MACS: a Modular Access Control System

Mario D. Santana
mds@medleysoft.com

Blake Mitchell
blake@medleysoft.com

Abstract

The Modular Access Control System (MACS) is a group of services which provide a front end to multiple heterogeneous authentication, authorization and user information environments. The goal is to serve as a one-stop shop for user-related requests and to fetch answers to those requests from authoritative sources.

By breaking down user information into pieces for which there is one authoritative source, and clearing requests for this information through MACS, the path from any information store to any information consumer is as short as possible. Processes become more straightforward, more flexible, and easier to manage. Gains are direct (less personnel) and indirect (better services.)

1 The Problem

1.1 Introduction

Many environments, especially in the enterprise, have user databases strewn in every conceivable corner. Unix accounts, NT accounts, dialup accounts. Intranets, extranets, content publishing, mainframes, databases, discussion forums, mailboxes — everything that requires any kind of login facility needs accounts which have to be created, maintained and deleted.

Keeping these databases synchronized is time-consuming and error-prone. More often than not, the process is manual. And there's no way to share authentication state between back-ends, even when they use the same front-end.

Imagine that you have a project to create an extranet for your company's suppliers. All your suppliers are in your ERP database, but there's no clean way to get your web server to authenticate against that. So you hack a JDBC servlet together to talk to it, but there's no real password field for suppliers. So you hack out a cross-table and finally get things working.

The project ends up being delivered late and over budget, which makes management unhappy. The solution is a hack, which makes your developers unhappy. Oh, and tomorrow, you've got to get your NNTP server to authenticate against that database, too.

1.2 The Typical Solution

Often, an organization will try to use one central database for all their user data. And indeed, there are several schemes for doing just that, most notably the aging NIS and the more buzz-worthy LDAP. Unfortunately, massive centralization comes with its own set of problems of flexibility, scale, interoperability and administration.

Flexibility suffers because no single data store is well suited for all types of information. For example, while LDAP is an excellent choice for organizational and other hierarchies, relational data such as inventories are best stored in relational databases.

Scale is only a problem for very large or very busy user databases, but then it's a big problem. Interoperability will only suffer if you need to connect to a system your scheme won't support, but then it suffers badly. And administering one huge cluster of authentication servers has, if not a larger, at least a very different set of pitfalls and caveats than administering a several smaller, independent servers.

What's required is a distributed responsibility for the information and flexible, centralized access to it. Especially in the case of identity management, a relatively new area where the myriad products and ideas on the market create a confusing situation. With MACS, LDAP, NIS, and a prorietary database are not mutually exclusive; instead, they are actually tied together to create synergies out of rapidly dimishing returns.

1.3 Alternatives

Now this isn't the first time anyone has ever thought about solving this problem. For starters, getAccess™[1], SiteMinder™[2], eTrust™[3] and AccessMaster™[4] are a few commercial approaches. Microsoft's™ Passport[5] and the Liberty Alliance™[6] have marketing muscle and a lot of developers behind them. The PingID™ project[7] has made some open source progress along these lines, and is being actively developed.

The great weakness, shared by all these solutions and others, is that they dictate how user information must be stored, accessed, and managed. An application or organization built around any one of these systems can not easily interoperate with another system. With MACS, even organizations standardizing on a single emerging standard can rest assured that interoperability will never be a problem.

Other serious objections include the fact that except for PingID, none of the products are open source. This is an important objection in most cases, but it's especially critical with security and interoperability software; MACS happens to be both.

An in-depth, feature-by-feature comparison of MACS and every known alternative is available elsewhere.

[1]Entrust Technologies' getAccess™ software. See their web site.

[2]Netegrity's SiteMinder™ software. See their web site.

[3]Computer Associates' eTrust™ software. See their web site.

[4]Evidian AccessMaste™ software. See their web site.

[5]See their web site.

[6]See their web site.

[7]See their web site.

