

TACACS Tricks with Scripts - Part 2

Michael J. Martin

Continuing from TACACS+ tricks with scripts, part 1, last month's tribute to resourcefulness and sloth, we are going to look at some scripts for controlling the tac_plus daemon and accounting log processing.

Running tac_plus the Old-fashioned Way

There is not much to "starting" the tac_plus daemon. Tac_plus, like most Unix service daemons, needs to run as root. The standard build installs the daemon in /usr/local/bin. To launch the standard service from the command line, use /usr/local/bin/tac_plus -C <config file>. The only required control flag is -c, which tells the daemon the location the tac_plus configuration file (expressed as an explicit path to the configuration file). Some of the other control options include:

- The -p <port> flag, to define an alternative TCP service port to the default port 49
- The -L flag, to force DNS resolution of the clients sending requests
- The -P flag, the daemon only parses the config file for errors and exits
- The -d <value> flag, which enables debugging and the debug level

Regardless of the options, when the daemon starts, it creates a process number identification (PID) file in the /etc directory. (An alternative location can be defined by editing the makefile prior to compiling the daemon.) When started in standard mode, this file is named tac_plus.pid. If an alternative TCP service port is defined, the port number is appended to the PID file name. For example, if the tac_plus service were started to listen on port 6700, then the PID file name would be tac_plus.pid.6700.

User account and profile information is loaded into RAM when the service is started. So any changes made to the configuration file will not take effect until the service is restarted. When debugging is enabled (-d), information about the tac_plus daemon's operational state and any debugging information is sent to a file named tac_plus.log, located in the /var/tmp/ directory. This is the same directory the tac_plus accounting file is stored, by default. In order to stop the tac_plus service or reload the configuration, a service signal handler must be sent to the daemon. The built-in shell command kill is used for sending signal handlers. One of the more commonly used handlers is kill -9, known as the "stop with extreme prejudice" mode. While it stops the process, it does so without any remorse, so no event record is issued. While you can stop the tac_plus daemon using this kill option, no state tracing will be sent to the tac_plus.log file. To stop the process gracefully, use the kill option TERM (kill -TERM). This will shut down the service properly, issuing a state change notice to the log file. To reload the configuration, the signal handler -USR1 or -HUP can be used to restart the process.

Checktacplus

The checktacplus script is an uber wrapper for starting and stopping the tac_plus daemon. Its default action is to simply start the tac_plus daemon listening on the default TCP port using a configuration file that is defined in the script. To stop the tac_plus service, simply run the command followed by the control flag -k. To have the daemon reload the configuration file, run the command with the -r flag. Regardless of the exercises, the script informs you of the following:

```
[orion:martin/Articles/1b-TACACS-Accounting] martin# ./checktac.sh
tac_plus is down
Restarting tac_plus PID 623
[orion:martin/Articles/1b-TACACS-Accounting] martin#
It includes the function being performed, its status and process ID information:
[orion:martin/Articles/1b-TACACS-Accounting] martin# ./checktac.sh -k
Shutting Down the tac_plus service PID 723
[orion:martin/Articles/1b-TACACS-Accounting] martin#
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

The script's default mode is intended to be used as a daemon status checker. Run either from the command line or have CRON run the script at a regular interval to check and see if the tac_plus service is running. If not, restart the service. Setting up the script to be run from CRON is easy. Since services like tac_plus need to be run as root, the script also needs to be run as root and/or from root's crontab file. To add a job to root's crontab file, su to root and run the command crontab -e, then add the command you want to run and when you want to run it. Commands executed by CRON are controlled by the five time and date variables that precede the command call. These values are:

- Field 1 = Minutes, expressed as a number between 0 and 59
- Field 2 = Hours, expressed as a number between 0 and 23
- Field 3 = Day of the month, expressed as a number between 1 and 31
- Field 4 = Month of the year, expressed as a number between 1 and 12
- Field 5 = Day of the week, expressed as a number between 0 and 6

For example, to have the checktacplus script run every 15 minutes, the crontab entry would look like this (in this example all of the output is sent to dev null):

```
15 * * * * /usr/local/checktacplus 2>&1 /dev/null
```

If you're interested in having the status data sent to you via e-mail, you can pipe the command output to the mail program. Here's an example crontab entry:

```
15 * * * * /usr/local/bin /checktacplus I mail -s "`date +%T` - tac_plus  
running on port 49 on `hostname`  
is up PID `cat /etc/tac_plus.pid`" user@myhost.com
```

Additionally, the checktacplus script can start the daemon to listen on a different TCP port and use an alternative configuration file. This option is intended to make it easier to manage multiple daemon instances on a single server (an approach used in large environments to segment accounting data). To assist in managing a multiple daemon environment, the script checks the process table each time it runs and prints out a list of the tac_plus processes on the server:

PID	Service	Config File	Port
991	/usr/sbin/tac_plus	/etc/tacacs.conf	4489
1384	/usr/sbin/tac_plus	/etc/tacacs.conf	
1405	/usr/sbin/tac_plus	/etc/tacacs.conf	56674
2931	/usr/sbin/tac_plus	/etc/tacacs2.conf	6638

Like most scripts, to function properly some customization is needed. The locations for the tac_plus daemon and default configuration file needs to be set in the scripts global variable section. The tac_plus daemon creates a PID file in the /etc directory called tac_plus.pid. If the environment uses a TCP port other than the default, then the default PID file needs to be defined as well:

```
# Global Variables  
TACPID="/etc/tac_plus.pid"  
# Tac_plus daemon location  
TAC="/usr/sbin/tac_plus"  
# Tac_plus default config  
TACCONF="/etc/tacacs.conf"
```

Here is a complete list of command flags and functions:

TACACS Tricks with Scripts - Part 2

Michael J. Martin

- Start the default service = checktacplus
- Stop the default service = checktacplus -k
- Reload the configuration of the default service = checktacplus -r
- Start an instance running on a non-standard port, using the default configuration = checktacplus -p <port>
- Start an instance running on a non-standard port, using an alternative configuration file = checktacplus -c <port> <configuration file>
- Reload an instance running on a non-standard port = checktacplus -pr
- Stop an instance running on a non-standard port = checktacplus -kp
- Review the help information and command options = -h

Tacdebug

The tacdebug script does just what the name says; it debugs tac_plus. The tac_plus daemon has extensive debugging options. Tac_plus debugging is enabled using the -d control flag (following the configuration file definition) and a base2 byte value to set the debug mode. Here is a list of the debug options and their assigned byte values:

```
Start service and parse the configuration file to daemon log file = 2
Start service in forked mode = 4
Start service in authorization debug mode = 8
Start service in authentication debug mode = 16
Start service in password processing debug mode = 32
Start service in accounting debug mode = 64
Start service in configuration debug mode = 128
Start service in packet debug mode = 256
Start service in hex debug mode = 512
Start service in MD5 debug mode = 1024
Start the service in XOR debug mode = 2048
Start the service in clean flag debug mode = 4096
Start the service in substitute flag debug mode = 8192
Start the service in proxy debug mode = 16384
Start the service in max sessions debug mode = 32768
Start the service in debug lock flag mode = 65536
```

The tacdebug script, like checktacplus, is a wrapper. The script "cleans up" the debugging interface to make it more efficient and a little more user-friendly. First, it converts the numeric mode interface to three-letter control flags that are a little easier to remember. Then, operationally, the daemon is loaded in a debug mode and the script loads the daemon log file in a text parser so that the administrator can see the output results. This provides a quick means of changing debug modes and viewing the output when troubleshooting. Here is a complete list of command flags and functions:

```
Stop the tac_plus service = -k (needs to be run after the parser is exited
with ctrl-c)
Start the service, parse configuration file to the log file = -par
Start the service in fork mode = -for
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
Start the service in authorization debug mode = -adz
Start the service in authentication debug mode = -auto
Start the service in password debug mode = -pas
Start the service in configuration debug mode = -cfg
Start the service in packet debug mode = -pak
Start the service in MD5 debug mode = -md5
Start the service in Low Level Encrypt/Decrypt debug mode = -enc
Start the service in proxy debug mode = -pro
Start the service in max-session debug mode (if compiled) = -max
Review the help information and command options = -h
```

Tacreport

The tacreport script processes tac_plus accounting and generates reports on user terminal access sessions, IOS command usage and system events. While quite useful, authentication, authorization, and accounting (AAA) logs can be cumbersome to process at times. The tac_plus account file's variable length and tab-delimited format makes generic reporting difficult, thus requiring some kind of customization of the report tool. This script processes the raw accounting file and generates some useful administration reports by parsing the number of fields down to only the relevant data. The parsing is done with AWK, so additional reports can be added easily. Each report is generated as a text file. The script can either direct the output to the terminal display or send it via e-mail to users defined within the script. The six report options are:

- User Requested Disconnects: Reports on user terminal sessions that have been terminated, normally using the <exit> or <logout> command.
- Idle Session Disconnects: Reports on user terminal sessions that have been terminated due to session inactivity.
- Lost Carrier Disconnects: Reports on user terminal sessions terminated due to modem or VTY disconnect.
- Failed logins attempts: Reports on failed user terminal sessions, due to authentication failures.
- Command Accounting: Reports on the commands executed on network access devices (NADs) and who executed them.
- System Event Activity: Reports on NAD system accounting events.

Of the six reports, four are about user access activity. In order to collect all of the data needed to generate the reports properly, the following AAA accounting features must be enabled on the NAD:

```
<aaa accounting exec default start-stop group tacacs+>
<aaa accounting send stop-record authentication failure>
```

The user activity reports are generated using the terminal session STOP accounting records as the search key. With START-STOP accounting enabled, when each terminal session starts (along with every other NAD event) a NAD unique task_id is assigned. (Each NAD generates and assigns its own task_ids, starting with ID 1, and the counter is reset after a reboot. So while the task_id is unique to the NAD, there is the possibility that two NADs could assign the same task_id for different events.) The same task_id is issued with the START and STOP accounting event for each terminal session. For reporting purposes, tracking and reporting in disconnect causes is more valuable from an administration standpoint and ironically easier. Terminal session START events are generic:

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
Thu May 1 16:05:00 2003      172.30.71.1      martin
tty6      172.30.71.4      start
task_id=65      timezone=EDTservice=shell      start_time=1051819498
```

This provides little data, aside from the session start timestamp. However, the STOP record provides a great deal of information:

```
Thu May 1 16:06:30 2003      172.30.71.1      martin
tty6      172.30.71.4      stop      task_id=65      timezone=E
DTservice=shell      start_time=1051819498      disc-
cause=1      pre-session-time=17      elapsed_time=90
```

Specifically, you can find out how long the session was (elapsed_time), how long it took the user to authenticate (pre-session-time) and why the session was terminated (disc-cause). There are 17 "BASE" disconnect-cause attribute value codes for describing why a session was terminated. There are an additional 25 optional "EXTENDED" disconnect cause codes, used in conjunction with the BASE codes, that are used to describe the event that caused the connection to be terminated. The majority of the disconnect codes and extended descriptor codes are esoteric. The script generates reports on the three most common events: User_Request, Idle-Timeout, and Lost Carrier.

The failed user attempt report is dependent on the AAA accounting option <aaa accounting send stop-record authentication failure> being enabled. Unless the network access server (NAS) is configured, failed login attempts are not logged to the accounting server. This is done to avoid the possibility of having the server DOSed by hammering the NAD with bogus logins and the server becoming overloaded or having the accounting file overgrow its file system. Potential attacks aside, this option is a great way to detect unauthorized login attempts that could indicate "recon snooping," which often goes unnoticed in large environments.

The remaining two reports, command and system event accounting reports, are quite straightforward. They report on what is enabled on the NAD in terms of command accounting and generic system events. While ideally each of the reports should be run daily, the command and system reports are really a must if you are interested in being able to track activity on your NADs.

In order for the script to function properly, the following global variables need to be defined within the script:

```
# Location of the report file
report=/var/tmp/tacreport
# User list to mail report results to
USER=anyone@outland.net, somegal@user.net
# Location of accounting file
acf=/var/tmp/acctfile
```

The script will not run without a report definition. If no output option (Mail or Terminal) is defined, the script will just generate the report and exit. The report files are deleted prior to each report run. If you want to save the reports, use the mail option. Here is a complete list of command flags and functions:

- User Requested Disconnects = -ur
- Idle Session Disconnects = -ut
- Lost Carrier Disconnects = -lc
- Failed logins attempts = -lf

TACACS Tricks with Scripts - Part 2

Michael J. Martin

- Command Activity = -a
- System Event Activity = -e

To display the report data to the terminal, use the -c flag following the report definition = taccreport <report flag> -c

To send the report via e-mail, use the -m flag following the report definition = taccreport <report flag> -m

Search the account file by task_id to see the entire accounting record use the -s in combination with the task id following the report flag = taccreport <report flag> -s <task_id>

