Kerberos Network Security in the Solaris™ Operating Environment

---

By Wyllys Ingersoll - SOE Network Security

Sun BluePrints™ OnLine - October 2001

http://www.sun.com/blueprints

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 816-1952-10
Revision 1.1, 10/30/01
Edition: October 2001

Please
Recycle

Kerberos Network Security in the Solaris™ Operating Environment

Kerberos is a network authentication protocol designed to provide strong authentication for client/server applications by using secret-key cryptography. Originally developed at the Massachusetts Institute of Technology, it has been included as part of the Solaris™ Operating Environment (Solaris OE) to provide strong authentication for Solaris OE network applications.

In addition to providing a secure authentication protocol, Kerberos also offers the ability to add privacy support (encrypted data streams) for remote applications such as `telnet`, `ftp`, `rsh`, `rlogin`, and other common UNIX® network applications. In the Solaris OE, Kerberos can also be used to provide strong authentication and privacy support for Network File System (NFS) filesystems, allowing secure and private file sharing across the network.

Because of its widespread acceptance and implementation in other operating systems, including Windows 2000, HP-UX, and Linux, the Kerberos authentication protocol can interoperate in a heterogeneous environment allowing users on machines running one OS to securely authenticate themselves on hosts of a different OS.

This article describes how to securely configure Kerberos in the Solaris OE beyond what is described in the online documentation or what is distributed with the Kerberos software.

Obtaining Kerberos Software

Kerberos software is available for Solaris OE versions 2.6, 7, and 8 in a separate package called the Sun Enterprise Authentication Mechanism™ software. For Solaris 2.6 and Solaris 7 OE, Sun Enterprise Authentication Mechanism software is included

as part of the Solaris™ Easy Access Server 3.0 (Solaris EAS) package. For Solaris 8 OE, the Sun Enterprise Authentication Mechanism software package is available with the Solaris 8 OE Admin Pack.

For Solaris 2.6 and Solaris 7 OE, the Sun Enterprise Authentication Mechanism software is available as part of the Solaris Easy Access Server 3.0 (Solaris EAS) package. Details about the package are here: `http://www.sun.com/software/solaris/7/ds/ds-seas`. It can be ordered from your local Sun sales representative, authorized reseller, and Solaris OEM Partners. This package is not available for download.

For Solaris 8 OE systems, Sun Enterprise Authentication Mechanism software is available in the Solaris 8 OE Admin Pack, available for download from `http://www.sun.com/bigadmin/content/adminPack/index.html`.

All of these Sun Enterprise Authentication Mechanism software distributions are based on the MIT KRB5 Release version 1.0. The client programs in these distributions are compatible with later MIT releases (1.1, 1.2) as well as with other implementations that are compliant with the standard.

Adding Encryption Support Modules

The Sun Enterprise Authentication Mechanism software packages available for download for Solaris 8 OE provides only the authentication mechanisms, no data-encryption functionality is provided. Kerberos provides a secure authentication mechanism by definition, but most Kerberos distributions also provide the ability for users to exchange their data privately and securely as well. The privacy enhancing packages are available for Solaris 8 OE in a separate, downloadable package called the “Solaris Encryption Pack” (available at `http://www.sun.com/solaris/encryption`). It is strongly recommended that all sites wishing to incorporate Kerberos security, download the encryption pack and install it. The privacy support provided by the encryption pack greatly enhances the security offered by the Sun Enterprise Authentication Mechanism software package because it enables users to use encrypted remote applications such as `telnet`, `ftp`, and `rlogin`, and also allows systems to use the NFS protocol with privacy protection. Additionally, the encryption package is necessary to secure administrative tools such as `kpasswd` and `kadmin` so that the principal database may be administered securely.

Installing and Securing the Kerberos Servers

The Kerberos Key Distribution Center (KDC) is a trusted server that issues Kerberos "tickets" to clients and servers to communicate securely. A Kerberos "ticket" is a block of data which is presented as the users credentials when attempting to access a Kerberized service. A ticket contains information about the user's identity and a temporary encryption key, all encrypted in the server's private key. In the Kerberos environment, any entity that is defined to have a Kerberos identity is referred to as a "principal."

A principal may be an entry for a particular user, host, or service (such as NFS or FTP) that is to interact with the KDC. Most commonly, the KDC server system also runs the Kerberos Administration Daemon, which handles administrative commands such as adding/deleting/modifying principals in the Kerberos database. Typically, the KDC, the admin server, and the database are all on the same machine, but they can be separated if necessary. Some environments may require that multiple realms be configured with master KDCs and slave KDCs for each realm. The principles applied for securing each realm and KDC should be applied to all realms and KDCs in the network to ensure that there isn't a single "weak link" in the chain.

One of the first steps to take when initializing your Kerberos database, is to create it using the `kdb5_util` command. When running this command, the user has the choice to create a `stash` file or not. The `stash` file is a local copy of the master key that resides on the KDC's local disk. The master key contained in the `stash` file is generated from the master password that the user enters when first creating the KDC database. The `stash` file is used to authenticate the KDC to itself automatically before starting the `kadmind` and `krb5kdc` daemons (e.g., as part of the machine's boot sequence).

If a `stash` file is not used when the database is created, then the administrator who starts up the `krb5kdc` process will have to manually enter the Master Key (password) every time they start the process. This may seem like a typical tradeoff between convenience and security, but if the rest of the system is sufficiently hardened and protected, then very little security is lost by having the master key stored in the protected `stash` file. It is recommended that at least 1 slave KDC server be installed for each realm to ensure that a backup is available in the event that the master server becomes unavailable and that slave KDC be configured with the same level of security as the master.

Currently, the Sun Enterprise Authentication Mechanism utility, `kdb5_util`, can create 3 types of keys, `DES-CBC-CRC`, `DES-CBC-MD5`, and `DES-CBC-RAW`. "`DES-CBC`" means "DES encryption with Cipher block chaining" and the CRC, MD5, and RAW designators refer to the checksum algorithm that is used. By default, the key

created will be DES-CBC-CRC, which is the default encryption type for the KDC. The type of key created is specified on the command line with the -k option (see kdb5_util(1M) man page). Choose the password for your stash file very carefully as this password can be used in the future to decrypt the master key and modify the database. The password may be up to 1024 characters long and may include any combination of letters, numbers, punctuation, and spaces.

