Boot Manager Technical Documentation

Aaron Klingaman <alk@cs.princeton.edu>

Table of Contents

1. Components 1
2. Soure Code 2
3. API Calls 2
3.1. Authentication 2
3.2. PLC API Calls 3
4. Core Package 3
4.1. Boot States 4
4.2. Flow Chart 4
4.3. Boot CD Environment 5
4.4. Node Configuration Files 5
5. User Interface for Node Management 7
5.1. Adding Nodes 7
5.2. Updating Node Network Settings 7
5.3. Removing Nodes 7
6. BootManager Configuration 8
7. Installer Hardware Detection 9
8. Backward Compatibility 10
9. Common Scenarios 10
Bibliography 11

1. Components

The entire Boot Manager system consists of several components that are designed to work together to provide the functionality outline in the Boot Manager PDN [1]. These consist of:

- A set of API calls available at PlanetLab Central
- An API authentication mechanism used exclusively by the BootManager, for the above API calls
- A package to be run in the boot cd environment on nodes containing core logic
- A user interface allowing authorized users to add and manage nodes and create node/BootManager configuration files

The previous implementation of the software responsible for installing and booting nodes consisted of a set of boot scripts that the boot cd would download and run, depending on the node's current boot state. Only the necessary script for the current state would be downloaded, and the logic behind which script the node was sent to the node existed on the boot server in the form of PHP scripts. However, the intention with the new BootManager system is to send the same script back for all nodes (consisting of the core BootManager code), in all boot states, each time the node starts. Then, the boot manager will run and detiremine which operations to perform on the node, based on the current boot state. All state based

logic for the node boot, install, debug, and reconfigure operations are contained in one place; there is no longer any boot state specific logic at PLC.

2. Soure Code

All BootManager source code is located in the repository 'bootmanager' on the PlanetLab CVS system. For information on how to access CVS, consult the PlanetLab website. Unless otherwise noted, all file references refer to this repository.

3. API Calls

Most of the API calls available as part of the PlanetLab Central API are intended to be run by users, and thus authentication for these calls is done with the user's email address and password. However, the API calls described below will be run by the nodes themselves, so a new authentication mechanism is required.

3.1. Authentication

As is done with other PLC API calls, the first parameter to all BootManager related calls will be an authentication structure, consisting of these named fields:

- AuthMethod

  The authentication method, only 'hmac' is currently supported

- node_id

  The node id, contained on the configuration file.

- node_ip

  The node's primary IP address. This will be checked with the node_id against PLC records.

- value

  The authentication string, depending on method. For the 'hmac' method, a hash for the call using the HMAC algorithm, made from the parameters of the call the key contained on the configuration file. For specifics on how this is created, see below.

Authentication is succesful if PLC is able to create the same hash from the values usings its own copy of the node key. If the hash values to not match, then either the keys do not match or the values of the call were modified in transmision and the node cannot be authenticated.

Both the BootManager and the authentication software at PLC must agree on a method for creating the hash values for each call. This hash is essentially a finger print of the method call, and is created by this algorithm:

1. Take the value of every part of each parameter, except the authentication structure, and convert them to strings. For arrays, each element is used. For dictionaries, not only is the value of all the items used, but the keys themselves. Embedded types (arrays or dictionaries inside arrays or dictionaries, etc), also have all values extracted.

2. Alphabetically sort all the parameters.

3. Concatenate them into a single string.

4. Prepend the string with the method name and [, and append].

The implementation of this algorithm is in the function serialize_params in the file source/BootAPI.py. The same algorithm is located in the 'plc_api' repository, in the function serialize_params in the file PLC/Auth.py.

The resultant string is fed into the HMAC algorithm with the node key, and the resultant hash value is used in the authentication structure.

This authentication method makes a number of assumptions, detailed below.

1. All calls made to PLC are done over SSL, so the details of the authentication structure cannot be viewed by 3rd parties. If, in the future, non-SSL based calls are desired, a sequence number or some other value making each call unique will would be required to prevent replay attacks. In fact, the current use of SSL negates the need to create and send hashes across - technically, the key itself could be sent directly to PLC, assuming the connection is made to an HTTPS server with a third party signed SSL certificate.

2. Athough calls are done over SSL, they use the Python class libary xmlrpclib, which does not do SSL certificate verification.

3.2. PLC API Calls

Full, up to date technical documentation of these functions can be found in the PlanetLab Central API documentation. They are listed here for completeness.

- BootUpdateNode(authentication, update_values)

  Update a node record, including its boot state, primary network, or ssh host key.

