Oracle® Solaris 10 开发者安全性指南

文件号码 E38863–02
2013 年 9 月

ORACLE

130924@25097

目录

图

表

示例

前言

《Oracle Solaris 开发者安全性指南》介绍了用于 Oracle Solaris 操作系统中的安全功能的公共应用编程接口 (Application Programming Interface, API) 和服务提供者接口 (Service Provider Interface, SPI)。术语**服务提供者**指插入框架以提供安全服务的组件，如加密算法和安全协议。

注 – 此 Solaris 发行版支持使用以下 SPARC 和 x86 系列处理器体系结构的系统：UltraSPARC、SPARC64、AMD64、Pentium 和 Xeon EM64T。支持的系统可以在 `http://www.oracle.com/webfolder/technetwork/hcl/index.html` 上的《Solaris OS: Hardware Compatibility Lists》（《Oracle Solaris OS：硬件兼容性列表》）中找到。本文档列举了在不同类型的平台上进行实现时的所有差别。

在本文档中，这些与 x86 相关的术语表示以下含义：

- "x86" 泛指 64 位和 32 位的 x86 兼容产品系列。
- "x64" 指出了有关 AMD64 或 EM64T 系统的特定 64 位信息。
- “32 位 x86”指出了有关基于 x86 的系统的特定 32 位信息。

若想了解本发行版支持哪些系统，请参见《Solaris OS: Hardware Compatibility Lists》（《Oracle Solaris OS：硬件兼容性列表》）。

目标读者

《Oracle Solaris 开发者安全性指南》适用于编写以下类型程序的 C 语言开发者：

- 可以覆盖系统控制的特权应用程序
- 使用验证和相关安全服务的应用程序
- 需要保护网络通信的应用程序
- 使用加密服务的应用程序
- 提供或使用安全服务的库、共享目标文件和插件

注 – 有关 Oracle Solaris 功能的 java 语言等效对象，请参见 `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html`。

阅读本书之前

本指南的读者应该熟悉 C 语言编程。了解安全机制的基础知识是有帮助的，但不是必需的。使用本书无需网络编程方面的专业知识。

本书的结构

本书分为以下各章。

- 第 1 章，面向开发者的 Oracle Solaris 安全（概述）介绍了 Oracle Solaris 安全性。
- 第 2 章，开发特权应用程序介绍了如何编写使用进程特权的特权应用程序。
- 第 3 章，编写 PAM 应用程序和服务介绍了如何编写可插拔验证模块 (Pluggable Authentication Module, PAM)。
- 第 4 章，编写使用 GSS-API 的应用程序介绍了通用安全服务应用编程接口 (Generic Security Service Application Programming Interface, GSS-API)。
- 第 5 章，GSS-API 客户机示例和第 6 章，GSS-API 服务器示例两章分别概括介绍了 GSS-API 示例。
- 第 7 章，编写使用 SASL 的应用程序介绍了如何编写简单验证和安全层 (Simple Authentication and Security Layer, SASL) 应用程序。
- 第 8 章，Oracle Solaris 加密框架介绍分别概述了用户级别和内核级别的 Oracle Solaris 加密框架。
- 第 9 章，编写用户级加密应用程序和提供者介绍了如何为用户级别的 Solaris 加密框架编写使用者和提供者。
- 第 10 章，使用智能卡框架介绍了 Oracle Solaris 智能卡框架。
- 附录 A，基于 C 的 GSS-API 样例程序提供了 GSS-API 示例的完整代码列表。
- 附录 B，GSS-API 参考提供了 GSS-API 中各个项的参考信息。
- 附录 C，指定 OID 介绍了如何指定机制。当使用缺省机制之外的机制时需要使用此技术。
- 附录 D，SASL 示例的源代码提供了 SASL 示例的完整代码列表。
- 附录 E，SASL 参考表对主要 SASL 接口进行了简要说明。
- 附录 F，打包和签署加密提供者介绍了如何对加密提供者进行打包和签名。
- 词汇表提供了本手册中使用的安全术语定义。

相关文档

有关安全功能的其他信息，请参见以下资料：

- 《System Administration Guide: Security Services》从系统管理员的角度介绍了 Oracle Solaris 安全功能。
- 《应用程序包开发者指南》提供了关于设计和生成 Oracle Solaris 10 系统软件包的信息。
- Generic Security Service Application Program Interface 文档 (`ftp://ftp.isi.edu/in-notes/rfc2743.txt`) 概述了 GSS-API 的概念。
- Generic Security Service API Version 2: C-Bindings 文档 (`ftp://ftp.isi.edu/in-notes/rfc2744.txt`) 介绍了基于 C 语言的 GSS-API 的具体信息。
- 《ONC+ Developer's Guide》提供了有关远程过程调用的信息。

获取 Oracle 支持

Oracle 客户可以通过 My Oracle Support 获取电子支持。有关信息，请访问 `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info`，或访问 `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs`（如果您听力受损）。

印刷约定

下表介绍了本书中的印刷约定。

表 P–1 印刷约定

字体或符号	含义	示例
`AaBbCc123`	命令、文件和目录的名称；计算机屏幕输出	编辑 `.login` 文件。 使用 `ls -a` 列出所有文件。 `machine_name% you have mail.`
`AaBbCc123`	用户键入的内容，与计算机屏幕输出的显示不同	`machine_name%` **`su`** `Password:`
aabbcc123	要使用实名或值替换的命令行占位符	删除文件的命令为 `rm` *filename*。
AaBbCc123	保留未译的新词或术语以及要强调的词	这些称为 *Class* 选项。 **注意**：有些强调的项目在联机时以粗体显示。

表 P–1 印刷约定 （续）

字体或符号	含义	示例
新词术语强调	新词或术语以及要强调的词	**高速缓存**是存储在本地的副本。 请**勿**保存文件。
《书名》	书名	阅读《用户指南》的第 6 章。

命令中的 shell 提示符示例

下表显示了 Oracle Solaris OS 中包含的缺省 UNIX shell 系统提示符和超级用户提示符。请注意，在命令示例中显示的缺省系统提示符可能会有所不同，具体取决于 Oracle Solaris 发行版。

表 P–2 shell 提示符

shell	提示符
Bash shell、Korn shell 和 Bourne shell	`$`
Bash shell、Korn shell 和 Bourne shell 超级用户	`#`
C shell	`machine_name%`
C shell 超级用户	`machine_name#`

◆ ◆ ◆ 第 1 章

1 面向开发者的 Oracle Solaris 安全（概述）

本手册介绍了 Oracle Solaris 操作系统 (Oracle Solaris Operating System, Oracle Solaris OS) 中安全功能的公共应用编程接口 (Application Programming Interface, API) 和服务提供者接口 (Service Provider Interface, SPI)。

本章介绍以下两个方面的内容：

面向开发者的 Oracle Solaris 安全功能概述

本手册介绍了 Oracle Solaris 操作系统中安全功能的公共 API 和公共 SPI。有关如何从系统管理员的角度实现这些安全功能的信息，请参见《System Administration Guide: Security Services》中的第 1 章 “Security Services (Overview)”。

Oracle Solaris OS 提供了一个基于标准行业接口的网络安全体系结构。通过使用标准化的接口，应当无需随着安全技术的演变对使用或提供加密服务的应用程序进行任何修改。

系统安全

为了实现系统安全，Oracle Solaris OS 提供了进程特权。**进程特权**是基于超级用户的 UNIX 标准模型的替换模型，可用来授予对特权应用程序的访问权限。系统管理员需要为用户指定一组允许其访问特权应用程序的进程特权。用户不必成为超级用户即可使用特权应用程序。

特权使得系统管理员可以将有限的权限委托给用户而不是授予用户完全的 root 用户权限，以忽略系统安全。相应地，创建新特权应用程序的开发者应该测试特定的特权，而不是检查 UID 是否等于 0。请参见第 2 章，开发特权应用程序。

为了实现非常严格的系统安全，请考虑使用 Oracle Solaris 的 Trusted Extensions 功能。Trusted Extensions 功能允许系统管理员指定能够由特定用户访问的应用程序和文件。Trusted Extensions 功能不在本书的讨论范围之内。有关更多信息，请参见 http://www.oracle.com/us/sun/index.htm。

Oracle Solaris OS 提供了以下几个公共安全接口：

- **加密框架**－加密框架是 Oracle Solaris OS 中加密服务的主干。该框架提供标准的 PKCS #11 接口，这些接口适用于加密服务的使用者和提供者。该框架有两个部分：用户级应用程序的用户加密框架和内核级模块的内核加密框架。连接到框架的使用者无需了解有关所安装加密机制的专业知识。插入到框架中的提供者不包括针对不同类型使用者的特殊代码。

  加密框架的使用者包括安全协议、某些机制和需要执行加密的应用程序。框架的提供者可以是加密机制，也可以是硬件和软件插件中的其他机制。有关加密框架的概述，请参见第 8 章，Oracle Solaris 加密框架介绍。如需了解如何编写使用框架中服务的用户级应用程序，请参见第 9 章，编写用户级加密应用程序和提供者。

  加密框架的库是基于 RSA PKCS#11 v2.11 规范实现的。使用者和提供者都可以通过标准的 PKCS #11 调用来与用户级加密框架进行通信。

- **Java API**－Java 安全技术包含大量的 API、工具以及常用安全算法、机制和协议的实现。Java 安全 API 跨越多个领域，包括加密、公钥基础结构、安全通信、验证和访问控制。Java 安全技术为开发者提供了一个全面的用于编写应用程序的安全框架，还为用户或管理员提供了一组用于安全管理应用程序的工具。请参见 http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html。

网络安全体系结构

网络安全体系结构使用行业标准接口，如 PAM、GSS-API、SASL 和 RSA Security Inc. 的 PKCS#11 加密令牌接口 (Cryptographic Token Interface, Cryptoki)。通过使用标准化的协议和接口，开发者可以编写无需随安全技术演变而进行修改的使用者和提供者。

使用安全服务的应用程序、库或内核模块称作**使用者**。向使用者提供安全服务的应用程序称作**提供者**，或称作**插件**。用来实现加密操作的软件称作**机制**。机制不只是算法，还包括算法的应用方式。例如，将 DES 算法应用于验证功能是一种机制，而将 DES 应用于逐块加密的数据保护功能是另外一种机制。

使用网络安全体系结构，使用者的开发者将不再需要编写、维护和优化加密算法。可以将经过优化的加密机制作为体系结构的一部分提供。

Oracle Solaris OS 提供了以下几个公共安全接口：

- **PAM**－可插拔验证模块 (Pluggable Authentication Module)。PAM 模块主要用于系统对用户进行初始验证。用户可以通过 GUI、命令行或其他方法进入系统。除了验证服务以外，PAM 还提供用来管理帐户、会话和口令的服务。诸如 `login`、`rlogin` 和 `telnet` 之类的应用程序便是典型的 PAM 服务使用者。PAM SPI 服务由安全提供者（如 Kerberos v5 和 Smartcard）提供。请参见第 3 章，编写 PAM 应用程序和服务。
- **GSS-API**－通用安全服务应用编程接口 (Generic Security Service Application Program Interface)。GSS-API 在对等应用程序之间提供安全通信。GSS-API 还提供验证、完整性和保密性服务。在 Solaris 中实现的 GSS-API 可以使用 Kerberos v5、SPNEGO 和 Diffie-Hellman 加密。GSS-API 主要用于设计或实现安全的应用程序协议。GSS-API 可以向其他类型的协议（如 SASL）提供服务。GSS-API 通过 SASL 来向 LDAP 提供服务。

