buildoutputpaths = Get-ChildItem ($settings.outputPath), "\", $outputfilename, ",", $settings.outputFileType
    if($settings.outputFileType -eq "csv")
    {
        $data -eq Import-Csv $settings.include
        $data = $data
    }
    return $data | where-object {$_.match $"searchstring"
}
}

function search-mandatory-software-status
{
    $settings = gc "C:\\lab-script-01\Scripts\GlobalSettings\settings.js" | ConvertFrom-Json
    $outputfile = "$settings.outputPath", "\", $outputfilename, ",", $settings.outputFileType
    $data = @()
    $buildoutputpaths = Get-ChildItem ($settings.outputPath), "\", $outputfilename, ",", $settings.outputFileType
    if($settings.outputFileType -eq "csv")
    {
        $data -eq Import-Csv $settings.include
        $data = $data
    }
    return $data | where-object {$_.match $"searchstring"
}
}
6.3 Modules

All the below scripts are modules that are run on every system and called within the SecurityHygiene script.

6.3.1 Get-AccessibilityFeatures.ps1

```powershell
$possibleStickykey = @()
$AccessibilityFeaturesFileHashes = @()
$cmdHashes = @()

$AccessibilityFeaturesFileHashes += Get-FileHash -Algorithm SHA256 -Path "C:\Windows\System32\sethc.exe"
$AccessibilityFeaturesFileHashes += Get-FileHash -Algorithm SHA256 -Path "C:\Windows\System32\magnify.exe"
$AccessibilityFeaturesFileHashes += Get-FileHash -Algorithm SHA256 -Path "C:\Windows\System32\Narrator.exe"
$AccessibilityFeaturesFileHashes += Get-FileHash -Algorithm SHA256 -Path "C:\Windows\System32\AtIprodSwitch.exe"

$importedCMDFileHashes = Import-Csv -Path (Settings.CmdFileHash.location + Settings CmdFileHashFileName)

foreach([Hash in $AccessibilityFeaturesFileHashes]){
    if($importedCMDFileHashes.hash -contains $hash.hash){
        $obj = New-Object -TypeName psObject
        $obj | Add-Member -MemberType NoteProperty -Name ComputerName -Value $env:COMPUTERNAME
        $obj | Add-Member -MemberType NoteProperty -Name AccessHash -Value $hash.hash
        $obj | Add-Member -MemberType NoteProperty -Name Path -Value $hash.Path
        $getCMDDosVersion = $ImportedCMDFileHashes | Where-Object {$_._hash -eq $hash.hash}
        $obj | Add-Member -MemberType NoteProperty -Name CMD_os_version -Value $getCMDDosVersion.OperatingSystem
        $obj | Add-Member -MemberType NoteProperty -Name cmdpath -Value $getCMDDosVersion.Path
        $obj | Add-Member -MemberType NoteProperty -Name cmdhash -Value $getCMDDosVersion.hash
        $possibleStickykey += $obj
    }
}
```

6.3.2 Get-AccesibilityFeatures2.ps1

```powershell
$url = "https://www.crowdstrike.com/blog/registry-analysis-with-crowdresponse/"
$smt2 = @{}
$path = "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options"
$smt2 | Add-ChildItem "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options"
	Where-Object {$_._Property -like "Debugger"} | Select-Object PSPATH
foreach($i in $possibleStickykeys){
    $smt2 += (Get-ItemProperty $i.PSPATH)
}
$smt2 += $smt2 | Select-Object @{name="ComputerName";expression={$env:COMPUTERNAME}},
#SchildName,Debugger,PSPATH,a@{name="reference_URL";expression=$url}
```

