To create a PowerShell script that searches through every machine on your network and checks for the presence of the Tanium Client, Tenable Client, and Cybereason client installations, you first need get or create a CSV file (clients.csv) with a list of the machines you want to check. The CSV file should have a single column named "ComputerName" containing the names, serial numbers and if needed other important informatio

To integrate NinjaRMM API to the existing PowerShell script, you'll first need to obtain an API key and Base URL from your NinjaRMM account. You can follow the steps in the official documentation to generate an API key: <https://ninjarmm.zendesk.com/hc/en-us/articles/360048497134-Getting-Started-with-the-NinjaRMM-API>

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

An in-depth walk through of the code and what it does:

Text

Description automatically generated

* Define NinjaRMM API key and Base URL:

$NinjaApiKey = "your\_ninjarmm\_api\_key"

$NinjaApiBaseUrl = "your\_ninjarmm\_base\_url"

* These two lines store the NinjaRMM API key and Base URL as variables, which will be used later in the script to interact with the NinjaRMM API.
  + Replace the placeholders with your actual API key and Base URL.
* Define the list of clients to search for:

Text

Description automatically generated

$clientsToSearchFor = @(

@{

Name = "Tanium Client"

ProcessName = "TaniumClient"

},

@{

Name = "Tenable Client"

ProcessName = "NessusAgent"

},

@{

Name = "Cybereason Client"

ProcessName = "CybereasonRaccine"

}

)

* + This part of the script creates an array called $clientsToSearchFor containing three hashtable elements, each representing a client to search for on the machines.
    - Each hashtable has two keys:
      * Name and ProcessName.
      * The Name key contains a human-readable client name, while the ProcessName key contains the corresponding service name that the script will search for on the machines.
* Define the Get-NinjaOnlineDevices function:

Text

Description automatically generated

function Get-NinjaOnlineDevices {

...

}

* + This function is used to query the NinjaRMM API to get a list of online devices.
    - It takes two parameters:
      * $ApiKey and $ApiBaseUrl, which are the NinjaRMM API key and Base URL, respectively.

**Inside the function:**

* Set the API request headers:

Text

Description automatically generated

$headers = @{

"Authorization" = "Basic $ApiKey"

"Content-Type" = "application/json"

}

* The $headers hashtable contains the required headers for the API request, including the Authorization header with the API key and the Content-Type header to indicate that the request and response data will be in JSON format.
* Send the API request:



$response = Invoke-WebRequest -Uri "$ApiBaseUrl/v2.1/devices" -Headers $headers -Method Get

* + This line sends an HTTP GET request to the NinjaRMM API's /v2.1/devices endpoint using the specified Base URL and headers.
    - The API response is stored in the $response variable.
* Parse the response JSON:

Text

Description automatically generated

$devices = $response | ConvertFrom-Json

* + The ConvertFrom-Json cmdlet is used to convert the JSON response data into PowerShell objects.
    - The resulting objects are stored in the $devices variable.
* Filter the devices to get only online ones:



$onlineDevices = $devices | Where-Object { $\_.online -eq $true }

* + This line filters the devices by checking the online property of each device object.
    - If the online property is $true, the device is considered online and included in the $onlineDevices variable.
* Return the online devices:



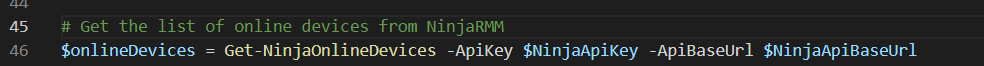
return $onlineDevices

* + This line returns the online devices found by the function to the caller.
* Get the list of online devices from NinjaRMM:



$onlineDevices = Get-NinjaOnlineDevices -ApiKey $NinjaApiKey -ApiBaseUrl $NinjaApiBaseUrl

* This line calls the Get-NinjaOnlineDevices function with the NinjaRMM API key and Base URL, and stores the resulting online devices in the:



$machines = $onlineDevices | ForEach-Object { [PSCustomObject]@{ ComputerName = $\_.name } }

* + This line processes the $onlineDevices array using the ForEach-Object cmdlet.
    - For each online device, it creates a new custom PowerShell object with a single property called ComputerName, which is set to the name property of the device object.
      * The result is a new array called $machines containing objects with the ComputerName property.
* Initialize the results array:

A screenshot of a computer

Description automatically generated with medium confidence

$results = @()

* + This line initializes an empty array called $results, which will be used to store the results for each machine as the script processes them.
* Iterate through each machine:

Text

Description automatically generated

foreach ($machine in $machines) {

...

}

* + This foreach loop iterates through each object in the $machines array.
    - Inside the loop, the script performs a series of steps to determine the presence of the clients on each machine.

