Mac Configuration Management at the Los Alamos National Laboratory

By Allan Marcus

April, 2010

Cleared for public release

LA-UR 10-03133

Overview

The Los Alamos National Laboratory (LANL) had a need for central configuration management of non-Windows computers. LANL has three to five thousand Macs and an equal number of Linux based systems. The primary goal was to be able to inventory all non-windows systems and patch Mc OS X systems. LANL examined a number of commercial and open source solutions and ultimately selected Puppet.

This paper will discuss why we chose Puppet, how we implemented it, and some lessons we learned along the way.

The Challenge

For the following reasons, LANL has determined that better non-windows configuration management was needed.

NNSA Administrative Policy (NAP) 14.2-C applies to LANL, and this NAP requires "the system owner must maintain configuration control over the system to ensure that the security posture of the system is not threatened by authorized or unauthorized changes to system software or hardware. [1]" In addition, the NAP's Configuration Management family of controls[2] has a number of requirements that essentially require monitoring and maintenance of configuration settings.

Also, over the years LANL has been notified through audit findings that better non-Windows configuration management is needed. For Windows LANL uses SCCM from Microsoft. For Red Hat Enterprise Linux (RHEL), LANL uses their satellite server. LANL still has a significant number of non-RHEL Linux machines, as well as a few Sun machines.

LANL's IT management decided that if the following criteria was met, LANL would have sufficient non-Windows configuration management, and would provide better IT service to our non-Windows users.

- Ability to report on what we have on the network
- Ability to respond to “find this file” data calls
- Ability to report on patch levels and SAV definition dates
- Ability for field teams (or central services) to easily patch machines

For the purposes of this paper, non-Windows configuration management is restricted to Mac OS X.

Our challenge was to find a solution that met the above criteria.

[1] NAP 14.2-C III-17
[2] NAP 14.2-C IV-35

Product Selection

We looked at a number of tools, both commercial off-the-shelf (COTS) and open source solutions. Our criteria was fairly basic:

- Hardware inventory
- Software Inventory
- Software and Patch Distribution
- ad-hoc script execution
- Direct SQL access to data
- Runs on Mac OS X
- Runs on Red Hat Enterprise Linux (RHEL)
- Runs on other UNIX and Linux variants

Initially budget was not a major concern, although the on-going costs–the personnel mortgage, if you will–was a concern. We also took into account the "buy or build" issue. The buy option would cost more upfront, be more feature rich, and is theoretically cheaper in the long run. The build option would be cheap up front, but would take more human resources to add features and manage over time.

Products

We identified a number of products that we felt warranted further examination. Please note that the following analysis is based on information when we did our study (2008). Product features and pricing may have changed since then.

- KBOX
- Quest QMX
- Puppet
- cfengine
- LANRev
- FileWave
- Casper Suite
- Radmind

KBOX

The KBOX from Kace[3] is an appliance that can also be run as a virtual appliance. The KBOX supports RHEL, Mac and Windows and requires almost no set-up time. The KBOX would allow LANL to deploy packages and populate custom inventory fields with minimal effort. The KBOX also included a rudimentary issue tracking module and a knowledge base tool.

[3] http://www.kace.com Note: our evaluation of KBOX was before Dell purchased it.

Quest Management Xtensions for SMS

The Quest Management Xtensions for SMS[4] was also a very interesting tool that garnered much attention from our evaluation team. This tool is an add-on to SMS 2003, which LANL already uses to manage our Windows environment. This tool runs on Mac OS X as well as many flavors of Linux and UNIX. We had some concerns that pushed us away from the Quest Management Xtensions for SMS. The first was tying non-windows management to SMS. We were concerned about our ability to upgrade or update the SMS server and how Quest Management Xtensions for SMS development might lag behind SMS development. We were also concerned with the perception of using a Microsoft product to manage non-Windows computers. Many non-windows users have a distrust of Microsoft products. Also, custom inventory items are not easy to implement using this tool and SMS. Finally, our SMS staff is sorely overworked, so we were concerned with the timeliness of the SMS staff to adress non-Windows requests.

Puppet

Puppet is a free and open sources solution for automating system administration. Puppet is maintained by Pupept Labs[5] (formerly Reductive Labs), Inc., who provide optional training and consultation services for Puppet. Puppet is a client server tool with clients pre-built for most flavors of Linux, Mac, and UNIX. Puppet reports basic inventory information to a central SQL database. Custom inventory data, including hardware, software, or data-call information, can fairly easily be added. Only current inventory is maintained, but an inventory history could be added fairly easily with custom database programming.

Puppet can be used to bring a machine to a known state and keep it in that state.

cfengine

Cfengine is an open sourse project managed by the Cfengine company. [6] Cfengine only has a CLI and requires programming knowledge to implement. LANL has used Cfengine on Linux for some time, but has limited success uisng cfengine on Mac OS X. Also, Cfengine was in a transition state between version 2 and version 3 at the time of our implementation.

Cfengine can be used to bring a machine to a known state and keep it in that state.

[4] http://www.quest.com/quest-management-xtensions-for-sms
[5] http://www.puppetlabs.com
[6] http://www.cfengine.org

LANRev

LANRev[7] is a COTS tools for management of Windows and Mac OS computers. [8] LANRev was considered the cream of the crop Mac management tool at the time, but its lack of RHEL support took it out of the running for us.

FileWave

FileWave is a COTS tools for management of Windows, Mac OS, and RHEL computers. [9] FileWave is a very mature product, but its asset inventory program did not run on RHEL, which took it out of the running for us.

Casper Suite

Casper Suite is a COTS tools for management of Mac OS computers. [10] A very robust suite of management tools, Casper was only for Mac and thus took it out of the running for us.

Radmind

Radmind is an open source project managed by the University of Michigan. [11] Radmind is a very interesting CLI tool for Mac OS X and other flavors of UNIX and Linux. Radmind, it seemed to us, is a great tool to keep a computer at a known state. We needed inventory, flexibility, and the ability to run ad-hoc scripts on remote machines. We didn't feel radmind met this criteria.

<u>Our selection: KBOX, no wait, Puppet</u>

We originally recommended and received approval to purchase the KBOX solution, but by the time we got our purchase request ready, management had to redirect funds to other higher priority projects. KBOX met all our needs for an out of the box, relatively easy solution to inventory machines and eventually patch them.

Once funding was withdrawn for KBOX we submitted a "no capital outlay" proposal to implement Puppet. We proposed to deploy a Puppet infrastructure that provided

[7] Now called Absolute Manage
[8] http://www.lanrev.com
[9] http://www.filewave.com
[10] http://www.jamfsoftware.com
[11] http://rsug.itd.umich.edu/software/radmind

basic inventory, including datacalls, of all *NIX based non-Windows computers at the Lab. With research and training, the infrastructure could be used for more sophisticated tasks such as patch deployment.

