# Configuring MAB with LDAP User Device Binding

Last updated: April 2013

## Contents

Abstract	3
Use of Cisco Secure ACS and Identity Services Engine (ISE)	3
Part I—Configuring MAB with LDAP	3
Introduction	3
Directory Organization	5
Device Class Considerations	6
Directory Definition in Cisco Secure ACS	7
MAB Access Service Definition Test in Cisco Secure ACS View	14 16
Conclusion	17
Part II—User Device Binding	17
Introduction	17
User Device Binding with Directories	
Cross-Referencing Objects	19
User Device Binding Implementation	21
ACS Device Restriction Policy	
Conclusion	24
Appendix	
LDIF Script	
Sample Input File Sample Output File	
References and Links	39

#### Abstract

The **first part** of this document describes the combination of MAB (MAC Authentication Bypass) with LDAP/Microsoft Active Directory. It discusses different options of MAC address storage and integration into RADIUS, specifically into Cisco Secure ACS 5. It provides detailed configuration examples and tools including a Perl script in the appendix to help provision and maintain MAC address objects in LDAP/Active Directory.

The **second part** of this document builds on the MAB/LDAP integration and proposes an alternative approach to allow administrators to tie specific users to specific devices sometimes known as *device-user-tie-in* or *User Device Binding* (UDB).

## Use of Cisco Secure ACS and Identity Services Engine (ISE)

While this document has been written with ACS 5 as the primary RADIUS server, it has to be noted that the same results can be achieved using ISE. When originally testing was done, ISE lacked a critical capability to make this work (dynamic attribute comparison as laid out later in the document). However, with ISE 1.1.1, this capability has been added.

## Part I-Configuring MAB with LDAP

#### Introduction

With identity-enabled networks, devices that don't have a supplicant are often authenticated via their MAC address using a concept called *MAC Authentication Bypass* or MAB for short. Typical examples are printers, cameras or any legacy devices without any IEEE 802.1X supplicant.



A common setup is depicted above. It consists of many devices accessing the network (with or without IEEE 802.1X capabilities) and network infrastructure (switches=authenticators) which acts as 'gatekeepers' enforcing the security policy provided by a central RADIUS server infrastructure. The RADIUS server either has a built-in identity database for users and devices or it interfaces to an external identity database. Often LDAP is used to query external identity databases, especially for MAB.

While MAC addresses can be spoofed and there is also no strong authentication, MAB offers at least *some* level of authentication compared to an *open door* policy.

Authenticators (e.g. routers, switches, etc.) configured for MAB typically take the learned MAC address as the username and as the password, and send an authentication request to the RADIUS server. This is turned into a PAP<sup>1</sup> request, but could also be in the form of an EAP-MD5 request, where the latter does not provide any additional security, as the username is still the MAC address and typically the password is identical to the MAC address<sup>2</sup> (username=password=MAC address). Cisco devices set RADIUS attribute 6 (Service-Type) to 10 (Call Check) to indicate MAB to the RADIUS server (see packet capture below).

```
🗉 Radius Protocol
   Code: Access-Request (1)
   Packet identifier: 0 \times 31 (49)
   Lenath: 138
   Authenticator: DFB4AE1E099D63BDBB6ED7B70434B98A
 Attribute Value Pairs
   ■ AVP: l=14 t=User-Name(1): 0018f809cfd7
   ■ AVP: 1=18 t=User-Password(2): Encrypted
   ■ AVP: l=6 t=Service-Type(6): Call-Check(10)
   ■ AVP: 1=6 t=Framed-MTU(12): 1500
   ■ AVP: 1=19 t=Called-Station-Id(30): 00-18-B9-FE-39-05
   ■ AVP: 1=19 t=Calling-Station-Id(31): 00-18-F8-09-CF-D7
   ■ AVP: 1=18 t=Message-Authenticator(80): A7115474C60F8C19
   ■ AVP: 1=6 t=NAS-Port-Type(61): Ethernet(15)
   ■ AVP: 1=6 t=NAS-Port(5): 50105
   ■ AVP: 1=6 t=NAS-IP-Address(4): 10.100.10.152
```

The RADIUS server then could have a local 'host database', where those MAC addresses are stored.

As an alternative to storing MAC addresses within the RADIUS server, external data stores can be used. When putting MAC addresses in an external data store, a common recommendation on creating MAC objects is to leverage user objects where the username is the MAC address and the password is also set to the MAC address. With Active Directory, several limitations or weaknesses exist:

- Because the username equals the password, the 'password complexity rule' has to be turned off. In earlier Windows AD versions<sup>3</sup>, this could only be performed on a domain wide basis (everyone or no one). With Windows Server 2008 and newer, this can be performed on a user group.
- 2) Without any further restrictions (for example, explicitly disable interactive logon for those MAC address objects), every MAC address has also a user object with a known password, and this can potentially be used to interactively login to any workstation, just by knowing an arbitrary MAC address, which exists within the directory.

A different and more efficient way to store MAC addresses uses external directories accessed via LDAP. Directories are widely deployed and available in enterprise networks.

In the first part of this document, the procedure to create a MAC address hierarchy in a popular LDAP implementation (Microsoft Active Directory with Lightweight Directory Services/LDS) is described. It also explains the configuration steps required on Cisco Secure ACS 5 to integrate the LDAP identity store for use with MAC addresses. Please refer to the TechNet article given in the appendix for information regarding how to install and configure LDS with Microsoft Windows Server. It should be noted however, that basically every LDAP server, like FreeRADIUS, could be used to achieve this.

<sup>&</sup>lt;sup>1</sup> PAP=Password Authentication Protocol. Username and Password are sent as clear text

<sup>&</sup>lt;sup>2</sup> With release 15.0(2)SE, a new feature has been introduced to make the password for MAC Authentication Bypass configurable

<sup>&</sup>lt;sup>3</sup> Microsoft Active Directory 2003 and earlier

## **Directory Organization**

An LDAP-directory will use a (top-level) container where all MAC addresses and groups can be stored. Even though all objects can go into that very same container, it makes sense to include additional hierarchy layers and grouping objects into functional containers and groups.

For example, all MAC address objects for a phone can go into an OU=phones container. Additional group objects can reference all individual MAC objects of a particular device class. An active directory example for a group object is the group 'Domain Users' which references all users of the domain. Grouping objects together allows creating policy rules which reference e.g. 'all phones' instead of individual objects.

The layout/organization within this container is up for discussion; there is definitely no 'best' or single way to achieve the goal of MAC address storage within a directory. Here's a suggestion for how to layout the containers:



device

Essentially, every MAC address lives under the OU<sup>4</sup>=MAC with sub-containers for different device groups (printers, cameras etc.). A group container 'MACGroups' holds the objects for groups. In the above example, every printer device lives in the container 'OU=MAC, OU=MACAddress, OU=printers'. It is also represented in the group object 'CN<sup>5</sup>=printers, OU=MACGroups, OU=MAC' as a reference, the group's member-attribute points back to the device. This way, one can also use group associations and build policy rules within CiscoSecure ACS 5 like isMemberOf('printers').

CN=myprinter004,OU=printers,OU=MACA

The difference between cameras in MACAddresses and cameras in MACGroups in the example given above is as follows:

 OU=cameras is a container of class 'organizational unit', holding all the MAC device objects of a certain type (here: cameras). This differentiation is technically not necessary, it just allows for more clarity/tidiness.

