The Role of Kerberos in Modern Information Systems

Introduction

Achieving adequate security for today's information systems has proven to be a very hard problem. In many cases, the problems of security have been made even harder by a history of treating security as an afterthought. Too often, instead of taking a comprehensive, architectural approach to security, our field has deployed security measures and technologies as individual bolt-on solutions to very narrow problems. Over time, the resulting mosaic of security-related technologies becomes brittle, difficult to extend, time-consuming to manage, and not necessarily effective.

This document takes an architectural look at security, but it is not a reference manual for security architects. Instead, it focuses on the role of the Kerberos authentication system as part of the overall *ecosystem* of security technologies, services, and software components likely to be encountered in a typical modern distributed-systems environment. It is intended for those who design, build, and manage enterprise computing infrastructure, as well as for those who develop software intended to operate in such environments. This document should answer many of the questions about how Kerberos fits into modern information systems, and how it relates to other vital system services.

This document is explanatory rather than prescriptive: it describes the context in which Kerberos exists, and provides some guidance, though that is not its purpose. Instead, it is intended as background for two other documents that *do* make recommendations, one focused on systems and platforms, and the other on applications:

- *Recommended Practices for Deploying and Using Kerberos in Mixed Environments (pdf)*
- *Best Practices for Integrating Kerberos into Your Application (pdf)*

Kerberos is a security technology that is mature and that is increasingly delivered as an integral component of modern systems. As such, it can be an important element of an overall security framework. However, because different developer communities have approached Kerberos integration in different ways, new challenges emerge when using Kerberos, especially in the kind of dynamic, open-ended, mixed platform enterprise computing environment that is typical today.

This document frames those challenges in the larger context of information security. It begins with a focus on *access control* as an overarching goal. It then describes the various *security services* that contribute to effective access control, and discusses the issues that arise in providing those services. Next, it describes the role of Kerberos in providing those services, focusing on the ways in which Kerberos integrates with the other technologies under a variety of scenarios. It pays particular attention to *directory services*, which, while not part of Kerberos, are often integrated with it. Finally, it introduces some details about the way Kerberos is used within specific operating system platforms such as Microsoft Active Directory, Open Directory, and others.

Access Control as a Primary Objective

Ultimately, a major purpose of information security technology is to implement *access control*: to enable authorized access to information, resources, and services and to prevent unauthorized access. Access control encompasses *policies* that indicate under what circumstances access is to be granted, and *mechanisms* to implement those policies.

Access Control Policies

An access control policy incorporates the following factors, either explicitly or implicitly:

Authorities and Regimes

An *authority* is the entity (person, role, or organization) responsible for establishing the other elements of an access control policy: defining and classifying resources, determining authorizations, defining allowed means of access, specifying confidence levels, etc. A given authority has responsibility for the policy elements that govern a specific set of users and resources, typically under common administrative control, for example, *all the users and hosts in the accounting department of Some Corporation*. This paper refers to such a grouping as a *regime*, although the terms *administrative realm* and *administrative domain* are also encountered. Since *realm* is also used in a technical sense by Kerberos, and *domain* by Windows Active Directory, the term *regime* is less overloaded.

Regimes can be related to each other hierarchically: the *Some-Corp Accounting Department* is part of the larger *Some-Corp.* They can also be related in other ways: the policies for *Some-Corp Accounting Department* might grant access to certain resources to employees of an outside company that provides contract accounting services to Some-Corp. In some contexts, external regimes, such as regulators or auditors may contribute to policies and impose certain practices.

Resources

A security policy must define the *resources* to which it applies; examples of resources include computers, abstract services, files, network connections, and information. Resources may be defined via identifiers (*e.g., the host at address 172.24.37.1)* or by attributes (*e.g., all employee records where department number is 47).*

Resource Classification

Resources can be classified according to the level of risk, costs, or consequences associated with managing access. Examples of classification include *top secret, subscription required, downloaded file is an application*).

Access Context

An access control policy often makes reference to the circumstances under which access is requested or the means by which access is provided. For example, "during normal business hours from a company-owned workstation in a secure office" might be associated with a different access policy than "via an unsecured Internet connection."

Allowed Use

Examples of possible uses for a resource include *read, modify, change ownership, modify access control policy, create, delete,* etc.

Parties

Just as an access control policy identifies the resources to which it applies, it also identifies the *parties* for whom access is authorized. Parties can be referred to either by *identifier* (*e.g.* user # 6718) or by *attribute* (*e.g.*, all users in group "Administrators"*).* A party may be a human user, or it may be an abstract entity (*e.g.,* the mail delivery process for mail addressed to addresses in some-corp.com)

Confidence in Authenticity

An access control policy may have different rules for access depending upon the system's confidence in the *authenticity* of a party. For example, *this request came from a workstation where Joe logged in 6 hours ago using a password and has not yet logged out* somewhat weakly authenticates the request as coming from Joe. A higher confidence authentication might be: *this request includes cryptographic evidence, less than 2 minutes old, that the requestor possessed a security token registered to Joe at the time the request was made.*

An Example Access Control Policy

As an example of an access control policy taken from a familiar context, consider the policy governing a file located on someone's PC:

Authority	Owner of the computer
Resource	File stored on directly-attached disk drive
Classification	Non-sensitive
Access Context	Locally logged-in user
Allowed use	Owner: read/write; Group: read; Others: no access. No execute

Authorized parties	File owner. Members of group “admin” Not members of group “guests” or “other”
Authenticity confidence:	User has presented password (single-factor)

Figure 1—An example of an access policy; this one governs files on a PC

This simple example helps illustrate some important points. First, establishing *who* has *authority* over the resource is vital to any interpretation of the policy. In any sufficiently complex environment, there are likely to be many different authorities, each with control over a distinct set of resources. If the owner of the computer is an employer, the employee who uses the computer and its file storage will only have whatever authority is delegated to them by their employer. It is also necessary to be clear about the *specific resource* covered by the policy—*i.e.,* a file is a resource distinct from the disk drive it happens to reside on, and the file may well exist in multiple locations, but still be governed by the same access policies.[1] While this example classifies the file resource as non-sensitive—perhaps because it is just a file containing user preferences—it would clearly matter if the file was instead a sensitive document for which there might be severe consequences if disclosed to unauthorized parties. The *means of access* is an often-overlooked factor, but it clearly matters whether the file is being accessed locally, or is being “shared” across a network, or retrieved from a backup archive. In practice, *allowed use* is often specified according to the designation of *authorized parties*—in this example, the file owner and group have different usage constraints while everyone else should have no access. Finally, the degree of *confidence* in the *authenticity of parties* is vital to any assessment of compliance with an access policy or characterization of risk. The level of confidence required in an authentication process generally relates directly to both the *means* of access and the *classification* of the resource. For example, it might be reasonable to require a higher level of authentication confidence from remote parties than from a locally logged in user. Similarly, greater authentication confidence might be required if the resource is classified as “top secret” as opposed to “restricted.” Or a bank might be willing to process low-value transactions based on telephone authorization, but require a witnessed, guaranteed signature for high-value transactions.

Access control policies can be quite complex and often specify many conditions that must be met before access is either granted or denied. Furthermore, there can be significant risks from *both* false positives and false negatives in any system that provides controlled access to resources. For example, an access control system that falsely denies a financial trader access to their trade execution system could have substantial adverse cost implications. In reality, access control systems can be attacked in order to gain unauthorized access, or as a means of denying access to authorized parties (*e.g.,* a denial-of-service attack).

[1] While this example illustrates access control to a file object, the file is just a container for information, and it may well be that the governing policies apply to the data contained in the file, and not the file object *per se.*

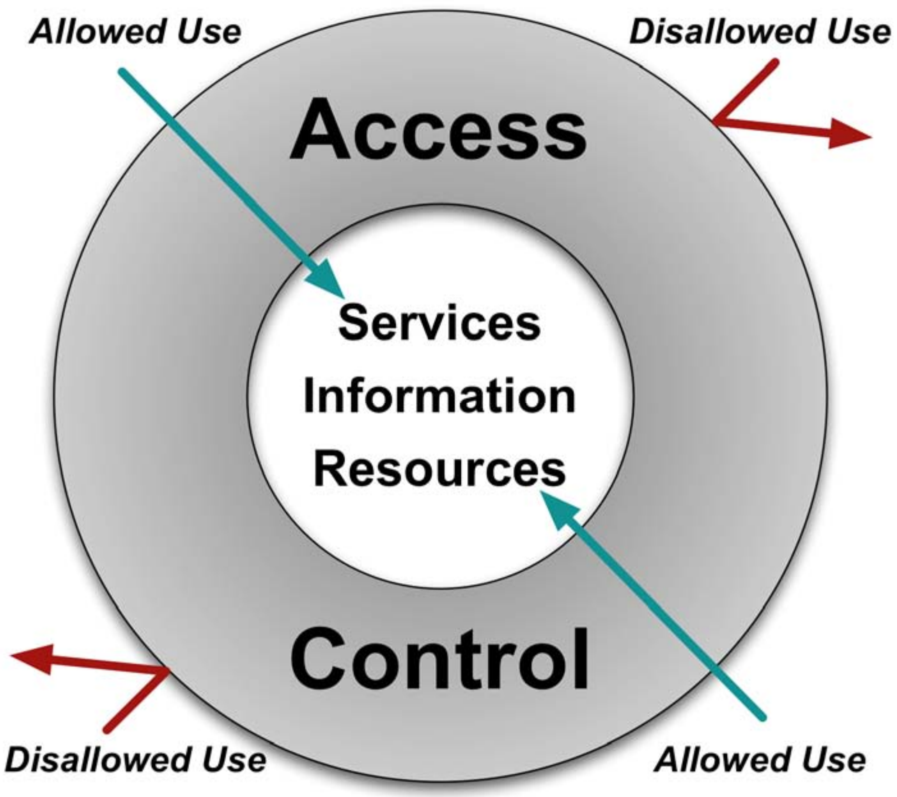


Figure 2—A simple conceptual model of access control

Access Control Mechanisms

Effective access controls depend on other essential *security services* and associated management facilities. Real world solutions are generally built on top of system platforms that provide these services and allow them to be combined in a building block fashion, though usually in an iterative manner. This allows access control solutions to be tailored to comply with access policies across a broad range of applications and usage scenarios. The security services essential to access control include:

- **Integrity:** information on which access control decisions are based must be of known integrity, and it must be possible to detect attempts to tamper with this information.
- **Confidentiality:** information intended for a specific recipient must be protected from eavesdropping. In particular, sensitive information used in access control (*e.g.,* passwords, session keys) must be protected from disclosure.
- **Authentication:** before making an access control decision, a system must determine to an acceptable level of confidence that the party requesting access is **authentic** (*e.g.,* is the legitimate user of the name under which access is requested.)
- **Authorization:** once a party has been authenticated, it is necessary to determine whether or not the party is authorized to gain the requested access.

These essential services are supported by additional system services, including:

- **Directory services:** typically, information about resources, policies, and various attributes of named parties is stored in databases; it must be possible to retrieve up-to-date information from these databases to support authentication and authorization decisions, with the added requirements that this information needs to be replicated and synchronized throughout a distributed system environment, and that the databases themselves must be protected.
- **Management services:** it must be possible to monitor operations, detect faults, configure system components, initialize and update information, and audit behavior

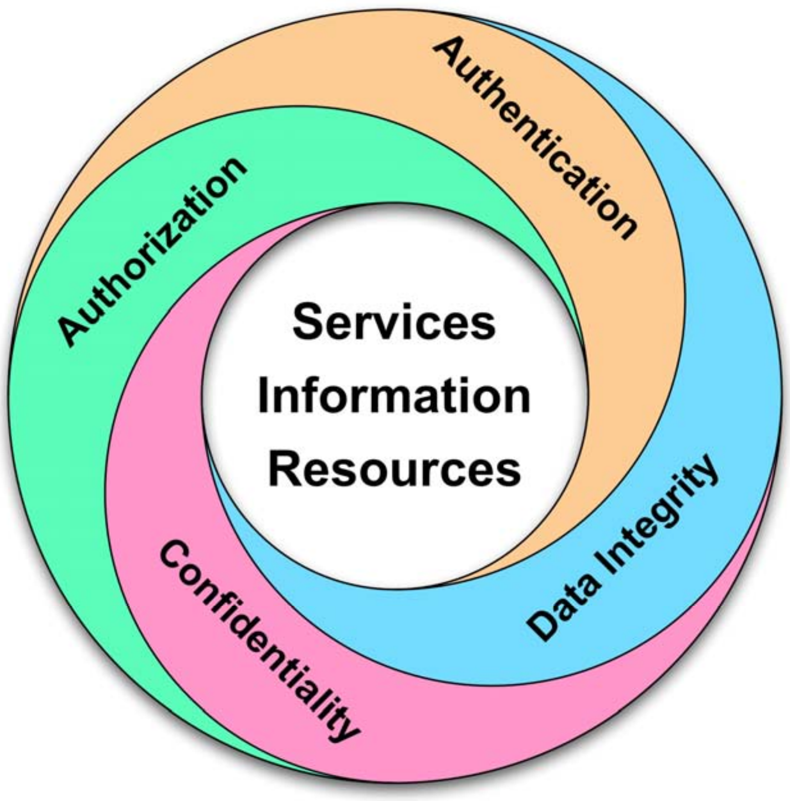


Figure 3—Access control depends on other essential security services, but generally in an iterative manner that interweaves security services in complex patterns. Access control is itself often applied in an iterative manner.

While there are many system components that can be used to provide these core services, two standards-based technologies have emerged as preferred building block services: Kerberos security services and LDAP-based directories. While Kerberos is strongly associated with authentication and directory services are perceived of as managing authorization and policy data, the distinctions are somewhat blurred since Kerberos also provides rudimentary authorization facilities, and directory services can provide some authentication mechanisms. In practice, Kerberos and directory services are complementary in nature, and most system platforms that provide access control facilities integrate these services to at least some degree (*e.g.*, Microsoft's Active Directory incorporates Kerberos and LDAP, tightly integrated with each other).

While Kerberos is often considered an *authentication service,* it also provides facilities for managing keys, assuring data integrity, authorizing use of services, establishing secure sessions, and protecting confidentiality of data exchanges. As an authentication service, it provides additional value by supporting *mutual* authentication of parties that enter into a secure communications session.

Issues for Security Services

The roles that Kerberos and directories play in providing core security services will be evaluated in greater depth below, but before doing so, there are other issues associated with the core services that should be reviewed:

- **Mutuality:** it is not enough that the service authenticate the user prior to granting access; it is also important that the user authenticate the service, so as to avoid providing sensitive data to a malicious impostor. As the plague of phishing attacks in recent years has demonstrated, one-way authentication exposes legitimate parties to a variety of attacks, even when one of the parties is not a participant in the fraudulent transaction—*e.g.,* a bank's reputation can be damaged when impostors purloin information from the bank's customers. It is now widely recognized that *mutual authentication* is essential to preventing such attacks. More generally, *mutuality* extends to access control, though usually in an asymmetric manner. In other words, both the service and client need to enforce access controls on the other, but for different reasons and to address different risks. For example, a web site that controls access so that only authorized users can access information or services, is at the same time accessing the users' browsers, and may even be executing code on users' systems in the form of JavaScripts or ActiveX controls.[2] The use of browser-side security zones and script-blocking plug-ins to constrain which sites are allowed to execute scripts illustrates one way that access controls have been introduced on the client side to minimize certain threats to users—*i.e.,* some browser users are now maintaining access control lists (ACLs) for sites they visit.
- **Principals:** In any application context, there are multiple *principals* involved in communications and performing various services. All of these principals need to be named, and appropriately authenticated. For example, a human user is an important principal in many applications, but the user is not the same principal as their workstation, nor is the user equivalent to the client software running on their workstation. Similarly, the client software and workstation are distinctly different principals, even though these principals are often treated as the same thing. In performing access control, it often matters which of these principals has been authenticated—*e.g.,* is it the user that was authenticated, their workstation, or their client software? Furthermore, it is important to clarify which principals have authority.
- **Delegation:** In complex distributed systems, there are many situations where it may be necessary for authorizations to be delegated to some system or agent to act on behalf of a primary authority. For example, a web front-end application

[2] While the wisdom of having servers exposed to the public Internet also downloading code onto a user's machine for execution without the knowledge of the user is dubious at best, the practice is wide-spread, and illustrates how other priorities can introduce new security challenges.

may need to retrieve information from a back-end database on behalf of a requesting user. Users also delegate authorizations to other individuals by simply sharing passwords or other authentication methods. Users even delegate authorizations to software agents to manage additional delegation by giving these agents userids and passwords to provide to other systems Unfortunately, many policy regimes do not adequately address delegation within stated policies, or take the naïve position that delegation is not allowed. In general, it is best to define rational policies for appropriate delegation of authority, and provide controls for assuring that delegation is supported in compliant ways.

