

# A Proposed Security Architecture University of Hawai'i Administrative Information Systems

## Objective:

The information systems of the University of Hawai'i are valuable assets that must be protected. At the same time, a wide variety of University personnel need to be able to access these systems in order to perform their duties. Additionally, in the future, there may be a need to allow appropriate access to people outside of the University. This Security Architecture identifies requirements and technologies necessary to provide this access while still protecting the University's information resources.

## Introduction:

This architecture will cover facilities for these key security components: authentication, authorization, data protection, administration, and auditing. This document will discuss these components and the technologies necessary to implement them. The following table summarizes these components:

Authentication	Authorization	Data Protection	Administration	Auditing
UserIDs & Passwords	Proprietary Systems: RACF, NIS/NIS+, Domain ACLs (NT, OS/2, Novell, etc.), etc.	Backup Systems: DFSMS, FDR, Tivoli Distributed Storage Manager, etc.	Domains	Audit tools (report writers, etc.)
Security Tokens	Standards-based Systems: DCE	Utility programs: tar, IEBCOPY, IEHMOVE	Access Control	Monitors & Filters
Smart Cards	Firewalls	Anti-Virus software	PKI: Certificate Authority	Integrity
Biometrics				Intrusion Detection
PKI				syslogd auditing tools
Kerberos				
RADIUS				
PAM				

## Guidelines:

The following guidelines apply to the security architecture. These guidelines exist to ensure the integrity of all systems involved. These guidelines are very general and apply to all topics within this document. Topic-specific guidelines will be included in the topics to which they apply.

1. The level of security utilized to protect a resource should be commensurate to the value of that resource to the University and must be sufficient to contain risk to an acceptable level.

Rationale:

- \* Security costs potentially increase beyond the value of the assets protected. Don't use more security than is required!
- \* Different transactions will have different security requirements.
- \* Requirements for security vary depending on the information system (e.g. Financials vs personal home pages), connection to other systems (e.g. the mainframe connects to the State Criminal Justice Data Center), sensitivity of the data (e.g. personal history information vs public office phone numbers), and probability of harm.

Example:

Read-only access to an on-line directory containing only public information should not be protected using high-end facilities like strong authentication security tokens. Conversely, update access to the financial system should be.

2. Integrate systems using open standards.

Rationale:

- \* Security services will have to span a number of different hardware and software platforms. While they each may have their own, often proprietary, security implementations, the "glue" that binds them to other systems and applications must be able to work consistently across all platforms.
- \* Security services will be provided as part of the infrastructure. Basing these services on open standards assures future interoperability and consistency.

Example:

3. Integrate existing native services rather than bypassing or overlying them where possible.

Rationale:

- \* Native mechanisms are generally well defined, robust and efficient.
- \* Overlaying or bypassing the native security systems will generally result in a higher probability of compromising the integrity of the system.
- \* Utilizing defined interfaces (e.g. exit routines, subsystems, etc.) minimizes the likelihood of introducing security exposures.

Example:

Web-enabling "legacy" applications can be done securely by making use of the existing security system to control access (e.g. RACF). "Screen-scraping", RPC, and SQL access can be set up to require the end-user to login thus preserving security and auditability.

A poor way to implement this would be to set up a single "super user" which a web-based application uses to login and execute transactions on behalf of web users. If the web server is compromised (either accidentally or by design), "super user" access may be improperly given to an unauthorized user. Auditability would be lost because:

- all transactions will now run under the ID
- multiple personnel (i.e. developers) will have access to this ID
- this ID and its related credentials will be stored somewhere on a server. compromising the server will compromise the entire application (i.e. damages will not be limited in scope).

4. Use existing services that are consistent with open standards where possible.

Rationale:

- \* Many common applications already have their own security services.
- \* If it works and it fosters interoperability (i.e. utilizes open standards), make use of it.

Example:

Web browsers have built-in support for SSL security. Making use of SSL does not hinder interoperability in the future because it is a widely-used, open standard. Changing components at either end (i.e. the web browser or the web server) would not be affected because the SSL standard is supported across all implementations.