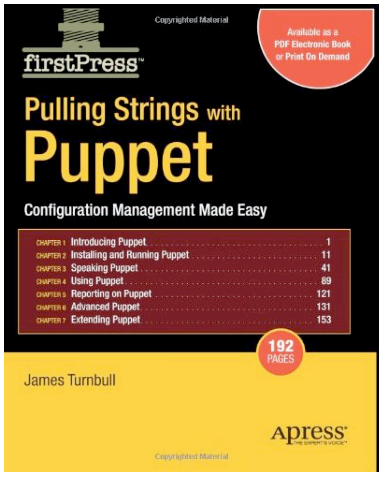
We chose Puppet for the following reasons:

- Inventory in an SQL database right out of the box
- No capital outlay
- Modern, well supported product
- Active open source community
- Server can run on Mac OS X Server

Getting Started

Getting started with Puppet was an exercise is Web research, although classes are offered. Because Puppet is open source, there is no single large company behind puppet producing professional tutorials, manuals, and set-up guides, although Puppet Labs does coordinate a very comprehensive wiki.

The puppet wiki[12] that Puppet Labs operates to help document puppet has much good material. Introductory pages, installation pages, configuration pages all helped up get started. We found an iterative, baby step approach to be very useful as we were able to better isolated issues as they came up, and we learned more about the product.


Copyrighted Material
Available as a PDF Electronic Book or Print On Demand
firstPress
Pulling Strings with Puppet
Configuration Management Made Easy
CHAPTER 1 Introducing Puppet 1
CHAPTER 2 Installing and Running Puppet 11
CHAPTER 3 Speaking Puppet 41
CHAPTER 4 Using Puppet 89
CHAPTER 5 Reporting on Puppet 121
CHAPTER 6 Advanced Puppet 131
CHAPTER 7 Extending Puppet 153
192 PAGES
James Turnbull
Apress
Copyrighted Material

We also purchased *Pulling Strings with Puppet: Configuration Management Made Easy* by James Turnbull, an active Puppet developer. The book is a bit dated by now (one to two major versions back), but it's still very helpful.

Our goal getting started was to set up the Puppet server, have clients connect to it, and have the client report the default puppet "fact" inventory to a central MySQL database. We were successful in under a day.

[12] http://docs.puppetlabs.com

Basic Implementation

We chose to use an Intel Apple Xserve as the puppet server. Our server runs Mac OS X 10.5 server, although we are in the process of testing on Mac OS X 10.6 server. This paper will cover the 10.6 configuration we are currently testing.

This paper won't be a detailed step by step guide, but we will cover all the steps necessary to get started. We'll get as detailed as we can, although time constraints writing this paper may require us to assign research tasks to the reader.

The basic general steps are as follows. Only the items in bold will be discussed in this paper. The other items show how we configured our server, but may be done differently by other organizations.

- **Install and update Mac OS X Server**
- Configure the server for networking
- Assign a host name for the server in DNS
- **Install the Xcode developer tools**
- Bind the server to Active Directory (or organizational directory seervice)
- In Workgroup Manager, add or configure accounts for the admins and developers
- In System Preferences: Sharing, allow the admins and developers to ssh into the server
- **Create a puppet user and group**
- **Install Facter (a part of puppet) and Puppet**
- **Configure Puppet**
- **Test Puppet**
- **Compile needed MySQL Libraries**
- **Configure MySQL**
- **Test Puppet Again**

Items not covered in this paper, but should be done as needed, include setting up Mac for networking, binding the Mac to a directory service, setting up administrative and developer accounts.

After setting up and testing the basic set up, we added a Ruby application server called Phusion Passenger to deal with server load. Out of the box the puppet server process (puppetmasterd) can only handle one client at a time. Passenger Fusion allows the puppet server to handle multiple clients simultaneously.

Install and update Mac OS X Server

A pretty basic step. You don't need a very powerful server for testing (we use a VMWare Fusion virtual machine on one of our developer's MacPro), but you will want a powerful machine for production. We use an 8 core XServe (pre Nehelam) with 8 GB of RAM. This server acts as the puppetmasterd server, MySQL server, and

Apace server. For basic inventory and light configuration management, we see no load issues with 1500 clients.

So, the first step is to install and update Mac OS X server. Don't turn any services on just yet, just install and update. Mac OS X client can be used, but the if using Apache and MySQL, they may need to be installed and configured. Since Mac OS X Server came for free with the Xserve, we chose to use it.

Install the Xcode developer tools

Straight forward task. Xcode is a free download from Apple.[13]

Create a puppet user and group

The puppermasterd daemon needs to run as a user, so create a puppet user and group. The documentation says to run "/usr/sbin/puppetmasterd --mkusers", but this command doesn't work under Mac OS X. Simply use the System Preferences Accounts pane to create a standard user and group, each called _puppet. The standard practice for a service account like this is to prepend "_" to the user and group name. The password for the _puppet account can be *, which will lock the account. Place the puppet user in the puppet group. To remove the puppet user from lists of users and groups in certain Mac OS X lists, change the ID of the user (and its primary group) and the group ID to 300. Use the Advanced Options in Accounts to validate the group name and ID, then the puppet user ID and group are correct.

Install Facter (a part of puppet) and Puppet

Facter is the inventory part of Puppet, and is required for Puppet operation. Even if not storing the inventory in a database, Facter is required. Facter will report basic hardware and OS inventory. Each item reported is called a Fact. Custom Facts can be programmed to expand the inventory. Fact can be used in Puppet to in conditional tests and in other ways. As Yoda might say, "Very useful, Facts are."

There are two ways to install Facter and Puppet. One is to get the source[14] and compile yourselves. There is even a script in the source that will compile and create a Mac installer package. Looking at the packaging script (conf/osx/createpackage.sh in the source code) is a good exercise for the inquisitive Mac admin, but it's not required since a ready to install Mac package is maintain by Google[15]. Google uses Puppet to manage its Macs.

Simply download the Facter and Puppet packages. Install Facter first, then Puppet.

[13] http://developer.apple.com/mac
[14] http://www.puppetlabs.com/misc/download-options
[15] https://sites.google.com/a/explanatorygap.net/puppet

Note: Version 0.25.4 will not work correctly on Macs. Install on older or more recent version.

Configure Puppet

The final step before actually testing puppet on the client and server is to configure the puppet server. Puppet Labs has a configuration guide[16], but some modifications are needed when running puppetmasterd on a Mac server.