The following is an example of creating a stash file:

```
kdc1 # /usr/krb5/sbin/kdb5_util create -r BLUEPRINTS.SUN.COM -s
Initializing database '/var/krb5/principal' for realm 'BLUEPRINTS.SUN.COM'
master key name 'K/M@BLUEPRINTS.SUN.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key: <type the key>
Re-enter KDC database master key to verify: <type it again>
```

Notice the use of the -s argument to create the stash file. The location of the stash file is in the /var/krb5 (not configurable). The stash file will appear with the following mode and ownership settings:

```
-rw-------   1 root    other        14 Feb 22 14:28 .k5.BLUEPRINTS.SUN.COM
```

Note – The directory used to store the stash file and the database SHOULD NOT be shared or exported. More information on how to harden the KDC system is covered later in this article.

Secure Settings in the KDC Configuration File

The KDC and Administration daemons both read configuration information from /etc/krb5/kdc.conf. This file contains KDC specific parameters that govern overall behavior for the KDC and for specific realms. The parameters in the kdc.conf file are explained in detail in both the kdc.conf(4) man page and in the Sun Enterprise Authentication Mechanism 1.01 documentation at http://docs.sun.com.

The kdc.conf parameters describe locations of various files and ports to use for accessing the KDC and the administration daemon. These parameters generally do not need to be changed, and doing so does not result in any added security. However, there are some parameters that may be adjusted to enhance the overall security of the KDC. The following are some examples of adjustable parameters that enhance security.

- `kdc_ports`

  Defines the ports that the KDC will listen on to receive requests. The standard port for Kerberos V5 is 88. 750 is included and commonly used to support older clients that still use the default port designated for Kerberos V4. Solaris OE still listens on port 750 for backwards compatibility. This is not considered a security risk.

- `max_life`

  Defines the maximum lifetime of a ticket, and defaults to 8 hours. In environments where it is desirable to have users re-authenticate frequently and to reduce the chance of having a principal's credentials stolen, this value should be lowered. The recommended value is 8 hours.

- `max_renewable_life`

  Defines the period of time from when a ticket is issued that it may be "renewed" (using `kinit -R`). The standard value here is 7 days. To disable renewable tickets, this value may be set to 0 days, 0 hrs, 0 min. The recommended value is 7d 0h 0m 0s.

- `default_principal_expiration`

  A Kerberos principal is any unique identity to which Kerberos can assign a ticket. In the case of users, it is the same as the UNIX system user name. The default lifetime of any principal in the realm, may be defined in the `kdc.conf` file with this option. This should be used only if the realm will contain temporary principals, otherwise the administrator will have to constantly be renewing principals. Usually, this setting is left undefined and principals do not expire. This is not insecure as long as the administrator is vigilant about removing principals for users that no longer need access to the systems.

- `supported_enctypes`

  The encryption types supported by the KDC may be defined with this option. At this time, Sun Enterprise Authentication Mechanism software only supports `des-cbc-crc:normal` encryption type but in the future this may be used to ensure that only strong cryptographic ciphers are used.

- `dict_file`

  The location of a dictionary file containing strings that are not allowed as passwords. A principal with any password policy (see below) will not be able to use words found in this dictionary file. This is not defined by default. Using a dictionary file is a good way to prevent users from creating trivial passwords to protect their accounts and thus helps avoid one of the most common weaknesses in a computer network—guessable passwords. The KDC will only check passwords against the dictionary for principals which have a password policy association, so it is good practice to have at least 1 simple policy associated with all principals in the realm.

The Solaris OE has a default system dictionary that is used by the `spell` program that may also be used by the KDC as a dictionary of common passwords. The location of this file is: `/usr/share/lib/dict/words`. Other dictionaries may be substituted, the format is 1 word or phrase per line.

The following is a SEAM `/etc/krb5/kdc.conf` example with suggested settings:

```
#
# Copyright (c) 1998-2001 by Sun Microsystems, Inc.
# All rights reserved.
#
#ident       "@(#)kdc.conf 1.2          98/08/17 SMI"

[kdcdefaults]
             kdc_ports = 88,750

[realms]
             BLUEPRINTS.SUN.COM = {
                          profile = /etc/krb5/krb5.conf
                          database_name = /var/krb5/principal
                          admin_keytab = /etc/krb5/kadm5.keytab
                          acl_file = /etc/krb5/kadm5.acl
                          kadmind_port = 749
                          max_life = 8h 0m 0s
                          max_renewable_life = 7d 0h 0m 0s
                          dict_file = /usr/share/lib/dict/words
             }
```

Access Control

The Kerberos administration server allows for granular control of the administrative commands by use of an ACL file (`/etc/krb5/kadm5.acl`). The syntax for the ACL file allows for wildcarding of principal names so that it is not necessary to list every single administrator in the ACL file. This feature should be used with great care. The ACLs used by Kerberos allow privileges to be broken down into very precise functions that each administrator can perform. If a certain administrator only needs to be allowed to have read-access to the database then that person should not be granted full admin privileges. Below is a list of the privileges allowed:

- `a` – Allows the addition of principals or policies in the database.
- `A` – Prohibits the addition of principals or policies in the database.
- `d` – Allows the deletion of principals or policies in the database.
- `D` – Prohibits the deletion of principals or policies in the database.
- `m` – Allows the modification of principals or policies in the database.

- `M` – Prohibits the modification of principals or policies in the database.
- `c` – Allows the changing of passwords for principals in the database.
- `C` – Prohibits the changing of passwords for principals in the database.
- `i` – Allows inquiries to the database.
- `I` – Prohibits inquiries to the database.
- `l` – Allows the listing of principals or policies in the database.
- `L` – Prohibits the listing of principals or policies in the database.
- `*` – Short for all privileges (`admcil`).
- `x` – Short for all privileges (`admcil`), and is identical to "`*`".

Adding Administrators

After the ACLs are set up, actual administrator principals should be added to the system. It is strongly recommended that administrative users have separate "`/admin`" principals to use only when administering the system. For example, user "Joe" would have 2 principals in the database - `joe@REALM` and `joe/admin@REALM`. The `/admin` principal would only be used when administering the system, not for getting ticket-granting-tickets (TGTs) to access remote services. Using the `/admin` principal only for administrative purposes minimizes the chance of someone walking up to Joe's unattended terminal and performing unauthorized administrative commands on the KDC.