<sup>&</sup>lt;sup>4</sup> OU Organizational Unit, a container

<sup>&</sup>lt;sup>5</sup> CN Common Name, object name, part of DN

 CN=cameras is a group object of class 'group'. All individual children (cameras in this case) are referenced by the groups 'member' attribute.

**Note:** References between objects can be built in different ways. Either by pointing the child to the parent, or by populating a list-attribute of the parent with its children (member attribute). In AD, the parent maintains a list of its children, so Cisco Secure ACS has to be configured accordingly.

**Note:** The directory names in the examples reflect the lab setup, so the names have to be adapted to your actual environment. Most notably, the end of the DN<sup>6</sup>'DC=wdf, DC=ibns, DC=lab' must be adapted to the local environment.

#### **Device Class Considerations**

In Microsoft Active Directory 2008 and newer, the class *device* inherits from class *ieee802Device*. This effectively means that each object of class device automatically has a *macAddress* attribute to store a MAC address.

What if the directory in use does not have the **macAddress** attribute? In this case, the CN of the device object must reflect the MAC address. Here are two examples that illustrate the difference:

#### Example without macAddress attribute:

```
dn: CN=00112233445C,OU=cameras,OU=addresses,OU=MAC,DC=wdf,DC=ibns,DC=lab
objectClass: top
objectClass: device
name: 00112233445C
showInAdvancedViewOnly: FALSE
description: in room 22/b (mycamera003)
```

Example with macAddress attribute:

```
dn: CN=mycamera003,OU=cameras,OU=addresses,OU=MAC,DC=wdf,DC=ibns,DC=lab
objectClass: top
objectClass: device
macAddress: 00112233445C
name: mycamera003
showInAdvancedViewOnly: FALSE
description: in room 22/b (00-11-22-33-44-5C)
```

In the first example, no **macAddress** attribute is available. The object's distinguished name ('dn') contains the MAC address in the 'cn' (common name) field and also in the 'name' attribute. In the second example, the 'cn'/'name' could contain a more meaningful device name (which still has to be unique in this context) and the actual MAC address is contained in the **macAddress** attribute, which is obviously more appropriate. However, from a functional point of view, both variants work perfectly fine.

**Note:** The above example puts the MAC address into the description for the object with the **macAddress** attribute. It puts the device name into the description for the object without the **macAddress** attribute. This way, those attributes will be visible in the 'Description' column (See the screen shot below).

<sup>&</sup>lt;sup>6</sup> DN Distinguished name, unique identifier

In the appendix of this document, a Perl script is included that creates a LDIF<sup>7</sup> file from an Excel table of MAC addresses via a CSV file.

Name	Type	Description A	
🖻 linux01	device	my Linux PXE 01 host (00-0C-29-85-35-3B)	
🖻 vmwarehost	device	my vmware MAB host (00-0C-29-9B-19-48)	
🖬 xpsp3	device	my XP SP3 MAB host entry (00-0C-29-7A-B5-24)	

## **Directory Definition in Cisco Secure ACS**

To allow Cisco Secure ACS 5 to access the LDAP database, a new LDAP identity store configuration must be set up. Steps 5 and 6 ('Directory Organization') are the most important ones, as they have to reflect the actual directory organization, container names and search bases as previously discussed.

1) Go to 'Users and Identity Stores', 'External Identity Stores', 'LDAP':



<sup>&</sup>lt;sup>7</sup> LDAP Data Interchange Format

2) Create a new LDAP Identity store and give it a meaningful name, click 'Next'

General S	erver Connection	Directory Organization
tep 1 - Ge	eneral	
Name:	AD-LDS	
	(	

3) In the next step (Step 2—Server Connection), define the LDAP server IP address/hostname and the port we will connect to. In addition, if authentication is required (usually it is), click on 'Authenticated Access'. The 'Admin DN' is the distinguished name of a user that has the appropriate rights to read from the directory. The easiest way to get the DN is by using the ADSI Edit snap-in<sup>8</sup> and select the administrator user from here:



<sup>&</sup>lt;sup>8</sup> For the Microsoft Management Console, mmc.exe

CN	=Administrator Properties ? x			
Attribute Editor Secur	ty	Clas		
Attributes:	· · · · · · · · · · · · · · · · · · ·	user		
Attribute	Value	grou		
accountExpires	(never)	grou		
adminCount	1	giu		
badPasswordTime	6/18/2012 10:23:09 AM W. Europe Davlight =	grou		
badPwdCount	0	grou		
cn	Administrator	grou		
codePage	0	grou		
countryCode	0	grou		
description	Built-in account for administering the compute	grou		
distinguishedName	CN=Administrator,CN=Users,DC=wdf,DC=ibn	grou		
String Attribute Editor				
Attribute: distinguishedName				
Value:				
CN=Administrator,CN=Users,DC=wdf,DC=ibns,DC=lab				
Clear OK Cancel				
ОК	Cancel Apply Help	arou		
		grou		

Then go to properties and copy the distinguishedName:

4) If everything has been configured, click 'Test Bind To Server' to see if there's no typo or misconfiguration. Cisco Secure ACS should respond with 'Connection test bind succeeded.'

General Serve	r Connection Directory Organizatio	on.	
Step 2 - Serve	r Connection		
Server Connection			
Enable Seco	ndary Server 🛛 🔿 Always Access Pr	imary Server First	
	🔘 Failback To Prima	ry Server After: Minutes	
Primary Server		Secondary Server	
😆 Hostname:	wdfdc01.wdf.ibns.lab	Hostname:	
😅 Port:	389	Port:	
O Anonymous	Access	<ul> <li>Anonymous Access</li> <li>Authenticated Access</li> </ul>	
Authenticated	I Access		
🔅 Admin DN:	CN=Administrator,CN=Users,DC	Admin DN:	
🜞 Password:		Password:	
Use Secure	Authentication	Use Secure Authentication	
Root CA:	IBNS-CA -	Root CA: IBNS-CA -	
Server Timeout:	10 Seconds	Server Timeout: 0 Seconds	
Max. Admin Connections:	20	Max. Admin Connections:	
C	Test Bind To Conver	Test Bind To Server	

This is what it should look like:

Connection to	est bind Succeeded.
	OK

5) Directory Organization/Schema.

Subject Objectclass: which object class has been used to create the device? ('device').

Subject Name Attribute: if the macAddress attribute is available, use 'macAddress'. Otherwise use 'name'.

Certificate Attribute: unused, can be left as is, or empty.

**Group Objectclass:** which object class has been used to create the objects that form object groups? Use 'group'.

**Group Map Attribute:** Within the group object, which attribute holds the references to the actual devices? Use 'member'.

Check 'Group Objects Contain Reference to Subjects' and select 'distinguished name' as the attribute for members in groups.

Subject Objectclass:	device	🔅 Group Objectclass:	group
Subject Name Attribute:	macAddress	Group Map Attribute:	member
Certificate Attribute:	usercertificate		
O Subject Objects 0	Contain Reference To Groups		
Group Objects Co	ontain Reference To Subjects		

6) Directory Organization/Structure

The search bases for the subjects and the groups can actually point to the same point within the directory. If you separate the groups and the subjects into different OUs, you can also point them into those OUs specifically—this will increase search performance.

