Service Registry Permissions Weakness Check

- https://github.com/SigmaHQ/sigma/blob/master/rules/windows/powershell/powershell_script/posh_ps_get_acl_service.yml

```
title: Service Registry Permissions Weakness Check
id: 95afc12e-3cbb-40c3-9340-84a032e596a3
status: test
description: |
    Adversaries may execute their own malicious payloads by hijacking the Registry entries
    Adversaries may use flaws in the permissions for registry to redirect from the original
    Windows stores local service configuration information in the Registry under HKLM\SYSTE
references:
    - https://github.com/redcanaryco/atomic-red-team/blob/f339e7da7d05f6057fdfcdd3742bfcf36
    - https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.security/get-
author: frack113
date: 2021/12/30
tags:
    - attack.persistence
    - attack.t1574.011
logsource:
    product: windows
    category: ps_script
    definition: 'Requirements: Script Block Logging must be enabled'
detection:
    selection:
        ScriptBlockText|contains|all:
            - 'get-acl'
            - 'REGISTRY::HKLM\SYSTEM\CurrentControlSet\Services\'
    condition: selection
falsepositives:
    - Legitimate administrative script
level: medium
```

Original Analytic Scoring

	Application (A)	User-mode (U)	Kernel-mode (K)
Core to (Sub-) Technique (5)			
Core to Part of (Sub-) Technique (4)			
Core to Pre-Existing Tool (3)			
Core to Adversary-brought Tool (2)	EventID: 4104 ScriptBlockText|contains|all: - ‘get-acl’ - ‘REGISTRY::HKLM\SYSTEM\CurrentControlSet\Services\’		
Ephemeral (1)			

Improved Analytic Scoring

	Application (A)	User-mode (U)	Kernel-mode (K)
Core to (Sub-) Technique (5)			EventID: 4663

			TargetObject: “*SYSTEM\CurrentControlSet\Se
Core to Part of (Sub-) Technique (4)			
Core to Pre-Existing Tool (3)			
Core to Adversary-brought Tool (2)			
Ephemeral (1)			

This analytic uses the Windows PowerShell logging Event ID 4104 and detects on specific values in the ScriptBlockText field[1]. While the specified registry key is core to the subtechnique[2], the actual observable is the string representation of that registry key inside the script text. It is relatively easy for an attacker to obfuscate keywords or values in a PowerShell script. For example, the cmdlet `get-acl` is defined and included in the Microsoft.PowerShell.Security module, but equivalent functionality can be accomplished with a renamed or custom cmdlet that doesn’t require `get-acl` exist in the script text. The registry key string can be obfuscated in other ways[3], several of which are shown below. Since the adversary can modify their tools and associated scripts before deployment to evade this analytic, it is **2A**.

```
# Let’s start with a simple example:
function Invoke-Malware {
  Write-Host ‘Malware!’;
}

# Simple signature: if script contains “Write-Host ‘Malware’” → Malicious
# Simple bypass:
function Invoke-Malware {
  Write-Host "Malware!";
}

# Simple signature: if re.findall(“Write-Host .Malware.”, script) → Malicious
# Simple bypass:
function Invoke-Malware {
  Write-Host (“Mal” + “ware!”);
}

# Let’s start being a little more sophisticated (just a bit):
function Invoke-NotMalware {
  $malware_base64 = "V3JpdGUtSG9zdCAiTWFsd2FyZSEi";
  $malware = [System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($ma
  IEX ($malware);
}

# Simple signature:
# if script contains “V3JpdGUtSG9zdCAiTWFsd2FyZSEi” → Malicious
# Simple bypass:
function Invoke-NotMalware {
  $malware_base64 = "VwByAGkAdABlAC0ASABvAHMAdAAgACIATQBhAGwAdwBhAHIAZQAhACIA";
  $malware = [System.Text.Encoding]::UNICODE.GetString([System.Convert]::FromBase64String($
  IEX ($malware);
}

# Security solutions are able to emulate base64 decoding
# So malware authors move to algorithm based obfuscation such as XOR:
$key = 0x64
$encodedMalware = "M2QWZA1kEGQBZElkLGQLZBdkEGREZEZkKWQFZAhkE2QFZBZkAWRFZEZk";
$bytes = [Convert]::FromBase64String($encodedMalware)
$decodedBytes = foreach ($byte in $bytes) {$byte -bxor $key}
$decodedMalware = [System.Text.Encoding]::Unicode.GetString($decodedBytes)
IEX ($decodedMalware)
```

A more robust way of detecting the original behavior involves setting a Security Access Control List (SACL) on the registry key. Setting a SACL on the registry key enables using a kernel-mode data source to detect the `get-acl` behavior of a script without looking at the contents of

the script itself. Once the SACL is set and configured, an EventID 4663 will be generated whenever an attempt is made to access the registry key.

Note

SACLs have configuration options which can change the precision of an analytic. One configuration option is to log the “Full Control” set of activity and get a complete view of registry key activity, and then query those results for when the registry key is read (when the `AccessMask` field has the corresponding value `READ_CONTROL`[4]). However, this approach could generate a large amount of benign noise. As an alternative, the SACL can be configured to generate an event only when the key is read.

References

[1] https://github.com/OTRF/OSSEM-DD/blob/5e16ccfe548c8c0249430247a99e213636b2a5a5/windows/etw-providers/Microsoft-Windows-PowerShell/events/event-4104_v1.yml#L22

[2] https://attack.mitre.org/techniques/T1574/011/

[3] https://i.blackhat.com/briefings/asia/2018/asia-18-Tal-Liberman-Documenting-the-Undocumented-The-Rise-and-Fall-of-AMSI.pdf

[4] https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-lsad/5ee8db78-5f0e-47b2-aba7-8447ff454e3b

Previous Next