OSTC
OMI AUTHENTICATION

Dev Design Specification

CONTACTS/Doc Info

ROLE	Name; Alias

Developer	Bruce Campbell (v-brucc);
	Nicolas Roy (niroy);
	Paul Allen (paulall);

Quality	Steve W or China Test Team
Program Manager	Michael Kelley (mikelley)
Manager	Li Li (lil)

Overview

OMI currently only supports Basic authentication which requires sending username and password. PowerShell remoting is limited to the omi authentication methods because it uses omi to communicate with the PowerShell provider on Linux. Customers have already raised concerns about the security of Basic authentication and would like alternative methods. In the Windows world for example Active Directory/Kerberos allows a frictionless single sign on experience with greater security. In the Linux world, computers with properly configured ssh keys can communicate securely without entering credentials.

Scope

This document serves as a developer reference to the implementation of the new authentication mechanisms to OMI. We will focus on using NT LAN Manager (NTLM) authentication through the Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO).

Goal

The intent is to enable new authentication schemes defined in the Powershell remoting document.

- Extend Configuration 3A (Linux to Win PSRP over WSMAN) to work with SPNEGO using NTLM. This is SPNEGO client side work on Linux.
- Once we have customer validation, extend Configuration 2A (Win to Linux PSRP over WSMAN) to work with SPNEGO using NTLM. This is SPNEGO server side work on Linux.

Secondary goals

- Implement Kerberos authentication
- Automatic ID mapping Windows/Linux
- Domain join Linux machine
- Use Windows Active Directory security policies
- Script to automate/ease domain joining a Linux machine
- Allowing extensibility for using Kerberos authentication in both directions

NTLM overview

NTLM authentication is a challenge-response scheme, consisting of three messages, commonly referred to as Type 1 (negotiation), Type 2 (challenge) and Type 3 (authentication). Since we are using the GSS-API with the gss-ntlmssp plugin, the details of NTLM are hidden and encapsulated in the opaque token.

1. The client sends a Type 1 message to the server. It contains a list of features supported by the client and requested of the server.

2. The server responds with a Type 2 message. This contains a list of features supported and agreed upon by the server. It contains a challenge generated by the server.

3. The client replies with a Type 3 message. This contains several pieces of information about the client, including the domain and username of the client user. It also contains one or more responses to the Type 2 challenge. The server challenge is encrypted with the hash of the password.

Since the type 3 message requires the server to know the hash of the password, a credential cache is required on the server side. (See “Set up cached credentials”)

Windows to Linux Authentication

We will use the Windows Remote Management (winrm) program on windows to test the NTLM authentication.

Installation

Ubuntu 16

Save this script to a file and run it

```
#!/bin/bash

# This script automates setting up an ubuntu 16 machine for testing ntlm

if [ "$EUID" = "0" ]; then
   echo "This script should NOT be run as root" 1>&2
   exit 1
fi
```

```
set -e
set -x

# Create credential cache file
HOSTNAME=`hostname`
USERNAME=`whoami`
PASSWORD=OpsMgr2007R2
[ ! -f ccache ] && echo "$HOSTNAME:$USERNAME:$PASSWORD" > ccache

echo "# Install Powershell"
if ! dpkg -s powershell > /dev/null 2>&1; then
    wget
https://raw.githubusercontent.com/PowerShell/PowerShell/master/tools/download.sh
    bash download.sh
fi

# Add exception to ssh config to make clone non interactive by not prompting for host
authenticity
if ! grep github ~/.ssh/config > /dev/null; then
    [ ! -d ~/.ssh ] && mkdir -p ~/.ssh && chmod 700 ~/.ssh
    touch ~/.ssh/config
    grep github ~/.ssh/config || echo -e "Host github.com\n\tStrictHostKeyChecking
no\n" >> ~/.ssh/config
fi


echo "# Install and configure GSS NTLM SSP"
if ! dpkg -s gss-ntlmssp > /dev/null 2>&1; then
    echo "Get up to date"
    sudo apt-get update
    echo "Install gss-ntlm ssp"
    sudo apt install gss-ntlmssp -y
    # Rename file so it is actually loaded to enable NTLM
    sudo mv /etc/gss/mech.d/mech.ntlmssp /etc/gss/mech.d/mech.ntlmssp.conf

    # Fix prefix
    sudo sed -i 's/${prefix}/\/usr/g' /etc/gss/mech.d/mech.ntlmssp.conf
fi


echo "# Build OMI and powershell provider"
if [ ! -d psl-omi-provider ]; then
    git clone --recursive https://github.com/PowerShell/psl-omi-provider.git


    echo "Install omi build time deps"
    sudo apt-get install git pkg-config make g++ rpm librpm-dev libpam0g-dev libssl-
dev -y
    # NTLM build time dependency
    sudo apt-get install libkrb5-dev -y
    sudo apt install libnss-winbind
    pushd psl-omi-provider/omi/Unix

    # Switch to NTLM branch
    git checkout v-brucc-client-auth

    ./configure --dev
    make -j
    popd

    sudo apt-get install cmake -y
    pushd psl-omi-provider/src
    cmake -DCMAKE_BUILD_TYPE=Debug .
    make -j
    popd
fi

# Run omiserver
sudo NTLM_USER_FILE=`pwd`/ccache psl-omi-provider/omi/Unix/output/bin/omiserver --
loglevel 4 --httptrace &

echo "On windows run:"
echo "winrm id -r:https://$HOSTNAME:5986 -auth:negotiate -u:$HOSTNAME\$USERNAME -
p:OpsMgr2007R2 -skipcncheck -skipcacheck -encoding:utf-8 -skiprevocationcheck"
```