Directory Structure			
Subject Search Base: OU=MAC,DC=wdf,DC=ibns,DC=lab			
😅 Group Search Base:	p Search Base: OU=MAC,DC=wdf,DC=ibns,DC=lab		
	Test Configuration		
Directory Structure			
🔅 Subject Search Base	: OU=MACAddresses,OU=MAC,DC=wdf,DC=ibns,DC=lab		
Group Search Base: OU=MACGroups,OU=MAC,DC=wdf,DC=ibns,DC=lab			
	Test Configuration		

7) If everything has been configured correct and if there are some MAC address objects already in the directory, the 'Test Configuration' button should give a result similar to this (note that there are objects retrieved for the subjects/MAC addresses and also for the groups):

Result of testing this	configuration is as follows
Primary Server:	
Number of Subjects	5: 9
Number of Groups:	3
Secondary Server:	
Not enabled.	
	ОК

 Check the MAC address format, it must reflect the MAC address format as they have been imported into the directory

MAC Address Format		
Search for MAC Address in Format:	XXXXXXXXXXXXXXX	٥

9) Click 'Finish'

Groups can now be selected within the 'Directory Groups' tab of the configured LDAP server. They are only available if they have been created in the directory structure as discussed above.

Cooret	
Searci	
	Group Name
	CN=cameras,OU=MACGroups,OU=MAC,DC=wdf,DC=ibns,DC=lab
	CN=phones,OU=MACGroups,OU=MAC,DC=wdf,DC=ibns,DC=lab
	CN=printers,OU=MACGroups,OU=MAC,DC=wdf,DC=ibns,DC=lab

These groups can then be used in policy decisions. Also, within the 'Directory Attributes' tab, additional attributes can be made available for policy decisions. Enter an existing MAC address to retrieve it from the directory and select attributes that should be used in the MAB access policy (the example shown below uses the macAddress attribute as part of the device object so the cn can actually be used for a device name, 'myprinter001'):

Sele	ct Attributes		
Seard	h Filter		Go
	Attribute Name	Туре	Example Value
	cn	String	myprinter001
	dSCorePropagationData	String	16010101000000.0Z
	description	String	some description (00-0c-11-22-33-44)
	distinguishedName	String	CN=myprinter001,OU=printers,OU=Addr
	instanceType	Integer 64	4
	macAddress	String	000C11223344
	memberOf	String	CN=printers,OU=Groups,OU=MAC,DC=s
	name	String	myprinter001

This example also shows the back-reference to the devices' group **printers** in the **memberOf** attribute of the device.

Note: MAC addresses are case sensitive; they are stored with upper case A-F letters.

This concludes the definition/setup of the LDAP identity store. We're now moving on to configuring the associated Access Service/rules.

## MAB Access Service Definition

Create a new service of type 'Network Access' and make sure that 'Process Host Lookup' and 'PAP' has been selected. This ensures that for Service-Type=10 the Calling-Station-ID will be used as a search key into the directory.

General Allowed Proto	cols	
Step 1 - General		
General		
XAME: SVC_MAB		
Description: MAC Auther	itication Bypass	
Access Service Policy Stru	cture	
O Based on service templa	ate	Select
O Based on existing service	e .	Select
<ul> <li>User Selected Service T</li> <li>User Selected Service Type</li> <li>Policy Structure</li> <li>Identity</li> <li>Group Mapp</li> <li>Authorizatio</li> </ul>	ype Network Access 🔅	
ess Policies > Access Services	> Create	
Step 2 - Allowed Pr	otocols	
Process Host Lookup		
Authentication Protocols		
Automotion Protocolo		

Select the previously created Active Directory LDS identity store for the MAB access service.

**Note:** The 'Advanced Options' allows configuring an unknown MAC address policy. This could be very useful in scenarios where users with unknown devices can be granted access to a self-service portal using dACL access restriction and URL-redirection. In the authorization policy,

(System:AuthenticationStatus=UnknownUser|AuthenticationFailed) can be taken into consideration to return appropriate Authorization Profiles.

Access Policies > /	Access Services > SVC_I	MAB > Identity	
Single resu	It selection 🔘 Rule ba	ased result sele	lection
Identity Source:	AD-LDS		Select
	<ul> <li>Advanced Options</li> </ul>		
	If authentication failed	Reject 😂	)
	If user not found	Reject Drop	
	If process failed	Continue	]
	Note: For authentication is not possible to contin found. If continue option	ns using PEAP, ue processing n is selected in	LEAP, EAP-FAST or RADIUS MSCHAP it when authentication fails or user is not these cases, requests will be rejected

#### Test in Cisco Secure ACS View

If everything has been configured correctly, and the switch sends a MAB AAA request to the RADIUS server, the result should look similar to the following screen shot (see below). Note that the selected service is the MAB service (SVC\_MAB), that we've selected the LDAP identity store (here 'LDS'), and that ACS classifies the request as 'Host Lookup'. Also note that ACS retrieved the associated groups for this MAC address ('printers'), which can be used within the authorization policy.

AAA Protocol > RA	DIUS Authentication Detail	
ACS session ID:acs Date:   Jun	53/129728054/753 e 22, 2012	
Generated on June 22,	2012 12:45:15 PM CEST	
Authentication Summa	ary	Actions
Logged At:	June 22,2012 11:36:11.503 AM	Troubleshoot Authenticati
RADIUS Status:	Authentication succeeded	View Diagnostic Messad
NAS Failure:		Audit Network Device Cor
Username:	00-11-22-33-44-55	View Network Device Con
MAC/IP Address:	00-11-22-33-44-55	View ACS Configuration
Network Device:	test-radius : 172.16.24.2 :	
Access Service:	SVC_MAB	
Identity Store:	LDS	
Authorization Profiles:	Permit Access	
CTS Security Group:		
Authentication Method	l: Lookup	
	llt	
User-Name=00-11- Class=CACS:acs53	-22-33-44-55 3/129728054/753	
■Session Events		
Jun 22,12 11:36:1	Radius authentication p MAC: 00-11-22-33-44-5	assed for USER: 00-11-22-33-44-5 5 AUTHTYPE: Host Lookup

Cisco-AVPairs:	
Other Attributes:	ACSVersion=acs-5.3.0.40-B.839 ConfigVersionId=8 Protocol=Radius Service-Type=Call Check Acct-Session-Id=1340357771K84qrk ExternalGroups=CN=printers,OU=MACGroups,OU=MAC,DC=wdf,DC= IdentityDn=CN=myprinter001,OU=printers,OU=MACAddresses,OU=M/ Device IP Address=172.16.23.1
<b>⊞_</b> Steps	
11001 Received RADIUS Ad	ccess-Request
11017 RADIUS created a ne	ew session
11027 Detected Host Look	up UseCase (Service-Type = Call Check (10))
Evaluating Service Selection	n Policy
15004 Matched rule	
15012 Selected Access Ser	vice - SVC_MAB

#### Conclusion

Evaluating Identity Policy 15006 Matched Default Rule 15013 Selected Identity Store - LDS

24031 Sending request to primary LDAP server

24005 Host search finished successfully 24024 Host's groups are retrieved 22037 Authentication Passed

24017 Looking up host in LDAP Server - 00-11-22-33-44-55

By using Active Directory Lightweight Directory Services or any other directory accessible via LDAP, a MAC Authentication Bypass database can be easily used together with Cisco Secure ACS as the RADIUS server.