  GSS-API 通常由两个对等应用程序在最初建立了凭证之后通过网络进行通信时使用。GSS-API 由登录应用程序 NFS、ftp 在其他应用程序之间使用。

  有关 GSS-API 的介绍，请参见第 4 章，编写使用 GSS-API 的应用程序。第 5 章，GSS-API 客户机示例和第 6 章，GSS-API 服务器示例介绍了两个典型 GSS-API 应用程序的源代码。附录 A，基于 C 的 GSS-API 样例程序提供了 GSS-API 示例的代码列表。附录 B，GSS-API 参考提供了 GSS-API 的参考资料。附录 C，指定 OID 说明了如何指定非缺省机制。
- **SASL**－简单验证和安全层 (Simple Authentication and Security Layer)。SASL 主要由协议使用，用来进行验证并提供保密性和数据完整性服务。SASL 适用于基于网络的较高级别的应用程序，这些应用程序使用动态安全协商机制来保护会话。LDAP 是 SASL 的一个众所周知的使用者。SASL 与 GSS-API 相似，但 SASL 的级别比 GSS-API 略高一些。SASL 使用 GSS-API 服务。请参见第 7 章，编写使用 SASL 的应用程序。
- **智能卡**－智能卡终端上 IFD（Interface Device，接口设备）处理程序的开发者可以通过智能卡框架的终端接口来向使用者提供服务。第 10 章，使用智能卡框架中提供了有关这些接口的信息。

◆ ◆ ◆ 第 2 章

开发特权应用程序

本章说明如何开发特权应用程序。本章涵盖以下主题：

特权应用程序

特权应用程序是可以覆盖系统控制并检查特定用户 ID（user ID, UID）、组 ID（group ID, GID）、授权或特权的应用程序。这些访问控制元素是由系统管理员指定的。有关管理员如何使用这些访问控制元素的概要讨论，请参见《System Administration Guide: Security Services》中的第 8 章 “Using Roles and Privileges (Overview)”。

Oracle Solaris OS 为开发者提供了两个元素，使用这两个元素可以授予更为精细的特权：

- **特权－特权**是一项可授予给应用程序的独立的权限。被授予特权后，一个进程可以执行原本会被 Oracle Solaris OS 禁止的操作。例如，没有正确的文件权限，进程通常无法打开数据文件。`file_dac_read` 特权可向进程提供覆盖 UNIX 文件和读取文件的权限。在内核级别执行操作必须具备特权。
- **授权－授权**是可以执行某一类操作的权限，如果不具备授权，则安全策略会禁止这类操作。授权可指定给某个角色或某位用户。在用户级别执行操作必须具备授权。

授权和特权之间的区别在于其级别，在某个级别允许谁可以执行某项操作的策略。在内核级别执行操作必须具备特权。如果不具备正确的特权，某项进程无法在特权应用程序中执行特定的操作。授权在用户应用程序级别强制执行策略。可能需要具备某项特权才能访问某个特权应用程序或者才能在特权应用程序中执行特定的操作。

关于特权

特权是授予某项进程的独立权限，允许该进程执行某项不具备该特权就会被 Oracle Solaris OS 禁止的操作。大部分程序不使用特权，因为程序通常在系统安全策略的界限内执行操作。

特权由管理员进行指定。将根据程序的设计启用特权。登录或进入某配置文件 shell 时，管理员所具备的特权指定适用于在该 shell 中执行的所有命令。应用程序运行后，将以编程方式打开或关闭特权。如果使用 exec(1) 命令启动了一项新程序，则该程序可能可以使用所有可从父进程继承的特权。但是，该程序不能新增任何特权。

管理员如何指定特权

系统管理员负责将特权指定给各个命令。有关特权指定的更多信息，请参见《System Administration Guide: Security Services》中的“Privileges (Overview)”。

如何实现特权

每项进程具有 4 组特权，这些特权决定了某项进程是否可以使用某个特定的特权：

- 允许特权集合
- 可继承特权集合
- 有限特权集合
- 有效特权集合

允许特权集合

进程可能会使用到的所有特权都必须包含在允许特权集合中。相反，从来不会用到的特权应从该程序的允许特权集合中排除。

进程启动后，该进程会从其父进程中集成允许特权集合。通常在登录时或在新的配置文件 shell 中，所有特权都包含在初始的允许特权集合中。该集合中的特权是由管理员指定的。每项子进程都可以从该允许特权集合中删除特权，但是子进程无法向允许特权集合中添加其他特权。作为一个安全事项，您应该将程序从不使用的特权从允许特权集合中删除。这样，可以通过避免使用指定有误的或继承有误的特权来保护程序。

从允许特权集合中删除的特权也会从有效特权集合中自动删除。

可继承特权集合

在登录时或在新的配置文件 shell 中，可继承特权集合包含已由管理员指定的特权。调用 exec(1) 后，这些特权可能会传递到子进程中。进程应删除不必要的特权来避免将这些特权传递到子进程中。允许特权集合通常与可继承特权集合相同。但是，在某些情况下，从可继承特权集合中删除的特权还保留在允许特权集合中。

有限特权集合

通过有限特权集合，开发者可控制某个进程可以使用哪些特权或可将那些特权传递到子进程。子进程和子孙进程只能从有限特权集合中获取特权。执行 `setuid(0)` 函数时，有限特权集合负责确定允许应用程序使用的特权。在执行 `exec(1)` 时，将强制使用有限特权集合。将 `exec(1)` 删除后，从有限特权集合中删除特权的操作才会影响其他集合。

有效特权集合

进程实际可以使用的特权位于该进程的有效特权集合中。启动程序时，有效特权集合与允许特权集合是等同的。然后，有效特权集合既不是允许特权集合的子集也不与其等同。

将有效特权集合减少到基本特权是一个好的做法。基本特权集合包含核心特权，第 25 页中的 “特权类别”对其进行了说明。将程序不需要的所有特权完全删除。将所有不需要的基本特权切换为禁用，直到程序需要该进程。例如，`file_dac_read` 特权，可以使所有文件被读取。一个程序可以具备多个读取文件的例程。程序起初将所有特权关闭，然后会打开 `file_dac_read` 特权执行相应的读取例程。这样，开发者就可以确保程序不会执行 `file_dac_read` 特权执行错误的读取例程。此做法称为**特权包围**。第 29 页中的 “特权编码示例”对特权包围进行了演示。

超级用户与特权模型之间的兼容性

为适应原有应用程序，特权的实施要同时与超级用户和特权模型兼容。该适应过程是通过使用 PRIV_AWARE 标志实现的，该标志表明程序与特权兼容。PRIV_AWARE 标志由操作系统自动处理。

请考虑不能识别特权的子进程。该进程将不具有 PRIV_AWARE 标志。从父进程继承来的所有特权在允许特权集合和有效特权集合中都是可用的。如果该子进程将 UID 设置为 0，则该进程的有效特权集合和允许特权集合被限制为有效特权集合中的那些特权。子进程无法获取超级用户的完整权限。这样，感知特权进程的有限特权集合将对所有未感知特权的子进程限制超级用户特权。如果子进程对任何特权集合进行了修改，则该进程的 PRIV_AWARE 标志将设置为 true。

特权类别

特权是根据其作用域进行逻辑分组的，如下所示：

- 基本特权－执行最小操作所需的核心特权。基本特权包括如下特权：
  - `PRIV_FILE_LINK_ANY`－允许进程创建硬链接，指向由某 UID 所有（而非该进程有效 UID 所有）的文件。
  - `PRIV_PROC_EXEC`－允许进程调用 `execve()`。

  - PRIV_PROC_FORK－允许进程调用fork()、fork1()或vfork()。
  - PRIV_PROC_SESSION－允许进程发送信号或追踪其会话外的进程。
  - PRIV_PROC_INFO－允许进程检查特定进程（查询进程可以将信号发送给这些进程）以外的进程的状态。没有此特权，就无法检查在/proc中不能看到的进程。

  一般情况下，应该将基本特权作为集合来指定，而不是分别指定。此方法将确保指定中包括 Oracle Solaris OS 更新版本中发布的所有基本特权。另一方面，应该显式禁用程序不需要的已知特权。例如，如果程序不适用于 exec(1) 子进程，则应禁用 proc_exec 特权。
- 文件系统特权。
- 系统 V 进程间通信 (Interprocess Communication, IPC) 特权。
- 网络特权。
- 进程特权。
- 系统特权。

有关 Solaris 特权的完整列表及其说明，请参见 privileges(5)手册页。

注 – Solaris 提供区域功能，通过此功能，管理员可以为运行的应用程序设置隔离环境。请参见 zones(5)。由于一个区域中的进程无法监视或干扰该区域外系统中的其他活动，因此该进程的所有特权也限于该区域。但是，如果需要，可以将 PRIV_PROC_ZONE 特权应用于全局区域中需要特权才能在非全局区域中操作的进程。

使用特权进行编程

本节讨论使用特权的接口。要使用特权编程接口，需要以下头文件。

```
#include <priv.h>
```

本节还提供了说明如何在特权应用程序中使用特权接口的示例。

特权数据类型

以下是特权接口使用的主要数据类型：

- **特权类型**－单个特权由 priv_t 类型定义表示。可以使用下列方式使用特权 ID 字符串初始化 priv_t 类型的变量：

  ```
  priv_t priv_id = PRIV_FILE_DAC_WRITE;
  ```
- **特权集合类型**－特权集合由 priv_set_t 数据结构表示。请使用表 2–1 中列出的一个特权处理函数初始化 priv_set_t 类型的变量。

- **特权操作类型**－对文件或进程特权集合执行的操作类型由 `priv_op_t` 类型定义表示。并不是所有的操作对每种类型的特权集合都有效。有关详细信息，请阅读第 26 页中的 “使用特权进行编程”中的特权集合说明。

  特权操作可以具有下列各值：

  - `PRIV_ON`－在指定的文件或进程特权集合中启用在 `priv_set_t` 结构中声明的特权。
  - `PRIV_OFF`－在指定的文件或进程特权集合中禁用在 `priv_set_t` 结构中声明的特权。
  - `PRIV_SET`－将指定文件或进程特权集合中的特权设置为在 `priv_set_t` 结构中声明的特权。如果将该结构初始化为空，则 `PRIV_SET` 会将特权集合设置为 `none`。

特权接口

下表列出了使用特权的接口。表后面提供了一些主要特权接口的说明。

表 2–1 使用特权的接口

目的	函数	其他注释
获取和设置特权集合	setppriv(2)、getppriv(2)、priv_set(3C)、priv_ineffect(3C)	`setppriv()` 和 `getppriv()` 是系统调用。`priv_ineffect()` 和 `priv_set()` 是为方便而使用的包装函数。
识别和转换特权	priv_str_to_set(3C)、priv_set_to_str(3C)、priv_getbyname(3C)、priv_getbynum(3C)、priv_getsetbyname(3C)、priv_getsetbynum(3C)	这些函数将指定的特权或特权集合映射到名称或编号。
处理特权集合	priv_allocset(3C)、priv_freeset(3C)、priv_emptyset(3C)、priv_fillset(3C)、priv_isemptyset(3C)、priv_isfullset(3C)、priv_isequalset(3C)、priv_issubset(3C)、priv_intersect(3C)、priv_union(3C)、priv_inverse(3C)、priv_addset(3C)、priv_copyset(3C)、priv_delset(3C)、priv_ismember(3C)	这些函数与特权内存分配、测试和各种设置操作有关。

