

Building A Secure TACACS+ Environment

Michael Martin

First in the three-part series, "Using TACACS+ to manage Cisco networking devices ."

Enterprise networks today are as complex as the computer systems that run on them. To manage user access and command/event auditing on network service devices (NSDs -- routers, switches and network access servers such as dial-in and VPN servers), one of the best things a network manager can do is to implement a system auditing and accounting system (SAA) to centralize these activities. This will be the first of three articles on how to implement a cost-effective and secure SAA system for Cisco networks based on Terminal Access Controller Access Control (TACACS). In this article, we will deal with creation and basic configuration of a Linux-based SAA server. Part Two will cover configuration options for secure access to Cisco routers and switches in a multi-user access environment. In Part Three, we will look at strategies for configuring SAA systems -- specifically, server configuration, operation and event report generation.

What Is System Auditing And Accounting?

System, or "user" accounting is the activities involved with the adding and removal of user login accounts and system access privileges. System auditing is monitoring of activities that occur on a computer network device. There are several types of audits that can be performed. Security, command, and process are the most common forms of system auditing that are implemented. Regardless of "scope," the method for auditing is basically the same. Event data is collected, collated and reported on. Collation and reporting are tied to two key data elements -- user id and process.

The Problem With System Auditing And Accounting

The problem with SAA is that managing users on a few systems is chore at best and a little tedious at worst. On hundreds or thousands of systems, it is an outright daunting task. The heart of the problem is that users do not like passwords, so to have a different account and password on each system is seen as quite a pain. To ease this pain, users choose passwords that are usually quite weak and easy to remember (if they're not just on a sticky note stuck to the monitor). In a large-scale computing environment, it is just not practical to manage user accounts and passwords individually on hundreds of systems. So, in these cases one of two scenarios develops. The "right" way is to implement a centralized SAA system. The "wrong" way is to create a single login and share it with everyone who needs access to the systems.

Does The "Wrong" Way Remind You Of Something?

If you're logging on to your PC through an NT domain controller or using NIS (Network Information System) or Kerberos to log in to your Unix servers, an SAA system is controlling access to the computing infrastructure. What about the network infrastructure; routers, switches, out-of-band terminal servers? What is controlling access and monitoring these vital components? Who last accessed the Internet router and what commands were executed? In large-scale network environments, the network infrastructure is constantly being accessed. Computer moves, adds and changes (MACs), router interface address changes, modifications to routing policy, or VLAN creation can all have dramatically adverse effects on the network if made incorrectly.

Who exactly is accessing these essential systems? Internal support staff, external service providers, contractors, and vendors, all making changes. Think about it, the most essential system in your environment is protected by two passwords that have been shared with countless people making changes you have no audit trail for. To ensure accountability (so you know who did what) and auditability (so you can know what to undo) of your network components, an SAA system is a must-

Building A Secure TACACS+ Environment

Michael Martin

have. And if your network is comprised of Cisco hardware, then the SAA for you is TACACS. (Extreme, Riverstone, and Foundry Networks products also support TACACS+ authentication.)

TACACS, XTACACS, and TACACS+

TACACS is a client/server protocol designed to provide user accounting services for network access servers (NAS). Cisco Systems, the developer of TACACS, has released three iterations of the protocol. The original version of the protocol is known as TACACS, and it is implemented in Cisco's CiscoSecure v1.x product. The update is known as X-TACACS (Extended-TACACS) and it is defined in IETF RFC 1492. Both TACACS and XTACACS use UDP (port 49) for transport and provide basic SAA services. TACACS uses Attribute=Value (AV) pairs to define user and service access operational parameters. The NAS and the TACACS daemon communicate by exchanging the AV pairs to relay AAA requests and response. The original motivator for the development of TACACS was to provide authentication, authorization and accounting (AAA) services for dial-in network access servers. *Authentication* provides user identification and determines system access. *Authorization* determines what the user is permitted to do on the network or on a given system. *Accounting* logs what the user did when logged on to the network or system.

