Ivanti Patch for Windows® Servers

API Quick Start Guide

Copyright and Trademarks

Document Information and Print History

Document number: N/A

Date	Version	Description
April 2017	Ivanti Patch for Windows® Servers 9.3	Initial release of this document.

HOW TO USE THE API FEATURE

Welcome

This document describes how to use the API feature in Ivanti Patch for Windows® Servers. The API feature exposes the Ivanti Patch for Windows® Servers API stack. You can execute API-level calls from the command-line or from a PowerShell console.

Overview

Tip: To view a video tutorial on this topic, click the video icon.

The API feature is meant for advanced users who have a working knowledge of PowerShell and who want to perform tasks beyond those available through the Ivanti Patch for Windows® Servers user interface. You can use the API feature to:

- Interact with different systems in your environment.

  You are now able to integrate your patching and power state processes with items such as vulnerability scanners, password management tools, SQL Server clusters and orchestrators such as Chef or Puppet.

- Script a sequence of complex events that contain dependencies.

  Using PowerShell, you can script out interesting and complicated workflows. You can include checks within the script to make sure that everything goes according to plan. For example, you might patch one machine in a cluster and make sure that everything goes according to plan before proceeding with the other machines in the cluster.

  You might also script out actions such as suspending nodes, starting and stopping services at certain points, restarting machines in a specific order, etc.

- Perform bulk operations or process list inputs from other systems.

  For example, you might dynamically perform a scan operation by targeting systems listed in a file.

- Programmatically stage patch deployments or initiate patch downloads

Requirements

The following are required in order to use the API feature:

- Ivanti Patch for Windows® Servers 9.3 or later

  The API and Windows PowerShell components and modules are automatically installed with Ivanti Patch for Windows® Servers 9.3 or later. No configuration is required.

- Must be run from the Ivanti Patch for Windows® Servers console.

  The API commands will run in a user session on the console. The user’s context is handed off to the Protect service so that the lifetime is decoupled from the interactive session.

- Windows PowerShell v4.0 or later must be installed on the console.

- The Windows PowerShell console must be run with administrator rights.

- The STProtect API module must be imported before executing any PowerShell commands.

Areas That Can be Accessed by the API

API commandlets are available for use with the following functional areas in Ivanti Patch for Windows® Servers:

- Patch scanning
- Patch deployment
- Credentials
- Machine groups
- Patch groups
- Power state management

You can leverage existing configuration and workflow concepts in any of these areas within Ivanti Patch for Windows® Servers. This means you can reference existing credentials, machine groups, templates, logging, etc. within your API commands. You can also use the API to add new credentials, templates and groups.

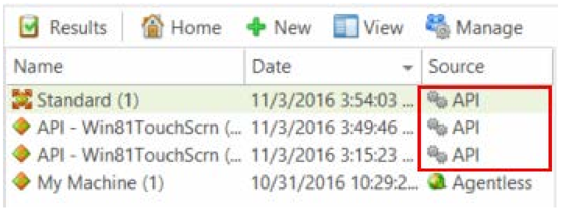
The commands that you issue from the API will interact the Ivanti Patch for Windows® Servers user interface. The results of actions that you drive from the API will be visible in Ivanti Patch for Windows® Servers. For example, if you initiate a patch deployment from the API, the deployment process can be monitored using Deployment Tracker.

Viewing and Tracking API-Driven Actions

Actions that you drive from the API will be reflected in the Ivanti Patch for Windows® Servers user interface. They are labeled to distinguish between API-initiated runs and console-initiated runs. Actions receive the same level of tracking regardless of whether they are initiated from the user interface or at the API level.

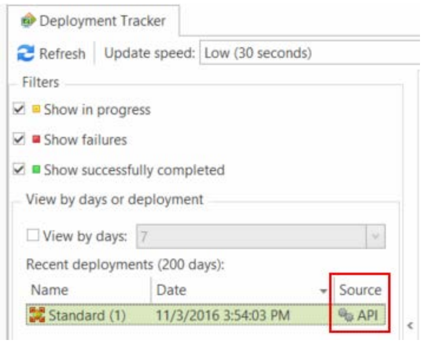
Examples

- The results of patch scans and deployments performed from the API are identified within the Results pane in the **Source** column.



Name	Date	Source
Standard (1)	11/3/2016 3:54:03 ...	API
API - Win81TouchScrn (...	11/3/2016 3:49:46 ...	API
API - Win81TouchScrn (...	11/3/2016 3:15:23 ...	API
My Machine (1)	10/31/2016 10:29:2...	Agentless

- Patch deployments performed from the API can be tracked using Deployment Tracker.



Update speed: Low (30 seconds)

☑ Show in progress
☑ Show failures
☑ Show successfully completed

☐ View by days: 7

Recent deployments (200 days):

Name	Date	Source
Standard (1)	11/3/2016 3:54:03 PM	API

How to Get Started

Here are the basic steps for getting started with API calls that are executed from a PowerShell console.