5. Use the best tool for the job that you can afford.

Rationale:

- \* Using a tool that meets 90% of your needs vs. a tool that meets 50% of your needs will simplify implementation by minimizing the amount of additional tools and customization you will need.
- \* The best tool is usually the one that does the job you require it to do and does it reliably.
- \* "Cost" is seldom directly related to "Price".

Example:

In the proposed implementation that follows, RACF will satisfy all of our authentication needs. Implementing authentication via a Kerberos server on a UNIX system would require replication and synchronization between RACF and the Kerberos server in order to provide the same functionality. This will require additional software and introduce additional security, maintainability, and availability exposures.

6. Implementation should include ALL active information systems if possible.

**Rationale:**

- \* Systems that are not included in the implementation will have to have their security overridden or bypassed in order to make use of them.
- \* Excluding systems will inhibit "buy in" to the architecture by all involved parties. Why invest in an architecture that obviously does not solve the problem?

**Example:**

Implementing Kerberos on a UNIX system that does not understand RACF-based systems basically excludes all of the RACF-based systems from the architecture. In order to make use of those systems, users need to either have a separate path into those systems or they will need to contrive ways to bypass RACF in order to get their work done.

7. Centralize as much as is possible.

**Rationale:**

- \* Putting everything in one place enhances your ability to audit/monitor the facilities.
- \* Centralized facilities are easier to maintain and reduces the likelihood of having things "fall through the cracks."
- \* Backup and Recovery facilities are simplified. For example, in a distributed model, should one component require recovery, how do we re-synchronize all of the disparate components?

**Technical Topics:**

This section discusses the technical issues and candidate technologies that may be used in implementing this architecture.

**Authentication**

Authentication is the process of identifying a user and verifying that user's identity. The most common form of authentication used is a user-ID and password. This mechanism works well but has many exposures, especially in today's networked environments. Other mechanisms have been developed to address these issues that may be used in combination with user-ID and password to enhance security. These mechanisms include (but are not limited to) smart card technologies, biometrics, security tokens, and special protocols like Kerberos.

Authentication technology: User-IDs

User-IDs and passwords have existed for a long time; even before computers came into use. Basically, User-ID/password is a "shared secret". A User-ID is a unique name that is used to identify you. The password is a secret shared by you and the person that needs to authenticate you. For example, you need to authenticate at a guard post to get into a

secured facility. You approach the Kanto Mura main gate and give the sentry your User-ID "I am John Smith from 5<sup>th</sup> Air Force, Strategic Air Command, Fuchu City". The sentry asks your password and you give it to him. If that password doesn't match what is in his codebooks, he shoots you.

Today's computer systems are much friendlier than those old sentries but they are performing the same basic function. When you try to get into a secured system, it asks who you are and then asks your password. If it doesn't match what's in your security database, it denies you access to the system.

Authentication technology: Kerberos

User-ID/password schemes in a networked environment share a common problem with the old guard-at-the-gate scenario: when you give your User-ID and password to the guard, somebody could be eavesdropping. If someone on the network "hears" your User-ID and password, then they too can authenticate themselves as you.

Kerberos is a software technology designed to mitigate the effects of someone listening in on your User-ID and password exchange. Kerberos is one of a class of mechanisms called "3<sup>rd</sup> party authentication systems." Basically, you share a secret with the Kerberos server: a private key. The application server that you want to authenticate to also shares a secret (private key) with the Kerberos server. You do not share a secret with the application server.

In order for you to authenticate with a given application server (we'll call it "S"), you contact Kerberos saying I want to talk to "S". Kerberos generates a "ticket" which contains a session key (and other miscellaneous identifying information) encrypted in S's key (so you can't read it). That ticket and the session key it contains is encrypted in your key and sent back to you. If you don't know your private key, you can't extract that ticket and that session key. That's how Kerberos knows you are who you say you are.

You then encrypt your identifying information (the same information Kerberos encrypted into the ticket) using the session key you just extracted and send it along with the ticket you got from Kerberos to S. S extracts the session key from the ticket (again, if S doesn't know the private key he shares with Kerberos, he will not be able to extract the session key and that's how you know he is who you think he is).