- BootCheckAuthentication(authentication)

  Simply check to see if the node is recognized by the system and is authorized.

- BootGetNodeDetails(authentication)

  Return details about a node, including its state, what networks the PLC database has configured for the node, and what the model of the node is.

- BootNotifyOwners(authentication, message, include_pi, include_tech, include_support)

  Notify someone about an event that happened on the machine, and optionally include the site PIs, technical contacts, and PlanetLab Support.

4. Core Package

The Boot Manager core package, which is run on the nodes and contacts the Boot API as necessary, is responsible for the following major functional units:

- Configuring node hardware and installing the PlanetLab operating system

- Putting a node into a debug state so administrators can track down problems
- Reconfiguring an already installed node to reflect new hardware, or changed network settings
- Booting an already installed node into the PlanetLab operating system

4.1. Boot States

Each node always has one of four possible boot states.

1. 'inst'

   Install. The boot state cooresponds to a new node that has not yet been installed, but record of it does exist. When the boot manager starts, and the node is in this state, the user is prompted to continue with the installation. The intention here is to prevent a non-PlanetLab machine (like a user's desktop machine) from becoming inadvertantly wiped and installed with the PlanetLab node software.

2. 'rins'

   Reinstall. In this state, a node will reinstall the node software, erasing anything that might have been on the disk before.

3. 'boot'

   Boot. This state cooresponds with nodes that have sucessfully installed, and can be chain booted to the runtime node kernel.

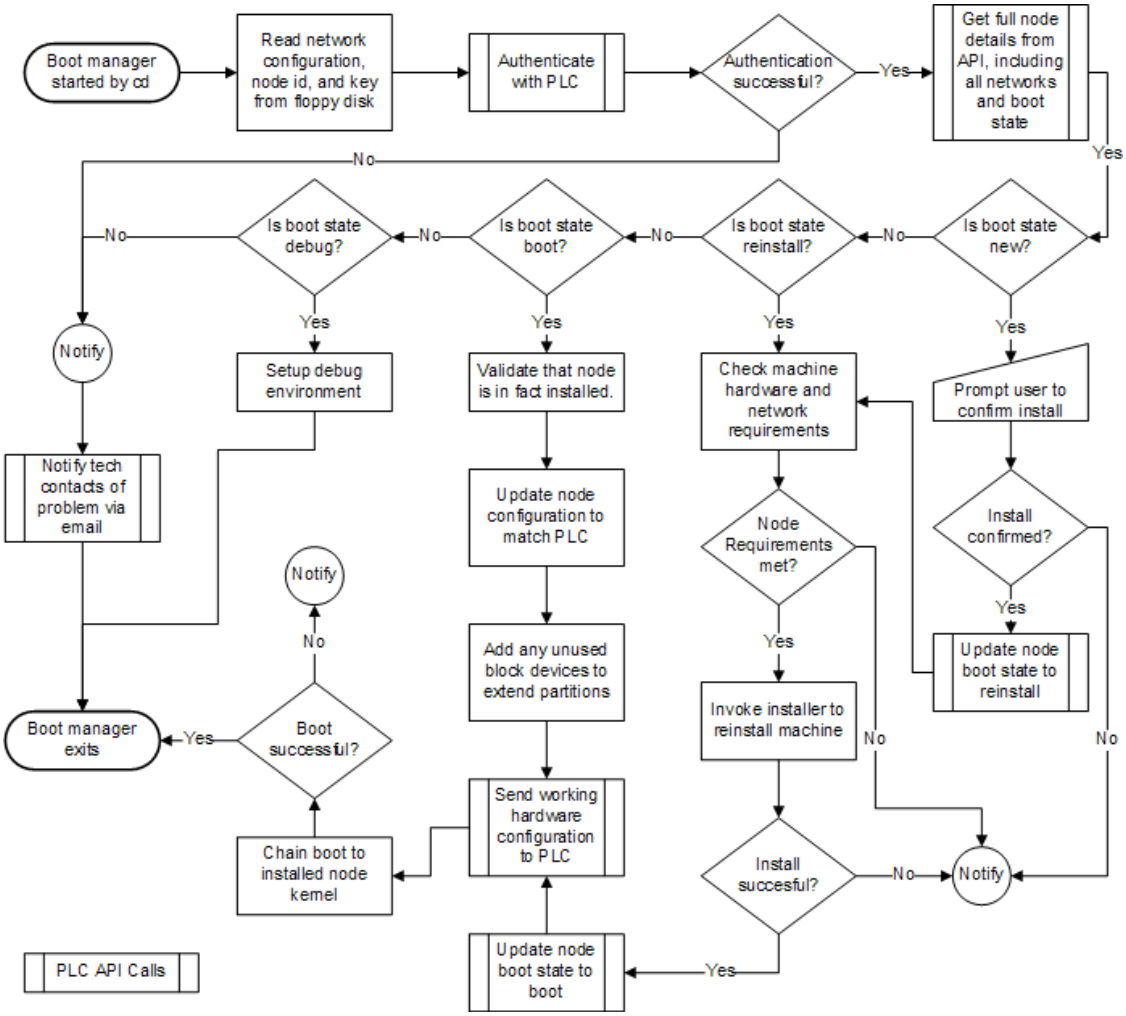
4. 'dbg'