Kerberos principals may be differentiated by the "instance" part of their principal name. In the case of user principals, the most common instance identifier is `/admin`. It is standard practice in Kerberos to differentiate user principals by defining some to be `/admin` instances and others to have no specific instance identifier (e.g. `joe/admin@REALM` versus `joe@REALM`). Those with the `/admin` instance identifier are assumed to have administrative privileges defined in the ACL file and should only be used for administrative purposes. A principal with an `/admin` identifier which does not match up with any entries in the ACL file will not be granted any administrative privileges, it will be treated as a non-privileged user principal. Also, user principals with the `/admin` identifier are given separate passwords and separate permissions from the non-admin principal for the same user.

The following is a sample `/etc/krb5/kadm5.acl` file:

```
#
# Copyright (c) 1998, by Sun Microsystems, Inc.
# All rights reserved.
#
#ident"@(#)kadm5.acl1.398/08/19 SMI"

# joe/admin is given full administrative privilege
joe/admin@BLUEPRINTS.SUN.COM *

#
# jane/admin user is allowed to query the database (d), listing
principals
# (l), and changing user passwords (c)
#
jane/admin@BLUEPRINTS.SUN.COM dlc
```

It is highly recommended that this file (`kadm5.acl`) be tightly controlled and that users be granted only the privileges that they need to perform their assigned tasks and nothing more.

Creating Host Keys

Creating host keys for systems in the realm such as slave KDCs is done the same way that creating user principal's is done. However, the `-randkey` option should ALWAYS be used so that no one ever knows the actual key for the hosts. Host principals are almost always stored in the `keytab` file to be used by root-owned processes that wish to act as Kerberos services for the local host. It is rarely necessary for anyone to actually know the password for a host principal since the key is stored safely in the `keytab` and is only accessible by root-owned processes, never by actual users.

The following is an example of creating a host key:

```
kadmin:  addprinc -randkey host/birdie.blueprints.sun.com
Principal "host/birdie.blueprints.sun.com@BLUEPRINTS.SUN.COM" created.
```

When creating `keytab` files, the keys should always be extracted from the KDC on the same machine where the `keytab` is to reside using the `ktadd` command from a `kadmin` session. If this is not feasible, then take great care in transferring the `keytab` file from one machine to the next. A malicious attacker who posesses the contents of the `keytab` file could use these keys from the file in order to gain access to another user or service's credentials. Having the keys would then allow the attacker to

impersonate whatever principal that the key represented and further compromise the security of that Kerberos realm. Some suggestions for transferring the `keytab` are to use Kerberized, encrypted `ftp` transfers, or to use the secure file transfer programs `scp` or `sftp` offered with the SSH package (`http://www.openssh.org`). Another safe method is to place the `keytab` on a removable disk, and hand deliver it to the destination.

Since hand delivery does not scale well for large installations, using the Kerberized `ftp` daemon that comes with the Sun Enterprise Authentication Mechanism software package is perhaps the most convenient and secure method available.

Synchronizing Clocks

All servers participating in the Kerberos realm need to have their system clocks synchronized to within a configurable time limit (default 300 seconds). The safest, most secure way to systematically synchronize the clocks on a network of Kerberos servers is by using the Network Time Protocol (NTP) service. Solaris OE comes with NTP client and NTP server software (SUNWntpu package). See the `ntpdate`(1M) and `xntpd`(1M) man pages for more information on the individual commands. For more information on configuring NTP, see the Sun BluePrints™ OnLine NTP articles posted at:

"Using NTP to Control and Synchronize System Clocks - Part I: Introduction to NTP" (July 2001)

`http://www.sun.com/blueprints/0701/NTP.pdf`

"Using NTP to Control and Synchronize System Clocks - Part II: Basic NTP Administration and Architecture" (August 2001)

`http://www.sun.com/blueprints/0801/NTPpt2.pdf`

"Using NTP to Control and Synchronize System Clocks - Part III: NTP Monitoring and Troubleshooting" (September 2001)

`http://www.sun.com/blueprints/0901/NTPpt3.pdf`

It is critical that the time be synchronized in a secure manner. A simple denial of service attack on either a client or a server would involve just skewing the time on that system to be outside of the configured clock skew value, which would then prevent anyone from acquiring TGTs from that system or accessing Kerberized services on that system. The default clock-skew value of 5 minutes is the maximum recommended value.

The online (`http://docs.sun.com`) Sun Enterprise Authentication Mechanism software 1.01 documentation has a very good section on synchronizing clocks using NTP in Kerberos environments at:

`http://docs.sun.com:80/ab2/coll.384.2/@Ab2CollView?Ab2Lang=C&Ab2Enc=iso-8859-1`

The NTP infrastructure must also be secured, including the use of server hardening for the NTP server and application of NTP security features. Using the Solaris Security Toolkit software (formerly known as JASS) with the `secure.driver` script to create a minimal system and then installing just the necessary NTP software is one such method. The Solaris Security Toolkit software is available at:

`http://www.sun.com/security/jass/`

Documentation on the Solaris Security Toolkit software is available at:

`http://www.sun.com/security/blueprints`

Establishing Password Policies

Kerberos allows the administrator to define password policies that may be applied to some or all of the user principals in the realm. A password policy contains definitions for the following parameters:

- *Minimum Password Length* – The number of characters in the password, for which the recommended value is 8.
- *Maximum Password Classes* – The number of different character "classes" that must be used to make up the password. Letters, numbers, and punctuation are the 3 classes and valid values are 1, 2, and 3. The recommended value is 2.
- *Saved Password History* – The number of previous passwords that have been used by the principal and cannot be reused. The recommended value is 3.
- *Minimum Password Lifetime* (secs) – The minimum time that the password must be used before it can be changed. The recommended value is 3600 (1 hour).
- *Maximum Password Lifetime* (secs) – The maximum time that the password can be used before it must be changed. The recommended value is 7776000 (90 days).

These values can be set as a group and stored as a single policy. Different policies can be defined for different principals. It is recommended that the minimum password length be set to at least 8 and that at least 2 classes be required. Since most people tend to choose easy to remember, easy to type passwords, it is a good idea to at least set up policies to encourage slightly more difficult to guess passwords through the use of these parameters. Setting the Maximum Password Lifetime value may be helpful in some environments to force people to change their passwords periodically. The period is up to the local administrator according to the overriding corporate security policy used at that particular site. Setting the Saved Password History value combined with the Minimum Password Lifetime value, will prevent people from simply switching their password several times until they get back to their original or "favorite" password.