Using that session key, the server extracts your identifying information and compares it with the identifying information sent in the ticket. If they match, you've just been authenticated to S. If not, he has to shoot you (aren't you glad computers don't carry firearms?).

Authentication technology: Smart Cards

Smart cards are credit-card-sized devices that contain memory and a computer processor ...

Authentication technology: Biometrics

Biometric technology uses unique features that can be measured on your physical body to authenticate you. Examples of such technologies include fingerprint scanners, retina scanners, iris scanners, handprint scanners, voiceprint scanners, etc.

They all work basically the same way. Some physical feature of your body is converted into a very large number and that number is used to identify you. For example, fingerprint scanners generally map your fingerprint onto a very fine grid. The location of ridges on your fingerprint determines what gridpoints are "0" and what gridpoints are "1". That binary stream is then converted into a very large integer and becomes your ID.

There are a number of problems with these technologies but their accuracy has improved to the point where they are viable for use as authentication mechanisms. Currently, their biggest drawback is their cost.

Authentication technology: Security Tokens

Security Tokens are similar to Smart Cards in that they also contain memory and a processor. The difference is that these tokens are highly-specialized. Examples of security tokens are RSA/SecurID, ActivCard, and CryptoCard. The way they work is that they generate a seemingly random, unpredictable number that the authentication system will know.

For example, in the case of SecurID tokens, the token generates a number by combining the token's serial number and the current date and time and encrypting it. The authentication software running on the SecurID server is synchronized and thus knows what numbers should be displaying on the token at any given time. The algorithm used to do this is such that it is extremely difficult to predict the sequence of numbers that will be displayed on any given token at any given time.

Authentication technology: RADIUS

*include brief discussion on the RADIUS protocol (what it is, how it works, and what problem it solves).*

Authentication technology: PKI

*include a brief discussion about cryptography in general. include discussion on private key technologies to contrast with public key technologies. discuss what problems PKI is designed to solve.*

Authentication technology: PAM

*include a brief discussion of the PAM interface (what it is, how it works, and what problem it solves).*

### **Authorization**

Authorization is the process of determining whether or not an already authenticated user is allowed to do what he/she is trying to do. For example "is the user authorized to cut a Purchase Order?" Authorization is generally managed through a structure of access controls. Access controls generally determine if the user has the ability to see and/or modify data resources (e.g. databases, files), execute programs (e.g. transactions, batch programs) or connect to systems (e.g. Production, Test, Q/A, etc.).

In general an authorization request such as "is the user allowed to cut a Purchase Order" may be translated to access controls such as:

- does the user have access to the Production System?
- does the user have access to the Financial application?
- does the user have access to run the purchase order transaction?
- does the user have update access to the necessary account code?

and so forth.

Authorization technology: Proprietary Systems

*discuss various proprietary security systems (RACF, NIS/NIS+, Domain ACLs, etc.)*

Authorization technology: Open Standard Systems

*discuss the only remaining open standard security system (DCE)*

Authorization technology: Firewalls

*include a brief discussion on firewall technologies (what they are, how they work, what problem they solve).*

### **Data Protection**

*include discussion on data protection:*

- *data integrity (preventing tampering, etc.)*
- *data backup and recovery*
- *business resumption planning*

Data Protection technology: Backup Systems

*discuss full-function backup and recovery systems (DFSMS, FDR, Tivoli Distributed Storage Manager, etc.)*

Data Protection technology: Utility Programs

*discuss various utility programs that may be used for minimal backup and recovery*

*functions (tar, IEBCOPY, IEHMOVE).*

Data Protection technology: mksysb

*discuss proprietary utility programs that fit somewhere between generic utilities and full-function backup systems.*

Data Protection technology: Anti-Virus Software

*discuss Anti-Virus software (what they are, how they work, what problem they address)*

### **Administration**

If there are simple truths in life, this is one of the most important to remember: "change happens." One of the biggest tasks that this architecture needs to address is having the security systems keep up with those changes.