Review the help information and command options = -h

AAA Accounting File Format

The taccreport script is a great tool for quick reporting. However, if you want to create your own reports, extend the tool, or simply make sense of the data, you will need to know a little something about the accounting attribute values. For starters, every AAA accounting record contains ten logical informational fields, along with a variable number of optional fields. A single record is sent for each event. A single record is made up of both the static and optional AV values sent as a tab-delimited string and written to the accounting file. The logical "static" event fields are:

- Timestamp = Day, month, time, and year
- NAS name = Network access server, the router or device IP address
- Username = The user name associated with the action
- TTY port = The VTY or TTY port the user was connected to on the NAS
- SRC host address = The IP address that the connection was opened from
- Record type = Describes the state change on the NAS expressed as a Start and/or Stop record
- Task_id = Unique ID associated with an event or action. The start and stop records will both have the same task id
- Timezone = The time zone the timestamp was generated in (configured on the NAD)
- Service = The user service used (typically "shell"); PPP, connection, system, and tty-daemon are also available.
- Start_time = When the action started, in seconds since the epoch (12:00AM Jan 1 1970)

The "optional" descriptors fields available for the NAS to report with are:

- Unknown = Used when the user is unknown
- Elapsed_time = Time in seconds taken for the action
- Priv_level = The privilege level associated with the action

TACACS Tricks with Scripts - Part 2

Michael J. Martin

- Protocol = The protocol associated with the event
- Cmd = The command associated with the event
- Bytes_in = The number of input bytes associated with session
- Bytes_out = The number of output bytes associated with session
- Paks_in = The number of input packets associated with session
- Paks_out = The number of output packets associated with session
- Pre-session-time = The length of time from the start of the session to when authentication has been completed
- Disc-cause = The reason a session was terminated, used with start and stop records
- Disc-cause-ext = The specific action that resulted in the session termination (optional)
- Stop_time= When the action started, in seconds since the epoch (12:00AM Jan 1 1970)

Here ends the advanced TACACS+ series. I hope you have found this series helpful and informative. Next month we'll take a look at IOS methods for authenticating network access using lock and key ACLs and the Auth-Proxy. So stay tuned.

