

Back to the Basics: Solaris Default Processes and init.d Pt. III

Hal Flynn 2000-06-25

Back to the Basics: Solaris Default Processes and init.d Pt. III

By Hal Flynn <hmflynn@earthlink.net>

last updated Monday, June 25, 2000

• Introduction

This article has been written to provide insight into a stock installation of Solaris 8, and the services started by default. Out of the box, Solaris 8 by runs many services. This article was written using Solaris 8, the latest version available, and a Sparcstation 20. Most of this document will apply to releases of Solaris prior to 8, and to both the Sparc and Intel architectures. For documentation purposes, a full OEM install was done. Many topics discussed will be familiar to seasoned administrators. However, this document will benefit all parties involved in the administration and security aspects of Solaris.

• Review

In the first part of this article, we performed a default installation of the Solaris 8 Operating System, and looked at the processes that run on a stock install of Solaris 8.

In the second part of this article, we reviewed Solaris bootstrap and initialization, rc scripts, and run levels on the Solaris 8 Operating System.

• Overview

Now that we understand how the system is booted, how the processes are started when the system is booted, and where the processes are started from, we'll discuss altering the processes started a boot. We'll additionally look further into the rc scripts and daemons for configuration options to disable their startup, and discuss ways that daemons may be disabled either through removal of the rc script responsible for their starting, or alternate configuration options.

• Processes and their origin

Our reference point for this section is Part I of this article. We'll start with the first true process started on a Solaris system, and work out way down the table to see where each process is started, and how it can be

disabled. We'll do this by locating and showing the rc script that actually starts each respective process while the system boots into run level 3.

These scripts may be changed to disable their startup on an individual basis. One can either move the script from an `S##name` (where `##` is the startup sequence number, and `name` is the name of the script) file to a `s##name`. This will disable the running of the script at boot time, although the script is capable of being executed on the command line. Many other scripts that launch daemons into the process table support additional configuration options that allow certain aspects to be disabled without entirely moving the script or hacking on the source. Some of these configuration options will be mentioned as encountered.

init is the first true process started on a Solaris System. *sched*, *init*, *pageout*, and *fsflush* we'll not talk about, as these processes are mandatory on any functional Solaris System.

Processes 52 and 56 are *devfseventd* and *devfsadm* and are started together from the `S50devfsadm` script in the `/etc/rc.S` directory. This script additionally configures the symbolic links to devices in the `/dev` directory, and sets the default frame buffer device.

Process 122 is *in.routed*. *in.routed* is started from the `/etc/rc2.d` directory in the `S69inet` script. This script has many responsibilities, including initializing IPsec, setting the randomization of sequence numbers for tcp, dhcp configuration options, and more. Keeping focus, *in.routed* is started if there's no default route set, and for the sake of router discovery. This daemon will also permit ip forwarding if more than two interfaces exist, at least one point-to-point interface exists, or `/etc/gateways` exists. Upon examining the script, we find the following code:

```
if [ ! -f /etc/notrouter -a $numdhcp -eq 0 -a          \
    \ ( $numifs -gt 2 -o $numtpifs -gt 0 -o -f /
etc/gateways \ ) ]; then
    < --- comments snipped --- >
    echo 'Machine is an IPv4 router.'
    /usr/sbin/ndd -set /dev/ip ip_forwarding 1
```

This piece of logic is pertinent. It states that if an interface is NOT configured via dhcp, or if the file `/etc/notrouter` exists, *in.routed* will permit ip forwarding, calling `ndd` to set this in the ip kernel module. Therefore, if a machine has more than one interface, and one does not wish to enable ip forwarding between interfaces, creating the blank file `/etc/notrouter` will disable this functionality. The supported arguments for command line execution are 'start' and 'stop'.

Process 129 is *in.ndpd*. *in.ndpd* is started from within the same script as *in.routed*. During execution, the script

searches for more than one loopback interface. If another loopback interface exists, parsing takes place to find out if the interface is IPv6. If this is a fact, the script searches for the file `/etc/inet/ndpd.conf`. If this file doesn't exist, the host portion of IPv6 is run. Otherwise, the script will execute `ndd`, and IPv6 based ip forwarding, send redirects, and ignore redirects will be enabled. In host mode, the script will launch the `in.ndpd` daemon, and perform IPv6 neighbor discovery.

