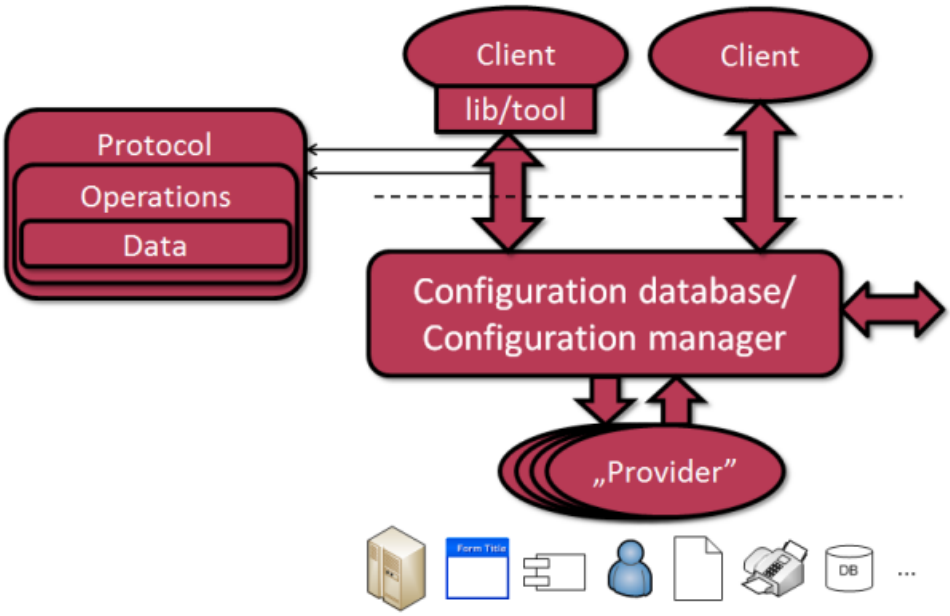
Konfigurációkezelési technológiák

Gyakorlati útmutató
Készítette: Micskei Zoltán
Utolsó módosítás: 2011.04.22.

A segédlet célja, hogy bemutassa a konfigurációkezelési technológiákhoz tartozó alap eszközöket, és hogy megismerkedjünk a *Common Information Model*lel (CIM), a kapcsolódó szabványokkal (pl. WS-Management, CIM-XML) és ezek implementációival.

Az általános konfigurációkezelési ábránk a következő:



1. ábra: Konfigurációkezelés általános architektúrája

Figyelem:

- Az utasításokat ne másoljuk, hanem tényleg gépeljük is be. Különben nem sok mindent tanulunk belőle, nem rögzül a szintaktika.
- A gyakorlati feladatsor végigcsinálása előtt nézzük át a kapcsolódó két előadást!

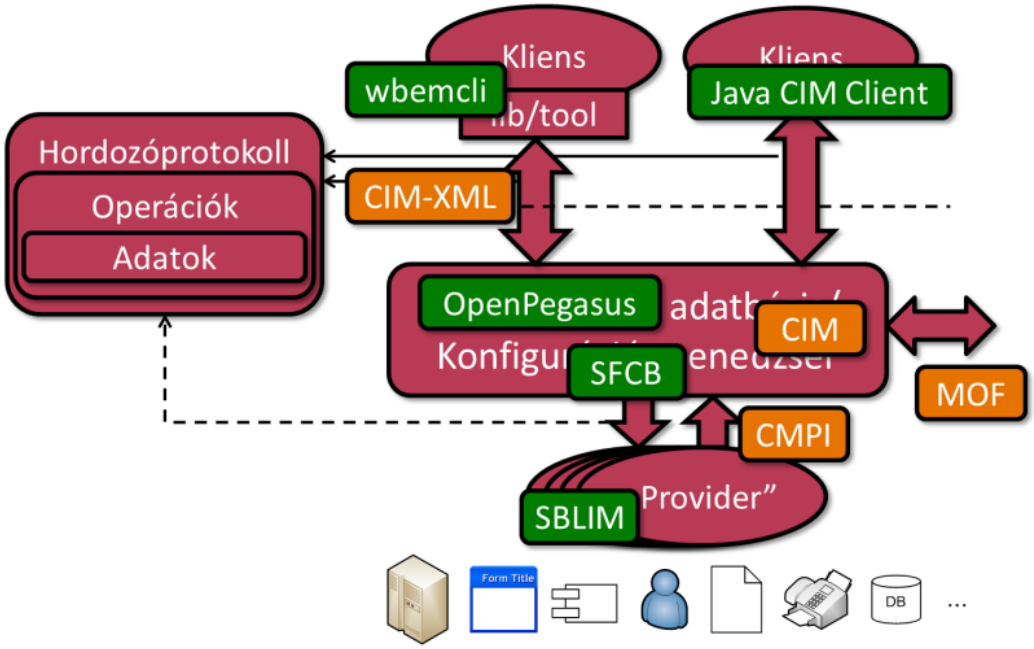
Tartalom

1 Linux: OpenPegasus, wbemcli és openwsman

A feladatok megoldásához például a kiadott VMware virtuális gépbe telepített CentOS rendszert lehet használni. Ez a virtuális gép előre telepítve tartalmazza a következőket:

- CIM Object Manager: OpenPegasus[1] 2.9.1
- CIM-XML kliens: wbemcli[2] 1.6.1
- WS-Management szerver: openwsman[3] 2.2.6
- WS-Management kliens: wsmancli 2.2.6

Az általános konfigurációkezelési ábránkat tehát a következő módon fedik le az eszközök.



2. ábra: Konfigurációkezelő technológiák Linuxon

1.1 OpenPegasus

Az első feladatban áttekintjük az *OpenPegasus*, a *CIM Object Manager* beállításait és pár egyszerű lekérdezést hajtunk végre.

1. Indítsuk el a virtuális gépet!

   A gépre SSH segítségével lépjünk be (így ki lehet pl. a kimenetről másolni adatokat).

   A gépbe történő bejelentkezéshez szükséges adatok:

[1] http://www.openpegasus.org/
[2] http://sourceforge.net/apps/mediawiki/sblim/index.php?title=Wbemcli
[3] Forrás: http://sourceforge.net/projects/openwsman,
RPM csomagok: https://build.opensuse.org/project/show?project=Openwsman

- Felhasználói név: **meres**
- Jelszó: **LaborImage**

2. Nézzük meg az OpenPegasus beállításait:

   A fájlok a `/etc/Pegasus` könyvtárban vannak. (A *pem* kiterjesztésű fájlok digitális tanúsítványok.)

   - Milyen állományokat találunk itt, ezek mit tárolnak?

   Nézzük meg, hogy kinek van joga hozzáférni a CIM szerverhez!

   ```
   nano /etc/Pegasus/access.conf
   ```

3. OpenPegasus elindítása

   Indítsuk el a CIM kiszolgálót:

   ```
   sudo /etc/init.d/tog-pegasus start
   ```

   Kérdezzük le, hogy tényleg elindult:

   ```
   sudo /etc/init.d/tog-pegasus status
   ```

   Ilyenkor visszaadja az elindított cimserver folyamat azonosítóját (PID).

4. OpenPegasus futási beállításai:

   Miután fut az OpenPegasus, megnézhetjük a futási idejű beállításait is:

   ```
   cimconfig -l -c
   ```

   - Engedélyezve van-e az SSL?

   Bizonyos paramétereket nem lehet futás közben módosítani, ilyenkor a módosításnál meg kell adni a –p kapcsolót, aminek a hatására ez csak a következő induláskor jut érvényre. Példa:

   ```
   sudo cimconfig -s enableHttpConnection=true -p
   ```

5. Providerek listázása:

   Nézzük meg, hogy milyen providerek vannak jelenleg telepítve:

   ```
   cimprovider -l -s
   ```

   - Keressük ki, hogy milyen IP-vel kapcsolatos providerek vannak (használjuk a grep parancsot).

6. OpenPegasus által használt port kiderítése:

   Nézzük meg, hogy milyen portokon figyel jelenleg a virtuális gép (a –t a TCP kapcsolatokat jeleníti meg, a –l a listening állapotban lévőket, a –p a hozzá tartozó folyamatot keresi ki):

```
sudo netstat -t -l -p
```

- Keressük ki a cimserver folyamathoz tartozó port számát (a --numeric hatására numerikus formában jeleníti meg az ismert portokat is, különben a `/etc/services` fájlban találjuk meg a megfeleltetést).

7. Alap osztály lekérdezése helyileg:

   Kérdezzünk le egy olyan osztályt, ami biztos létezik:

   ```
   wbemcli gc 'http://localhost:5988/root/cimv2:CIM_OperatingSystem'
   ```

   A `gc` a `GetClass` művelet rövidítése.

   Az előbb a *netstat* kimenetéből kiderült, hogy például az 5989-es porton (ez a wbem-https port) és az 5988-as porton is figyelünk (ez a wbem-http port). Amíg tesztelünk és hibát keresünk, érdemes a http portot használni, hogy például WireSharkban meg tudjuk nézni a forgalmat, azonban ha összeállt már a rendszer, érdemes kipróbálni https használatával is, éles rendszerben annak a használata a javasolt.

   Az objectPath-ban még kell adni a névteret (`root/cimv2`), majd egy kettőspont után az osztály nevét (`CIM_OperatingSystem`).

   A válasz erre az, hogy hitelesíteni kell magunkat, így adjunk meg felhasználónevet és jelszavat is:

   ```
   wbemcli gc 'http://pegasus:LaborImage@localhost:5988/root/cimv2:CIM_OperatingSystem'
   ```

   A pegasus felhasználó az OpenPegasus telepítésével létrejövő helyi felhasználó.