TACACS+ (CiscoSecure2.0) is the currently supported version; the IETF RFC is in draft status and is available at <ftp://ftp-eng.cisco.com/pub/tacacs/tac-rfc.1.78.txt>. TACACS+ is a complete rewrite of the TACACS protocol. It provides extensive AAA services for routers and switch access control and accounting, and extends NAS support for VPN services. TACACS+ uses TCP (port 49) for reliable transport, MD5 encryption for messages and supports "hooks" for offloading services data stores to the database management system. TACACS+ is available as the commercial product CiscoSecure 2.x, which runs on NT/2000 and Unix. There is also a "development" implementation available freely from Cisco at ftp://ftp-eng.cisco.com/pub/tacacs/tac_plus.F4.0.4.alpha.tar.Z.

The Components Of An SAA System

The SAA solution purposed here is implemented to provide a secure method for accessing network components. To meet this end, we need to ensure that the communication channel between the NSD and the SAA daemon is secure, and that the communication channel between the user, the NSD and the SAA server is secure. Lastly, we need to ensure that user passwords, the front line of our security system, cannot be easily compromised.

Secure NAS To Daemon Communication

TACACS+ uses MD5 hashing to encrypt NAS to daemon communication messages, which is a step up from the earlier versions of TACACS and its alternative RADIUS, which offers no message encryption. Be aware there are some security concerns with the TACACS+ protocol. Notably, concern about the protocol's susceptibility to "man in the middle" attacks creating a potential for undetectable corruption of messages. To see a complete analysis, take a look at what the Openwall security auditor came up with at http://www-arc.com/sara/cve/tacacs_server.html. Overall, TACACS+ is not a perfect solution, but it is better then the previous versions and current alternatives like RADIUS.

One of the great things about the Open Source movement and Linux is that if the product does not do everything you need it to, you can add what you need. In this spirit, if someone has added something really cool to something that already exists, they will typically make it available. There are a number of enhancements to the Cisco TACACS+ development daemon available on the Net. The folks at Shrubbery.net have added some great security enhancements; the code is available at

Building A Secure TACACS+ Environment

Michael Martin

ftp://ftp.shrubbery.net/pub/tac_plus/tac_plus.F4.0.4.alpha+acl+libwrap.tar.gz/. This makes their "distribution" a great choice for our SAA system. The Shrubbery enhancements provide you with:

1. The ability to limit NAS access to the daemon using libwrap, from Wietse Venema's TCP Wrappers. The source is available at <ftp://ftp.porcupine.org/pub/security/index.html/>.
2. The ability to permit or deny (by user or group) access to NAS's using REGEX-based access control lists.
3. The daemon also expands the TACACS+ ability to provide password authentication by adding the ability to define and enable a password for a user or group.

These features extend the capabilities of the network administrator to control access to the server level and create hierarchical user and group login and enable access policies at the NSD level.

A Secure Communication Channel

Network-based login is a relatively modern convention. Early multi-user systems used hardwired terminals so session information was relatively secure, which is to say the password represented the largest security vulnerability. When you add communication across ubiquitous meshed networks, user and password information can be captured with moderate ease, leaving us with a large security risk. To address potential communication snooping and ineffective passwords, we need to strengthen these vulnerabilities with encryption.

To provide secure communications channel between other hosts, NSDs, and the TACACS+ server, Secure Shell (SSH) is an answer. SSH provides encrypted virtual terminal access, file transfer and TCP client tunneling. SSH supports a number of authentication options including standard password login over a secure channel, public key encryption, and Diffie-Hellman. There are two versions of SSH, v1 and v2. They are functionally the same, architecturally different, and both are IETF RFC drafts. SSHv1 has broad client support and is the more prevalent of the two. SSHv1 server daemons are available for the majority of Unix platforms. SSHv1 server support is also available on Data Encryption Standard (DES) and 3DES 12.x Cisco IOS versions. Initially, support for SSHv2 was only available with commercial products. Until OpenSSH became available, a free implementation, providing client and server support for both SSHv1 and SSHv2 on both Unix and Windows was available through Cygwin at <http://sources.redhat.com/cygwin/>. Source and binary distributions are downloadable at <http://www.openssh.com/>.

Password Integrity Using One-Time Passwords

To defend against password compromises, a one-time password system (OTP) is the way to go. Bellcore's S/KEY OTP system fits the bill for our SAA system. There are a number of freely available S/KEY implementations available; the OpenBSD S/KEY distribution is stable and provides support for MD4 (s/key spec default) and MD5. The source is available at <http://www.sparc.spb.su/solaris/skey/skey-1.1.5.tar.gz/>. Wietse Venema also provides an S/KEY implementation as part of his logdaemon distribution, which provides S/KEY applications, S/KEY supported login, ftp and telnet daemons, available at <ftp://ftp.porcupine.org/pub/security/index.html/>.

