The following paper was originally published in the
Proceedings of the Twelfth Systems Administration Conference (LISA '98)
Boston, Massachusetts, December 6-11, 1998

Anatomy of an Athena Workstation

Thomas Bushnell, BSG
Karl Ramm, MIT Information Systems

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

Anatomy of an Athena Workstation

Thomas Bushnell, BSG and Karl Ramm – MIT Information Systems

ABSTRACT

This paper presents work by many developers of the Athena Computing Environment, done over many years. We aim to show how the various components of the Athena system interact with reference to an individual workstation and its particular needs. We describe Hesiod, Kerberos, the locker system, electronic mail, and the software release process for workstation software, and show how they all interrelate to provide high-reliability computing in a very large network with fairly low staffing demands on programmers and systems administrators.

Overview

The Athena system is an integrated campus network of Unix workstations for academic computing at MIT, comprising thousands of workstations and hundreds of servers. We manage Athena with a fairly reasonably sized central staff. Workstations are located in public clusters managed by the central staff, in private clusters maintained by various academic departments, in faculty or staff offices, in public hallways and lobbies, in dormitories, in libraries, and laboratories. Most of these computers are available to any member of the MIT community (it is in this sense that we use the term ‘‘public’’ in this paper – MIT does not provide any computing facilities for the general public).

An Athena workstation is a typical Unix workstation which can provide a platform for users to run their applications: sending and receiving email, running courseware, text editing and formatting tasks, and so forth. Much that an Athena workstation does is done locally as on any other Unix computer. Much is provided by contacting servers over the network. Servers do not trust the integrity of the workstation or its software which thus enables users to run their own Athena workstations, and us to publish to the entire Athena community the root password for public workstations.

In this environment, Athena handles most security issues by the strategy of serial reuse: a given user logs in, and has full and total control over the workstation. Then she logs out, the workstation cleans itself up in preparation for another user, and then waits until another user logs in and has total control. We do not attempt to fully address simultaneous-use problems, except in special cases.

Because Athena workstations are located in such a diverse array of places, and because there are so many of them, they are managed in a way to require almost no intervention from the cluster maintenance staff. A workstation requires a certain ineradicable amount of hardware support. Beyond that, it requires typing a few lines into a PROM monitor and perhaps inserting a floppy to install a workstation, and from then on it will not only install itself, but also automatically update itself for years as the Athena system develops and changes, all without needing any manual intervention.

The purpose of this paper is to describe some of the key Athena services that hold this arrangement together, from the standpoint of an Athena workstation and its maintenance. Many important Athena features, such as the Zephyr messaging service or the extensive courseware developed at MIT, are not described here.

Hesiod

Key to the operation of the Athena environment is the Hesiod directory service [Hesiod]. Hesiod provides directories for many different things, such as filesystems, users, printers, and post office servers. Hesiod lookup is done through the Domain Name System, by requesting TXT records. Originally we used the separate class HS, but now we store the records in the IN class. In this way, large databases need not be replicated across many machines, which significantly reduces administrative overhead and risk.

For example, each user in the Athena environment has a password file entry stored in Hesiod, as shown in Example 1. (Note that the actual password is not stored in Hesiod; user authentication is provided by the Kerberos system, as described below.)

Each of the tables managed by Hesiod has a name; the one above is the ‘passwd’ table. Each user also has a post office box assigned on one of many possible POP servers; the ‘pobox’ table in Hesiod is used to determine on which post office server a given user’s mail should be found. For example, the ‘pobox’ entry for one of us is:

```
POP PO8.MIT.EDU tb
```

Programs which need to read the database use library functions which issue properly formatted DNS requests to the normal DNS servers, which respond

```
tb:*:7722:101:Thomas Bushnell BSG,,E40-342d,31368,6230654:/mit/tb:/bin/athena/tcsh
```

Example 1: Sample Hesiod password file entry.

with the requested entry. The libraries then dissect the DNS reply and return the desired entry.

One disadvantage of this mechanism is that many user programs must be modified to issue Hesiod library calls at the proper places. For example, any program that fetches mail from the mail spool must have code added to request the identity of the post office server from Hesiod. In some cases this problem can be ameliorated: the Athena login process, for example, fetches the password file entry for the user at login and temporarily stores it in the password file on the local workstation, so that other programs can simply call getpwnam and have it work.