   Hogy a kimenet olvashatóbb legyen, adjuk meg az `-nl` paraméter (new line, új sorokat szúr be az egyes tulajdonságok után).

   ```
   wbemcli -nl gc 'http://pegasus:LaborImage@localhost:5988/root/cimv2:CIM_OperatingSystem'
   ```

   Még annyit nézzünk meg, hogy a háttérben milyen üzeneteket küld és kap a CIM-XML kliensünk. Erre a `-dx` kapcsoló való.

   - Keressük ki a kapott üzenetekből, hogy a TotalSwapSpaceSize tulajdonság milyen típusú!

   Kérdezzük le az osztály példányait is. Erre az `ei`, `EnumerateInstances` művelet használható.

   ```
   wbemcli -nl ei 'http://pegasus:LaborImage@localhost:5988/root/cimv2:CIM_OperatingSystem'
   ```

   A kimenetben keressük meg, hogy mikor bootolt fel az operációs rendszer (LastBootUpTime tulajdonság).

8. Lekérdezhető példányok listája:

   Nézzük meg, hogy a rendszer providerei milyen CIM osztályokat nyújtanak:

```
wbemcli -nl ei 'http://pegasus:LaborImage@localhost:5988/root/PG_InterOp:PG_ProviderCapabilities' |
grep ClassName
```

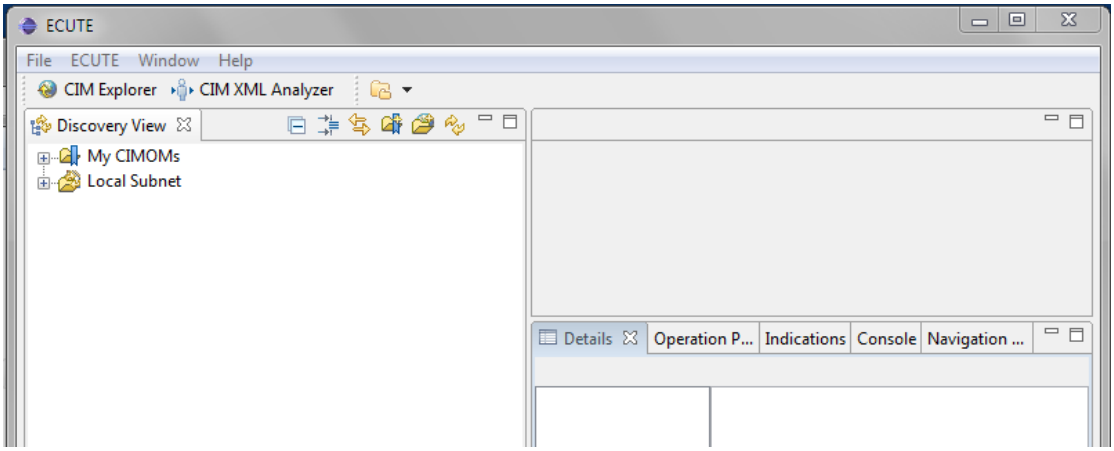
1.2 ECUTE

Az ECUTE (Extensible CIM UML Tooling Environment)[4] az SBLIM projekt része. Egy Eclipse alapú GUI-t biztosít CIMOM-ok megnézésére. Az adatokat CIM-XML segítségével kérdezi le. SLP (Service Location Protocol) segítségével fel is tudja deríteni, hogy milyen CIMOM-ok vannak a helyi hálózaton. A következő részekből áll:

- *Explorer*: CIM osztályok és példányok adatainak lekérdezése.
- *Analyzer*: kommunikáció megfigyelésére és az üzenetek tartalmának megjelenítésére szolgáló komponens.
- *(Modeler)*: UML modellekből képes CIM leírásokat készíteni, de ehhez kell az IBM Rational Software Architect program is.
- *ECUTE Rich Client Platform*: ha a fenti komponenseket nem egy meglévő Eclipse példányban, hanem önálló alkalmazásként akarjuk futtatni, akkor ehhez kell az ECUTE RCP.

A telepítéshez töltsük az ECUTE RCP-t, majd az Analyzer és Explorer tartalmát másoljuk be az RCP könyvtárába.

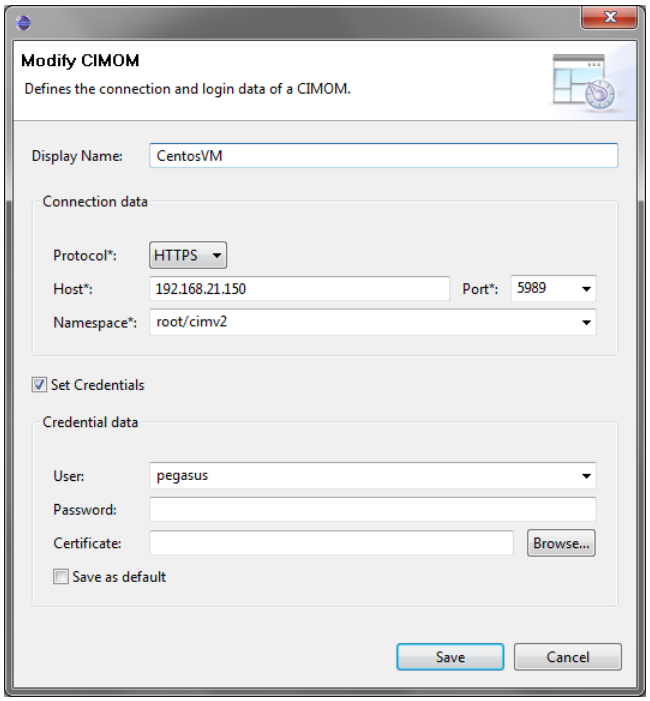
Elindítás után a következő kép fogad.



3. ábra: ECUTE képernyő

Az Explorer használatához először hozzá kell adni a My CIMOMs részhez egy új elemet.

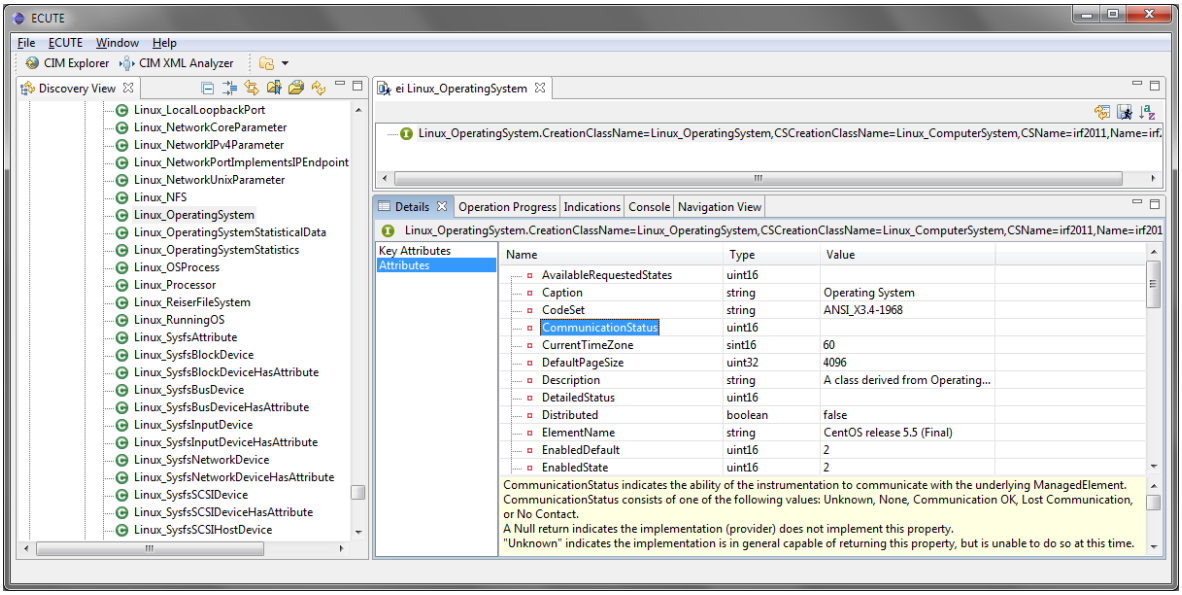
[4] http://sourceforge.net/apps/mediawiki/sblim/index.php?title=Ecute



4. ábra: CIMOM hozzáadása

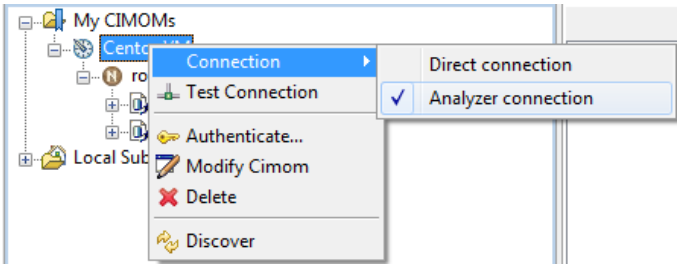
Ezután már tudunk osztályneveket lekérdezni, az osztályok tulajdonságait megjeleníteni, valamint példányokat felsorolni. Az ECUTE súgója elég használható, ha elakadunk, azt érdemes megnézni.

A következő képen a Linux_OperatingSystem példányait kérdeztük le, majd annak egy tulajdonságát nézzük meg.