As shown, the direct approach of creating user objects for MAB is not always the best approach. Using appropriate directory objects, and structuring them in specific containers, is actually a lot more secure and scalable, by allowing for automation and specific permissions applied to those containers.

#### Part II—User Device Binding

#### Introduction

A common requirement within enterprise network environments is to restrict access to network resources if the device is not known or corporate-owned. Especially in environments where users are able to login using a username and password, non-corporate devices can be used to authenticate to the network, because user authentication is not specifically bound to a device.

There have been some approaches in the past to address this problem, e.g.:

- 1) **Machine Access Restrictions (MAR).** Allow a user authentication on a given device only with a prior machine authentication.
- Certificates (Certs). Use non-password based authentication methods, so credentials are tied to a physical device (certificate stores).

3) **EAP Chaining.** Tie the machine authentication and the user authentication together into a single, atomic authentication.

All of the items listed above have their pros and cons, which can be briefly (and non-conclusively) summarized in the following table:

	Pro	Con
MAR	Works with PEAP (password based authentication methods)	<ul> <li>Handles media changes or location changes poorly (e.g.: going from wired to wireless. Sending PC at home to sleep, waking it up in the office)</li> <li>Only works in with devices which have a 'machine authentication concept', no support for other devices, most notably smart phones and tablets (e.g.: iPad, iPhone, Android devices, etc.)</li> </ul>
Certs	Works out of the box if built-in certificate stores are used (tie user certificate to device)	<ul> <li>Does not apply for corporations without Cert infrastructure</li> <li>Does not work with mobile certificate stores (e.g.: smart cards, USB tokens etc.); The standard Windows Supplicant cannot have different authentication methods for Machine and User (e.g. EAP-TLS/Certificate for Machine and Username/Password/PEAP for the User)</li> </ul>
EAP-Chaining	<ul><li>Best integrated</li><li>Most secure method</li></ul>	<ul> <li>Limited to EAP-FAST</li> <li>Limited in terms of platform support (supplicant and authenticator must support EAP chaining)</li> <li>No OS built-in supplicant support; however, starting with release 3.1, Cisco AnyConnect Secure Mobility Client supports EAP-Chaining</li> </ul>
User Device Binding	<ul><li>Easy to implement</li><li>No client modification required</li><li>Media independent</li></ul>	<ul><li> 'only' MAC address security</li><li>MAC database required</li><li>Not supported on every RADIUS server</li></ul>

## User Device Binding with Directories

This document suggests a different approach to solve this problem, using specific attributes stored in a directory and bound to a user.

The advantage here is certainly the reduced complexity for this approach: There are no certificates required and this will work with standard supplicants. It will also work independent of the medium used (wired/wireless) and the operating system.

On the downside, it has to be noted that the security of this approach is reduced to the MAC address of the device. A user with proper knowledge and adequate access rights can forge an arbitrary MAC address on that particular device to circumvent the mechanism.

Setup and organization of MAC address objects has been discussed in the previous section of this document. Now we look at the linkage of MAC addresses to user objects, within a directory.

**Note:** To enable *User Device Binding*, it is not required to create a MAB directory structure and/or device objects. It's sufficient to reference the MAC addresses in the user object. However, creating these additional objects and references may actually be useful for other, advanced scenarios. Examples are PXE booting machines for OS restore/setup or restricted access scenarios in case of credential failure. Both would use MAB as a fallback mechanism to allow the device on the network.

This **User Device Binding** can be used to augment plain IEEE 802.1X, and can enhance the overall security by allowing access to the network only to devices that are associated to a particular user. It also has the benefit that there is no dependency on the availability of any supplicant/client-software specific feature on the end device.

**Note:** Since the method relies on a MAC/Username match, it does work for all authentication methods involving a username and a MAC address. Namely EAP-TLS, PEAP and EAP-FAST. It does not work for MAB (the MAC **is** the username) and also not for WebAuth (no MAC address is available in the CallingStationID).

**Note:** In current (as of late 2012) WebAuth for wired switch ports does not send the MAC address in the CallingStationID. However, for Wireless it can send the MAC address in the CallingStationID. This is the default setting when RADIUS NAC is enabled on the WLC.

For every IEEE 802.1X authentication request, the RADIUS **CallingStationID** is set to the MAC address of the device requesting access. We can check the validity of the access request if the presented MAC address is associated (thus allowed) with the user (from the RADIUS **User-Name**), by checking this association in the directory.

To enforce this, we implement a policy on the authentication server, which checks the ownership of the device and restricts the access if the user accesses the network with a device that is not owned by the user.

**Note:** This technique is only relevant if user authentications are being handled. Machine authentications are OK **per se** because the machine account (or machine credentials) is more tightly controllable. However, the same method could also be applied to a machine object, if needed.

#### **Cross-Referencing Objects**

Ideally, device objects would be directly connected to person objects. In fact, AD allows building an object hierarchy to reflect the ownership. As described in the previous section, all the devices will be created within a specific OU and of type 'device', and owned by user objects. This is illustrated by the following screen shot (see below). Note that the **owner** attribute of the device points to the DN of the user object:

🖃 🟥 stu.ibns.lab	<ul> <li>Nan</li> </ul>	ne T	ype	Description	
🕀 🧮 Builtin		xpsp3 d	levice	my XP SP3 MAB host entry (00-0C-2	
🛨 🧮 Computers		2.0		9 X	1
🖅 📄 DeviceAdmin	×	osp3 Properties			ų
王 🧾 Domain Controllers		Ganaral   Object   S	Attribut	te Editor	I
🖅 🧮 ForeignSecurityPrincipals	11	deneral   object   of	ocumy rando	1	I
🛨 🧮 LostAndFound		Attributes:			I
🖃 📄 MAC		Auchula	1 Velue		I
Addresses		Attnoute	value		I
Cameras		cn	xpsp3		I
mabhosts		description	my XP S	P3 MAB host entry (00-0C-29-7A-B5-24)	I
nabinosis nabinosis		macAddress	000C29	7AB524	I
priories		objectCategory	CN=Dev	vice CN=Schema CN=Configuration DC=	I
princers		objectClass	ton: dev	ice	I
a regularnosts		objectoidaa	top, dev		I
🛨 🗖 Groups		owner	CN=Kar	Napt,CN=Users,DC=stu,DC=ibns,DC=la	I
🛨 🧾 Program Data		showInAdvanced	/ie FALSE		

This ownership is then reflected within the user object as a backtracking list to the devices the user actually owns (in the user object schema, this list object has the name 'ownerBL'<sup>9</sup>):

Attributes:		Multi-valued String Editor	
Attribute	Value	Attribute: ownerBL	
objectCategory	CN=Person.(		
objectClass	top; person;	Value to add:	
objectGUID	51e6e014-et		Add
objectSid	S-1-5-21-991	Values:	
ownerBL	3.0	CN=mycamera007.011=cameras.011=Addresses.011	Romour
primaryGroupID	010	CN=myphone002.0U=phones.0U=Addresses.0U=N	Themove
pwdLastSet	4/7/2011 2:	CN=xpsp3,OU=regularhosts,OU=Addresses,OU=MA	
replPropertyMetaData	AttID Ver		0.
sAMAccountName	knapf		
sAMAccount Type	805306368 :		
sDRightsEffective	15		
sn	Napt		
structuralObjectClass	top; person;		
subschemasubEntry	CIN=Aggrega		
•			
14		٠	
view			
		OK	Cancel