CentOS 7

```
#!/bin/bash
# This script automates setting up an cent7 machine for testing ntlm

if [ "$EUID" = "0" ]; then
   echo "This script should NOT be run as root" 1>&2
   exit 1
fi

set -e
set -x

# Create credential cache file
HOSTNAME=`hostname`
USERNAME=`whoami`
PASSWORD=OpsMgr2007R2
[ ! -f ccache ] && echo "$HOSTNAME:$USERNAME:$PASSWORD" > ccache


if ! rpm -q powershell > /dev/null 2>&1; then
 echo "# Install Powershell"
 wget
https://raw.githubusercontent.com/PowerShell/PowerShell/master/tools/download.sh
```

```
 bash download.sh
fi

# Add exception to ssh config to make clone non interactive by not prompting for host
authenticity
if ! grep github ~/.ssh/config > /dev/null; then
    echo "Adding github exception"
    [ ! -d ~/.ssh ] && mkdir -p ~/.ssh && chmod 700 ~/.ssh
    touch ~/.ssh/config
    grep github ~/.ssh/config || echo -e "Host github.com\n\tStrictHostKeyChecking
no\n" >> ~/.ssh/config
fi

if ! rpm -q gssntlmssp > /dev/null 2>&1; then
    echo "# Install and configure GSS NTLM SSP"
    # echo "Get up to date"
    # sudo yum update
    echo "Install gss-ntlm ssp"
    sudo yum -y install epel-release
    sudo yum install gssntlmssp.x86_64 -y
fi


if [ ! -d psl-omi-provider ]; then
    echo "Install omi deps"
    sudo yum install git bind-utils gcc-c++ rpm-devel pam-devel openssl-devel rpm-
build -y
    git clone --recursive https://github.com/PowerShell/psl-omi-provider.git

    #NTLM dependency
    sudo yum install krb5-libs -y
    pushd psl-omi-provider/omi/Unix

    # Switch to NTLM branch
    git checkout v-brucc-client-auth
    ./configure --dev
    echo "# Build OMI and powershell provider"
    make -j
    popd

    sudo yum install cmake -y
    pushd psl-omi-provider/src
    cmake -DCMAKE_BUILD_TYPE=Debug .
    make -j
    popd
fi


# Need to setup active zone to open http port for network communication
firewall-cmd --get-active-zones
sudo firewall-cmd --zone=public --add-port=5986/tcp --permanent
sudo firewall-cmd --zone=public --add-port=5985/tcp --permanent

# Run omiserver if not already running
if [ ! -f psl-omi-provider/omi/Unix/output/var/run/omiserver.pid ]; then
    sudo NTLM_USER_FILE=`pwd`/ccache psl-omi-provider/omi/Unix/output/bin/omiserver -
-loglevel 4 --httptrace &
fi

set +x
echo "On windows run: for the basic OMI verification"
echo "winrm id -r:https://$HOSTNAME:5986 -auth:negotiate -u:$HOSTNAME\\$USERNAME -
p:OpsMgr2007R2 -skipcncheck -skipcacheck -encoding:utf-8 -skiprevocationcheck"

echo "On windows run: for the PSRP verification"
echo "winrm set winrm/config/Client @{TrustedHosts="*"}"
echo "powershell"
echo "$o=New-PSSessionOption -NoEncryption"
Echo "$cred=Get-Credential"
echo "Enter-PSSession -ComputerName $HOSTNAME -Credential $cred -Authentication
negotiate -SessionOption $o"

echo "Bootstrap ok"
```

Once the NTLM feature gets merged into the OMI master branch, customers will be able to simply install the prebuilt packages.

Dependencies

To get NTLM running on the Linux server, the machine needs an extension to libgss: gss-ntlmssp.

gss-ntlmssp is available on Ubuntu 16 but not Ubuntu 14.

On Ubuntu we also need:

libnss-winbind, libpam-winbind

Extra packages needed on Ubuntu for building omi

In addition to the standard packages described here : https://github.com/Microsoft/Build-SCXcore#dependencies-to-build-a-native-package

The gssapi headers are required to build the authentication feature:
`sudo apt install libkrb5-dev`

Credentials on Linux server

Credentials need to be present on the Linux server to answer the NTLM challenge. GSS_NTLMSSP supports a plain text credential cache file for development or through a previously authenticated Winbind process.

Credential Cache File

Create a file /usr/local/etc/gss/cred/ccache

Enter the credentials, one on each line, in the format:

```
DOMAIN:username:password
```

If your Linux server does not join the domain, use its machine name. On the client side, you need to enter the credential as the format as “machine name/user name” with password accordingly.