1.4 History

MACS was born from attempts to implement single sign-on across several extranets at a major software house. That part was fairly straightforward — the tricky bits cropped up when we thought about authorization. An extranet needed to store information about users who lived in a database it couldn't know about. Our solution was to name-space logins, which led to thinking about a general layer that managed the name-spaces as transparently as possible. MACS is an implementation of that general layer.

It's not clear right away what the impact of this layer will be on individual applications. Most applications will at least want to benefit from single sign-on, but not necessarily all of them. At the very least, every application's user store needs to be accessible by every other application. This doesn't need to intrude in the host application's functionality. For example, a MACS service that makes SAP user information available can be written against SAP libraries so that the SAP software proper isn't modified at all. Then every MACS-enabled application can give access to SAP users.

2 The Solution

To be able to authenticate against another application's users, you have to change your login process. Fortunately, most robust applications provide at least rudimentary hooks into that process, and MACS is designed to need only the most basic hooks to provide its services.

These services are essentially the proxying or mashalling of various data for three functions — authentication, authorization, and user profiles. Each of these is distinct and fairly independent, though it's difficult for an application to work without plugging into the authentication service, since it would have to ask for authorization or profile information about users it doesn't know how to name.

There are several levels of MACS-enablement. An application can choose to authenticate against

MACS, but do all its authorization and user profiling internally. Or any combination of the three, depending on requirements. The classic example is a web application. APXS, NSAPI, and ISAPI all provide the necessary hooks to MACS-enable almost any web application.

2.1 Resources

The resource hierarchy is MACS' object store. Everything MACS deals with is represented here, directly or indirectly. There are several types of resource objects: users, groups, name-spaces and plain resources.

Although any resource can be used as any type at any time, a resource's root ancestor determines its conventional type. Children of */macs_group* are groups, children of */macs_user* are users, children of */macs_name_space* are name-spaces, and all others are plain resources. Resources are almost always used in accordance with their conventional type.

All resources can be referred to by their resource path. Much like a file path, A resource path starts with a '/' character and specifies a node in the hierarchy by listing the resource's lineage, beginning with its root node, separated by the '/' character.

Resources which are represented directly in the resource hierarchy have a resource identifier (RID) associated with them. A RID is a string that may not contain the '/' character.

Resources which are represented only indirectly are called virtual resources. They must be children of a "real" resource. These have a virtual resource identifier (VRID) assocated with them. A VRID contains two parts: the RID of the "real" resource at which the virtual resource is rooted; and the extra-info component, which is a method-specific string used to pass custom information between clients and methods. The two parts are separated by a '/' character. A virtual resource's VRID look like its path with the "real" part of the path replaced with the "real" RID.

For example, if the resource hierarchy contains */fileshares/homes* as a leaf node with a RID of *5382*, then */fileshares/homes/accounting/tmontana* would have a VRID of *5382/accounting/tmontana*.

2.2 Architecture

The services are supplied by a Foreman process, which manages Worker processes. A Factory process listens for incoming requests and hands them off to Workers. Each service is registered with the Foreman in the configuration file, and manages the name-spaces transparently. The services are: the Authentication Server or AUS, the Authorization Server or ATS, and Resource Profile Server or RPS.

The parts of MACS that live inside an application are the Authentication Client or AUC, the Authorization Client or ATC, and the Resource Profile Client, or RPC. The parts that expose an application's information for MACS to make available are the Login Method Client or LMC, the Authorization Method Client or AMC, and the Profile Method Client or PMC. Method clients may register internally with the Foreman like services, or remotely over the network.

Note: one major flaw in MACS' design is the proliferation of TLAs. There are more to come. But while they may sound confusing when they're launched at you all at once, MACS' design is actually pretty simple when you look at it one piece at a time.

2.3 Authentication

The authentication scheme is very straightforward for the usual cases, but there's some complexity in dealing with exceptions.