表 2–1 使用特权的接口 （续）

目的	函数	其他注释
获取和设置进程标志	getpflags(2)、setpflags(2)	PRIV_AWARE 进程标志指示进程是否了解特权或是否在超级用户模型下运行。PRIV_DEBUG 用于特权调试。
低级凭证处理	ucred_get(3C)	这些例程用于调试、底层系统调用和内核调用。

setppriv()：用于设置特权

用于设置特权的主要函数为 setppriv()，该函数具有以下语法：

```
int setppriv(priv_op_t op, priv_ptype_t which, \
const priv_set_t *set);
```

op 表示要执行的特权操作。*op* 参数具有以下三个可能值之一：

- PRIV_ON－将 *set* 变量指定的特权添加到 *which* 指定的特权集合类型中
- PRIV_OFF－从 *which* 指定的特权集合类型中删除 *set* 变量指定的特权
- PRIV_SET－使用 *set* 变量指定的特权替换 *which* 指定的特权集合类型中的特权

which 用于指定要更改的特权集合类型：

- PRIV_PERMITTED
- PRIV_EFFECTIVE
- PRIV_INHERITABLE
- PRIV_LIMIT

set 指定要在更改操作中使用的特权。

此外，还提供了便利函数：priv_set()。

用于映射特权的 priv_str_to_set()

这些函数便于使用其数值映射特权名称。priv_str_to_set() 是此系列中的典型函数。priv_str_to_set() 具有以下语法：

```
priv_set_t *priv_str_to_set(const char *buf, const char *set, \
const char **endptr);
```

priv_str_to_set() 采用 buf 中指定的特权名字符串。priv_str_to_set() 返回可以与四个特权集合之一组合的一组特权值。**endptr 可用于调试解析错误。请注意，可以在 buf 中包括以下关键字：

- "all" 表示所有已定义的特权。使用 "all,!*priv_name* ,..." 可以指定除指示特权以外的所有特权。

注 – 使用 "*priv_set* , "!*priv_name* , . . ." 的构造将从指定的特权集合中删除指定的特权。如果事先没有指定集合，请不要使用 "!*priv_name* , . . .", 因为如果没有从中删除特权的特权集合，该构造将从空的特权集合中删除指定的特权，并有效指示无特权。

- "none" 表示无特权。
- "basic" 表示执行登录标准 UNIX 操作系统的所有用户一般都可以执行的操作所需的特权集合。

特权编码示例

本节对使用超级用户模型和最小特权模型包围特权的方式进行比较。

包围在超级用户模型中的特权

以下示例说明如何在超级用户模型中包围特权操作。

示例 2–1 超级用户特权包围示例

```
/* Program start */
uid = getuid();
seteuid(uid);

/* Privilege bracketing */
seteuid(0);
/* Code requiring superuser capability */
...
/* End of code requiring superuser capability */
seteuid(uid);
...
/* Give up superuser ability permanently */
setreuid(uid,uid);
```

包围在最小特权模型中的特权

此示例说明如何在最小特权模型中包围特权操作。此示例使用以下假定：

- 该程序为 `setuid 0`。
- 由于 `setuid 0`，允许集合和有效集合最初设置为所有特权。
- 可继承集合最初设置为基本特权。
- 有限集合最初设置为所有特权。

代码后面是该示例的说明。

示例 2–2 最小特权包围示例

```
1  #include <priv.h>
2  /* Always use the basic set. The Basic set might grow in future
3   * releases and potentially retrict actions that are currently
4   * unrestricted */
5  priv_set_t *temp = priv_str_to_set("basic", ",", NULL);

6  /* PRIV_FILE_DAC_READ is needed in this example */
7  (void) priv_addset(temp, PRIV_FILE_DAC_READ);

8  /* PRIV_PROC_EXEC is no longer needed after program starts */
9  (void) priv_delset(temp, PRIV_PROC_EXEC);

10 /* Compute the set of privileges that are never needed */
11  priv_inverse(temp);

12  /* Remove the set of unneeded privs from Permitted (and by
13   * implication from Effective) */
14  (void) setppriv(PRIV_OFF, PRIV_PERMITTED, temp);

15  /* Remove unneeded priv set from Limit to be safe */
16  (void) setppriv(PRIV_OFF, PRIV_LIMIT, temp);

17  /* Done with temp */
18  priv_freeset(temp);

19  /* Now get rid of the euid that brought us extra privs */
20  (void) seteuid(getuid());

21  /* Toggle PRIV_FILE_DAC_READ off while it is unneeded */
22  priv_set(PRIV_OFF, PRIV_EFFECTIVE, PRIV_FILE_DAC_READ, NULL);

23  /* Toggle PRIV_FILE_DAC_READ on when special privilege is needed*/
24  priv_set(PRIV_ON, PRIV_EFFECTIVE, PRIV_FILE_DAC_READ, NULL);

25  fd = open("/some/retricted/file", O_RDONLY);

26  /* Toggle PRIV_FILE_DAC_READ off after it has been used */
27  priv_set(PRIV_OFF, PRIV_EFFECTIVE, PRIV_FILE_DAC_READ, NULL);

28  /* Remove PRIV_FILE_DAC_READ when it is no longer needed */
29  priv_set(PRIV_OFF, PRIV_ALLSETS, PRIV_FILE_DAC_READ, NULL);
```

该程序定义了名为 *temp* 的变量。*temp* 变量确定此程序不需要的特权集合。最初，在第 5 行将 *temp* 定义为包含基本特权集合。在第 7 行将 `file_dac_read` 特权添加到 *temp* 中。`proc_exec` 特权对 exec(1) 新进程（在此程序中不允许）是必需的。因此，在第 9 行中从 *temp* 中删除了 `proc_exec`，从而使 exec(1) 命令无法执行新进程。

此时，*temp* 仅包含该程序所需的那些特权，即基本集合加上 `file_dac_read`，再删除 `proc_exec`。在第 11 行中，`priv_inverse()` 函数将计算 *temp* 的逆向值，并将 *temp* 的值重置为补值。逆向值是从所有可能特权集合中删除指定集合（在本例中为 *temp*）所得的结果。作为第 11 行的结果，*temp* 现在包含该程序永不使用的那些特权。在第 14 行中，从允许集合中删除了 *temp* 定义的不需要的特权。此删除操作还从有效集合中有效

地删除了这些特权。在第 16 行中，从有限集合中删除了不需要的特权。在第 18 行中，因为不再需要 *temp*，因而释放了 *temp* 变量。

该程序可以识别特权。因此，该程序不使用 `setuid`，但可以将有效的 UID 重置为第 20 行中的用户的实际 UID。

在第 22 行中，通过从有效集合中删除 `file_dac_read` 特权禁用了该特权。在实际的程序中，需要 `file_dac_read` 特权之前，还将发生其他活动。在该样例程序中，读取第 25 行中的文件需要 `file_dac_read`。因此，在第 24 行中，启用了 `file_dac_read`。读取文件后，将再次从有效集合中立即删除 `file_dac_read`。读取所有文件后，通过在所有特权集合中禁用 `file_dac_read`，可永久地删除 `file_dac_read`。

下表说明了随着程序的运行如何转换特权集合。已指出了行号。

表 2–2 特权集合转换

步骤	*temp* 集合	允许特权集合	有效特权集合	有限特权集合
最初	–	所有	所有	所有
第 5 行－将 *temp* 设置为基本特权	基本	所有	所有	所有
第 7 行－将 `file_dac_read` 添加到 *temp* 中。	基本 + `file_dac_read`	所有	所有	所有
第 9 行－从 *temp* 中删除了 `proc_exec`。	基本 + `file_dac_read` – `proc_exec`	所有	所有	所有
第 11 行－*temp* 重置为逆向值。	所有－（基本 + `file_dac_read` – `proc_exec` ）	所有	所有	所有
第 14 行－在允许集合中禁用不需要的特权。	所有－（基本 + `file_dac_read` – `proc_exec` ）	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`	所有
第 16 行－在有限集合中禁用不需要的特权。	所有－（基本 + `file_dac_read` – `proc_exec` ）	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`
第 18 行－释放了 *temp* 文件。	–	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`
第 22 行－禁用 `file_dac_read` 直到需要时再启用。	–	基本 - `proc_exec`	基本 - `proc_exec`	基本 + `file_dac_read` – `proc_exec`
第 24 行－需要时启用 `file_dac_read`。	–	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`

表 2–2 特权集合转换 （续）

步骤	*temp* 集合	允许特权集合	有效特权集合	有限特权集合
第 27 行－执行 `read()` 操作后禁用 `file_dac_read`。	–	基本 - `proc_exec`	基本 - `proc_exec`	基本 + `file_dac_read` – `proc_exec`
第 29 行－不再需要 `file_dac_read` 时，从所有集合中删除该特权。	–	基本 - `proc_exec`	基本 - `proc_exec`	基本 - `proc_exec`

特权应用程序开发指南

本节为开发特权应用程序提供了以下建议：

- **使用隔离系统。**绝不应在生产系统中调试特权应用程序，因为不完整的特权应用程序可能会危及安全。
- **正确设置 ID。**调用进程的有效集合中需要具备 `proc_setid` 特权，以更改其用户 ID、组 ID 或补充组 ID。
- **使用特权包围。**应用程序使用特权时，将覆盖系统安全策略。应该包围特权任务并仔细进行控制，以确保不会破坏敏感信息。有关如何包围特权的信息，请参见第 29 页中的 “特权编码示例”。
- **使用基本特权启动。**基本特权是最基本的操作所必需的。特权应用程序应该使用基本集合启动。然后，该应用程序应该适当地删除和添加特权。以下是典型的启动方案。

  1. 守护进程以 root 身份启动。
  2. 守护进程启用基本特权集合。
  3. 守护进程禁用不必要的所有基本特权，例如 PRIV_FILE_LINK_ANY。
  4. 守护进程添加需要的所有其他特权，例如 PRIV_FILE_DAC_READ。
  5. 守护进程切换到守护进程 UID。

- **请避免 shell 转义序列。**shell 转义序列中的新进程可以使用父进程的可继承集合中的任何特权。因此，最终用户可以通过 shell 转义序列违反信任。例如，某些邮件应用程序可能会将 `!command` 行解释为命令并执行该行。因而，最终用户可以创建脚本，以充分利用所有邮件应用程序特权。建议删除不必要的 shell 转义序列。

关于授权

授权存储在 `/etc/security/auth_attr` 文件中。要创建使用授权的应用程序，请执行以下步骤：

1. 扫描 `/etc/security/auth_attr` 以查找一个或多个应用程序授权。
2. 请在程序开始时使用 `chkauthattr(3SECDB)` 函数检查所需的授权。`chkauthattr()` 函数将在以下位置按顺序搜索授权：

- policy.conf(4) 数据库中的 AUTHS_GRANTED 键－AUTHS_GRANTED 指示缺省情况下指定的授权。
- policy.conf(4) 数据库中的 PROFS_GRANTED 键－PROFS_GRANTED 指示缺省情况下指定的权限配置文件。chkauthattr () 将针对指定授权检查这些权限配置文件。
- user_attr(4) 数据库－此数据库存储为用户指定的安全属性。
- prof_attr(4) 数据库－此数据库存储为用户指定的权限配置文件。

chkauthattr() 在上述任何位置中都找不到权限授权，则将拒绝用户访问该程序。

3. 使管理员了解此应用程序需要哪些授权。您可以通过手册页或其他文档通知管理员。

示例 2–3 检查授权

以下代码段说明如何使用 chkauthattr() 函数检查用户的授权。在本例中，该程序将检查 solaris.job.admin 授权。如果用户具有此授权，则该用户可以读取或写入其他用户的文件。如果没有此授权，则用户只能对其拥有的文件执行操作。

```
/* Define override privileges */
priv_set_t *override_privs = priv_allocset();

/* Clear privilege set before adding privileges. */
priv_set(PRIV_OFF, PRIV_EFFECTIVE, PRIV_FILE_DAC_READ,
            priv_FILE_DAC_WRITE, NULL);

priv_addset(override_privs, PRIV_FILE_DAC_READ);
priv_addset(override_privs, PRIV_FILE_DAC_WRITE);

if (!chkauthattr("solaris.jobs.admin", username)) {
    /* turn off privileges */
    setppriv(PRIV_OFF, PRIV_EFFECTIVE, override_privs);
}
/* Authorized users continue to run with privileges */
/* Other users can read or write to their own files only */
```