OMI must be started with the environment variable NTLM_USER_FILE set to point to the credential cache file. The systemctl should be modified to take this into account.

```
sudo NTLM_USER_FILE=/usr/local/etc/gss/cred/ccache /opt/omi/bin/omiserver
```

Using Winbind

Install SAMABA Server on CentOS

You need to install several packages to support winbind authentication.

```
sudo yum install samba samba-client samba-common samba-winbind-clients -y
```

Check the version of installed samba software by using this command:

```
smbd --version
```

Configure SAMBA

/etc/samba/smb.conf

```
[global]
    workgroup = SAMBA
    domain logons = Yes
    security = USER
    winbind offline logon = Yes
    winbind use default domain = Yes
    idmap config * : range = 1000-1000000
    idmap config * : backend = passdb
    passdb backend = tdbsam
    template homedir = /home/%U
    template shell = /bin/bash

[homes]
    comment = Home Directories
    browseable = No
    inherit acls = Yes
    read only = No
    valid users = %S %D%w%S
```

To make the smbd, nmbd and winbindd daemons reload the configuration run:

```
sudo smbcontrol all reload-config
```

We can create ‘admins’ group using groupadd utility:

```
sudo groupadd administrator
```

When groups are ready, we can associate them with the well-known domain groups using net groupmap commands:

```
$ sudo net groupmap add ntgroup="Domain Admins" unixgroup=administrator rid=512
type=d
Successfully added group Domain Admins to the mapping db as a domain group
$ sudo net groupmap add ntgroup="Domain Users" unixgroup=users rid=513
Successfully added group Domain Users to the mapping db as a domain group
$ net groupmap add ntgroup="Domain Guests"  unixgroup=nobody rid=514
Successfully added group Domain Guests to the mapping db as a domain group
```

Finally, add users. Users should have their primary group associated with any of the groups mapped to the domain because Samba needs to recognize them. So there should be SID to POSIX ID mapping for primary groups. Let's pretend that all our users are members of 'users' group:

```
$ sudo useradd -m -g users -G administrator administrator
$ sudo pdbedit -a -u administrator
new password:retype new password:Unix username:        administratorNT username:
Account Flags:        [U          ]User SID:             S-1-5-21-1345368309-
3761995768-4153620981-1008
Primary Group SID:    S-1-5-21-1345368309-3761995768-4153620981-513
Full Name:            Home Directory:        \\smb\administrator
HomeDir Drive:
Logon Script:         Profile Path:          \\smb\administrator\profileDomain:
SAMBAAccount desc:         Workstations:         Munged dial:          Logon time:
0Logoff time:          Wed, 06 Feb 2036 17:06:39 EETKickoff time:         Wed, 06 Feb
2036 17:06:39 EETPassword last set:    Mon, 19 Sep 2016 12:43:45 EESTPassword can
change:  Mon, 19 Sep 2016 12:43:45 EESTPassword must change: neverLast bad password
: 0Bad password count  : 0Logon hours         :
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

In the screen output above 'Primary Group SID' was automatically inferred from the group mapping.

We can now ask winbindd to resolve user information based on the IDMAP and PASSDB databases:

```
$ wbinfo -i administratoradministrator:*:1002:100::/home/administrator:/bin/bash
$ wbinfo -n administratorS-1-5-21-1345368309-3761995768-4153620981-1008 SID_USER (1)#
wbinfo -s S-1-5-21-1345368309-3761995768-4153620981-1008SAMBA\administrator 1
```

Automated setup for Centos

Configure system to use winbind for authentication. On CentOS systems this can be done with authconfig:

```
      authconfig –-enablewinbind --enablewinbindauth --update
```

From the man page:

```
The […] --enablewinbind options are used to configure user information services
in /etc/nsswitch.conf, the --enablecache option is used to configure naming services
caching, and the […] –enablewinbindauth options are used to configure authentication
functions via /etc/pam.d/system-auth
```

After automated setup is done, open /etc/nsswitch.conf to double check the psswd, group and shadown, you should see winbind in these entries. If you don't see winbind in these entries, please use the manual setup below to make sure you add winbind entries.

Manual setup

On Ubuntu, there is no authconfig utility so we have to do the configuration manually.

Add Winbind in /etc/nsswitch.conf

For Ubuntu,

```
      passwd:         compat winbind
      group:          compat winbind
      shadow:         compat winbind
```

For CentOS,

```
      passwd: files sss winbind
      group: files sss winbind
      shadow: files sss winbind
```

Set up user account (Needed by both Centos and Ubuntu)

Now set password for the user administrator:

```
      sudo smbpasswd -a administrator
```

The above command will modify the below files automatically. You don't need to change anything manually.

    /var/lib/samba/private/passdb.tdb
    /var/lib/samba/gencache.tdb
    /var/lib/samba/gencache_notrans.tdb

and start Samba:

```
      Systemctl start smb.service
      systemctl start nmb.service
      systemctl start winbind.service
```

You can now try to authenticate for ssh:

```
      ssh administrator@`hostname`
```

and that authentication over PAM would be done by winbindd.
If you install gssntlmssp, then you can see that winbindd actually is used:

As administrator make the credentials cached:
Wbinfo is part of the **samba-winbind-clients** package

```
      wbinfo -K administrator
```

This attempts to authenticate via Kerberos but will fail and fallback to samlogon as seen in the Winbind log:

```
      krb5 auth requested but domain is not Active Directory
      falling back to samlogon
```

Check ntlm authentication works locally:

```
      ntlm_auth --username <username> --password <password>
```

Test Command on Windows

Using https:

```
winrm id -r:https://<hostname>:5986 -auth:negotiate -u:SAMBA\administrator -
p:<Password> -skipcncheck -skipcacheck -encoding:utf-8 -skiprevocationcheck
```

Using gss_wrap encryption over http:

```
winrm id -r:http://<hostname>:5985 -auth:negotiate -u:SAMBA\administrator -
p:<password> -encoding:utf-8
```

For any GSS questions, send email to gssapi-interop@redhat.com

Debugging Winbind

By default, logs are placed in /var/log/samba/log.winbindd
The log location may be changed by a config setting in /etc/samba/smb.conf

```
log file = /var/log/samba/log.%m
```

It is also possible to stop the Winbind service and start it manually with the desired log level (1-10) default is 5.