- **Network Infrastructure:** Today's network infrastructure does not provide security services, relying on other parts of the system to provide these often vital services. Consequently, every aspect of the network—even a "private" network—must be considered untrusted. This includes the addresses claimed by network nodes, the DNS system that maps names to addresses, DHCP services that assign addresses to devices, network time services that are used to synchronize clocks, service discovery protocols, and each and every packet sent or received. In particular, "Man-in-the-Middle" (MitM) attacks are feasible, and increasingly common. The untrusted nature of networks substantially complicates access control, and requires careful planning and deployment of robust security infrastructure to achieve policy compliance. Of course, there are many examples where people choose to trust systems reached over untrusted networks where the risks are acceptable, such as using the public Internet for access to news and other content. There can also be tightly controlled subnets that are trustable, even for high-risk transactions. In the end, it's all about acceptable risks.
- **Registration:** Procedures are needed to register or enroll users, systems and resources within an access control context, yet problems or vulnerabilities in these procedures have the potential to completely undermine an entire access control system. For example, if an impostor is able to enroll as an authorized user in a system by exploiting some vulnerability in the enrollment procedure, then it does not matter how strong or robust the access control system is, since the impostor will be granted access. Note that "privilege escalation" is another potential vulnerability in registration/enrollment procedures if a legitimate party is able to acquire more authorizations than allowed by policies. Furthermore, registration problems exist for both human users and *machines.* As the varieties and capabilities of network-attached machines (*e.g.,* servers, application services, printers, network appliances) continue to evolve, registration problems with machinery result in new or increased threats.
- **Revocations and Authorization Updates:** Procedures must also be in place to modify the authorizations granted to parties. In particular, it is usually necessary to be able to revoke some or all of a party's authorizations if the party is found in violation of policies, or if authentication credentials are compromised. Conversely, a party may also be granted new authorizations that allow upgraded access to resources. Another common scenario is that a party's role changes, which necessitates adding some new authorizations and simultaneously rescinding other authorizations. An important consideration with these procedures is the ability to propagate such changes throughout an entire access control environment. For example, if a user's access privileges must be revoked, it could represent a significant exposure (*i.e.,* risk) if the revocation will not be

recognized until some time later. Similarly, if a user is granted new privileges, it could be counter productive if they encounter a prolonged delay before they can make use of their new privileges.

- **Key Management:** The secrets used to authenticate parties—including passwords— must be protected from disclosure and will need to be replaced periodically, including whenever there is any potential that such secrets have been exposed or compromised. The procedures for handling replacement of secret passwords and keys represent potential exposures that could lead to substantial risks if exploited. Secrets also need to be protected from inadvertent disclosures, as they represent attractive targets. Passwords used by people have long been recognized as fundamentally weak—a risk that has only grown in recent years as many more attacks have been developed to fool users into disclosing their passwords. However, secret keys used by applications and servers are also being targeted by adversaries using highly sophisticated attacks. The reality is that today's access control systems are only as strong as the management procedures used to maintain secret keys and passwords.
- **Liveness:** It is common for policies to mandate that access be provided only to authorized *human* users, and not to machines or software agents. However, it is quite difficult for an automated access control system to determine if a remote party is truly a live human being (the "reverse Turing test"). While various hardware and software techniques have been devised to provide *liveness* tests, the efficacy of many tests is being undermined by "smarter" machines. In general, liveness testing now requires some sort of hardware device—other than a keyboard on a computer—that requires human activation or use. For example, a one-time password dongle that requires the user to transcribe a code displayed on the dongle into a data entry field on their computer screen provides reasonable confidence that a live human is in the loop.
- **Scalability:** As information services continue to scale up in terms of information content scope, geographical distribution, and number of users, modern access control systems face escalating challenges of scale. Unfortunately, the multi-dimensional nature of access control systems usually leads to at least polynomial scaling. For example, an organization that has linear growth in information being accessed and also in its user population is likely to find access control challenges increasing in proportion to the product of these two linear growth trends. Not only does scale challenge access control systems in the familiar areas of performance and maintainability, scale also increases the "attack surface" in ways that can be difficult to fully assess and adequately address.
- **Federated Environments:** Many organizations today are far-flung enterprises that work closely with business partners, outsourced service providers, and even customers. Therefore, access control often extends beyond traditional organizational boundaries (and, accordingly, across policy regimes), and new *federated* models have emerged for extending access authorizations across such boundaries. Consequently, access control systems have had to evolve to support federated interactions as well as access from non-traditional entities. Federation also leads to new policies and new challenges in assuring compliance with both new and existing policies.

- **Auditability:** Access controls for modern information systems generate potentially significant events at enormous rates. Events may relate to access granted or denied, authentication of parties, authorizations issued, changes to directories, replications, changes in secrets or keys, revocations, management operations, configuration updates, and myriad other state transitions. Even assuming that all of these events are logged, effective audit practices require the ability to analyze logs for patterns of aberrant behavior or trends that portend potential problems. Furthermore, the ability to prevent parties from repudiating events that are recorded in logs may be necessary to insure credibility of log records. In some cases, audit requirements might extend to *accounting for access* in support of various business models.
- **User Convenience:** Security services must function without imposing undue burden on end users. Users who are required to establish and memorize large numbers of different passwords for different services, who are required to re-authenticate repeatedly during the course of a session, or who are wrongly denied access, will regard security as excessive, intrusive, and counter-productive, and will, in some cases, find ways to work around security protections that often leave the system worse off than it would have been without the security measures. For one example, users who are required to use complex, difficult-to-memorize passwords often respond by posting their passwords on their office walls next to their workstations.

Kerberos, especially when integrated with robust, enterprise-level directory services, provides a solid foundation for addressing many of the issues summarized above. However, the effectiveness of any security tool set is directly related to the practices employed in managing the overall information systems environment. Furthermore, no one tool set is sufficient in today's threat context, and appropriate combinations will be needed to achieve adequate levels of security.

Naming	Every user, host, and service (collectively, every *principal* has a unique name.
Authentication	It is possible to determine that the party using a name is its legitimate owner
Directory Access	It is possible to look up *attributes* for any name, e.g., group membership, entitlements.
Authorization	It is possible to look up the applicable *access control policy* and determine whether or not the proposed access is *authorized*.
Confidentiality	Eavesdroppers cannot steal the content of messages.
Integrity	The message that a party receives is the message that the other party sent, has not been corrupted (deliberately or accidentally) en route.

Management	It must be possible to add and remove principals, change attribute data, edit access control policies, change secrets (passwords) and otherwise manage the data on which the system depends.
Audit	The system must record each of its actions.

Single Sign-On (SSO)

The ability for users to sign on (login or authenticate) once per session, and get access to all the information resources they need without subsequently being bothered to authenticate again, is an important user convenience and a much sought-after capability. Kerberos has, from its inception, provided a Single Sign-On (SSO) facility for users, which it implements by caching time-stamped, limited-use credentials and presenting them on behalf of the user when the user requests access to resources and services.

Single Sign-On does not change any aspect of the overall access control framework. Each time the system grants access to a service, it conducts an authentication step, but with SSO the authentication step is based on retrieving cached credentials rather than upon asking the user to enter a password again. Software clients and services using Kerberos always authenticate with each new interaction, and may use session keys to authenticate each message they exchange. Without such procedures, access controls would not be effective, and SSO would be of little utility.

What Kerberos offers to users is the ability to prove their authenticity once in a period, and to cache the fact that authentication was successful so that subsequent authentications conducted by client software on behalf of the user need not involve the user. However, access control policies may impose other requirements that could cause users to have to re-authenticate, or provide additional proofs of authenticity. For example, if a user originally signed in using a password, and is now attempting to access a resource whose policy requires a higher degree of confidence (for example, use of a smart card or other physical token), the user will need to authenticate again. Or, an access policy for some services might allow authentication via cached credentials up to eight hours old, while another might require that the user has authenticated within the past 2 minutes, or has specifically authenticated the current request.

There are other real-world policy constraints on the scope of SSO. For example, a user that authenticated under one policy regime may not be recognized in another policy regime unless a trust relationship has been established. Since access controls also depend on authorization, a user signed on in one policy regime might not be able to provide acceptable authorizations in another policy regime, even if the two regimes agree on authentication of users.

Consequently, while it is reasonable that users should only need to sign on once, it is often impractical to offer universal access to services throughout an extended enterprise, and generally not across major policy boundaries, such as between enterprises or across national political borders (though national borders do sometimes get blurred in large, multi-national enterprises).

The Role of Kerberos

Kerberos is a foundational service on which applications and other security services can be built. At its core, it provides the means to authenticate clients to servers, and vice versa. Furthermore, it also provides an array of other services that have proven both useful and vital to achieving security objectives within today's information systems. The capabilities of Kerberos will be explored below with the objective of improving understanding of the role that Kerberos plays, and how its role can be further leveraged. Note that the reader is assumed to have a basic understanding of how Kerberos protocols work and the means by which clients and servers interact with KDCs and associated Authentication Services and Ticket Granting Services. Tutorials on how Kerberos works are available from the MIT Kerberos Consortium.

- *Kerberos Protocol Tutorial (html)*
- *The MIT Kerberos Administrator's How-to Guide (pdf)* (Chapter 1 is a brief tutorial)

Naming in the Kerberos Context

A common terminology problem in security discussions is confusion about the meaning of *names*, and how to relate names to the things being named. In particular, there is an unfortunate tendency to confuse *identifiers* and *identities.*

- Strictly speaking, an **identity** is an abstraction—that which distinguishes an entity from all other entities in the universe. In other words, the essence or uniqueness of an entity. An identity is not something that can be pointed to, written down, or stored in a database.
- An **identifier** on the other hand, is that which refers to an identity. Identifiers are merely "handles" that can be used to conveniently reference some entity, often in an indirect manner. A name such as "John" or "the billing department of Some-Corp." is one type of an identifier. Other identifiers can be numeric ("employee 237") or attribute-based ("the user currently logged in at workstation F3B288"). Relevant types of identifiers include userids, Kerberos principals, and X.500 Distinguished Names.
- Identifiers may be unique but need not be. "John" refers to many people. Some identifiers are deliberately constructed so as to be unique, either globally or within a specific context. For example, an IP address is conceptually a unique identifier, but RFC 1918 IP addresses (so called "private" IP addresses) are only unique within the context of a specific physical network. Furthermore, there is nothing that guarantees uniqueness of IP addresses, either locally or globally, and specific hosts can have dynamically assigned IP addresses that change over time. In general, uniqueness of names requires the existence of some registration process to guarantee that naming collisions do not occur. For example, domain names are usually globally unique due to the registration process administered by ICANN. Another technique used to approximate uniqueness without requiring a registrar is to concatenate multiple names used to reference an entity. For example, a person's full name combined with their birth date and city of birth is usually unique. DNS names and names used by Kerberos are also concatenated sequences of simpler names.

- The same entity may be referenced by many names or *aliases.* For example, a person is known by their given name, surname, birth date, residential address, telephone number(s), social security number, passport number, drivers' license number, and numerous account numbers and userids. Aliases are commonly used for entities authenticated by Kerberos.
- The same name may refer to different entities in different contexts. For example, many programs running as part of the operating system are loosely referred to as simply, "root"—an example of a name that has meaning only within a specific context, and is more accurately the name of a *role.* "Root" may refer to different entities on different machines within the same environment. Virtualization is a recent technical trend that is greatly complicating the naming of computer systems, as well as the attributes associated with computers, since a single computer system can be simultaneously running multiple OSs, each with its own naming conventions.
- Some identifiers are associated with attributes, or are encoded to indicate attributes. For example, model numbers and some serial numbers indicate attributes of the associated equipment, and might even relate to things like date of manufacture. Other names are addresses that can be used to locate an entity in some space, or to establish communications with the entity.

Fortunately, Kerberos is fairly consistent in what it names and how names are structured, though there are differences in naming conventions between current versions of Kerberos and with earlier versions. There are also some differences in naming conventions used in Microsoft environments versus other environments.

The entities named by Kerberos are termed *principals* and *realms.* A principal refers to an entity such as a user, machine, or service; a realm refers to a policy regime: an organization or specific network.

Principals in Kerberos can be either users or software applications (*a.k.a.,* services, servers, computers, applications, nodes, PCs). User principals, or often just "principals," are normally associated with persons, but might be associated with roles fulfilled by persons, or even agents acting on behalf of real persons in some cases. Host or service principals can represent a single physical computer, or a service running on one or more computers. A single physical computer might host multiple services, each given its own Kerberos principal name. It is also possible for a single named service principal to be hosted across multiple physical computers. A principal name includes a text string such as *joe*, *system-administrator*, or *mail-transport-daemon* and the *realm name* in which the principal is defined.

The most common form of a realm name is derived directly from a DNS domain name such as *accounting.some-corp.com*. DNS domain names are globally unique to the extent that they are registered through official domain name registrars.[3] By convention, Kerberos realm names are usually presented in upper case: the realm associated with *accounting.some-corp.com* would be *ACCOUNTING.SOME-CORP.COM.* Note, though, that unlike DNS domain names, Kerberos realm names are, in reality, *case sensitive.* This provides an opportunity for misconfiguration: if a realm is referred to in one place as

[3] Actually, domain name registration processes do not guarantee uniqueness of domain names; they just make duplicates less likely, and more easily detectable.

ACCOUNTING.SOME-CORP.COM and in another as *Accounting.Some-Corp.com* the two will not match.

A complete Kerberos principal name takes the form of the principal name, followed by an '@' character, followed by the realm name, e.g. '*joe@SALES.SOME-CORP.COM*'. Although this looks syntactically like an e-mail address, it has nothing to do with the delivery of e-mail. Simple principal names (conceptually similar to what Microsoft refers to as User Principal Names or UPNs) comprise a *userid* or *account name* that is unique within the realm. Another form of user principal comprises a userid concatenated with a role name (*e.g.,* "admin").[4] This convention allows a single user to have multiple principal names that they can use depending on what role they are performing.[5] Principal names for services (referred to as Service Principal Names, or SPNs, in Microsoft terminology) always concatenate one or more components; typically, one of these components is the Fully-Qualified Domain Name (FQDN) of the host. This allows multiple services to be uniquely referenced, even though they are hosted on a single computer with one FQDN.[6] In most situations, additional conventions are employed to name services in a consistent manner throughout an enterprise or administrative realm.

The importance of naming conventions cannot be overstated, as nearly everything else in the overall security context depends on the ability to reference entities by consistent names, and to further be able to associate attributes with these names. Naming issues are of particular concern in mixed environments where differences in naming conventions can lead to serious interoperability problems. Some points worth noting about Kerberos and its naming conventions are summarized below:

- Kerberos KDCs also serve as registrars that guarantee uniqueness of principal names within a given realm. Note that component names used to construct a principal are not required to be unique within a realm, but any given sequence of component names used to construct a principal name must be unique within the realm. Failure to assure uniqueness of principal names can lead to operational problems, and is one of the sources of problems in mixed environments.
- If a realm name is the same as a publicly registered DNS name, then the realm name can be assumed globally unique, and by inference, all Kerberos principal names incorporating that realm name will be globally unique as well.
- Kerberos has evolved with certain dependencies on DNS services to be able to confirm FQDNs or perform reverse lookups on IP addresses to match requests against FQDNs. While reliance on DNS is useful in an administrative context, the inherent insecurity of DNS means that DNS lookups should not be considered *authoritative* to any significant degree.

[4] Concatenated components of principal names use forward slashes ('/') as separators between component names in the current Kerberos version 5.

[5] An example of a principal name for the person Robert Smith, an employee of Galactic Industries Inc. might be rsmith@GII.COM, and this same person might have another principal name of rsmith/admin@GII.COM for when they are acting as a Kerberos administrator.

[6] An example of principal names for email and file services on a host named Atlas at Galactic Industries Inc. might be smtp/atlas.gii.com@GII.COM and host/atlas.gii.com@GII.COM.