6.3.3 Get-Arp-a.ps1

---

Hoelzer, Daveksnihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

6.3.4 Get-ChocolateyPackages.ps1

```powershell
$DetailedList = @()
if(Test-Path -Path C:\ProgramData\chocolatey){
    $InstalledPackages = choco list --local-only
    foreach ($i in $InstalledPackages){
        if($i -match "packages installed"){
            $ret = New-Object PSObject
            $ret | Add-Member -MemberType NoteProperty -Name "Computer" -Value $env:COMPUTERNAME
            $ret | Add-Member -MemberType NoteProperty -Name "Version" -Value $Version[0]
            $ret | Add-Member -MemberType NoteProperty -Name "Version" -Value $Version[-1]
            $DetailedList += $ret
        }
    }
} #if
$DetailedList
```

6.3.5 Get-ChromeExtensions_T1176.ps1

```powershell
$sarp = arp -a
$finalList = @()
foreach ($a in $sarp)
{
    $a - $a -replace '^\s*\'', ''
    if($a[0] -ne $null -and $a[2] -ne $null -and $a[2] -ne $null -and $a[0] -ne "Interface:" -and $a[0] -ne "Internet")
    {
        $object = New-Object -TypeName PSObject
        $object | Add-Member -Name 'ComputerName' -MemberType NoteProperty -Value $env:COMPUTERNAME
        $object | Add-Member -Name 'Address' -MemberType NoteProperty -Value $a[0]
        $object | Add-Member -Name 'MAC' -MemberType NoteProperty -Value $a[1]
        $object | Add-Member -Name 'Type' -MemberType NoteProperty -Value $a[2]
        $finalList += $object
    }
} #foreach
$finalList
```
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

Hoelzer, Daveksnihur@protonmail.com

6.3.6 Get-CredentialGuardStatus.ps1

```powershell
Get-CredentialGuardStatus -Credential $Credential
```

© 2020 The SANS Institute, Author Retains Full Rights
© 2020 The SANS Institute, Author retains full rights.
6.3.7 Get-DNSCache.ps1

```powershell
$dnsclientCache = Get-DnsClientCache | Select-Object @{name="ComputerName";expression={$env:COMPUTERNAME}}
$dnsclientCache
```

6.3.8 Get-EnvironmentalVariablePaths.ps1

```powershell
$var = Get-ChildItem env:Path
$list = $var | Select-Object Value, PSDrive,Name
$list = $list.Value -split ";"
$masterList = @()
Foreach($i in $list){
    if($i -ne $null -and $i -ne ""){
        $obj = New-Object -TypeName psobject
        $obj | Add-Member -MemberType NoteProperty -Name ComputerName -Value $env:COMPUTERNAME
        $obj | Add-Member -MemberType NoteProperty -Name EnvironmentalVariable -Value $i
        $masterList += $obj
    }
}
$masterList
```

6.3.9 Get-etcHosts.ps1

```powershell
$var = Get-content c:\windows\system32\drivers\etc\hosts
$finalObject = @()
$ip = (regex):
    if($i -notlike "#" -and $i -notlike $null -and $i -notlike ""){
        $set = New-Object PSObject
        $set | Add-Member -MemberType NoteProperty -Name "ComputerName" -Value $env:COMPUTERNAME
        $set | Add-Member -MemberType NoteProperty -Name "hostnames" -Value $i
        $finalObject + $set
    }
}
```

6.3.10 Get-FireFoxExtensions_T1176.ps1

```powershell
Hoelzer, Daveksnihur@protonmail.com
```
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

6.3.11 Get-HotFixes.ps1

```powershell
$data = @()
$firefoxextensions = Get-ChildItem "C:\users\" | appdata\Roaming\mozilla\Firefox\Profiles" -Recurse -Filter addons.json
foreach($details in Get-Content $_.FullName | ConvertFrom-Json).addons
foreach($ext in $details)
    $ext = New-Object PSObject
    $ext | Add-Member -MemberType NoteProperty -Name ComputerName -Value $env:COMPUTERNAME
    $ext | Add-Member -MemberType NoteProperty -Name User -Value $env:UserName
    $ext | Add-Member -MemberType NoteProperty -Name Creator -Value $ext.creator
    $ext | Add-Member -MemberType NoteProperty -Name Version -Value $ext.version
    $ext | Add-Member -MemberType NoteProperty -Name sourceURI -Value $ext.sourceURI
    $ext | Add-Member -MemberType NoteProperty -Name InstalledPath -Value $_.FullName
    $data += $ext
$data = $data
```