The maximum password length supported is 255 characters, unlike the UNIX password database which only supports up to 8 characters. Passwords are stored in the KDC encrypted database using the KDC default encryption method,

DES-CBC-CRC. In order to prevent password guessing attacks, it is recommended that users choose long passwords or passphrases. The 255 character limit allows one to choose a small sentence or easy to remember phrase instead of a simple 1-word password.

Also, recall that it is possible to use a dictionary file that can be used to prevent users from choosing common, easy to guess words (see the previous section on configuring the KDC). The dictionary file is only used when a principal has a policy association, so it is highly recommended that at least 1 policy be in affect for all principals in the realm.

The following is an example password policy creation:

```
usage: add_policy [options] policy
  options are:
    [-maxlife time] [-minlife time] [-minlength length]
    [-minclasses number] [-history number]
kadmin:  addpol -minlife "1 hour" -maxlife "90 days" -minlength 8
-minclasses 2 -history 3 goodpwds
kadmin:  getpol goodpwds
Policy: goodpwds
Maximum password life: 7776000
Minimum password life: 3600
Minimum password length: 8
Minimum number of password character classes: 2
Number of old keys kept: 3
Reference count: 0
```

The above example created a password policy called goodpwds which enforces a maximum password lifetime of 90 days, minimum length of 8 characters, a minimum of 2 different character classes (letters, numbers, punctuation), and a password history of 3.

To apply this policy to an existing user, modify the following:

```
kadmin:  modprinc -policy goodpwds joe
Principal "joe@BLUEPRINTS.SUN.COM" modified.
```

To modify the default policy that is applied to all user principal's in a REALM, change the following:

```
kadmin: modpol -maxlife "90 days" -minlife "1 hour" -minlength 8
-minclasses 2 -history 3 default
kadmin:  getpol default
Policy: default
Maximum password life: 7776000
Minimum password life: 3600
Minimum password length: 8
Minimum number of password character classes: 2
Number of old keys kept: 3
Reference count: 1
```

The `Reference count` value indicates how many principals are configured to use the policy.

Note – The "default" policy is automatically applied to all new principals that are NOT given the same password as the principal name when they are created. Any account with a policy assigned to it will also use the dictionary (defined in the `dict_file` parameter in `/etc/krb5/kdc.conf`) to check for common passwords.

Hardening the KDCs

Since the KDC holds the database (usually) and the keys to access that database, it is critical that the KDC system be secured and monitored closely. The techniques and recommendations described in the Sun BluePrints OnLine article, *"Solaris Operating Environment Security™ - Updated for Solaris 8 Operating Environment"* (April 2001) located at `http://www.sun.com/blueprints/0401/security-updt1.pdf`, should be studied and applied to the KDC wherever applicable In addition, the documentation for Sun Enterprise Authentication Mechanism software installation at `docs.sun.com` has a section on *Increasing Security* that should be read and applied. It is best to only allow secure shells such as SSH (`http://www.openssh.org`) or Kerberized `telnet` access with privacy protection.

Users accessing the system with Kerberized `telnet` clients must use the option to enable encryption of the `telnet` session traffic to further protect the privacy of the session. If console access is not a practical solution for administrators to use for accessing the system to do KDC troubleshooting, Secure Shell (SSH) access should be considered as a safe way to access the KDC remotely. Non-encrypted remote

access over the network to the KDC MUST BE avoided at all costs, because this would expose all passwords used to network sniffer attacks and could thus compromise the entire KDC.

Below is output from a system installed using the Solaris Security Toolkit software (JASS) `secure.driver` script and running a KDC and Kerberos Administration server. This system also uses SSH for secure network access to administer the KDC.

```
UDP: IPv4
   Local Address           Remote Address      State
-------------------- -------------------- -------
       *.kerberos                              Idle
       *.750                                    Idle
       *.*                                      Unbound

TCP: IPv4
   Local Address     Remote Address   Swind  Send-Q  Rwind  Recv-Q  State
   -------------     --------------  -----  ------  -----  ------  -------
       *.*               *.*             0       0    24576      0    IDLE
       *.22              *.*             0       0    24576      0    LISTEN
       *.22              *.*             0       0    24576      0    LISTEN
       *.kerberos-adm    *.*             0       0    24576      0    LISTEN
       *.*               *.*             0       0    24576      0    IDLE

Active UNIX domain sockets
Address  Type            Vnode      Conn   Local Addr       Remote Addr
30000969dd8 stream-ord 3000096f320 00000000 /var/spool/prngd/pool
```

This output indicates that the KDC is only offering service to the necessary Kerberos ports (88 - kerberos, 749 - kerberos-adm) along with the SSH service on port 22.

Port 88 and 750 are both used by the Kerberos KDC. Port 750 is provided for backwards compatibility with older clients. Port 88 is the new, standard, default KDC port (see `http://www.iana.org/assignments/port-numbers`).

If the host is also configured to allow Kerberized log in and `telnet` sessions then the output may also contain these lines:

```
*.eklogin     *.*            0       0 24576       0 LISTEN
*.telnet      *.*            0       0 24576       0 LISTEN
```

It is not likely that an `ftp` daemon should be necessary on a KDC system, except in the case where `keytab` files are being transferred between the master KDC and a slave. In this situation, the Kerberized `ftp` daemon is recommended and users should be sure to use the encryption option to ensure the privacy of the transfer. Information on configuring the `inetd.conf` file for these services is in the online Sun Enterprise Authentication Mechanism software documentation in the *How to Restrict Access to the KDC* section at:

`http://docs.sun.com:80/ab2/coll.384.2@Ab2CollView?Ab2Lang=C&Ab2Enc=iso-8859-1`

The following is a minimal KDC `inetd.conf` file:

```
#
# Kerberos V5 Warning Message Daemon (optional)
#
# 100134/1  tli rpc/ticotsord   wait root /usr/lib/krb5/ktkt_warnd ktkt_warnd

#
# GSS Daemon
# (only needed if sharing kerberos-protected NFS mounts)
# 100234/1  tli rpc/ticotsord   wait root /usr/lib/gss/gssd gssd

#
# Kerberized login daemon
#   -k = allow kerberos authentication
#   -e = force encrypted sessions
#   -c = Require Kerberos V5 clients to present a  cryptographic
#        checksum  of  initial  connection  information like the
#        name of the user that the client is trying to access in
#        the  initial  authenticator.
#
eklogin stream  tcp nowait root /usr/krb5/lib/rlogind rlogind -k -c -e

#
# Simple Kerberos authenticated login (no encryption)
# - less secure the 'eklogin'
# - uncomment to enable.
#
#klogin stream  tcp nowait root /usr/krb5/lib/rlogind rlogind -k -c

#
# Kerberized telnet daemon
#    -a user = Only allow connections when remote user can provide valid
#                Kerberos authentication information to identify the user and
#                is allowed access to account without providing password.
#                In some situations it is good to allow non-kerberized
#                access for situations where the KDC is unreachable or
#                unavailable for some reason.
#
telnet stream  tcp nowait root /usr/krb5/lib/telnetd telnetd -a user

#
# Set up the daemon for propogating the Kerberos database
# Uncomment this ONLY if there are slave KDCs for the domain(s)
# served by the local KDC.
#krb5_prop stream  tcp     nowait  root    /usr/krb5/lib/kpropd kpropd
```