- The syntactical and semantic simplicity of principal names in Kerberos is an advantage. This makes it possible for all other systems to easily recognize and parse Kerberos principal names.
- While it is theoretically possible to encode various attributes about principals by concatenating component names that represent attributes, doing so is overloading the semantics of Kerberos names, and is not considered a good practice. Principal names should only comprise as many components as necessary to assure uniqueness. In particular, it would be a mistake to use component names as a means for providing authorization information, primarily because naming is not sufficiently flexible to accommodate authorization attributes, which may need to change on a more frequent basis than names.
- Kerberos KDC databases comprise simple directories that map principal names to authentication information (*e.g.,* shared secrets). However, the need to keep KDC operations efficient and secure has been a strong argument for keeping the KDC database simple.

Kerberos Authentication

Authentication is essentially a process that determines the *authenticity of a claim to some degree of confidence.* Usually, claims are expressed as, *"I am the party known by this name."*[7] In order to confirm the authenticity of a claim, an authentication system will conduct one or more tests involving the claimant or other context associated with their claim. For example, a classic test of a claim involves the challenge, *"if you're the party that goes by that name, then prove it by presenting the password we previously shared."* However, many other tests can be used, including other so-called authentication factors, such as proof of possession of something (*e.g.,* a security token) or demonstration of unique attributes (*e.g.,* biometric parameters). Contextual clues can also be used that might not involve presenting a challenge to the claimant. Examples of contextual clues include things like IP or MAC addresses, reverse DNS lookups, or past patterns of behavior.

Effective authentication involves both positive and negative tests. Challenge-response interactions are examples of positive tests since, if the response is correct, then there is positive information supporting the claim. An example of a negative test might involve checking the IP address used by the claimant. If the IP address is part of the subnet associated with the claimant, then there is no real positive confirmation, since any member of the subnet could easily spoof any IP address in that subnet. However, if the IP address is from a subnet where the claimant should not be located, then evidence is at hand indicating the claim is likely to be false.

Traditionally, Kerberos utilizes pre-shared secrets to authenticate parties. In the case of user principals, the pre-shared secret is derived from a user's password. Service or host principals are authenticated through pre-shared encryption keys. Unlike other

[7] Note that *authentication* is not generally *identification.* Only when the claim is that a party is a *specific* individual or thing does it become reasonable to treat authentication as identification. Normally, identification is hard for computers (and for people where computers are involved). Consequently, identification is only employed when a party initially registers, and subsequently only simpler authentication procedures are used for efficiency's sake.

authentication schemes involving pre-shared secrets, Kerberos only requires principals to disclose their secrets once at the time of initial registration. Subsequently, authentication can take place without exchanging the pre-shared secret.

In the case of user passwords, some Kerberos implementations only know a key derived from the password, and do not actually know the user's password. Furthermore, passwords are *not* exchanged between parties during authentication. This substantially reduces risks that the authentication secrets will be disclosed during normal authentication operations. However, if these pre-shared secrets are used by non-Kerberos authentication mechanisms, then the risk of exposure might be much greater. For example, there are systems where a user's password might be used in another method of authentication that involves passing the password between the user and the authenticating party with every authentication operation. Clearly, best practices avoid situations where a user's Kerberos password is also used by less secure authentication methods.

Modern Kerberos systems are also able to utilize other authentication tests in addition to, or as alternatives to, passwords and shared secrets. One option is to use public key cryptographic techniques in conjunction with digital certificates to avoid any necessity to pre-share secret keys or passwords. In this context, only public keys are disclosed, and their disclosure to other parties does not represent a risk unless the corresponding private key is somehow disclosed. Private keys can be protected by using various hardware schemes (*e.g.,* smart cards), including some hardware devices that are now commonly built into computers (*e.g.,* TPM chips). Other authentication options that can be used with Kerberos include One-Time Password (OTP) dongles, scratch cards, and biometric scanners. An important implication is that a system built using Kerberos can easily add new authentication methods without having to re-engineer the system.

Mutual Authentication

One of the greatest benefits to using Kerberos as an authentication service is that it provides the means for both parties in an exchange to authenticate each other—*i.e., mutual* authentication. Mutual authentication is a readily available facility to any application that uses Kerberos. Experience indicates that mutual authentication should always be used, and is an important (necessary) best practice.

It has long been recognized that mutual authentication is necessary for any secure dialog. However, the asymmetric nature of communications between computers and persons has been frequently used as an excuse for allowing one-way authentication. If there is one lesson learned from attempts to secure systems it is that *any vulnerability allowed to exist will be exploited.* This has certainly been shown to be the case for the vulnerabilities associated with one-way authentication, and the surge of phishing attacks in recent years is just one illustration of this reality.

The Kerberos protocols result in tickets being issued to the party requesting access to a service that include both authentication information the service can use to authenticate the requesting party, as well as a session key that can be used in responding to the requesting party. If the requesting party receives back from the service a response based on the session key, then they have confirmation of the authenticity of the service. This is due to the way that Kerberos tickets are formed by the KDC where the session key and user authentication information are encrypted with the service provider's key. Therefore, when the service returns a response to the requesting party that uses the session key, the requesting party knows that only a service that possesses the secret key shared with the KDC could have correctly responded—hence, *mutual authentication.* A further benefit is that, by completing this exchange, both parties possess a new shared session key known only to them that can be used to authenticate every message passed between them, and even to establish an encrypted session.[8]

Source Authentication

So far, only one type of authentication has been discussed—the authentication of *peer entities.* Another type is known as *data origin* or *source* authentication, and involves claims of the sort: *"this data is provided by a specific named party."* Kerberos also provides support for this second type of authentication.

One means offered by Kerberos for authenticating the source of information is provided through exchange of a secret session key between a client and service. If the session key is used to encrypt information exchanged between the parties, or in producing keyed hashes of the information, then either party knows that only a party in possession of the session key could have provided the information. In essence, every message can be source authenticated through use of the session key. Again, best practices call for both parties to utilize the session key to authenticate subsequent exchanges of information.

If services are delivered in a distributed manner, a single client may be communicating with multiple service providers at the same time. Kerberos supports multiple mechanisms for clients to establish distinct session keys with each service provider, which allows the source or origin of data to be authenticated in each case.

Source authentication is also provided for ticket responses issued by a Kerberos KDC, since the requesting party (client) can determine that the response was encrypted with a key known only by the KDC and client, and similarly, the service provider can determine that the *ticket* was encrypted with its secret key that was previously shared with the KDC. This form of source authentication is an inherent feature of the Kerberos protocols. It also means that other information in the ticket encrypted with the pre-shared secret is known to have originated from the KDC. This allows tickets to be used as *authorizations,* and they can be used to carry additional authorization information (see Kerberos support for Authorization below)

Data Integrity and Confidentiality

Kerberos also provides the means for determining that data contained in tickets has not been tampered with or altered since the ticket was generated—*i.e.,* data integrity is assured. This is essential to prevent third parties from manipulating tickets as a means

[8] The original session key is also known to the KDC. However, applications typically establish a new session key that the KDC could only learn by observing the exchange between parties when mutual authentication is used.

of undermining authentication procedures, or changing authorization information. In particular, a client requesting a ticket to access a service cannot manipulate the portion of the ticket intended for the service provider to authenticate the client, since this portion of the ticket is encrypted with the secret key known only to the service provider and the KDC.

Assuming that the client and service complete the exchange of a session key, then this session key can be used to provide data integrity for subsequent exchanges between the parties. Data integrity can be provided by computing a hash of the data that is keyed using the session key (*e.g.,* use of an HMAC function). This is the same mechanism used to assure source authentication. If the session key is instead used to encrypt data sent between the client and service, then confidentiality, data integrity, and source authentication are all provided as a by product of encrypting with a secret known only to the two parties.

Replay Prevention in Kerberos

Since Kerberos tickets are exchanged over untrusted networks, attackers must be prevented from replaying tickets to interpose themselves as apparently authenticated parties. Originally, Kerberos tickets included the IP addresses of the requesting client and the service, which provides some limited protection against replay attacks. However, modern Kerberos implementations employ timestamps to limit the validity interval of tickets to only a few minutes, which helps prevent replay of tickets except during this narrow time window. This assumes that all participants in Kerberos protocol exchanges have some means of maintaining time synchronization within at least a few minutes.[9] In addition, services are supposed to maintain a cache of tickets seen within the window of time that tickets can be valid. This allows a service to detect any replay attempts within this time window. Use of client-specific session keys is another effective technique for preventing replays. Best practices require all services that rely on Kerberos to implement counter measures against replay attacks.

Non-Repudiation Support in Kerberos

The exchange of session keys and timestamps in Kerberos tickets that are generated by the third party KDC also provides basic non-repudiation protections. Repudiation is defined as one party denying participation in all or part of a communications dialog with another party.[10] In essence, the KDC serves as a third party witness to the establishment of the session between client and service. However, this only protects services from attempts by clients to repudiate that they requested access to the service. Clients cannot rely on the KDC to vouch for whether or not a service ever responded to

[9] Use of NTP services is a common means for achieving time synchronization, but care must be taken to prevent attacks that force a system to update its clock to an earlier time when replayed tickets would still be valid. In practice, systems that rely on NTP should avoid resetting their local clocks backwards in time to synchronize with an NTP server. Use of NTP security options is also a recommended practice.

[10] Repudiation might consist of claims by clients such as, *"I did not attempt to access that service,"* or *"I did not attempt to access that service at that time."*

a client request. Furthermore, this limited form of non-repudiation is not equivalent to the protections offered by electronic notary services that are designed to deal with complex repudiation scenarios.

The chief benefit of this limited form of non-repudiation protection is that it can enhance the credibility of system logs that indicate which clients accessed which services and approximately when the access attempts were made. This improves the integrity of audit procedures and the strength of audit controls. However, for this to be effective protection, both the KDC and service need to keep log records of every request from a client, what service the client requested access to, when the request was made, and the actual ticket issued. Since the ticket is encrypted with the key shared by the KDC and service, neither could change the ticket without causing a discrepancy in the log records kept by each. The KDC will also need to keep log records of requests from clients for tickets as encrypted in the TGT session key, as well as log records for initial authentications and TGT tickets issued. It is standard practice for KDCs to log this information.

If a client later tries to repudiate that they requested access to a specific service, then the KDC has proof that the client made a request for the ticket to that service. If the service gets the ticket, it could have only come from the client because the client's authenticator will have been encrypted in the session key established by the KDC for the client and service to use. If the service is able to demonstrate that its log record of the ticket matches what the KDC logged for the ticket, then the service did not generate the ticket on its own. Furthermore, the client could only have made the request for access to the service after the time it requested the ticket from the KDC, which partially prevents repudiation claims as to when the client made the request.

Kerberos support for Authorization

Although Kerberos is primarily an authentication service, it does provide basic authorization services for principals registered within the KDC database. Unless some *authority* (usually a human being) has registered a principal by entering their principal name into the KDC database through some administrative procedure, the principal will not be known to the KDC, and will not be able to use the services of the KDC to authenticate to other principals. Consequently, only *authorized principals* are included in the KDC database, and presumably, authorities with administrative access can also remove principals from the database to de-authorize them.

Furthermore, Kerberos supports rudimentary authorization services when issuing *Ticket Granting Tickets (TGTs)* and for *session establishment* between clients and services—services that are provided only to *authorized principals.* Kerberos also provides facilities for securely passing authorization information to services. These authorization services are described in the following subsections.

TGT Authorization

When clients initially authenticate with the Kerberos *Authentication Service* (normally, a subset of the KDC services), they receive back TGTs that are used as *authorizations* to conduct subsequent requests for tickets to access other services. This is the way that

Kerberos supports Single Sign-On (SSO) so that a client need only authenticate once, and can then subsequently access other services without having to present their password or other authentication information each time.

The TGT returned by the KDC to a client requesting initial authentication establishes a session (see Session Authorization below) between the client and the Ticket Granting Service (TGS)—also typically deployed as a subset of the KDC. This session is based on a new secret session key shared only between the client and the TGS (a.k.a., the KDC). The TGT is a time-limited authorization where normally the time limit is at least several hours, but rarely more than a day.

Mutual authentication between the client and KDC is typically established as part of obtaining the TGT.[11] This exchange assures both parties that the other party knows the previously shared secret (*e.g.,* the client password). Furthermore, by establishing a new secret session key between the client and KDC, the client need only expose its copy of the pre-shared secret just long enough to decrypt the TGT received back from the KDC as part of the initial authentication exchange. In addition, data integrity is assured since all subsequent requests from a client to the KDC for service tickets and responses from the KDC back to the client will be encrypted with the TGT session key. Replay protection is provided by requiring clients to provide time-stamped *authenticators* in each request for a new service ticket that make every such request unique.

When a client wants to access a service in another Kerberos realm (see Cross-Realm Authentication below), it obtains from the KDC in the client's realm a TGT it may use with the KDC in the services realm to then request a ticket for the service. This effectively extends authorization from the client's KDC to the remote KDC realm and allows the client to establish an authorized session with the remote KDC for requesting tickets. Note that this description of cross-realm access is simplistic, but the concepts hold in scenarios that are more complex.

Session Authorization

Kerberos also provides basic authorization services for establishing *communications sessions* between principals. In fact, Kerberos tickets are secured authorizations that a service provider uses to authorize establishment of a communications session with a client. The authorizing party is the Kerberos KDC that issues tickets to clients. However, the true authority is the party responsible for entering the named principal in the Kerberos KDC database of *authorized* users and services—*i.e.,* the *registration authority.* If a party does not have a principal name in the KDC database, that party is not authorized for *any service* that depends on Kerberos for authentication.

A common statement about Kerberos is that it only authenticates principals to each other, and that authorization is a decision made by the principals based on some application context. This is because Kerberos realms may have very liberal policies for

[11] The client always authenticates the KDC in the first exchange. The KDC can use a facility called pre-authentication to authenticate the client before issuing the ticket, or the KDC can wait until the client first uses the ticket to know that it was able to successfully decrypt the ticket. Current best practice is to use pre-authentication.

what principals are registered. For example, a realm may register any unused username based on a request at a webpage. Work is on-going to allow anonymous Kerberos authentication without registration in any KDC database. Similarly, some realms have very liberal policies about what services are registered. As a result, application designers cannot make any assumptions about authorizations implied by registration in a particular realm; potentially any entity that requested such registration may be able to obtain it in some realms.

On the other hand, setting policies for what principals are registered in a realm is a powerful access control tool for IT administrators. When a client presents a ticket to a service, the service is presented with information that not only authenticates the named principal, it also indicates that the client principal is registered in the KDC database. The service receiving a ticket from a requesting client must decide whether or not to establish communications with the client based on the ticket contents, and any other information the service may have at hand (*e.g.*, a list of principals authorized to use the service). However, this decision will be based in part on the ticket that is issued by the KDC, which is, in fact, an authorization issued by the KDC stating that the principal is an authorized party within the KDC database.

Section 1.4 of RFC 4120 advises that *"Applications should not accept the mere issuance of a service ticket by the Kerberos server (even by a modified Kerberos server) as granting authority to use the service."* This is sound advice, since *authorization to establish a session* is not the same thing as *authorization to use the service.* Only within a specific application and policy context can rational decisions be made regarding appropriate authorizations. At the same time, effective access control often depends on multiple levels of authorization, and session authorization is a useful first step in an access control strategy. In some specific cases, such as remote access to a network, it may even be sufficient. Application designers should provide tools that make it easy for IT administrators to take advantage of this level of authorization; for example, in some cases it may be desirable to grant access to all principals in a realm.

Authorization Information passed by Kerberos

Kerberos can also play a supporting role in access control decisions by passing *authorization information* securely to a service within a ticket used to authenticate a client requesting access to the service. The Kerberos protocol specifications provide for two classes of authorization information. Clients or KDCs can add restrictions to tickets restricting the context in which the ticket is valid or the access that should be granted to a party authenticating with this ticket. In addition, KDCs can add authorization information granting additional privileges or describing authorization attributes of the entity that the client principal names. Facilities are provided so that clients cannot create the second type of authorization information. While the Kerberos protocol specifications do not indicate how this authorization information is to be used, or even all the details of how the information is structured, this option can be used to extend the utility of Kerberos services in managing access control. Since the ticket is encrypted with the pre-shared secret key known only to the service and the KDC, the service is able to trust the information in this ticket, both to authenticate the client and the authorization granted to the client.

For example, when a client requests a service ticket, if the KDC includes a list of groups this client principal belongs to in the authorization data field of the ticket, then the service will be able determine what resources the client is allowed to access based on group memberships. This very scheme is widely deployed by Microsoft as part of its Windows Server platforms. Other examples of authorization data that could be passed within a ticket include time-of-day restrictions, subscription information, transaction limits, or usage quotas.