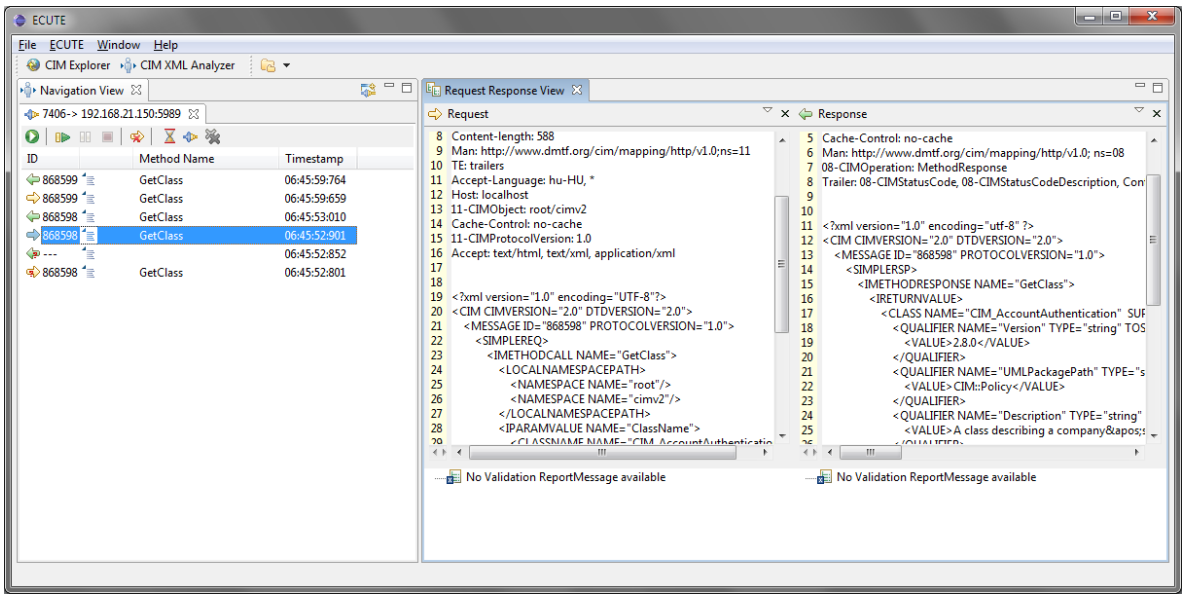
5. ábra: Lekérdezés az ECUTE-ban

Használjuk most az *Analyzer* komponenst is. Ehhez a CIMOM tulajdonságainál a kapcsolat típusát állítsuk át *Analyzer Connection* értékre.



6. ábra: Kapcsolat átállítása Analyzer módba

Innentől kezdve, ha végrehajtunk egy lekérdezést, akkor az az *CIM XML Analyzer* nézetben megjelenik, és a *Request Response View* nézetben meg is tudjuk nézni a tartalmát.



7. ábra: Analyzer nézet használata

Próbáljuk ki az ECUTE-ot:

- Nézzük meg a definiált CIM osztályokat, nézzük meg néhánynak a tulajdonságait.
- Kérdezzük le néhány osztálynak a példányait. Figyelem, sok osztályhoz nincs megfelelő provider, ilyenkor „Not supported” választ fogunk visszakapni. (Tipikusan a Linux kezdetű osztályokhoz van provider).
- Kapcsoljuk be az Analyzert, és nézzünk meg egy-két konkrét CIM XML üzenetet, próbáljuk megtalálni benne, hogy milyen osztályt kérdezünk le.

1.3 Távoli lekérdezés wbemcli segítségével

Ahhoz, hogy távoli lekérdezéseket tudjunk végrehajtani, nyilván kell egy távoli gép is.

1. Távoli gép létrehozása

   Másoljuk le a CentOS virtuális gépet (kikapcsolt állapotban természetesen).

   A másolat indításakor válasszuk az *'I copied'* opciót, hogy biztos új MAC címet kapjon.

   Érdemes megváltoztatni a gép nevét, hogy az SSH ablakban és a konzolon ne keverjük össze a gépeket. Ehhez a következő fájlt kell megváltoztatni:

   ```
   /etc/sysconfig/network
   ```

   Ebben a `hostname` részt kell átírni. A változás után újra kell indítani a gépet (a `reboot` paranccsal).

2. Távoli lekérdezés futtatása

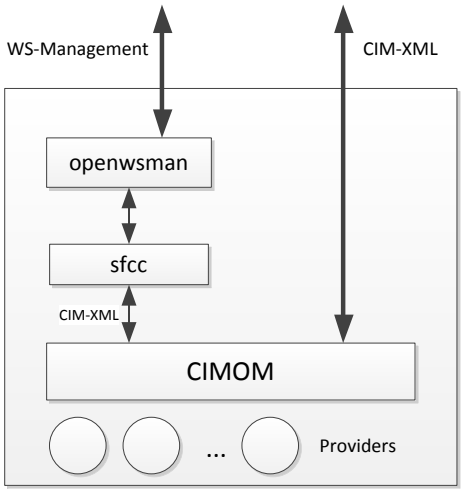
   Mivel amikor helyi lekérdezést futtatunk, akkor is megadtunk minden információt (protokoll, jelszó…), ezért most csak annyit kell tennünk, hogy a localhost helyett a távoli gép IP címét írjuk be.

   ```
   wbemcli gc 'https://pegasus:LaborImage@<tavoli_gep_ip_cim>:5989/root/cimv2:CIM_OperatingSystem'
   ```

   Nyilván ehhez az is kell, hogy a megfelelő port ki legyen engedve a tűzfalon. A kiadott virtuális gépen ki van kapcsolva a tűzfal, ez éles rendszerben nem engedhető meg.

1.4 openwsman

Az *openwsman* egy WS-Management protokollt megvalósító kiszolgáló. Maga konfigurációs adatokat nem kezel, hanem csak egy WS-Management interfészt biztosít egy meglévő CIMOM kiszolgálóhoz. (Tehát a CIMOM kiszolgálót az openwsmantól teljesen függetlenül kell beállítani, az openwsman egyszerűen csak CIM-XML kliensként csatlakozik hozzá.)



8. ábra: openwsman architektúrája

A fenti ábra szemlélteti az openwsman és a szükséges komponensek viszonyát (8. ábra). Az openwsman az SFCC[5] (Small Footprint CIM Client) könyvtárat használja, ami egy C nyelvű API-t ad CIMOM-ok elérésére CIM-XML-en keresztül. (Megjegyzés: az SFCC támogatja egy SfcbLocal nevű speciális kapcsolatot is, amivel az SFCB nevű CIMOM-hoz tud gyorsabb módon csatlakozni).

1. openwsman kiszolgáló beállítása

   Az openwsman a beállításait a következő konfigurációs fájlban tárolja:

   ```
   /etc/openwsman/openwsman.conf
   ```

   (Ha nem csomagkezelővel raktuk fel az openwsmant, hanem forrásból fordítottuk, akkor a fájl a `/usr/local/etc/openwsman/` könyvtár alá kerülhet.)

   Nézzük meg a konfigurációs fájl tartalmát!

   - Milyen porton figyel a szerver?
   - Milyen porton próbál meg csatlakozni a CIMOM-hoz?

   A konfigurációs beállításokról minimális információ található az openwsman README fájlban. Ez tipikusan a `/usr/share/doc` alatt található, a pontos helyét a `locate` segít megkeresni (ez kiírja az összes olyan fájl teljes elérési útját, ami tartalmazza a megadott szöveget):

   ```
   locate openwsman
   ```

   A konfigurációs beállításokról további leírás a *Users Guide*-ban[6] található.

   Többféle hitelesítési módszert lehet alkalmazni, ez lehet GSS, HTTP Digest vagy Basic (a Digest használata már nem javasolt). A felhasználói információkat tárolhatjuk sima password fájlokban vagy PAM (Pluggable authentication modules) segítségével az operációs rendszer felhasználói adatbázisát is használhatjuk (ez utóbbi a javasolt).

2. openwsman kiszolgáló futtatása

   Nézzük meg, hogy milyen kapcsolói vannak az openwsman kiszolgálónak:

   ```
   openwsmand --help
   ```

   Indítsuk el debug módban, ilyenkor a konzol kimenetre írja ki a hibákat és információs üzeneteket.

   ```
   sudo openwsmand -d
   ```

   - Az openwsmand log üzenetei hosszúak, érdemes az SSH ablakot átméretezni, hogy a nagy részük egy sorba kiférjen.

[5] http://sourceforge.net/apps/mediawiki/sblim/index.php?title=Sfcc
[6] http://www.openwsman.org/openwsman-users-guide/configuration-file-options, de a segédlet írásakor nem elérhető az oldal, Google cache-ben még megtalálható.

- Hányféle művelet kezelőjét regisztrálta be az openwsman a kimenet alapján?

A következő paranccsal lehet háttérszolgáltatásként futtatni:

```
sudo /etc/init.d/openwsmand start
```

(Ehhez előbb be kéne konfigurálni az SSL-hez szükséges tanúsítványokat, ami a kiadott virtuális gépen nem történt meg.)

1.5 openwsman lekérdezése a wsmancli parancssal

1. Ismerkedés a *wsmancli* paranccsal.

   A WS-Management protokollhoz használhatjuk a *wsmancli* csomagban lévő `wsman` parancssori eszközt. Nézzük meg először a paraméterezését:

   ```
   wsman --help
   ```

   Így az alapvető kapcsolódáshoz szükséges paramétereket írja ki. A wsman ennél több paraméter kezelésére is képes (pl. event subscription), ezekre most a kezdeteknél nem lesz szükség. Ezeket egyébként a következő paranccsal tudjuk megnézni:

   ```
   wsman --help-all
   ```

   A wsman parancshoz nincs manual oldal, így néha kicsit nehézkes a paraméter pontos jelentését kitalálni. Némi segítséget nyújthat a forráskezelőjében lévő *doc* könyvtárban szereplő *wsman.xml* docbook forrás[7]. Ha ez se segít, akkor pedig a wsman forrásában meg lehet nézni, hogy mit csinál az adott paraméterrel, a fő része egy darab C fájlból áll.