Employees are hired, transferred, re-assigned, and terminated regularly in most organizations. These changes usually result in changes to user accounts, privileges, authentication and security policies, etc. These changes must be quickly reflected in the security environment.

Why? Let's look at a worst case scenario: an employee with extensive access is terminated for cause (i.e. "canned"). If his security credentials and privileges are not revoked immediately, think of the damage he can inflict on the organization's information systems.

Not all situations are so dramatic, but even in everyday situations there are reasons for concern. For example, a Fiscal Officer (FO) transfers to another department. If that FO's access to his former department's accounts is not removed, he could accidentally be spending his former department's funds for purchases for his new department.

While the task is difficult, there are ways to minimize the complexity of administration:

- \* Structure responsibility for security. For example, create an organization structure with defined responsibilities.
- \* Simplify the complexity of security requirements. For example, use role-based administration rather than user-based. This allows you to build "templates" for the various roles and assigning the appropriate privileges by fitting a user to that role.
- \* Create security "domains" by grouping common security requirements and policies.
- \* Invest in tools to simplify the administration function.

Administration technology: Security Administration

*discuss the security administration process.*

Administration technology: Domains  
*discuss the concept of security domains and how to implement it.*

Administration technology: Role-based vs User-based security  
*discuss the concept of user-based security. discuss the concept of role-based security. give example how both models can be implemented within a security system like RACF.*

Administration technology: Access Control  
*discuss the concept of access control and how it is generally implemented via RACF profiles and via ACLs.*

Administration technology: PKI/Certificate Authority  
*discuss the concept of Certificate-based authentication. discuss the concept of "trust". discuss the concept of Certificate Authorities.*

### **Auditing**

There must be a way to ensure that the implemented architecture is performing the job we intended it to do. The architecture must provide for a way to collect tracking and monitoring information pertaining to successful and unsuccessful interaction with the information systems. Accountability for interactions must be tied to specific users. The architecture must be able to audit all significant security events including authentication, access, and administration.

Auditing technology: Audit tools  
*discuss generic audit tools (exception reporting, user privilege monitoring/reporting, program privilege monitoring/reporting, etc.)*

Auditing technology: Monitors & Filters  
*discuss tools to actively monitor systems for security exceptions. discuss tools for filtering security event data.*

Auditing technology: Integrity  
*discuss data/system integrity issues and tools for verifying integrity.*

Auditing technology: Intrusion Detection  
*discuss intrusion detection concepts and tools to monitor for intrusions.*

### **Proposed Implementation:**

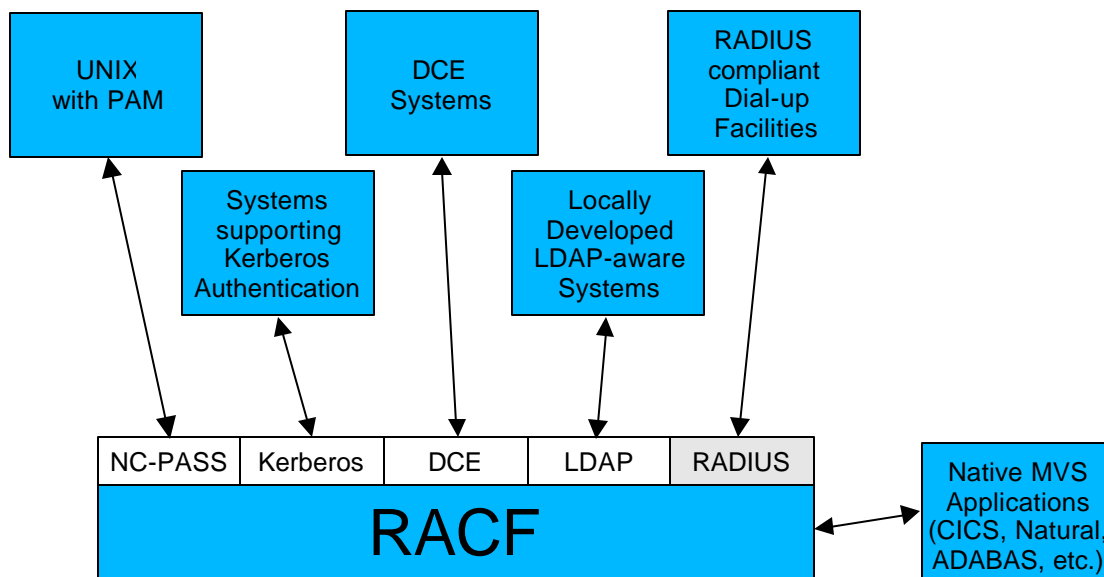
The following is a proposal for the implementation of a security architecture at the University of Hawai'i. This proposal takes into account the various production systems currently in use and the various software and hardware platforms that they run under.