Puppet is controlled by an configuration file located in /etc/puppet. Edit this file and add the following lines to the top to tell puppet to use the user and group defined earlier.

```
sudo mkdir /etc/puppet
sudo echo "user = _puppet" > /etc/puppet/puppet.conf
sudo echo "group = _puppet" >> /etc/puppet/puppet.conf
```

The last item to configure is the site manifest. mkdir /etc/puppet/manifests, and then create /etc/puppet/manifest/site.pp

```
# site.pp
file { "/etc/sudoers":
    owner => root, group => wheel, mode => 440
}
```

Run puppetmasterd in debug mode to see what it is doing. Use control-C to exit

```
/usr/sbin/puppetmasterd --no-daemonize -d
```

Test Puppet

Testing puppet is documented on the Puppet wiki, but we will outline the steps

- On the server run the puppetmasterd daemon (see Configure Puppet above)
```
sudo puppetmasterd --no-daemonize -d
```
- Install Facter and Puppet on the client
- Run facter to validate it's working
- Run puppetd as:
```
sudo puppetd -o --no-daemonize -v -d \
--server=<your puppet server FQDN>
```
- On the server, control-c
- On the server run:
```
sudo puppetca --sign --all
sudo puppetmasterd --no-daemonize -d
```
- Now run the same puppetd command as above (just up arrow on the client) on the client. It should work.

At this point we now have a working Puppet server and client! Time to add the inventory database.

[16] http://docs.puppetlabs.com/guides/configuring.html

Compile needed MySQL Libraries

Since we want Puppet to report inventory data to a MySQL database, the underlying Ruby code of Puppet needs some MySQL libraries that are not installed on Mac OS X Server by default. Apple may not provide the binaries for these libraries, but Apple does provide a configured download of the source for the MySQL used by Mac OS X. This step is only necessary if using puppetmasterd on a Mac OS X server and if storing the inventory data to MySQL.

The libraries are only used for the Ruby code. If the Apple included MySQL server is used, the Apple maintain code is used for the server, not the code downloaded through this process.

Either way, source or binaries, Apple posts all to their open source web site.[17] Get the MySQL from the Apple site and note where they get installed. Some suggestions that we used are below.

If compiling from source

Given a choice, use the binaries from Apple, as that process is much faster.

Get the source code from Apple and expand the archives. Execute the following commands to compile the code and move all the files to a useable location. Change the <MySQL expanded archive path> as necessary.

```
sudo -i
cd <MySQL expanded archive path>
# something like: /Users/admin/Downloads/MySQL-53
rm -rf /tmp/MySQL.root
env RC_ARCHS="i386 x86_64" \
RC_CFLAGS="-arch i386 -arch x86_64 -pipe" \
RC_NONARCH_CFLAGS="-pipe"  \
SRCROOT=<MySQL expanded archive path>  \
OBJROOT=/tmp/MySQL.root/MySQL~obj  \
SYMROOT=/tmp/MySQL.root/MySQL~sym  \
DSTROOT=/tmp/MySQL.root/MySQL~dst  \
make install

# now move the binaries to a useful location
mkdir -p /usr/local/mysql
cp -R /tmp/MySQL.root/MySQL~dst/usr/* /usr/local/mysql
exit
```

If using the Apple binaries

Get the binaries from Apple, and expand the archives. You will see a usr folder. Execute the following commands to move all the files to a useable location. Change the <MySQL expanded binaries path> as necessary.

```
sudo mkdir -p /usr/local/mysql
cp -R /<MySQL expanded binaries path>/usr/* \
```

[17] http://www.opensource.apple.com/darwinsource/Current

```
/usr/local/mysql
```

Install Ruby GEMS

Perl has CPAN. PHP has PEAR. Red Hat has YUM. Ruby has GEMS. Gems allows for easy installation of Ruby related code. We need to install the needed GEMS for MySQL so Puppet can report inventory data to the MySQL database. Assuming the MySQL binaries are located in /usr/local/mysql:

```
# if an http proxy is used on the network
export http_proxy=http://proxy.company.com:8080
env ARCHFLAGS="-arch x86_64" gem install mysql \
-- --with-mysql-config=/usr/local/mysql/bin/mysql_config
```

Configure MySQL

The MySQL database is used to store configuration information from the clients.[18] One of the advantages of Puppet is that configuration database is really easy to set up, and is very valuable. On a Mac server, other than the MySQL binaries discussed above and the Ruby GEM, there is no additional configuration needed to use the database.

Complete configuration and management of a MySQL database is beyond the scope of this paper, but we will go through the basics of using the MySQL database included with Mac OS X Server. Here are the basic steps:

- In Server Admin add the MySQL service
- In the MySQL service, change the root password for the MySQL service. Leaving it blank is a huge security risk
- Start the MySQL service

That's it; the MySQL service is now running on the server. We still need to create a database and database user for puppetmasterd.

- From the terminal run: mysql -u root -p
- From within mysql run:

```
create database puppet;
grant all privileges on puppet.* to puppet@localhost identified by
'StrongPasswordHere';
```

A database and user have now been created. The final step is to tell puppetmasterd to use the database. Edit the /etc/puppet/puppet.conf file and add the following section:

```
[puppetmasterd]
  storeconfigs = true
  dbadapter = mysql
  dbuser = puppet
  dbpassword = StrongPasswordHere
  dbserver = localhost
  dbsocket = /var/mysql/mysql.sock
```

[18] http://projects.puppetlabs.com/projects/puppet/wiki/Using_Stored_Configuration

Note the database password is in the puppet.conf file in clear text, so the file and server needs to be protected accordingly. By default Mac OS X Server doen't allow network access to the database, so that's a good failsafe, but it's not complete security.

Test the Puppet Client Again

Running the same puppet test as above in the Test Puppet section, we can browse the MySQL database and it will show that the client information (the Facts from the client) are now stored in the MySQL database.[19] Note the *hosts*, *fact_names*, and *fact_values* tables.

[19] There are a number of good tools to browse a MySQL database. One favorite free Mac tool is http://www.sequelpro.com/

Configuration Management

Installing Packages

We chose Puppet because of inventory management and configuration management. For LANL, installing packages is central to our configuration management.

The bulk of our configuration management is done by an in house security tool we call Security Tool On Mac (STOM). STOM is a Perl script wrapped in an AppleScript Studio application. We package it up and push it to all clients via Puppet. In a recursive twist, we also use STOM to install the puppet client! So the sequence of events is 1) the user installs and runs STOM; 2) STOM installs and configures Puppet; 3) Puppet maintains the version of STOM. We use STOM to install Puppet for a number of reasons. First, STOM is installed on all machines, even stand alone machines, so our users are used to installing STOM. Second, STOM was installed on hundreds of Macs at LANL before we started using Puppet. We had coded in some basic self-updating code into STOM, so when we started to roll out Puppet, we had STOM install it. This allowed us to get Puppet on hundreds of machines without needing to resort to sneakernet.

Puppet has the ability to install pkg or mpkg packages. When Puppet installs such a package it leaves a receipt in /var/db so it knows not to install that package (and version) again. Since the Mac has no native package management capability like RPM, Puppet can't dynamically reinstall a package if it is removed. This is a bit of a short coming, but can be handled with a receipt/installation check written into the Puppet class. We haven't implemented this check yet, but we are thinking about it.

Our goal is to use Puppet to install base version of all our core software, then use Puppet to push package updates and keep the core software up to date.