   A legegyszerűbb művelet az IDENTIFY, ez csak lekérdezi a WS-Management kiszolgálót.

   A teszteléshez érdemes megnyitni két külön SSH munkamenetet, az egyikben futtathatjuk az openwsmand programot debug módban, a másikban pedig a wsman parancsokat. Paraméterként adjuk meg, hogy melyik géphez akarunk csatlakozni:

   ```
   wsman identify --hostname localhost
   ```

   (A paramétereknek van itt is rövid neve, a későbbiekben ezt fogjuk használni).

   Hát az eredmény nem túl meggyőző:

   ```
   Connection failed. response code = 0
   couldn't connect to server
   ```

   Ilyenkor következik ugye a hibakeresés. Fut-e az opewsmand? Írt-e ki hibát az elindulásakor? Hova is akarunk pontosan csatlakozni? Szerencsére a wsman is tud elég részletes debug üzeneteket is kiírni (-d kapcsoló), az sokszor segít:

   ```
   [root@irf2011 ~]# wsman identify --hostname localhost -d 6
   Mar 29 14:36:26  cl->authentication.verify_peer: 1
   ```

[7] http://openwsman.git.sourceforge.net/git/gitweb.cgi?p=openwsman/wsmancli;a=blob;f=doc/wsman.xml;

```
Mar 29 14:36:26  *****set post buf len = 238******
* About to connect() to localhost port 5986
*   Trying 127.0.0.1... * Connection refused
...
```

Itt rögtön meg is van a hiba oka. Az SSL portra akar csatlakozni, és nem a sima http portra. (Ha megnézzük a forrását a 2.2.6-os verzióban a *wsman.c*-ben a port alapérték beállításánál elrontották, hogy hogyan működik a ?: operátor C-ben.) Állítsuk be kézzel a portot is:

```
wsman identify --hostname localhost -P 5985
```

Erre válaszként hitelesítést kér. Itt még bármelyik, a rendszerben létező felhasználót megadhatjuk, hisz még csak az openwsman kiszolgálóhoz csatlakozunk, ahol PAM segítségével a helyi felhasználói adatbázist használjuk, és nem adtunk meg semmi jogosultsági korlátot. Később, amikor már az openwsman segítségével a CIMOM kiszolgálót akarjuk lekérdezni, akkor olyan felhasználót kell majd megadni, akinek van joga a CIMOM-hoz is hozzáférni (a kiadott virtuális gépen ugye ez a *pegasus* felhasználó).

Megadhatjuk a hitelesítési információkat paraméterként is:

```
wsman identify --hostname localhost -P 5985 -u pegasus -p LaborImage
```

Így most már végre tudunk hajtani egy egyszerű kérést. Ha a *hostname* paramétert lecseréljük, akkor akár távolról is meg tudjuk szólítani az openwsman kiszolgálónkat.

2. CIM objektumok lekérdezése WS-Management segítségével

Egy egyszerű lekérdezéshez meg kell adni egy műveletet és egy erőforrás URI-t (lásd a kapcsolódó előadásban). A művelet lekérdezéseknél GET vagy ENUMERATE, az erőforrás URI pedig egy névtér prefixből, osztálynévből és esetlegesen példány kiválasztókból (selector) áll.

Nézzünk akkor egy ENUMERATE műveletet valamelyik alap CIM osztályhoz. URI prefixként a DMTF által szabványosított URI-t lehet használni:

```
http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2
```

Az előző részben összerakott paraméterek alapján a lekérdezés így néz ki:

```
wsman enumerate 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor'
-h localhost -P 5985 -u pegasus -p LaborImage
```

Ez eredmény két XML dokumentum lesz. Az első egy *EnumerationContext* azonosítót ad vissza, amin végiglépkedve megkapjuk a következő XML dokumentumokban a lekérdezés eredményét.

```
...
    <wsen:EnumerateResponse>
```

```
      <wsen:EnumerationContext>57f05744-a180-1180-8005
17eee3290c00</wsen:EnumerationContext>
    </wsen:EnumerateResponse>
...
  <s:Body>
    <wsen:PullResponse>
      <wsen:Items>
        <n1:Linux_Processor>
          <n1:AddressWidth xsi:nil="true"/>
          <n1:Availability xsi:nil="true"/>
          <n1:CPUStatus>1</n1:CPUStatus>
...
```

A `-O` paraméter megadásával kérhetjük azt, hogy a minden egyes XML dokumentumot külön fájlba mentsen el.

3. Egy konkrét objektum lekérdezése (GET művelet).

   Az előbbiekben egy adott osztály összes példányát kérdeztük le. Most nézzük hogyan lehet egy konkrét osztályt lekérdezni:

   ```
   wsman get 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor'
   -h localhost -P 5985 -u pegasus -p LaborImage
   ```

   Válaszként egy *Fault* üzenetet kapunk, aminek a *Text* mezője tartalmazza a hiba okát:

   ```
   The Selectors for the resource are not valid.
   ```

   A hiba teljesen jogos, hisz nem adtuk meg, hogy melyik konkrét példányt akarjuk lekérdezni. Ehhez kéne az úgynevezett kiválasztókat (selector) még hozzáfűzni az URI-hoz. Az osztály összes kulcs attribútumát meg kéne itt adni. Az, hogy melyek ezek, a legkönnyebben egy wbemcli-s lekérdezéssel deríthetjük ki:

   ```
   wbemcli -nl ei
   'https://pegasus:LaborImage@localhost:5989/root/cimv2:CIM_Processor'
   ```

   Az elején kiírja a példány teljes nevét:

   ```
   localhost:5989/root/cimv2:Linux_Processor.CreationClassName="Linux_Processor",Dev
   iceID="0",SystemCreationClassName="Linux_ComputerSystem",SystemName="irf2011.loca
   ldomain"
   ```

   Ennek tehát négy darab kulcsa van, és mindet meg kell majd adnunk.

   ```
   wsman get 'http://schemas.dmtf.org/wbem/wscim/1/cim-
   schema/2/CIM_Processor?CreationClassName=Linux_Processor,DeviceID=0,SystemCreatio
   nClassName=Linux_ComputerSystem,SystemName=irf2011.localdomain' -h localhost -P
   5985 -u pegasus -p LaborImage
   ```

   Arra figyeljünk, hogy itt nem kell idézőjelek közé rakni az attribútumok értékét. Az eredményként vissza is kapjuk a keresett példányt.

4. Nem DMTF szabványos osztály lekérdezése.

Az előző feladatban a wbemcli kimenetében láthattuk, hogy a lekérdezett CIM_Processor tulajdonképpen a Linux_Processor példánya. Próbáljuk meg akkor ezt lekérdezni, a fenti parancsban csak írjuk át az osztály nevét. Az eredmény a következő hiba:

```
No route can be determined to reach the destination role defined by the WS-
Addressing To.
```

A probléma az, hogy nem jó névtér prefixet használtunk, hisz ezt az osztály már nem a szabványos CIM séma tartalmazza. Az openwsman konfigurációs beállításai között megtalálható, hogy milyen további plusz sémákat képes kiszolgálni, ezek között szerepel a Linux is:

```
Linux=http://sblim.sf.net/wbem/wscim/1/cim-schema/2
```

Használjuk tehát ezt a prefixet:

```
wsman get 'http://sblim.sf.net/wbem/wscim/1/cim-
schema/2/Linux_Processor?CreationClassName=Linux_Processor,DeviceID=0,SystemCreat
ionClassName=Linux_ComputerSystem,SystemName=irf2011.localdomain' -h localhost -P
5985 -u pegasus -p LaborImage
```

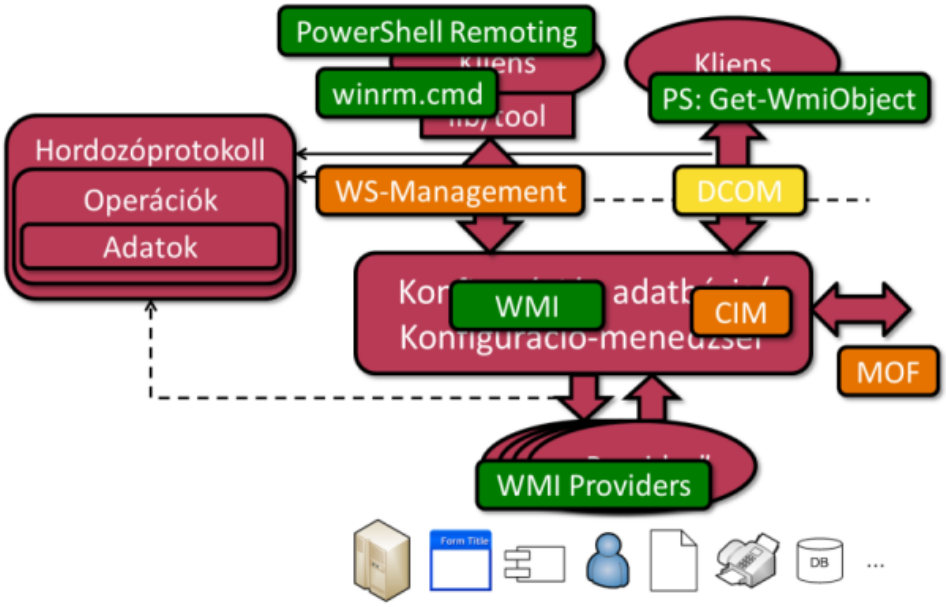
Legnagyobb megelégedésünkre így már működik ez a lekérdezés is.