Process 143 is *rpcbind*. `rpcbind` is started from the script `S71rpc` in the `/etc/rc2.d` directory. When `init` executes the `S71rpc` script, it tests for the `rpcbind` daemon, and if the daemon is not running, executes it. The `S71rpc` script is not responsible only for `rpc`, though. Within the source of the script, it creates a directory `/var/run/rpc_door`, NIS and/or NIS+ is configured, and the NIS+ Password Update Daemon is launched. Command line execution of the script supports three arguments: `rpcstart`, `start`, and `stop`. `start` and `rpcstart` are synonymous. The `stop` argument is self explanatory.

Process 166 is *inetd*. `inetd` is started from the `S72inetsvc` script located in `/etc/rc2.d`. `inetd` is the topic of previous writing in this series. "`Solaris inetd.conf`" contains information about the configuration of daemon accessible via the `inetd` service. Part I and Part II can be found at their respective links. `inetd` is started with the `-s` flag by default. This is due to dependency on the Service Access Facility. If the `sac` (Service Access Controller) is not running at the time `inetd` is executed (which it usually is not, as it's executed later in the boot sequence), `inetd` will not start. The `-s` flag will run `inetd` as a stand alone process, thereby allowing functionality. Two more notable flags supported by the `inetd` service at execution are the `-t` and `-r` flags. The `-t` flag forces `inetd` to perform a trace on all TCP services within its control. The initiating clients IP address and port number are then logged, and stored in `syslog`. The `-r` flag instructs `inetd` to monitor and suspend datagram servers within its control that are misbehaving or performing unexpectedly. Such services can fail while filling and request, and since under the control of `inetd`, will be restarted immediately after failure. This can begin a vicious cycle that will consume resources and impact performance. Upon suspension, the service will be reinitialized after a specified amount of time (the count and interval are specified as arguments to `-r`). See the man page `inetd(1M)` for more details on this subject.

Processes 174 and 178 are the `nfs statd` and `nfs lockd`. `statd` and `lockd` are both started from the `S73nfs.client` script in `/etc/rc2.d`. This script additionally mounts all remotely shared `nfs` file systems, as well as all cache file systems. Supported command line arguments are `start` and `stop`.

Process 186 is the automounter daemon. This daemon is started in the `S74autofs` script in `/etc/rc2.d`. This script executes `automountd`, and `automount`. `automountd` is the `autofs` manager, while `automount` maps `autofs` filesystems and mounts them upon execution. After a period of inactivity (10 minutes by default) `autofs` mounts will be unmounted via `automountd`. The standard `start` and `stop` arguments apply.

Process 197 is *syslogd*. *syslogd* is started in *S74syslog*, located in */etc/rc2.d*. The script first checks for the existence of */etc/syslog.conf* (the *syslog* configuration file), and if it exists, attempts to execute *syslogd*. This script will additionally make an attempt to save any crash data generated by executing */usr/bin/savecore* and saving it per the configurations in */etc/dumpadm.conf*. This script also checks for the existence of */var/adm/messages*, the output file for *syslogd*. If this file does not exist, a copy is made from */dev/null*, and *chmod 644* is executed on the newly created file. Standard arguments to execution apply.

Process 201 is *cron*. *cron* is started in the *S75cron* file in */etc/rc2.d*. The sole purpose of this file is the graceful starting and stopping of *cron*. It checks for the file */etc/cron.d/FIFO* and the process *cron* in the process table, and if *cron* is already running, will not execute an additional process. If *cron* is not running, it executes *rm* to remove */etc/cron.d/FIFO* from the file system in case it exists, and executes a new *cron* process. Standard arguments apply.

Process 212 is *nscd*. *nscd* is started in the *S76nscd* file in */etc/rc2.d*. This script checks for */etc/nscd.conf* prior to executing *nscd*, and attempts execution if the check is successful. Additionally, this script check */etc/nsswitch.conf* for NIS+ configuration, and if NIS+ configuration information is found, attempts to run *nscd* in a more secure mode. Standard arguments apply.

Process 218 is *lpsched*. *lpsched* is executed from *S80lp* in */etc/rc2.d*. When started, this script will run *lpsched*. If the stop argument is specified when this script is executed, *lpschut* is called. Standard arguments apply.