The previous example allows only Kerberos authenticated `telnet` sessions or Kerberos protected `rlogin` sessions (with encryption) for remote access. The `ktkt_kwarnd` and `gssd` services are also allowed. `ktkt_kwarnd` is used to send out alerts to users when their credentials are about to expire. `gssd` is a user mode daemon that operates between the kernel `rpc` and the Generic Security Service Application Program Interface (GSS-API) to generate and validate GSS-API security tokens. `gssd` is needed when Kerberos protected NFS disks are being mounted or shared (see NFS section below). If Kerberos is not used for NFS mounts, then the `gssd` service may be removed from the `/etc/inetd.conf` file.

We strongly recommend that hosts have an empty `/etc/hosts.equiv` file and that there not be a `.rhosts` file in root's home directory. Kerberos-authenticated root access may be granted to specific Kerberos principals by placing those principals in the file `.k5login` in root's home directory. This is useful to allow a non-root principal from another system to log in to the local system as root without providing the root principal's password (as long as the remote user has valid Kerberos credentials). This is a dangerous feature that should be used sparingly and only if absolutely necessary. Allowing anyone to access the root account without any authentication on the local system is dangerous. Anyone with a principal in the root user's `.k5login` file becomes a potential target for attack. Guessable passwords, unexpired local tickets, unattended terminals, letting someone else" borrow" a shell for a minute (and other social engineering attacks) could all be used to subvert the remote account in order to gain easy root access to the actual target system.

The `.k5login` file may be used by any user who wishes to grant access to their account to another user with valid credentials. This can be both useful and dangerous depending on the situation. It is useful because it allows a user to grant access to his own account without giving away the password; however, it leaves that account vulnerable to attack by anyone who can gain access to the accounts that are being granted access. For example, suppose Joe has an entry for Jane in his `.k5login` file, and Jane authenticates and gets a valid TGT and then leaves her terminal for a while. Anyone who has access to Jane's terminal can then log into Joe's account without authenticating as either user because Jane left her terminal with valid credentials still cached, and Joe trusts Jane implicitly by granting her access to his account by leaving an entry in his `.k5login` file. Using the `.k5login` file should be discouraged if at all possible.

Finally, disks on the KDC must not be exported, especially the ones which contain the Kerberos database, `stash` file, or `keytab` file. All of these items should be kept locally and should only be readable by root.

Brief Summary of Securing Kerberos Servers

1. Remove unnecessary services from the `/etc/inetd.conf` file.

2. Remove the `/etc/hosts.equiv` file.

3. Monitor any root owned `.k5login` files carefully (remove them if possible).

4. Do not export disks containing the KDC database, ACL file, or `stash` file.

Backing Up a KDC

Backups of a KDC system should be made regularly or according to local policy. However, backups should exclude the `/etc/krb5/krb5.keytab` file. If the local policy requires that backups be done over a network, then these backups should be secured either through the use of encryption or possibly by utilizing a separate network interface that is only used for backup purposes and is not exposed to the same traffic as the non-backup network traffic. Backup storage media should always be kept in a secure, fireproof location.

Monitor the KDC

Once the KDC is configured and running, it should be continually and vigilantly monitored. The Solaris Fingerprint Database (sfpDB) is a good starting point for monitoring the files on a system. Refer to the Sun BluePrints OnLine article, *"The Solaris™ Fingerprint Database: A Security Tool for Solaris Operating Environment Files"* (May 2001) located at `http://www.sun.com/blueprints/0501/Fingerprint.pdf` for more information on the `sfpDB`. Other tools such as *tripwire* (`http://tripwire.com`) and *swatch* (`http://oit.ucsb.edu/~eta/swatch`) can be used to monitor changes in the filesystem and in the log files. The *logcheck* utility (`http://psionic.com/abacus/logcheck`) is also a good tool for monitoring activity logged on your system. The Sun Enterprise Authentication Mechanism software KDC logs information into the `/var/krb5/kdc.log` file, but this location can be modified in the `/etc/krb5/krb5.conf` file, in the *logging* section. By default, the `/etc/krb5/krb5.conf` file is defined as follows:

```
[logging]
         default = FILE:/var/krb5/kdc.log
         kdc = FILE:/var/krb5/kdc.log
```

The KDC log file should have read/write permissions for the root user only, as follows:

```
-rw------ 1   root   other     75025   May 30 17:55   /var/krb5/kdc.log
```

Securing Client Systems

Many of the principals used to secure the KDCs can be applied to securing the hosts on the network that will be members of the Kerberos realm. Host-based security modifications should be applied to all systems in the Kerberos realm to further enhance the overall security of the network. Following the instructions mentioned in the Sun BluePrints OnLine, Solaris Security Toolkit software (formerly referred to as JASS) articles, is an excellent way to ensure that all of the systems are sufficiently hardened in a uniform matter. Disallowing remote access to any non-secure protocol is recommended here as it is for the KDC. This is important for protecting the sensitive `keytab` files that may be present on the client.

Authentication with PAM and Kerberos

The Solaris OE ships with a Kerberos V5 module for PAM that can be used by both Kerberized and non-Kerberized applications (e.g. CDE) for authentication, account, session, and password management. When using the `PAM-KRB5` module with the Kerberized daemons such as `ktelnetd`, `krshd`, or `krlogind`, the `acceptor` option should be used. This prevents the PAM module from obtaining the initial TGT. The Kerberized daemon will automatically perform this step upon successful authentication.