Building A Secure TACACS+ Environment

Michael Martin

S/KEY works using a non-reversible MD4 or MD5 hash of a secret password. The user initializes S/KEY by accessing the system and executing the skeyinit key generation tool. Here is an example of the key initialization process (using OpenBSD distribution):

```
thor@hammer$ skeyinit -md4
```

```
Adding thor:
```

```
Reminder -- Only use this method if you are directly connected. If you are  
using telnet or rlogin exit with no password and use keyinit -s.
```

```
Enter secret password:
```

```
Again secret password:
```

```
ID thor s/key is 99 1e49885
```

```
AWE JOAN JAG LOON PAD PEP
```

```
thor@hammer bin$
```

At initialization, user derived secret password is hashed 99 times using the MD4 or MD5 hash algorithm. The last key of the run (in this case number 99) is then stored in the S/KEY password file /etc/skeykeys. When the user logs in, S/KEY asks for the next recursive password to the one has stored in /etc/skeykeys. Look at this login example:

```
login: thor
```

```
s/key 98 1e49885
```

```
Password:
```

In this example, since it is the first login, it uses the hash that generated when the user initialized S/KEY. Here is what the /etc/skeykeys file looked like before the user authenticated:

```
thor 0099 1e49885 04b43c7d59231866 Jan 08,2002 23:18:16
```

The user password response can be calculated using an S/KEY calculator or the user could print out a list of OTPs. (See the end of this article for a list of S/KEY calculator sources.) The password is calculated using the hash number, salt, derived form the user login or hostname (depending on implementation) and the user's secret password. The result is 6 strings of text characters or words:

```
LIFE BRAN OATH ROOD BOGY TUM
```

Once the password is calculated and entered, it is hashed and compared to the stored key. If they match, your login is successful. S/KEY then updates /etc/skeykeys with the hash you just authenticated with, to use as a comparison for the next login. You can see that from looking at the /etc/skeykeys file:

```
thor 0098 1e49885 6c84eb77dbb3b69a Jan 09,2002 00:15:26
```

When the key count reaches zero, you will be unable to login. So prior to reaching zero, you will want to generate a new key sequence. By default, 99 keys are generated. It is possible to generate more at initialization using the [-n count] flag where count is 1 to 10,000, indicating the number of keys you want generated. To generate new keys run the skeyinit with the [-s] flag.

Building A Secure TACACS+ Environment

Michael Martin

Building your SSA System

The assumption here is that you have loaded Linux onto the server, it is up and running, and it has access to the Internet. To start building the server, you will need to download each of the packages and build and install them in the following order:

Step 1. Building TCP Wrappers

TCP wrappers (TCPD) is an application that allows us to permit or deny access or to log and respond to an unauthorized access to any "wrapped" inetd service. Inetd is the "super process" responsible for launching network services like telnet and ftp. Functionally, TCPD operates as a proxy and an access control filter. Tcpsd "listens" for incoming connections to the wrapped service's port. Connections are filtered, and those with permitted origin addressing are handed off the service daemon. Many network service daemons also provide the ability to use the "wrapper: permit/deny" facility for access control when running as a standalone service. Access control for wrapped services are managed using two files: /etc/hosts.allow and /etc/hosts.deny. These files make up the permit/deny rule base, which can be defined for all services, or on a service-by-service basis. Filtering definitions can use either hostnames/ipaddresses to define access for a single hosts or domain names or IP prefixes to define access for groups.

Building the TCPD daemon is easy and comes with good documentation. There are also numerous resources on TCP wrappers available on the web. Here are the steps of building TCPD and its supporting applications:

Extract the source:

```
tar xvfz tcp_wrappers_7.6.tar.gz
cd tcp_wrappers_7.6
```

Edit the Makefile, and uncomment the STYLE option to enable the TCP wrappers language extension option, which gives you the ability to have TCPD execute commands and print message banners when users log in to services (see the hosts_options.5 man page for all of the options).

```
STYLE = -DPROCESS_OPTIONS # Enable language extensions.
```