```
winbindd -F -d 5 -S
```

Example of successful auth log:

```
[2016/09/13 23:58:22.567415,  3]
../source3/winbindd/winbindd_misc.c:395(winbindd_interface_version)
  [53846]: request interface version (version = 27)
[2016/09/13 23:58:22.567481,  3]
../source3/winbindd/winbindd_misc.c:383(winbindd_info)
  [53846]: request misc info
[2016/09/13 23:58:22.567514,  3]
../source3/winbindd/winbindd_misc.c:416(winbindd_netbios_name)
  [53846]: request netbios name
[2016/09/13 23:58:22.567607,  3]
../source3/winbindd/winbindd_misc.c:405(winbindd_domain_name)
  [53846]: request domain name
[2016/09/13 23:58:22.567643,  3]
../source3/winbindd/winbindd_misc.c:237(winbindd_domain_info)
  [53846]: domain_info [SAMBA]
[2016/09/13 23:58:22.568947,  3]
../source3/winbindd/winbindd_misc.c:383(winbindd_info)
  [53846]: request misc info
[2016/09/13 23:58:22.568992,  3]
../source3/winbindd/winbindd_ccache_access.c:196(winbindd_ccache_ntlm_auth)
  [53846]: perform NTLM auth on behalf of user SAMBA\toto2
```

```
[2016/09/13 23:58:22.569082,  3]
../source3/winbindd/winbindd_pam_auth_crap.c:73(winbindd_pam_auth_crap_send)
  [53846]: pam auth crap domain: [SAMBA] user: toto2
[2016/09/13 23:58:22.569800,  3]
../source3/winbindd/winbindd_getpwnam.c:56(winbindd_getpwnam_send)
  getpwnam SAMBA\toto2
```

Test command

In a terminal on a windows machine:

```
winrm e http://schemas.microsoft.com/wbem/wscim/1/cim-
schema/2/SCX_Agent?__cimnamespace=root/scx -r:https://hv-cent7-01:1270 -
auth:negotiate -skipcncheck -skipcacheck -encoding:utf-8 -skiprevocationcheck
```

Headers

Windows/Winrm/server to Linux/OMI/client:

```
POST /wsman HTTP/1.1
Connection: Keep-Alive
Content-Type: application/soap+xml;charset=UTF-8
User-Agent: Microsoft WinRM Client
Content-Length: 0
Host: niroy64-cent7x-01:1270
Authorization: Negotiate
YIGeBgYrBgEFBQKggZMwgZCgGjAYBgorBgEEAYI3AgIeBgorBgEEAYI3AgIKonIEcE5FR09FWFRTAAAAAAAAA
ABgAAAAcAAAABut3iQBL4cnNwvT7Exad2MEN1pkAyXI5o3AKZR+qvVUG2MNQoU1SpuaimASB1uG1AAAAAAAAA
AAYAAAAAEAAAAAAAAAAAAAAMNbiTClcBVAolAmCpGQrZA=
```

This differs from rfc4559 in two ways:

- A post request is used instead of a get request
- The first client server roundtrip is skipped because the windows client already knows Negotiate should be used. This has to be taken into consideration in the client.

Linux/OMI to Windows/Winrm reply:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Negotiate TlRMTVNTUA[…]
Content-Length: 0
```

GSS-API usage

The omiserver will use the gss-api to be compatible with multiple authentications schemes and underlying implementations.

Server side authentication loop according to rfc2744:

```
   gss_ctx_id_t context_hdl = GSS_C_NO_CONTEXT;
   do {
     receive_token_from_peer(input_token);
     maj_stat = gss_accept_sec_context(&min_stat,
                                       &context_hdl,
                                       cred_hdl,
                                       input_token,
                                       input_bindings,
                                       &client_name,
                                       &mech_type,
                                       output_token,
                                       &ret_flags,
                                       &time_rec,
                                       &deleg_cred);
     if (GSS_ERROR(maj_stat)) {
       report_error(maj_stat, min_stat);
     };
     if (output_token->length != 0) {
       send_token_to_peer(output_token);

       gss_release_buffer(&min_stat, output_token);
     };
     if (GSS_ERROR(maj_stat)) {
       if (context_hdl != GSS_C_NO_CONTEXT)
         gss_delete_sec_context(&min_stat,
                                &context_hdl,
                                GSS_C_NO_BUFFER);
       break;
     };
   } while (maj_stat & GSS_S_CONTINUE_NEEDED);
```

Linux to Windows Authentication

In this scenario work on the omiclient is required to add the NTLM authentication capability.

Dependencies

GSS-NTLMSSP (Run time)

GSS-NTLMSSP is a GSSAPI mechanism plugin that implements NTLMSSP. NTLMSSP is a Microsoft Security Provider that implements various versions and flavors of the NTLM challenge-response family.

GSS-NTLMSSP, implements both NTLM and NTLMv2 and all the various security variants to the key exchange that Microsoft introduced and documented over time.
https://fedorahosted.org/gss-ntlmssp/

There does not seems to be any standalone NTLM library for Linux. The recommended way to use NTLM is through GSSAPI.