   Debug. Regardless of whether or not a machine has been installed, this state sets up a node to be debugged by administrators.

4.2. Flow Chart

Below is a high level flow chart of the boot manager, from the time it is executed to when it exits. This core state machine is located in source/BootManager.py.

Figure 1. Boot Manager Flow Chart



4.3. Boot CD Environment

The boot manager needs to be able to operate under all currently supported boot cds. The new 3.0 cd contains software the current 2.x cds do not contain, including the Logical Volume Manager (LVM) client tools, RPM, and YUM, among other packages. Given this requirement, the boot cd will need to download as necessary the extra support files it needs to run. Depending on the size of these files, they may only be downloaded by specific steps in the flow chart in figure 1, and thus are not mentioned.

See the PlanetLab BootCD Documentation for more information about the current, 3.x boot cds, how they are build, and what they provide to the BootManager.

4.4. Node Configuration Files

To remain compatible with 2.x boot cds, the format and existing contents of the configuration files for the nodes will not change. There will be, however, the addition of three fields:

1. NET_DEVICE

If present, use the device with the specified mac address to contact PLC. The network on this device will be setup. If not present, the device represented by 'eth0' will be used.

2. NODE_KEY

   The unique, per-node key to be used during authentication and identity verification. This is a fixed length, random value that is only known to the node and PLC.

3. NODE_ID

   The PLC assigned node identifier.

An example of a configuration file for a dhcp networked machine:

```
IP_METHOD="dhcp"
HOST_NAME="planetlab-1"
DOMAIN_NAME="cs.princeton.edu"
NET_DEVICE="00:06:5B:EC:33:BB"
NODE_KEY="79efbe871722771675de604a227db8386bc6ef482a4b74"
NODE_ID="121"
```

An example of a configuration file for the same machine, only with a statically assigned network address:

```
IP_METHOD="static"
IP_ADDRESS="128.112.139.71"
IP_GATEWAY="128.112.139.65"
IP_NETMASK="255.255.255.192"
IP_NETADDR="128.112.139.127"
IP_BROADCASTADDR="128.112.139.127"
IP_DNS1="128.112.136.10"
IP_DNS2="128.112.136.12"
HOST_NAME="planetlab-1"
DOMAIN_NAME="cs.princeton.edu"
NET_DEVICE="00:06:5B:EC:33:BB"
NODE_KEY="79efbe871722771675de604a227db8386bc6ef482a4b74"
NODE_ID="121"
```

Existing 2.x boot cds will look for the configuration files only on a floppy disk, and the file must be named 'planet.cnf'. The new 3.x boot cds, however, will initially look for a file named 'plnode.txt' on either a floppy disk, or burned onto the cd itself. Alternatively, it will fall back to looking for the original file name, 'planet.cnf'. This initial file reading is performed by the boot cd itself to bring the nodes network online, so it can download and execute the Boot Manager.

However, the Boot Manager will also need to identify the location of and read in the file, so it can get the extra fields not initially used to bring the network online (primarily node_key and node_id). Below is the search order that the BootManager will use to locate a file.

Configuration file location search order:

File name	Floppy drive	Flash devices	Root file system, in /	CDRom, in /usr/boot	CDRom, in /usr
plode.txt	1	2	4	5	6
planet.cnf	3				

5. User Interface for Node Management

5.1. Adding Nodes

New nodes are added to the system explicitly by either a PI or a tech contact, either directly through the API calls, or by using the appropriate interfaces on the website. As nodes are added, their hostname, network configuration method (dhcp or static), and any static settings are required to be entered. Regardless of network configuration method, IP address is required. When the node is brought online, the records at PLC will be updated with any remaining information.

After a node is added, the user has the option of creating a configuration file for that node. Once the node is added, the contents of the file are created automatically, and the user is prompted to download and save the file. This file contains only the primary network interface information (necessary to contact PLC), the node id, and the per-node key.

