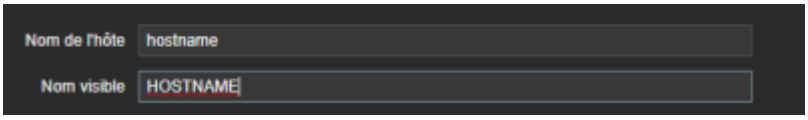
Monitoring des mises à jour Windows

Configuration de l'hôte Zabbix

Vérification du nom d'hôte

Sur la configuration de l'hôte Zabbix vérifiez le nom d'hôte :

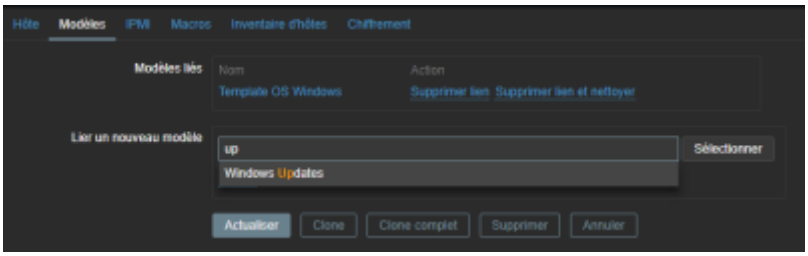


Ce dernier doit correspondre soit au nom d'hôte de la machine, soit à ce dernier en minuscule.

Pour vérifier le nom d'hôte d'une machine en powershell : **$env:computername**

Ajout du modèle à l'hôte

Ajoutez le modèle Windows Update à l'hôte. Si il s'agit de votre première configuration et que le modèle n'est pas existant, installez ce modèle.



Configuration de la machine

Ajout du script sur la machine

Dans un premier temps vous allez devoir créer un fichier .ps1 (par exemple updates.ps1) sur la machine cible, puis copier le contenu suivant :

```
param(
    [Parameter(Mandatory=$true)][string]$proxyAddr,
    [switch]$noLowerCase
)

$source = @"
public class ZabbixData
{
```

```
    public System.String host;
    public System.String key;
    public System.String value;
}

public class ZabbixSender
{
    public System.String request = "sender data";
    public System.Collections.ArrayList data = new
System.Collections.ArrayList();
}
"@

if (-not ([System.Management.Automation.PSTypeName]'ZabbixData').Type)
{
    Add-Type -TypeDefinition $source -Language CSharp
}

function _Receive ([System.Net.Sockets.Socket] $socket, [Byte[]] $buffer,
[int] $offset, [int] $size, [int] $timeout)
{
    $startTickCount = [System.Environment]::TickCount
    $received = 0

    $size
    do
    {
        if ([System.Environment]::TickCount -gt ($startTickCount +
$timeout))
        {
            throw "Timeout"
        }

        $received += $socket.Receive($buffer, $offset + $received, $size -
$received, [System.Net.Sockets.SocketFlags]::None);

    } while ($received -lt $size)
}

function ConvertTo-Json20([object] $item){
    Try {
        add-type -assembly system.web.extensions
    }
    Catch [System.IO.FileNotFoundException]
    {
        Throw "Can't add type, missing System.Web.Extensions."
    }
    $ps_js=new-object system.web.script.serialization.javascriptSerializer
    return $ps_js.Serialize($item)
}
```

```
$port = 10051

$session = New-Object -ComObject Microsoft.Update.Session
$searcher = $session.CreateUpdateSearcher()
$result = $searcher.Search("IsInstalled=0 and Type='Software'")
$nbUpdate = $result.Updates.Count
$nbSecurityUpdate = 0
$nbCriticalUpdate = 0

$result.Updates | ForEach {
    $strUpdateCat = $_.Categories.Item(0).Name
    if ($strUpdateCat.ToLower().contains("security")) {$nbSecurityUpdate +=
1}
    elseif ($strUpdateCat.ToLower().contains("critical")) {$nbCriticalUpdate
+= 1}
}

$nbOtherUpdate = $nbUpdate - $nbCriticalUpdate - $nbSecurityUpdate

$z = New-Object ZabbixSender

$hostname = $env:computername
if ( -Not $noLowerCase )
{
    $hostname = $hostname.toLower()
}

$s = New-Object ZabbixData
$s.host = $hostname
$s.key = "updates.security"
$s.value = $nbSecurityUpdate

$c = New-Object ZabbixData
$c.host = $hostname
$c.key = "updates.critical"
$c.value = $nbCriticalUpdate

$n = New-Object ZabbixData
$n.host = $hostname
$n.key = "updates.nonsecurity"
$n.value = $nbOtherUpdate

$z.data.Add($s)
$z.data.Add($c)
$z.data.Add($n)

$json = ConvertTo-Json20($z)

$header = [System.Text.Encoding]::ASCII.GetBytes("ZBXD" + [char]1)
$length = [System.BitConverter]::GetBytes([long]$json.length)
$data = [System.Text.Encoding]::ASCII.GetBytes($json)
```

```
$all = $header + $length + $data

$sock = New-Object
System.Net.Sockets.Socket([System.Net.Sockets.AddressFamily]::InterNetwork,
[System.Net.Sockets.SocketType]::Stream,
[System.Net.Sockets.ProtocolType]::TCP)
Try
{
    $sock.Connect($proxyAddr, $port)
}
Catch
{
    "Socket not connected, check that your port is open and listening"
    exit
}


$sock.Send($all)

$buffer = New-Object Byte[] 5

(_Receive $sock $buffer 0 5 10000)
if (("ZBXD" + [char]1) -ne [System.Text.Encoding]::ASCII.GetString($buffer,
0, $buffer.Length))
{
    Write-Error "Invalid Response"
}

$buffer = New-Object Byte[] 8
(_Receive $sock  $buffer 0 8  10000)

$dataLength = [System.BitConverter]::ToInt32($buffer, 0)

if ($dataLength -eq 0)
{
    Write-Error "Invalid Response (size)"
}

$buffer = New-Object Byte[] $dataLength
(_Receive $sock $buffer 0 $buffer.Length 10000)

[System.Text.Encoding]::ASCII.GetString($buffer, 0, $buffer.length)



$sock.Close()
```

Il est nécessaire de copier le script plutôt que de le télécharger depuis un serveur

distant si vous ne souhaitez pas utiliser la politique d'exécution la plus permissive.