At the heart of this implementation, is the RACF Security Server running on our OS/390 system. Of all the security systems available on all of the platforms in use at the UH, RACF is the only system that supports a variety of open, standards-based, security mechanisms. Using RACF will allow us to integrate our legacy mainframe-based applications with new, distributed, web-based applications currently being developed.

This discussion will focus on how RACF can be used to support the various components of this architecture.

### Implementation: Authentication

The following illustration depicts a logical view of RACF as an authentication engine:



*Illustration 1 Security Architecture Implementation: Authentication Services*

RACF under OS/390 is an MVS external security system, an MIT version 5 compliant Kerberos server, a DCE Security Server, and an LDAP server all rolled into one. Because all services are provided out of a single security database, no synchronization is necessary between RACF, DCE, LDAP, and Kerberos. For example, when updating user credentials in LDAP you are also updating the Kerberos, DCE, and RACF credentials as well.

For systems that cannot use one of the open standards-based authentication protocols (i.e. DCE, Kerberos, or LDAP) NC-PASS (a product we already have) can provide a TCP/IP-enabled process for providing this authentication service. For example, any UNIX system can use the Sun Microsystems-developed PAM interface for calling such external facilities. Also, application systems such as PeopleSoft may not directly interface with open

standards-based protocols. However, most supply installation exit facilities to implement this capability. Such systems may use this NC-PASS facility or interact directly with something like Kerberos.

For user logins, just about every UNIX implementation supports the use of Kerberos technology. AIX systems tend to favor the DCE Security Server but can use Kerberos as well. NIS/NIS+ systems use a version of Kerberos that can interoperate with MIT-based Kerberos servers.

Systems such as dial-up servers and terminal servers often use specialized protocols like RADIUS or TACACS+ for authentication. Symantec offers an NC-PASS add-on that provides RADIUS Server capability for the RACF Security Server.

### **Implementation: Authorization**

Authorization is a tricky business. Some applications have authorization controls built into the application itself (e.g. FMIS, PeopleSoft, etc.). Some may already be using ACLs via the native security system on the platform on which they run (e.g. UNIX applications using NIS+ security). Still others may be using popular rules engines (e.g. one of the Buzzeo demonstration projects used the Blaise Rules Engine).

Where possible, centralizing ACL support through the same LDAP engine used for authentication is desirable. The RACF LDAP server requires a DB/2 database in order to serve ACLs. While this poses no technical problems, there may be issues with this from our management. Hence, it may be necessary to set up an external LDAP (e.g. use the iPlanet LDAP) engine and synchronize it with the RACF LDAP authentication engine. This again introduces its own problems, but is something that can be dealt with.

For applications that utilize proprietary ACL or rules engines, synchronizing them with the authentication server may be the only workable implementation strategy (other than re-tooling the application to use LDAP-served ACLs).

Realistically though, none of our Solaris systems are using NIS+ (otherwise, they'd be authenticating via Kerberos instead of using the passwd file) and hence probably are not making use of ACLs. Our WinNT- and Novell-based applications are all being developed to be LDAP-aware. The rules-engine-based Buzzeo applications were merely demonstration projects that have not been moved into production. So, in our case, implementation choices boil down to two scenarios:

1. Acquire DB/2 for OS/390 and serve ACLs directly from the RACF LDAP server.
2. Utilize the iPlanet LDAP server to serve ACLs and synchronize it with the RACF LDAP server.