The second change is a custom option, which is to direct TCPD syslog messages to a specific log file. This has a few steps. First, comment out the FACILITY definition and add a new one. In the example below, event messages are sent to syslog facility LOCAL7; the priority should be left unchanged at INFO.

```
#FACILITY= LOG_MAIL # LOG_MAIL is what most sendmail daemons use
FACILITY= LOG_LOCAL7 # local7 reporting requires that you edit syslog.conf
```

Now save your edits and close the Makefile. To have syslogd direct the messages to the correct file, the log file needs to be created and the /etc/syslogd.conf file needs to be edited. To create the log file type the following:

Building A Secure TACACS+ Environment

Michael Martin

```
touch /var/log/tcpd
```

This creates the log file var/log/tcpd. Then edit the /etc/syslog.conf file and add the following line:

```
local7.info /var/log/tcpd
```

Close the /etc/syslog.conf file and restart syslog, Now, build the daemon:

```
make REAL_DAEMON_DIR=/usr/sbin linux
```

Once the build is completed, install the daemon and supporting apps.

```
cp -f safe_finger /usr/sbin
cp -f tcpd /usr/sbin
cp -f tcpdchk /usr/sbin
cp -f try-from /usr/sbin
cp -f libwrap.a /usr/lib
```

Now TCPD and libwrap.a are installed, so we are ready to move on to building S/KEY.

Step 2. Building S/KEY

The S/KEY build is very straightforward, just use the following commands:

```
tar xvfz skey-1.1.5.tar.gz
cd skey-1.1.5.
./configure --prefix=/usr --sysconfdir=/etc
make
make install
```

Once installed, the S/KEY applications will be installed in /usr/bin, the libskey.a will be installed in /usr/lib, and the skeykeys file will be located in /etc.

Step 3. Building SSH

Once SSH is installed, there is no reason to install ftp or telnet on the server. If you are using a Red Hat "workstation" install, you will have the ftp and telnet clients but not the service daemons (by default, no network services are configured). The SSH daemon will support TCPD-style host filtering using the /etc/hosts.allow and /etc/hosts.deny files. By default, all hosts are permitted access. To restrict access to localhost and by IP subnet prefix, edit the hosts.allow and hosts.deny files as follows (change the addresses to match you network requirements):

```
# hosts.allow
# service : access control (ALL, {ip prefix}, host or domain name, {host ip
# address}
#
sshd : LOCAL, 192.168.100, 192.168.101, 192.168.2.1

#hosts.deny
```

Building A Secure TACACS+ Environment

Michael Martin

```
#  
ALL:ALL
```

The SSH build takes some time, but is also easy. To start, OpenSSL must be installed. The build process is as follows:

```
tar xvfz openssl-0.9.6c.tar.gz cd openssl-0.9.6c  
./config -prefix=/usr  
make install
```

OpenSSH is dependent on OpenSSL, so to install the package or compile the source you will need to get OpenSSL at <http://www.openssl.org/>. Specifically, you need the OpenSSL header files that will be installed in /usr/include/openssl, and libssl.a for installation in /usr/lib. If you do not use the [-prefix] config option flag as used in the example above, the library and header files will be installed in /usr/local/include and /usr/local/lib.

Once OpenSSL is installed, OpenSSH can be built. Like the OpenSSL build, we need to define some configuration parameters to include support for S/KEY OTP and TCP wrappers support. Here is the build process:

```
tar xvfz openssh-3.0.2p1.tar.gz  
cd openssh-3.0.2p1  
./configure --prefix=/usr --sysconfdir=/etc --with-tcp-wrappers=/usr/lib /  
--with-skey=/usr/lib  
make install
```

The build will install the SSH client programs in /usr/bin, configuration file and host keys in /etc, and the daemon in /usr/sbin. To have sshd launch at boot, edit /etc/rc.d/rc.local and add the following:

```
Echo Starting sshd  
/usr/sbin/sshd
```

Save your changes to /etc/rc.d/rc.local, and the SSH install is complete.