A policy rule on the authentication server (RADIUS server) to reflect the ownership would look something like this (in meta-syntax):

```
device-owner=searchLDAP(Calling-Station-ID)->owner
If user->distinguishedName == device-owner Then
    # the user actually owns the device
Permit-all
Else
    # the user does *not* own the device
Restrict-access
EndIf
```

<sup>&</sup>lt;sup>9</sup> ownerBL=Owner Backtracking List

ioure control   Security			incore coror   Security	
tributes:		_	Attributes:	
Attribute	Value	<u> </u>	Attribute	Value
fromEntry	TRUE		canonicalName	stulibns.lab/Users/Karl Napf
instanceType	0x4 = (WRITE)		cn	Karl Napf
macAddress	000c297ab524		codePage	0
memberOf	CN=mabhosts.OU=Groups.OU=MAC.DC=stu		countryCode	0
modify Time Stamp	9/19/2012 1:14:47 PM W. Europe Standard		create Time Stamp	4/7/2011 2:25:00 PM W. Europe Standard 1
msDS-Approx-Immed	0		displayName	Karl Napf
msDS-ReplAttributeM	<ds_repl_attr_meta_data><pszattrik< td=""><td></td><td>distinguishedName</td><td>CN=Karl Napf,CN=Users,DC=stu,DC=ibns,D</td></pszattrik<></ds_repl_attr_meta_data>		distinguishedName	CN=Karl Napf,CN=Users,DC=stu,DC=ibns,D
msDS-ReplValueMet	<ds_repl_value_meta_data><pszattr< td=""><td></td><td>dSCorePropagationD</td><td>0x0 = ( )</td></pszattr<></ds_repl_value_meta_data>		dSCorePropagationD	0x0 = ( )
msDS-RevealedListBL		0	fromEntry	TRUE
name	xpsp3	- ~	givenName	Karl
objectCategory	CN=Device,CN=Schema,CN=Configuration,I	100	instance Type	0x4 = (WRITE)
objectClass	top; device	S	lastLogoff	(never)
objectGUID	013rd64d-c024-4bd5-bd19-a75c85d71005		lastLogon	4/7/2011 2:26:29 PM W. Europe Standard 1
owner	CN=Karl Napf,CN=Users,DC=stu,DC=ibns,D		lastLogon Timestamp	4/7/2011 2:26:29 PM W. Europe Standard 1
d			41	
Ede	Fiter	1	Exte	Filter
	Liter		L'UN	1 80

Unfortunately, neither ACS nor ISE can actually perform an additional LDAP search with a key that is different from the RADIUS User-Name (in this case, the key would have to be the MAC address which is actually available within the Calling-Station-ID of the user authentication request).

The Perl Script in the appendix is able to create the relationship between the device objects and the user objects, given user IDs are provided as part of the input table/CSV file.

#### User Device Binding Implementation

Another, alternative way to address the User Device Binding problem is to leverage a user object attribute, which is readily available and can be used for this task. This is the 'Calling Station ID'/'Verify Caller-ID' attribute every user object contains.

This attribute has the added benefit of being easily configurable/editable through the standard user object properties dialog on the 'Dial-in' tab.

And since this is a text field, it can also hold a string of multiple MAC addresses. This suits the requirement that a user can typically own more than one device. Or the fact that a typical PC device has at least two MAC addresses: Wired and wireless.

#### **Directory Tie-in**

The following screen shot illustrates an example of how a user can own a couple of devices based on the MAC addresses of these devices.

**Note:** This technique uses a standard Active Directory attribute, which is readily available without any schema modification. In addition, it is easily editable via the GUI or via automation tools.

Another way to populate these fields is via an LDIF file. This file can be automatically created from, for example, an Excel datasheet and imported into Active Directory using ldifde.exe on Windows.

The Perl script in the appendix can take a comma-separated table, creates the device objects and optionally, fills the 'Verify Caller-ID' in the user object as well.

This field can hold multiple MAC addresses by concatenating them with a separator. In this example we're using '|' as this could be used as a regular expression with ISE or with a substring match on Cisco Secure ACS 5.

			? >
Security Environment S Terminal Services Profile CO General Address Account Profil Published Certificates Member Of Pass	essions   )M+   e   Telepho word Replicatio	Remote c Attribute Ed nes   Orga on Dial-in	ontrol ditor nization Object
Network Access Permission C Allow access C Deny access C Control access through NPS Network	Policy		
Verify Caller-ID:	00-11-22-33-4	44-57 00-0C-2	29-1
Caliback Options			
Caliback Options No Caliback Set by Caller (Routing and Remote Ac Always Caliback to:	cess Service o	unly)	_
Caliback Options   No Callback  C Set by Caller (Routing and Remote Ac  Always Callback to:  Assign Static IP Addresses	cess Service o	only)	
Caliback Options Caliback C Set by Caller (Routing and Remote Ac Always Caliback to: Assign Static IP Addresses Define IP addresses to enable for this Dial-in connection.	Cess Service o	o <b>nly)</b> Addresses	

**Note:** The "|" (pipe) character separates the MAC addresses to allow for RegEx<sup>10</sup> support. MAC addresses must be formatted like xx-xx-xx-xx, including the dashes (they must be entered in the same way as the access device sends them in the **CallingStationID**).

**Note:** If the 'Verify Caller-ID:' attribute is actually in use for regular dial-up caller screening, the regular caller ID can be included into the attribute as well separating by "|". Alternatively, another unused attribute could be used for the MAC address. The 'Notes' field on the 'Telephones' tab can also successfully be used.

<sup>&</sup>lt;sup>10</sup> RegEx Regular Expression

<sup>© 2013</sup> Cisco and/or its affiliates. All rights reserved. This document is Cisco Public.

#### ACS Device Restriction Policy

Now, with this tie-in of device ownership into a user object, we can actually build a policy in ACS. For the actual rule, we are leveraging a specific feature of ACS (the dynamic LHS/RHS attribute comparison which was introduced with 5.3). Here are the steps to actually make this work, see below.

#### ACS Policy

Make the **msNPCallingStationID** field available for policy decisions in ACS as shown in the next screen shot. The 'Policy Condition Name' can be set to something meaningful.

General	Directory Groups	Directory At	tributes			
Directory a policy rule If you wish Name of e	attributes of user or s s here. Specify a san to modify the Defaul example Subject to Se	ubject records nple user / su It and Policy C elect Attributes	can be referenced a bject name below, the ondition Name for an	s policy conditions in ; en click 'Select' to lau attribute, edit it in the t	policy rules. If you wish to do this, define t unch a dialog to select attributes from this table below.	ne attribut subject.
rschmied	1				Select	
Attribute	l Name	Tj	/pe	Default	Select Policy Condition Name	

Build an Authorization Policy Rule for your IEEE 802.1X authentication service that says (in plain English): If the MAC address of the AAA request in the Calling-Station-ID is **not** contained in the **msNPCallingStationID** attribute of the user object, then restrict the user-access.

ietw	ork Ac	cess Aut	norization	Policy		
Filter	r: Stat	us		Match if: Equals 💌 Enabled 💌 Clear Filter Go 🗢		
		Status	Name	Conditions Compound Condition	Results Authorization Profiles	Hit Cour
1	П	0	Rule-1	AD-AD1:msNPCallingStationID not contains RADIUS-IETF:Calling-Station-ID	RESTRICTED	21

As stated before, this is possible because with ACS 5.3, a 'dynamic' comparison between attributes is available.