◆ ◆ ◆ 第 3 章

3 编写 PAM 应用程序和服务

可插拔验证模块 (Pluggable Authentication Module, PAM) 为系统登录应用程序提供了验证和相关的安全服务。本章适用于希望通过 PAM 模块提供验证、帐户管理、会话管理和口令管理的系统登录应用程序开发者。此外，还为 PAM 服务模块的设计者提供了相关的信息。本章包含以下主题：

PAM 最初是由 Oracle Corporation 开发的。自此以后 PAM 规范提交给了 X/Open，即现在的 "Open Group"。PAM 规范可在《X/Open Single Sign-On Service (XSSO) - Pluggable Authentication》（Open Group，英国出版，ISBN：1-85912-144-6，1997 年 6 月）中找到。pam(3PAM)、libpam(3LIB) 和 pam_sm(3PAM) 手册页中介绍了 Oracle Solaris 的 PAM 实现。

PAM 框架介绍

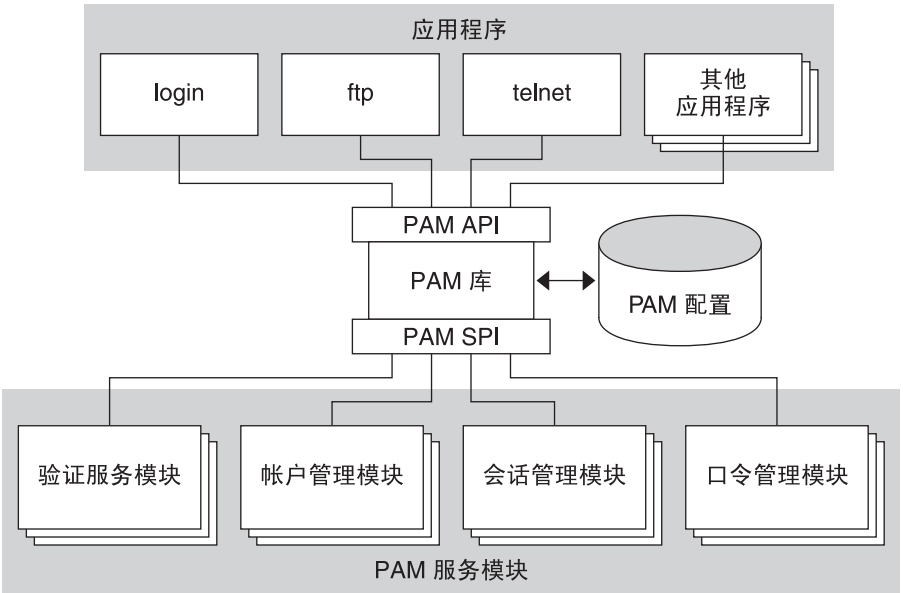
PAM 框架由四个部分组成：

- PAM 使用者
- PAM 库
- pam.conf(4) 配置文件
- PAM 服务模块，也称为提供者

该框架为与验证相关的活动提供了统一的执行方式。采用该方式，应用程序开发者不必了解策略的语义即可使用 PAM 服务。算法是集中提供的。可以独立于各个应用程序对算法进行修改。借助 PAM，管理员可以根据特定系统的需要调整验证过程，而不必更改任何应用程序。调整是通过 PAM 配置文件 pam.conf 来实现的。

下图说明了 PAM 体系结构。应用程序通过 PAM 应用编程接口 (Application Programming Interface, API) 与 PAM 库进行通信。PAM 模块通过 PAM 服务提供者接口 (Service Provider Interface, SPI) 与 PAM 库进行通信。通过这种方式，PAM 库可使应用程序和模块相互进行通信。

图 3–1 PAM 体系结构



PAM 服务模块

PAM 服务模块是一个共享库，用于为系统登录应用程序（如 `login`、`rlogin` 和 `telnet` 提供验证和其他安全服务。PAM 服务的四种类型是：

- **验证服务模块**－用于授予用户访问帐户或服务的权限。提供此服务的模块可以验证用户并设置用户凭证。
- **帐户管理模块**－用于确定当前用户的帐户是否有效。提供此服务的模块可以检查口令或帐户的失效期以及限时访问。
- **会话管理模块**－用于设置和终止登录会话。
- **口令管理模块**－用于强制实施口令强度规则并执行验证令牌更新。

PAM 模块可以实现其中的一项或多项服务。将简单模块用于明确定义的任务可以增加配置灵活性。因此，应该在不同的模块中实现 PAM 服务。然后，可以按照 `pam.conf(4)` 文件中定义的方式根据需要使用这些服务。

例如，Oracle Solaris OS 为系统管理员提供了用于配置站点口令策略的 pam_authtok_check(5) 模块。pam_authtok_check(5) 模块可以检查符合各种强度条件的建议口令。

有关 Oracle Solaris PAM 模块的完整列表，请参见手册页第 5 节：标准、环境与宏。PAM 模块的前缀为 pam_。

PAM 库

PAM 库 libpam(3LIB) 是 PAM 体系结构中的中心元素：

- libpam 可以导出 API pam(3PAM)。应用程序可以调用此 API 以执行验证、帐户管理、凭证建立、会话管理以及口令更改。
- libpam 可以导入主配置文件 pam.conf(4)。PAM 配置文件可指定每种可用服务的 PAM 模块要求。pam.conf 由系统管理员进行管理。
- libpam 可以导入 SPI pam_sm(3PAM)，而导出则由服务模块完成。

PAM 验证过程

以使用者如何使用 PAM 库进行用户验证为例，请考虑 login 如何验证用户：

1. login 应用程序通过调用 pam_start(3PAM) 并指定 login 服务来启动 PAM 会话。
2. 该应用程序将调用 pam_authenticate(3PAM)，它是 PAM 库 libpam(3LIB) 导出的 PAM API 的一部分。
3. 该库将在 pam.conf 文件中搜索 login 项。
4. 对于 pam.conf 中为 login 服务配置的每个模块，PAM 库将调用 pam_sm_authenticate(3PAM)。pam_sm_authenticate() 函数是 PAM SPI 的一部分。pam.conf 控制标志和每个调用的结果将确定是否允许用户访问系统。《System Administration Guide: Security Services》中的“PAM Configuration (Reference)”对此过程进行了详细介绍。

通过此方式，PAM 库可以将 PAM 应用程序与系统管理员已配置的 PAM 模块连接起来。

PAM 使用者的要求

PAM 使用者必须与 PAM 库 libpam 链接。应用程序可以使用模块提供的任何服务之前，必须通过调用 pam_start(3PAM) 初始化其 PAM 库的实例。调用 pam_start() 可初始化必须传递给所有后续 PAM 调用的句柄。应用程序完成使用 PAM 服务后，系统将调用 pam_end() 以清除 PAM 库已使用的任何数据。

PAM 应用程序与 PAM 模块之间的通信是通过**项**进行的。例如，以下各项有助于进行初始化：

- PAM_USER－当前验证的用户
- PAM_AUTHTOK－口令
- PAM_USER_PROMPT－用户名提示
- PAM_TTY－用户借此进行通信的终端
- PAM_RHOST－用户借此进入系统的远程主机
- PAM_REPOSITORY－对用户帐户系统信息库的任何限制
- PAM_RESOURCE－对资源的任何控制

有关可用项的完整列表，请参见 pam_set_item(3PAM)。应用程序可以通过 pam_set_item(3PAM) 对项进行设置。应用程序可以通过 pam_get_item(3PAM) 检索模块已设置的值。但是，应用程序不能检索 PAM_AUTHTOK 和 PAM_OLDAUTHTOK。无法设置 PAM_SERVICE 项。

注 – PAM 使用者必须拥有唯一的 PAM 服务名，该名称传递到 pam_start(3PAM)。

PAM 配置

PAM 配置文件 pam.conf(4) 用于为系统服务（如 login、rlogin、su 和 cron）配置 PAM 服务模块。系统管理员可以管理此文件。如果 pam.conf 中的项顺序有误，会导致无法预料的负面影响。例如，如果 pam.conf 配置错误，则会将多个用户锁定在外，这样必须采用单用户模式才能进行修复。有关 PAM 配置的信息，请参见《System Administration Guide: Security Services》中的“PAM Configuration (Reference)”。

编写使用 PAM 服务的应用程序

本节提供了使用多个 PAM 函数的应用程序样例。

简单 PAM 使用者示例

以下将以 PAM 使用者应用程序作为示例。该示例是一个基本的终端锁定应用程序，用于检验尝试访问终端的用户。此示例执行以下步骤：

1. 初始化 PAM 会话。

   PAM 会话通过调用 pam_start(3PAM) 函数来启动。调用任何其他 PAM 函数之前，PAM 使用者应用程序必须首先建立 PAM 会话。pam_start(3PAM) 函数采用以下参数：

   - plock－服务名，即应用程序的名称。PAM 框架使用服务名来确定配置文件 /etc/pam.conf 中适用的规则。服务名通常用于日志记录和错误报告。
   - pw->pw_name－用户名，即 PAM 框架所作用的用户的名称。

- &conv－对话函数 conv，用于提供 PAM 与用户或应用程序进行通信的通用方法。对话函数是必需的，因为 PAM 模块无法了解如何进行通信。通信可以采用 GUI、命令行、智能读卡器或其他设备等方式进行。有关更多信息，请参见第 42 页中的 “编写对话函数”。
- &pamh－PAM 句柄 pamh，即 PAM 框架用于存储有关当前操作信息的不透明句柄。成功调用 pam_start() 后将返回此句柄。

注 – 调用 PAM 接口的应用程序必须具有足够的特权才能执行任何所需的操作，如验证、口令更改、进程凭证处理或审计状态初始化。在该示例中，应用程序必须可以读取 /etc/shadow 才能验证本地用户的口令。

---

2. 验证用户。

   应用程序将调用 pam_authenticate(3PAM) 来验证当前用户。通常，系统会要求用户输入口令或其他验证令牌，具体取决于验证服务的类型。PAM 框架会调用 /etc/pam.conf 中为验证服务 auth 列出的模块。服务名 plock 用于确定要使用的 pam.conf 项。如果不存在与 plock 对应的项，则缺省情况下会使用 other 中的项。如果应用程序配置文件中明确禁止使用 NULL 口令，则应该传递 PAM_DISALLOW_NULL_AUTHTOK 标志。Solaris 应用程序将检查 /etc/default/login 中的 PASSREQ=YES 设置。

3. 检查帐户有效性。

   该示例使用 pam_acct_mgmt(3PAM) 函数检查已验证的用户帐户的有效性。在该示例中，pam_acct_mgmt() 用于检查口令是否到期。

   pam_acct_mgmt() 函数还会使用 PAM_DISALLOW_NULL_AUTHTOK 标志。如果 pam_acct_mgmt() 返回 PAM_NEW_AUTHTOK_REQD，则应调用 pam_chauthtok(3PAM) 以允许已验证的用户更改口令。

4. 如果系统发现口令已到期，则会强制用户更改口令。

   该示例使用循环调用 pam_chauthtok()，直到返回成功信息为止。如果用户成功更改其验证信息（通常为口令），则 pam_chauthtok() 函数将返回成功信息。在该示例中，循环将继续直到返回成功信息为止。通常，应用程序会设置终止前应尝试的最多次数。