6.3.12 Get-InsecurePermissions_V2.ps1

```powershell
function Get-InsecurePermissions { [CmdletBinding()] Param($Path, $FolderFile, $StartName) $ACL = Get-ACL $Path; $AccessACL = $ACL | Select-Object -Expand Access; ForEach ($Ace in $AccessACL) {
    $SecurityPrincipal = $Ace.IdentityReference;
    $Permissions = $Ace.FileSystemRights.Tostring().Split ":";
    ForEach ($Permission in $Permissions){
        if ($Permission -eq "FullControl") {
            $Object = New-Object PSObject
            $Object | Add-Member -MemberType NoteProperty -Name ComputerName -Value $env:computername
            $Object | Add-Member -MemberType NoteProperty -Name StartName -Value $startName
            $Object | Add-Member -MemberType NoteProperty -Name Path -Value $Path
            $Object | Add-Member -MemberType NoteProperty -Name IsObjectFolder -Value $true
            $Object | Add-Member -MemberType NoteProperty -Name SecurityPrincipal -Value $SecurityPrincipal
            $permissions = Add($Object)
        }
    }
} }
```

Hoelzer, Daveksnihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

6.3.13 Get-InstalledFeatures.ps1

```powershell
if($WindowsVersion -notlike "Windows 10"){
    $list = Get-InstalledFeature | Where { $_.Installed } | Select @Name="ComputerName",DisplayVersion,Name,Installed | Sort-Object DisplayName
}
elseif($WindowsVersion -like "Windows 10"){
    $tempList = Get-WindowsOptionalFeatureOnline | Where-Object { $_.state -eq "enabled" } | Select @Name="ComputerName",DisplayName,FeatureName,InstallLocation,Expression
    $list = $tempList | Select @Name="ComputerName",Expression,DisplayName,FeatureName,InstallLocation,Expression | Sort-Object DisplayName
}
list
```

6.3.14 Get-InstalledSoftwareByRegistry.ps1

```powershell
$Software = Get-ItemProperty HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\* | Select-Object DisplayName,DisplayVersion,Publisher,InstallDate,InstallString,Location
$Software = Get-ItemProperty HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\* | Select-Object DisplayName,DisplayVersion,Publisher,InstallDate,InstallString,Location
$Software = $Software [String]::TrimWhiteSpace($_.displayname) -eq $false | Select-Object @name="ComputerName",expression=($env:COMPUTERNAME),# | Sort-Object DisplayName
```

6.3.15 Get-InterfaceDetails.ps1

```powershell
```

Hoelzer, Daveksnihur@protonmail.com

© 2020 The SANS Institute
Author retains full rights.
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

6.3.16 Get-IPv6Check.ps1

```powershell
$Adapters = Get-NetAdapter
$OutputArray = @()::

if ($Adapters) {
    $AdapterConfigs = Get-CimInstance Win32_NetworkAdapterConfiguration -Filter "IPEnabled = TRUE"
    foreach ($Adapter in $Adapters) {
        $temp = New-Object PSObject
        $tempconfig = $AdapterConfigs | where { $_.InterfaceIndex -eq $Adapter.InterfaceIndex }
        $temp | Add-Member -MemberType NoteProperty -Name "Computer" -Value $Adapter.ComputerName
        $temp | Add-Member -MemberType NoteProperty -Name "DNS" -Value $tempconfig.DNSServerSearchOrder[0]
        $temp | Add-Member -MemberType NoteProperty -Name "DHCP" -Value $Adapter.DHCPEnabled
        $temp | Add-Member -MemberType NoteProperty -Name "IP" -Value $tempconfig.IPAddress[0]
        $temp | Add-Member -MemberType NoteProperty -Name "Gateway" -Value $tempconfig.DefaultIPGateway
        $OutputArray += $temp
    }
    return $OutputArray
}
```