Conditions		
Compound Condition:		
Condition:		
Dictionary:	Attribute:	_
AD-AD1	msNPCallingStationID	Select
Operator:	Value:	
not contains 💌	Dynamic 🔽	
Dictionary:	Attribute:	
RADIUS-IETF	Calling-Station-ID	Select
Current Condition Set:		
Add ▼ Edit ∧	Replace V Delete	
AD-AD1:msNPCallingStati	ionID not contains RADIUS-IETF:Calling-S	tation- 📥

This condition performs a simple string match of the RHS<sup>11</sup> (the incoming RADIUS Calling-Station-ID) and the LHS (the **msNPCallingStationID** field coming from the Active Directory user object). As mentioned above, the MAC address has to match exactly, including the hyphens. The "|" separating multiple MAC addresses in the **msNPCallingStationID** string could also be a different character, as long as the complete MAC address can be matched ('contains'/not contains' operator).

**Note:** Starting with version 1.1.1, ISE does also support variable LHS:RHS conditions. In Addition, ISE can perform a RegEx match. Therefore, the proposed concept can also be implemented with ISE (though not tested at the time of writing of this document). Microsoft NPS can also do the RegEx match, as documented in the TechEd article in the appendix.

#### Enforcement

If this rule matches, the administrator then can limit/restrict the access of the user who is using a device that is not registered to him/her.

In this simple proof of concept setup, we've returned a specific ACL for restricted access combined with a URLredirection using a specific Cisco Attribute Value Pair (AV-Pair). Further details on how to setup URL-Redirection can be found in the Web Authentication Deployment Guide (see link in the appendix).

Whenever the user opens up a browser, it will be redirected to a web page, stating that the user has 'Restricted Access' and why this is happening:

<sup>&</sup>lt;sup>11</sup> LHS Left hand side / RHS Right hand side

## Radlogin

**ACCESS is RESTRICTED!** 

you're not allowed to access the network with a device that is not registered to you!

## Conclusion

With this simple device restriction policy, the use of unknown devices can be effectively restricted, even with password-based authentication methods.

It might not scale for thousands of devices and it is potentially less secure than the EAP-Chaining or any certificate based approach, but it is certainly one step up from using the user password on an arbitrary, non-corporate device to access the network.

## Appendix

#### LDIF Script

This Perl script reads a comma separated value (CSV) file from the standard input or from the first command line argument, and converts it into an LDIF file. There are a number of parameters to influence the LDIF generation. See the help text using 'mac-ldif.pl --help' for reference.

The script can also cross-reference devices with user IDs, by querying the LDAP store for DNs of the user ID and referencing the owner in the device (--owner switch). It assigns the DN of the user ID to the owner attribute of the device object.

