1 Defining the Initial Topology

GrIDS uses a hierarchy of departments to manage the collection of machines being monitored. Each host being monitored by GrIDS must be placed under the control of a single department (graph engine/aggregator). Departments, in turn, can be managed by a department at the next highest level in the hierarchy. An initial placement of hosts into departments and departments into the hierarchy must be configured when starting GrIDS. This initial configuration is defined and added to the GrIDS database using the *$GRIDSPATH/user_interface/grids.db* file. Figure 1 shows a sample two-tiered hierarchy. The examples in this section will explain how one can define this hierarchy when starting GrIDS.

Figure 1: Sample two tier hierarchy used in this section's examples.

When grids is started it will examine the file *$GRIDSPATH/user_interface/grids.db* to determine the initial hierarchy. The syntax and structure of this file will be illustrated in the following examples.

1.1 Choose the hosts to monitor

First, as shown in the following excerpt from *grids.db*, all hosts to be monitored in the initial configuration are added to the `host_to_os` perl hash reference. One simply includes the host-name and it's corresponding OS as a key/value pair in the has. This list must contain all hosts in all departments that will be included in the initial hierarchy.

```
#  Standard startup config stuff

$grids_rsh = 'rsh';

##
##  These structures are the *authoritative* master lists of hosts.
##  NOTE :: All hosts must be *DECLARED* here, even if their OS type is unknown:
##

$self->{'host_to_os'} = {
'host1.edu' => 'solaris',
'host2.edu' => 'irix',
'host3.edu' => 'sunos',
'host4.edu' => 'ultrix',
'host5.edu' => 'linux',
'host6.edu' => 'unknown',
'host7.edu' => 'solaris',
'host8.edu' => 'solaris',
'host9.edu' => 'unknown',
};
```

1.2 Group hosts into their respective departments

Next, data structures are defined that associate host-names to departments in the hierarchy. The following portion of code from the *grids.db* file illustrates a typical setup assigning hosts to three different departments.

```
$self->{'ROOT_hosts'} = [
'host1.edu',
```

```
'host2.edu',
'host3.edu',
];
$self->{'dept1_hosts'} = [
'host4.edu',
'host5.edu',
'host6.edu',
                                ];
$self->{'dept2_hosts'} = [
'host7.edu',
'host8.edu',
'host9.edu',
                                ];
```

Each list reference simply contains all the hostnames that will be grouped in that department. Notice that the key names of the list references; `ROOT_hosts`, `dept1_hosts` and `dept2_hosts`, are completely arbitrary as they will be refered to by the user later in the *grids.db* script.

1.3 Assign GrIDS software modules to hosts

After all initial hosts have been defined and grouped by department, the startup hierarchy database is configured. The following section of code from the *grids.db* file illustrate how this can be done.

```
-------------------------------------------------------------------------------
#
#  AGG Hierarchy - Partitions all hosts into departments based upon their
#                  room location.  Hierarcy is only 2 deep.
#

 GEOM_AGG:   #  This sections gets entered when using the "agg" option in the
             #  grids startup script ( e.g. "% grids start agg" )

    $num_hosts = 0;
    foreach $key ( keys %{$self->{'host_to_os'}} ) {
$num_hosts++;
    }

#  Choose the host for the Hierarchy Server
    $self->{'ohs_host'} = 'root_server.edu';

#  Define the geometry of all departments
    @dept_geometry = (

      {
  DEPT_NAME => 'ROOT',  PARENT_DEPT => 0,
  SM_HOST => $self->{'ROOT_hosts'}[1],
  AGG_HOST => $self->{'ROOT_hosts'}[0],
  NUM_HOSTS => $#{$self->{'ROOT_hosts'}}+1,
  SPECIFIC_HOSTS => [ @{$self->{'ROOT_hosts'}} ],
      },

      {
```

```
  DEPT_NAME => 'dept1', PARENT_DEPT => 'ROOT',
  SM_HOST => $self->{'dept1_hosts'}[1],
  AGG_HOST => $self->{'dept1_hosts'}[0],
  NUM_HOSTS => $#{$self->{'dept1_hosts'}}+1,
  SPECIFIC_HOSTS => [ @{$self->{'dept1_hosts'}} ],
      );

      {
  DEPT_NAME => 'dept2', PARENT_DEPT => 'ROOT',
  SM_HOST => $self->{'dept2_hosts'}[1],
  AGG_HOST => $self->{'dept2_hosts'}[0],
  NUM_HOSTS => $#{$self->{'dept2_hosts'}}+1,
  SPECIFIC_HOSTS => [ @{$self->{'dept2_hosts'}} ],
      );

    last CONFIG;
# end AGG section

-------------------------------------------------------------------------------
```

Each element of the `dept_geometry` array is a perl hash reference. Keys of each hash define the item being configured. Departments are defined by name and their parent department in the hierarchy, if any, is identified. Hosts running critical componants; the software manager and the aggregator host are selected, and all the managed hosts grouped in the previous section are added.

Note that it is possible to include several alternate hierarchy configurations in the same *grids.db* database file. A specific hierarchy is then selected via an argument passed to the startup script.

2 Running GrIDS

2.1 Startup, Checking and Stopping GrIDS

Starting and stopping GrIDS is done using the *$GRIDSPATH/user_interface/grids* script. This script also is used to access a running GrIDS hierarchy to perform consistency checks and dump the current hierarchy definition. Once the initial hierarchy configuration has been set in *grids.db*, issuing

`grids start`

from the command line will start all GrIDS modules on all hosts using the default configuration. Alternate hierarchies defined in *grids.db* can be selected using the second command line argument (e.g. `grids start agg` will select the `GEOM_AGG:` block in *grids.db*). When the startup script finishes without errors, all GrIDS modules will be running with open communication ports with one another. Sniffers and other data sources, however, are not yet running so GrIDS will not be collecting any possible intrusion data.

To verify that all GrIDS modules are operating correctly, the command

`grids check`

will query all relevant modules for their current state and verify that all maintain a consistent view of the global configuration. If some modules fail to reply or an incosistent response is recieved then a error is reported. In this event, GrIDS should be shutdown and restarted.

The *grids* script is also used to shutdown all currently running GrIDS modules. All GrIDS related processes on each machine will be killed via the

`grids destroy`

command. Before turning on any sniffers or data sources, one should verify that starting GrIDS, checking it's consistency and shuting down GrIDS works without any errors.

2.2 Starting and Stopping Sniffers

There are two methods for starting and stopping sniffer data sources; through the user interface gui, and using a perl script. The **??** section describes how to start data sources using the gui. Briefly, one right clicks on the chosen host and selects the sniffer item in the displayed dialog box. Additionaly, the *sniff.pl* perl script is provided to start sniffers from the command line. *sniff.pl* connects to the currently running GrIDS hierarchy, starts sniffers in each department on the same host that runs the graph aggregator engine. In both cases, a message will be printed to `stdout` indicating that tcpdump is listening on the network interface when the sniffer is started correctly.