Apple Software Update

One of the other major purposes of configuration management is to keep the OS and Apple supplied software up to date. Apple provides a command line tool (softwareupdate) to perform updates, but if the update requires a restart, life gets complicated. We also needed a way for non-admins to keep the Mac patched.

To deal with this issue we developed a script that puppet installs and maintains. Puppet also install a launchd daemon that runs at a random time (so as not to overwhelm our update server). The launchd job run as root and executes our script. Our script then checks softwareupdate for updates, and if there are any, displays a dialog to the user. The user is then given an opportunity to save and quit all running applications, then presses an install button that opens the GUI Software Update. Since the GUI Software Update program is being run as root, an administrative password is not required and the user can install all the Apple updates.

Advanced Implementation

Autosign

By default puppet requires a key exchange between the server and each client.[20] In an open internet environment this is desirable to prevent unauthorized computers from connecting to the puppet server and having potentially private or commercial software or configurations downloaded to the unauthorized client. When the puppet client first connects to a server, the client uploads its certificate request to the server, and the server must sign the request before it will begin to mange the client. The signing process can be manual, using the puppetca command, or automatic.

In a small managed environment, manually signing the certificate requests may be manageable. When the number of clients is too high to easily manage manual signing, puppetmasterd offers the autosign option.

The Autosign feature is turned on in the puppet.conf file on the server. Autosign then simply checks the clients certname[21] name against entries in the /etc/puppet/autosign.conf file, and if there is a match, the certificate request is automatically signed when the client connects for the first time.

The /etc/puppet/autosign.conf file can be as simply as:
```
hostname.domain.com
*.secure.domain.com
```

One caveat we learned is that if the client needs to request a new signed certificate, autosign will fail since the certificate request has already been signed for that client. Puppet handles this by using the puppetca tool with the --clean option. This become very relevant when dealing with a large number of clients. See the Lessons Learned section for how we handled this.

Environments

Puppet supports multiple environments, which is essentially a means of applying different configurations.[22] An environment can be applied to a client by a non-volatile setting in the client's puppet.conf file, or by using the --environment command line switch. Environments can be used in a development/production paradigm, or for different levels of support.

[20] For a complete discussion of certificates and Puppet security, see http://projects.puppetlabs.com/projects/puppet/wiki/Certificates_And_Security
[21] Certname by default is the client hostname, but can be set manually on the client to something else. There are advantages to managing certname. See the Lessons Learned section of this paper for more information.
[22] http://projects.puppetlabs.com/projects/puppet/wiki/Using_Multiple_Environments

Custom Facts

Custom Facts offer a way to gather inventory information from clients.[23] A Fact is simply a ruby script that returns some small value. LANL has a number of custom facts. Facts can also be used by Puppet in the configuration classes, and are primarily used to determine is something should or should not be done to a client. for example, one built in Fact is operatingsystem. In a Puppet class one can add a test for operatingsystem and have one configuration applied if the result is "darwin" and a different configuration applied is the result is "centos" .

Note: neither Fact names nor the values can contain spaces.

One custom Fact we find very useful is property number. We use the nvram command to add a custom non-volatile variable containing our internal LANL property number to every Mac. We then use a custom Fact to retrieve this property number. This allow us cross reference the records in our Puppet database with those in our property database. Since the Mac doesn't support DMI and a BIOD asset_id tag, this is the next best thing.

The custom Fact is fairly easy to implement, even if you don't know Ruby. The custom Fact for our PropertyNumber is simply a file called lanl_property_number:

```
# lanl_property_number.rb
Facter.add("lanl_property_number") do
    confine :kernel => :darwin
         setcode do
                 %x{nvram asset_id | awk '{print $2}'}.chomp
         end
end
```

The file is called the same name as the custom Fact. The code only runs if the kernel is Darwin. The Ruby code simply executes a one line bash command that returns the asset_id nvram variable.

Here's the great part: *custom Facts are automatically added to the Puppet database*! No additional work is required. Just create the Fact, deploy the file, and after a few hours all the data is collected. One note, the fact is run every time puppet runs (every 30 minutes, by default). Use a custom Fact for a data call, for example, would require some careful planning. The use of a temp file as a semaphore, for example, could solve the issue.

Reporting

Puppet supports very detailed reporting[24], although LANL is not using this feature. When the puppetd client runs on a node, it can be set to report back to puppet all sorts of information about the run. From the Puppet Web site: "Each client-side

[23] http://projects.puppetlabs.com/projects/puppet/wiki/Adding_Facts
[24] http://projects.puppetlabs.com/projects/puppet/wiki/Puppet_Reporting

transaction generates a transaction report containing metrics and log messages." These report can be used for gathering metrics on Puppet itself. Eventually Puppet Labs would like to add more information into the reports to make them more useful.

External Node Classifiers

Technically, Puppet refers to a client as a "node". This makes sense since Puppet can manage servers as well as clients. In fact, Puppet was designed to manage servers.

Puppet allows for different classes to be applied to different nodes based on a number of factors. Facts can be tested, nodes can be identified directly, or an external database can be used to store nodes and what classes to apply to them. This last option is called External Node Classifiers.[25] There are some very technical reasons to use External Node Classifiers, but the best one is the ability to manage nodes from a database. While a database isn't technically required--Puppet will call a script to get node information--the use of a database adds much value.

An External Node Classifier database could be used to dynamically assign classes to be applied to nodes. This could allow a field organization to install software or configure a machine simply by makes some changes in a Web database. We are very excited about implementing External Node Classifiers, but it is not a simple task. Technically is very easy to implement, but realistically the structure of the External Node Classifier database can be complicated and requires great forethought.

Puppet Dashboard and Foreman

Pupept Labs offers a tool called Dashboard.[26] Dashboard show a snapshot of the Puppet installation, and allow for some manipulation of nodes. The tool shows promise, but as of April 2010, it still more of a technology preview than a useful tool.

Foreman is also a reporting and node management tool.[27] Foreman is similar to Dashboard in that it shows a current snapshot of the Puppet installation. It also allows for some manipulation of nodes through a Web interface. Like Dashboard, it's rather immature but shows great promise.

Phusion Passenger

As stated earlier, out of the box the puppet server process (puppetmasterd) can only handle one client at a time. Phusion Passenger[28] allows the puppet server to handle

[25] http://projects.puppetlabs.com/projects/puppet/wiki/External_Nodes
[26] http://www.puppetlabs.com/puppet/related-projects/dashboard/
[27] http://theforeman.org/wiki/foreman/Puppet_Reports
[28] http://www.modrails.com

multiple clients simultaneously. Installing Passenger is beyond the scope of this document, but Puppet Labs does offer some assistance on the Web.[29]

[29] http://projects.puppetlabs.com/projects/puppet/wiki/Using_Passenger

How LANL uses Puppet

Machine Inventory

One of the primary reasons we chose Pupept was the ease of getting an inventory database up and populated. Puppet provides out of the box inventory for OS, and hardware. We add a few addition inventory items (see Custom Facts below). Through creative SQL queries and de-normalization, we gain good insight into our Mac population. One example is s query that can tells us how many machines we have running 10.4, and how many are capable of running 10.5 or 10.6.