6.3.17 Get-LinuxSubsystems.ps1

```powershell
$Interfaces = (Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter "IPEnabled = TRUE")
$OutputArray = @()::

foreach ($Adapter in $Interfaces) {
    $temp = New-Object PSObject
    $tempconfig = $AdapterConfigs | where { $_.InterfaceIndex -eq $Adapter.InterfaceIndex }
    $temp | Add-Member -MemberType NoteProperty -Name "Computer" -Value $Adapter.ComputerName
    $temp | Add-Member -MemberType NoteProperty -Name "DNS" -Value $Adapter.DNSDomain
    $temp | Add-Member -MemberType NoteProperty -Name "IP" -Value $tempconfig.IPAddress[0]
    $temp | Add-Member -MemberType NoteProperty -Name "Gateway" -Value $tempconfig.DefaultIPGateway
    $OutputArray += $temp
}
```

---

Hoelzer, Daveksnihur@protonmail.com

© 2020 The SANS Institute

Author retains full rights.
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

6.3.18 Get-MandatorySoftwareStatus.ps1

```powershell
# Gets users that are using the Linux subsystem (includes Kali, Ubuntu, Debian, openSUSE)
$var = gc "C:\Users\"AppData\Local\packages\" | Select-Object Fullname, name, @name:={User}; expression {($_.fullname -split \"\"[2])}`
    where-Object {$_-name -like "canonical*" -or $_-name -like "Kalilinux*" -or $_-name -like "openSUSE*" -or $_-name -like "Alpine*" -or $_-name -like "Debian*"}

$detailedList = @()
Foreach($_ in $var){
    $set = New-Object PSObject
    $set | Add-Member -MemberType NoteProperty -Name "Computer" -Value $env:COMPUTERNAME
    $set | Add-Member -MemberType NoteProperty -Name "User" -Value $1.user
    $set | Add-Member -MemberType NoteProperty -Name "Name" -Value $1.Name
    $set | Add-Member -MemberType NoteProperty -Name "path" -Value $1.FullName
    $detailedList += $set
}
$detailedList
```

Hoelzer, Daveksnihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

6.3.19 Get-NetBIOS.ps1

```powershell
Get-NetBIOS -Stop
```

6.3.20 Get-PowerShellProfiles.ps1

```powershell
Get-PowerShellProfiles -Stop
```

6.3.21 Get-Processess.ps1

```powershell
Get-Processess -Stop
```
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

6.3.22 Get-RDPUSers.ps1

```
$rdpUsers = net localgroup "remote desktop users" | where { $_.Name -match "remote desktop users" } | select -skf 4 | select-object @(name = "UserName"; expression = {$env:COMPUTERNAME}), @(name = "Name"; expression = {$_})
```

6.3.23 Get-ScheduledTaskV2.ps1

```
$path = "\" + $env:COMPUTERNAME + "\$\Windows\System32\Tasks"
$taskList = Get-ChildItem -recurse -Path $path -File
foreach ($task in $tasks) {
    $TaskInfo = [xml](Get-Content ("\$\Task.directory.fullname\", $task.Name)))
    $obj = New-Object PSObject -Property (ordered)@{
        ComputerName = $env:COMPUTERNAME
        Task = $task.name
        User = $TaskInfo.Task.Principals.Principal.userId
        Enabled = $TaskInfo.Task.enabled
        Application = $TaskInfo.Task.Actions.exec.command
    }
    $masterList += $obj
}
$masterList = $masterList | Where-Object { $_.arg -notlike "" -and $_.user -notlike "" } | select-object task, user, enabled, application, arg | $masterList
```