Test Command

Allow unencrypted on Windows until we have encryption working.

```
winrm set winrm/config/Service @{AllowUnencrypted="true"}
```

Allow Basic auth for base test

```
winrm set winrm/config/Service/Auth @{Basic="true"}
```

On Linux

```
output/bin/omicli -u zoot -p LasVegas2020! --auth Basic --hostname v-brucc-win10 gi
root/cimv2 { Win32_Process Handle 0 }
```

```
sudo /opt/omi/bin/omicli -t -u test -p a1b2c3d4@ --auth Negotiate --hostname
10.123.175.205 gi root/cimv2 { Win32_Process Handle 0 }

./omicli -t -u zoot -p LasVegas2020! --auth NegoWithCreds --hostname v-brucc-win10 gi
root/cimv2 { Win32_Process Handle 0 }

./omicli -t -u v-brucc@KERBOSOFT -p <PASSWORD REDACTED> --auth NegoWithCreds --
hostname kerb-dc-01.kerbosoft.com -t gi root/cimv2 { Win32_Process Handle 0 }
```

Windows Machine Configuration

The winrm service must be running and a firewall exception must be present.

```
Winrm quickconfig
```

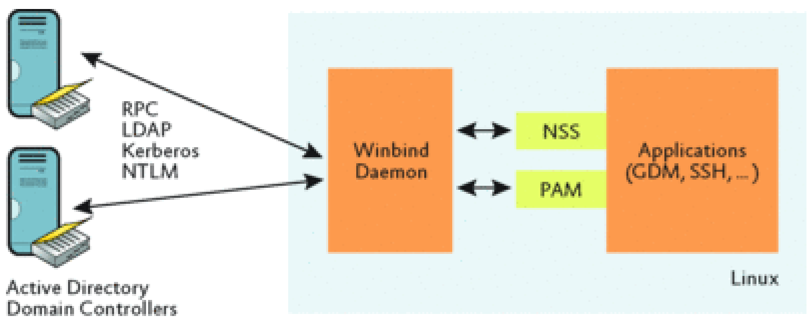
https://msdn.microsoft.com/en-us/library/aa384372(v=vs.85).aspx

Headers

The headers are similar to Windows to Linux except that the roles are reversed.

Diagrams

Architecture diagram


RPC
LDAP
Kerberos
NTLM
Winbind Daemon
NSS
PAM
Applications (GDM, SSH, ...)
Linux
Active Directory Domain Controllers

Flow diagram

Component diagram

Calling Sequence

- Windows Winrm
    - Linux OMI
        - GSSAPI authentication
            - NTLM

Packaging

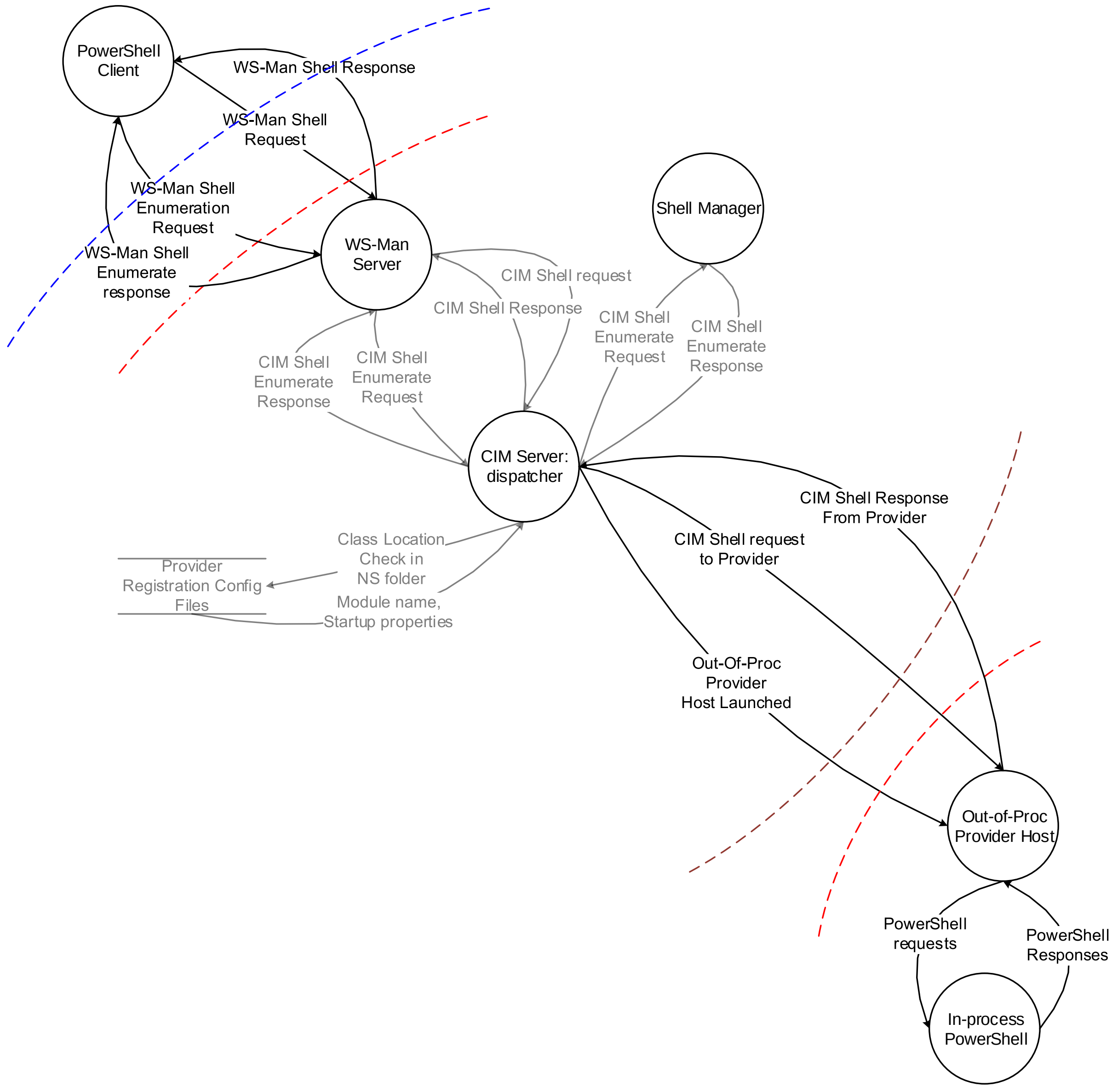
We would like to distribute OMI as it is packaged right now: Usually an rpm inside a shell bundle of Operations manager.