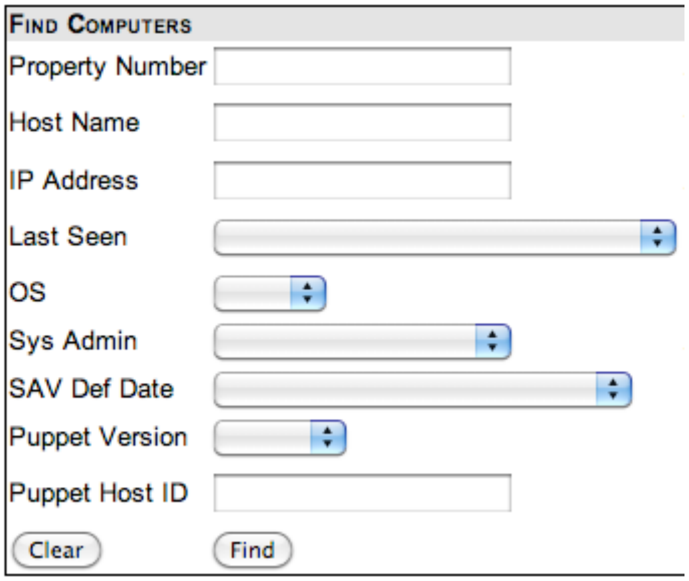
Custom Facts

As stated earlier, a Fact is a unit of information about a client. A Fact might be the OS version, or the number of processors. A custom Fact is simply a Ruby script run on the client that returns a value.

LANL has implemented a number of custom Facts. We are able to get the version of SAV installed, the SAV virus definition file date, the version if STOM (LANL's Mac hardening tool), the status of SAV autoprotect, the Macs LANL property number, and a couple of other items. We plan on using custom Facts to report the results of data calls.

Web Front End to Inventory

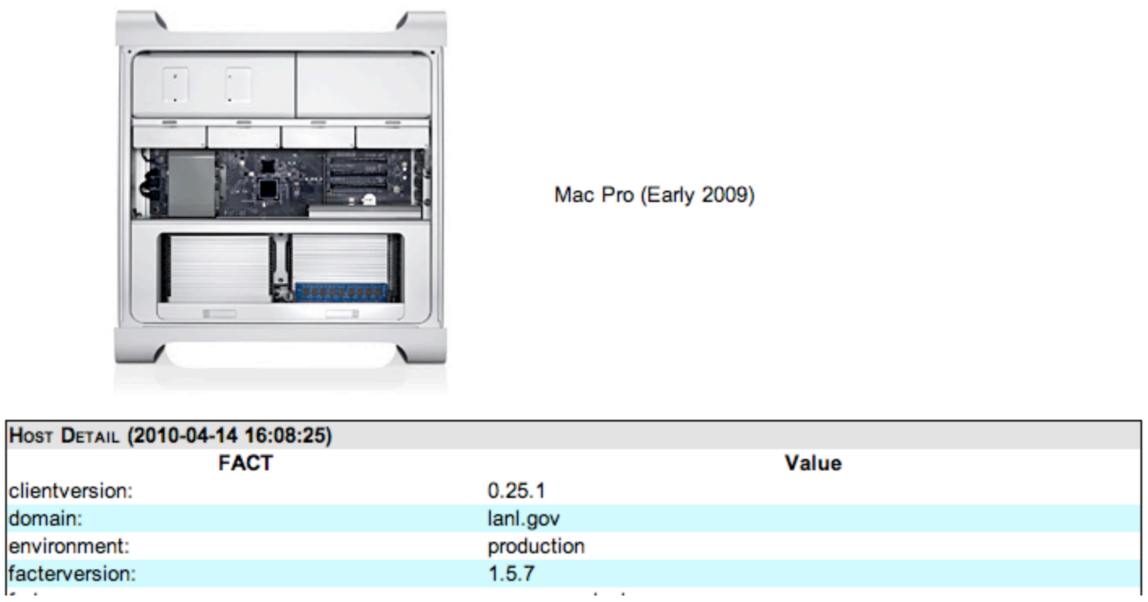
The inventory database is a fairly simple MySQL database. Any tools that access MySQL can be used to build a reporting front end. LANL chose to use PHP. Our Web site allows our central services folks, help desk, and field technicians to report any number of fields.



Find Computers

Property Number	
Host Name	
IP Address	
Last Seen	
OS	
Sys Admin	
SAV Def Date	
Puppet Version	
Puppet Host ID	

The results are shown in list format and the user can click on a client to see all of its Facts and software inventory. We wrote a separate tools that collects software inventory and reports into a database on the Puppet server. We also have a routine that checks the Apple warranty date and reports how much is left on the Apple warranty. Te next page is a sample of the information in our Web site. We also have a number of repots on the Web site to help our field teams manage their Macs.

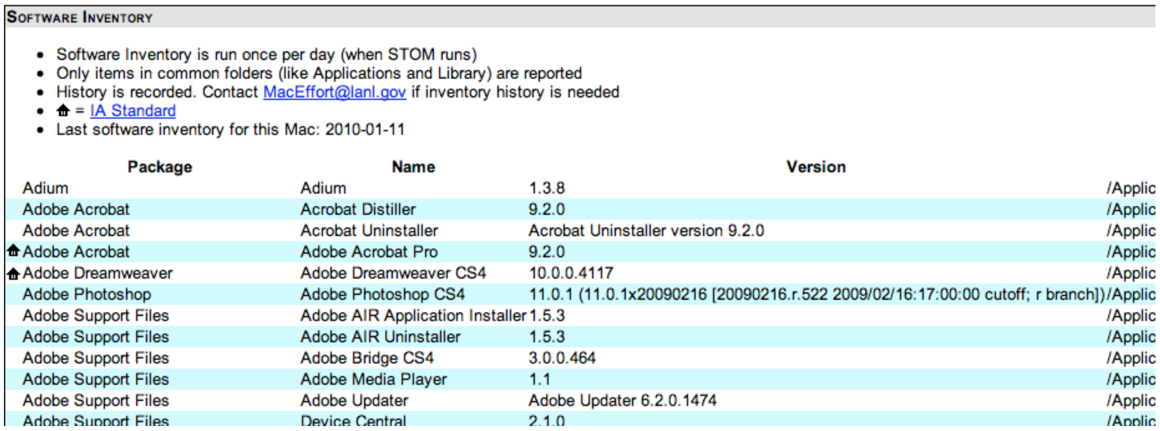


Mac Pro (Early 2009)

Host Detail (2010-04-14 16:08:25)

FACT	Value
clientversion:	0.25.1
domain:	lanl.gov
environment:	production
facterversion:	1.5.7

Shows a picture of the machine, the it lists all the Facts.

