Configure WinRM Security

Create an encrypted communication channel for WinRM by using Puppet tasks to configure WinRM with SSL. This lightweight approach works in the cloud, on-prem, and in testing and production environments.

★ Prerequisites

Before you begin, you need:

- Puppet Enterprise (PE) 2018 or later with Code Manager configured.
- Permission to run tasks.
- Basic understanding of adding modules to PE by using Code Manager.
- Basic understanding of using git and version control.
- Windows machine running Windows Server 2016 or later.
- Puppet Development Kit.
- Puppet Bolt.
- PE master that can communicate with the Windows machine by using WinRM on ports 5985 and 5986.

I. Generate an SSL certificate

Generate a self-signed SSL certificate by using a built-in PowerShell cmdlet in Windows Server 2016 or later. This command generates a self-signed certificate with the DNS name that you provide and adds the certificate to the machine's certificate store. Be sure to use the fully qualified domain name.

Run the following command (you can copy the code from GitHub):

```
$certname = [System.Net.Dns]::GetHostByName($env:computerName).Hostname
$cert = New-SelfSignedCertificate -DnsName $certname -CertStoreLocation Cert:\
LocalMachine\My
```

Result: PE stores the command output as a variable, which you'll use later.

II. Configure an HTTPS WinRM listener

Now that the certificate is in the local machine's certificate store, create an HTTPS WinRM listener.
Open a text editor and paste the following code into your PowerShell script (copy the code from GitHub):

```
$valueset = @{}
$valueset["Hostname"] = $certname
$valueset["CertificateThumbprint"] = $cert.Thumbprint
$selectorset = @{}
$selectorset["Transport"] = "HTTPS"
$selectorset["Address"] = "*"
New-WSManInstance -ResourceURI 'winrm/config/Listener' -SelectorSet $selectorset
-ValueSet $valueset
```

Result: You now have a PowerShell script to configure a WinRM HTTPS listener on your Windows machines.

III. Create a module by using PDK

Create the module. Later, you'll create a task and update it with your PowerShell script.

1. On the command line, run `pdk new module`.
2. For each question, enter the information for your system or accept the default answer. Modify your answers at any time by manually updating the metadata.json file in the module.
   - Enter a module name that's relevant to the module's content. For this example, enter `winrm_security`.
   - Enter your Puppet Forge username, if you have one. Press Enter if you don't.
   - Name the author of this module..
   - Specify which license the module code falls under. See https://spdx.org/licenses/ for identifiers. Common values are **Apache-2.0**, **MIT**, or **proprietary**. For this example, enter `Apache-2.0`.
   - Select the supported operating systems. For this example, select `Windows`.
   - To generate metadata based on this information, enter `Y`.

IV. Create a task in the module

Modules contain tasks that are typically related and accomplish a single outcome. You can download a module that already contains tasks or you can create the tasks within the module from scratch.

1. Go to the *winrm_security* directory created by the `pdk new module` command.
2. Run the following command: `pdk new task configure_https_listener`

   Result: By default, this command creates two files in the tasks directory: a metadata JSON file for the task and a shell script template.
3. Delete the default shell script, which doesn't work on Windows. Later, you'll replace it with a PowerShell script.

V. Update the task metadata

Update the task metadata file with a description and add a *certificate_validity_days* parameter to specify the number of days that the SSL certificate is valid.

1. Navigate to the *tasks* directory of the module.
2. Open the *configure_https_listener.json* metadata file with a code editor.
3. Replace the contents with the following code (copy the code from GitHub) and save your changes.

```
{
  “puppet_task_version”: 1,
  “supports_noop”: false,
  “description”: “Configure a WinRM HTTPS listener with a self-signed SSL certificate”,
  “parameters”: {
    “certificate_validity_days”: {
      “description”: “The number of days the SSL certificate is valid.”,
      “type”: “Optional[String]”
    }
  }
}
```

VI. Convert the script to a task

Create a .ps1 file for the contents of the task and update it with the WinRM SSL PowerShell script content.