Microsoft's use of Kerberos as a means to pass group memberships and other account information in service tickets is the most widely deployed example of extending Kerberos in this manner. The authorization data is packaged into a structure known as a *Privilege Attribute Certificate*, or *PAC* (refer to the section Kerberos and the Microsoft Privilege Attribute Certificate (PAC) below). In a Microsoft server context, the KDC is a subset of a Domain Controller, and since Domain Controllers maintain accounts for all users and systems operating within the Domain, it is feasible for the PAC to be inserted in service tickets when clients request access to services.

One concern with trusting authorization data in tickets is that a service could generate tickets to itself. This is because the keys shared between services and KDCs are symmetric encryption keys, which allow either side to encrypt the data. This allows the service to trust a ticket received from a KDC, since only the KDC should know the secret key. However, a service could generate its own ticket using its key. While this does not undermine Kerberos authentication, the ability to generate new tickets could be used to elevate privileges if authorization information in the ticket is trusted by a party other than the service for whom the ticket is issued. To avoid this potential exposure, software such as the local operating system that wishes to use authorization information from a ticket issued to another service should confirm authorization information with a trusted source before making access control decisions based on this information. For example, Microsoft addresses this problem by having the "Local Security Authority" confirm PACs received by unprivileged services with the Domain Controller via an independent secure channel established using their Netlogon facility.

Integrating Kerberos into Applications

The process of integrating Kerberos into an application is colloquially referred to as *Kerberizing* an application. The companion document, *Best Practices for Integrating Kerberos into Your Application (pdf)*, provides a comprehensive guide to application integration issues. However, a brief review of integration options is provided here to help clarify the role that Kerberos plays in other system applications, especially directory services, which will be discussed in greater depth in The Role of Directory Services section below.

The GSS-API and Kerberos

The *Generic Security Services Application Program Interface*, or *GSS-API*, was developed as a way to decouple applications from underlying security services, especially authentication methods. Kerberos and GSS-API are closely related, even though the GSS-API (ref. RFC 2743) is designed to allow any authentication technology to be used by an application. It just happens that Kerberos is the dominant authentication technology used with GSS-API. Furthermore, the GSS-API has become the standardized API for Kerberos (ref. RFC 4121), since the Kerberos protocol specifications do not define any APIs, and various Kerberos implementations utilize incompatible program interfaces.

The GSS-API, when used with Kerberos, actually defines "wire-level" protocols that are exchanged between parties using GSS-API so that data confidentiality and integrity assurance can be provided in addition to authentication. In this sense, it is more than just an API. An important implication is that applications could be built that are compatible with the wire-level GSS-API Kerberos protocols, but that do not actually use the application programming interface defined by the GSS-API—*e.g.*, Microsoft's *Security Support Provider Interface,* or *SSPI.*[12] Even though the programming interfaces provided by the GSS-API and Microsoft's SSPI are quite different, it is still relatively straight-forward to build applications using one API that will interoperate with applications using the other API. However, due to fundamental differences in approach, it is also possible to build GSS-API applications that cannot be made to interoperate with SSPI applications, and vice versa.

The *Simple and Protected Negotiation,* or *SPNEGO,* mechanism (ref. RFC 4178) can be used to extend GSS-API and is also used with Microsoft's SSPI. SPNEGO is actually a *pseudo-mechanism* (or *pseudo-service*) that allows parties to negotiate what security services they can both support. In other words, SPNEGO uses the security services of an underlying mechanism, such as Kerberos, to provide the security service of protected negotiation to an application. The application selects the desired underlying security service while minimizing the possibility that an attacker can disrupt the negotiation. An important benefit of using this form of negotiation is that it helps ease the transition from less secure authentication services toward more secure services, such as Kerberos. Consequently, in an environment where applications based on the GSS-API have been deployed, new authentication services can be introduced in such a way that stronger services are preferred and older, less secure authentication services can be phased out in a graceful manner.

SASL and Kerberos

The *Simple Authentication and Security Layer,* or *SASL,* framework was introduced as a way to further insulate applications from underlying authentication and other security services (ref. RFC 4422). SASL provides an application framework (more than just an API) that simplifies building secure applications that only need a simple, bidirectional stream-oriented communications facility between peers. It is particularly effective at allowing applications to simultaneously support various authentication mechanisms with optional confidentiality and integrity assurance services. SASL uses the GSS-API for Kerberos authentication support, and optionally for data confidentiality and

[12] Microsoft tends to describe applications that are compatible with GSS-API wire-level protocols as "GSS Aware" applications. The SSPI interface is a proprietary Microsoft specification described in this document: http://msdn.microsoft.com/en-us/library/aa380493.aspx

integrity. It is also common for SASL applications to support TLS (SSL) as an alternative means for providing data confidentiality and integrity protection.

Some popular Internet applications that are available in SASL-based versions include:

- BEEP: Block Extensible Exchange Protocol, which is itself a framework for applications that need to exchange block-oriented messages and manage queries and responses
- IMAP: Internet Message Access Protocol, which is a popular protocol for email clients to interact with email servers, including Microsoft Exchange servers
- LDAP: Lightweight Directory Access Protocol, which is the foundation for most of the modern directory services, including Microsoft's Active Directory
- POP3: The widely used Post Office Protocol (version 3) used by email clients to download email from servers
- SMTP: Simple Mail Transfer Protocol, the "push" protocol used to send (and receive) email from clients to servers and to exchange email between servers
- XMPP: eXtensible Messaging and Presence Protocol, an instant messaging protocol used in Jabber and other IM applications, such as Pidgin

Kerberos is widely used by SASL versions of the email protocols and LDAP directory services, as well as some applications based on XMPP.

Channel Binding

A more recent approach to allowing authentication methods to be used in conjunction with other security protocols is termed "channel binding" and is described in RFC 5056. Channel binding allows a mutual authentication exchange to authenticate a secure "channel" between the parties as well as the parties to each other. This is useful where applications may have access to secure protocol services, or "channels," such as TLS or IPsec that can be used for communications between parties. The problem is: how do applications know that the party they have authenticated is also the party on the other end of the secure channel? If the channel can be uniquely referenced, and some channel identifier can be included in the authentication exchange, then it is possible to "bind" the channel to the mutually authenticated session between the parties. In other words, they will know that the party they have authenticated is also the party on the other end of the secure channel.

Channel binding provides a more general approach than what is provided by the SASL framework where it is desirable to use Kerberos with other security services, such as TLS or IPsec. While not widely deployed yet, this approach is likely to become a best practice for deploying secure applications that need flexible approaches for decoupling authentication from other security services.

Pluggable Authentication Modules (PAM)

The Pluggable Authentication Modules (PAM) framework was originally proposed by Sun as a mechanism to decouple authentication from login applications. PAM was originally proposed as part of the Common Desktop Environment. Since then, PAM has gained acceptance on most Linux and UNIX platforms.

PAM provides system administrators with a configuration file allowing them to specify what PAM modules (and thus what authentication mechanisms) are used. Applications such as the Gnome or KDE login screen call into PAM with a userid and password. PAM runs through each of the configured modules in series, allowing them to interact with the authentication process. Modules can ask for additional information (*e.g.,* a one-time-password challenge) from the user. Modules can indicate that authentication has failed or indicate that the current module has succeeded. Typically, if all modules succeed, then authentication is successful. PAM also provides some facilities for session management and authorization.

Most Linux distributions and Solaris include a PAM Kerberos module. This module performs Kerberos initial authentication and obtains Kerberos tickets for the user. Often these modules are used to authenticate a user to the local machine: if the user is able to obtain Kerberos tickets, then the machine running the PAM module has confidence that the Kerberos principal is a name for the user. To gain this confidence, the PAM module must take the resulting Kerberos TGT and use this ticket to authenticate to some service (*e.g.,* the host principal or computer account) on the local computer. This is because the initial Kerberos exchange authenticates the user and KDC to each other but does *not authenticate the local machine* to either. Consequently, in order to have confidence that the user is who they claim to be, the local machine must be registered as a Kerberos service.

PAM serves two roles in Kerberos environments. First, it serves the role described above: authenticating local users of Linux and UNIX environments. However, PAM can also be used with network applications to allow users to use Kerberos principals and the corresponding password to authenticate over the network. This is *very different* from the way GSS-API or SASL use Kerberos. GSS-API does not send the long-term shared secret over the network; instead the Kerberos application authentication exchange is used to prove knowledge of the session key associated with a ticket. Consequently, GSS-API requires support in the client application. When PAM is used for authentication with the network application, the server receives the long-term password and Kerberos exchanges are performed on the server in order to confirm that the supplied password is correct. Unlike the GSS-API case, the PAM application needs to be trusted with the user's long-term secret. In addition, the secret is likely to be intercepted as it is transmitted over the network. Clearly, using PAM with network applications does not provide the security advantages typical of Kerberos. However, it does allow the Kerberos infrastructure to be used even when clients do not directly support it. Best practice is to prefer client-side support for Kerberos to the use of PAM for network applications.

PAM modules are also available to move from other authentication systems to Kerberos. These migration modules will take a supplied userid and password that has already been verified by a PAM module for the existing authentication infrastructure and register the principal in the KDC database. The migration module needs to be trusted to register principals in the Kerberos database. Typically, this trust is limited by allowing the module to create new principals but not to modify existing principals.

Unfortunately, while PAM is available on many platforms, there are significant differences in how PAM works on each platform. Multiple attempts have been made to standardize PAM, but the implementations in use today do not match each other or the standards. As a result, system administrators and application developers should be careful to consult the documentation for PAM on their platform.

Kerberos and Public Key Cryptography

Kerberos was developed before *asymmetric* (a.k.a., *public key)* cryptography was a viable technology for deployment in commercially available systems. Consequently, Kerberos authentication is based on use of *symmetric* cryptography. The implication is that symmetric encryption keys must be exchanged between parties as a prerequisite to being able to perform authentication. For human users, passwords are used to derive symmetric encryption keys for authentication and exchange of subsequent session keys.

In practice, every user and service principal must go through an initial setup process where passwords or symmetric encryption keys are exchanged with a KDC. While this can be done securely with proper procedures, it is still an administrative headache. Furthermore, the shared secret key can be compromised by either party. Changing passwords and keys can also be a burden for users and system administrators.

The appeal of asymmetric cryptography is that there are two keys, where anything encrypted with one key can only be decrypted with the one-and-only corresponding key. When asymmetric "key pairs" are generated, it is common practice to treat one as a "public key" that can be freely shared with other parties while the other key in the pair is kept as a "private key" that should not be shared with any other parties. This greatly simplifies key sharing, since the public key can be freely exchanged between parties provided the private key is not disclosed.

The primary concern with distributing public keys is determining whether or not a public key really belongs to a specific party. This is generally solved through use of public key certificates (ref. RFC 5280), which include the name of the party (subject) and their public key in a document that is digitally signed by some "Certification Authority" or "CA." Any party that receives a certificate can validate the CA's signature to confirm that the public key really belongs to the subject named in the certificate—presuming, of course, that the CA that issued the certificate is considered trustworthy.

Over the past decade, asymmetric cryptography and public key certificates have become widely supported technologies that are now built into most computing platforms. Furthermore, Kerberos has been extended to take advantage of these technologies, primarily for initial authentication of users requesting TGTs, though other extensions have been proposed for including public key certificates in Kerberos tickets.

PKINIT—Public Key Cryptography for Initial Authentication in Kerberos

RFC 4556 (PKINIT) specifies extensions to KDC Authentication Services that allow clients (users) to be initially authenticated by presenting their public key certificates instead of using previously shared secret passwords. Only the initial authentication procedure is changed; all other Kerberos interactions remain the same. In particular, clients continue to receive Ticket Granting Tickets (TGTs) that will be used to subsequently request tickets for services, and services continue to authenticate users via tickets issued by KDCs. A service has no need to support public key cryptography, or even know how the client initially authenticated to the KDC. From an application perspective, client and service interactions are exactly as before.

An important consequence of using PKINIT is that users can more easily be given an account in a realm they have never visited before. If an employee or contractor with a trusted certificate moves to a new part of an organization, their account can be waiting for them with no need to register a new password. Furthermore, use of public key certificates helps mitigate risks with weak user passwords, since the user's private key is used during initial authentication instead of a password. A user may still enter a password, but only on their local workstation or device to unlock the private key. Password changes tend to be handled at the user device level, and do not have to be coordinated with other systems.

At least theoretically, using PKINIT, user principals no longer need to be registered in the KDC database before initial authentication, since the KDC does not need to look up the corresponding shared secret for a user with a certificate. Service principals must still be registered in the KDC database along with a shared symmetric encryption key. A KDC database could be much smaller, since there tend to be more user principals than service principals. However, in practice, users still need to be registered to store policy and other information. Also as discussed below, entitlement information and other attributes about a user are typically stored in some directory. PKINIT does not reduce the need for this information. With PKINIT, the administrative overhead for maintaining the KDC database should be reduced, and user password changes would no longer need to be supported for users with certificates. For organizations that already issue certificates to users, it is possible to allow users to authenticate with their certificate across multiple authentication systems, including systems that use TLS/SSL and various VPN technologies in addition to Kerberos. At the same time, applications based on Kerberos do not need to be changed.

Another benefit is that certificates can be associated with hardware cryptographic tokens, including "smart cards" and USB devices with embedded "smart chips." Some biometric authentication devices can also be used with certificates, but where the biometric device is used to unlock access to the user's private key.

Of course, these benefits come at the expense of deploying a Public Key Infrastructure (PKI) that includes Certification Authorities (CAs), certificate request and issuing services, certificate revocation procedures, status checking services, and directories for retrieving certificates for specific users or other principals. However, PKI has become an integral part of several vendor platforms, and it is widely available. In many environments, the advantages of public key cryptography and certificates more than justify the added burden of deploying and maintaining a PKI. Alternatively, PKINIT can be used without a PKI. The KDC database can store self-issued certificates associated with user principals. This avoids reliance on a CA, since the KDC can trust a public key in its database, and no password secret is needed for the user. However, other advantages such as easy migration between realms and support for applications beyond Kerberos tend to require a PKI.

One concern with certificates used in authentication is that they tend to have relatively long lifetimes on the order of months to even a couple of years. This leads to greater exposures when private keys are compromised, or a user's certificate needs to be revoked for some cause. This means that a KDC might allow a user with a revoked certificate to authenticate, and subsequently access application services. There are two approaches for mitigating this risk, (1) distribution of Certificate Revocation Lists (CRLs) and (2) use of Online Certificate Status Protocol (OCSP) services (ref. RFC 2560). Use of OCSP services is particularly attractive, as it allows certificate status to be checked in real time as part of the initial authentication process. RFC 4557 specifies how PKINIT can be further extended to integrate OCSP checks during Kerberos initial authentication.

Support for Multiple Kerberos Realms

A Kerberos realm must be administered as a single database of principals and associated keys. For each realm, there is some registration process that enrolls principals into the realm's database, and only these enrolled principals are able to authenticate each other. Although an enterprise may choose to have only a single realm, practical considerations often result in multiple realms being established under different administrative regimes. While having separate realms for different administrative regimes solves some problems, it also leads to additional complexity and new problems.

Before discussing support for multiple realms, it is worth restating that a single realm can be supported by multiple KDCs. This may be done to improve availability by having redundant KDCs or KDCs that are distributed closer to users and services. In some cases, it may also help improve performance by having the workload distributed across multiple KDCs. However, all KDCs within a realm use the same KDC database, which must be replicated from a master KDC to all other KDCs in the realm. The reason for emphasizing this point is that other deployment strategies duplicate realms along with KDCs. In particular, a Microsoft Active Directory deployment will tend to distribute Domain Controllers in much the same way that KDCs might be distributed. However, while a Domain Controller is also a KDC, each Domain is *also a separate Kerberos realm* with its own subset of principals that are allowed to authenticate to the Domain.

Cross-Realm Authentication

When multiple realms are used to support a shared population of users and services, the obvious problem that emerges is how a client in one realm can authenticate to a service in another realm, and vice versa? Kerberos solves this in a direct manner by making the Ticket Granting Service (TGS) of one realm, a principal in the other realm. Then, when a client wants to access a service in another realm, it first asks a KDC in its realm for a ticket to the TGS of the realm where the service resides. Technically, clients make this request to the Ticket Granting Service associated with their KDC, but since the TGS is essentially always part of the KDC, it is common to use "KDC" synonymously with "TGS," even though this is not strictly correct terminology.