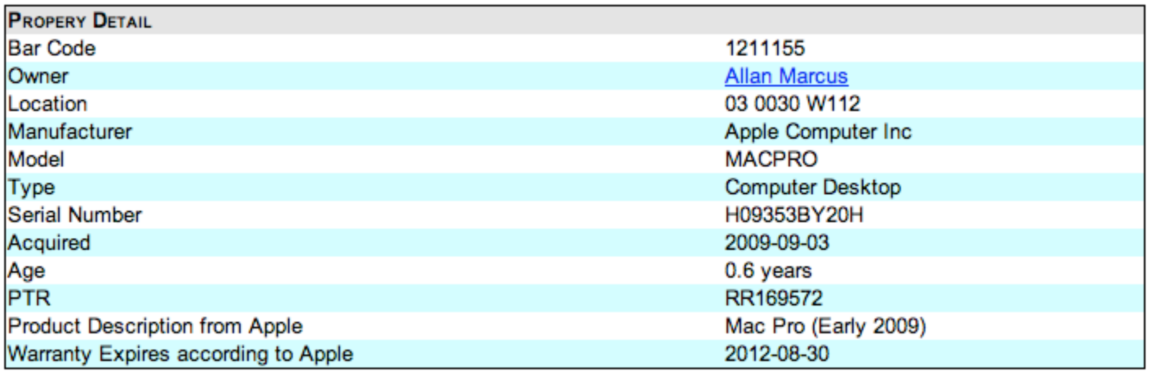


SOFTWARE INVENTORY

- Software Inventory is run once per day (when STOM runs)
- Only items in common folders (like Applications and Library) are reported
- History is recorded. Contact MacEffort@lanl.gov if inventory history is needed
- 🏠 = IA Standard
- Last software inventory for this Mac: 2010-01-11

Package	Name	Version	
Adium	Adium	1.3.8	/Applic
Adobe Acrobat	Acrobat Distiller	9.2.0	/Applic
Adobe Acrobat	Acrobat Uninstaller	Acrobat Uninstaller version 9.2.0	/Applic
🏠 Adobe Acrobat	Adobe Acrobat Pro	9.2.0	/Applic
🏠 Adobe Dreamweaver	Adobe Dreamweaver CS4	10.0.0.4117	/Applic
Adobe Photoshop	Adobe Photoshop CS4	11.0.1 (11.0.1x20090216 [20090216.r.522 2009/02/16:17:00:00 cutoff; r branch])	/Applic
Adobe Support Files	Adobe AIR Application Installer	1.5.3	/Applic
Adobe Support Files	Adobe AIR Uninstaller	1.5.3	/Applic
Adobe Support Files	Adobe Bridge CS4	3.0.0.464	/Applic
Adobe Support Files	Adobe Media Player	1.1	/Applic
Adobe Support Files	Adobe Updater	Adobe Updater 6.2.0.1474	/Applic
Adobe Support Files	Device Central	2.1.0	/Applic

After the Facts we show the software inventory for the machine.



PROPERY DETAIL	
Bar Code	1211155
Owner	Allan Marcus
Location	03 0030 W112
Manufacturer	Apple Computer Inc
Model	MACPRO
Type	Computer Desktop
Serial Number	H09353BY20H
Acquired	2009-09-03
Age	0.6 years
PTR	RR169572
Product Description from Apple	Mac Pro (Early 2009)
Warranty Expires according to Apple	2012-08-30

After the software inventory, we show the basic property information.

STOM Distribution

One of the first things we did beyond Fact inventory was to use Puppet to install and maintain STOM. In a circular twist, we also use STOM to install and configure Puppet! Our sequence is to install and run STOM on a Mac. STOM then downloads the latest Puppet from our internal server, installs it, copies out puppet.conf file, and sets up a launchd job to run puppet hourly. We then us Puppet to keep the STOM applications as well as the puppet client, up to date.

SAV Distribution

We also use Puppet to install and keep SAV up to date. The administrative console for managing SAV that Symantec provides did not meet our needs, so we needed a way to install new versions of SAV, as well as ensure SAV is installed and running properly.

Software Inventory

We wrote out one simple inventory program using PHP and the Apple provide system_profiler command. system_profiler reports on all globally installed software. Our PHP script parses this list, and sends the results to our central server. We use Puppet to install the script, and set up and maintain a launchd job to run the script daily.

Apple Software Update

LANL operates an internal Apple Software Update Service to distribute Apple software updates within LANL. We needed a way for clients to connect and check the server, and then allow patches to be installed without requiring an admin password, but not forcing a reboot at arbitrary times.

We solve this challenge with an AppleScript application that is installed by Puppet. Puppet also maintains a launchd job that launches the AppleScript daily. Since the AppleScript is launched by puppet, it's run with root privileges. The AppleScript allows the user to cancel (thus will be prompted again the next day), or to run the Apple Installer and install the patches.

Mac Imaging

LANL has a standard set of software we install on all Macs. We developed a puppet environment that allows us to install all this core software on a newly imaged Mac with one command line. The advantage of using Puppet to install all the core software is many fold.

- Updating one package does not require us to re-make an image.
- When a new update is available, we only need to update the package on the puppet server.
- We don't do this yet, but once puppet has installed the packages, maintain them through puppet is not that hard. The reason we hesitate is that some upgrades can change functionality, and we may have users that rely on the old functionality.

Lessons Learned

Environments

LANL uses environments for both a separation between development and production, as well as to allow different support groups provide their own configurations to their computers.

Directory Structure

Puppet has a default directory structure that works well for a single puppet developer, or for a team of puppet developers that have the same access rights on the server, but care and forethought ware required to develop a directory structure and ACLs to allow for multiple developers across multiple teams to work on multiple environments on the same puppet server.

We chose use /Groups on our Mac OS X server to store all our puppet code. We have all the production code in /Groups/Production/<modules> and a similar path for development. We use Server Admin to set the ACLs for these folder appropriately. We use ACL's and not posix permissions since ACLs can me have to inherit.

Contributing to the Puppet project through GIT

I have to admin that I haven't gone through this process, but my colleague, Roy Nielsen has. Items to keep in mind when contributing to the puppet project include code release restrictions your company may impose, understanding GIT[30], understanding the puppet development lifecycle[31], and joining the puppet developers mailing list[32]. I would say the process wasn't trivial, but once figured out, it's a workable process.

Setting up puppetmasterd on a Mac server

For the most part, using a Mac server has not been an issue. Having puppetmasterd autostart requires the setup of a launchdaemon, which is different from other UNIX systems. Another issue we ran into was having to manually create the puppet user and group for puppetmasterd. and finally, dealing with the MySQL binaries was also a bit of a chore until we got it all documented.