Process 231 is *powerd*. *powerd* is started after *S85power* is executed from */etc/rc2.d*. This script checks for */etc/power.conf*, and if the test is successful, executes *pmconfig*. This will spawn *powerd* to standby and monitor status according to configuration in */etc/power.conf*. Standard arguments apply.

Process 241 is *utmpd*. *utmpd* is started in *S88utmpd* located in */etc/rc2.d*. This script checks for a running process of *utmpd* in the process table, and also searches for the file */etc/utmppipe*. If the process is not running in the process table, this script removes any occurrence of */etc/utmppipe* that may exist, and executes the daemon. Standard arguments apply.

Process 243 is *cimomboot*. This process is started from the *S90wbem* script in */etc/rc2.d*. Upon execution of this script, first the directory */var/sadm/wbem/log* is checked for. If the existence is false, *mkdir* is executed to create the directory, and the *cimomboot* daemon is started. To communicated the pertinence of the */var/sadm/wbem/log* directory, after *cimomboot* is started, a file named *cimombootserver.pid* is created in the directory, which stores the pid of the running process. This is important, as the stop argument to the rc script will cat this file and parse it for the servers pid, then kill the process from the returned pid.

Process 248 is *vold*. *vold* starts from the execution of the *S92volmgt* script in */etc/rc2.d*. *S92volmgt* simply checks for the */etc/vold.conf* configuration file, and then executes *vold* if the test for the configuration file returns true. Standard arguments apply.

Process 268 is *snmpdx*. This process starts from *S76snmpdx* in */etc/rc3.d*. This script checks for the existence of */etc/snmp/conf/snmpdx.rsrc* and the executable */usr/lib/snmp/snmpdx*. If these exist, */usr/lib/snmp/snmpdx* is executed using the */etc/snmp/conf* file. Standard arguments apply.

Process 271 is *mibiisa*. As you may have noted, this process is parented by *snmpdx* (PPID 268). This is part of the standard execution of the *snmpdx* package.

Process 276 is *dtlogin*. This process is started from the */etc/rc2.d* directory in script *S99dtlogin*. Numerous command line arguments exist in this script. The one used that we're interested in is 'start'. See this script for other information and options.

Processes 277 and 278 are *dmispd* and *snmpXdmid*. These processes are started from the *S77dmi* script in */etc/rc3.d*. *dmispd* is executed using a block of logic that first checks for the existence of */etc/dmi/conf/dmispd.conf* and the executable */usr/lib/dmi/dmispd*. Next, the existence of */etc/dmi/conf/snmpXdmid.conf* is tested as well as the executable */usr/lib/dmi/snmpXdmid*. If both of these tests return true, *snmpXdmid* is executed. Standard command line arguments apply to this script.

Process 282 is *sac*. *sac* is started from the */etc/inittab*.

Process 283 is *ttymon*. *ttymon* is started from */etc/inittab* as well.

Process 287 is also *ttymon*. This *ttymon* is spawned as part of *sac*.

Process 304 is *sendmail*. This process is started in directory */etc/rc2.d* from *S88sendmail*. This script checks for *sendmail* and a *sendmail.cf*. Additionally, it checks for the existence of */var/spool/mqueue*, and if this directory doesn't exist, makes it and sets it to mode 0750. This script additionally changes the */var/spool/mqueue* directory to root owned and bin group. The default flag used to launch *sendmail* is *-bd*. Later in the script, things such as queue checking and log levels are set. Standard arguments apply.

Process 1783 is *fbconsole*. This program is started from */usr/openwin/lib/openwin-sys*.

Finally, we see the remaining processes not previously discussed. These processes are children of parents started

previously by inetd. This reasserts our original statement that init is the parent of ALL processes to later run on a system.

- **Conclusion**

In this volume, we've discussed the default installation of Solaris 8, and what processes run on a stock installation. We've also spoken about run levels, rc scripts, the origin of processes at boot time, and configuration options of interest. We can see the importance of init. We can also see the importance of understanding the boot process, and the configuration of services that may or may not be necessary. From this document forth, we can now find the origin of a process that may or may not be necessary, configure it for further security, or completely disable it.

- **Acknowledgements**

Special thanks to Jeremy Rauch of Securityfocus.com.

[Privacy Statement](#)

Copyright 2006, SecurityFocus