Step 4. Building the TACACS+ Daemon

The last step is to build the TACACS+ daemon. Once you have downloaded the source from Shrubbery.net and expanded the tar file, you can proceed.

```
tar xvfz tac_plus.F4.0.4.alpha+acl+libwrap.tar.gz  
cd tac_plus.F4.0.4.alpha+heas/
```

The distribution Makefile needs to be edited somewhat. Specifically, the OS compiler options, library paths, and some operational parameters need to be defined. By default, the Makefile is configured to build a daemon for Solaris. To build the definition to Linux, the Solaris options need to be commented out:

Building A Secure TACACS+ Environment

Michael Martin

```
# For Solaris (SUNOS 5.3, 5.4, 5.5, 5.6) uncomment the following two lines
#OS=-DSOLARIS
#OSLIBS=-lsocket -lnsl
```

And the Linux definitions need to be uncommented. Since we are running Red Hat 6.x, use the second Linux definition.

```
# For LINUX
# OS=-DLINUX
#
# On REDHAT 5.0 systems, or systems that use the new glibc,
# you might instead need the following:
OS=-DLINUX -DGLIBC
OSLIBS=-lcrypt
```

Next, you need enable S/Key functionality and define where your S/Key library and include files are. Make the following edits:

```
# Definitions for SKEY functionality DEFINES = -DSKEY
LIBS = /usr/lib/libskey.a
INCLUDES =-I../usr//include
```

Once S/Key is defined, you need to define where your libwrap.a library is located. If you followed the install steps so far, use the following path values:

```
# Use tcp_wrappers to authenticate connecting host(s)
TCPWRAP = -DLIBWRAP
LIBWRAP = -L/usr/lib -lwrap
```

After S/KEY, the process id (PID) file location needs to be defined. Select the /etc directory option.

```
#PIDFILE = -DTAC_PLUS_PIDFILE="/var/run/tac_plus.pid"
PIDFILE = -DTAC_PLUS_PIDFILE="/etc/tac_plus.pid"
```

The last edit defines the installation paths for the tac_plus daemon and the man page, which are installed in /usr/local by default. Change the daemon install location to /usr/sbin (which is where all of the other network service daemons are located) and the man page to /usr/man/man1.

```
install:
cp tac_plus /usr/sbin
cp tac_plus.1 /usr/man/man1/tac_plus.1
```

Save your changes to the Makefile then run:

```
make
make install
```

Now that the daemon is installed, we need to configure the system to run the service at boot. We will

Building A Secure TACACS+ Environment

Michael Martin

follow the same steps we used for configuring the sshd daemon to load at boot time. Edit /etc/rc.d/rc.local and add the following:

```
echo Starting TACACS....  
/usr/sbin/tac_plus -C /etc/tacacs.conf -d 16
```

Save your changes to /etc/rc.d/rc.local. The daemon can be launched from the command line, but before we can do so we need to create a configuration file. The TACACS+ distribution comes with extensive documentation that you should review. In Part Three of this series, we will look at advanced daemon configuration. To get the daemon started, we need to create a simple configuration file that will define the TACACS server/client key, accounting file location, and a single login user.

```
#Base Configuration File For TACACS+ development version  
#  
# Server/Client Shared Key  
key = testkey  
#  
# Accounting File  
accounting file = /var/tmp/acctfile  
#  
# Users  
  
user = testuser {  
login = cleartext "password"  
}  
  
# Global enable password  
user = $enabl5$ {  
login = cleartext "enablepswd"  
}
```

Now save the file as /etc/tacacs.conf. We're ready to launch the daemon:

```
/usr/sbin/tac_plus -C /etc/tacacs.conf -d 16
```

The [-C] defines where the configuration file is located, and the [-d 16] flag turns on debugging and sets the debugging level. Output is sent to /var/tmp/tac_plus.log. Here is the debug output indicating a successful launch of the daemon:

```
Thu Jan 10 11:28:51 2002 [30374]: Reading config  
Thu Jan 10 11:28:51 2002 [30374]: Version F4.0.4.alpha Initialized 1  
Thu Jan 10 11:28:51 2002 [30374]: tac_plus server F4.0.4.alpha starting  
Thu Jan 10 11:28:51 2002 [30375]: Backgrounded  
Thu Jan 10 11:28:51 2002 [30376]: uid=0 euid=0 gid=0 egid=0 s=0
```

To verify the daemon is running, you can use the following command string:

Building A Secure TACACS+ Environment

Michael Martin

```
ps -aux | grep `cat /etc/tac_plus.pid`
```

Conclusion

With the server up and running, the next step is to configure the routers and switches to use TACACS+ authentication and accounting, which we will look at in Part Two of this series. While this is sizable task, you will find the implementation of an SAA will become a valuable asset to your ability to securely manage your network infrastructure. So, stay tuned.