[30] GIT is source code management system similar to SVN or CVS.
[31] http://projects.puppetlabs.com/projects/puppet/wiki/Development_Development_Lifecycle
[32] http://groups.google.com/group/puppet-dev?pli=1

Client ID

Puppet uniquely identifies a client, or node, by an identifier. This identifier is used as the name of the certificate for the node, so puppet calls it certname. By default, the certname is the FQDN of the node. This system probably works fine if a static set of servers are being managed, but in LANLs dynamic environment with 2000-3000 Macs, host names are an unreliable unique identifier. Moreover, we intend on implementing a history database, and host names are not a good identifier for computers over time. Finally, many of our Macs have more than one host name, especially those laptops that move between multiple locations.

To solve this challenge we decided to use LANL property number as the certname. Since we store the property number in vnram, getting it is easy via the command line. In the even the propery number is not available, we fall back on the serial number, then hostname, then the ethernet address.

The certname can be stored in the puppet.conf file, or can be passed on the command line for puppetd. We chose to pass it on the command line. We developed a puppetd.sh script we use to launch the puppet client that figures out what the certname should be. We also allow for a "-v" argument to the script to run puppetd in debug mode to help our field techs diagnose issues. The script also deals with the issue of cleaning a client certificate from the server (discussed later). The script stored the certname in nvram so it can be quickly retrieved.

The bottom line here is that if host name is a good identifier for your purposes, then puppetd handles that out of the box. For use, host names change too often for it to be useful to us, so we have to come up with a different identifier and way to implement it.

Job Launch Time Randomization

Puppetd will run every 30 minutes. If started from a well designed launchd job and the keep alive flag[33] is used, thre should be sufficient randomization in launch time so the server is not hit too hard at any one time.

We wanted to be sure puppetd was running, so we do not rely on puppetd in daemon mode. We use a lunchd job to run our puppetd.sh script to launch puppetd every hour. We didn't want the job running at the same time across all our machines, so we set the hourly launch time to a random number between 0 and 60.

[33] See: man launchd.plist

Purging Machines

Puppet has no mechanism to auto-remove machines that haven't checked in to the puppet server in a while. We are in the process of writing a series of scripts to notify field teams of the age of their puppet clients, and then to remove clients not seen in 90 days or more.

MySQL Binaries

As discussed in the Basic Implementation section earlier, when using Mac OS X as the server and using the storeconfig inventory database with MySQL, some MySQL binaries are needed.

Ruby GEMS

Not much a lesson, but I had not worked with Ruby GEMS before. Right now I believe Puppet can use the latest version of all the required GEMS, but for a long time certain GEMS had to be at specific versions, not all of which were the latest. Reading the "gem" man page was needed, and then researching whcih GEMS needs to be installed.

Puppetd run job

Mac OS X can use cron, but it's considered deprecated on the Mac. The way more cool Mac Way is to use launchd. We use launchd daemons for both the puppetmasterd server (whenever the server reboots) and for the clients (once per hour.

Cleaning old client certificates

Managing client certificates on the puppetpasterd server can be a very labor intensive task. With a static number of known machines, the task might be manageable, but with thousands of machines, dozens being added or removed weekly, the task is one ripe for automation.

The issue concerns how the puppet client and server communicate. When the puppetmasterd daemon is first run, it creates a certificate for the server. When the client is first run, it creates a certificate for the client. When a client connects to the server, the client gets the server's certificate and the client requests that its certificate be signed by the server. At that point the client knows about the server and the server knows about the client, and communication can begin.

The signing of the client request can be handled manually using puppetca, or can be automatically handled using the autosign feature discussed earlier. If the network or server is not protected by a firewall, a rouge machine might request configuration information from the server and thus receive security significant or intellectually protected files. For this reason, autosign is discouraged. We use it at LANL since our entire network is protected by a perimeter firewall. We also limit our signing to *.lanl.gov. The issue arises when a machine uses a new certificate. This happens if the certificate is deleted from the client (say through rebuilding the machine). If the client attempts to communicate with the server using a new certificate, the serer will see this as different as the one on file and reject communications with the client.

To ameliorate this situation, our puppetd.sh launch script checks to see if the error message from the server indicates the certificate is invalid. If this is detected, the client requests the server clean the certificate from the server. The next time the client runs, the new certificate can be accepted by the server.

We also learned that the _www user needs to have sudo rights to run the puppetca command. We edited sudoers to allow this. See Appendix 2 for how and the script.

One would not want to take this approach on a network directly connected tot he internet, but in a controlled environment, we feel this automation is safe.

MySQL Socket

When running the included MySQL server from Mac OS X Server, the default MySQL socket is located at /var/mysql/mysql.sock. This is not standard, so it's a good idea to note this. The socket path is displayed in the MySQL log in Server Admin when the service is started.

Closing Thoughts

There are no conclusions to be drawn from this paper other than Puppet is a viable option for managing Macs. LANL manages about 2000 Macs now, and we expect to grow to about 3500.

A Mac server is not required, but for Mac folks, the familiarity of the OS is appreciated. Mac OS X client, UNIX, or Linux can easily be used for the server.

If given the opportunity, I would recommend taking a puppet class from Puppet Labs. Although not required, the class offered a boot camp approach to getting up to speed.

About the Author

I am a Solutions Architect at the Los Alamos National Laboratory. I graduated from UC Berkeley in 1986 with a B.S. in Political Economy of National Resources. I worked at Apple Computer, Inc. as a database application developer from 1987 until 1993. While at Apple I earned my M.B.A. from a joint Apple / San Jose State program.

In 1993 I came to Los Alamos, NM to work at the National Lab as a database application developer. For about ten years I developed applications using 4D, Omnis 7, Oracle, and eventually Lotus Notes. In 2002 I made the switch to systems administration. I managed LANL's SourceForge Enterprise Edition servers and services (using Red Hat Linux) for a few years, then I became LANL's lead Mac person.

In 2005 I published a paper on how to use Mac for diskless booting.[34]

In 2008 I lead the Center for Internet Security's Mac benchmark team. The team produced the Mac OS X Leopard Level 1 and Level 2 Benchmark[35], which is widely used to harden and configure Mac across government, industry, and education. I also lead the Snow Leopard committee, although work is progressing slowing.

My current job is to research and write LANL's Configuration and Security Guidelines for Mac OS X. I also developed and maintain the tool we use to harden and configure Macs. With Roy Nielsen, I manage LANL's Mac configuration management servers. I also help develop LANL's compliance and cyber security policies.