Very similar functionality is provided by the NIS service designed by Sun Microsystems. However, Hesiod is different in several important respects. First, and perhaps most importantly, it uses the very well-tested DNS infrastructure, including caching, and does not depend on the broadcast characteristics of networks as NIS does. NIS is generally used to handle information which needs to be secure, but in the Athena environment, Hesiod is never used for such information and other services are responsible for security.

NIS databases can be completely downloaded; this ability is fundamental to the operation of NIS. However, Hesiod never depends on this functionality, and BIND makes possible various kinds of discrimination on what kinds of zone transfers should be permitted. (Because of privacy considerations, we do not permit arbitrary downloading of the Hesiod tables.) Hesiod can be easily extended to support new database types with little effort, but NIS maps are basically limited to the default set.

The current implementation of Hesiod limits responses to 1K bytes, but nothing in the actual protocol has such a limitation. One serious disadvantage, however, is that Hesiod cannot be updated dynamically. NIS also has problems in this area. As a consequence, MIT generates the Hesiod databases once a day, and updates them overnight.

Hesiod will automatically be able to benefit from DNSsec and once a standard for dynamic DNS updates is approved (one is on the IETF standardization track now), Hesiod will be able to easily take advantage of it.

Despite some of these disadvantages, we have found Hesiod to function better than NIS could in such a large environment. Hesiod is never used for information that needs security. The reliability and scalability of DNS cannot be beat, and we have had no significant problems with limitations on record lengths or the delay in updating DNS databases.

Kerberos

Kerberos [Kerberos] is the system in Athena which manages authentication. It provides a global space of identification names and a secure facility for mutual authentication using those names. Athena servers and workstations then use the Kerberos system to make authorization decisions.

A Kerberos name is composed of two parts, a name and a realm. The name is composed of multiple separate strings, but no interpretation is imposed by Kerberos itself on the contents and relation of those strings to each other. (In the older version 4 of Kerberos there were exactly two strings in the name, one called the ‘‘principal’’ and one called the ‘‘instance.’’) A realm identifies indirectly the servers and authorization scope of the name. Each system using Kerberos must maintain a table mapping realm names to the associated Kerberos servers.

Kerberos servers function by creating entities known as ‘‘tickets.’’ A ticket is an encrypted data block specifying a Kerberos name, a time stamp, an expiration stamp, the Kerberos name of a service, and a secret key. By presenting the ticket to the named service, the service can be assured that the presenter has the claimed identity. Such services might be POP servers, file servers, telnet servers, and so forth. The ticket is encrypted using a key known only to the server, and the key contained in the ticket is also known by the client. In order to use a ticket successfully a client must also know the secret key, and so sniffing the ticket in transmission does not help an attack. A secret key together with its associated ticket is known as a ‘‘credential.’’

One important service is the Kerberos server itself. Tickets for the Kerberos server are called ‘‘ticket granting tickets.’’ Possession of tickets for a file server allows you to use the fileserver, and possession of tickets for a post office server lets you retrieve mail (in both cases, of course, subject to further authorization on the server). Possession of a ticket granting ticket, however, lets you obtain tickets for any other service you desire. Users obtain ticket granting tickets when they log in, upon proving to the Kerberos server that they possess a correct password. From that point, the ticket granting ticket is used as necessary to obtain service tickets for the various servers the user needs.

These credentials (tickets and associated client keys) are conventionally stored in a file in /tmp. They are not stored in user home directories because it is important for the integrity of the system that they not be transmitted by a sniffable remote file protocol. The ticket file is kept protected using the normal Unix facilities; thus someone who has broken into the workstation might steal the tickets. (We generally avoid such problems by the serial reuse model described above.) Because tickets have a built-in expiration time, the window of risk is fairly well-known. When users log out, a special program called kdestroy is used to delete the ticket file, by writing null bytes to the file containing the tickets before unlinking it, thus guaranteeing its true destruction.

Because all Athena services are authorized based upon possession of proper Kerberos tickets, no service depends on the integrity of the user's workstation. Accordingly, the root password for public workstations is advertised to the entire Athena user community. Because many of the facilities of Unix are available only to the superuser, this greatly improves the utility of the Athena system, as well as improving its security. It should be noted that we are open to the thread of a malicious user modifying workstation software to capture passwords; we believe that this risk is inherent with having unsupervised hardware and we would not actually improve security by trying to hide the root password.

Shared workstations do exist, however, the most important being the public dialup machines. These have a secret root password, and are very carefully maintained and managed. Because the Athena model is based on serial reuse, these simultaneously shared machines must be given special attention. Some facilities are not provided on them, and Kerberos tickets on them are theoretically more vulnerable.