OMI should still function with Basic Auth if there are missing dependencies for the other authentication mechanisms.

Proxy authentication

Proxy authentication may be a required feature in the future. We will not cover it in this spec.

Threat Modeling


PowerShell Client
WS-Man Shell Response
WS-Man Shell Request
WS-Man Shell Enumeration Request
WS-Man Shell Enumerate response
WS-Man Server
Shell Manager
CIM Shell request
CIM Shell Response
CIM Shell Enumerate Request
CIM Shell Enumerate Response
CIM Shell Enumerate Response
CIM Shell Enumerate Request
CIM Server: dispatcher
CIM Shell Response From Provider
CIM Shell request to Provider
Class Location Check in NS folder
Provider Registration Config Files
Module name, Startup properties
Out-Of-Proc Provider Host Launched
Out-of-Proc Provider Host
PowerShell requests
PowerShell Responses
In-process PowerShell

The PowerShell client (Windows) connects to the server (on Linux using OMI as a network end-point) to create a shell and does so as a specific user per the basic or negotiate authentication header. An out-of-process provider host is then created and PowerShell is loaded in-process to that host.

Assessment Scope

If an attacker gains access to a PowerShell remoting session, he will be able to run arbitrary commands as the user.

Identify Threat Agents and possible Attacks

- Someone who is trying to gain access to a system running PowerShell.

    - If a windows machine is compromised, the attacker can pivot on other systems using the cached credentials.
- Someone trying to disrupt machines running PowerShell.
    - An attacker can try to flood the provider with simultaneous requests

STRIDE

Spoofing of user identity

Through tools like mimikatz, it is possible to extract NTLM hashes on one system and use them to spoof users.
NTLM has a pass the hash vulnerability. https://en.wikipedia.org/wiki/Pass_the_hash

Tampering

The content of the messages are either encrypted using ssl when using the https protocol or with gss_wrap when using http. In both cases tampering is difficult.

Repudiation

Shared user accounts prevent knowing the true identity of the sender.

Information disclosure (privacy breach or data leak)

A breach of authentication would also disclose the information readable on the remote machine.

Denial of service (D.o.S)

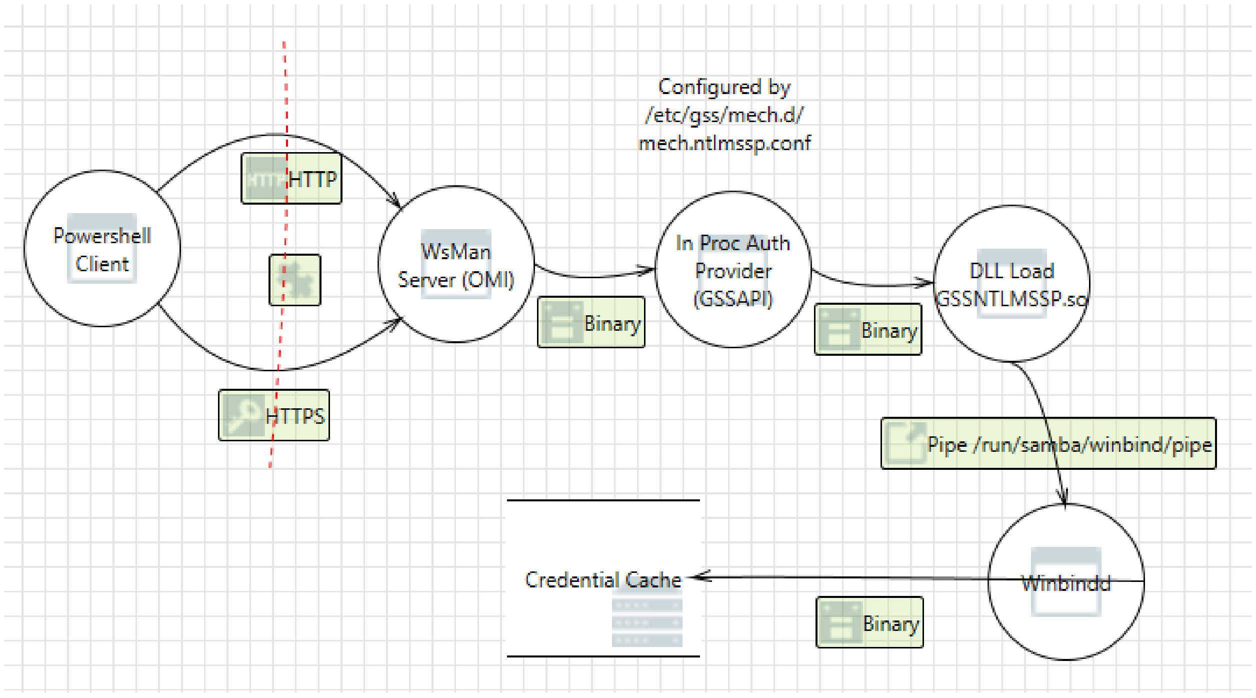
An attacker can create multiple simultaneous authentication to try to exhaust all the resources of the server. It may be possible to prevent a real user from logging in.