5. 调用 pam_setcred(3PAM)。

   pam_setcred(3PAM) 函数用于建立、修改或删除用户凭证。pam_setcred() 通常在验证用户之后进行调用。调用是在检验帐户后和打开会话前进行的。将 pam_setcred() 函数与 PAM_ESTABLISH_CRED 标志结合使用可建立新的用户会话。如果该会话是对现有会话（如对于 lockscreen）的更新，则应调用带有 PAM_REFRESH_CRED 标志的 pam_setcred()。如果会话要更改凭证（如使用 su 或承担角色），则应调用带有 PAM_REINITIALIZE_CRED 标志的 pam_setcred()。

6. 关闭 PAM 会话。

   PAM 会话通过调用 pam_end(3PAM) 函数进行关闭。pam_end() 还将释放所有的 PAM 资源。

以下示例给出了 PAM 使用者应用程序样例的源代码。

示例 3–1 PAM 使用者应用程序样例

```
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <strings.h>
#include <signal.h>
#include <pwd.h>
#include <errno.h>
#include <security/pam_appl.h>

extern int pam_tty_conv(int num_msg, struct pam_message **msg,
         struct pam_response **response, void *appdata_ptr);

/* Disable keyboard interrupts (Ctrl-C, Ctrl-Z, Ctrl-\) */
static void
disable_kbd_signals(void)
{
    (void) signal(SIGINT, SIG_IGN);
    (void) signal(SIGTSTP, SIG_IGN);
    (void) signal(SIGQUIT, SIG_IGN);
}

/* Terminate current user session, i.e., logout */
static void
logout()
{
    pid_t pgroup = getpgrp();

    (void) signal(SIGTERM, SIG_IGN);
    (void) fprintf(stderr, "Sorry, your session can't be restored.\n");
    (void) fprintf(stderr, "Press return to terminate this session.\n");
    (void) getchar();
    (void) kill(-pgroup, SIGTERM);
    (void) sleep(2);
    (void) kill(-pgroup, SIGKILL);
    exit(-1);
}

int
/*ARGSUSED*/
main(int argc, char *argv)
{
    struct pam_conv conv = { pam_tty_conv, NULL };
    pam_handle_t *pamh;
    struct passwd *pw;
    int err;

    disable_kbd_signals();
    if ((pw = getpwuid(getuid())) == NULL) {
```

示例 3–1 PAM 使用者应用程序样例 （续）

```
        (void) fprintf(stderr, "plock: Can't get username: %s\n",
            strerror(errno));
        exit(1);
    }

    /* Initialize PAM framework */
    err = pam_start("plock", pw->pw_name, &conv, &pamh);
    if (err != PAM_SUCCESS) {
        (void) fprintf(stderr, "plock: pam_start failed: %s\n",
            pam_strerror(pamh, err));
        exit(1);
    }

    /* Authenticate user in order to unlock screen */
    do {
        (void) fprintf(stderr, "Terminal locked for %s. ", pw->pw_name);
        err = pam_authenticate(pamh, 0);
        if (err == PAM_USER_UNKNOWN) {
            logout();
        } else if (err != PAM_SUCCESS) {
            (void) fprintf(stderr, "Invalid password.\n");
        }
    } while (err != PAM_SUCCESS);

    /* Make sure account and password are still valid */
    switch (err = pam_acct_mgmt(pamh, 0)) {
    case PAM_SUCCESS:
        break;
    case PAM_USER_UNKNOWN:
    case PAM_ACCT_EXPIRED:
        /* User not allowed in anymore */
        logout();
        break;
    case PAM_NEW_AUTHTOK_REQD:
        /* The user's password has expired. Get a new one */
        do {
            err = pam_chauthtok(pamh, 0);
        } while (err == PAM_AUTHTOK_ERR);
        if (err != PAM_SUCCESS)
            logout();
        break;
    default:
        logout();
    }

if (pam_setcred(pamh, PAM_REFRESH_CRED) != PAM_SUCCESS){
    logout();
}

    (void) pam_end(pamh, 0);
    return(0);
    /*NOTREACHED*/
}
```

其他有用的 PAM 函数

前面的示例 3–1 是一个简单的应用程序，它仅说明了几个主要的 PAM 函数。本节将介绍一些其他有用的 PAM 函数。

成功验证用户后，将会调用 pam_open_session(3PAM) 函数打开新会话。

调用 pam_getenvlist(3PAM) 函数可以建立新环境。pam_getenvlist() 将返回要与现有环境合并的新环境。

编写对话函数

PAM 模块或应用程序可以采用多种方式与用户进行通信：命令行、对话框等。因此，与用户通信的 PAM 使用者的设计者需要编写**对话函数**。对话函数用于在用户与模块之间传递消息，而与通信方式无关。对话函数可从对话函数回调 pam_message 参数中的 *msg_style* 参数派生消息类型。请参见 pam_start(3PAM)。

开发者不应对 PAM 如何与用户进行通信做出假设。而应用程序应该与用户交换消息，直到操作完成为止。应用程序会显示与对话函数对应的消息字符串，但不进行解释或修改。一条单独的消息中可以包含多行、控制字符或额外的空格。请注意，服务模块负责本地化发送给对话函数的任何字符串。

下面提供了对话函数样例 pam_tty_conv()。pam_tty_conv() 采用以下参数：

- *num_msg* – 传递给函数的消息数。
- ***mess* – 指向缓冲区的指针，该缓冲区包含来自用户的消息。
- ***resp* – 指向缓冲区的指针，该缓冲区包含对用户所做的响应。
- **my_data* – 指向应用程序数据的指针。

函数样例从 stdin 获取用户输入。例程需要为响应缓冲区分配内存。可以设置最大值 PAM_MAX_NUM_MSG 以限制消息的数量。如果对话函数返回错误，则对话函数会负责清除并释放为响应分配的任何内存。此外，对话函数必须将响应指针设置为 NULL。请注意，应使用零填充方法来完成内存清除。对话函数的调用者负责释放返回给它的所有响应。要进行对话，该函数将循环处理来自用户应用程序的消息。有效的消息将写入 stdout，所有错误将写入 stderr。

示例 3–2 PAM 对话函数

```
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident    "@(#)pam_tty_conv.c    1.4    05/02/12 SMI"

#define    __EXTENSIONS__    /* to expose flockfile and friends in stdio.h */
#include <errno.h>
```

示例 3–2 PAM 对话函数 （续）

```
#include <libgen.h>
#include <malloc.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <stropts.h>
#include <unistd.h>
#include <termio.h>
#include <security/pam_appl.h>

static int ctl_c;    /* was the conversation interrupted? */

/* ARGSUSED 1 */
static void
interrupt(int x)
{
    ctl_c = 1;
}

/* getinput -- read user input from stdin abort on ^C
 *    Entry    noecho == TRUE, don't echo input.
 *    Exit    User's input.
 *        If interrupted, send SIGINT to caller for processing.
 */
static char *
getinput(int noecho)
{
    struct termio tty;
    unsigned short tty_flags;
    char input[PAM_MAX_RESP_SIZE];
    int c;
    int i = 0;
    void (*sig)(int);

    ctl_c = 0;
    sig = signal(SIGINT, interrupt);
    if (noecho) {
        (void) ioctl(fileno(stdin), TCGETA, &tty);
        tty_flags = tty.c_lflag;
        tty.c_lflag &= ~(ECHO | ECHOE | ECHOK | ECHONL);
        (void) ioctl(fileno(stdin), TCSETAF, &tty);
    }

    /* go to end, but don't overflow PAM_MAX_RESP_SIZE */
    flockfile(stdin);
    while (ctl_c == 0 &&
        (c = getchar_unlocked()) != '\n' &&
        c != '\r' &&
        c != EOF) {
        if (i < PAM_MAX_RESP_SIZE) {
            input[i++] = (char)c;
        }
    }
    funlockfile(stdin);
    input[i] = '\0';
```

示例 3–2 PAM 对话函数 （续）

```
    if (noecho) {
        tty.c_lflag = tty_flags;
        (void) ioctl(fileno(stdin), TCSETAW, &tty);
        (void) fputc('\n', stdout);
    }
    (void) signal(SIGINT, sig);
    if (ctl_c == 1)
        (void) kill(getpid(), SIGINT);

    return (strdup(input));
}

/* Service modules do not clean up responses if an error is returned.
 * Free responses here.
 */
static void
free_resp(int num_msg, struct pam_response *pr)
{
    int i;
    struct pam_response *r = pr;

    if (pr == NULL)
        return;

    for (i = 0; i < num_msg; i++, r++) {

        if (r->resp) {
            /* clear before freeing -- may be a password */
            bzero(r->resp, strlen(r->resp));
            free(r->resp);
            r->resp = NULL;
        }
    }
    free(pr);
}

/* ARGSUSED */
int
pam_tty_conv(int num_msg, struct pam_message **mess,
    struct pam_response **resp, void *my_data)
{
    struct pam_message *m = *mess;
    struct pam_response *r;
    int i;

    if (num_msg <= 0 || num_msg >= PAM_MAX_NUM_MSG) {
        (void) fprintf(stderr, "bad number of messages %d "
            "<= 0 || >= %d\n",
            num_msg, PAM_MAX_NUM_MSG);
        *resp = NULL;
        return (PAM_CONV_ERR);
    }
    if ((*resp = r = calloc(num_msg,
        sizeof (struct pam_response))) == NULL)
        return (PAM_BUF_ERR);
```

示例 3–2 PAM 对话函数 （续）

```
    /* Loop through messages */
    for (i = 0; i < num_msg; i++) {
        int echo_off;

        /* bad message from service module */
        if (m->msg == NULL) {
            (void) fprintf(stderr, "message[%d]: %d/NULL\n",
                i, m->msg_style);
            goto err;
        }

        /*
         * fix up final newline:
         *     removed for prompts
         *     added back for messages
         */
        if (m->msg[strlen(m->msg)] == '\n')
            m->msg[strlen(m->msg)] = '\0';

        r->resp = NULL;
        r->resp_retcode = 0;
        echo_off = 0;
        switch (m->msg_style) {

        case PAM_PROMPT_ECHO_OFF:
            echo_off = 1;
            /*FALLTHROUGH*/

        case PAM_PROMPT_ECHO_ON:
            (void) fputs(m->msg, stdout);

            r->resp = getinput(echo_off);
            break;

        case PAM_ERROR_MSG:
            (void) fputs(m->msg, stderr);
            (void) fputc('\n', stderr);
            break;

        case PAM_TEXT_INFO:
            (void) fputs(m->msg, stdout);
            (void) fputc('\n', stdout);
            break;

        default:
            (void) fprintf(stderr, "message[%d]: unknown type "
                "%d/val=\"%s\"\n",
                i, m->msg_style, m->msg);
            /* error, service module won't clean up */
            goto err;
        }
        if (errno == EINTR)
            goto err;

        /* next message/response */
        m++;
```

示例 3–2 PAM 对话函数 （续）

```
        r++;
    }
    return (PAM_SUCCESS);

err:
    free_resp(i, r);
    *resp = NULL;
    return (PAM_CONV_ERR);
}
```

编写提供 PAM 服务的模块

本节介绍如何编写 PAM 服务模块。

PAM 服务提供者要求

PAM 服务模块使用 pam_get_item(3PAM) 和 pam_set_item(3PAM) 与应用程序进行通信。要相互进行通信，服务模块需要使用 pam_get_data(3PAM) 和 pam_set_data(3PAM)。如果同一项目的服务模块需要交换数据，则应建立该项目的唯一数据名称。然后，服务模块即可通过 pam_get_data() 和 pam_set_data() 函数共享此数据。