In addition, it can populate the 'Verify Caller ID' field of the DN (the 'user object') with the MAC addresses of the user owned devices (--clid switch). Multiple devices will be separated by a '|' (pipe) character. This is useful if the RADIUS server supports a RegEx search of the CallingStationID in this string, or a substring match (in this case, both variants would work).

```
#!/usr/bin/perl
#
#
# (c) 2012 Cisco
#
# build LDIF from CSV w/ MAC address info
# use --help
#
use strict;
use warnings;
use File::Basename;
use Getopt::Long;
```

```
use Net::LDAP;
use Net::LDAP::Util qw(ldap_error_text);
use Term::ReadLine;
# stuff to adapt
my $domain="DC=stu,DC=ibns,DC=lab";
my $container="OU=MAC".",".$domain;
my $mac_ou="MACAddresses";
my $group_ou="MACGroups";
# for user lookup (adapt!)
# domain will be appended
# if password is empty, you will be prompted for it.
my $ldap_host="localhost";
my $ldap_port=389;
my $user_base="CN=Users";
my $binddn="CN=Administrator";
my $bindpw="your_ldap_password_here";
my $clid_sep="|";
# various flags (see help)
my $ieee802=0;
my $help=0;
my $separate=0;
my $owner=0;
my $clid=0;
# variables
my $ldap;
#
# read a password, don't echo anything
# unix specific, look into Term::ReadPassword
# for something more generic.
#
sub read_password {
   my ($prompt)=@_;
   my $term=Term::ReadLine->new('ldap');
   system('stty','-echo');
   my $password=$term->readline($prompt);
   system('stty','echo');
   print "\n";
   return $password;
}
```

```
#
```

```
# print a formatted MAC address
#
sub format_mac {
   my ($mac)=@_;
   $mac =~ tr/a-f/A-F/;
   return join ("-", unpack ("A2A2A2A2A2A2",$mac));
}
#
# do ldap related stuff + error handling
# if something goes wrong
#
sub ldap_call {
   my $mesg=shift;
   my $action=shift;
   if ($mesg->code) {
      die "An error occurred $action: "
       .ldap_error_text($mesg->code)."\n";
   }
   return $mesg;
}
#
# generate some header info
#
sub output_header {
   print "#\n#\n";
   print "# generated by ".basename($0)."\n";
   print "# ".(localtime)."\n";
   print "#\n";
   print "# import via ldifde.exe into Active Directory.\n";
   print "# ldifde -i -k -f input.ldf\n";
   print "#\n#\n\n";
}
#
# output the container OU
#
sub output_container {
   my ($dn, $desc)=@_;
   my $cn;
   print "dn: ".$dn."\n";
   print "changetype: add\n";
   print "objectClass: top\n";
   print "objectClass: organizationalUnit\n";
   ($cn=$dn) =~ s/^OU=(\w+),.*$/$1/;
```

```
print "ou: ".$cn."\n";
   if ($desc) { print "description: ".$desc."\n"; }
   print "\n";
}
#
# returns the formatted MAC address
# remove all ':', '.' and '-'
# convert all uppercase hex chars to lowercase
#
sub flatten_mac {
   my ($mac)=@_;
   $mac =~ s/[:.-]//g;
   $mac =~ tr/a-f/A-F/;
   return $mac;
}
#
# output an individual device
#
sub output_device {
   my ($ou, $mac, $name, $desc, $ieee802, $owner)=@_;
   my $output=$ieee802 ? $name : $mac;
   print "dn: CN=".$output.",".$ou."\n";
   print "changetype: add\n";
   print "objectClass: top\n";
   print "objectClass: device\n";
   # print "distinguishedName: CN=".$output.",".$ou."\n";
   if ($ieee802) {
      print "macAddress: $mac\n";
      print "name: ".$name."\n";
      $desc=$desc . " (" . format_mac($mac) . ")";
   } else {
      print "name: ".$mac."\n";
      $desc=$desc . " (" . $name . ")";
   }
   print "showInAdvancedViewOnly: FALSE\n";
   if ($owner) { print "owner: ".$owner."\n"; }
   if ($desc) { print "description: ".$desc."\n"; }
   print "\n";
}
# output top part of the device group
#
sub output_group_top {
   my ($ou, $name)=@_;
```

```
print "dn: CN=".$name.",".$ou."\n";
   print "changetype: add\n";
   print "objectClass: top\n";
   # print "objectClass: groupOfUniqueNames\n";
   print "objectClass: group\n";
   print "cn: ".$name."\n";
   # print "distinguishedName: CN=".$name.",".$ou."\n";
}
#
# output trailing part of the device group
#
sub output_group_bottom {
   my ($name)=@_;
   print "name: ".$name."\n\n";
}
#
# output a member for the device group
#
sub output_group_member {
   my ($member, $ou)=@_;
   # print "uniqueMember: CN=".$member.",".$ou."\n";
   print "member: CN=".$member.",".$ou."\n";
}
sub get_optional_ou {
   my ($ou)=@_;
   if (length($ou) > 0) {
      return "OU=".$ou.",";
   } else {
      return "";
   }
}
sub help;
#
# main()
#
GetOptions(
            'container=s' => \$container,
   'ieee802'
                => \$ieee802,
                 => \$help,
   'help'
   'macou:s'
                => \$mac_ou,
   'groupou:s' => \$group_ou,
   'owner'
                 => \$owner,
   'l|clid'
                 => \$clid,
```

```
'separate'
                => \$separate
) or die "Incorrect usage! Try --help\n";
if ($help) {
   help;
   exit;
}
# setup LDAP connection if needed
if ($owner || $clid) {
   # ask for password if there's none
   my $dn=$binddn.",".$user_base.",".$domain;
   if ($bindpw eq "") {
      $bindpw=read_password("Password for $dn: ");
   }
   $ldap=Net::LDAP->new($ldap_host, port=>$ldap_port)
      or die "Unable to connect to LDAP server $ldap_host: $@\n";
   my $result=ldap_call($ldap->bind(dn => $dn, password => $bindpw),
      "binding to server");;
}
# create header and top container
output header;
output_container $container, "Top MAC address container";
# create optional sub containers
if ($mac_ou ne "") {
   my $ou=(get_optional_ou $mac_ou).$container;
   output_container $ou, "MAC container";
}
if ($group_ou ne "") {
   my $ou=(get_optional_ou $group_ou).$container;
   output_container $ou, "Group container";
}
my %groups;
my %owners_dn;
my %owners macs;
#
# for every input line
# comma separated list
# mac-address, group, name, description, owner
#
while (<>) {
   my $line=$_;
   my $lives_in;
```

```
chomp $line;
   if ($line) {
      my @fields=split "," , $line;
      my ($mac, $name, $group, $desc, $owner_id)=@fields;
      if ($separate) {
          $lives_in="OU=$group,".(get_optional_ou $mac_ou).$container;
          if (not exists ($groups{$group})) {
             output_container $lives_in, "Container for $group";
          }
      } else {
          $lives_in=(get_optional_ou $mac_ou).$container;
      }
      $mac=flatten_mac($mac);
      # get the DN from LDAP, if required
      my $owner_dn="";
      if (($clid || $owner) && defined $owner_id) {
          # if not previously found
          $owner_dn=$owners_dn{$owner_id};
          if (not defined $owner_dn) {
             my $sr=ldap_call($ldap->search(base => $user_base.",".$domain,
                filter => "(&(objectclass=person)(sAMAccountName=$owner_id))",
                scope => "one"),
                 "searching the LDAP server");
             if ($sr->count == 1) {
                $owner_dn=$sr->entry(0)->dn();
                $owners_dn{$owner_id}=$owner_dn;
             }
          }
          push (@{$owners_macs{$owner_id}}, $mac );
      }
      output_device ($lives_in, $mac, $name, $desc,
          $ieee802, ($owner?$owner_dn:undef));
      push (@{$groups{$group}}, [ ($ieee802?$name:$mac, $lives_in) ]);
   }
}
#
# build the groups from the MAC addresses
# hash of arrays of tuples (MAC addresses,OU)
#
{
   my $group_lives_in=(get_optional_ou $group_ou).$container;
```

```
foreach my $key (keys %groups) {
       output_group_top $group_lives_in, $key;
       foreach( @{$groups{$key}}) {
          my ($mac, $group)=($_->[0], $_->[1]) ;
          output_group_member ($mac, $group);
       }
      output_group_bottom $key;
   }
}
#
# this last section populates the msNPCallingStationID
# with the MAC addresses of the user if option is set
#
if ($clid) {
   sub output_user_top {
      my ($dn)=@_;
      print "dn: ", $dn, "\n";
      print "changetype: modify\n";
      print "replace: msNPCallingStationID\n";
      print "msNPCallingStationID: ";
   }
   foreach my $owner (keys %owners_dn) {
       output_user_top $owners_dn{$owner};
      my $total=$#{$owners_macs{$owner}};
       foreach (@{$owners_macs{$owner}}) {
          print format_mac($_);
          if ($total--) {
             print $clid_sep;
          }
       }
      print "\n-\n\n";
   }
}
exit;
#
# output some help text
#
sub help {
   my $helptext;
   my $name=basename($0);
   $helptext=<< "EOF";</pre>
Usage: $name [inputfile]
```

```
$name reads a comma separated list of MAC address devices and
converts these MAC addresses into an LDAP LDIF file on the
standard output. The input can be either the standard input or
files passed as arguments to $name.
The format is
MAC_address, Device_Name, Group_Name, Description, Owner
00:11:22:33:44:57,myprinter003,printers,device-description,owner1
Optional command line parameters are:
--container, -c: specify the OU where the objects should be stored.
                 (default is "$container")
--help, -h:
                this text
--ieee802 -i:
                make use of ieee802Device objects (AD 2008+)
--macou, -m:
                OU of the MAC address container, can be empty
--groupou, -g: OU of the group container, can be empty
--separate, -s: flag to indicate creation of individual group OUs
--owner, -o:
                create owner backreferences (LDAP setup in script required)
--clid -l:
                create msNPCallingStationID references
                 (LDAP setup in script required)
Example:
$name --container="OU=devices,DC=labtest,DC=local" input.csv
$name -m= -g= input.csv
Note:
$name doesn't do a lot of testing of the command line paramters
and the data format of the input file. Use with caution!
All MAC addressess will be stripped of any '.', ':' and '-'.
LDAP setup:
If owner backreferences or calling station ID information should be
set, a working LDAP connection is requried to resolve the user id
information in the data file to a DN of that user. Therefore, some
script variables have to be setup with the correct values to do
an LDAP search using the user ID (sAMAccountName) to get the DN
of this user. Look at the top of the script.
EOF
   print $helptext;
}
```

#### Sample Input File

Here's an example that shows the structure of the input file for the mac-ldif.pl script. The table structure is as follows:

MAC	Name	Group	Description	Owner
-----	------	-------	-------------	-------

00:11:22:33:44:55,myprinter001,printers,updated device text,guest 0011.2233.4456,myprinter002,printers,,rmueller 00:11:A2:33:44:59,myphone001,phones,description, 0011:2A33:445b,mycamera007,cameras,detailed description of cam7,knapf 00.11.22.33.44.57,myprinter003,printers,description of this device,rschmied 00:11:FF:33:44:5a,myphone002,phones,phone device description,netdudes 0011-2233-445c,mycamera003,cameras,in room 22/b,rmueller 00:11:22:33:44:58,myprinter004,printers,dummy text,owner3 000C299B1948,vmwarehost,mabhosts,my vmware MAB host,rschmied 000C2985353B,linux01,mabhosts,my Linux PXE 01 host,rschmied 000C297AB524,xpsp3,regularhosts,my XP SP3 MAB host entry,knapf