Scripts

```
#####  
# checktacplus.sh  
#!/bin/sh  
#  
# Simple script checks to see if tac_plus is running and if it is not  
# it restarts it. The tool can be run from CRON or from the command line  
#  
# Global Variables  
  
TACPID="/etc/tac_plus.pid"  
# Tac_plus daemon location  
TAC="/usr/sbin/tac_plus"  
# Tac_plus default config  
TACCONF="/etc/tacacs.conf"  
  
##### List running tac_plus processes  
  
echo " "  
  
rm -rf /var/tmp/trun  
touch /var/tmp/trun  
  
echo "PID      Service          Config File      Port" > /var/tmp/trun; ps -aux |  
grep $TAC | awk '{print $2"\t" $11, $13,$15}' >> /var/tmp/trun  
  
cat /var/tmp/trun | grep -v "grep"
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
echo " "
##### Tac_Plus running on the standard service port report to the CLI
#####
# Variables

if [ "$1" = "-s" ]
then
more /var/tmp/trun;exit
fi

#if [ "$1" = "" ]
#then
#$TACPID
#fi

# If no PID file exists

if [ "$1" = "" ]
then
ls -A $TACPID 2>/dev/null | grep -c "$TACPID" > /var/tmp/cpid
fi

if [ "$1" = "" ]
then
if [ "`cat /var/tmp/cpid`" = "0" ]
then
$TAC -C $TACCONF -d 32 ;echo "No PID File, Starting tac_plus on port 49, PID `cat
$TACPID`;exit
fi
fi

#
# If PID File Exists

ps -aux | awk '{print $2}' | grep -c "`cat $TACPID`" > /var/tmp/testf

if [ "$1" = "" ]
then
if [ "`cat /var/tmp/testf`" -gt "0" ];
then echo "tac_plus is up PID `cat $TACPID`;exit
elif [ "`cat /var/tmp/testf`" = "0" ];
then
echo "tac_plus is down";$TAC -C $TACCONF -d 32;echo Restarting tac_plus PID `cat
$TACPID`;exit
fi
fi

# Stopping the "default" tac_plus service

if [ "$1" = "-k" ]
then
ps -aux | awk '{print $2}' | grep -c "`cat $TACPID`" > /var/tmp/pcheck
fi

if [ "$1" = "-k" ]
then
if [ "`cat /var/tmp/pcheck`" = "1" ]
then
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
kill -TERM `cat $TACPID`;echo "Shutting Down the tac_plus service PID `cat
$TACPID`;exit
else
echo "The tac_plus process does not exist";exit
fi
fi

# Reload the "default" tac_plus service

if [ "$1" = "-r" ]
then
ps -aux | awk '{print $2}' | grep -c "`cat $TACPID`" > /var/tmp/pcheck
fi

if [ "$1" = "-r" ]
then
if [ "`cat /var/tmp/pcheck`" = "1" ]
then
kill -USR1 `cat $TACPID`;echo "Reloading the tac_plus service PID `cat
$TACPID`;exit
else
echo "The tac_plus process does not exist";exit
fi
fi

##### Tac_Plus running on a non standard service port report to the CLI
#####

# Stopping the non-standard port tac_plus service

if [ "$1" = "-kp" ]
then
TACPID="/etc/tac_plus.pid.$2"
fi

if [ "$1" = "-kp" ]
then
PORT="$2"
fi

if [ "$1" = "-kp" ]
then
if [ "$2" = "" ]
then
echo "The PID value is missing";exit
fi
fi

if [ "$1" = "-kp" ]
then
if [ "`awk '{print $4}' /var/tmp/trun | grep -c $2`" = "0" ]
then
echo PID does not exist;exit
fi
fi

if [ "$1" = "-kp" ]
then
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
ps -aux | awk '{print $2}' | grep -c "`cat $TACPID`" > /var/tmp/nspcheck
fi

if [ "$1" = "-kp" ]
then
if [ "`cat /var/tmp/nspcheck`" = "1" ]
then
kill -TERM `cat $TACPID`;echo "Shutting Down the tac_plus service PID `cat
$TACPID`;exit
else
echo "The tac_plus process does not exist";exit
fi
fi

# Reloading the non-standard port tac_plus service

if [ "$1" = "-rp" ]
then
TACPID="/etc/tac_plus.pid.$2"
fi

if [ "$1" = "-rp" ]
then
PORT="$2"
fi

if [ "$1" = "-rp" ]
then
if [ "$2" = "" ]
then
echo "The PID value is missing";exit
fi
fi

if [ "$1" = "-rp" ]
then
if [ "`awk '{print $4}' /var/tmp/trun | grep -c $2`" = "0" ]
then
echo PID does not exist;exit
fi
fi

if [ "$1" = "-rp" ]
then
ps -aux | awk '{print $2}' | grep -c "`cat $TACPID`" > /var/tmp/nspcheck
fi

if [ "$1" = "-rp" ]
then
if [ "`cat /var/tmp/nspcheck`" = "1" ]
then
kill -TERM `cat $TACPID`;echo "Shutting Down the tac_plus service PID `cat
$TACPID`;exit
else
echo "The tac_plus process does not exist";exit
fi
fi
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
# Starting tac_plus on non-standard port

if [ "$1" = "-p" ]
then
PORT="$2"
fi

if [ "$1" = "-c" ]
then
PORT="$2"
fi

if [ "$1" = "-p" ]
then
TACPID="/etc/tac_plus.pid.`echo $2`"
fi

if [ "$1" = "-c" ]
then
TACPID="/etc/tac_plus.pid.`echo $2`"
fi

if [ "$1" = "-p" ]
then
if [ "$2" = "" ]
then
echo "Missing Port Definition";exit
fi
fi

if [ "$1" = "-c" ]
then
if [ "$2" = "" ]
then
echo "Missing Port Definition";exit
fi
fi

if [ "$1" = "-c" ]
then
if [ "$3" = "" ]
then
echo "Missing configuration file";exit
fi
fi

# Starting tac_plus on non-standatd port if no PID file exists with the default
config file definition

if [ "$1" = "-p" ]
then
if [ "$3" = "" ]
then
if [ "$1" = "-p" ]
then
ls -A $TACPID 2> /dev/null | grep -c $TACPID > /var/tmp/nspid-new
fi

```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
fi
fi

if [ "$1" = "-p" ]
then
if [ "$3" = "" ]
then
if [ "$1" = "-p" ]
then
if [ "`cat /var/tmp/nspid-new`" = "0" ]
then
/usr/sbin/tac_plus -C /etc/tacacs.conf -p `echo $PORT`;echo "No PID File, Starting
tac_plus on port `echo $PORT`, PID `cat $TACPID`;exit
fi
fi
fi
fi