服务模块必须返回以下三类 PAM 返回码之一：

- 如果模块在所请求的策略中做出了明确决定，则返回 PAM_SUCCESS。
- 如果模块未做出策略决定，则返回 PAM_IGNORE。
- 如果模块参与的决定导致失败，则返回 PAM_*error*。*error* 可以是常规错误代码或特定于服务模块类型的代码。错误不能是其他服务模块类型的错误代码。有关错误代码，请参见特定的 pam_sm_*module-type* 手册页。

如果服务模块执行多个函数，则应将这些函数分成单独的模块。使用此方法，系统管理员可对策略配置进行更为精细的控制。

应该为任何新的服务模块提供手册页。手册页应该包括以下各项：

- 模块接受的参数。
- 模块实现的所有函数。
- 标志对算法的影响。
- 任何所需的 PAM 项。
- 特定于此模块的错误返回信息。

服务模块必须支持 PAM_SILENT 标志，以防止显示消息。建议使用 *debug* 参数将调试信息记录到 syslog 中。请将 syslog(3C) 与 LOG_AUTH 和 LOG_DEBUG 结合使用来记录调试。其他消息应发送到具有 LOG_AUTH 和相应优先级的 syslog()。决不能使用 openlog(3C)、closelog(3C) 和 setlogmask(3C)，因为这些函数会干扰应用程序设置。

PAM 提供者服务模块样例

以下是一个 PAM 服务模块样例。此示例将查看用户是否是允许访问此服务的组中的成员。如果成功，则提供者随后将授予访问权限，如果失败则记录错误消息。此示例执行以下步骤：

1. 解析从 /etc/pam.conf 中的配置行传递到此模块的选项。

   此模块可接受 nowarn 和 debug 选项以及特定的选项 group。使用 group 选项可以配置模块，允许其访问除缺省情况下使用的 root 组以外的特定组。有关示例，请参见源代码中 DEFAULT_GROUP 的定义。例如，要允许属于组 staff 的用户访问 telnet(1)，用户可以使用以下行（位于 /etc/pam.conf 中的 telnet 栈中）：

   ```
   telnet  account  required  pam_members_only.so.1 group=staff
   ```

2. 获取用户名、服务名和主机名。

   用户名通过调用 pam_get_user(3PAM)（用于从 PAM 句柄中检索当前用户名）获取。如果未设置用户名，则会拒绝访问。可通过调用 pam_get_item(3PAM) 来获取服务名和主机名。

3. 验证要使用的信息。

   如果未设置用户名，则拒绝访问。如果未定义要使用的组，则拒绝访问。

4. 检验当前用户是否是允许访问此主机并且授予访问权限的特殊组的成员。

   如果该特殊组已定义但根本不包含任何成员，则将返回 PAM_IGNORE，表示此模块不参与任何帐户验证过程。该决策将留给栈中的其他模块。

5. 如果用户不是特殊组的成员，则会显示一条消息，通知用户访问被拒绝。

   记录消息以记录此事件。

以下示例给出了 PAM 提供者样例的源代码。

示例 3–3 PAM 服务模块样例

```
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <stdio.h>
#include <stdlib.h>
#include <grp.h>
#include <string.h>
#include <syslog.h>
#include <libintl.h>
#include <security/pam_appl.h>

/*
 * by default, only users who are a member of group "root" are allowed access
 */
#define    DEFAULT_GROUP "root"

static char *NOMSG =
```

示例 3–3 PAM服务模块样例 （续）

```
    "Sorry, you are not on the access list for this host - access denied.";

int
pam_sm_acct_mgmt(pam_handle_t * pamh, int flags, int argc, const char **argv)
{
    char *user = NULL;
    char *host = NULL;
    char *service = NULL;
    const char *allowed_grp = DEFAULT_GROUP;
    char grp_buf[4096];
    struct group grp;
    struct pam_conv *conversation;
    struct pam_message message;
    struct pam_message *pmessage = &message;
    struct pam_response *res = NULL;
    int i;
    int nowarn = 0;
    int debug = 0;

    /* Set flags to display warnings if in debug mode. */
    for (i = 0; i < argc; i++) {
        if (strcasecmp(argv[i], "nowarn") == 0)
            nowarn = 1;
        else if (strcasecmp(argv[i], "debug") == 0)
            debug = 1;
        else if (strncmp(argv[i], "group=", 6) == 0)
            allowed_grp = &argv[i][6];
    }
    if (flags & PAM_SILENT)
        nowarn = 1;

    /* Get user name,service name, and host name. */
    (void) pam_get_user(pamh, &user, NULL);
    (void) pam_get_item(pamh, PAM_SERVICE, (void **) &service);
    (void) pam_get_item(pamh, PAM_RHOST, (void **) &host);

    /* Deny access if user is NULL. */
    if (user == NULL) {
        syslog(LOG_AUTH|LOG_DEBUG,
            "%s: members_only: user not set", service);
        return (PAM_USER_UNKNOWN);
    }

    if (host == NULL)
        host = "unknown";

    /*
     * Deny access if vuser group is required and user is not in vuser
     * group
     */
    if (getgrnam_r(allowed_grp, &grp, grp_buf, sizeof (grp_buf)) == NULL) {
        syslog(LOG_NOTICE|LOG_AUTH,
            "%s: members_only: group \"%s\" not defined",
            service, allowed_grp);
        return (PAM_SYSTEM_ERR);
    }
```

示例 3–3 PAM 服务模块样例 （续）

```
    /* Ignore this module if group contains no members. */
    if (grp.gr_mem[0] == 0) {
        if (debug)
            syslog(LOG_AUTH|LOG_DEBUG,
                "%s: members_only: group %s empty: "
                "all users allowed.", service, grp.gr_name);
        return (PAM_IGNORE);
    }

    /* Check to see if user is in group. If so, return SUCCESS. */
    for (; grp.gr_mem[0]; grp.gr_mem++) {
        if (strcmp(grp.gr_mem[0], user) == 0) {
            if (debug)
                syslog(LOG_AUTH|LOG_DEBUG,
                    "%s: user %s is member of group %s. "
                    "Access allowed.",
                    service, user, grp.gr_name);
            return (PAM_SUCCESS);
        }
    }

    /*
     * User is not a member of the group.
     * Set message style to error and specify denial message.
     */
    message.msg_style = PAM_ERROR_MSG;
    message.msg = gettext(NOMSG);

    /* Use conversation function to display denial message to user. */
    (void) pam_get_item(pamh, PAM_CONV, (void **) &conversation);
    if (nowarn == 0 && conversation != NULL) {
        int err;
        err = conversation->conv(1, &pmessage, &res,
            conversation->appdata_ptr);
        if (debug && err != PAM_SUCCESS)
            syslog(LOG_AUTH|LOG_DEBUG,
                "%s: members_only: conversation returned "
                "error %d (%s).", service, err,
                pam_strerror(pamh, err));

        /* free response (if any) */
        if (res != NULL) {
            if (res->resp)
                free(res->resp);
            free(res);
        }
    }

    /* Report denial to system log and return error to caller. */
    syslog(LOG_NOTICE | LOG_AUTH, "%s: members_only: "
        "Connection for %s not allowed from %s", service, user, host);

    return (PAM_PERM_DENIED);
}
```

◆ ◆ ◆ 第 4 章

4 编写使用 GSS-API 的应用程序

通用安全服务应用编程接口 (Generic Security Service Application Programming Interface, GSS-API) 为应用程序提供了一种用于保护发送到对等应用程序的数据的方法。通常，连接是指从一台计算机上的客户机到另一台计算机上的服务器。本章介绍了有关以下主题的信息：

GSS-API 介绍

使用 GSS-API，程序员在编写应用程序时，可以应用通用的安全机制。开发者不必针对任何特定的平台、安全机制、保护类型或传输协议来定制安全实现。使用 GSS-API，程序员可忽略保护网络数据方面的细节。使用 GSS-API 编写的程序在网络安全方面具有更高的可移植性。这种可移植性是通用安全服务 API 的一个特点。

GSS-API 是一个以通用方式为调用者提供安全服务的框架。许多底层机制和技术（如 Kerberos v5 或公钥技术）都支持 GSS-API 框架，如下图所示。

图 4–1 GSS-API 层

从广义上讲，GSS-API 主要具有以下两种功能：

1. GSS-API 可创建一个安全**上下文**，应用程序可在该上下文中相互传递数据。上下文是指两个应用程序之间的信任状态。由于共享同一个上下文的应用程序可相互识别，因此可以允许在上下文存在期间进行数据传送。
2. GSS-API 可向要传送的数据应用一种或多种类型的保护，称为**安全服务**。第 53 页中的 “GSS-API 中的安全服务”介绍了安全服务。

此外，GSS-API 还可执行以下功能：

- 转换数据
- 检查错误
- 授予用户特权
- 显示信息
- 比较标识

GSS-API 中包括许多支持函数和便利函数。

使用 GSS-API 的应用程序的可移植性

GSS-API 为应用程序提供了以下几种类型的可移植性：

- **机制无关性**。GSS-API 提供了一个用于实现安全性的通用接口。通过指定缺省的安全机制，应用程序无需了解要应用的机制以及该机制的任何详细信息。
- **协议无关性**。GSS-API 与任何通信协议或协议套件均无关。例如，GSS-API 可用于使用套接字、RCP 或 TCP/IP 的应用程序。

  RPCSEC_GSS 是用于将 GSS-API 与 RPC 顺利集成的附加层。有关更多信息，请参见第 53 页中的 “使用 GSS-API 的远程过程调用”。
- **平台无关性**。GSS-API 与运行应用程序的操作系统的类型无关。

- **保护质量无关性**。保护质量 (Quality of Protection, QOP) 是指一种算法类型，用于加密数据或生成加密标记。通过 GSS-API，程序员可使用 GSS-API 所提供的缺省设置忽略 QOP。另一方面，应用程序可以根据需要指定 QOP。

GSS-API 中的安全服务

GSS-API 提供了三种类型的安全服务：

- **验证**－验证是 GSS-API 提供的基本安全性。验证是指对身份进行验证。如果用户通过了验证，则系统会假设其有权以该用户名进行操作。
- **完整性**－完整性是指对数据的有效性进行验证。即使数据来自有效用户，数据本身也可能会损坏或遭到破坏。完整性可确保消息与预期的一样完整（未增减任何内容）。GSS-API 提供的数据附带有一个名为消息完整性代码 (Message Integrity Code, MIC) 的加密标记。MIC 可用于证明收到的数据与发送者所传送的数据是否相同。
- **保密性**－保密性可确保拦截了消息的第三方难以阅读消息内容。验证和完整性机制都不会修改数据。如果数据由于某种原因而被拦截，则其他人可以阅读该数据。因此，可通过 GSS-API 对数据进行加密，前提是提供了支持加密的基础机制。这种数据加密称为保密性。

GSS-API 中的可用机制

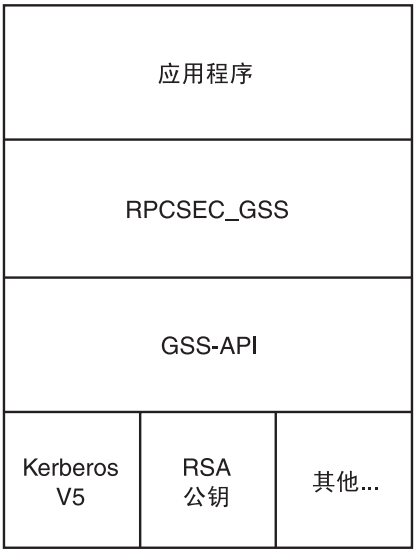
当前的 GSS-API 实现可使用以下机制：Kerberos v5、Diffie-Hellman 和 SPNEGO。有关 Kerberos 实现的更多信息，请参见《System Administration Guide: Security Services》中的第 21 章 “Introduction to the Kerberos Service”。如果任何系统上运行了能够识别 GSS-API 的程序，则应在该系统上安装和运行 Kerberos v5。