Lockers

In a substantial distributed computing environment, it can be problematic to have all the filesystems on the campus mounted on every workstation. Many remote filesystem techniques behave poorly when servers are unavailable, and many operating systems cannot handle a large number of filesystem mounts at a single time.

Accordingly, file space in the Athena environment is separated into ''lockers'' [ATPD], each of which appears as a distinct filesystem to users, but is not mounted all the time on each workstation. Each locker has a name, and the 'attach' command will look up the name and mount the filesystem in the /mit directory. (In special cases lockers may be mounted elsewhere.)

Each user home directory is a locker. A locker might also hold one or more third-party software packages, or anything else. Student organizations (even informal or transient ones) can generally get lockers created with little trouble, and manage and maintain them.

All file storage in the Athena environment is handled through the locker mechanism. Users mount lockers with the 'attach' command, and these lockers are generally unmounted when the user logs out.

Lockers which contain programs intending to be run are generally mounted by the 'add' command, which runs attach, and then adds the appropriate subdirs of the locker to the user's command execution path. This frees users from needing to understand the ways that path search happens or the manner in which lockers organize the various binaries that have been compiled for the various supported architectures.

Lockers do not all use the same remote filesystem mechanism, making them more flexible than using NFS or AFS alone. Each locker can use whichever filesystem is most convenient, and users in general need not consider the difference.

Locker names are stored in a Hesiod table named 'filsys'. Some sample Hesiod records for lockers are shown in Example 2. These specify the filesystem protocol type, where the locker is to be found, how to mount it (writable in these examples) and where it will be mounted on the local workstation.

Each locker of course is associated with some things not covered in the scope of this paper, such as disk allocation issues on servers, backup policies, and the like. All these are hidden behind the locker abstraction, and generally need not be noticed by users at all.

Email

Electronic mail on Athena follows the now fairly standard pattern of central mail servers accessed via POP (or some other mail-specific protocol such as IMAP). Rather than a single large POP server, Athena has several POP servers and a Hesiod table (called 'pobox') is used to identify the correct POP server for a given user.

Athena workstations do not normally run sendmail daemons, but do run sendmail on demand to handle outgoing user mail. All outgoing mail is forwarded to a local mailhub, where it is immediately delivered or queued. In this way we avoid maintaining queued mail on the clients as much as possible. Sometimes mailhubs are too loaded to accept mail or are even down, in which case mail does end up being queued on the local workstations, so workstations have a periodic cron job to deliver any queued mail.

Reactivation

Each workstation periodically runs a script called 'reactivate' when no one is logged into it, which runs various housekeeping functions. The goal of the reactivation procedure is to improve the hands-off maintainability of workstations, relying on them to keep themselves in a sensible state as much as possible.

Much of what reactivate does is ordinary cleanup duties. Any stale server state, for NFS, or AFS, or the Zephyr notification system is deleted or synchronized as necessary. Stray processes are killed and attached

```
AFS /afs/athena.mit.edu/user/t/b/tb w /mit/tb
NFS /u1/bitbucket JASON.MIT.EDU w /mit/bitbucket
```

Example 2: Sample Hesiod locker entries.

lockers are detached. Special lockers that are always to be in place, called the ‘‘system packs’’ are reattached to make sure the correct ones are mounted in the correct locations. On public workstations, the password file is reset to the default and many local files are checked to make sure they match the prototype files for public workstations.

One very important function provided by the reactivate script is to check whether a software update is necessary for this workstation and to schedule it.

System Packs

Originally, Athena maintained its own operating system, a variant form of BSD 4.3. This was stored in a special locker known as the ‘‘system pack’’ and mounted on /srvd. Over time, this broke down, for two central reasons: first, maintaining an operating system is a complex and time consuming task, and as new hardware came and went it became attractive to use the vendor operating system rather than porting our own. Second, various third party software became attractive to various members of the MIT community. Much of this software is distributed only in binary form for the various vendor operating systems; using an Athena-specific operating system would complicate greatly the use of such software.

We therefore now use the standard vendor operating systems, and what used to be the srvd pack is now split into two parts. One is the os pack, mounted on /os, and the other is the srvd pack, on /srvd. The os pack contains an essentially unmodified copy of the vendor operating system. (We do need to make some modifications. Currently the packs are stored on AFS, which can’t do hard links properly, so those must generally be turned into symlinks. Also, some security and permission fixes must be made in the os pack.)

