Zephyr Installation and Operation Guide

John T Kohl

DRAFT—20 November 1989

1 Configuration required

Zephyr was developed and tested on 4.3BSD systems (Digital VAX and IBM RT/PC). If you are using some other 4.2BSD derivative system, Zephyr should work, but it has not been extensively tested on non-4.3BSD systems.

If you have troubles installing and running on a 4.2 or 4.3 based system, you may send mail to zephyr-bugs@athena.mit.edu. No assistance is guaranteed, but help may be available.

Zephyr was also originally developed assuming use of the Kerberos authentication system. If you are not using Kerberos, the system can make no guarantees about authenticity of messages. Any message which claims to be authentic will be considered authentic. Again, this version of the code has not been extensively tested.

The WindowGram client assumes presence of the X Window system, Version 11 release 2 or 3, and the Athena Widget Set and X Tool kit. If you do not have X11R2 or X11R3, the WindowGram client in this release will only support a simple terminal-based output interface (similar to the Unix *write*(1) command output).

2 Configuration suggestions

2.1 Server machine(s)

If you are using the Kerberos authentication system, you should probably dedicate machines of the MicroVAX-II class for use as Zephyr server machines. The Zephyr server is CPU-intensive. As the Kerberos system uses software-implemented encryption for secure authentication, the server consumes CPU time performing cryptographic checks on all authentic notices. If you don't dedicate a machine, you may reduce the performance level of the Zephyr server if it competes heavily with other processes for CPU time. For a small site, this will probably not be a problem, but for a large site (such as M.I.T.), the load will necessitate a fast and/or dedicated CPU.

The server is also memory-intensive. Its database may grow to 1.5Mb or more of core image at a large site. If your server machine has sufficient main memory, there should be no problem with the server. However, if the server is paging its image in and out of memory, reduced performance will result.

For reliability, you should run at least two servers. The entire database is stored only in the server virtual memory, so if all your servers crash the database will be lost.

2.2 Client machines

If you are using the *Hesiod* name service, you should set up Hesiod records to spread the hosts among the available servers. The HostManager finds servers by the following method:

- If there are server names specified on the command line, they are taken as a list of servers to use. The first such server name is chosen as the primary server.
- If no server names were specified on the command line, the HostManager resolves `(<hostname>, "CLUSTER")` (using hes_resolve()), expecting to find a string "`ZEPHYR <servername>`" as one of the answers. If found, the server name is taken as the primary server. The HostManager then resolves `("zephyr", "sloc")`, expecting to find a list of strings, each containing a server name. This list is cached until the HostManager terminates execution. If no primary server was designated, one of the servers returned by this resolution is chosen as the primary server.
- The list of servers considered for use is the server list plus the primary host.

For example, if you have three servers, you should set up the cluster information such that 1/3 of your clients end up with primary server "A", 1/3 with primary server "B", etc.

If you are not using Hesiod, you should arrange the order of host names on the HostManager command line so that the hosts are spread among the available servers, to help balance the load. The HostManager will initially try to contact the first server in the list, and will cycle through the names in order whenever a server failure is detected.

2.3 Class registry and access control lists

The class registry should contain a list of registered classes. Each class should appear on a line by itself, terminated by a colon (:). All the class names should appear in lowercase.

At least the following classes should be registered (you may wish to register more for use at your site): wg_ctl, hm_ctl, hm_stat, locate, login, zephyr_admin, zephyr_ctl, zmatch_all.

For each registered class, there should be a set of (possibly empty) access control lists (ACL's). Each ACL specifies which ZID's (Zephyr ID's) are granted the specified access. When using Kerberos, the ZID is the fully-qualified principal name (e.g. jtkohl.root@ATHENA.MIT.EDU). When not using Kerberos, the ZID is the username concatenated with an at-sign and the realm name (e.g. jtkohl@BIG-NAME-UNIVERSITY.EDU) as defined in a configuration file.

It is permissible to place wildcard specifications in the access control lists. A '*' (asterisk) will match any value for a given field. The entry '*.*@*' matches all ZID's, while '*.*@foobar' only matches ZID's in realm foobar, and 'jtkohl.*@*' matches jtkohl in any realm with any instance (or jtkohl at any site, if Kerberos is not in use).