使用 GSS-API 的远程过程调用

针对网络应用程序使用 RPC（Remote Procedure Call，远程过程调用）协议的程序员可以使用 RPCSEC_GSS 来提供安全性。RPCSEC_GSS 是位于 GSS-API 上面的一个独立层。RPCSEC_GSS 可提供 GSS-API 的所有功能，但其方式是针对 RPC 进行了调整的。实际上，RPCSC_GSS 可用于向程序员隐藏 GSS-API 的许多方面，从而使 RPC 安全性具有更强的可访问性和可移植性。有关 RPCSEC_GSS 的更多信息，请参见《ONC+ Developer’s Guide》中的“Authentication Using RPCSEC_GSS”。

下图说明了 RPCSEC_GSS 层在应用程序和 GSS-API 之间的位置。

图 4–2 RPCSEC_GSS 和 GSS-API



GSS-API 的限制

虽然 GSS-API 使数据保护变得很简单，但是它回避某些任务的做法并不符合 GSS-API 的一般本性。因此，GSS-API **不**执行以下活动：

- 为用户或应用程序提供安全凭证。凭证必须由基础安全机制提供。GSS-API **确实**允许应用程序自动或显式获取凭证。
- 在应用程序之间传送数据。应用程序负责处理对等应用程序之间的**所有**数据传送，无论数据是与安全有关的数据还是明文数据。
- 区分不同类型的传送数据。例如，GSS-API 无法识别数据包是明文数据还是经过加密的数据。
- 指示由于异步错误而导致的状态。
- 缺省情况下保护多进程程序的不同进程之间已发送的信息。
- 分配要传递到 GSS-API 函数的字符串缓冲区。请参见第 55 页中的 “GSS-API 中的字符串和类似数据”。
- 取消分配 GSS-API 数据空间。此内存必须通过 `gss_release_buffer()` 和 `gss_delete_name()` 等函数显式取消分配。

GSS-API 的语言绑定

本文档目前仅介绍 GSS-API 的 C 语言绑定，即函数和数据类型。现已推出 Java 绑定版本的 GSS-API。Java GSS-API 包含通用安全服务应用编程接口 (Generic Security Service Application Program Interface, GSS-API) 的 Java 绑定，如 RFC 2853 中所定义。

有关 GSS-API 的更多参考信息

以下两个文档提供有关 GSS-API 的更多信息：

- "Generic Security Service Application Program Interface"（通用安全服务应用编程接口）文档 (`ftp://ftp.isi.edu/in-notes/rfc2743.txt`) 概述了 GSS-API 的概念。
- "Generic Security Service API Version 2: C-Bindings"（通用安全服务 API 版本 2：C 绑定）文档 (`ftp://ftp.isi.edu/in-notes/rfc2744.txt`) 介绍了基于 C 语言的 GSS-API 的具体信息。

GSS-API 的重要元素

本节介绍了以下 GSS-API 的重要概念：主体、GSS-API 数据类型、状态码和令牌。

- 第 55 页中的 “GSS-API 数据类型”
- 第 64 页中的 “GSS-API 状态码”
- 第 65 页中的 “GSS-API 令牌”

GSS-API 数据类型

以下几节介绍了主要的 GSS-API 数据类型。有关所有 GSS-API 数据类型的信息，请参见第 207 页中的 “GSS-API 数据类型和值”。

GSS-API 整数

由于 `int` 的长度因平台而异，因此 GSS-API 提供了以下整数数据类型：`OM_uint32`，此为 32 位无符号整数。

GSS-API 中的字符串和类似数据

由于 GSS-API 处理所有内部格式的数据，因此在将字符串传递到 GSS-API 函数之前必须将其转换为 GSS-API 格式。GSS-API 可处理具有 `gss_buffer_desc` 结构的字符串：

```
typedef struct gss_buffer_desc_struct {
     size_t     length;
     void       *value;
}  gss_buffer_desc *gss_buffer_t;
```

gss_buffer_t 是指向此类结构的指针。在将字符串传递到使用它们的函数之前，必须先将其放到 gss_buffer_desc 结构中。在以下示例中，通用的 GSS-API 函数在发送消息之前会先对此消息应用保护机制。

示例 4–1 在 GSS-API 中使用字符串

```
char *message_string;
gss_buffer_desc input_msg_buffer;

input_msg_buffer.value = message_string;
input_msg_buffer.length = strlen(input_msg_buffer.value) + 1;

gss_generic_function(arg1, &input_msg_buffer, arg2...);

gss_release_buffer(input_msg_buffer);
```

请注意，完成对 input_msg_buffer 的操作之后，必须通过 gss_release_buffer() 取消对 input_msg_buffer 的分配。

gss_buffer_desc 对象不只是用于字符串。例如，令牌可以作为 gss_buffer_desc 对象进行处理。有关更多信息，请参见第 65 页中的 “GSS-API 令牌”。

GSS-API 中的名称

名称是指主体。在网络安全术语中，**主体**是指用户、程序或计算机。主体可以是客户机或服务器。下面是主体的一些示例：

- 登录到另一台计算机的用户，如 *user@machine*
- 网络服务，如 *nfs@machine*
- 运行应用程序的计算机，如 *myHost@eng.company.com*

在 GSS-API 中，名称会存储为 gss_name_t 对象，该对象对于应用程序是不透明的。名称可以通过 gss_import_name() 函数从 gss_buffer_t 对象转换为 gss_name_t 形式。所导入的每个名称都有一个指示名称格式的相关**名称类型**。有关名称类型的更多信息，请参见第 62 页中的 “GSS-API OID”。有关有效名称类型的列表，请参见第 208 页中的 “名称类型”。

gss_import_name() 具有以下语法：

```
OM_uint32 gss_import_name (
       OM_uint32          *minor-status,
       const gss_buffer_t input-name-buffer,
       const gss_OID      input-name-type,
       gss_name_t         *output-name)
```

minor-status 基础机制返回的状态码。请参见第 64 页中的 “GSS-API 状态码”。

input-name-buffer 包含要导入的名称的 gss_buffer_desc 结构。应用程序必须显式分配此结构。请参见第 55 页中的 “GSS-API 中的字符串和类似数据”以及示例 4–2。应用程序不再使用为该参数分配的空间时，必须通过 gss_release_buffer() 取消分配该参数。

input-name-type 用于指定 *input-name-buffer* 的格式的 `gss_OID`。请参见第 63 页中的 “GSS-API 中的名称类型”。另外，第 208 页中的 “名称类型”中还包含一个有效名称类型表。

output-name 接收名称的 `gss_name_t` 结构。

示例 4–1 中所示的通用示例进行了小修改，说明如何使用 `gss_import_name()`。首先，将规则字符串插入到 `gss_buffer_desc` 结构中。然后，使用 `gss_import_name()` 将该字符串放到 `gss_name_t` 结构中。

示例 4–2 使用 `gss_import_name()`

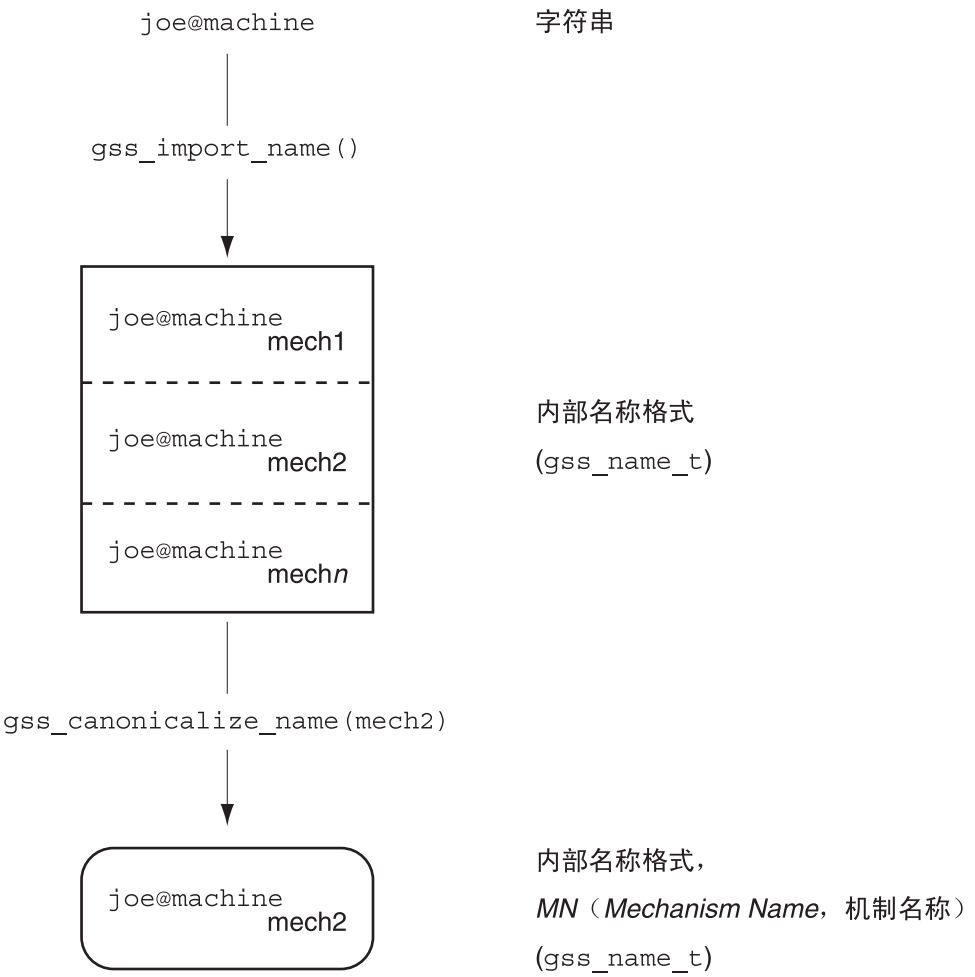
```
char *name_string;
gss_buffer_desc input_name_buffer;
gss_name_t      output_name_buffer;

input_name_buffer.value = name_string;
input_name_buffer.length = strlen(input_name_buffer.value) + 1;

gss_import_name(&minor_status, input_name_buffer,
                    GSS_C_NT_HOSTBASED_SERVICE, &output_name);

gss_release_buffer(input_name_buffer);
```

使用 `gss_display_name()` 可以将所导入的名称放回到 `gss_buffer_t` 对象中，以便能够以可读格式显示。但是，鉴于基础机制存储名称的方式，`gss_display_name()` 不能保证所得到的字符串与原来的字符串相同。GSS-API 包括若干个用于处理名称的其他函数。请参见第 201 页中的 “GSS-API 函数”。

`gss_name_t` 结构可以包含一个名称的多个版本。可以为 GSS-API 所支持的每个机制都生成一个版本。也即是说，*user@company* 的 `gss_name_t` 结构可能会包含该名称的两个版本：一个版本由 Kerberos v5 提供，另一个版本由其他机制提供。`gss_canonicalize_name()` 函数将内部名称和机制用作输入。`gss_canonicalize_name()` 可生成另一个内部名称，其中包含一个特定于该机制的名称版本。

此类特定于机制的名称称为**机制名称** (Mechanism Name, MN)。机制名称是指指定机制所生成的主体名称，而不是指机制的名称。下图对此流程进行了说明。