1. From the Ivanti Patch for Windows® Servers console, launch a Windows PowerShell console.

   Be sure to specify *Run as Administrator* when launching the PowerShell console.

2. Load the STProtect API module by executing the following PowerShell command:

   ```
   PS C:\Import-Module STProtect –PassThru
   ```

   The STProtect API module contains API calls that allow you to interact with many of the functional areas within Ivanti Patch for Windows® Servers. The optional `–PassThru` option provides diagnostic feedback about how the module is loaded.

   You should always load the module first whenever you write a script. One way to ensure that the module is always loaded is to add the module import command to your PowerShell user profile.

3. Load any other required Windows or third-party modules.

   You may need commandlets from other modules in order to interact with other devices in your environment. For example:

   - Servermanager: Required in order to interact with the SQL cluster.
   - Failoverclusters: Loads Windows feature RSAT-Clustering.

4. Execute individual commands or scripted commands.

   Here are a few popular examples:

   Perform a patch scan

   ```
   Start-PatchScan –MachineGroups "My Machine"
   ```

   Start and view results of a patch scan

   ```
   Start-PatchScan –MachineGroups "My Machine" | Watch-PatchScan
   ```

   Deploy missing patches

   ```
   Start-PatchDeploy –Uid 12345678-3BD2-A0ED-FFCC-9876DE8EBBAA
   ```

5. When finished, close the Windows PowerShell console.

   All modules that you imported or items that you stored in memory will be erased when the PowerShell console is closed. Scan or deployment operations that you initiated will not stop when PowerShell closes.

How to View Detailed Help Information About the PowerShell Commands

There are a number of ways you can find help information for the available PowerShell commands. Simply launch a PowerShell console, import the STProtect module and then type any of the following commands. Full localized help documentation is included as part of the Ivanti Patch for Windows® Servers installation.

View the complete list of available commands

```
PS C:\> (Get-Module –Name STProtect).ExportedCommands
```

View detailed help for an individual command

```
PS C:\> Get-Help Add-STCredential -detailed
```

View complete help for all commands

```
PS C:\> (Get-Module STProtect).ExportedCommands.Values |
ForEach-Object { $_ | Get-Help –Full}
```

To pipe the output to a text file:

```
PS C:\> (Get-Module STProtect).ExportedCommands.Values |
ForEach-Object { $_ | Get-Help –Full} | Out-File
“C:\SampleDirectory\HelpOutputFull.txt”
```

Tip and Tricks

Here are a few tips and tricks that will improve your experience with the API.

- When typing PowerShell commands, use the Tab button to auto-complete the command.
- Use pipelining within your commands to string together a series of actions.
- If you want to view the output when performing a patch scan, be sure to use the **Watch-PatchScan** parameter.

  Example:

  ```
  Start-PatchScan –MachineGroups “Sample Group” | Watch-
  PatchScan
  ```

- You can assign the result of any command to a variable and interact with that variable later in the same PowerShell session.

  Examples:

  ```
  $credReference = Get-STCredential

  $credReference | Where-Object
                   { $_.UserName.Contains(“foo”) }

  $myScan = Start-PatchScan –MachineGroups “My Machine”

  Wait-PatchScan –Uid ($myScan.Uid)
  ```

- If you want to perform a patch deployment against the results of a particular patch scan, store the patch scan in a variable and pipe it to the deployment operation.

  Example:

  ```
  $myScan = Start-PatchScan –MachineGroups “Sample Group”;
  $myScan | Watch-PatchScan
  ```

  ```
  $myScan | Start-PatchDeploy –ScanUid ($MyScan.Uid) –
  TemplateName “Sample Deploy Template”
  ```

- If you are scripting a patch scan followed by a patch deployment, be sure to use the **Wait-PatchScan** parameter to allow time for the scan to complete before the deployment is initiated.

  Example:

  ```
  $MyScan = Start-PatchScan –MachineGroups “Sample Group” |
  Wait-PatchScan
  ```

  ```
  Start-PatchDeploy –ScanUid ($MyScan.Uid) –TemplateName
  “Sample Deploy Template”
  ```

- Be sure to load any additional modules that might be needed.

  Example: When interacting with a SQL cluster you might load the following modules:

  - Import-Module ServerManager
  - Add-WindowsFeature RSAT-Clustering
  - Import-Module FailoverClusters

- To minimize downtime, you can use the `Invoke-DownloadMissingPatches` command before deploying.

  This would enable you to perform a scan one day and then use the `Get-PatchScan` command to deploy from that scan on a different day.

Use Examples

There are many possible uses of the API feature. Here are two examples that many Ivanti Patch for Windows® Servers customers are likely to find useful.

Integrate Ivanti Patch for Windows® Servers with a Vulnerability Scanner

If you use a vulnerability scanner to identify weaknesses in your network, the scanner may detect hundreds or even thousands of issues on your machines. At first this might seem a bit overwhelming, but what's likely happening is that the vulnerability scanner is simply producing a lot of noise. The scanner will often report every single missing patch as a vulnerability, when in reality the machines are probably only missing a few key patches that supersede a large number of older patches. It often takes very few patches to resolve many so-called vulnerabilities.