The default boot state of a new node is 'inst', which requires the user to confirm the installation at the node, by typing yes on the console. If this is not desired, as is the case with nodes in a co-location site, or for a large number of nodes being setup at the same time, the administrator can change the node state, after the entry is in the PLC records, from 'inst' to 'reinstall'. This will bypass the confirmation screen, and proceed directly to reinstall the machine (even if it already had a node installation on it).

5.2. Updating Node Network Settings

If the primary node network address must be updated, if the node is moved to a new network for example, then two steps must be performed to successfully complete the move:

1. The node network will need to be updated at PLC, either through the API directly or via the website.
2. Either the floppy file regenerated and put into the machine, or, update the existing floppy to match the new settings.

If the node ip address on the floppy does not match the record at PLC, then the node will not boot until they do match, as authentication will fail. The intention here is to prevent a malicious user from taking the floppy disk, altering the network settings, and trying to bring up a new machine with the new settings.

On the other hand, if a non-primary network address needs to be updated, then simply updating the record in the configuration file will suffice. The boot manager, at next restart, will reconfigure the machine, and update the PLC records to match the configuration file.

5.3. Removing Nodes

Nodes are removed from the system by:

1. Deleting the record of the node at PLC
2. Shutting down the machine.

Once this is done, even if the machine attempts to come back online, it cannot be authorized with PLC and will not boot.

6. BootManager Configuration

All run time configuration options for the BootManager exist in a single file located at source/configuration. These values are described below.

- `VERSION`

  The current BootManager version. During install, written out to /etc/planetlab/install_version

- `BOOT_API_SERVER`

  The full URL of the API server to contact for authenticated operations.

- `TEMP_PATH`

  A writable path on the boot cd we can use for temporary storage of files.

- `SYSIMG_PATH`

  The path were we will mount the node logical volumes during any step that requires access to the disks.

- CACERT_PATH

  Variable not used anymore.

- `NONCE_FILE`

  Variable not used anymore.

- `PLCONF_DIR`

  The path that PlanetLab node configuration files will be created in during install. This should not be changed from /etc/planetlab, as this path is assumed in other PlanetLab components.

- `SUPPORT_FILE_DIR`

  A path on the boot server where per-step additional files may be located. For example, the packages that include the tools to allow older 2.x version boot cds to partition disks with LVM.

- `ROOT_SIZE`

  During install, this sets the size of the node root partition. It must be large enough to house all the node operational software. It does not store any user/slice files. Include 'G' suffix in this value, indicating gigabytes.

- `SWAP_SIZE`

  How much swap to configure the node with during install. Include 'G' suffix in this value, indicating gigabytes.

- `SKIP_HARDWARE_REQUIREMENT_CHECK`

  Whether or not to skip any of the hardware requirement checks, including total disk and memory size constraints.

- `MINIMUM_MEMORY`

  How much memory is required by a running PlanetLab node. If a machine contains less physical

memory than this value, the install will not proceed.

- `MINIMUM_DISK_SIZE`

  The size of the small disk we are willing to attempt to use during the install, in gigabytes. Do not include any suffixes.

- `TOTAL_MINIMUM_DISK_SIZE`

  The size of all usable disks must be at least this sizse, in gigabytes. Do not include any suffixes.

- `INSTALL_LANGS`

  Which language support to install. This value is used by RPM, and is used in writting /etc/rpm/macros before any RPMs are installed.

- `NUM_AUTH_FAILURES_BEFORE_DEBUG`

  How many authentication failures the BootManager is willing to except for any set of calls, before stopping and putting the node into a debug mode.

7. Installer Hardware Detection

When a node is being installed, the Boot Manager must identify which hardware the machine has that is applicable to a running node, and configure the node properly so it can boot properly post-install. The general procedure for doing so is outline in this section. It is implemented in the `source/systeminfo.py` file.

The process for identifying which kernel module needs to be load is:

1. Create a lookup table of all modules, and which PCI ids coorespond to this module.
2. For each PCI device on the system, lookup its module in the first table.
3. If a module is found, put in into one of two categories of modules, either network module or scsi module, based on the PCI device class.
4. For each network module, write out an 'eth<index>' entry in the modprobe.conf configuration file.
5. For each scsi module, write out a 'scsi_hostadapter<index>' entry in the modprobe.conf configuration file.

This process is fairly straight forward, and is simplified by the fact that we currently do not need support for USB, sound, or video devices when the node is fully running. The boot cd itself uses a similar process, but includes USB devices. Consult the boot cd technical documentation for more information.