1. Go to the *tasks* directory.
2. Use a text editor to create a file named *configure_https_listener.ps1*.
3. Copy the code below from GitHub and add it to the .ps1 file. Then, save your changes.
   - The *certificate_validity_days* parameter specifies the number of days that the SSL certificate is valid.
   - PowerShell Write-Output statements generate formatted output.
   - Conditional logic evaluates whether a WinRM HTTPS listener exists.

```
[CmdletBinding()]
param (
  # The number of days the SSL certificate is valid
  [Parameter(Mandatory=$false)]
  [int]
  $certificate_validity_days
)

$listeners = Get-ChildItem WSMan:\localhost\Listener

# Evaluate if an existing HTTPS listener exists
If (!($listeners | Where-Object {$_.Keys -like "TRANSPORT=HTTPS"})) {

    # Grab the system FQDN for the SSL certificate
    $certname = [System.Net.Dns]::GetHostByName($env:computerName).Hostname

    # Generate a self-signed SSL certificate and add it to the local machine
certificate store
    $cert = New-SelfSignedCertificate -DnsName $certname -CertStoreLocation Cert:\
LocalMachine\My -NotAfter (Get-Date).AddDays($certificate_validity_days)

    # Create the hashtables of settings to be used.
    $valueset = @{
        Hostname = $certname
        CertificateThumbprint = $cert.Thumbprint
    }

    $selectorset = @{
        Transport = "HTTPS"
        Address = "*"
    }

    # Create a HTTPS listener
    New-WSManInstance -ResourceURI 'winrm/config/Listener' -SelectorSet $selectorset
-ValueSet $valueset
    Write-Output "{""status"":""Configured HTTPS listener""}"
} Else {
    Write-Output "{""status"":""HTTPS listener already configured""}"
}
```

Result: You now have a module with a task for configuring a WinRM HTTPS listener.

VII. Validate the task

Now that you converted the script into a task, test it by running it locally using Bolt before adding it to the Puppet master. Use Bolt to test tasks locally: It's faster than testing in PE.

1. From the directory where you ran the pdk new module command, run the following command. Remember to include the relative path to the winrm_security directory.

```
bolt task run -m . winrm_security::configure_https_listener --targets=localhost
```

Result: You now have a task that configures a WinRM HTTPS listener, and the task is part of the module that you created with PDK.

2. Run the following command to confirm that the HTTPS listener exists:

```
Get-ChildItem WSMan:\localhost\Listener
```

The output contains a listener entry with Transport=HTTPS.

```
Type            Keys                                Name
----            ----                                ----
Container       {Transport=HTTPS, Address=*}        Listener_1305953032
Container       {Transport=HTTP, Address=*}         Listener_1084132640
```

Result: You can now install the module on the Puppet master.

VIII. Install the module on the Puppet master

Use Code Manager to install the module on the Puppet master so that you can run the task from the PE console.

- If webhooks **are configured** to automatically deploy the module, the module is installed on the master moments after you commit and push the code changes to source control.
- If webhooks **are not configured**, you must manually deploy the code by using the puppet code deploy command.

The following is an example entry in the Puppetfile for installing the module. This example assumes that the module is added to the production branch of the codebase.

```
mod 'winrm_security',
    git: 'https://github.com/puppetlabs/winrm_security.git',
    branch: 'master'
```

Result: The winrm_security module is installed on the Puppet master. You can now configure the WinRM HTTPS listener on a Windows node by running your task from the PE console.

IX. Add the Windows node to the PE inventory

The PE inventory is the list of nodes that Puppet manages. Add the node to the inventory so that you can run your task against the node.

1. In the PE console, click **Inventory**, and then click **Connect over SSH or WinRM** to create an agentless node connection.
2. Select *WinRM* from the **Transport method** drop-down list.
3. In the **WinRM hosts** field, enter the node name. To specify multiple nodes, use a comma-separated list.

4. Enter your credentials in the **User** and **Password** fields.
5. To test the connection to the node, leave the **Test Connection** checkbox selected.
6. Click **Add nodes**.
7. To view details about the node, click **View nodes**. To view details for multiple nodes, click **Nodes** in the sidebar, and then click the node name **> Connections**.

X. Run the task from the PE console

Now that the nodes are in the inventory, run the task against other Windows nodes from the PE console. After the task runs against a node, the communication channel is encrypted.

1. In the PE console, click **Task** in the sidebar.
1. Click **Run a Task**.
2. Ensure the **Code environment** is set to *production*.

   Note: The example assumes you're using a production environment. If you're using a different environment, enter its name in the Code Environment field.
3. In the **Task** drop-down list, select *winrm_security::configure_https_listener*. To view the task parameters, click **view task metadata**.
4. In the **Task parameters** section, in the **Parameter** field, enter *certificate_validity_days* and in the **Value** field, enter *365*. Then, click **Add parameter**.
5. Select the target node.
   - In the **Select targets** drop-down menu, select *Node list*.
   - In the Add nodes field, enter the node name, click **Search**, and select the checkbox beside the node name. To add all the nodes, scroll and click **Add all**.
6. In the **Node** section, click the node checkbox and click **Run job**.

   Result: After you run the task against one Windows node, check the PE logs to confirm the task ran successfully. If so, WinRM traffic is now encrypted on that node.

   From the console, you can then run the task against the other Windows nodes in your environment to encrypt WinRM communications on those nodes.

   By running the task in the console instead of by using Bolt, logs are generated, which is important for audits And even though you might run this task only once, remember that by using PE to run tasks, you can use RBAC to give specific people permission to run the task.