Since the client's realm has a principal registered in its KDC database for the remote realm's KDC (TGS), it is able to authenticate the client and remote KDC to each other in the same way as for any other service located within the client's realm, and the client gets back from its KDC a ticket to the remote KDC (TGS). In this case, the ticket is actually a Ticket Granting Ticket (TGT) for the remote KDC realm. The client can then access the remote KDC, and use the TGT it has received to request a ticket for any service in the remote realm. This is illustrated in Figure 4 below.

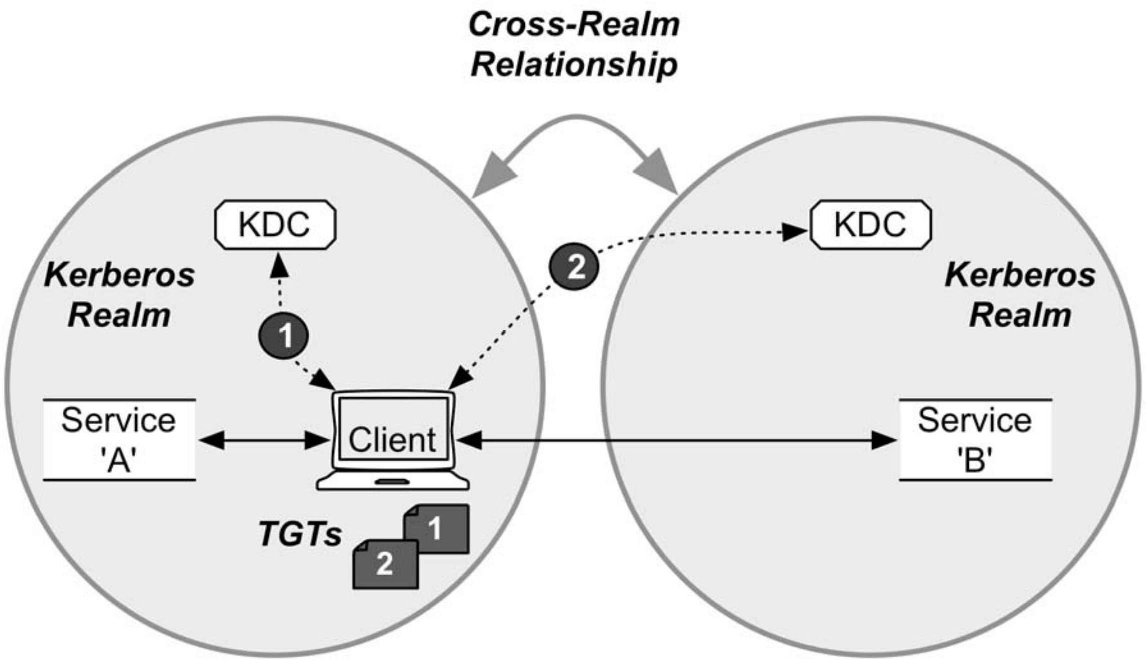


Figure 4: A client in one realm can authenticate to a service in another. This is done by requesting ❶ a ticket from the local KDC for the TGS of the destination realm's KDC. The client is then able to request ❷ a service ticket from the destination realm.

The salient points regarding cross-realm authentication are:

- Before anything else, the client must initially authenticate to the Authentication Service (AS) for its local KDC and receive back a TGT.
- The client must know the realm of the service it intends to access, and be able to recognize when a service is in a different realm from its own. In large, distributed environments, this can be a more difficult problem than it first appears.
- The remote KDC's TGS must be a registered principal in the client's realm. The principal name for the remote KDC will be:
  krbtgt/REMOTE-REALM@LOCAL-REALM
  where normally the remote and local realm names will be uppercased versions of the corresponding DNS names. This convention makes it possible for a client that knows its own realm and the remote realm name to be able to request a ticket from the correct TGS service.

- The client must ask its KDC for a ticket to the KDC (TGS) in the remote realm. From the local KDC perspective, this is the same as any other request for a service ticket, except that the ticket provided to the client is a TGT for the remote realm.
- The client must use the new TGT it receives to request that the remote KDC grant it a service ticket for the actual service it wants to access. From the client's perspective, this is the same as requesting a service ticket from its local KDC. From the remote KDC's perspective, the client is similar to any other client that was locally authenticated and given a TGT.
- When the client receives a ticket for the service it wants to access in the remote realm, it presents the ticket to the actual service in the same way it would a service that was located in the same realm.
- A service receiving a ticket from a client in a remote realm can process the ticket in the same way it would a ticket received from any client in its realm. However, the service is able to examine the ticket to determine that the client is from another realm, and to make access control decisions based on the client's realm.

This direct cross-realm authentication process provides an elegant solution to the problem of authenticating across realms. In practice, if clients and services from two realms need to be able to interact with each other, then each realm must register the other realm's KDC (TGS) as a service principal. This implies two shared secret keys will be needed between the realms to establish a *bidirectional cross-realm relationship* between the realms.[13] If there are more than two realms, then every realm will need to establish a cross-realm relationship with all other realms by registering as a service in every other realm, and also registering every other realm's KDC as a service within its realm. This is a full mesh topology with bidirectional setup required where, for *n* realms, there will need to be $n(n-1)$ cross-realm registrations for the bidirectional relationships. If there are more than a few realms, the management of the cross-realm registrations can become an administrative challenge.

Transitivity and Scaling Cross-Realm Relationships

The approximate *n*-squared growth in cross-realm relationships for *n* realms presents a scalability challenge. Several approaches have been defined for managing scale in multi-realm situations, as outlined below:

- Configurable Authentication Paths
- Hierarchical relationships based on mapping of realm names to a DNS name hierarchy
- Automated configuration tools
- Use of client public key certificates and PKINIT

[13] Kerberos also supports *unidirectional cross-realm relationships,* where clients can authenticate to services in a remote realm, but not vice versa. This might make sense if a realm is defined where services are located that will be accessed from other realms, but no clients that need to access services outside the realm.

Configurable Authentication Paths

The first approach depends on the idea that cross-realm relationships can be extended to be transitive relationships. Authentication flows along an "authentication path" from the local realm, through one or more intermediate realms, to the remote realm. In a typical example, some central realm might set up bidirectional relationships with all other realms. The client would obtain a TGT for the central realm from the local KDC and use that TGT to obtain a TGT for the remote realm from the central realm's KDC. Then the client could contact the remote KDC and obtain a ticket for the service.

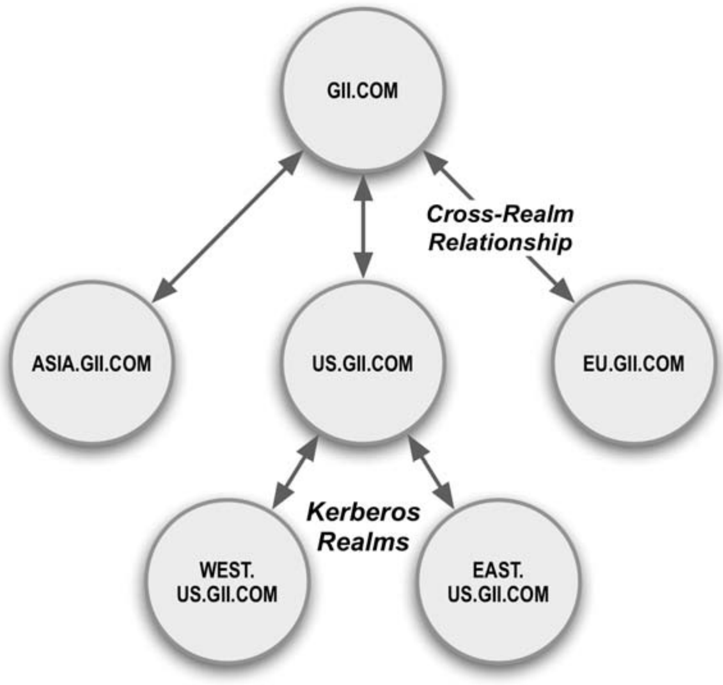


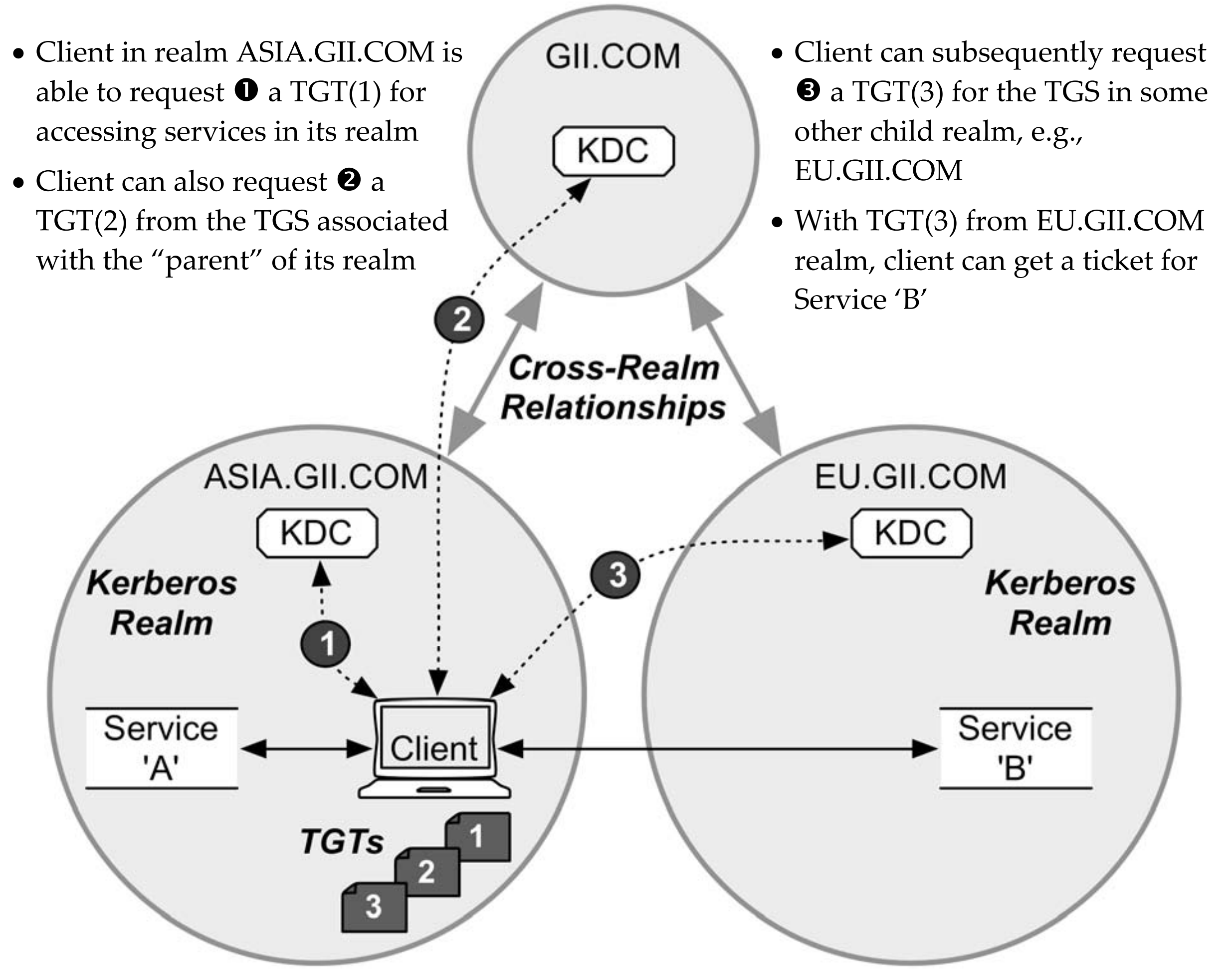
Figure 5: Example of hierarchically organized cross-realm relationships

The first approach permits configuration of authentication paths. These configurable authentication paths (*capaths*) specify the set of intermediate realms that need to be traversed to get from a local realm to a remote realm. The term capath is not based on the concept of a public-key certification authority or the concept of a certificate validation path; it is an abbreviation for configurable authentication path.

A complexity with capaths is that every client needs to know the available pathways.[14] This configuration matrix has to be built and installed on every system that will operate across the extended network of realms. Furthermore, to avoid problems with inappropriate capaths, service tickets are issued with the list of intermediate realms included, and services or KDCs must check the path in the ticket against their local configuration of acceptable paths. This requirement to distribute and maintain capath

[14] Either the service or the KDC also needs to be able to validate the path used by the client. Ultimately the service is responsible for deciding whether the path is acceptable and whether authorization should be restricted because of the path used. However KDCs provide a facility for checking transited path policy that services should rely on.

information presents its own scalability challenges. However, if clients and KDCs support realm referrals as discussed in Support for Kerberos Referrals below, then only KDCs need to be configured with capaths. For some environments, such as Active Directory Domains, this scales well.



• Client in realm ASIA.GII.COM is able to request ❶ a TGT(1) for accessing services in its realm
• Client can also request ❷ a TGT(2) from the TGS associated with the "parent" of its realm
GII.COM
KDC
• Client can subsequently request ❸ a TGT(3) for the TGS in some other child realm, e.g., EU.GII.COM
• With TGT(3) from EU.GII.COM realm, client can get a ticket for Service 'B'
2
Cross-Realm Relationships
ASIA.GII.COM
KDC
Kerberos Realm
1
3
EU.GII.COM
KDC
Kerberos Realm
Service 'A'
Client
Service 'B'
TGTs
1
2
3

Figure 6—Example of hierarchical transitivity

Hierarchical Transitivity

A variation on the inter-realm path concept is a strictly *hierarchical path* that leverages consistent hierarchical realm naming conventions to allow authentication paths to be implicitly known. Generally, the naming hierarchy is based on DNS names. For example, if Galactic Industries Inc. has the DNS domain name of *gii.com,* and there are three regions in Asia, Europe, and the United States with the corresponding domain names of *asia.gii.com, eu.gii.com,* and *us.gii.com,* then a hierarchy with four realms can be established with Asia, Europe and the U.S. all transiting an intermediate realm named GII.COM (see Figure 5 above). Since the path is implied by the naming, the need to explicitly configure capaths in clients and services can be avoided. Arbitrary hierarchies can be established, provided they follow strict hierarchical naming conventions. However, not all organizational structures will fit a strict hierarchy, or will be able to use one hierarchical naming scheme. Figure 6 below illustrates how a client in one realm can use hierarchical transitivity to access services in other realms.

Configuration Tools

Another way to manage relationships amongst Kerberos realms is to build an administrative tool that can automate establishing cross-realm KDC registrations, and that might even handle deploying capath matrices to clients and services. Assuming that a database of realms and desired relationships is built and maintained within a tool, the tool can use configuration rules and policies to automatically generate all required configurations and setups. In practice, directory services have become preferred tools for centrally managing information required to deploy and operate complex distributed systems. Consequently, practical solutions to automating configuration of cross-realm relationships have been built based on directories. For example, Microsoft uses Active Directory to maintain cross-realm relationships throughout an arbitrarily complex organizational structure.

PKINIT for Client Authentication

A distinctly different approach leverages the PKINIT Kerberos extensions as described above in the section on PKINIT—Public Key Cryptography for Initial Authentication in Kerberos. Instead of clients using their own KDC to get tickets for services in a remote realm, a client with a public key certificate could contact a PKINIT-capable KDC directly in the remote realm. Since the client does not have to be previously registered in the remote realm to get a TGT from the KDC, there is essentially no cross-realm path to be followed. In essence, trust relationships are established and maintained within the Public Key Infrastructure (PKI) used to issue and manage certificates. These trust relationships are used to facilitate Kerberos authentication.

Hybrid approaches can also be pursued. For example, the hierarchical approach could be used with capaths, but where the capaths are only used by a subset of clients and services. This might allow for direct cross-realm relationships in addition to hierarchical relationships. Another hybrid strategy might use PKINIT with hierarchical relationships to support both traditional clients, and clients able to authenticate using public key certificates.

Cross Realm Relationships and Trust

Realms may have significantly different policies for what principals are registered in the KDC database. Some realms may allow anyone to obtain a principal and may have loose control over the security of the KDC. Other realms have strict policies on what procedures are followed before registering a principal and tightly control KDCs. Consequently, it is reasonable for services to have significantly different confidence in the authentication of principals from one realm than principals from another realm. Clients also need to evaluate their confidence in the authentication of the service. Of course, confidence in authentication should be limited to the weakest realm that is involved in the authentication process. Services and users then decide whether to trust the authentication in the context of their access control policies based on their confidence that the asserted principal actually names the party involved in the communication.