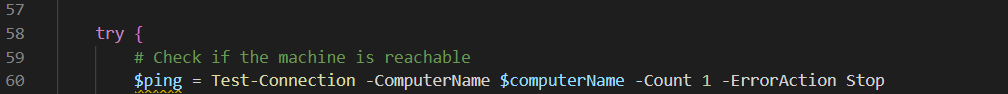
**Inside the loop:**

* Get the computer name:



$computerName = $machine.ComputerName

* + This line retrieves the ComputerName property from the current $machine object and stores it in the $computerName variable.
* Check if the machine is reachable:



$ping = Test-Connection -ComputerName $computerName -Count 1 -ErrorAction Stop

* + The Test-Connection cmdlet is used to check if the machine is reachable by sending a single ping packet.
    - If the machine is not reachable, the script jumps to the catch block to handle the error.
* Get the list of installed services on the machine:



$services = Get-WmiObject -Class Win32\_Service -ComputerName $computerName -ErrorAction Stop

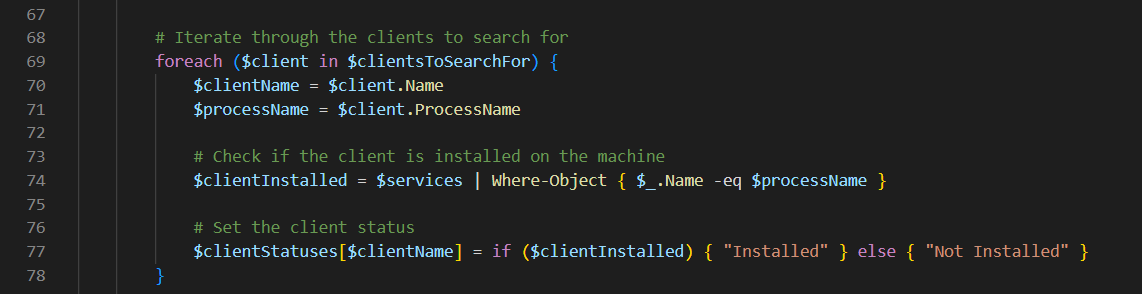
* + This line uses the Get-WmiObject cmdlet to retrieve the Win32\_Service instances from the remote machine, which represent the installed services on the machine.
    - The services are stored in the $services variable.
* Initialize the client statuses:

Graphical user interface, text

Description automatically generated

$clientStatuses = @{}

* + This line initializes an empty hashtable called $clientStatuses, which will be used to store the status of each client on the current machine.
* Iterate through the clients to search for:



foreach ($client in $clientsToSearchFor) {

...

}

* + This nested foreach loop iterates through each client in the $clientsToSearchFor array.
    - Inside the loop, the script checks if the client is installed on the machine and sets the client status accordingly.
* Get the client name and process name:



$clientName = $client.Name

$processName = $client.ProcessName

* + These two lines retrieve the Name and ProcessName properties from the current $client hashtable and store them in the $clientName and $processName variables, respectively.
* Check if the client is installed on the machine:



$clientInstalled = $services | Where-Object { $\_.Name -eq $processName }

* + This line filters the $services array by comparing the Name property of each service object to the $processName of the current client.
    - If a match is found, the client is considered installed on the machine, and the matching service object is stored in the $clientInstalled variable.
* Set the client:



$clientStatuses[$clientName] = if ($clientInstalled) { "Installed" } else { "Not Installed" }

* + This line sets the status of the current client in the $clientStatuses hashtable.
    - If the $clientInstalled variable contains a matching service object, the status is set to "Installed".
      * Otherwise, the status is set to "Not Installed".
* Add the result for the machine:

Text

Description automatically generated

$results += [PSCustomObject]@{

ComputerName = $computerName

Status = "Online"

TaniumClient = $clientStatuses["Tanium Client"]

TenableClient = $clientStatuses["Tenable Client"]

CybereasonClient = $clientStatuses["Cybereason Client"]

}

* + This part of the script creates a new custom PowerShell object with properties for the computer name, online status, and the status of each client.
    - The object is then added to the $results array.
* Handle unreachable machines:

A picture containing text

Description automatically generated

catch {

...

}

* The catch block is used to handle errors that occur during the processing of a machine, such as when the machine is not reachable.

**Inside the catch block:**

* Add the result for an unreachable machine:

Text

Description automatically generated

$results += [PSCustomObject]@{

ComputerName = $computerName

Status = "Offline"

TaniumClient = "N/A"

TenableClient = "N/A"

CybereasonClient = "N/A"

}

* + This part of the script creates a new custom PowerShell object with properties for the computer name and offline status, and sets the status of each client to "N/A".
    - The object is then added to the $results array.
* Export the results to a CSV file:

Text

Description automatically generated

$results | Export-Csv -Path "ClientStatusReport.csv" -NoTypeInformation

* + This line pipes the $results array to the Export-Csv cmdlet, which generates a CSV file named "ClientStatusReport.csv" containing the results for each machine.
* Display a message on the console:

Text

Description automatically generated

Write-Host "Client status report generated: ClientStatusReport.csv"

* This line uses the Write-Host cmdlet to display a message on the console indicating that the client status report has been generated and saved as "ClientStatusReport.csv".

That's the complete, in-depth explanation of the script. This script queries the online machines from the NinjaRMM API, checks for the presence of the specified clients (Tanium, Tenable, and Cybereason) on each machine, and generates a CSV report with the results.