Elevation of privilege

If a user can spoof another one through stealing cached credentials on a shared system, he may gain the priviledge of the spoofed user.

It may also be possible to brute force passwords.

Detailed Authentication



ID	Description	Type		
0	Elevation Using Impersonation	Elevation Of Privilege	Pipe /run/samba/winbind/pipe	Winbindd may be able to impersonate the context of DLL Load GSSNTLMSSP.so in order to gain additional privilege.
1	Spoofing the Powershell Client Process	Spoofing	HTTP	Powershell Client may be spoofed by an attacker and this may lead to unauthorized access to WsMan Server (OMI). Consider using a standard authentication mechanism to identify the source process.
2	Spoofing the WsMan Server (OMI) Process	Spoofing	HTTP	WsMan Server (OMI) may be spoofed by an attacker and this may lead to information disclosure by Powershell Client. Consider using a standard authentication mechanism to identify the destination process.
3	Potential Lack of Input Validation for WsMan Server (OMI)	Tampering	HTTP	Data flowing across HTTP may be tampered with by an attacker. This may lead to a denial of service attack against WsMan Server (OMI) or an elevation of privilege attack against WsMan Server (OMI) or an information disclosure by WsMan Server (OMI). Failure to verify that input is as expected is a root cause of a very large number of exploitable issues. Consider all paths and the way they handle data. Verify that all input is verified for correctness using an approved list input validation approach.
4	Potential Data Repudiation by WsMan Server (OMI)	Repudiation	HTTP	WsMan Server (OMI) claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.
5	Data Flow Sniffing	Information Disclosure	HTTP	Data flowing across HTTP may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a

				disclosure of information leading to compliance violations. Consider encrypting the data flow.
6	Potential Process Crash or Stop for WsMan Server (OMI)	Denial Of Service	HTTP	WsMan Server (OMI) crashes, halts, stops or runs slowly; in all cases violating an availability metric.
7	Data Flow HTTP Is Potentially Interrupted	Denial Of Service	HTTP	An external agent interrupts data flowing across a trust boundary in either direction.
8	Elevation Using Impersonation	Elevation Of Privilege	HTTP	WsMan Server (OMI) may be able to impersonate the context of Powershell Client in order to gain additional privilege.
9	WsMan Server (OMI) May be Subject to Elevation of Privilege Using Remote Code Execution	Elevation Of Privilege	HTTP	Powershell Client may be able to remotely execute code for WsMan Server (OMI).
10	Elevation by Changing the Execution Flow in WsMan Server (OMI)	Elevation Of Privilege	HTTP	An attacker may pass data into WsMan Server (OMI) in order to change the flow of program execution within WsMan Server (OMI) to the attacker's choosing.
11	Spoofing the Powershell Client Process	Spoofing	HTTPS	Powershell Client may be spoofed by an attacker and this may lead to unauthorized access to WsMan Server (OMI). Consider using a standard authentication mechanism to identify the source process.
12	Potential Data Repudiation by WsMan Server (OMI)	Repudiation	HTTPS	WsMan Server (OMI) claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.
13	Potential Process Crash or Stop for WsMan Server (OMI)	Denial Of Service	HTTPS	WsMan Server (OMI) crashes, halts, stops or runs slowly; in all cases violating an availability metric.
14	Data Flow HTTPS Is Potentially Interrupted	Denial Of Service	HTTPS	An external agent interrupts data flowing across a trust boundary in either direction.
15	Elevation Using Impersonation	Elevation Of Privilege	HTTPS	WsMan Server (OMI) may be able to impersonate the context of Powershell Client in order to gain additional privilege.
16	WsMan Server (OMI) May be Subject to Elevation of Privilege Using Remote Code Execution	Elevation Of Privilege	HTTPS	Powershell Client may be able to remotely execute code for WsMan Server (OMI).
17	Elevation by Changing the Execution Flow in WsMan Server (OMI)	Elevation Of Privilege	HTTPS	An attacker may pass data into WsMan Server (OMI) in order to change the flow of program execution within WsMan Server (OMI) to the attacker's choosing.
18	Elevation Using Impersonation	Elevation Of Privilege	Binary	In Proc Auth Provider (GSSAPI) may be able to impersonate the context of WsMan Server (OMI) in order to gain additional privilege.
19	Elevation Using Impersonation	Elevation Of Privilege	Binary	DLL Load GSSNTLMSSP.so may be able to impersonate the context of In Proc Auth Provider (GSSAPI) in order to gain additional privilege.
20	Spoofing of Destination Data Store Credential Cache	Spoofing	Binary	Credential Cache may be spoofed by an attacker and this may lead to data being written to the attacker's target instead of Credential Cache. Consider using a standard authentication mechanism to identify the destination data store.

21	Potential Excessive Resource Consumption for Winbindd or Credential Cache	Denial Of Service	Binary	Does Winbindd or Credential Cache take explicit steps to control resource consumption? Resource consumption attacks can be hard to deal with, and there are times that it makes sense to let the OS do the job. Be careful that your resource requests don't deadlock, and that they do timeout.