Ezek alapján egy Linuxon futó CIMOM-ból tudunk helyileg és távolról adatokat lekérdezni, még pedig CIM-XML és WS-Management protokollon keresztül is.

2 Windows: WMI, WinRM

A feladatokat egy Windows 7 virtuális gépen fogjuk végrehajtani. A Windows 7 már alapból tartalmazza a CIMOM-ot (WMI Object Manager), a WS-Management klienst és kiszolgálót (WinRM) és az ezek kezeléséhez szükséges PowerShell 2.0-s cmdleteket, így a feladatunk pusztán annyi lesz, hogy ezeket megfelelően beállítsuk.

Az általános konfigurációkezelési ábránkat tehát a következő módon fedik le az eszközök.



9. ábra: Konfigurációkezelési technológiák Windowsra

2.1 WMI használata

A következő feladatban egyszerű WMI lekérdezéseket fogunk végrehajtani helyi és távoli gépen.

1. PowerShell WMI Explorer

   Hogy könnyebben eligazodjunk, hogy milyen CIM osztályok és példányok vannak a rendszerben, érdemes a *PowerShell WMI Explorer WPF Edition*[8] kis segédprogramot használni. Ennek használatához a PowerShell konzolt a *-sta* paraméterrel kell indítani.

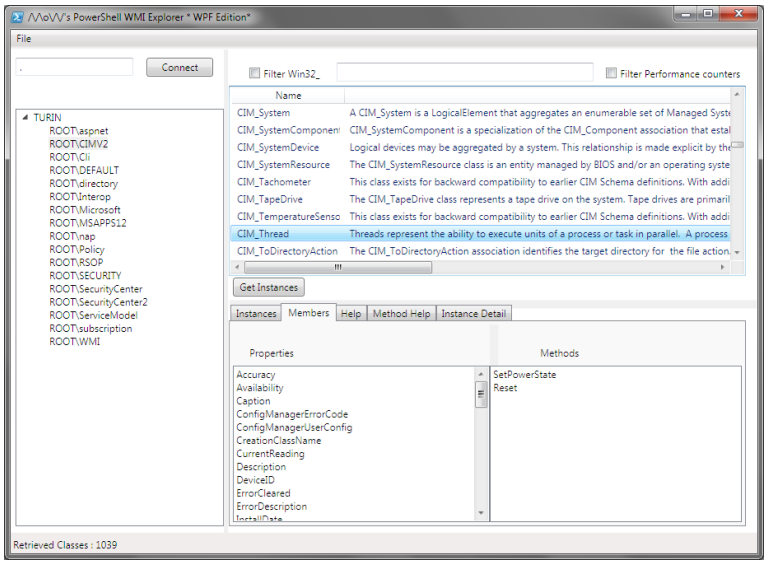
   ```
   powershell -sta
   ```

   Majd elindítani a letöltött szkriptet:

   ```
   .\WpfWmiExplorer.ps1
   ```

   A következőhöz hasonló kép fogad majd minket:

[8] http://thepowershellguy.com/blogs/posh/pages/powershell-wmi-explorer.aspx



10. ábra: PowerShell WMI Explorer WPF Edition

Itt ki lehet listázni az egyes névterekben szereplő osztályokat, azok részletes leírását, valamint le lehet kérdezni a példányaikat.

(Ennél egy picit több funkcióit valósít meg a wbemtest.exe is, csak annak nehezebben használható a GUI-ja.)

2. WMI kezelésére szolgáló cmdletek

Keressük meg WMI kezelésére szolgáló cmdleteket:

```
Get-Command -type Cmdlet | ? {$_.Name -like "*WMI*"}
```

Ezek közül mi legtöbbször a *Get-WmiObject* cmdletet fogjuk használni, de érdemes a többiről is tudni.

Nézzük néhány példát a Get-WmiObject használatára:

```
Get-Help Get-WmiObject -examples
```

Ezzel nagyjából képet lehet kapni, hogy mit tud a Get-WmiObject. Érdemes viszont rászánni az időt, hogy a teljes leírást is megnézzük (ezt egyszer úgyis meg kell tenni), különben folyamatosan elakadunk majd csak később.

```
Get-Help Get-WmiObject -full | more
```

Ezzel a tudással felvértezve most már le is tudunk valamit kérdezni.

```
Get-WmiObject CIM_Memory
```

Válaszként *System.Management.ManagementObject* típusú objektumokat kapunk vissza, amiknél szépen tulajdonságokkal érjük el az egyes attribútumokat. Ez alapján már könnyen tudunk szűrni vagy rendezni a PowerShellben korábban megtanult módon.

```
Get-WmiObject CIM_Memory | select Name, Status, InstalledSize | where
{$_.InstalledSize -gt 1024}
```

- Keressünk ki további 2 CIM osztályt, és kérdezzük le azok példányait.
- Szűrjük az így visszakapott listát, majd számoljuk ki a maximumát az egyik tulajdonságnak.

3. WMI lekérdezés távolról

   Ha egy távoli gépről akarunk információt lekérni, akkor csak a -ComputerName paramétert kell megadni:

   ```
   Get-WmiObject CIM_Memory -ComputerName 192.168.21.151
   ```

   Erre alapesetben a következő hibaüzenet a válasz:

   ```
   The RPC server is unavailable. (Exception from HRESULT: 0x800706BA)
   ```

   Ahogy az előadáson is szerepelt, a WMI távoli lekérdezésekhez DCOM-ot használ, és az ehhez szükséges portok nincsenek kinyitva alapértelmezetten. A fólián megadott paranccsal nyissuk ki a távoli gépen a megfelelő portot.

   ```
   netsh firewall set service RemoteAdmin enable
   ```

   Most megismételve a távoli lekérdezést már más hibát kapunk:

   ```
   Access is denied. (Exception from HRESULT: 0x80070005 (E_ACCESSDENIED))
   ```

   A Windows ilyenkor a helyi felhasználónk adataival próbált áthitelesíteni. Ha ez nem létezik a távoli gépen, akkor ilyen hibát kaphatunk. A *-Credential* paraméterrel tudjuk megadni, hogy milyen felhasználó nevében csatlakozzon. Ilyenkor egy dialógusablakban vagy a konzolon a parancs futtatásakor bekéri hozzá a jelszót is.

   Az eredmény továbbra is *Access Denied*, holott olyan felhasználót adtunk meg, aki tagja a távoli gépen az *Administrators* csoportnak. Ez azért van, mert Vista óta a *User Account Control (UAC)* funkció miatt az ilyen felhasználók távoli hozzáférés esetén olyan biztonsági tokent kapnak, amiben nincsenek benne a rendszergazdai jogok. Bővebben lásd a megfelelő KB cikket [6]. A következő registry kulcs értékét kell egyre állítani:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\
   LocalAccountTokenFilterPolicy
   ```

   Így már sikeresen le lehet távoli gépek adatait is kérdezni.

4. Szűrés távoli lekérdezés esetén

   Arra figyeljünk oda, hogy távoli lekérdezés esetén csak a szükséges adatokat kérdezzük le, és inkább a távoli gépen szűrjünk. Erre valók a WQL lekérdezések. A részleteket lásd az előadás fóliákon, itt most csak egy egyszerű példát nézünk:

```
Get-WmiObject -ComputerName 192.168.21.151 -Credential meres -Query "Select
name, state FROM Win32_Service WHERE StartMode = 'Auto'"
```

WQL segítségével bonyolultabb lekérdezéseket is meg tudunk fogalmazni.

5. Még egy érdekes feladat van hátra, mégpedig az, amikor szeretnénk egy adott példányhoz kapcsolódó másik példányokat lekérdezni.

   A CIM-ben erre a kapcsolóosztályok (association class) szolgálnak. Például a *Win32_DiskDriveToDiskPartition* a *Win32_DiskDrive* és a *Win32_DiskPartition* osztályokat köti össze, a *Win32_SoftwareFeatureSoftwareElements* pedig egy *Win32_SoftwareFeature* példányhoz tartozó *Win32_SoftwareElement* példányokat adja meg.

   Ilyen osztályok mentén navigálhatunk kézzel is, de egyszerűbb az `ASSOCIATORS OF` kulcsszó használata [9]. Például így kérdezhetjük le egy konkrét *Win32_LogicalDisk* példányhoz tartozó összes kapcsolódó példányt.

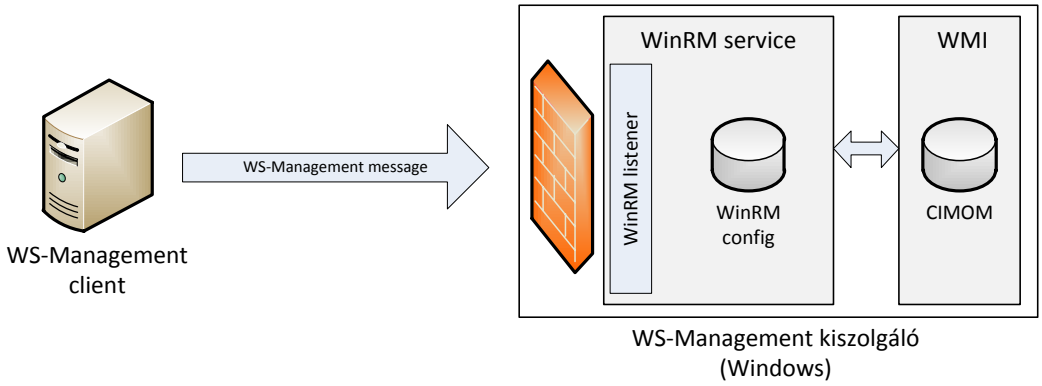
   ```
   Get-WmiObject -Query "ASSOCIATORS OF {Win32_LogicalDisk.DeviceID='C:'}"
   ```

   Ez eredményként az összes kapcsolóosztályt megvizsgálja, így kapunk például *Win32_ComputerSystem*, *Win32_DiskPartition* és *Win32_Directory* példányt is. Ha csak egy konkrét kapcsolatra vagyunk kíváncsiak, akkor az `AssocClass` szűrőfeltételt lehet használni.

   Érdemes megnézni az ASSOCIATORS OF leírását, még tud további hasznos dolgokat.

2.2 WinRM használata

A WinRM a Microsoft WS-Management protokollt használó távoli menedzsment implementációja.



11. ábra: A WinRM architektúrája

A WinRM maga egy szolgáltatás, ami WS-Management üzeneteket fogad (vagy küld), és az abban megfogalmazott kéréseket végrehajtja a megadott erőforráson, pl. egy WMI példányon. A szolgáltatás alapesetben a megvalósítást adó kódból, egy saját belső konfigurációs adatbázisból, és egy vagy több úgynevezett figyelőből (listener) áll, amik megadják, hogy melyik helyi IP címen, milyen porton és milyen csatornán (HTTP, HTTPS)

figyeljen a WinRM szolgáltatás. A WS-Management kérések fogadásához a megadott porton természetesen a tűzfalon is át kell engedni.

1. Ismerkedés a WinRM-mel

   Gyors áttekintést ad a WinRM-ről, valamint arról, hogy hogyan tudjuk PowerShellből kezelni a következő súgó téma:

   ```
   Get-Help about_wsman | more
   ```

   Ezt érdemes elolvasni, sokat segít az orientálásban.

   A WinRM engedélyezését végzi el a `Set-WSManQuickConfig` cmdlet. A leírásában benne van, hogy mit csinál:

   ```
   The cmdlet performs the following:
   1. Checks whether the WinRM service is running. If the WinRM service is not
   running, the service is started.
   2. Sets the WinRM service startup type to automatic.
   3. Creates a listener to accept requests on any IP address. By default, the
   transport is HTTP.
   4. Enables a firewall exception for WinRM traffic .
   ```

   A végrehajtásához rendszergazdaként kell elindítani a PowerShell konzolt.

   Ezek a beállítások csak ahhoz kellenek, ha a WinRM-es gépünket kiszolgálóként akarjuk használni, tehát róla akarunk adatokat lekérdezni. Ha a WinRM-es gép a kommunikációban kliensként viselkedik, tehát ő kérdez le adatokat, akkor elég csak annyi, hogy a WinRM szolgáltatást elindítjuk kézzel.

   Meg lehetne adni neki egy `-UseSSL` kapcsolót is, ilyenkor nem a WS-Management protokollhoz rendelt HTTP porton (5985) hanem a HTTPS porton (5986) figyelni. Ehhez viszont kell a gépre telepíteni egy tanúsítványt, ami a számítógép nevére szól (ez lehet akár nem hiteles tanúsítvány is, amit pl. az OpenSSL program segítségével generáltunk magunknak).

   A parancs végrehajtásakor kaphatjuk a következő hibaüzenetet:

   ```
   WinRM firewall exception will not work since one of the network connection types
   on this machine is set to Public. Change the network connection type to either
   Domain or Private and try again.
   ```

   Ehhez át kell állítani a *Network and Sharing Center* ablakban a hálózati hely típusát Workre (figyelem, ennek például az is a következménye, hogy megnyitja a fájlmegosztásokhoz tartozó portot). Vagy ha ez nem lehetséges (mert például olyan VMware virtuális interfész van a gépen, amit nem lehet a Public profilról átállítani) vagy nem akarjuk, akkor a Set-WSManQuickConfig által elvégzett lépéseket kézzel is megcsinálhatjuk [7].

   Ha ellenőrizni akarjuk, hogy sikerül-e távolról csatlakozni, akkor használjuk a *Test-WSMan* cmdletet. Ez a WS-Management IDENTIFY műveletét használja.

```
Test-WSMan -ComputerName 192.168.21.151
```

Ilyenkor névtelen módon csatlakozik a távoli géphez.

2. Egyszerű lekérdezés WinRM segítéségével

Vegyük elő valamelyik szokásos CIM osztályunkat, és kérdezzük le a példányait WS-Management segítéségével. Erre jó a `Get-WSManInstance` cmdlet. Ennek is egy erőforrás URI-t kell megadni. Szerencsére itt használhatunk aliasokat, nem kell a teljes prefixet mindig kiírni. Az aliasok listája itt található:

```
winrm help alias
```

Nézzünk akkor ez alapján egy egyszerű Enumerate kérést.

```
Get-WSManInstance wmicimv2/CIM_Processor -ComputerName 192.168.21.151 -Enumerate
```

Erre egy nagyon tipikus választ fogunk kapni:

```
The WinRM client cannot process the request. If the authentication scheme is
different from Kerberos, or if the client computer is not joined to a domain,
then HTTPS transport must be used or the destination machine must be added to the
TrustedHosts configuration setting. Use winrm.cmd to configure TrustedHosts. Note
that computers in the TrustedHosts list might not be authenticated. You can get
more information about that by running the following command: winrm help config.
```

Olvassuk végig a hibaüzenetet, mert pontosan leírja, hogy mi a gond, és mi a megoldás. Mivel nem tartományi környezetben vagyunk, így nem lehet Kerberost használni. Ekkor ha nem akarunk SSL-t használni, akkor a távoli gépet fel kell venni a helyi gép TrustedHosts listájába, és ezzel vállalva, hogy küldhetünk neki nem titkosított tartalmat is.

A WinRM beállításait a winrm.cmd parancssori eszközzel módosíthatjuk, de használhatjuk erre a WSMAN: PowerShell providert is[9]. Nyissunk egy új PowerShell konzolt *rendszergazdaként*, majd hajtsuk végre a következetőket:

```
cd WSMAN:
cd localhost
ls
```

Figyelem: ehhez a rendszergazda jogú felhasználónknak kell jelszóval rendelkeznie, ha üres a jelszava, akkor *Access Denied* hibát kapunk.

Itt találhatók a WinRM beállításai: kliens, szolgáltatás, és a bejövő kéréseket fogadó úgynevezett listenerek (ezek döntik el, hogy melyik IP-n és milyen porton figyel a WinRM). Egy friss Windows 7 gépen a következőt kapjuk:

[9] Harmadik lehetőség, hogy csoportházirendet használunk erre. Ezt akár nem tartományi gépen is megtehetjük. Nyissunk meg egy MMC-t, adjuk hozzá a *Group Policy Object Editor* snap-in konzolt a helyi gépen. A beállítások a *Computer Configuration / Administrative Templates / Windows Components / Windows Remote Management* rész alatt találhatóak.

```
   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost

Name                        Value
----                        -----
MaxEnvelopeSizekb           150
MaxTimeoutms                60000
MaxBatchItems               32000
MaxProviderRequests         4294967295
Client
Service
Shell
Listener
Plugin
ClientCertificate
```

Most nekünk a helyi gép kliens beállításait kell módosítani, így váltsunk át arra.

```
cd Client
ls
```

- Nézzük végig, hogy milyen főbb beállítások vannak?
- Milyen hitelesítési módszerek vannak jelenleg a kliensben engedélyezve?

Állítsuk át a TrustedHosts értékét, adjuk hozzá a távoli gép IP címét is.

```
Set-Item .\TrustedHosts -Value "192.168.21.151"
```

(Arra figyeljünk, hogy ha volt már valami korábban a TrustedHosts mezőben, akkor annak az értékét tartsuk meg, ha kell még. Az egyes elemek vesszővel vannak elválasztva a TrustedHosts értékében.)

Ha ezzel megvagyunk, akkor elvileg már megy a lekérdezés.

```
Get-WSManInstance wmicimv2/CIM_Processor -ComputerName 192.168.21.151 -Credential
meres -Enumerate
```

3. Szűrés távoli lekérdezésben

   Többféle módon lehetne szűrni, itt most csak a legegyszerűbbet nézzük meg, amikor az URI-ba írjuk be a szűrőfeltételeket. A többi módszert lásd a parancs leírásában.

   A szűréshez megint ki kéne találni, hogy mi az adott osztály kulcsa. Segítségként megint egy WMI lekérdezéssel megnézhetjük az adott példány adatait:

   ```
   Get-WmiObject CIM_Processor | select __PATH
   ```

   Itt szerepel az egyes példányok teljes neve. Arra figyeljünk csak, hogy ez CIM specifikus név, ezt még alakítgatni kell, hogy WS-Management konform legyen.

   ```
   Get-WSManInstance "wmicimv2/Win32_Processor?DeviceID=CPU0" -ComputerName
   192.168.21.151 -Credential meres
   ```

4. Töredék (fragment) kezelés

Ha a visszaadott válaszból csak egy konkrét értékre vagyunk kíváncsiak, akkor a kérhetjük, hogy csak azt adja vissza a kiszolgáló. Erre a `-Fragment` paraméter való.

A paraméter pontos jelentését a WS-Man szabvány leírásában találhatjuk (7.7 Fragment-Level Access) [1]. Eszerint a visszaadandó XML dokumentumhoz egy XPath lekérdezést adhatunk meg, és az az által kijelölt XML csomópontokat kapjuk csak vissza. A szabvány függeléke tartalmazza, hogy az XPath milyen részhalmaza támogatott, ám ez elég szűk (a WinRM a *Level 1* szintet implementálta).

Valami hasonló szerkezetű XML dokumentumot kapunk általában vissza (ezt WireSharkból tudjuk például megnézni, vagy a *winrm.cmd*-ben végrehajtva a lekérdezést a *-format:pretty* kapcsolóval):