When using the `PAM-KRB5` module for non-Kerberized applications such as the normal `telnetd` or `rlogind`, the `try_first_pass` option can be used to only request authentication using the first password that the user provides, regardless of the PAM module being used. Thus, if the PAM modules are stacked so that `PAM-KRB5` is used before `PAM-UNIX`, the same password would be tried for both modules instead of prompting the user for a new one for each module. It is important to note that using non-Kerberized applications with the `PAM_KRB5` module will expose the users password to network snooping attacks because the password is still passed over the wire in the clear. The password is only protected on the network when the client has valid credentials (e.g., a TGT) locally before attempting to authenticate to a remote server which supports Kerberos for authentication. In that situation, only the protected Kerberos credentials are passed over the wire, not the clear text password itself.

In a Kerberized network environment, it is recommended that Kerberos authentication be tested before reverting back to UNIX system passwd/shadow authentication. This helps facilitate a single-signon environment; in addition, once a user is authenticated by `PAM-KRB5`, or by the Kerberized network daemon, that user automatically has a TGT that can then be used to access other network applications without re-entering a password (see `pam_krb5`(5) man page).

It is also recommended to configure systems to use the Kerberized `telnet` client (`/usr/krb5/lib/telnetd`) instead of the normal default `telnet` daemon (`/usr/sbin/in.telnetd`). The Kerberized `telnet` daemon uses Kerberos authentication as its primary authentication method and also supports having encrypted sessions.

The following is an example of the `/etc/pam.conf` file Kerberos entries:

```
ktelnet auth sufficient /usr/lib/security/$ISA/pam_krb5.so.1 acceptor
ktelnet auth optional  /usr/lib/security/$ISA/pam_unix.so.1
```

The Kerberized `telnetd` automatically uses Kerberos authentication, so it is not necessary to have a `PAM_KRB5` module in the stack. With the `acceptor` option, the `PAM_KRB5` module will not prompt for the password if the user has already successfully authenticated or has a valid TGT. Stacking in this manner allows the `PAM_UNIX` module to be used ONLY when the `PAM_KRB5` authentication fails. To allow access in the event that the KDC is not available or cannot be reached from the client system for some reason, having the `PAM_UNIX` module included as an optional authentication method is a good idea. This will allow users with a local account to use their UNIX system password to gain access if all else fails. This fallback PAM module should only be uncommented if this is absolutely necessary.

The following is an example of the `PAM_UNIX` and `PAM_KRB5` entries in the `/etc/pam.conf` file:

```
telnet auth sufficient /usr/lib/security/$ISA/pam_krb5.so.1
telnet auth required /usr/lib/security/$ISA/pam_unix.so.1
```

The previous example indicates that the standard `telnetd` is to try to use `PAM_KRB5` to authenticate the user and if that fails, revert to standard UNIX system authentication. The `try_first_pass` option could be used with the `PAM_KRB5` module to force the same password to be used for both mechanisms. Again, stacking the `PAM_UNIX` module after the `PAM_KRB5` module is only desirable if absolutely necessary as a fallback in case of problems with the Kerberos configuration.

Note – The Kerberos `ftp` daemon does not use PAM for authentication.

Kerberos Options

The `/etc/krb5/krb5.conf` file contains information that all Kerberos applications use to determine what server to talk to and what realm they are participating in. Configuring the `krb5.conf` file is covered in the Sun Enterprise Authentication Mechanism software installation guide.

The `appdefaults` section in the `krb5.conf` file contains parameters that control the behavior of many Kerberos client tools. Each tool may have its own section in either the `appdefaults` section of the `krb5.conf` file.

Many of the applications that use the `appdefaults` section, use the same options; however, they may be set in different ways for each client application. This section provides a list of the most common `appdefault` parameters and how they affect the Kerberos applications.

Common Options Affecting Kerberos Client Applications

The following options may be used in the specific client sections of all the supported client applications. Their meaning is the same with each application, and thus is described here rather than repeated in each client-specific section below.

- `renewable [true/false]` – Renew the TGT as long as it has not expired. `true` indicates that tickets created will be renewable by the user. Renewing a ticket simply extends the life of the ticket by the maximum ticket lifetime allowed for that system. The `max_life` value is defined and enforced by the KDC itself and is a parameter in the `/etc/krb5/kdc.conf` file on the KDC. Tickets are renewed by using the `kinit` client with the `-R` option, a password is not required to renew a ticket as long as it remains valid and within the `max_renewable_life` time period.
- `forwardable [true/false]` – Issue forwardable TGTs to users. If `true` is set, the local TGT is forwarded to other Kerberized network services on the network and used to access additional Kerberos services without reauthenticating.
- `forward [true/false]` – Forward a copy of the users credentials (TGT) to the remote server after authenticating. The forwarded credentials cannot be forwarded beyond the remote server. This differs from the `forwardable` option above because the local credentials are only usable on the remote server, beyond that point, they are *non-forwardable*.
- `encrypt [true/false]` – Encrypt all traffic between the client and the remote server.

Kerberos Client Applications

The following Kerberos applications may have their behavior modified through the user of options set in the `appdefaults` section of the `/etc/krb5/krb5.conf` file or by using various command line arguments. These clients and their configuration settings are described below.

`kinit`

The `kinit` client is used by people who want to obtain a TGT from the KDC. The `/etc/krb5/krb5.conf` file supports the following `kinit` options: `renewable`, `forwardable`, and `proxiable`.

- `proxiable [true/false]` – `kinit` may request that the KDC issue a ticket that may be used by the remote service to perform a remote request on the users behalf. An example of this is that a print service client can give the print server a proxy to access the client's files on a particular file server in order to satisfy a print request.

`telnet`

The Kerberos `telnet` client has many command line arguments that control its behavior, refer to the man page for complete information. However, there are several interesting security issues involving the Kerberized `telnet` client.

The `telnet` client will use a session key even after the service ticket which it was derived from has expired. This means that the `telnet` session remains active even after the ticket originally used to gain access, is no longer valid. This is insecure in a strict environment, however, the tradeoff between ease-of-use and strict security tends to lean in favor of ease-of-use in this situation. It is recommended that the `telnet` connection be re-initialized periodically by disconnecting and reconnecting with a new ticket. The overall lifetime of a ticket is defined by the KDC (`/etc/krb5/kdc.conf`), normally defined to be 8 hours.