Although there's a built-in authentication mechanism, MACS' login scheme is designed to live on top of other authentication schemes. You can write (or use already written) LMC services to let your people log in via MACS using logins/passwords from NT, or Unix, or SAP, or some proprietary system written in COBOL that's been in continuous production since before blood sacrifices went out of style. There is at least one login method for each authentication mechanism.

2.3.1 TLAs

First, a relevant TLA dictionary.

Login Authority (`LIA`) — The piece that requests a login on behalf of a user. This can be a CGI, it can be embedded into another application that's been MACS-enabled, it can be a PAM module, or whatever.

Authentication Server (`AUS`) — Processes login requests by passing them to the appropriate LMC. Maps entries in different userstores to and from each other. Stores login sessions for future authentication and single signon.

Login Method Client (`LMC`) — This piece actually does the authentication. An LMC registers a login method with an AUS. The AUS hands off login requests of that method to the LMC which does a database lookup, or an NT call, or consults the magic eight ball. One LMC can register multiple methods, but usually an LMC is written for each login method as they are added to the MACS installation.

Authentication Client (`AUC`) — The piece that makes sure a user is logged into MACS, by checking with the AUS. This can be a web server plugin, it can be embedded into another application that's been MACS-enabled, or whatever.

2.3.2 Protocols

The protocols are all text, so you can poke around with telnet or netcat. The pieces that talk to each other are LIA/AUS and AUC/AUS. Only the AUS/LMC one has any initialization (the LMC must register the method(s) it can handle.) Communication channels may be encrypted.

2.3.3 Logging In

So here's the scheme, step by step.

1. The user enters login, password and method.
2. LIA passes this to AUS.
3. The AUS picks an LMC and sends it the login and password.
4. LMC returns yeay or nay to AUS.
5. AUS initializes a session for the user, and returns yeay to the LIA, which does appropriate things, like redirect the user to where they were trying to go when they needed to log in.

LIA says to AUS, "Is this `login/passwd` a valid `method` login?"

AUS looks for an LMC that has registered `method`. If it doesn't find one, it replies to LIA, "No", and LIA does some appropriate things, like tell the user, etc., and that's that. On the other hand, if an LMC for `method` has registered with AUS, AUS says to that LMC, "Is `login/passwd` a valid combo?"

LMC does the appropriate lookup. If `login/passwd` are invalid, LMC replies to AUS, "No", and AUS replies to LIA, "No", and that's that. But, if `login/passwd` match up, LMC replies to AUS, "Yes".

AUS demaps the login into a username. (More on this later. Suffice to say that the method-specific login is turned into a "canonical" name by which MACS knows this user.) AUS creates a session, and says to LIA, "Yes, and here's the `sessionkey`."

LIA does something appropriate like let the user continue accessing the resource.

2.3.4 Demapping

A central feature of MACS is login name-spaces. Here's a quick rundown of how that works.

1. Joe Random Hacker might have several logins: jrh on Unix, hackerjr on NT, joerh on SAP, and hootinannie on the IRC server. Right now, he might be logging into MACS for the first time. He submits a `login/passwd` pair for a particular method (say jrh/cantcrackthis for Unix) to LIA, which forwards it to AUS. AUS finds that this LMC over here registered the Unix method and asks her about jrh/cantcrackthis. LMC says it's good. Now comes the demap.

2. AUS looks for a username that's mapped to "jrh on Unix", but finds that there's no match, because Joe's never used that login to get in through MACS before. Actually, he's never used **any** login to get in through MACS before.

3. Now the AUS has to create a mapping for "jrh on Unix". AUS asks LMC how to map users for this particular method. LMC returns a rule that has been configured during the MACS implementation, and AUS looks for existing MACS users that have the same data. These rules are very flexible and may include user information from the userstore of another method. If a match is found, AUS creates a mapping from "jrh on Unix" to that username. In our case, of course, it finds none, because Joe's never logged in via MACS and therefore has no username there yet.

4. So AUS creates a MACS username for Joe. It will try to use jrh but will munge the username to make it unique. Let's say the username becomes jrh123. AUS then makes a mapping to "jrh123" from "jrh on Unix".