The creation of the PCI id to kernel module table lookup uses three different sources of information, and merges them together into a single table for easier lookups. With these three sources of information, a fairly comprehensive lookup table can be generated for the devices that PlanetLab nodes need to have configured. They include:

1. The installed `/usr/share/hwdata/pcitable` file

   Created at the time the hwdata rpm was built, this file contains mappings of PCI ids to devices for a

large number of devices. It is not necessarily complete, and doesn't take into account the modules that are actually available by the built PlanetLab kernel, which is a subset of the full set available (again, PlanetLab nodes do not have a use for network or video drivers, and thus are not typically built).

2. From the built kernel, the `modules.pcimap` from the `/lib/modules/<kernelversion>/` directory.

   This file is generated at the time the kernel is installed, and pulls the PCI ids out of each module, for the modules list they devices they support. Not all modules list all devices they sort, and some contain wild cards (that match any device of a single manufacturer).

3. From the built kernel, the `modules.dep` from the `/lib/modules/<kernelversion>/` directory.

   This file is also generated at the time the kernel is installed, but lists the dependencies between various modules. It is used to generate a list of modules that are actually available.

It should be noted here that SATA (Serial ATA) devices have been known to exist with both a PCI SCSI device class, and with a PCI IDE device class. Under linux 2.6 kernels, all SATA modules need to be listed in modprobe.conf under 'scsi_hostadapter' lines. This case is handled in the hardware loading scripts by making the assumption that if an IDE device matches a loadable module, it should be put in the modprobe.conf file, as 'real' IDE drivers are all currently built into the kernel, and do not need to be loaded. SATA devices that have a PCI SCSI device class are easily identified.

It is enssential that the modprobe.conf configuration file contain the correct drivers for the disks on the system, if they are present, as during kernel installation the creation of the initrd (initial ramdisk) which is responsible for booting the system uses this file to identify which drivers to include in it. A failure to do this typically results in an kernel panic at boot with a 'no init found' message.

8. Backward Compatibility

Given the large number of nodes in PlanetLab, and the lack of direct physical access to them, the process of updating all configuration files to include the new node id and node key will take a fairly significant amount of time. Rather than delay deployment of the Boot Manager until all machines are updated, alternative methods for aquiring these values is used for existing nodes.

First, the node id. For any machine already part of PlanetLab, there exists a record of its IP address and MAC address in PlanetLab central. To get the node_id value, if it is not located in the configuration file, the BootManager uses a standard HTTP POST request to a known php page on the boot server, sending the IP and MAC address of the node. This php page queries the PLC database, and returns a node_Id if the node is part of PlanetLab, -1 otherwise.

Second, the node key. All Boot CDs currently in use, at the time they request a script from PLC to run, send in the request a randomly generated value called a boot_nonce, usually 32 bytes or larger. During normal BootManager operation, this value is ignored. However, in the absense of a node key, we can use this value. Although it is not as secure as a typical node key (because it is not distributed through external mechanisms, but is generated by the node itself), it can be used if we validate that the IP address of the node making the request matches the PLC record. This means that nodes behind firewalls can no longer be allowed in this situation.

9. Common Scenarios

Below are common scenarios that the BootManager might encounter that would exist outside of the doc-

umented procedures for handling nodes. A full description of how they will be handled by the BootManager follows each.

- A configuration file from previously installed and functioning node is copied or moved to another machine, and the networks settings are updated on it (but the key and node_id is left the same).

  Since the authentication for a node consists of matching not only the node id, but the primary node ip, this step will fail, and the node will not allow the boot manager to be run. Instead, the new node must be created at PLC first, and a network configuration file for it must be generated, with its own node key.

- After a node is installed and running, the administrators mistakenly remove the cd and media containing the configuration file.

  The node installer clears all boot records from the disk, so the node will not boot. Typically, the bios will report no operating system.

- A new network configuration file is generated on the website, but is not put on the node.

  Creating a new network configuration file through the PLC interfaces will generate a new node key, effectively invalidating the old configuration file (still in use by the machine). The next time the node reboots and attempts to authentication with PLC, it will fail. After two consecutive authentication failures, the node will automatically put itself into debug mode. In this case, regardless of the API function being called that was unable to authentication, the software at PLC will automatically notify the PlanetLab administrators, and the contacts at the site of the node was able to be identified (usually through its IP address or node_id by searching PLC records.).

Bibliography

[1] *The PlanetLab Boot Manager*. January 14, 2005. Aaron Klingaman.