There are four types of access. Each type of access is governed by a separate ACL. Each ACL is stored in a separate file. The files are named 'xxx-yyy.acl' where 'xxx' is a prefix denoting the access type (see the following list), and 'yyy' is the name of the class this ACL governs. 'yyy' must be all lower-case. The access types are:

1. Subscription (governed by sub-*.acl): If a user is on this list, he is granted permission to subscribe to notices of this class, pursuant to restrictions by iws-*.acl lists. If a user is not on this list, he is not granted permission to subscribe to notices of this class.
2. Instance-wildcard-subscription (governed by iws-*.acl): If a user is on this list, he is granted permission to subscribe with a wildcard instance to notices of this class. If a user is not on

this list, he may only subscribe with non-wildcard instances to notices of this class, pursuant to restrictions by the corresponding sub-*.acl list.

3. Transmit (governed by xmt-*.acl): If a user is on this list, he is granted permission to transmit notices of this class, pursuant to restrictions under iui-*.acl lists. If a user is not on this list, he is not granted permission to transmit notices of this class.

4. Instance-UID-identity (governed by iui-*.acl): If a user is on this list, he is granted permission to to transmit notices of this class with any instance. If a user is not on this list, he is granted permission to transmit notices of this class only if the instance of the notice is identical to his ZID.

If the file representing the ACL specifying an authorization type for a given registered class is missing, all authenticated ZID's are granted that authorization, but no unauthenticated ZID's are granted that authorization.

Notices with a class field set to a registered class must be authentic or they will be rejected by the Zephyr server. This is necessary so that the access check can be trusted.

The zmatch_all subscription ACL (sub-zmatch_all.acl) should usually be empty. This class is used for debugging, and matches every notice passing through the server.

The login transmit ACL (xmt-login.acl) should be wildcarded so everybody may send login notices. The iui ACL (iui-login.acl) should be empty, however, so that users cannot spoof logins. The subscription ACL (sub-login.acl) should be wildcarded, and the iws ACL (iws-login.acl) empty, so that users may subscribe to login notices for specific colleagues or friends, but not to all login notices (this is a privacy concern).

The locate, wg_ctl, hm_ctl, hm_stat, zephyr_admin and zephyr_ctl classes should have completely empty ACL files for all four access types; these classes are reserved for Zephyr functions and will never be treated by the server as a normal notice, so there is no need to permit users subscribe to or send these notices as normal notices.

The access control files as distributed implement these suggestions.

2.4 Syslog

The 4.3BSD syslogd provided with Zephyr allows you to specify that notices should be delivered via Zephyr rather than to a user or to a file. The provided manual page describes the Zephyr features of the new syslogd.

It is strongly suggested that you not use the Zephyr option in the syslog.conf file on the Zephyr server machines, to avoid a runaway loop of syslog notices generated by the Zephyr server.

3 Setting up and compiling

Before compiling the system, you should check and probably edit a few configuration files containing information used when compiling the Zephyr system.

When you have finished changing the site-wide configuration, you should create separate compilation directory trees for each machine type you support. See the following section for more details.

The following files will probably need modification for installation of Zephyr at your site:

config.Imakefile in src/config/. This file contains many configuration parameters controlling what flags are passed to the compiler, installation directories for various components, and pathnames of various utilities and libraries used during compilation. Most options are accompanied by comments explaining when and how to configure the option. Be sure to check carefully the entries between the markings denoting the likely site-specific configuration options.

This file is run through a cpp-like pre-processor and inserted into every Makefile.

This file is **copied** to the compilation trees by the `makeconfig` Makefile, so you may customize the definitions for each compilation tree. **WARNING:** We do not recommend changing installed pathnames between configurations, since some definitions in C include files must match the definitions in this file, and these C include files are **not copied** to the compilation trees.

zephyr_conf.h in src/include/zephyr/. Change the pathnames as appropriate.