To address this, you can use the API to:

- Make calls to the vulnerability scanner
- Extract the vulnerability list (consisting of CVEs)
- Import those CVEs into a Ivanti Patch for Windows® Servers patch group
- Perform patch scans and deployments using that patch group

The patch engine will eliminate all of the superseded patches and will identify the handful of patches that are actually missing. If you rerun the vulnerability scanner after deploying the patches, most of the vulnerabilities will likely go away.

Use a PowerShell Script to Patch a Clustered SQL Server

This example shows how to use a PowerShell script to control the patch scanning and patch deployment processes on machines in an SQL cluster. The goal in this example is to coordinate events and provide zero downtime while patching.

```
Import-Module STProtect –PassThru
Import-Module ServerManager
Add-WindowsFeature RSAT-Clustering
Import-Module FailoverClusters

Write-Host “Scanning SampleSQL1”
$scan = Start-PatchScan –EndpointName “SampleSQL1”
–CredentialFriendlyName “Sample Domain Cred” –TemplateName
“Security Patch Scan”

$scan | Watch-PatchScan
Write-Host “Scanning SampleSQL1 Complete”

$scanDetail = $scan | Get-PatchScan
Write-Host “SampleSQL1 Final Details”
$scanDetail | Format-PatchScanTable

#Identify if there are missing patches on the target node. If
yes, stop the node in the cluster and patch the system.
If ($scanDetail.MachineStates[0].MissingPatches –ne 0)
{
   $totalMissingPatches =
scanDetail.MachineStates[0].MissingPatches
```

```
    Write-Host “$totalMissingPatches total missing patches will
       be deployed. Starting deploy . . .”
    Write-Host “Stopping cluster node on SampleSQL1”
    Suspend-ClusterNode –Name SampleSQL1 –Cluster
       SampleSQLCluster –Drain

    $deploy = $scan | Start-PatchDeploy –TemplateName “Standard”
    $deploy | Watch-PatchDeploy | Format-Table

    Write-Host “Starting cluster node on SampleSQL1”
    Resume-ClusterNode –Name SampleSQL1 –Cluster SampleSQLCluster
       –Failback NoFailback
}
```

In particular, this example script will:

1. Import the required modules.
2. Scan a single node in a SQL cluster.
3. Watch the patch scan, blocking further action until it completes.
4. Get the results of the scan.

   The output will include details such as information about the targeted machine, the number of missing and installed patches, a message indicating that the scanning process is complete, etc.
5. If there are missing patches, suspend the node in the SQL cluster and prepare to patch the system.

   Do this by suspending the node in the cluster and draining all current sessions from this node to another node.
6. Start the patch deployment using the Standard template.
7. Watch the progress of the patch deployment, blocking other actions until it completes.

   This will display the current status of the deployment including scheduled, complete and reboot, pending rescan, and finished.
8. Add the node back to the SQL cluster and resume operation.

Repeat the process as needed for additional SQL Server instances or nodes in the cluster.

Frequently Asked Questions

Here are some questions that are frequently asked by new Ivanti Patch for Windows® Servers API users.

Can I use the API to schedule scans and deploys?
Yes, you can schedule a script to run at a specific time. This is independent of the Ivanti schedulers. You can utilize the existing scheduling capabilities of an orchestrator or of Windows Task Scheduler, or you can do a scheduled scan in Ivanti Patch for Windows® Servers.

Can I use the API to create, configure and delete credential, templates and groups?
Yes. You can reference existing credentials, machine groups, templates, etc. within your API commands. You can also use the API to add new credentials, templates, machine groups and patch groups.

Can I still get reports?
Yes, all Ivanti Patch for Windows® Servers reports will continue to work and will include operations initiated by the API.

What about logging?
The same logging levels are available at both the user interface level and the API level.

Can we show members of an object coming out of a scan commandlet?
Yes. Here's an example of how to show the 10 most recent scans:

```
> $scans = Get-PatchScan –count 10
> $mostRecentScan = $scans[0]
> $mostRecentScan | Get-Member
> $mostRecentScan | Format-PatchScanTable
> $mostRecentScan.MachineStates[0] | Get-Member
>
```

Can I exclude certain machines before performing a deployment?
Yes, you will do this by controlling the scan input. You will specify which machines should be scanned, pare that down using attributes and then feed the pared down scan into the deployment process.

Can I check to see if a patch install fails (post-deploy)?
Yes. See the error code. You can view this information using Deployment Tracker or by using the Get-PatchDeploy command.

Can I retrieve patch scan deployment results (even UI-initiated scans) for reporting purposes?
Yes.

CONTACT SUPPORT

For technical assistance with the API feature, please refer to one of the following support options:

- Browse the community site at: http://community.shavlik.com
- Open a support request at: http://support.shavlik.com/CaseLogging.aspx
- Phone Technical Support at: 866-407-5279 or +1-651-407-5279