The `telnet` client allows the user to forward a copy of the credentials (TGT) used to authenticate to the remote system using the `-f` and `-F` command line options. The `-f` option sends a non-forwardable copy of the local TGT to the remote system so that the user may access Kerberized NFS mounts or other local Kerberized services on that system only. The `-F` option sends a forwardable TGT to the remote system so that the TGT could be used from the remote system to gain further access to other remote Kerberos services beyond that point. `-F` is a superset of `-f`. If the `Forwardable` and/or `forward` options are set to `false` in the `krb5.conf` file, these command line arguments may be used to override those settings, thus giving individuals the control over whether and how their credentials are forwarded.

The -x option should be used to turn on encryption for the data stream. This will further protect the session from eavesdroppers. If the telnet server does not support encryption, the session will be closed. The /etc/krb5/krb5.conf file supports the following telnet options: forward, forwardable, encrypt, and autologin.

- autologin [true/false] – This parameter tells the client to try and attempt to log in without prompting the user for a user name. The local user name is passed on to the remote system in the telnet negotiations.

rlogin/rsh

The Kerberos rlogin and rsh clients behave much the same as their non-Kerberized equivalents. Because of this, it is recommended that—if they are required to be included in the network—files such as /etc/hosts.equiv and .rhosts in the root user's directory be removed. The Kerberized versions have the added benefit of using Kerberos protocol for authentication and can also use Kerberos to protect the privacy of the session using encryption.

Similar to telnet described previously, these clients will use a session key after the service ticket which it was derived from has expired. Thus for maximum security, the rlogin/rsh session should be re-initialized periodically. rlogin also uses the -f, -F, and -x options in the same fashion as the telnet client. The /etc/krb5/krb5.conf file supports the following rlogin options: forward, forwardable, and encrypt.

Command line options override configuration file settings. For example, if the rsh section in the krb5.conf file indicates *encrypt false*, but the -x option was used on the command line, an encrypted session will be used.

rcp

Kerberized rcp can be used to transfer files securely between systems using Kerberos authentication and encryption (with the -x command line option). It does not prompt for passwords, the user must already have a valid TGT before using rcp if they wish to use the encryption feature. However, beware if the -x option is not used and no local credentials are available, the rcp session will revert to the standard, non-Kerberized (and insecure) rcp behavior. It is highly recommended that users always use the -x option when using the Kerberized rcp client.The /etc/krb5/krb5.conf file supports the encrypt [true/false] option.

login

The Kerberos `login` program (`login.krb5`) is forked from a successful authentication by the Kerberized `telnet` daemon or the Kerberized `rlogin` daemon. Note that this Kerberos `login` daemon is separate from the standard Solaris OE `login` daemon and thus, the standard Solaris OE features such as BSM auditing are not yet supported when using this daemon. The `/etc/krb5/krb5.conf` file supports the `krb5_get_tickets [true/false]` option.

If this option is set to `true`, then the `login` program will generate a new Kerberos ticket (TGT) for the user upon proper authentication.

ftp

The SEAM version of the `ftp` client uses the GSSAPI [RFC 2743] with Kerberos V5 as the default mechanism. This means that it uses Kerberos authentication and (optionally) encryption through the Kerberos V5 GSS mechanism. The only Kerberos-related command line options are `-f` and `-m`. The `-f` option is the same as described above for `telnet` (there is no need for a `-F` option). `-m` allows the user to specify an alternative GSS mechanism if so desired, the default is to use the `kerberos_v5` mechanism.

The protection level used for the data transfer can be set using the `protect` command at the `ftp` prompt. Sun Enterprise Authentication Mechanism software `ftp` supports the following protection levels:

- `clear` – unprotected, unencrypted transmission
- `safe` – data is integrity protected using cryptographic checksums
- `private` – data is transmitted with confidentiality and integrity using encryption

The following is an example of the private protection level:

```
ftp> protect private
200 Protection level set to Private.
```

It is recommended that users set the protection level to *private* for all data transfers.

The `ftp` client program does not support or reference the `krb5.conf` file to find any optional parameters. All `ftp` client options are passed on the command line. See the man page for the Kerberized `ftp` client, `ftp`(1).

The following is an example of the `appdefaults` section in the `/etc/krb5/krb5.conf` file:

```
[appdefaults]
         kinit = {
                  renewable = true
                  forwardable= true
                  proxiable = false
         }
         rlogin = {
                  renewable = true
                  forwardable= true
                  encrypt = true
         }
         rsh = {
                  renewable = true
                  forwardable= true
                  encrypt = true
         }
         rcp = {
                  encrypt = true
         }
         telnet = {
                  autologin = true
                  renewable = true
                  forwardable= true
         }
         login = {
                  krb5_get_tickets = true
         }
```

Kerberos Remote Application Server Behavior

Kerberized remote application servers such as `telnetd`, `ftpd`, `rlogind`, and `rshd` also have command line arguments that govern their behavior. Below is an abbreviated list of the arguments that they support which relate directly to their security.

telnetd

- `-a [authmode]`

  Where `authmode` can be one of the following options: `debug`, `user`, `valid`, `other`, `none`, or `off`—which are described as follows:

  - `debug`

    Turns on authentication debugging code.

  - `user`

    Only allow connections when the remote user can provide valid authentication information to identify the remote user, and is allowed access to the specified account without providing a password. This is the recommended setting as it forces the user to present valid kerberos credentials and does not involve having the user enter a password that would pass over the network in the clear.

  - `valid`

    Only allow connections when the remote user can provide valid authentication information to identify the remote user. The `login`(1) command will provide any additional user verification needed if the remote user is not allowed automatic access to the specified account.

  - `other`

    The same as specifying *valid* (above).

  - `none`

    DEFAULT state—authentication information is not required. If none is presented, the `login`(1) program will provide the necessary authentication.

  - `off`

    This disables the authentication code. All authentication is then handled through the PAM framework.

- `-X [authtype]`

  This argument *disables* the use of the indicated `authtype`.

rlogind

The Kerberized remote `login` daemon that services `rlogin` requests.

The following security related options are supported. For a full list of command line arguments, see the man page, `rlogin`(1M).

- `-k`, `-5` *(REQUIRED)*

  Allow Kerberos authentication (version 5). In Sun Enterprise Authentication Mechanism software, this is the only authentication mode supported so one of these two options must be used.

- `-e`, `-E`, `-x`, `-X` *(RECOMMENDED)*

  Create an encrypted session. Note that all of these options have the same effect, so the user should only choose one.

- `-c` *(RECOMMENDED)*

  Require Kerberos clients to present a cryptographic checksum of the initial connection information. This provides extra security by preventing an attacker from changing the initial connection information.

- `-i`

  Ignore authenticator checksums.