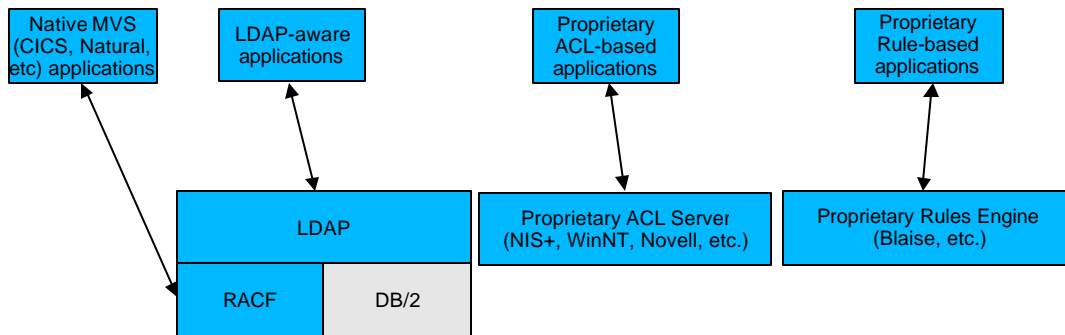


Illustration 2 Security Architecture Implementation: Authorization Services

**Implementation: Data Protection**

*discuss the use of DFSMSdss and DFSMS/ABARS for mainframe-based applications. discuss the use of the Tivoli DSM (formerly ADSM) for distributed applications and applications using the OS/390 UNIX Hierarchical File System.*

**Implementation: Administration**

*discuss administration via RACF where applicable. discuss possible scenarios for managing administration of other proprietary environments (will need input on this from users of other proprietary environments).*

**Implementation: Auditing**

*discuss reporting tools (RACF Report Writer, DFSORT, SMF, etc.) available on the mainframe. discuss reporting tools available on other systems (I'm gonna need help on this, I'm not aware of any SMF equivalent on UNIX, NT, etc.).*

**Why do it this way?**

Why implement the architecture this way? The most obvious reason is that this implementation is the only one available that will accomodate everything from our legacy, mainframe-based applications to our new java, web-based applications without compromising security on any given system. RACF and its kin (CA/Top Secret and CA/ACF2) are the only external security systems today that are designed to work with CICS and other mainframe subsystems as well as supporting open technologies. Security systems such as those found on UNIX, NT, and Novell environments so far do not offer support for mainframe applications and most do not support cross-platform use.

Cost to implement is also a factor. LDAP authentication, Kerberos authentication, GSS-API support, and DCE authentication are built-in to the RACF Security Server. We have it now on our OS/390 2.10 system and only need to configure it in order to start using it.

There are no additional expenditures necessary for us to implement cross-platform, cross-application, "single signon" authentication services using RACF. There are some "people costs" involved: configurations need to be changed to utilize services like Kerberos and the "name space" needs to be consolidated. But these are things that have to happen anyway no matter how we choose to implement.

Also of value from this implementation is the effort involved in the maintenance of the authentication service. Maintenance (adding, updating, deleting users and profiles) can be done via standard RACF, LDAP, GSS-API, or DCE facilities. Since all interfaces are actually a "view" of RACF, updates through any interface will update all interfaces. For example, when you change your password via RACF that change is reflected immediately to Kerberos, LDAP, GSS-API and DCE because they are all served out of the same RACF database.

One of the reasons to consider using a mainframe for anything is its strength in the areas of reliability, availability, servicability, scalability, security, and integrity. Because of the "business critical" nature of this function, the mainframe's characteristics make it an ideal environment for implementation.

This implementation, being based on utilizing open-standards-based components, also has the benefit of not tying us to any particular platform. If we migrate all of our CICS and MVS batch applications off of our mainframe and therefore no longer require native RACF support, then the security server can be replaced without change to any of the other systems with a generic system running on any other platform. Should we decide to replace all of our Solaris systems with Linux systems, no change will be necessary to support the new systems.