The srvd pack contains all the Athena-specific software that we build ourselves. Some of it contains programs often found in other vendor operating systems, but for which we need to have a single standard version, or because we have added special local modifications. Alss the srvd contains Athena-specific programs, such as Hesiod- and Kerberos-capable versions of mail-reading software.

Workstations’ local disks contain some software locally, but much is accessed through symlinks into the /srvd or /os packs. The decision about which goes where is made separately from the organization of the packs themselves, which contain the complete system, including those parts normally found on the workstations’ disks themselves.

Workstations determine which system packs to attach via Hesiod. At any given time, for any given architecture, we will usually have packs for two minor versions of the current major version; one in testing and one that is deployed widely. We also keep old versions online for a reasonable approximation of forever (the oldest kept online right now is from 1989).

A third system pack has recently come into use on some platforms, the /install pack. This locker contains the raw OS install contents, for example, the inst packages for Irix or the tar files for NetBSD. This locker is not used except as a resource during the update or install procedure.

Release Cycle

Most software in the Athena computing environment is available through lockers. Each locker may be maintained and released differently, and no coordination is inherently necessary. However, much software is either located directly on each workstation, or is so central to the environment, that it must be released in a coordinated manner. For this software, we have a careful release cycle mechanism in place [RelEng].

Full-blown releases take place once a year, but smaller patch releases are made from time to time as necessary. As the name suggests, patch releases are generally intended to solve minor problems or security concerns that have developed. Major changes (for example, the upgrade from Kerberos IV to Kerberos V) are reserved for the major release.

The software that is part of the release comprises two main categories: the operating system, and Athena-specific modifications. Operating system updates take place only in the major releases, and require a great deal of care and attention. We currently support Solaris and Irix, and for each we have a designated engineer who has primary responsibility for that operating system’s installation and functioning.

In addition, we have a release engineer who monitors the work of the operating-system-specific work and also manages the Athena-specific code. Many developers are able to modify this code as necessary, but having a single release engineer responsible for its overall function and for the coordination of the actual release process enables more consistent management of the release process.

A team meets weekly to discuss release issues and changes to the release; this team includes not only the release developers, but also members of the server operations team, and user support people. The major releases are carefully coordinated to academic schedules, server needs, and administrative requirements, and require extensive public documentation (posters, web pages, and so forth) describing the new features and changes associated with the release.

Update and Install

The process of updating a workstation begins with the workstation itself. Once a new release is available, the workstation’s reactivate script notices via Hesiod that it should be running a new release, and it picks a time to actually execute the update. Updates do not occur immediately upon noticing that one is available in order to avoid swamping fileservers and networks, or the specter of all available workstations

being busy updating, and a user not being able to log in. And of course, an update is taken only when the workstation is otherwise idle.

An update begins by mounting the system packs corresponding to the new release. If the new release involves no operating system upgrade, then the procedure will be fairly simple: new files as necessary are copied from the new srvd, and the workstation is rebooted. If an operating system upgrade is necessary, the procedure will be much more complex, because new shared libraries and kernels must be installed in a careful manner. In either case, however, the procedure is the same from the workstation's standpoint. Upon mounting the new system packs, a script in those packs is run which takes over and manages the entire update automatically.

A key goal of the Athena system is to provide hands-off maintenance and the orderly update procedure is a major part of that. Workstations update themselves without needing any manual intervention. In major updates, a certain number of workstations fail the update procedure for various reasons and do require manual intervention, but this is a tiny percentage of those which upgrade themselves without a hitch.

Installing a workstation is a similar procedure. A few commands are typed to the PROM monitor. The workstation is booted over the network and completely automated scripts handle everything else to install the workstation. In this way a large cluster of new workstations can be installed with very little typing necessary.

The creation of the update and install scripts often requires some subtlety. But nothing is seriously complex beyond understanding, and the entire process is the only one feasible for an environment this large. We have workstations scattered across campus, not only in general clusters, and the labor of having to attend to each one individually in an update (even if only to type a few commands) would be exceedingly tiresome.

Customized Workstations and Servers

The typical Athena workstation lives in a publicly accessible cluster and manages itself as described above. Such workstations should not be customized – to do so would defeat the uniformity expected of each workstation by Athena users. Accordingly, the periodic reactivate process attempts to notice customization, and if it detects it, then the customization is reversed by recopying the modified files.