- `-p`

  Prompt for password if any authentication check fails.

- `-P`

  Prompt for password regardless of authentication check.

ftpd

`ftpd` is the Kerberized FTP daemon. It should be noted that this daemon is Kerberized through the use of the GSSAPI interface. The Kerberos GSSAPI mechanism is then used to perform authentication and privacy support if requested by the client.

- `-a`

  Only permit anonymous or Kerberos authenticated connections. One can set this behavior by either providing the `-a` option on the command line of `/usr/krb5/lib/ftpd`, or by inserting a line of the form, `AUTH=1`, into the `/etc/default/ftpd` file.

Other Kerberos Services

This section discusses the NFS security services offered with Kerberos.

NFS

Solaris OE offers the ability to share filesystems over NFS with optional security modes (see `nfssec`(5) man page). Without the Sun Enterprise Authentication Mechanism software package, the only security mode available is Diffie-Helmen (`sec=dh` in the `nfssec.conf` file). This is sometimes known as “SecureNFS.” The Solaris 8 OE with the Sun Enterprise Authentication Mechanism software Kerberos package provides additional security modes for disks shared using the NFS protocol. The level of security provided may be configured by the exporting server through the use of the `sec=` option (see `share`(1M) man page). The three levels of Kerberos protection provided are:

- `krb5` – Kerberos authentication
- `krb5i` – Kerberos authentication with integrity
- `krb5p` – Kerberos authentication with integrity and privacy

`krb5p` is the most secure because it not only authenticates and ensures the integrity of the share, but also encrypts all the NFS data between the client and server. `krb5i` authenticates the user and verifies the integrity of the packets with cryptographic checksums (MD5), but does not encrypt the actual data. `krb5` simply authenticates the user by verifying that the user has a valid TGT before allowing access to the NFS share.

The Sun Enterprise Authentication Mechanism software guide at `http://docs.sun.com`, mentions that the root principal may be added to the local `keytab` file in order to make mounting the Kerberized, NFS file systems possible on client systems. This makes it easier for users to mount these file systems without having to know the root principal’s password (which should NOT be the same as the system root user password). However, if the `automount` daemon is being used to control NFS mounts, it is safer to not have the *root* principal in the client system `keytab` file. Users will then be forced to have valid credentials for their own principal in order to access the Kerberized NFS mounted disks instead of using the root principal credentials in the `keytab` file. Having root in the `keytab` is also a slight security risk as it could possibly be compromised, thus granting the attacker access to the root principal. In theory, a root principal is no more privileged than a non-root principal from the KDC’s point of view; however, quite often the root

principal is directly associated with the actual root user on the system and should thus be afforded higher protection than normal principals. TABLE 1 lists an example of NFS mounting options.

TABLE 1 NFS Mounting Options

NFS Mounting Options	Comments
direct 'mount' command	Must have 'root' privilege to execute this command
automount	Users must have valid Kerberos credentials in order to access automounted shares which have Kerberos protection

The following is a sample configuration file (`/etc/dfs/dfstab`) for exporting disks with Kerberos protection:

```
share -F nfs -o sec=krb5p -d "KRB5 protected share" /export/krb5
```

Note that multiple security modes may be specified for a particular disk, so that clients who do not want full protection can choose one of the other modes, by using the following command:

```
share -F nfs -o sec=krb5p:krb5i:krb5 -d "KRB5 protected share" /export/krb5
```

The first mode listed is the default, but if a client requests one of the other modes, it will be allowed.

The `automount` program can also be configured to automatically mount disks with Kerberos protection. The following is an example entry from the `/etc/auto_master` file:

```
/home            auto_home        -nosuid,sec=krb5p
```

On the NFS client system, the user performing the mount operation must have a valid TGT in order to mount the Kerberos protected disk. It is recommended that the `automount` daemon [man page - `automount`(1M)] be used for mounting of Kerberos protected disks because it will allow any individual user with a valid TGT to mount or read a Kerberos protected share. The alternative is to have remote disks mounted by the `root` user, but this requires the `root` user to have a valid TGT. The `root` user can get a TGT by requesting one as itself (i.e., with a `root` principal in the database, run `kinit` while logged in as root) or as another user (i.e., `kinit`

someuser while logged in as root) and use that TGT for the `mount` command. Once a Kerberos protected share is mounted by root, only users with valid TGTs will have access to that space.

The recommended method is to configure the `automount` program to do all mounts that users will need. Only users with valid TGTs will be able to access the protected mounts.

Summary

Adding Kerberos to a network can increase the overall security available to the users and administrators of that network. Remote sessions can be securely authenticated and encrypted, shared disks can be secured and encrypted across the network. In addition, Kerberos allows the database of user and service principals to be managed securely from any machine which supports the Sun Enterprise Authentication Mechanism software Kerberos protocol. SEAM is interoperable with other RFC 1510 compliant Kerberos implementations such as MIT `Krb5` and some MS Windows 2000 Active Directory services. Adopting the practices recommended in this article will further secure the Sun Enterprise Authentication Mechanism software infrastructure to help ensure a safer network environment.

References

- Sun BluePrints OnLine Security Articles

  `http://www.sun.com/security/blueprints/`

- Solaris Security Toolkit (JASS)

  `http://www.sun.com/security/jass`

- Sun Security Products & Solutions,

  `http://www.sun.com/security`

- Sun Enterprise Authentication Mechanism information,

  `http://www.sun.com/software/solaris/ds/ds-seam`

- SEAM Documentation

  `http://docs.sun.com:80/ab2/coll.384.2/`
  `@Ab2CollView?Ab2Lang=C&Ab2Enc=iso-8859-1`

- MIT Kerberos Home Page

```
http://web.mit.edu/kerberos/www/
```

- Lance Spitzner's Hardening Solaris Guide

```
http://www.enteract.com/~lspitz/armoring.html
```

- Tripwire

```
http://www.tripwire.com
```

- SWATCH

```
http://www.oit.ucsb.edu/~eta/swatch
```

- Logcheck

```
http://www.psionic.com/abacus/logcheck
```

- OpenSSH

```
http://www.openssh.org
```

Author's Bio: Wyllys Ingersol

The author has worked in the network security field for over 6 years. Since joining Sun in 2000, he has worked in the network security group developing Kerberos software for Solaris. Prior to joining Sun he worked as a developer on a commercial firewall product used by many Fortune 100 companies including banks, financial service institutions, and engineering consulting companies.