# Starting tac_plus on non-standard port if a PID File Exists using the default
config

ps -aux | awk '{print $2}' | grep -c "`cat $TACPID`" > /var/tmp/nspid 2>$1
/dev/null

if [ "$1" = "-p" ]
then
if [ "$3" = "" ]
then
if [ "$1" = "-p" ]
then
if [ "`cat /var/tmp/nspid`" -gt "0" ];
then echo "tac_plus is up PID `cat $TACPID`;exit
elif [ "`cat /var/tmp/nspid`" = "0" ];
then
echo "tac_plus is down";/usr/sbin/tac_plus -C /etc/tacacs.conf -p `echo
$PORT`;echo "Restarting tac_plus PID `cat /etc/tac_plus.pid`;exit
fi
fi
fi
fi

# Starting tac_plus on non-standatd port if no PID file exists with config file
definition

echo 1

if [ "$1" = "-c" ]
then
ls -A $TACPID 2> /dev/null | grep -c $TACPID > /var/tmp/nspidconf-new
fi

if [ "$1" = "-c" ]
then
if [ "`cat /var/tmp/nspidconf-new`" = "0" ]
then
/usr/sbin/tac_plus -C $3 -p `echo $PORT`;echo "No PID File, Starting tac_plus on
port `echo $PORT`, PID `cat $TACPID`;exit
fi
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
fi

#####
echo 2

if [ "$1" = "-c" ]
then
ls -A $TACPID 2> /dev/null | grep -c $TACPID > /var/tmp/nspidconf-new
fi

echo 3

if [ "$1" = "-c" ]
then
if [ "`cat /var/tmp/nspidconf-new`" = "1" ]
then
ps -aux | awk '{print $2}' > /var/tmp/confpid
fi
fi

if [ "$1" = "-c" ]
then
grep -c `cat $TACPID` /var/tmp/confpid > /var/tmp/nspidconf-ext
fi

if [ "$1" = "-c" ]
then
if [ `cat /var/tmp/nspidconf-ext` = "1" ]
then
echo "tac_plus listening on port `echo $2` Is Up PID `cat $TACPID` ";exit
fi
fi

if [ "$1" = "-c" ]
then
if [ `cat /var/tmp/nspidconf-ext` = "0" ]
then
echo "tac_plus is down";/usr/sbin/tac_plus -C $3 -p `echo $PORT`;echo "Restarting
tac_plus istening on port `echo $2` PID `cat /etc/tac_plus.pid`"
fi
fi

echo 5

if [ "$1" = "-f" ]
then
echo "clearing PID files"; rm -rf /etc/tac_plus.pi*;exit
fi

if [ "$1" = "-h" ]
then
clear;echo "

##### checktacplus #####

Here is a complete list of command flags and functions:

Start the default service = checktacplus
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

Stop the default service = checktacplus Dk

Reload the configuration of the default service = checktacplus -r

Start an instance running on a non-standard port, using the default configuration = checktacplus Dp <port>

Start an instance running on a non-standard port, using an alternative configuration file = checktacplus Dc <port> <configuration file>