Because services must make this trust decision, Microsoft refers to cross-realm relationships as “Domain trusts.” While cross-realm relationships do involve trust decisions, the level of trust that a service chooses to place in a given realm may be very low. In a Microsoft context, it rarely makes sense to establish a relationship with a relatively untrusted Domain.

Administrators should think carefully about what level of trust in the authentication process is appropriate when they place a principal from a foreign realm on an access control list or otherwise grant that principal access. If an organization has relatively liberal policies about when a cross-realm relationship is established, it is easier to establish cross-organizational collaboration. Those granting access to foreign principals can evaluate whether the foreign practices are sufficient. However, auditing whether these decisions to grant access are justified may be difficult. If this is problematic, an organization may establish minimum requirements for cross-realm relationships. Application authors should endeavor to make it easier to audit access control decisions with an open cross-realm policy by providing mechanisms to establish realm level policies regarding what access is appropriate.

Managing Multiple Kerberos Realms

In addition to the challenges of finding the authentication path, described above, there are other problems or issues that need to be addressed in real-world deployments involving multiple realms, especially in mixed environments. Perhaps the most significant issue is how clients find the correct name for services in other realms, and how they determine what realm name to use for the service they want to access. One approach to dealing with this challenge is to leverage DNS names for services to map to realm names. This approach is discussed further below in the section on Issues with Relying on DNS. Another common approach is to provide a directory service that can be used to discover services and associated realm names.

Another concern with establishing cross-realm relationships is the means used to share secret keys between KDCs. To avoid disclosure of secret keys used by KDCs, each new relationship will require separate secret keys used for each direction. Since KDCs are specialized services, the process of establishing these shared secret keys can vary from implementation to implementation. This results in added complexity when multiple realms are used with different implementations. Often, the only approach available is to set up passwords as shared secrets. Obviously, such passwords must be immune to password-guessing attacks, and should be generated as true random strings. Another potential problem is determining what encryption algorithm will be used, since stronger algorithms are clearly preferred, but may not be the default. Particular care is needed to make sure that DES is not used as the encryption algorithm, given its weaknesses. When registering a KDC (TGS) in another realm, especially when a different KDC implementation is involved, it is vital to check relevant documentation and determine an approach that will provide the greatest protection against attacks on the secrets.

As noted above, policies may not be consistent across realms. For example, one realm may place stringent restrictions on what principals are registered in the realm, while another realm might take a more relaxed approach to registering principals. Such discrepancies in policies may result in unintended weakening of a realm's policies, or a general tendency to devolve to policies that reflect the lowest common denominator. Kerberos, by itself, does not provide adequate means for expressing policies across realms. Consequently, this is a matter of agreements between administrative regimes and may require external methods for expressing acceptable policies across realms.

Audit controls are also more complex in a multi-realm environment, especially when applied inconsistently across realms. For example, if a service in one realm expects the KDC to provide an audit trail (system log) of all client authentications and ticket requests, but some clients are initially authenticated by a KDC in another realm, then audit practices will be weakened if it is not possible to access the remote realm's audit records.

The Role of Directory Services

All modern information services rely to some degree on tables or files that can map names to other names or attributes. However, when this information is built into individual applications or configured into every instance of an application, then it can become awkward to maintain this information throughout an evolving system. Furthermore, there is usually considerable duplication of information, and often there are critical requirements to keep this information up-to-date and synchronized across distinct system entities. Consequently, various services have evolved that maintain directories of information that can be used across applications as a way to reduce administrative burdens. For example, while it was once acceptable for every Internet host to have a "hosts file" used to map host names to IP addresses, the impracticality of this approach as the Internet grew led to the DNS service providing a common directory service that every host can use to look up other hosts throughout the global Internet.

In any access control context, directories are vital services for name mapping, associating named entities with attributes, and for maintaining information relationships necessary to enforcing policy rules. These directories may reside in applications, such as userid/password lookup files or access control lists (ACLs). Alternatively, directory services may be used to maintain such information throughout an enterprise.

Directory services are often associated with authorization, since they can greatly simplify keeping track of attributes used in authorization decisions when controlling access to information or other resources. For example, directories can maintain group memberships or ACLs that can be shared across many applications that need to make authorization decisions. However, directories are also used to maintain information used in authentication or to share public keys used to provide confidentiality or data integrity.

Kerberos relies on the KDC database—a type of directory—for maintaining information used in authentication and ticket granting decisions. Kerberos also relies on DNS services and is increasingly associated with LDAP directory services. The following sections explore the role of the Kerberos database (KDB) and other Kerberos dependencies on external directory services.

The KDC Database (Directory)

In the section on Naming in the Kerberos Context, the concept of principal names was introduced as the means for associating an authorized party with a contextually unique name. In practice, a party is *authorized* if it has a principal name within the KDC database, or KDB. Put differently, no party can authenticate via Kerberos unless their principal name has been entered into the KDB, and such unregistered parties are implicitly *not authorized.* A party authenticates to their unique principal name via Kerberos protocols based on secrets known only to the valid party and the KDC, or via other tests using information shared between the party and Kerberos service.

The KDC database exists to map principal names to associated secret keys, along with administrative information such as “last modified date” and policy information, such as “maximum ticket life.” In other words, the KDC database[15] is a simple directory used to look up keys and attributes for a specified principal name. However, this database/directory exists only to support the operations of KDCs and administration of a Kerberos realm. It is not offered as a service to other parties. The only exposed interfaces are for administrative functions.

Administrative functions—such as entering new principal names (*i.e.,* registration), changing secrets (*e.g.,* passwords), or modifying policy parameters—while necessary for any functional Kerberos system, are not provided in the same manner on all platforms. While the Kerberos protocols are well standardized and interoperate in mixed environments, administrative functions are not well standardized, and interoperability has proven to be a challenge for system administrators. Typically, administrative functions are secured using the access control facilities of the host on which KDC services operate, or Kerberos is itself used to provide access control by using special “admin” principals and configuration files that determine which principals are authorized to perform administrative functions.

Since KDCs can be distributed within a network to provide redundancy and improved accessibility in large networks, the KDC database needs to be replicated onto each KDC instance. However, KDC database replication tends to be simplistic, and lacks the flexibility and scalability normally associated with directory replication. Traditionally, a KDC database is kept on a master KDC and pushed *en masse* to all other KDCs on a periodic basis. Recently, some Kerberos implementations have begun to introduce replication schemes that operate incrementally, pushing out only changed records, instead of the entire database. Another approach to maintaining KDC databases is to utilize a more robust directory service for building, administering and distributing the

[15] The KDC database is sometimes called the “account database” or “account directory,” especially in Microsoft technical literature.

information required by KDCs. Current Kerberos implementations directly support use of standard directory services as an alternative to a standalone KDC database. This approach is discussed further in Integrating Kerberos with Directory Services below.

There are two primary advantages to the simple KDC database model. First, separating the KDC database from other directories reduces the size of the most sensitive security data. KDC database backups must be kept confidential, and changes to the KDC database need to be carefully audited. If this sensitive information is combined with another directory, then the entire directory will need to be treated with this level of care. Secondly, the simple KDC database allows relatively efficient operation. As a result, a single KDC realm can support tens of thousands of principals with adequate performance. Similarly, in a campus setting, a single KDC can handle requests from user populations that exceed ten thousand users. However, modern directory services also scale well, so that even larger populations can be supported using Kerberos integrated with directory services. For larger populations, the administrative advantages of directories argue for using integrated approaches. When Kerberos is combined with an efficient, well-administered, distributed directory service, it is feasible to scale to global populations.

DNS and Kerberos

In addition to the KDC database, Kerberos uses local configuration (*e.g.*, configuration files or registry entries) to establish operational parameters and mappings for realms and KDCs. In particular, a mapping between realm names and corresponding DNS domain names is necessary, plus a lookup table of KDCs within each realm. It may also be necessary to provide lookup facilities for Kerberos administration services and password changing services. Since at least a subset of this information is needed by every Kerberos client, there is a scalability challenge in maintaining these configuration files across an entire population of Kerberos users. Every time a new realm is added, or a new KDC added to a realm, the configuration files will need to be updated for many, if not all, clients.

Since DNS is a ubiquitous service in practically every internetwork (private or public), Kerberos has evolved to rely on DNS as a means for centrally managing required mappings. By adding SRV records to the local DNS service, it is possible to enable Kerberos clients to discover KDCs, realms, and related Kerberos admin and password-changing services. In addition, the SRV records provide information about priority order in which to access KDCs or other Kerberos services, as well as weighting information that can be used in load balancing between KDCs at the same priority level. This makes it relatively easy to add or change Kerberos KDCs and services within a realm, and to implement either redundancy or load balancing, or both—all without having to modify the configuration files of clients and servers that use Kerberos.

Realm to domain name mappings can also be maintained within a DNS service by using TXT records that simply register a domain name with an associated Kerberos realm name. Note that if the realm name is not a trivial transformation of the domain name, then new DNS zones will have to be created for each Kerberos realm, and any SRV records will need to reside in the DNS zone that corresponds to the realm name.

DNS is also a simple directory service with built-in replication mechanisms for extending knowledge about domain name mappings throughout an enterprise, across multiple enterprises, and across the global Internet. Applications in general, and Kerberos specifically, depend on DNS to provide mappings for domain names outside of the local domain or realm. Without DNS replication, Kerberos would not be able to provide authentication between clients and services located in different realms—*i.e.*, cross-realm authentication requires working DNS services.

Issues with Relying on DNS

While use of DNS can greatly simplify administration of Kerberos realms and associated clients and servers, it also introduces some constraints and various problems, including security vulnerabilities.

One notable constraint introduced by reliance on DNS is that case-sensitive realm names must not be equivalent if transformed (or "folded") into case-insensitive domain names. To illustrate by example, if there are two realms named "KERBEROS.ORG" and "Kerberos.org" where case sensitivity is required to distinguish between them, then such realm names cannot be used with DNS, since domain names are case insensitive, and there is no way to establish DNS mappings for SRV records that would differentiate between such names. In practice, this is rarely a problem, since Kerberos realm names are seldom differentiated by case alone. Such realm names are forbidden by the Kerberos specification. The established convention of using the all-caps version of a domain name as the realm name avoids this problem with DNS, and is recommended for most situations, and even required by some implementations.

Another constraint with using DNS is that Kerberos principals must be able to access the DNS service in order to perform Kerberos authentications. In particular, use of Kerberos facilities will generate DNS traffic in addition to the Kerberos protocol exchanges. This adds complexity that must be handled properly within applications, especially with regard to handling of DNS error conditions or the inability to access DNS services. For example, converting a login procedure to first try Kerberos authentication may result in failures or long timeouts related to DNS queries even if other authentication options are available.

Earlier Kerberos specifications and implementations also made use of DNS to help discover the name of the service being requested. For example, the IP address of a requested party would be presented to DNS as a reverse lookup to get back the host name associated with the IP address, which would then be used to construct the principal name or used to discover realm names. While such use of DNS queries is discouraged in current Kerberos standards, it is still quite common in widely deployed Kerberos implementations. Consequently, it is usually required that if hosts offering *Kerberized* services have DNS entries that can be resolved via a reverse lookup, the resulting hostname is the one registered in Kerberos. In environments where hosts are assigned dynamic IP addresses, this can be a problem unless a dynamic DNS service is available to automatically update IP-address-to-host-name mappings. DNS can also be coupled to a DHCP service in such a way that as hosts are assigned IP addresses by the DHCP service, they are also registered with that IP address in the local DNS service.

Another concern with using DNS is the inherent insecurity of the basic DNS service.[16] In particular, it is easy for attackers to impose their own replacement DNS services, or to cause local DNS services to provide invalid responses to queries (*e.g.*, by "poisoning" the caches of local DNS servers). Since DNS responses are not secured in any manner, it is not reasonable to place even modest trust in DNS services. In particular, it is *not* possible to trust:

- DNS host name to IP address mappings
- DNS reverse lookups
- Kerberos host name to realm name mappings (if used)

As a result of these DNS vulnerabilities, an attacker can often mount a *denial-of-service* attack, preventing access to resources. Fortunately, most of Kerberos's uses of DNS do not depend on the integrity of DNS data. In particular, the SRV-based discovery mechanism for KDCs and admin servers, and the use of DNS to translate names to IP addresses are not critical to the security of Kerberos. After information is retrieved from DNS, Kerberos authentication is used to confirm that the right party is contacted. If the DNS information is modified, this modification will be detected.

The use of DNS and DNS reverse resolution to determine the name of the service to contact is much more problematic. Unfortunately, there are no viable solutions for the DNS vulnerabilities in this context, and the latest Kerberos standards (ref. RFC 4120) state that Kerberos implementations *must not* use DNS queries to map one name to another to determine the host part of the principal name. The concern is that a client that gets an erroneous mapping to a host name from the DNS service may wind up authenticating to the wrong service. However, this service would still have to be registered in the KDC database of some realm, so for applications that require successful Kerberos authentication, the exposure is limited to attacks that corrupt DNS to replace one service with another. As a practical matter, current Kerberos implementations *do* rely on DNS services, which implies that system administrators must take reasonable care in protecting DNS against attacks, or at least to be able to quickly detect attacks that affect DNS services. In practice, the problems with DNS insecurities can be mitigated through effective administrative practices for both DNS and Kerberos services. In particular, restricting which hosts and services are registered in the KDC database will reduce potential exposures.

Microsoft has already adopted extensions to the KDC database to avoid dependence on DNS for mapping host names to principals. The best long-term practice for avoiding the problems of DNS is for other implementations to adopt similar strategies.

Directories and Kerberos

While DNS and Kerberos both utilize simple internal directories, and DNS even offers a simple distributed directory service, neither provides sufficiently robust or secure directory facilities to meet the complex challenges of controlling access to modern

[16] While a secure DNS service (known as "DNSSEC") does exist, it has seen only scant adoption, and is not viable in the vast majority of system deployments today. This may change in the future.

information systems. Consequently, a common strategy has emerged over the past decade to leverage secure, robust directory services alongside of DNS and Kerberos. The directory services that are commonly deployed today are based on the Lightweight Directory Access Protocol, or LDAP,[17] which is in turn based on X.500 directory standards.[18]

However, directories are being deployed for many reasons besides supporting access control requirements; some important system benefits include:

- *Update once—access everywhere* data model where information is entered or updated once by any authorized source and becomes available throughout an enterprise, or even globally.
- Reduce duplication of information across diverse applications operating within a distributed context.
- Common administrative procedures and tools can be used to maintain information throughout an enterprise, including replication of information.
- Applications no longer require their own internal directories or even some configuration files, and can also make use of new information objects with minimal development effort.
- Centralized control of access to directory information simplifies security measures, which also implies that information retrieved from directories is more reliable and trustworthy.

Additional features and capabilities of distributed directory services that are relevant from a security perspective include:

- Many types of names and addresses can be mapped to each other—*e.g.,* associate a person's name with their userid, Kerberos principal name, organization, department, office, telephone number, email address, employee number, distinguished name.
- Globally unique identifiers can be established that allow entities in a directory to be moved and renamed without losing track of who or what is being referenced.
- Groupings (a.k.a., containers) can be maintained that list entities within the directory that share some common attributes, purpose, or role—*e.g.,* a "department group" that lists all the members of a department within a larger organization.
- Each directory entry can be associated with various attributes—*e.g.,* roles, group memberships, status, privileges, entitlements, quotas, rank, security classification, subscriptions.

---

[17] LDAP v3 standards are listed on the archived IETF Working Group page at: http://www.ietf.org/html.charters/OLD/ldapbis-charter.html with additional information maintained at: http://www.openldap.org/

[18] The X.500 series of standards documents can be downloaded from the ITU-T web site at: http://www.itu.int/rec/T-REC-X/e and further references can be found at: http://www.x500standard.com/

- Common attributes can be defined and used consistently across many applications.
- Directories can serve as repositories for specialized information objects, such as digital certificates or protected keys.

The ability to retrieve attributes associated with a “principal” (*e.g.*, user or service) from a directory greatly facilitates *authorizations* within a distributed applications environment. Instead of each application having its own Access Control Lists (ACLs) or maintaining internal group memberships, an application can query a central directory for attributes associated with a party to determine what authorizations (*e.g.*, privileges, permissions) the party is entitled to.