If you are using Kerberos, SERVER_SRVTAB should be the name of the file containing Kerberos service keys for the Zephyr server. This file should be mode 0400, readable only by the UID of the zephyr server process (usually root). SERVER_SERVICE and SERVER_INSTANCE specify the principal and instance names used by clients to generate tickets for presentation to the Zephyr servers. You should put a service key file for SERVER_SERVICE.SERVER_INSTANCE into the file named by SERVER_SRVTAB.

If you not are using Kerberos, you may ignore these definitions, they will not be used. However, be sure to define KRB_REALM to be a realm name to be used for the clients of your Zephyr installation. This will typically be related to your Internet domain name, e.g. `"ATHENA.MIT.EDU"`. **Be sure to comment out the** `#define KERBEROS` **lines indicated by the comments in this file.** This realm name will be tacked onto your username to generate the sender field in the Zephyr notices, (e.g. jtkohl@BIG-NAME-UNIVERSITY.EDU, as above).

DEFAULT_VARS_FILE should be the name of the file which will contain the system default variables.

HM_SVCNAME and SERVER_SVCNAME are strings passed to `getservbyname()` to determine port numbers to use for UDP packets. You may change these symbolic names to anything you wish, as long as you add entries for the names used into your `/etc/services` file. The two names must resolve to distinct UDP port numbers. [At Project Athena, we use the names "`zephyr-hm`" and "`zephyr-clt`" at UDP ports 2104 and 2103, respectively.]

If you are using an Ultrix version prior to 3.0, be sure to uncomment the `#define ULTRIX22` lines indicated by the comments in this file.

file_defs.h in src/zwgc/. Change the pathname of DEFDESC to the location of your system default description file. **NOTE:** This should be in the directory ETCATHDIR as specified in *config.Imakefile*. Change the pathname of ZCTLPROG to the installed location of the `zctl` program. **NOTE:** This should be in the directory CLIENTDIR as specified in *config.Imakefile*.

The changes to this file are not strictly necessary, since the generated Makefile usually constructs these symbols on-the-fly and defines them on the C compilation line, but just to be safe we recommend you change them anyway.

zsrv_conf.h in src/server/. Change the pathnames as appropriate.

If you are not using KERBEROS, be sure to comment out the indicated lines in this file.

If you are not using Hesiod, SERVER_LIST_FILE should be the name of a file containing a list of the server machines on which you plan to run Zephyr servers. Each machine name should appear on a separate line.

If you are using Hesiod, you may ignore SERVER_LIST_FILE.

ZEPHYR_ACL_DIR should be the name of the directory which will contain the access control lists. It must terminate in a virgule (slash, /). **NOTE:** It must be the same as ACLDIR as specified in config.Imakefile.

ZEPHYR_TKFILE should be the name of the file to contain active tickets used by the Zephyr servers when authenticating themselves to each other. It must be on a writable file system, as the contents will change over time as servers stop and restart.

DEFAULT_SUBS_FILE should be the name of the file on the server machine containing the system-wide default subscriptions. It must be in the same directory as the access control files (ZEPHYR_ACL_DIR).

If you are using a version of Ultrix prior to 3.0, be sure to uncomment out the indicated lines in this file.

default.subscriptions in src/server/libdir/. This file contains the default subscriptions provided to every Zephyr client registering subscriptions. **NOTE:** the entries in this file cannot specify "macro" names as used by the *zctl* program. In addition, any recipient specified in the file is converted to the subscriber's name unless the recipient is "*", in which case the recipient is a multicast.

Access Control Lists in src/server/libdir/. The ACL's should be set up appropriately. Consult the suggestions above on what to put into the ACL's.

zephyr.desc in src/zwgc/. This is the default WindowGram description file. You may wish to modify it for your environment. The manual page for the WindowGram client (zwgc(1)) describes the grammar of this file.

zephyr.vars in src/zwgc/. This is the default variables file (which should be available on every client machine). You should be sure to examine the value of *initprogs* and change it if appropriate for your environment. If you are not using a remote file system at your site, you may wish to leave *initprogs* undefined in the default variables file.