5. Finally, AUS creates a session for Joe as jrh123.

Now the next time Joe logs in via MACS using jrh/cantcrackthis on Unix, AUS will find a username that's mapped to "jrh on Unix" in step 2 above and will skip steps 3 and 4.

But if Joe decides to use a `login/password` for a different method (say hackerjr/easypwd for NT), AUS will find a username to map to in step 3, because the LMC that registered to service NT logins tells AUS that "hackerjr on NT" has a profile that matches what AUS already has for jrh123. AUS makes a mapping to "jrh123" from "hackerjr on NT" and creates a session for Joe as jrh123.

2.3.5 Authentication

So now getting an application to authenticate against MACS is simple: ask the AUS if there's a valid session. Talking to the AUS almost always entails customization of the application. This customization is what we call the Authentication Client, or AUC. There are AUCs available for APXS, NSAPI, ISAPI, and WebCrossing, with more to come. They are, by design, easy to write; moreover, they can use protocols other then MACS native. For example, the WebCrossing AUC talks exclusively to the XRS, which front-ends MACS with an XML-RPC protocol. At the moment there is an XML-RPC and a SOAP frontend, with more to come.

If an user attempts to access a protected resource without a valid session, the AUC sends the user to the LIA. The user logs in, and the LIA sends the user back to the protected resource. This time, the AUC finds a valid session and the user is authenticated.

2.3.6 Miscellaneous

There are a couple of pieces in place, designed for easy customization, to handle miscellaneous tasks associated with authentication. These pieces, which are part of the administration interface, are:

Note: Documentation for the administration interface is available elsewhere.

- The User Registration Agent (URA): requests registration information from the user and creates a MACS account. The account will be authenticated via the internal MACS method (called "NULL"). It will belong to the guest group. The new user is logged into MACS and is redirected to the referring URL.
- The Registration Change Authority (RCA): allows an existing MACS user to modify their profile information, including their password.
- The Lost Password Authority (LPA): The user is asked for various information about the account in question. Matching accounts are displayed, and the user selects one. The account's password is reset to a random string and the string is sent to the email registered for the account. The account's password-change flag is set, forcing the user to change their password at next login.

2.4 Authorization

The ATS is in charge of authorization. Applications must implement an ATC to authorize against the ATS.

2.4.1 Overview

Authorization in MACS is based on a simple, hierarchical ACL scheme. All permissions refer to groups, so a user has no permissions on anything except by virtue of group membership. Groups contain users and other groups. Each node in a hierarchy of resources has a list of groups attached to it, and permissions for each group. Permissions are inherited by children in the hierarchy of resources. Each child can explicitly add or negate some or all permissions for some or all groups, modifying the permissions of the parent. This new set of permissions is inherited by children further down the resource hierarchy.

2.4.2 Groups

Multpile hierarchies means that the ATS can directly protect multiple sets of resources. One such set of resources is the groups themselves. Modifying who belongs to which groups is an action that must be authorized by the ATS, so that only certain groups can modify the contents (i.e., the members) of any particular group. This logic isn't circular, I swear.

2.4.3 Permissions

There are a set of internal permissions: read, write, admin, and own. User-defined permissions are possible as well. For example, an application may want to check for an execute permission. Also, a user-defined permission may be mapped to an Authorization Method Client or AMC. These base authorization decisions on information outside the ATS' control, e.g., on a SAP database or an NT query.

2.5 User Profiles

The facilities built into MACS for maintaining user information are extremely flexible. The important thing is to present a simple and consistent interface to profile data from various sources. MACS aims to be authoritative, or at least to need to be authoritative, for as little information as possible. This means the responsibility for maintaining user records is up to the individual applications.

A PMC specific to each user store makes the information from that user store available to the RPS. An application is extended to query the RPS for user information by embedding an RPC in it.