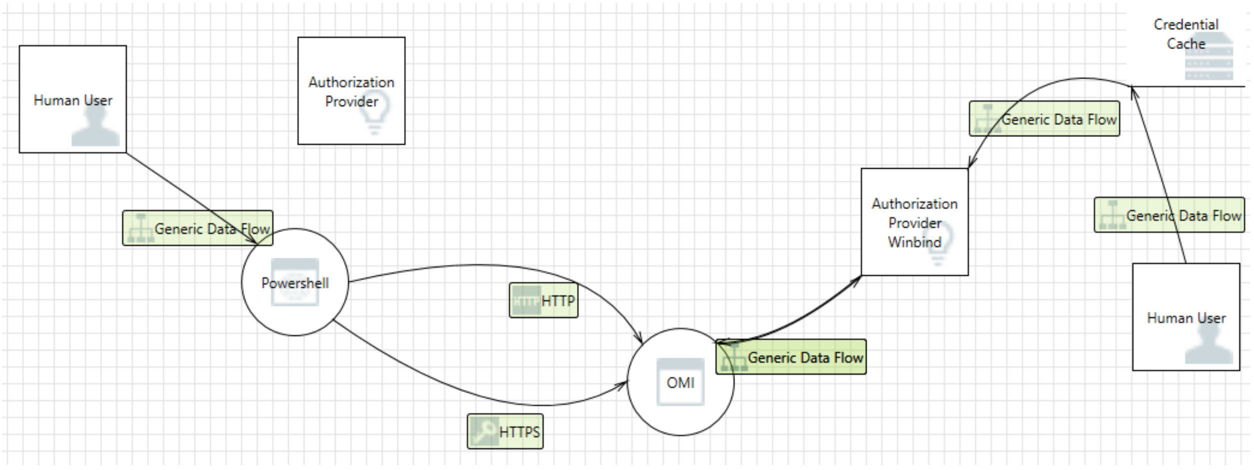
Understand existing **Countermeasures**

- Limit the amount of possible simultaneous unauthorized request.
- Timeout request.

Testing

Penetration testing

Since we are adding new authentication mechanisms to OMI, penetration testing should be done to ensure that a machine running OMI cannot be managed by a malicious party.



Unit Tests

The existing OMI nits framework will be used to write Unit tests.

System Tests

Kerberos authentication system tests will require a machine running Windows Server with active directory configured properly.

NTLM System Tests

Tests should be performed both ways

- Authentication success
- Authentication failure
- Failure during the authentication handshake
- Fuzzing of the gssapi token
- Enable auth in tests

End to End Testing

Using Powershell remoting protocol with NTLM authentication should work from:

Windows to Linux with Basic Auth

In an admin command prompt:

```
winrm set winrm/config/Client @{TrustedHosts="*"}
```

Test winrm before testing PSRP to detect problems in the transport.

```
winrm id -r:https://<hostname>:5986 -auth:basic -u:<username> -p:<password> -
skipcncheck -skipcacheck -encoding:utf-8 -skiprevocationcheck
```

Test that a session can be initiated with no encryption:

```
$cred= Get-Credential
$so = New-PSSessionOption -NoEncryption
New-PSSession -ComputerName niroy-xenial4 -Credential $cred -Authentication negotiate
-SessionOption $so

 Id Name            ComputerName    ComputerType    State         ConfigurationName
Availability
 -- ----            ------------    ------------    -----         -----------------
------------
 57 Session57       niroy-xenial4   RemoteMachine   Opened
Microsoft.PowerShell     Available
```

Test that we can enter the created session:

```
PS C:\Users\niroy> Enter-PSSession 57
[niroy-xenial4]: PS /home/administrator> ls
```

Test that we can exit the session:

```
[niroy-xenial4]: PS /home/administrator> Exit-PSSession
```

Test that we can create a session with Encryption:

```
New-PSSession -ComputerName niroy-xenial4 -Credential $cred -Authentication negotiate
```

Test that we can use Enter-PSSession:

```
$cred = Get-Credential "linux-machine-hostname\linux-username"
$o = New-PSSessionOption -SkipCACheck -SkipRevocationCheck -SkipCNCheck
Enter-PSSession -ComputerName <IP address of Linux machine> -Credential $cred -
Authentication basic -UseSSL -SessionOption $o
```

The hostname in the get credential step is used as a workaround since

Windows to Linux with Negotiate auth

```
$so = New-PSSessionOption -NoEncryption
$cred = Get-Credential <linux machine hostname>\<linux username>
Enter-PSSession -SessionOption $so -ComputerName <linux machine hostname> -Credential
$cred -Authentication negotiate -SessionOption $so
```

Linux to Windows

Linux to Linux

Open Issue

Reference

In depth NTLM reference
From http://davenport.sourceforge.net/ntlm.html#whatIsNtlm

Negotiate Protocol
From https://msdn.microsoft.com/en-us/library/cc236767.aspx

Generic Security Service API Version 2: C-bindings
From <https://tools.ietf.org/html/rfc2744>

The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism
From <https://tools.ietf.org/html/rfc4178>

HTTP-Based Cross-Platform Authentication by Using the Negotiate Protocol
From <https://msdn.microsoft.com/en-us/library/ms995330.aspx>

Azure API security guidance

https://github.com/mspnp/azure-guidance/blob/master/API-security.md

Winbind

winbind is a component of the Samba suite of programs that solves the unified logon problem. Winbind uses a UNIX implementation of Microsoft RPC calls, Pluggable Authentication Modules (PAMs), and the name service switch (NSS) to allow Windows NT domain users to appear and operate as UNIX users on a UNIX machine.
https://www.samba.org/samba/docs/man/Samba-HOWTO-Collection/winbind.html