[34] http://www.afp548.com/article.php?story=20050520183831903
[35] http://cisecurity.org/en-us/?route=downloads.benchmarks

Appendix 1: puppet.sh

Please note that line wrapping in MS Word may have added returns to this code.

```
#!/bin/sh

# this script is run from a launchd job

# use -v for verbos output
# mainly for debugging

# must be run as root
WHOAMI=`whoami`
if [ "$WHOAMI" != "root" ]; then
    echo "The puppet.sh script (part of STOM) must be run as root. You are
$WHOAMI"
    exit 0
fi

# need to make sure /usr/sbin is in the path, or puppet and facter will not
run
export PATH=$PATH:/usr/sbin

# avoid unsightly errors in the logs
export TERM_PROGRAM=Apple_Terminal
export TERM=xterm-color

# this suffix is added to the value to make it look like
# a FQDN. This allows for auto sign to work on the server
SUFFIX=example.gov
if [ "$1" = "-v" ]; then echo "SUFFIX set to $SUFFIX "; fi

# this is the server to sent a puppetca clean to
SERVER=puppet.example.gov
if [ "$1" = "-v" ]; then echo "SERVER set to $SERVER "; fi

# see if the MAC_UID is in nvram already
MAC_UID=`/usr/sbin/nvram MAC_UID 2>/dev/null | awk '{print $2}'`
if [ -z "$MAC_UID" ]; then
    # flag that nothing is in nvram yet
    NVRAM="no"
    if [ "$1" = "-v" ]; then echo "MAC_UID not found in nvram"; fi
fi

# get the serial number for this Mac
if [ -z "$MAC_UID" ]; then
    MAC_UID=`facter | grep sp_serial_number | awk '{print $3}' | sed
's/[\/\~\!\@\#\$\%\^\&\*\(\)\+\=]/_/g'`

    # test to see if the sreial number is fubar
    FUBAR=`echo $MAC_UID | grep -i system`
    if [ -n "$FUBAR" ]; then
            MAC_UID=""
    fi
fi

# if we got nothing, get the property number]
# property number is a bar code value that we use and set separately
# delete this section if you don't set asset_id in nvram
if [ -z "$MAC_UID" ]; then
    if [ "$1" = "-v" ]; then echo "No serial number found, checking
asset_id"; fi
```

```
    MAC_UID=`/usr/sbin/nvram asset_id 2>/dev/null | awk '{print $2}'`
fi

# if the MAC_UID is still null
# get the primary MAC address
if [ -z "$MAC_UID" ]; then
    if [ "$1" = "-v" ]; then echo "No asset_id found, checking primary MAC
address"; fi
    MAC_UID=`facter | grep 'macaddress =>' | awk '{print $3}'`
fi

# if all the above fails, get the hostname
if [ -z "$MAC_UID" ]; then
    if [ "$1" = "-v" ]; then echo "No primary MAC address found, using
hostname for MAC_UID"; fi
    MAC_UID=`hostname`
fi

# assuming we have something, write it to nvram
# getting it from nvram is much faster and is limited to this
# specific computer
if [ "$NVRAM" == 'no' ]; then
    # cert names must be lowercase
    MAC_UID=`echo $MAC_UID | tr "[:upper:]" "[:lower:]"`
    MAC_UID=${MAC_UID}.${SUFFIX}
    /usr/sbin/nvram MAC_UID=${MAC_UID}
fi

if [ "$1" = "-v" ]; then
    tempfoo=`basename $0`
    PUPPET_RESULTS_FILE=`mktemp /tmp/${tempfoo}.XXXXXX`
    touch $PUPPET_RESULTS_FILE
    echo "puppetd -o --no-daemonize -v --certname=$MAC_UID --debug"
    puppetd -o --no-daemonize -v --certname=$MAC_UID --debug 2>&1 |
/usr/bin/tee $PUPPET_RESULTS_FILE
    #cat $PUPPET_RESULTS_FILE
    CERT_ERROR_RESULTS_1=`cat $PUPPET_RESULTS_FILE | grep 'Certificate
request does not match existing certificate'`
    CERT_ERROR_RESULTS_2=`cat $PUPPET_RESULTS_FILE | grep 'Retrieved
certificate does not match private key'`
    rm $PUPPET_RESULTS_FILE
else

    RESULTS=`puppetd -o --no-daemonize -v --certname=$MAC_UID 2>&1`
    CERT_ERROR_RESULTS_1=`echo $RESULTS | grep 'Certificate request does not
match existing certificate'`
    CERT_ERROR_RESULTS_2=`echo $RESULTS | grep 'Retrieved certificate does
not match private key'`
fi

# this cert error says the key on the client has changed, so need to clean
the cert on the server
if [ -n "$CERT_ERROR_RESULTS_1" ]; then
    CMD="http://${SERVER}/cgi-bin/cleanCert.rb?certname=${MAC_UID}"
    if [ "$1" = "-v" ]; then echo "Sleeping for 10 seconds, then cleaning
cert on server"; fi
    sleep 10;
    if [ "$1" = "-v" ]; then echo "curl $CMD"; fi
    curl "$CMD"
fi

# this is also a cleaning issue, but the server needs to be cleaned and the
client SSL cert needs to be cleaned
if [ -n "$CERT_ERROR_RESULTS_2" ]; then
    # clean the cert on the server
    CMD="http://${SERVER}/cgi-bin/cleanCert.rb?certname=${MAC_UID}"
    if [ "$1" = "-v" ]; then echo "Sleeping for 10 seconds, then cleaning
cert on server"; fi
    sleep 10;
    if [ "$1" = "-v" ]; then echo "curl $CMD"; fi
    curl "$CMD"
    # clear the local cached cert
    if [ "$1" = "-v" ]; then echo "Cleaning local cached cert"; fi
    rm -rf /etc/puppet/ssl/certs/${MAC_UID}.pem
    if [ "$1" = "-v" ]; then echo "Run puppet again and it should work now";
fi

    # all should be good on the next puppet run
fi
```

Appendix 2: cleanCert.rb

```
#!/usr/bin/ruby

# clearCert.rb
# cgi to clean a cert
# note: _www needs to have sudo rights to use puppetca
# visudo
# www ALL = NOPASSWD: /usr/bin/puppetca, !/usr/bin/puppetca --clean --all

class Puppetca
    # removes old certificate if it exists
    # parameter is the certname to use
    # need to allow the _www user to use sudo with the puppetca command
    # added using visudo
    # _www    ALL = NOPASSWD: /usr/sbin/puppetca, !/usr/sbin/puppetca --clean
--all
    def self.clean certname, addr
            command = "/usr/bin/sudo /usr/sbin/puppetca --clean #{certname}"
            %x{#{command}}
            command = "logger '#{addr} cleaned #{certname}'"
            %x{#{command}}
            return true
    end
end

=begin
CGI starts here
=end

# get the value of the passed param in the URL Query_string
require 'cgi'
cgi=CGI.new
certname = cgi["certname"]

# define the characters that are allow to avoid an injection attack
# 0-9, a-z, period, dash, and colon are allowed. All else is not
pattern = /[^a-z0-9.\-:]/
# determine if any other characters are in the certname
reject = (certname =~ pattern) ? 1 : 0

if ((reject == 0) && Puppetca.clean(certname, ENV['REMOTE_ADDR']))
    cgi.out("status" => "OK", "connection" => "close") {"OK #{certname}
cleaned\n"}
else
    cgi.out("status" => "BAD_REQUEST", "connection" => "close") {"Not
Processed: #{certname}\n"}
end
```