```
<wsman:Results xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman/results">

<p:Win32_Processor xsi:type="p:Win32_Processor_Type" xml:lang="en-US" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://schemas.microsoft.co
m/wbem/wsman/1/wmi/root/cimv2/Win32_Processor" xmlns:cim="http://schemas.dmtf.or
g/wbem/wscim/1/common">
    <p:AddressWidth>32</p:AddressWidth>
    <p:Architecture>9</p:Architecture>
    <p:Availability>3</p:Availability>
...
```

A támogatott XPath kifejezések alapján (pl. nincsen „a | b” operátor) ez praktikusan azt jelenti, hogy egy-egy csomópontot tudunk kiválasztani a -Fragment segítségével.

Ráadásul -Enumerate esetén nem lehet a WinRM kliensben töredéket megadni, különben a következő hibát kapjuk:

```
Parameter set cannot be resolved using the specified named parameters.
```

5. Kapcsolóosztályok használata

Zárásként nézzük meg itt is, hogy hogyan lehet a kapcsolóosztályok mentén adatokat lekérdezni.

Két Windows esetén „csalhatunk”, és a lekérdezés dialektusának megadhatunk WQL lekérdezéseket, ilyenkor például működik az ASSOCIATORS OF kulcsszó.

Azonban ha tényleg platform-független módon szeretnénk kezelni a kapcsolatokat, akkor a WS-Management Association típusú szűrőjét kell használni. A témához kapcsolódó hivatalos WinRM dokumentáció [10] viszonylag szűkszavú, a Get-WSManInstance dokumentációja pedig hibás (például egyes verziókban még szerepel a már aktuálisan nem létező -References kapcsoló). Egy jó összefoglaló található itt [11] a témáról.

A lekérdezések pontos jelentését a DMTF szabványban találhatjuk meg (8.2 Association Queries) [2]. Nézzünk itt most egy konkrét példányt, hogy WinRM-ből hogyan lehet ezt használni.

```
Get-WSManInstance -ComputerName 192.168.21.151 -ResourceURI wmicimv2/* -Dialect
Association -Filter "{object=Win32_LogicalDisk?deviceid=C:}" -Enumerate
-Credential meres -Associations
```

Figyelem: kapcsolatok lekérdezésekor a kliensen a PowerShellt is rendszergazdai jogokkal kell futtatni, különben hozzáférési hibát kapunk.

Ennek eredményeképp a kapcsolatok példányait kapjuk meg (a -Associations kapcsoló miatt), és nem a kapcsolódó példányokat. Ilyenkor hasonló kimenetet láthatunk:

```
type           : p:Win32_LogicalDiskRootDirectory_Type
GroupComponent : GroupComponent
PartComponent  : PartComponent

type            : p:Win32_LogicalDiskToPartition_Type
Antecedent      : Antecedent
Dependent       : Dependent
```

A GroupComponent és az Antecedent összetett objektumok, azért nem írja ki a konkrét értékét. De a konkrét visszakapott objektumon a következő tulajdonságok mentén elérjük:

```
.GroupComponent.ReferenceParameters.SelectorSet.Selector
```

A másik lehetőség, hogy a konkrét példányokat kérjük vissza:

```
Get-WSManInstance -ComputerName 192.168.21.151 -ResourceURI wmicimv2/* -dialect
association -filter
"{Object=Win32_DiskDrive?deviceID=\\\\.\\PHYSICALDRIVE0;AssociationClassname=Win3
2_DiskDriveToDiskPartition}" -enumerate -credential meres
```

Tipikus hibaüzenet szokott lenni a következő:

```
Get-WSManInstance : The data source could not process the filter. The filter
might be missing, invalid or too complex to process. If a service only supports a
subset of a filter dialect (such as XPath level 1), it may return this fault for
valid filter expressions outside of the supported subset. Change the filter and
try the request again.
```

Ezt akkor kaphatjuk, ha például a kapcsolóosztályok lekérése során is megpróbáljuk megadni az AssociationClassName szűrőt. Figyeljük meg a két eset leírásában [11], hogy különböző elemeket használhatunk szűrőben.

3 Platformok közötti lekérdezések

Az igazi erejét az adja ezeknek a szabványoknak és technológiáknak, hogy a lekérdezések platformok között is működnek. Nézzünk erre egy példát.

3.1 WS-Management Linux klienssel és Windows szolgáltatással

Nézzük most azt az esetet, amikor kliensnek a wsmancli programot használjuk Linuxról, a kiszolgáló pedig egy WinRM-et használó Windows lesz.

1. Kapcsolódás kipróbálása

```
wsman identify -h 192.168.21.151 -P 5985
```

Erre *authentication failed* választ kapunk. Ki kéne akkor választani, hogy milyen hitelesítési módszert használjunk. A wsman tud HTTP Basic-et és GSS-t (amivel Kerberost lehet használni).

A WinRM kiszolgáló beállításait így nézhetjük meg (rendszergazdai PowerShell konzolból):

```
cd wsman:
cd .\localhost\Service\Auth
ls
```

Eredmény:

```
Name                    Value
----                    -----
Basic                   false
Kerberos                true
Negotiate               true
Certificate             false
CredSSP                 false
```

Kerberost ugyan mindkettő tud, ahhoz azonban kéne egy Kerberos szerver (például egy Active Directory tartományvezérlő). Ha ez nincs, akkor marad a Basic. Ez nem titkosítva küldi a jelszót, de legalább biztos mindenki ismeri.

Tehát engedélyezni kell a Basic hitelesítési módszert a WinRM kiszolgálón is:

```
Set-Item WSMan:\localhost\Service\Auth\Basic -Value true
```

Egy dolog kell még a WinRM oldalán. Alapból nincs engedélyezve, hogy titkosítatlan csatornán kommunikálhat bárkivel is, ezért ezt a hibaüzenetet kapjuk:

```
Connecting to remote server failed with the following error message : The WinRM
client cannot process the request. Unencrypted traffic is currently disabled in
the client configuration. Change the client configurati on and try the request
again.
```

Így ezt most be kell kapcsolni:

```
Set-Item WSMan:\localhost\Service\AllowUnencrypted -value true
```

Vissza a wsman klienshez, és most adjuk meg, hogy Basic hitelesítést akarunk használni, és mi a felhasználó és jelszó:

```
wsman identify -h 192.168.21.151 -P 5985 --auth basic -u meres -p LaborImage
```

Most már megy tökéletesen az IDENTITY művelet.

Ha gondunk lenne, első lépésként érdemes WireSharkban megnézni a forgalmat, és például ellenőrizni, hogy jól adja-e át a Base64 segítségével kódolt felhasználót és jelszót, lefolyik-e rendesen a HTTP Basic hitelesítés, stb. Például ha a Windows gép a kliens, akkor előfordul néha az a probléma, hogy a felhasználónév elé egy \ jelet rak, és ez csak a hálózati forgalomban látszik[10].

2. Adatok lekérdezése

Nézzünk akkor kezdésnek egy ENUMERATE műveletet:

```
wsman enumerate
'http://schemas.microsoft.com/wbem/wsman/1/wmi/root/cimv2/CIM_Processor' -h
192.168.21.151 -P 5985 --auth basic -u meres -p LaborImage -O wsman.out
```

Két dologra kell figyelni. Az egyik, hogy ugye emlékszünk, hogy az aktuális wsmancliben az Enumerate csak kimeneti fájl megadásakor megy. A másik, hogy az erőforrás URI-ban itt a Microsoft specifikus prefixet használtuk. A DMTF által szabványosított prefix használata nem megy tökéletesen [8].

Nézzünk meg egy GET műveletet is:

```
wsman get
'http://schemas.microsoft.com/wbem/wsman/1/wmi/root/cimv2/Win32_Processor?DeviceID=CPU0'
-h 192.168.21.151 -P 5985 --auth basic -u meres -p LaborImage
```

Ez a része is működik, így különböző platformok között is tudunk adatokat lekérdezni. Bár elsőre egy XML fájl visszakapása nem tűnik nagy haditettnek, de ha belegondolunk ez egy egész sor érdekes alkalmazás előtt nyitja meg az utat.

3. Biztonsági beállítások

Az első részben beállított beállításokkal megy a rendszer, csak éppen semmi védelem nincsen benne. A jelszavak és a teljes forgalom nyílt szövegként utazik, egyik fél sem ellenőrzi a másik kilétét.

A következő lépés az lenne, hogy ezeket beállítjuk valami elfogadható szintre. Több lehetőségünk is van.

- Kerberos használata: akár AD, akár egy linuxos LDAP címtárral.

[10] Ilyenkor megoldás lehet például, hogy a Get-Credential cmdletet átállítjuk, hogy konzolon kérje be a jelszót: http://social.technet.microsoft.com/Forums/pl-PL/winserverpowershell/thread/8eb1a046-acbf-4fda-b4b3-e7599dcf53a3

- SSL használata: ez a teljes forgalmat titkosítaná. Ehhez a windowsos gépen kell egy tanúsítványt telepíteni, majd átállítani a listenert.

3.2 WS-Management Windows klienssel és Linux kiszolgálóval

Nézzük most meg a másik esetet, tehát a WinRM lesz a kliens, és egy openwsmand a kiszolgáló. Az előző fejezetben elmondottak továbbra is érvényesek, a Basic lesz a közösen ismert hitelesítési protokoll, és egyelőre sima HTTP-n próbálkozunk.

1. Biztonsági beállítások a kliensen.

   Első lépésben engedélyezzük a Basic hitelesítés a WinRM kliens beállításaiban.

   ```
   Set-Item WSMan:\localhost\Client\Auth\Basic -Value true
   ```

   Ha csatlakozni próbálnánk, akkor a TrustedHosts beállításra panaszkodik, így állítsuk be azt a 2.2 fejezetben megismert módon:

   ```
   cd wsman:
   cd localhost\client
   Set-Item .\TrustedHosts -Value "192.168.21.151"
   ```

   (Az IP címre természetesen helyettesítsük be a kiszolgáló IP címét.)

2. Kiszolgáló beállítása

   A kiszolgáló oldalon ellenőrizzük, hogy fut-e a CIM szerver és az openwsmand kiszolgáló, be van-e állítva a Basic hitelesítés a konfigurációs állományában, végül pedig, hogy helyileg tudunk-e lekérdezni róla.

3. Kapcsolat ellenőrzése

   Ezután már csak ellenőrizni kell a kapcsolatot.

   ```
   Test-WSMan -ComputerName 192.168.21.150 -Authentication basic -Credential pegasus
   ```

   Ha sikeres, akkor próbáljunk lekérdezni valamit.

   ```
   Get-WSManInstance -ComputerName 192.168.21.150 -Authentication basic -Credenti
   al pegasus -ResourceURI cimv2/CIM_Processor -Enumerate
   ```

   Arra figyeljünk csak, hogy itt most a platform-független (DMTF) vagy a Linux-specifikus URL prefixeket és osztályneveket kell használni.

4 Összefoglalás

A gyakorlat során megnéztünk különböző konfigurációkezelési technológiákat a gyakorlatban. A célunk az volt, hogy különböző platformról tudjunk változatos konfigurációs adatokat lekérdezni, és ezeket az alkalmazásainkban felhasználni.

Az általános módszer és szabványkészlet a következő volt. A CIM definiált egy általános adatmodellt, ilyen modelleket tárolnak a CIMOM-jaink. A CIMOM-ot többféle protokollon lehet elérni, a CIM-XML és a WS-Management protokollokat néztük.

Linux platformon megismerkedünk az OpenPegasus CIMOM-mal, és a CIM-XML protokollt használó wbemcli lekérdező eszközzel. A CIMOM-unk számára egy WS-Management interfészt biztosított pluszban az openwsman, ezt a wsmancli eszközzel tudtuk lekérdezni.

Windows esetén a WMI biztosította a CIMOM-ot. Ezt vagy a nem szabványos DCOM felületen lehet elérni, vagy pedig a WinRM által biztosított WS-Management felületen. Mindkét esetben PowerShell cmdleteket használtunk a lekérdezésekhez.

5 További információ

Általános szabványok

[1] DMTF. „Web Services for Management (WS Management)”, 1.1.0, DSP0226, 2010. URL: http://dmtf.org/sites/default/files/standards/documents/DSP0226_1.1.pdf

[2] DMTF. „WS-Management CIM Binding Specification”, DSP0227. URL: http://www.dmtf.org/sites/default/files/standards/documents/DSP0227_1.1.0.pdf

Linux

[3] Praveen Kumar Paladugu. „WBEM Based Management in Linux”, Dell Technical White Paper, http://en.community.dell.com/dell-blogs/enterprise/b/tech-center/archive/2011/02/28/wbem-based-management-in-linux.aspx

[4] RMS’s GDB Debugger Tutorial, URL: http://www.unknownroad.com/rtfm/gdbtut/

[5] Micskei Zoltán, „Openwsman telepítése CentOS-re”, URL: http://blog.inf.mit.bme.hu/?p=77 (Vigyázat, a leírás már némileg elavult, ez az openwsman korábbi verziójához készült!)

Windows

[6] Microsoft, Description of User Account Control and remote restrictions in Windows Vista, Knowledge Base article 951016, http://support.microsoft.com/kb/951016

[7] Micskei Zoltán, „WinRM teszt két Vista között”, http://blog.inf.mit.bme.hu/?p=63 (Vigyázat, a leírás már némileg elavult, ez WinRM 1.0-hoz készült!)

[8] Micskei Zoltán, „CIM osztályok lekérdezése WinRM-ben DMTF URI-val”, http://blog.inf.mit.bme.hu/?p=144

[9] MSDN. „ASSOCIATORS OF Statement”,http://msdn.microsoft.com/en-us/library/aa384793%28v=vs.85%29.aspx

[10] MSDN. „DMTF Profile Discovery Through Association Traversal”, http://msdn.microsoft.com/en-us/library/ee309363%28VS.85%29.aspx

[11] WMI Blog. „Association Traversal Using WSMan cmdlets”, http://blogs.msdn.com/b/wmi/archive/2009/05/02/association-traversal-using-wsman-cmdlets.aspx

6 Függelék

6.1 wsmancli 2.2.5 segmentation fault hiba

A wsmancli 2.2.5-ös verziójában egy egyszerű ENUMERATE kérés esetén is segmentation fault hibával leállt a program. Tanulságos lehet, hogy hogyan lehet megkeresni a hiba okát, és azt egyszerűen megoldani.

```
wsman enumerate 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor'
-h localhost -P 5985 -u pegasus -p LaborImage
```

A lekérdezés eredménye a wsman 2.2.6-os verziójával:

```
<s:Envelope xmlns:s=http://www.w3.org/2003/05/soap-envelope
...
  <wsen:EnumerateResponse>
      <wsen:EnumerationContext>c229af53-9fa3-1fa3-8002</wsen:EnumerationContext>
  </wsen:EnumerateResponse>
..
</s:Envelope>
Segmentation fault
```

Visszakapjuk tehát a WS-Management protokoll üzenetét egy SOAP[11] borítékban, azonban a feldolgozás közben segmentation faultot kapunk. Nem túl szép.

Mit lehet ilyenkor tenni? Nyilván az egyik lehetőség, hogy feladjuk, és elkezdünk panaszkodni. Ettől azonban a probléma még nem fog megoldódni. Egy fokkal jobb megoldás, ha rákeresünk, hogy találkozott-e más ezzel a problémával. Sajnos túl sok találat nincs a releváns kereső kifejezésekre (pl. „wsman segmentation fault enumeration"). Akkor most mit tegyünk? Ha a Google se tudja a választ, akkor hogyan tovább?

Első körben jó lenne tudni, hogy pontosan hol akad el a program. Ezt egy debuggerrel könnyen meg tudjuk nézni. Linux alatt a `gdb` az egyik parancssori gyakran használt parancssori debugger. Ehhez elég sok gyorstalpalót lehet találni, pl. [4]. Nekünk most pusztán annyi kell, hogy elindítsuk, és a hibánál nézzük meg az aktuális verem állapotot.

```
gdb wsman
```

Majd a gdb konzolján el kell indítani a programot a megfelelő argumentumokkal:

```
run enumerate 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor'
-h localhost -P 5985 -u pegasus -p LaborImage
```

Az eredmény:

```
Program received signal SIGSEGV, Segmentation fault.
0x004f6333 in strlen () from /lib/libc.so.6
```

Érdemes megnézni a stack trace-t:

[11] http://en.wikipedia.org/wiki/SOAP

```
(gdb) backtrace
#0  0x004f6333 in strlen () from /lib/libc.so.6
#1  0x0804ba4d in main ()
```

Sajnos mivel nincsenek debug szimbólumok a programhoz, ezért például sorszámokat és lokális változókat nem látunk. Erre figyelmeztet is a gdb az elején:

```
Reading symbols from /usr/bin/wsman...(no debugging symbols found)...done.
```

Annyit mindenesetre látunk, hogy az `strlen()` függvényt hívjuk meg nem megfelelő argumentumokkal.

Hogy könnyen tudjunk továbbhaladni, érdemes készíteni egy olyan verziót az wsman programból, amiben vannak debug szimbólumok is. Mivel elérhető a forrása, ezért ez, ha nem is triviálisan, de megoldható. A pontos részleteket itt most átugorjuk (a wsmancli egy korábbi verziójához lásd itt [5]), a lényeg annyi, hogy a *configure* szkript futtatása előtt a C fordítónak meg kell adni, hogy generáljon debug szimbólumokat is (*export CFLAGS=-g*). Az így kapott verzióval már a következőket látjuk a debuggerben:

```
#1  0x0804ba4d in main (argc=Cannot access memory at address 0x0
) at wsman.c:447
447         int count = strlen(output_file) + 16;
```

A gond tehát a 447. sorban van, ahol az output_file mutatóra hívjuk meg az strlen függvényt. Mivel nem adtunk meg olyan bemeneti kapcsolót, ami kimeneti fájlt írna elő, ezért az output_file értéke NULL, így nyilván segmentation fault lesz az eredmény. Ha megnézzük a wsman.c-t, akkor a WSMAN_ACTION_ENUMERATION ág lekezelésekor feltétel nélkül meghívjuk a *wsman_output_pull* függvényt, ami fájlba szeretne írni.

Az enumeration műveletnél tehát adjunk meg kimeneti fájlt:

```
wsman enumerate 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor'
-h localhost -P 5985 -u pegasus -p LaborImage -O ~/wsman.out
```

Ez már hiba nélkül lefut, és létrejön egy wsman.out nevű fájl, amibe az enumeration context lekérése kerül be, valamint egy wsman.out-1.xml fájl, amiben a lekért CIM objektumok vannak XML-be ágyazva.

Bár nem volt egyszerű, de azért sikerült adatokat lekérnünk a CIMOM-tól WS-Management protokollon keresztül.

A tanulság az, hogy az openwsman ugyan eléggé aluldokumentált és vannak benne hibák, de ha kicsit gondolkozunk és használjuk az eddig megszerzett tudásunkat, akkor meg lehet oldani az előkerülő hibákat.