Reload an instance running on a non-standard port = checktacplus Dpr

Stop an instance running on a non-standard port = checktacplus Dkp

Review the help information and command options = -h

```
" | more
fi;exit
```

```
#####
```

```
# tacdebug.sh
```

```
#!/bin/sh
```

```
# tac_plus configuration file location
```

```
CONF=/etc/tacacs.conf
```

```
# daemon location
```

```
TAC=/usr/sbin/tac_plus
```

```
# LOGFILE location
```

```
LOG=/var/tmp/tac_plus.log
```

```
# PID location
```

```
TACPID="/etc/tac_plus.pid"
```

```
#####
```

```
rm -rf $LOG
```

```
if [ "$1" = "" ]
```

```
then
```

```
TACPID="/etc/tac_plus.pid"
```

```
fi
```

```
if [ "$1" = "-k" ]
```

```
then
```

```
kill -9 `cat $TACPID`;echo "Shutting Down the tac_plus service PID `cat $TACPID`";exit
```

```
fi
```

```
# If no PID file exists
```

```
if [ "$1" = "" ]
```

```
then
```

```
ls -A /etc/tac_plus.pid 2>/dev/null | grep -c "$TACPID" > /var/tmp/cpid
```

```
fi
```

```
if [ "$1" = "" ]
```

```
then
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
if [ "`cat /var/tmp/cpid`" = "0" ]
then
/usr/sbin/tac_plus -C /etc/tacacs.conf;echo "No PID File, Starting tac_plus on
port 49, PID `cat $TACPID`;exit
fi
fi

#
# If PID File Exists

ps -aux | awk '{print $2}' | grep -c "`cat $TACPID`" > /var/tmp/dbugpid

if [ "$1" = "" ]
then
if [ "`cat /var/tmp/dbugpid`" -gt "0" ];
then echo "tac_plus is up PID `cat /etc/tac_plus.pid`;exit
elif [ "`cat /var/tmp/dbugpid`" = "0" ];
then
echo "tac_plus is down";$TAC -C $CONF;echo Restarting tac_plus PID `cat
/etc/tac_plus.pid`;exit
fi
fi

if [ "$1" = "-h" ]
then
echo "
This script provides a wrapper starting the tac_plus TACACS+
daemon. In default mode (no flags) the deamon starts with no
debug options. The debug options are addative each mode inherets
the previous modes, yielding the debug output of all of the

-k Stop the service

-par Start service parse configuration file to logfile

-for Start service in fork mode

-auz Start service in authorization debug mode

-aut Start service in authentication debug mode

-pas Start service in password debug mode

-cfg Start service in config debug mode

-pak Start service in packet debug mode

-md5 Start service in MD5 debug mode

-enc Start service in Low Level Encrypt/Decrypt debug mode

-pro Start service in proxy debug mode

-max Start service in max-session debug mode (if compiled)
" | more
fi
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
if [ "$2" = "-C" ]
then
CONF="$3";echo "Using Alternative Configuration File"
fi

if [ "$1" = "-par" ]
then
$TAC -C $CONF -d 2;echo "Starting In Parse Mode";more $LOG
fi

if [ "$1" = "-for" ]
then
$TAC -C $CONF -d 4;echo "Starting In Fork Mode";tail -f $LOG
fi

if [ "$1" = "-auz" ]
then
$TAC -C $CONF -d 8;echo "Starting In Authorization Debug Mode";tail -f $LOG
fi

if [ "$1" = "-aut" ]
then
$TAC -C $CONF -d 16;echo "Starting In Authentication Debug Mode";tail -f $LOG
fi

if [ "$1" = "-pas" ]
then
$TAC -C $CONF -d 32;echo "Starting In Password File Processing Debug Mode";tail -f
$LOG
fi

if [ "$1" = "-acc" ]
then
$TAC -C $CONF -d 64;echo "Starting In Accounting Debug Mode";tail -f $LOG
fi

if [ "$1" = "-cfg" ]
then
$TAC -C $CONF -d 128;echo "Starting In Config File Parsing And Lookup Debug
Mode";tail -f $LOG
fi

if [ "$1" = "-pak" ]
then
$TAC -C $CONF -d 256;echo "Starting In Packet Level Debug Mode";tail -f $LOG
fi

if [ "$1" = "-hex" ]
then
$TAC -C $CONF -d 512;echo "Starting In Authentication and Authorization Debug
Mode";tail -f $LOG
fi

if [ "$1" = "-md5" ]
then
$TAC -C $CONF -d 1024;echo "Starting In MD5 Hash Debug Mode";tail -f $LOG
fi
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
if [ "$1" = "-enc" ]
then
$TAC -C $CONF -d 2048;echo "Starting In Low Level Encrypt/Decrypt Debug Mode";tail
-f $LOG
fi

if [ "$1" = "-pro" ]
then
$TAC -C $CONF -d 16348;echo "Starting In Proxy Debug Mode";tail -f $LOG
fi

if [ "$1" = "-max" ]
then
$TAC -C $CONF -d 32768;echo "Starting In Max Session Debug Mode";tail -f $LOG
fi
```

```
#####
# tacreport.sh
#!/bin/sh
#
# Global Variables
# Location of the report file
report=/var/tmp/tacreport
# User list to mail report results to
USER=anyone@outland.net
# Location of accounting file
acf=/var/tmp/acctfile

##### End Global Variables#####

# Report search values

dc1=disc-cause=1
dc4=disc-cause=4
dc2=disc-cause=2
dc17=disc-cause=17

# Delete the report before starting the run

rm -rf $report

##### Script Starts Here #####

if [ "$1" = "" ]
then
echo "No report type defined see help -h for options";exit
fi

if [ "$1" = "-h" ]
then
echo "This script processes the tac_plus accounting and generates
reports on session disconnects, IOS command usage and system events.
While quite useful, AAA accounting logs can be cumbersome at times.
This script processes the file down into some useful general reports
and provides a search option, using the task_id value, to see the
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

complete account record. There are six report options:

1. User Requested Disconnects
2. Idle Session Disconnects
3. Lost Carrier Disconnects
4. Failed Logins Attempts
5. Command Activity
6. System Event Activity

The script will not run without a report definition. If no output option (Mail or Terminal) is defined the script will just generate the report and exit. The command flags are:

User Requested Disconnects = -ur

Idle Session Disconnects = -ut

Lost Carrier Disconnects = -lc

Failed logins attempts = -lf

Command Activity = -a

System Event Activity = -e

To display the report data to the terminal use the -c flag following the report definition