Si vous avez déjà un agent Zabbix sur la machine, il peut être intéressant de placer le script dans un dossier Script dans la même arborescence.

Maintenant essayez d'exécuter le script :

```
<PATH_TO_SCRIPT_PS1> IP_du_proxy_Zabbix [-noLowerCase]
```

L'option facultative **-noLowerCase** est à utiliser lorsque le nom d'hôte sur le Zabbix correspond à **$env:computername**, si elle n'est pas spécifié alors l'hôte devra correspondre à **$env:computername.toLower()**

Par exemple :

```
C:\zabbix\Scripts\updates.ps1 192.168.0.240 -noLowerCase
```

Si cela fonctionne vous devriez obtenir quelque chose comme

```
{"response":"success","info":"processed: 3; failed: 0; total: 3; seconds
spent: 0.000200"}
```

Si vous ne pouvez pas exécuter le script à cause de la politique d'exécution vous pouvez changer cette dernière.

Par exemple :

```
Set-ExecutionPolicy RemoteSigned
```

Si vous obtenez une erreur ou que le json de retour contient **failed: 3** référez-vous à la partie Troubleshooting

Définition de la tâche

Afin de récupérer les informations sur la machine cible à un intervalle régulier il est possible de configurer une tâche, voici un exemple de configuration pour une exécution une fois par heure (remplacez <cmd> par la commande avec lequel vous appelez le script en utilisant le chemin complet vers le .ps1) :

```
schtasks /create /TN ZabbixWinUpdates /SC HOURLY /RU SYSTEM /TR
```

```
"powershell.exe -file <cmd>"
```

Ce qui donne par exemple :

```
schtasks /create /TN ZabbixWinUpdates /SC HOURLY /RU SYSTEM /TR
"powershell.exe -file C:\zabbix\Scripts\updates.ps1 192.168.0.240 -
noLowerCase"
```

Troubleshooting

failed: 3

Si vous obtenez un json comportant failed: 3 il peut s'agir (ou non) d'une erreur de configuration. Pour s'en assurer :

- Vérifiez que le nom d'hôte mis en minuscule est le même que sur la configuration Zabbix (ou que le nom d'hôte correspond complètement si vous utilisez -noLowerCase)
- Vérifiez que l'hôte a bien été configuré (qu'il utilise bien le modèle **Windows Updates**)

Dans le cas où vous êtes certain que ces éléments sont correct réessayez ultérieurement, il arrive qu'il soit nécessaire d'attendre un certain temps (dont je ne connais pas la durée exact) pour que le trapper commence à fonctionner.

Socket not connected

Si vous obtenez cette erreur vérifiez tout d'abord que l'IP de votre Zabbix/Proxy Zabbix est la bonne. Si c'est le cas, vérifiez que le port 10051 est en écoute par le serveur, pour cela vous pouvez utiliser telnet.

```
telnet [IP] 10051
```

Si la machine que vous utilisez ne dispose pas d'un client telnet, utilisez une machine sur laquelle vous pouvez vous permettre des modifications (ou une machine avec un client telnet), l'installation du client telnet peut être effectué de la manière suivante :

```
dism /online /Enable-Feature /FeatureName:TelnetClient
```

Si telnet retourne une erreur d'échec de connexion, alors le port n'est pas ouvert ou pas à l'écoute, rendez-vous sur le serveur :

- Vérifiez que le proxy Zabbix est opérationnel et à l'écoute sur le port 10051
- Vérifiez que le pare-feu ne bloque pas la connexion au port

Si vous utilisez FirewallD et que vous avez besoin d'autoriser la connexion au port 10051 voici une liste de commande permettant de réaliser l'opération :

```
firewall-cmd --new-service=zabbix --permanent
firewall-cmd --service=zabbix --add-port=10051/tcp --permanent --set-description="zabbix trapper" --permanent
firewall-cmd --zone=public --add-service=zabbix --permanent
firewall-cmd --reload
```

Can't add type, missing System.Web.Extensions.

La dll permettant la sérialisation en json est manquante. Le problème étant assez rare il est probable qu'il soit question de fonctionnalités de Windows disponibles dans une mise à jour non installé sur certaines machines. Axes de résolutions sans mises à jour :

- Recoder la sérialisation en json des objets (voir cette implémentation)
- Se passer de la sérialisation et faire un json à trou (moins propre mais plus rapide), voir le code ci-dessous à placer dans le fichier à la place de la fonction existante.

```
function ConvertTo-Json20([object] $item){
    $ret = '{"request":"sender data", "data":['
    $item.data | ForEach {
        $ret += '{"host":"' + $_.host + '","key":"' + $_.key + '","value":"' + $_.value + '"},'
    }
    $ret = $ret.Substring(0,$ret.Length -1)
    $ret += ']}'
    return $ret
}
```

Un tel code fonctionnerait partout mais ajoute un plus grand nombre de modifications nécessaires en cas d'adaptation du script après une mise à jour ou pour une autre utilisation, c'est pour cela que ce snippet n'est pas intégré.