3.1 Compilation directory trees

Once you have made any site-specific changes to these files, you should create a new compilation tree for each machine type you wish to support. To do this:

1. Create a top-level directory for the compilation, e.g. `mkdir vax`.
2. Change your working directory to that directory.
3. Make the compilation tree by using the `makeconfig` makefile:

```
make -f /xxx/src/tools/makeconfig SRCDIR=/xxx/src
```

4. If you wish to make machine-specific changes to config.Imakefile, edit the version copied by this procedure (into config/config.Imakefile, relative to the top of the build tree).
5. Type '`make world`'. This should create all the Makefiles and dependencies, and compile the entire source tree.

If you wish to install the entire system after compiling, change your working directory to top of the compilation directory and type '`make install`'. Everything which is installed by this command should be made available to all client machines, with the exception of the access control lists and default subscriptions (installed from ${cwd}/server into ZEPHYR_ACL_DIR by default), which only need to be installed on the server machines.

4 Operation

4.1 Server

In the server host boot script (e.g. */etc/rc.local*), start the Zephyr server like this:

```
if [ -f /usr/etc/zephyrd ] ; then
        echo -n "zephyrd..."                              >/dev/console
        /usr/etc/zephyrd &                                >/dev/console
else
        echo ""                                           >/dev/console
        echo "Zephyr server not found"                    >/dev/console
fi
```

The server normally runs indefinitely without intervention.

You may wish to occasionally trim the system log entries it produces. Under 4.3BSD systems, the server will log any messages to the LOG_LOCAL6 facility. Under 4.2BSD systems, the log messages are placed with other log messages in whatever files syslogd is configured to send to. Under both systems, messages are logged to LOG_INFO, LOG_WARNING, LOG_ERR and LOG_CRIT levels.

You may also wish to make the server refresh its cache of the list of servers and its cache of the default subscriptions. This is achieved by delivering a SIGHUP to the server process.

Delivering a SIGUSR1 will enable logging of debugging information, while SIGUSR2 will disable this logging. Note that much of the debugging log information is not produced unless the server is compiled with option DEBUG (e.g. `cc -DDEBUG ...`).

To examine the in-core database of locations and subscriptions, deliver a SIGFPE to the server. It leaves a text representation of the database in /usr/tmp/zephyr.db.

4.2 HostManager

In the client host boot script (e.g. */etc/rc.local*), start the HostManager like this:

```
if [ -f /etc/athena/zhm ]; then
        echo -n "zhm..."                                  >/dev/console
        /etc/athena/zhm &
fi
```

If a host is running both HostManager and server, the HostManager should be started after the server. The HostManager normally runs indefinitely without intervention.

Under 4.3BSD systems, the HostManager will log any messages to the LOG_DAEMON facility. Under 4.2BSD systems, the log messages are placed with other log messages in whatever files syslog is configured to send to. Under both systems, messages are logged to LOG_INFO, LOG_WARNING and LOG_ERR levels.

The zctl program has several options which send control notices to the HostManager. The manual pages describe these.

Delivering a SIGHUP to the HostManager causes it to request its server to flush all state information for the host. SIGTERM causes the same effect, but the HostManager exits after receiving an acknowledgment.

4.3 WindowGram client

Each user will typically start the WindowGram client upon login from a `.login` or `.profile`. To do this, the user should place the single line:

```
/usr/etc/zwgc
```

into the appropriate file (this example assumes you install zwgc into /usr/etc). Upon logout, if the system is correctly configured, SIGHUP will be delivered to the process group of the terminal and the WindowGram client will clean up.

NOTE: 4.3BSD does not normally deliver a SIGHUP to the process group of a pseudo-terminal upon logout [Normal hard-wired terminals running /etc/getty *may* work properly in this regard.]. For this reason, you may wish to inform your users to invoke the WindowGram client by placing

```
/usr/etc/zwgc -nofork &
```

into their startup files, and instruct them to be careful to kill the WindowGram client (with either SIGHUP or SIGTERM) before logging out. Under the X window system, zwgc will exit when the server shuts down.

The zctl program has several options which send control notices to the WindowGram client. The manual pages describe these.

4.4 Other clients

Other clients are described in the manual pages.