6.3.24 Get-SecuritySupportProvider_T1101.ps1

```
$url = "https://adsecurity.org/?p=1760"
$scriptFolder = "C:\Scripts\GlobalSettings\settings.json" | ConvertFrom-Json
$baselineCSV = Import-Csv $scriptFolder "GlobalSettings\SecurityPackages\T1101.csv"
$MasterList = @()
$securityPackages = Get-ItemProperty "HKLM:SYSTEM\CurrentControlSet\Control\LSA" -Name "Security Packages"
if($securityPackages.SecurityPackages.Count -gt 0){
    foreach($sp in $securityPackages.SecurityPackages){
        if($baselineCSV.SecurityPackage.Name -notcontains $sp -and $sp -notlike ""){
            $obj = New-Object PSObject
            $obj | Add-Member -MemberType NoteProperty -Name "ComputerName" -Value $env:COMPUTERNAME
            $obj | Add-Member -MemberType NoteProperty -Name "PossiblePersistenceAuth" -Value $sp
            $obj | Add-Member -MemberType NoteProperty -Name "Reference:sink" -Value "https://adsecurity.org/?p=1760"
        }
    }
    $masterList += $obj
}
$masterList
```
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

6.3.25 Get-Services.ps1

```powershell
Get-Services | Select-Object Name, Status, DisplayName, StartType
```

6.3.26 Get-SharePermissions.ps1

```powershell
Get-SharePermissions
```

6.3.27 Get-TCPConnections.ps1

```powershell
Get-TCPConnections
```

6.3.28 Get-TPMInfo.ps1

```powershell
Get-TPMInfo
```

Hoelzer, Daveksnihur@protonmail.com
Creating an Active Defense PowerShell Framework to Improve Security Hygiene and Posture

Hoelzer, Daveksnihur@protonmail.com

6.3.29 Get-UDPConnections.ps1

```powershell
$var = Get-Tpm
$keyattestation - Get-TpmSupportedFeature
if($keyattestation -ne $null)
{
    $tempv = $keyattestation
    $var = $var | Select-Object @
    name="ComputerName";expression=
    {($env:COMPUTERNAME)}`
    , @
    name="Get-TpmSupportedFeature";expression=
    "$tempv"
}
else
{
    $tempv = "not supported"
    $var = $var | Select-Object @
    name="ComputerName";expression=
    {($env:COMPUTERNAME)}`
    , @
    name="Get-TpmSupportedFeature";expression=
    "$tempv"
}

$var
```

6.3.30 Get-UserProcesses.ps1

```powershell
$processes = Get-Process | Select-Object @
    name="ComputerName";expression=
    {($env:COMPUTERNAME)}`
$List = @()
foreach($process in $processes)
{
    $var = Invoke-CimMethod -InputObject $process -MethodName GetOwner
    $createdObject = New-object -TypeName psobject
    $createdObject = Add-Member -MemberType NoteProperty -Name ComputerName -Value $env:computername
    $createdObject = Add-Member -MemberType NoteProperty -Name Owner -Value $var.Owner
    $createdObject = Add-Member -MemberType NoteProperty -Name ParentProcessId -Value $process.ParentProcessId
    $createdObject = Add-Member -MemberType NoteProperty -Name Path -Value $process.Path
    $List += $createdObject
}
$List
```
## Upcoming SANS Training

Click here to view a list of all SANS Courses

<table>
<thead>
<tr>
<th>Course</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Amsterdam August 2020 Part 1</td>
<td>Amsterdam, NL</td>
<td>Aug 03, 2020 - Aug 08, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS FOR508 Canberra August 2020</td>
<td>Canberra, AU</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Amsterdam August 2020 Part 2</td>
<td>Amsterdam, NL</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Virginia Beach 2020</td>
<td>Virginia Beach, VAUS</td>
<td>Aug 31, 2020 - Sep 05, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>OnlineUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
<tr>
<td>SANS SelfStudy</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>