```
taccreport <report flag> -c
```

To send the report via e-mail use the -m flag following the report definition

```
taccreport <report flag> -m
```

To search the account file by task_id to see the entire accounting record use the -s in combination with the task id following the report flag:

```
taccreport <report flag> -s <task_id>
```

To review the help information and command options use the -h flag

```
taccreport -h
```

```
" | more ;exit  
fi
```

```
# Report on Disconnects that are user initiated
```

```
if [ "$1" = "-ur" ]  
then
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
echo User Requested Disconnect Report for `date` > $report;echo " " >>
$report;echo "TaskID          Time          NAS          User" >>
$report;echo " " >> $report
fi

if [ "$1" = "-ur" ]
then
grep "stop" $acf | grep "$dc1" | awk '{print $11,"\t""\t"$4,$2,$3,"\t"$6,"\t"$7}'
>> $report
fi

# Report on Disconnects that are a result of a idle session timeout

if [ "$1" = "-ut" ]
then
echo User Idle Session Timeout Disconnect Report for `date` >> $report;echo " " >>
$report
fi

if [ "$1" = "-ut" ]
then
grep stop $acf | grep $dc4 | awk '{print $11,"\t"$4,$2,$3,"\t"$6,"\t"$7}' >>
$report
fi

# Report on Disconnects that are a result from a Lost Carrier event

if [ "$1" = "-lc" ]
then
echo Lost Carrier Disconnect Report for `date` >> $report;echo " " >> $report
fi

if [ "$1" = "-lc" ]
then
grep stop $acf | grep $dc2 | awk '{print $11,"\t"$4,$2,$3,"\t"$6,"\t"$7}' >>
$report
fi

# Report on failed logins

if [ "$1" = "-lf" ]
then
echo Failed Login Attempts Report for `date` >> $report;echo " " >> $report;echo
"Task ID - Time/Date - NAS - SRC HOST - USER">> $report;echo " " >> $report
fi

if [ "$1" = "-lf" ]
then
grep stop $acf | grep $dc17 | awk '{print $11,"\t"$4,$2,$3,"\t"$6,"\t"$9"\t"$7}'
>> $report
fi

# Generate a Command Activity Report

if [ "$1" = "-a" ]
then
echo Command activity report for `date` >> $report;echo " " >> $report;echo "Task
ID - User - NAS - Command" >> $report;echo " " >> $report
```

TACACS Tricks with Scripts - Part 2

Michael J. Martin

```
fi

if [ "$1" = "-a" ]
then
more /var/tmp/acctfile | grep cmd | awk '{print $11"\t"$7, $9, $16,$17,$18,$19,
$20, $21,$22}' >> $report
fi

# Generate a System Event Report

if [ "$1" = "-e" ]
then
echo System Event Report for `date` >> $report;echo " " >> $report;echo "Task ID -
User - NAS - Event - Reason" >> $report;echo " " >> $report
fi

if [ "$1" = "-e" ]
then
more /var/tmp/acctfile | grep service=system | awk '{print
$11"\t"$7,$6,$13,$14,$15 }' >> $report
fi
# Report Output

if [ "$2" = "-c" ]
then
more $report
elif [ "$2" = "" ]
then
exit
fi

if [ "$2" = "-m" ]
then
cat $report | mail -s "`date +%b%d` - NAS session report" $USER
fi

if [ "$2" = "-s" ]
then
if [ "$3" = "" ]
then
echo "Missing Search Attribute";exit
fi
fi

if [ "$2" = "-s" ]
then
grep "task_id=$3" $acfb
fi
```