However, for many Athena computers this would not be appropriate. Such are considered ''private'' workstations and may be customized at will by their owners. A private workstation might live on a user's desk, be part of a specialized departmental cluster, be a server of some sort, or in some other fashion need special treatment. Sometimes a private workstation has no customizations beyond limiting who may log in to the computer.

Private workstations of course must have the customization-prevention measures disabled. Also, while a normal public Athena workstation can be centrally administered and needs no direct intervention to keep it running, a private workstation, if customized, will generally require more attention.

Automated customizations are made possible through a tool known as 'mkserv'. Perhaps the most frequent customization on a private workstation is 'mkserv remote' which enables remote login access to the workstation via telnet, rlogin, or ssh.

Athena servers are also set up with the mkserv customization mechanism. (In fact, this is reason for the program's name.) An NFS server can be easily set up with 'mkserv nfs', a Hesiod server with 'mkserv hesiod' and so forth. With a hundred servers or so in the Athena environment having this ease of management becomes crucial.

The software which makes up these automated customizations is considered part of the Athena release. Generally the programs and data files the service needs are copied onto the local disk. The release update process also updates any customizations made through mkserv, which enables the great majority of customized workstations to benefit from the hands-off management style available for public cluster workstations.

Conclusion

Upgrading to a new version of the vendor operating system takes a fair amount of time. Because our release cycle is fairly rigid, significant delay might occur between the release of the operating system and our ability to get it into each workstation. Some of this time is imposed because of the complexity of creating the update and install procedure for the new release, but also some of it is inherent in our academic setting: we must not make major changes to workstation software while an academic term is in progress.

Our security model gains much clarity and agility by assuming serial reuse. Several problems still inhere. First, dialup servers are used by many users at once, and must be carefully managed. We have no general way to deal with such machines, and they require specialized support of various kinds. Second, we have no way to easily handle long-running interactive jobs. And finally, users cannot be fully confident that the software on a given workstation has not been compromised in some security-related way.

All told, however, the Athena system has borne the years very well. We manage a very large heterogeneous network with a fairly small staff of programmers and developers. It is hard to imagine how we could do so without the tools described in this paper, or their equivalents.

Author Information

Thomas Bushnell, BSG works for MIT Information Systems as a systems programmer in the Athena team. Previously he worked for over seven years as a programmer and software architect for the Free Software Foundation, writing the GNU Hurd and helping with several other projects. He studies philosophy, classics, and many other things, and can be reached via email at tb@mit.edu .

Karl Ramm is a systems programmer for MIT Information Systems, where he helps keep Athena from falling over. He can be reached via U.S. Mail at Room E40-342C, Massachusetts Institute of Technology; 1 Amherst St.; Cambridge MA 02139, while he can be reached electronically at kcr@mit.edu. He has written five different mail reading programs and has never written a windowing system.

References

[Kerberos] Steiner, Neuman, Schiller, ''Kerberos: An Authentication Service for Open Network Systems,'' *USENIX Technical Conference*, Dallas, Texas, Winter 1988. <ftp://athena-dist.mit.edu/pub/ATHENA/usenix/kerberos.PS>

[Hesiod] Dyer, Stephen P., ''The Hesiod Name Server,'' *USENIX Technical Conference*, Dallas, Texas, Winter 1988. <ftp://athena-dist.mit.edu/pub/ATHENA/usenix/hesiod.PS>

[ATPB] Lerman and Saltzer, ''Section B: Technical Objectives and Requirements,'' *Project Athena Technical Plan*, M.I.T. Project Athena, Cambridge, Massachusetts, July 23, 1987.

[ATPC] Balkovich, Parmaless, and Saltzer, ''Section C: Project Athena's Model of Computation,'' *Project Athena Technical Plan*, M.I.T. Project Athena, Cambridge, Massachusetts, September 16, 1985.

[ATPD] Saltzer, J. H., ''Section D: Evolution to the Athena Workstation Model: An Overview of the Development Plan'', *Project Athena Technical Plan*, M.I.T. Project Athena, Cambridge, Massachusetts, March 6, 1986.

[RelEng] Davis, Don, ''Project Athena's Release Engineering Tricks,'' M.I.T. Project Athena, Cambridge, Massachusetts. <ftp://athena-dist.mit.edu/pub/ATHENA/usenix/rlseng.PS>

[Track] Nachbar, Daniel, ''When Network File Systems Aren't Enough: Automatic File Distribution Revisited.,'' *USENIX Technical Conference*, Summer, 1986.