#### Sample Output File

The following LDIF file is a result of the mac-ldif.pl script run on the above data given the following command line:

```
mac-ldif.pl -con "OU=mab,DC=cisco,DC=com" -g "groups" -m "macs" -i -o -l data.txt
>import.ldf
#
#
# generated by mac-ldif.pl
# Tue Aug 14 14:26:48 2012
#
# import via ldifde.exe into Active Directory.
# ldifde -i -k -f input.ldf
#
#
dn: OU=mab, DC=cisco, DC=com
changetype: add
objectClass: top
objectClass: organizationalUnit
ou: mab
description: Top MAC address container
dn: OU=macs,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: organizationalUnit
ou: macs
description: MAC container
dn: OU=groups,OU=mab,DC=cisco,DC=com
```

changetype: add

```
objectClass: top
objectClass: organizationalUnit
ou: groups
description: Group container
```

```
dn: CN=myprinter001,OU=macs,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: device
macAddress: 001122334455
name: myprinter001
showInAdvancedViewOnly: FALSE
owner: CN=Guest,CN=Users,DC=cisco,DC=com
description: updated device text (00-11-22-33-44-55)
```

dn: CN=myprinter002,OU=macs,OU=mab,DC=cisco,DC=com changetype: add objectClass: top objectClass: device macAddress: 001122334456 name: myprinter002 showInAdvancedViewOnly: FALSE owner: CN=Roland Mueller,CN=Users,DC=cisco,DC=com description: (00-11-22-33-44-56)

```
dn: CN=myphone001,OU=macs,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: device
macAddress: 0011A2334459
name: myphone001
showInAdvancedViewOnly: FALSE
description: description (00-11-A2-33-44-59)
```

dn: CN=mycamera007,OU=macs,OU=mab,DC=cisco,DC=com changetype: add objectClass: top objectClass: device macAddress: 00112A33445B name: mycamera007 showInAdvancedViewOnly: FALSE owner: CN=Karl Napf,CN=Users,DC=cisco,DC=com description: detailed description of cam7 (00-11-2A-33-44-5B)

dn: CN=myprinter003,OU=macs,OU=mab,DC=cisco,DC=com changetype: add objectClass: top

```
objectClass: device
macAddress: 001122334457
name: myprinter003
showInAdvancedViewOnly: FALSE
owner: CN=Ralph Schmieder,CN=Users,DC=cisco,DC=com
description: description of this device (00-11-22-33-44-57)
```

```
dn: CN=myphone002,OU=macs,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: device
macAddress: 0011FF33445A
name: myphone002
showInAdvancedViewOnly: FALSE
description: phone device description (00-11-FF-33-44-5A)
```

```
dn: CN=mycamera003,OU=macs,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: device
macAddress: 00112233445C
name: mycamera003
showInAdvancedViewOnly: FALSE
owner: CN=Roland Mueller,CN=Users,DC=cisco,DC=com
description: in room 22/b (00-11-22-33-44-5C)
```

```
dn: CN=myprinter004,OU=macs,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: device
macAddress: 001122334458
name: myprinter004
showInAdvancedViewOnly: FALSE
description: dummy text (00-11-22-33-44-58)
```

```
dn: CN=vmwarehost,OU=macs,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: device
macAddress: 000C299B1948
name: vmwarehost
showInAdvancedViewOnly: FALSE
owner: CN=Ralph Schmieder,CN=Users,DC=cisco,DC=com
description: my vmware MAB host (00-0C-29-9B-19-48)
```

```
dn: CN=linux01,OU=macs,OU=mab,DC=cisco,DC=com
changetype: add
```

```
objectClass: top
objectClass: device
macAddress: 000C2985353B
name: linux01
showInAdvancedViewOnly: FALSE
owner: CN=Ralph Schmieder, CN=Users, DC=cisco, DC=com
description: my Linux PXE 01 host (00-0C-29-85-35-3B)
dn: CN=xpsp3,OU=macs,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: device
macAddress: 000C297AB524
name: xpsp3
showInAdvancedViewOnly: FALSE
owner: CN=Karl Napf, CN=Users, DC=cisco, DC=com
description: my XP SP3 MAB host entry (00-0C-29-7A-B5-24)
dn: CN=printers,OU=groups,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: group
cn: printers
member: CN=myprinter001,OU=macs,OU=mab,DC=cisco,DC=com
member: CN=myprinter002,OU=macs,OU=mab,DC=cisco,DC=com
member: CN=myprinter003,OU=macs,OU=mab,DC=cisco,DC=com
member: CN=myprinter004,OU=macs,OU=mab,DC=cisco,DC=com
name: printers
dn: CN=cameras,OU=groups,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: group
cn: cameras
member: CN=mycamera007,OU=macs,OU=mab,DC=cisco,DC=com
member: CN=mycamera003,OU=macs,OU=mab,DC=cisco,DC=com
name: cameras
dn: CN=mabhosts,OU=groups,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: group
cn: mabhosts
member: CN=vmwarehost,OU=macs,OU=mab,DC=cisco,DC=com
member: CN=linux01,OU=macs,OU=mab,DC=cisco,DC=com
name: mabhosts
```

dn: CN=regularhosts,OU=groups,OU=mab,DC=cisco,DC=com

```
changetype: add
objectClass: top
objectClass: group
cn: regularhosts
member: CN=xpsp3,OU=macs,OU=mab,DC=cisco,DC=com
name: mabhosts
```

```
dn: CN=phones,OU=groups,OU=mab,DC=cisco,DC=com
changetype: add
objectClass: top
objectClass: group
cn: phones
member: CN=myphone001,OU=macs,OU=mab,DC=cisco,DC=com
member: CN=myphone002,OU=macs,OU=mab,DC=cisco,DC=com
name: phones
```

```
dn: CN=Karl Napf,CN=Users,DC=cisco,DC=com
changetype: modify
replace: msNPCallingStationID
msNPCallingStationID: 00-11-2A-33-44-5B|00-0C-29-7A-B5-24
```

```
dn: CN=Ralph Schmieder,CN=Users,DC=cisco,DC=com
changetype: modify
replace: msNPCallingStationID
msNPCallingStationID: 00-11-22-33-44-57|00-0C-29-9B-19-48|00-0C-29-85-35-3B
```

```
dn: CN=Roland Mueller,CN=Users,DC=cisco,DC=com
changetype: modify
replace: msNPCallingStationID
msNPCallingStationID: 00-11-22-33-44-56|00-11-22-33-44-5C
```

```
dn: CN=Guest,CN=Users,DC=cisco,DC=com
changetype: modify
replace: msNPCallingStationID
msNPCallingStationID: 00-11-22-33-44-55
```

## **References and Links**

\_

Remote Authentication Dial In User Service (RADIUS)

MAC Authentication Bypass Deployment Guide

Local WebAuth Deployment Guide

Web Authentication Deployment and Configuration Guide

Using LDIFDE to Import and Export Directory Objects to Active Directory

Enhance Your 802.1x Deployment Security with MAC Filtering

Lightweight Directory Access Protocol

Perl (including binary versions for Windows)

Active Directory Lightweight Directory Services

AD DS Fine-Grained Password and Account Lockout Policy Step-by-Step Guide



Americas Headquarters Cisco Systems, Inc. San Jose, CA Asia Pacific Headquarters Cisco Systems (USA) Pte. Ltd. Singapore Europe Headquarters Cisco Systems International BV Amsterdam, The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Third party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

Printed in USA