Of course, applications must be able to trust responses they receive from queries made to a directory service for authorization information. This implies that the directory application itself must be secured.

Securing Directory Services with Kerberos

Modern LDAP-based directory services are themselves sophisticated applications that are used throughout an enterprise by many types of users and services. In many enterprises, every user is entitled to access at least a subset of the information contained within the enterprise’s directory service, and every user may even have permissions to modify certain data fields in a subset of directory entries. For example, one use of the directory service may be as a simple “white pages” for finding names, telephone numbers and addresses. Individual users might be allowed to modify certain fields associated with their own directory entry, such as a home telephone number or address.

However, much of the information in a directory must be tightly controlled in terms of *who* can access or modify information, as well as *what* entries or attributes can be read or modified. In other words, access control is essential to directory management and use, especially if directories are going to be used to store and distribute authorization information or other security resources, such as digital certificates.

While various technologies are used to secure access to directory services, Kerberos is often a preferred means for securing directories. Other technologies used either with, or as an alternative to, Kerberos include TLS (SSL), Microsoft’s NTLM and Netlogon, one-time password systems, various challenge-response schemes, and even simple userid/password authentication. For directory services that play a vital role in the overall security infrastructure of an enterprise, Kerberos and TLS are preferred security technologies, and are frequently deployed in combination.

Advantages offered by Kerberos in securing directory services include single sign-on for users, authentication of clients *and services,* mutual authentication between clients and the directory service, secure session establishment that provides data integrity between a directory service and users, cross-realm authentication that facilitates access throughout an extended enterprise, and support for delegation to proxy services.

While various approaches are used for building directory services to utilize Kerberos, a popular approach is to build directory applications using the Simple Authentication and Security Layer (SASL) framework, which decouples the application from the actual authentication methods used as well as the means for securing sessions. SASL, in turn, utilizes Kerberos via the GSS-API as one supported authentication method. Since SASL can also be used with TLS for securing sessions between parties, this approach provides robust security that can be flexibly configured according to the needs of a particular enterprise. While small enterprises might use simple password-based authentication for their directory services, larger enterprises typically use some combination of Kerberos with SASL and TLS to secure their directory services.

Integrating Kerberos with Directory Services

As noted above, a benefit of using a directory service is that it centralizes management of information and reduces the need for applications to maintain their own local directories. Kerberos is an example of an application that has an internal directory that duplicates information that must be kept and maintained in other places. Consequently, there are advantages to integrating directory services into Kerberos to replace or augment the KDC directory.

Not only does this eliminate some of the administrative overhead for Kerberos in a systems environment, it also provides robust methods for replicating directories across multiple KDCs—*i.e.,* KDCs no longer need to deal with replication or propagation of their databases. Furthermore, administration of some KDC information can be performed using standardized tools that have evolved with directory services. For example, password changes can be handled using directory applications that allow users to change only their own passwords (write only), while restricting password read access only to KDCs.[19] However, account creation and manipulation of KDC policy information requires specialized knowledge about the particular Kerberos implementation. If standard directory applications are used for these tasks, the applications need to be expanded to understand Kerberos and the specific KDC implementation.

There are two obvious approaches to integrating directory services with Kerberos: (1) synchronizing the KDC database with the directory, or (2) replacing the KDC database with direct use of the directory. The first approach minimizes changes to existing KDC implementations, since periodic extracts from the directory can be used to replace the KDC database with a fresh copy, but the KDC still uses the same internal database. One potential benefit is that if the directory service becomes unavailable, Kerberos operations would be able to continue. The second approach requires new implementations of KDC services, but has the advantage of enabling expanded Kerberos services. In reality, most KDC implementations already include options to be built using standard LDAP directory services. Microsoft chose to re-implement Kerberos as an integral component of its Active Directory services.

The ability of directories to store public key certificates for users and services provides new options for improving initial authentication to Kerberos through use of public keys instead of, or in addition to, passwords. This facilitates use of smart cards or other cryptographic tokens for initial authentication. Similarly, directory services can facilitate

[19] Of course, passwords are hashed and not used directly by KDCs. However, administrators may need to recover actual user passwords, which might be handled through a secure directory.

use of one-time password dongles for initial user authentication. The net result is that enterprises can more freely decide which initial authentication methods they prefer, or even to use different methods for different types of users or for specific applications.

Since directories can associate authorization attributes with principals, combining Kerberos with directory services enables new options for streamlining authentication and authorization for applications. For instance, Kerberos can authenticate a client and service to each other, and provide the service with an authenticated principal name that can be used in a subsequent directory query to pull authorization information for the client. Another option would be for Kerberos to pull authorization information from the directory service and include this information in the ticket provided to the application. This latter technique is used by Microsoft to embed Privilege Attribute Certificates (PACs) for the client within Kerberos tickets that are sent to services.

It is also possible to integrate DNS with directory services to further reduce system administration burdens.

Kerberos Role in Directory Trust Relationships

While a directory service can be deployed as a single application on a single host, it is more typical to deploy distributed directory services that employ multiple applications on distinct hosts. Not only does this distributed model help improve reliability and accessibility; it is also essential to deploying directory services that span multiple organizational boundaries.

A typical example of a distributed directory might be where directory services are deployed at the department or site level, and then joined to form a directory service for a business unit or region. For an entire enterprise, directories could be joined across business units or sites. A large multi-national enterprise might further extend the model to span multiple countries or regions. The resulting distributed directory system is described as a Directory Information Tree (DIT), with clients able to query a local directory service at the department or local site level to get information about entries, not only from their own local environment, but from any other organizational unit within the extended enterprise.

A requirement for this model to be useful is that every individual directory application must be able to communicate securely with peer applications running on other hosts that reside on other networks. Mutual authentication between directory peers is essential to insuring trustworthy operation of a distributed directory service, as are all of the other security services necessary to assuring adequate access control. Kerberos is often used as the means for establishing secure, mutually authenticated communications sessions between individual directory applications. One reason that Kerberos is particularly effective in a distributed directory environment is that it provides a common authentication and session establishment facility that can be used for client access to directory services, for administrator access, and for peer interactions between directory applications.

Directories can also be extended between discrete enterprises using what is frequently referred to as a *federated model*. Building federated systems is a complex and challenging problem that involves many issues besides just directory integration. Furthermore, directory integration is usually constrained to only subsets of the directories in different enterprises. For example if an equipment manufacturer establishes federated relationships with its parts suppliers, it would not make sense to expose the entire directory of each enterprise to the others, and the relationships between the parts suppliers and each other are quite different from the relationships parts suppliers have with the manufacturer. Consequently, it might make sense for a parts supplier to have access to portions of the manufacturer's directory that relate to purchasing and inventory management, but not to other departments or business units.

Kerberos is commonly used to secure directory integration between federated parties, especially within Microsoft Active Directory federated scenarios. Specific trust relationships can be established between just the internal organizational units in one party that need to interact with counterparts in the other party using Kerberos mutual authentication and cross-realm mechanisms. In these situations, Kerberos initial authentication can utilize certificates and public key cryptography to further strengthen security across organizational boundaries.

Specific Implementations

Microsoft's Active Directory and Kerberos

In 1993, Microsoft decided to adopt Kerberos as a primary authentication mechanism within its products, and in 1999 released a fully integrated version of Kerberos with Windows Server 2000. In the same timeframe, Microsoft adopted the LDAP approach for its Active Directory services, which were also introduced with Windows Server 2000. Active Directory is more than a directory service, as it integrates the Kerberos KDC functions, DNS services, and related services, such as DHCP. It also supports standard LDAP directory interfaces. This is illustrated in Figure 7 below.

Microsoft has incorporated Kerberos into its Active Directory services as a component in support of a larger strategy. Some of their strategic objectives include:

- Provide comprehensive access control facilities that integrate all security mechanisms within a consistent framework for establishing and managing policy controls.
- Allow central administration of all access control facilities through a common set of administrative tools that automate configuration of specific underlying mechanisms (*e.g.*, Kerberos).
- Support multiple authentication options as a way to allow users and applications to choose the most appropriate authentication mechanisms, and also as a way to facilitate transition from older authentication mechanisms to newer ones.
- Insulate applications from underlying security technologies and even from being aware of what specific security technologies are actually in use.

In support of these strategic objectives, Active Directory has been positioned at the core of Microsoft's entire family of Windows Server products, and serves as the foundation for building distributed systems on Windows platforms, including the .NET framework. Active Directory is the "account database" used by all users and computers to manage resources and provide security controls in Windows Server environments. It supports a consistent account model and policy controls that can be applied across an entire, extended enterprise, and even between enterprises. The implication is that, if policies allow, a user in one part of an enterprise can access services or resources anywhere else in the enterprise.

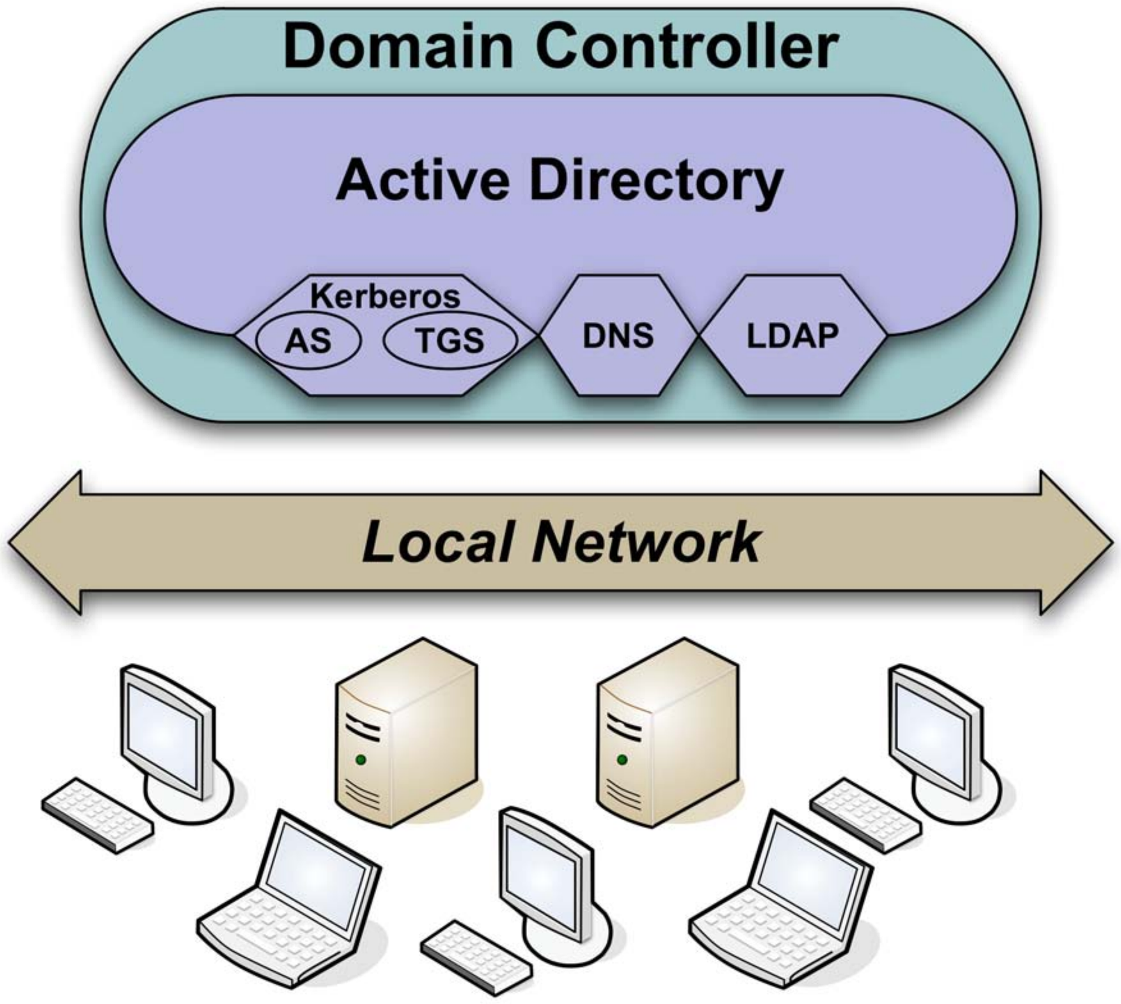


Figure 7: Active Directory is hosted on Domain Controllers, and provides service interfaces for Kerberos, DNS, LDAP, and others not shown, such as DHCP. Collectively, the workstations, servers, Domain Controller(s) and local network comprise a "Domain," with Active Directory managing Domain accounts, and integrating the Domain into an enterprise-wide directory "tree" or "forest."

The Kerberos KDC services are tightly integrated with Active Directory, and all Kerberos authentication requests are processed using information provided by Active Directory—*i.e.,* there is no distinct KDC database (a.k.a., KDB). Windows and non-Windows systems are able to interoperate with Active Directory KDC services using standard Kerberos protocols, including GSS-API Kerberos wire-level protocols. At the same time, other authentication mechanisms are supported, including the ability for applications to negotiate which authentication mechanism to use.

Relevant Microsoft Terminology and Naming Conventions

In Microsoft's terminology, an *account* is a *security principal,* which can be a user, a computer, or a *group.* Groups are lists of accounts—including other groups—that share common permissions to use information objects and resources. There is also the concept of an *Account Domain,* which is the *issuing authority* responsible for establishing accounts. A *Local* Domain is constrained to issuing accounts for a single computer, whereas a *Remote* Domain issues accounts for a set of networked computers and users. A Domain Controller (DC) is a Windows Server that manages accounts for a collection of computers and users on a shared network through Active Directory facilities. It is common to refer collectively to the computers and users served by a single DC as a Domain. Kerberos realms map directly to Domains, but a Domain is more than a realm.

Like Kerberos realm names, names for Domains are usually derived from DNS names.[20] However, since all names are administered through Active Directory, it is preferable to think in terms of names being established first in Active Directory, and then instantiated within DNS and Kerberos. Like other directories, Active Directory is deployed in a tree-like structure, where nodes on the tree are Domains.[21] The top level, or root, Domain is normally given the name corresponding to the enterprise's registered DNS domain.

To illustrate with an example, the company, Galactic Industries Inc. might have registered the domain name *gii.com,* and the root Domain would therefore be named *gii.com* as well.[22] Assuming that this company has operations in the United States, Europe, and Asia, three additional Domains could be defined as *us.gii.com, eu.gii.com,* and *asia.gii.com*. If there were operations on the east and west coasts in the US, then there could be two additional Domains named *east.us.gii.com* and *west.us.gii.com*. Once an Active Directory is configured for the west.us.gii.com Domain, a Kerberos realm will be established with the realm name, WEST.US.GII.COM, though in Microsoft's terminology, this is often referred to as the principal name *suffix.* The resulting "Domain Tree" is illustrated in Figure 8 below.

[20] Terminology collisions can lead to some confusion. A Microsoft *Domain* is conceptually similar to a Kerberos *realm,* though Microsoft rarely uses the term realm. However, a DNS *domain name* is not the same as the name of a Microsoft *Domain,* and is therefore referred to as a *DNS name* in Microsoft terminology. Furthermore, the *DN* acronym refers to an X.500 *Distinguished Name* and not a DNS domain name. Of course, all of these terms overlap to some degree, and use similar naming conventions. In particular, it is usually possible to directly convert a DNS name to both a realm name and Microsoft Domain name, which are often the same. A Distinguished Name incorporates the same elements as in DNS names, though usually with additional naming constructs.

[21] In Active Directory, the tree structure is referred to as a *Domain tree,* which is essentially equivalent to a Directory Information Tree or DIT in LDAP and X.500 terminology.

[22] Actually, Active Directory names follow the X.500 directory naming conventions, so this example name would be expressed as *dc=gii, dc=com*. Similarly, the west coast operation would be named *dc=west, dc=us, dc=gii, dc=com*.