A profile attribute is a key/value pair attached to a resource. MACS generally uses name-spaces as the key. The value is fetched by the appropriate PMC for the applicable method. Each name-space has a default method, which is inherited by the children of that name-space in the resource hierarchy. However, by explicitly specifying a method in the profile query, any method may be used to service a given profile query. In this way, a resource may have multiple values for any given profile name-space, stored in various data-stores, and fetched by various PMCs.

3 Case Studies

Here, it may be useful to present an overview of MACS in use. To date, MACS has been implemented in production to protect several applications, but only one that is not web-based. So we'll look at MACS in action for one web-based and one non-web application.

3.1 An Extranet Example

An extranet needs to have available, and be protected based on, certain data about users. This data is spread over multiple stores. In this case, all the information about each user is in a single store.

3.1.1 Authentication

An APXS AUC is plugged into the Apache serving the extranet. It authenticates users based on custom apache configuration directives, and makes the user's identity available for the rest of the request. If the user has no valid session, she is redirected to the LIA, with instructions to return after logging in.

3.1.2 Authorization

After the AUC, an APXS ATC is run. It asks the ATS if the user (as authenticated by the AUC) can perform the function (as specified by the request) on the node (as specified by the url.) If so, nothing happens; otherwise, the user is redirected to an access denied page.

3.1.3 User Profiles

Finally, the UPC contacts the UPS and exposes user data to the rest of the application. The user data to be exposed is configurable per-url with custom apache directives.

3.2 A Proprietary Application

Proprietary software implementing a discussion board needs to be able to authenticate against users in a different database. The software has a proprietary scripting language with hooks to do all sorts of things, but we want to interfere with the software's operation as little as possible. User profiles are to come from MACS, but authorization is local.

The software has built-in support for XML-RPC; so, rather than implement socket communications with a limited scripting language, we frontend all the calls to AUS, ATS and UPC through the XRS.

We decide to go with a hybrid storage scheme, where each AUS user is mapped to a local user. The steps in the authentication process are

1. Check for a local session. If the user has a valid local session, skip the following steps.
2. Check for a MACS session. If the user has no valid session registered with the AUS, redirect her to the LIA.
3. Create a local user. If the user is unknown, create a local user with a login that maps to the MACS login name.
4. Populate the user's profile. Fill in or update the user's profile information from the UPS.
5. Create local session. Log the user in to the local application, and proceed as usual.

This approach was feasible because the parts of the application that allow the user to change her profile information are overwritten with pointers to MACS' pages for that purpose.

4 The Status

MACS in various states is now in production protecting several web applications and at least one non-web based service. So far, there have been relatively few real implementation problems, thanks to the transparency of perl and simple ASCII protocols.

An LMC implementing authentication against NT is available. APXS, NSAPI and WebCrossing AUC code, as well as the beginnings of an ISAPI module, exist today.

5 The Future

After 4 years in production, MACS is maturing quickly. Its interfaces and internal structures are stabilizing at a rapid rate. Transparent distribution of the services for high-availability environments and encrypted, two-way authenticated communications are feasible now. Much of the remaining work is polish and documentation. And there will always be more systems out there to MACS-enable by writing clients and methods for them.

Most importantly, MACS needs the feedback of the community. It has been so far almost entirely the brain-child of two people. If this document does nothing more than attract some insightful comments, it will have done its job.

6 Acknowledgments

The authors would like to thank Autodesk for their release of MACS under the GPL; Matt for his, uh, leadership; SourceForge for their service; and Adam and Brett for their mints.

7 Availability

MACS is licensed under the GPL. It is available via CVS and FTP from Source Forge. For more information, see the project's SourceForge home page at http://sourceforge.net/projects/macs

References

[netscape-sso] “Planning and Deploying a Single Sign-On Solution.” Netscape Communications Corp. A description of single sign-on functionality as implemented in Netscape SuiteSpot^{TM} . See http://developer.netscape.com/docs/manuals/security/SSO/sso.htm

[XSSO] The Open Group’s XSSO (PAM) specification, available at http://www.opengroup.org/security/sso/