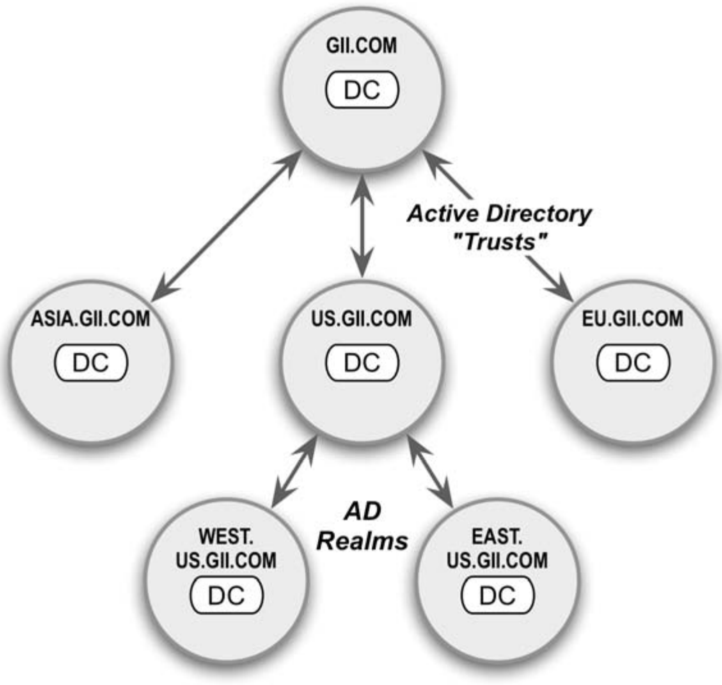


Figure 8—Active Directory "Domain Tree" example showing hierarchical naming convention, Domain Controllers (DCs), Active Directory realms, and "Trusts"

Given this description and example, it seems that Domain trees follow the traditional DNS naming conventions, and they usually do. However, this is not a requirement, and other naming conventions can be used. A full discussion of Active Directory naming strategies is beyond the scope of this document, however, it should be noted that Active Directory can also combine multiple Domain trees into what is termed a Domain *forest.* This is illustrated in Figure 9 below where two Domain trees are combined into a forest. A forest does not have a hierarchical root in the traditional sense, but uses a special AD trust relationship called a forest trust to integrate multiple disjoint trees into an enterprise-wide Active Directory. Forests are used where the trees are managed by autonomous administrative regimes, such as when a company has multiple independent subsidiaries.

When user accounts are created in Active Directory, they include User Principal Names, or UPNs, which are used for logon access and authentication via Kerberos on other mechanisms. The UPN is the user's *userid* followed by an '@' followed by the *UPN suffix.* Normally, the UPN suffix is the Domain name, but it can also be a shortened version configured in the Active Directory. Using the above example, a user, Bob, in the Galactic Industries' west coast operation could have the UPN of bob@west.us.gii.com, but if the west coast Domain administrator configures the suffix of "gii-west," then this user would have the UPN of bob@gii-west (see discussion below on Support for Kerberos Referrals for further information on how users can access services throughout an enterprise). Either way, the UPN is the user's Kerberos principal name. Note that Microsoft does not use role extensions for user principal names.

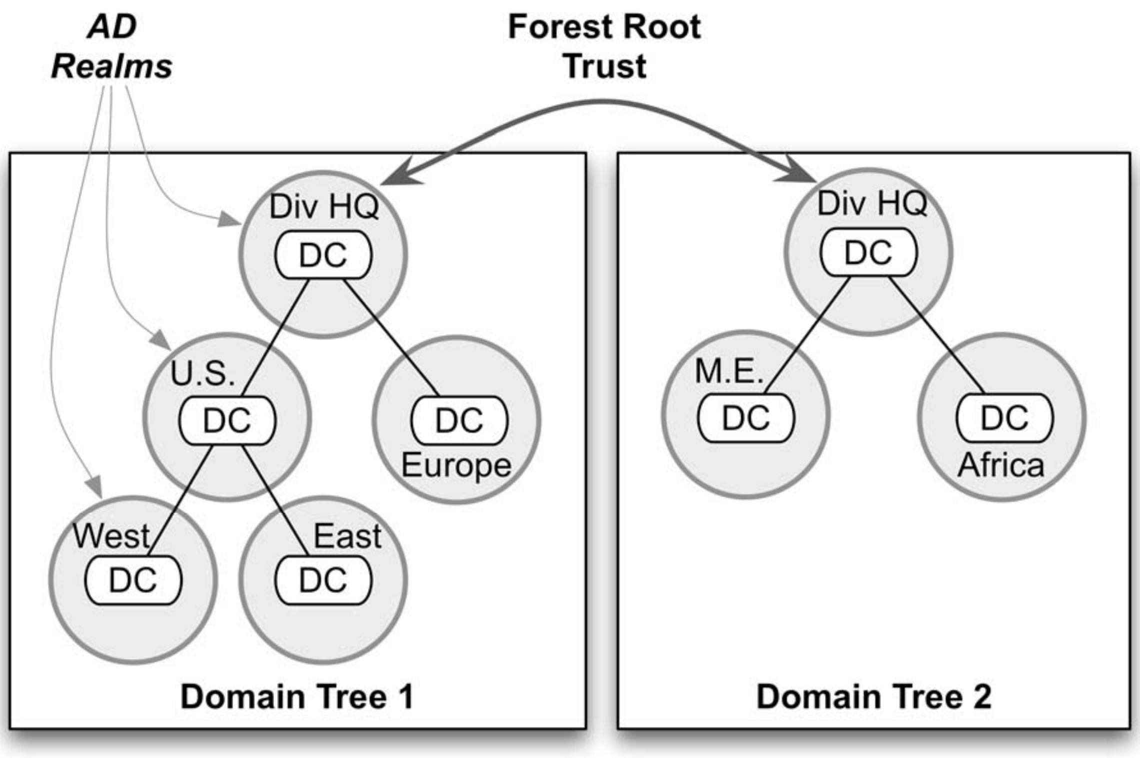


Figure 9—Example of a Domain Forest with two Domain Trees

Computer accounts are assigned Service Principal Names, or SPNs. In the default case, the SPN is constructed from the DNS name of the host, but there are various other options depending on how the local Active Directory Domain has been configured. In practice, Microsoft SPNs are created in a manner consistent with Kerberos principal names for services.

Since Active Directory is tightly integrated with DNS and KDC services, there is greater flexibility in assigning principal names and other names used within the Domain. Put another way, whatever names are defined within Active Directory are reflected into the DNS and KDC services, since both are subsets of the Active Directory service. This tight integration provides greater flexibility in configuring and using names within an enterprise. It may also lead to interoperability problems in mixed environments where other Kerberos implementations might be more constrained in how principals are named.

Microsoft's Kerberos Support

Microsoft's implementation of Kerberos is consistent with the protocols specified in RFC 4120, and the GSS-API Kerberos wire-level protocols defined in RFC 4121. However, to support its overall strategy, Microsoft has introduced several extensions in its Kerberos implementation, and there are some behavioral differences that can impact interoperability with non-Windows systems. Microsoft has published a document, MS-KILE, that describes their use of, and extensions to, the Kerberos protocols. Other Microsoft documents that help explain the overall role of Kerberos in Microsoft's products include the Security Overview MS-SECO and Protocol Overview MS-PROTO.

Basic Kerberos Support in Windows Server

A Microsoft Domain Controller (DC) supports the standard KDC functions specified in RFC 4120, including an Authentication Server for initial authentication and issuing TGTs, as well as a Ticket Granting Service that clients use to get tickets for connecting to services. Most Kerberos clients or services should be able to interact with the KDC services offered by a DC, with the *caveat* that tickets may be much larger than typical (because of inclusion of PACs, see below) and TCP may be required to interact with the KDC to reliably exchange larger tickets. There are also a few features not supported in Microsoft's implementation, though these are not widely used. The MS-KILE document details the capabilities of Microsoft's Kerberos support.

Microsoft also supports GSS-API with Kerberos in compliance with RFC 4121 for wire level protocol exchanges (ref. The GSS-API and Kerberos above). However, this is different than providing GSS-API support within the Windows platforms. For applications built on Windows platforms, Microsoft provides the Security Support Provider Interface (SSPI) as the native Windows API for authentication and other security operations. While it is possible for applications based on SSPI to interoperate with applications based on GSS-API, it is generally possible only for Kerberos authentication and secure sessions. In particular, SSPI supports various Microsoft proprietary capabilities that are not supported by GSS-API, whereas GSS-API supports capabilities not available with SSPI. Microsoft refers to applications that are compatible with wire level GSS-API as "GSS aware."

One area where SSPI provides unique capabilities is its support for *impersonation* (ref. MS-SECO section 5). Conceptually, impersonation allows a service to take on the full privileges (authorizations) of the client account, even when doing so elevates the privileges of the service's own account. This is different than delegation, where a service is allowed to subsequently authenticate to another service as the client. Delegation is supported by Kerberos, but impersonation involves authorizations, and must be supported by the platform OS. Most other operating systems do not allow any account to take on elevated privileges, and recent versions of Windows now restrict impersonation to only a limited set of service accounts.

Microsoft also supports SPNEGO (ref. RFC 4178, RFC 4559, and MS-SPNG) for negotiating which authentication mechanisms can be supported between a client and service. SPNEGO is a pseudo mechanism that works within the GSS framework to allow GSS aware applications to use SPNEGO instead of a specific authentication mechanism. SPNEGO then handles negotiation between the parties to select an actual mechanism that can be used for authentication, but within the restrictions of defined policies. For example, there might be a system-wide policy to require mutual authentication, in which case SPNEGO would prevent use of one-way authentication mechanisms—*e.g.*, NT LAN Manager authentication protocol, or NTLM.

One reason for introducing SPNEGO was to provide options for transitioning to new authentication mechanisms in the future, should better options emerge. Another reason, though, was to ease the transition away from older, less secure authentication mechanisms. Microsoft's implementation of SPNEGO essentially establishes Kerberos as the default authentication mechanism in a Windows Server environment and treats the older Windows NTLM authentication mechanism as a secondary option to be used only if Kerberos is not supported and policies allow.

Kerberos and the Microsoft Privilege Attribute Certificate (PAC)

The most significant extension to Kerberos introduced by Microsoft is the inclusion of the Privilege Attribute Certificate (PAC) in the authorization data field of Kerberos tickets as specified in MS-PAC. Use of the PAC is central to Microsoft's strategy to provide comprehensive access controls, since it allows authentication and authorization to be conducted in an integrated manner. The PAC delivers information about a client's authorization (privileges) to a service. By including it in the Kerberos ticket, the service receives both authentication of the client and authorization information in a trusted manner.[23] Although tickets are generated by KDCs, in the Microsoft context, the KDC is really an integral part of the Active Directory service, so it is really Active Directory that provides tickets with authentication and authorization information, and Kerberos is just the protocol for requesting and delivering tickets.

Conceptually, a PAC is a listing of all Active Directory group accounts that the client is a member of. Group policy rules can be used to determine authorization within the application. Since the PAC is provided on the fly, applications need only know their group policy rules; they do not have to maintain group membership lists or Access Control Lists.

While inclusion of PACs in tickets is consistent with the Kerberos specifications (ref. RFC 4120, section 5.2.6),[24] it has some important implications. First, a service that does not know how to interpret a PAC should ignore it if encountered in a ticket. It is also possible for clients to specify whether or not a PAC should be included when they issue a request for a ticket to use a service.

One consequence of including PACs is that Kerberos tickets are much larger than normal, which can lead to requirements to utilize TCP instead of UDP for carrying Kerberos messages. Microsoft's Kerberos support will default to using UPD, but frequently requires use of TCP to support larger messages. Some older non-Microsoft applications based on Kerberos did not handle Kerberos exchanges using TCP, or could not accept large tickets with PACs, which led to some interoperability problems. These problems are fading, though, as most applications today fully support Kerberos over TCP and properly handle larger ticket sizes.

Support for Kerberos Referrals

In most Windows Server environments, there will be multiple Domains, each with its own Domain Controller(s). Each Domain Controller provides Active Directory services within the Domain, and acts as a KDC for any computers or users operating within the Domain. Since a Domain is also a Kerberos realm, there tends to be multiple (many) Domains/realms within a single enterprise's Windows server environment. By contrast,

[23] Microsoft also introduces PACs into client-server dialogs where TLS (SSL) is used for authentication and secure communications.

[24] In the Kerberos protocol specifications, "AD" stands for "Authorization Data," and should not be confused with "Active Directory."

non-Windows Kerberos deployments are more likely to use a single Kerberos realm for an entire enterprise. The existence of many realms within an enterprise leads to two challenges:

1. Users would like to be able to use one name throughout an extended enterprise. Users do not want to change the name they use when they (and their account) move from one domain to another in the enterprise. In enterprises with complex organizations, users would rather use some principal like *user@some-corp.com* that is independent of what domain their account is located in.
2. Users (clients) must know the Domain name for each application service they want to access. This is because the client will need to get a TGT from the Domain KDC where the service resides.

In order to deal with these challenges, Microsoft requires full support for cross-realm operations as specified in RFC 4120 plus extended referrals as defined in the IETF-Draft Kerberos Referrals document. Microsoft supports extended referrals, as specified in sections 3.1.5.8 and 3.3.5.3.1 of MS-KILE.

A typical approach to dealing with the first of the two challenges above is to allow users to have alias names that are, for example, their email address. To illustrate using the example of Galactic Industries Inc, suppose that Alice is an employee based in the East Coast office with a Domain name of east.us.gii.com while Bob is another employee based in the West Coast office where the Domain name is west.us.gii.com. However, both Alice and Bob have regular email addresses (RFC 822) of the form alice@gii.com and bob@gii.com respectively. Instead of using their traditional Kerberos principal names when requesting an initial authentication, they use their email names. Since the request is to a KDC that is integrated with Active Directory, the user's name can be mapped via the directory service to their home Domain. Therefore, if Alice requests initial authentication from her east.us.gii.com Domain, then she will get back a TGT as normal. However, if Alice visits the West Coast office, then she might make an initial authentication request to the local west.us.gii.com Domain, she will get an error response indicating "wrong realm," with a better realm name provided, which might be east.us.gii.com, us.gii.com, or even just gii.com. Alice would then send another authentication request to the indicated realm KDC, and would get back either a TGT, or another "wrong realm" response pointing to the next realm along a path that will eventually get her to the correct realm. Normally, Active Directory will be able to look up the user's correct Domain/realm, and provide the actual realm in the first "wrong realm" response.

The second challenge is addressed by allowing clients to request a ticket for a service via their current local KDC, but without specifying the realm. Instead of using DNS name services to find the Domain/realm where the service actually resides, the KDC leverages Active Directory services to map the name of the service requested by the client to the correct realm, or to at least an intermediate realm that is "closer" to the correct destination realm. Instead of getting a service ticket back, the client gets a realm referral response that provides a TGT for the actual destination realm, or for an intermediate realm. The client then uses the TGT and indicated realm name to request a ticket from the corresponding KDC. If the client was referred to an intermediate realm, then they will have to repeat this process until they get to the final realm. One point worth noting is that, in Active Directory, services may have multiple alias names to make it easy for users in different Domains to find services.

Public Key Initial Authentication (PKINIT)

The Windows Server platform includes a comprehensive Public Key Infrastructure (PKI) that includes support for issuing public key certificates to all user and machine accounts, and life cycle support for account certificates. Consequently, Microsoft fully supports PKINIT (ref. RFC 4556) to allow clients to request initial authentication based on their certificate instead of using a pre-shared secret key (password). Microsoft's specifications for using PKINIT are provided in MS-PKCA.

This is an optional service, and requires that an enterprise set up PKI support for issuing and managing certificates. However, this is becoming increasingly common, and provides all of the benefits noted in PKINIT—Public Key Cryptography for Initial Authentication in Kerberos above. In addition, use of PKINIT is another way to address the challenge of having users roaming between Domains/realms. It may also be useful in cross-realm authentication, especially where no Active Directory trusts have been configured.

Active Directory also responds to requests for certificates for specific users and other types of accounts, which provides further options for authenticating clients, services, and even KDCs or establishing secure dialogs based on certificates.

Other Microsoft Kerberos Extensions

Microsoft makes use of other extensions to Kerberos. Two worth noting are the "Service for User" extensions as described in MS-SFU and a particular style of User-to-User Kerberos involving extensions to the GSS-API protocols described in IETF Draft-Swift-win2k-krb-user2user, a now obsolete IETF draft.

Support for Kerberos from other Vendors

Kerberos support provided by other vendors tends to be consistent, and normally follows the Unix model for implementations and deployments. Most implementations are based on either the MIT or Heimdal versions. However, as vendors integrate Kerberos more closely within their platforms, or with other services, innovations are being introduced that may set new directions for Kerberos use in the future.

A future update to this document may be developed to provide more comprehensive coverage of the role Kerberos plays in other vendor's platforms and services.