图 4–3 内部名称和机制名称



在 GSS-API 中比较名称

请考虑以下情况：服务器从客户机收到一个名称，并且需要在访问控制列表中查找该名称。**访问控制列表**（即 ACL）是指具有特定访问权限的主体的列表。下面是一种用于进行查找的方法：

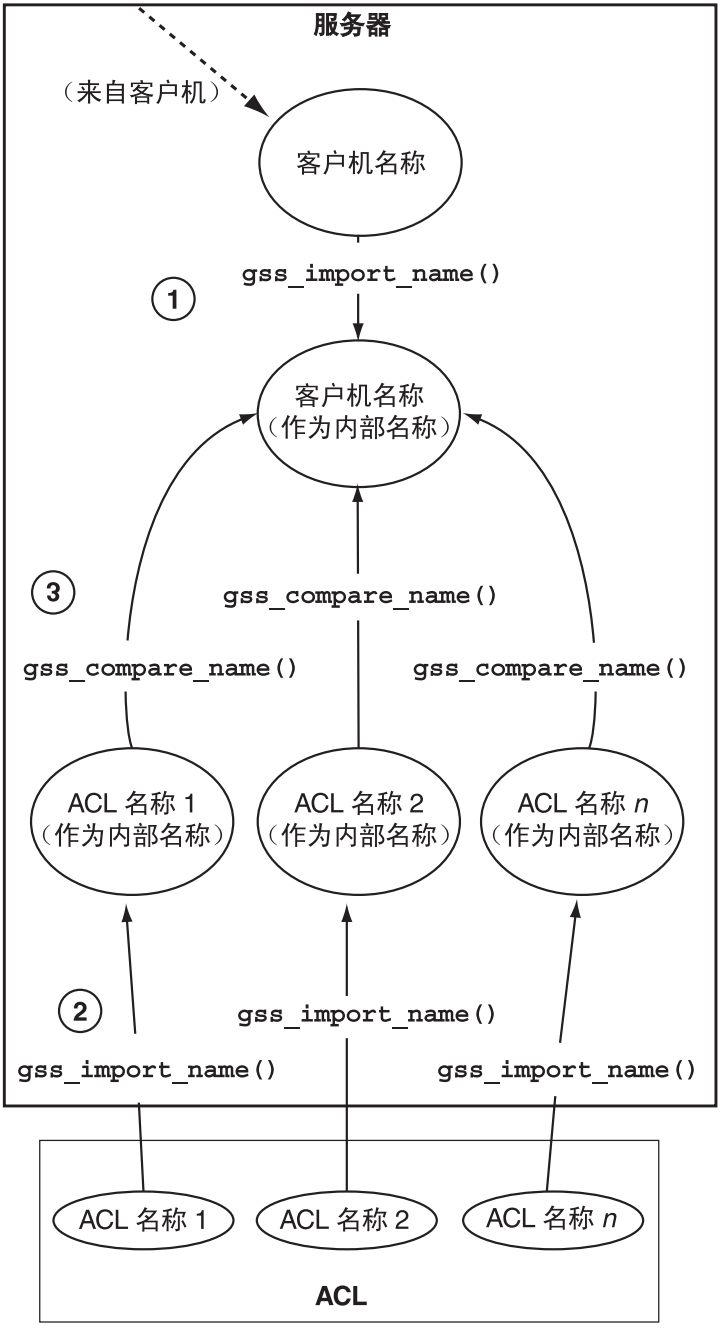
1. 使用 `gss_import_name()` 以 GSS-API 内部格式导入客户机名称（如果该名称尚未导入）。

在某些情况下，服务器将以内部格式接收名称，因此可不必执行该步骤。例如，服务器可能会查找客户机本身的名称。在启动上下文的过程中，客户机本身的名称以内部格式进行传递。

2. 使用 `gss_import_name()` 将每个名称导入到 ACL 中。
3. 使用 `gss_compare_name()` 将所导入的每个 ACL 名称与所导入的客户机名称进行比较。

此过程如下图所示。在本示例中，假设步骤 1 是必需的。

图 4–4 比较名称（慢速）


服务器
（来自客户机）
客户机名称
gss_import_name()
1
客户机名称
（作为内部名称）
3
gss_compare_name()
gss_compare_name()
gss_compare_name()
ACL 名称 1
（作为内部名称）
ACL 名称 2
（作为内部名称）
ACL 名称 n
（作为内部名称）
2
gss_import_name()
gss_import_name()
gss_import_name()
ACL 名称 1
ACL 名称 2
ACL 名称 n
ACL

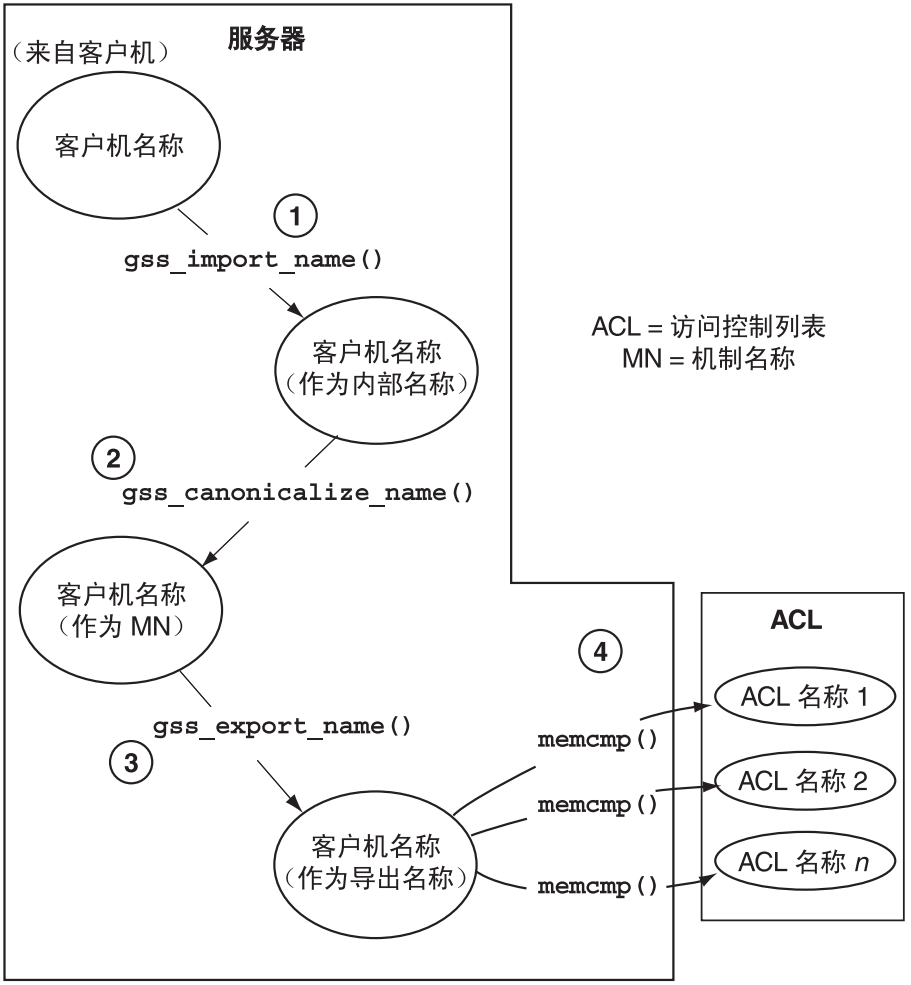
如果只有少数名称，则可以接受使用上述单独比较名称的方法。如果有大量名称，则使用 gss_canonicalize_name() 函数会更高效。此方法会执行以下步骤：

1. 使用 gss_import_name() 导入客户机的名称（如果该名称尚未导入）。

   和上述比较名称的方法一样，如果名称已经采用内部格式，则不必执行此步骤。
2. 使用 gss_canonicalize_name() 生成机制名称版本的客户机名称。
3. 使用 gss_export_name() 生成所导出的名称，这是连续字符串形式的客户机名称。
4. 使用 memcmp() 对所导出的客户机名称与 ACL 中的每个名称进行快速比较，该函数的开销较低。

此过程如下图所示。同样，再次假设服务器需要导入从客户机收到的名称。

图 4–5 比较名称（快速）



由于 `gss_export_name()` 需要使用机制名称 (Mechanism Name, MN)，因此**必须**首先针对客户机名称运行 `gss_canonicalize_name()`。

有关更多信息，请参见 `gss_export_name(3GSS)`、`gss_import_name(3GSS)` 和 `gss_canonicalize_name(3GSS)`。

GSS-API OID

对象标识符 (Object Identifier, OID) 用于存储以下几种数据：

- 安全机制
- QOP－保护质量的值

- 名称类型

OID 存储在 GSS-API `gss_OID_desc` 结构中。GSS-API 提供了指向 `gss_OID` 结构的指针，如以下示例中所示。

示例 4–3 OID 结构

```
typedef struct gss_OID_desc_struct {
        OM_uint32   length;
        void        *elements;
     } gss_OID_desc, *gss_OID;
```

并且，`gss_OID_set_desc` 结构中可能会包含一个或多个 OID。

示例 4–4 OID 集合结构

```
typedef struct gss_OID_set_desc_struct {
        size_t    count;
        gss_OID   elements;
     } gss_OID_set_desc, *gss_OID_set;
```

注意 – 应用程序不应尝试使用 `free()` 取消分配 OID。

GSS-API 中的机制和 QOP

尽管 GSS-API 允许应用程序选择基础安全机制，但是应用程序还是应当尽可能使用 GSS-API 已选择的缺省机制。同样，尽管 GSS-API 允许应用程序指定用于保护数据的保护质量级别，但还是应当尽可能使用缺省 QOP。可以通过向需要使用机制或 QOP 作为参数的函数传递值 `GSS_C_NULL_OID` 来表示接受缺省机制。

注意 – 显式指定安全机制或 QOP 会背离使用 GSS-API 的初衷。类似的特定选择会限制应用程序的可移植性。其他 GSS-API 实现可能无法以预期方式支持 QOP 或机制。尽管如此，附录 C，指定 OID 中还是简要讨论了如何确定可以使用的机制和 QOP 以及如何进行选择。

GSS-API 中的名称类型

除了 QOP 和安全机制以外，OID 还用于指示名称类型，从而指明关联名称的格式。例如，`gss_import_name()` 函数（用于将主体的名称从字符串转换为 `gss_name_t` 类型）将需要转换的字符串的格式用作一个参数。例如，如果名称类型为 `GSS_C_NT_HOSTBASED_SERVICE`，则该函数便会知道所输入的名称是采用 *service@host* 形式。如果名称类型为 `GSS_C_NT_EXPORT_NAME`，则该函数需要使用 GSS-API 导出的名

称。应用程序可以使用 gss_inquire_names_for_mech() 函数来确定指定机制可用的名称类型。第 208 页中的 “名称类型”中提供了 GSS-API 所使用的名称类型的列表。

GSS-API 状态码

所有 GSS-API 函数都会返回两种类型的代码，其中提供了有关该函数执行成败与否的信息。这两种类型的状态码都以 OM_uint32 值的形式返回。这两种类型的返回码如下所示：

- **主状态码**－指示以下错误的代码：
  - 常规的 GSS-API 例程错误，如为例程提供无效机制
  - 特定于指定 GSS-API 语言绑定的调用错误，如函数参数无法读取、无法写入或格式有误
  - 以上两种类型的错误

  此外，主状态码还可以提供有关例程状态的补充信息。例如，代码可能会指示操作尚未完成或者令牌未按顺序发送。如果未出现任何错误，例程将返回主状态值 GSS_S_COMPLETE。

  主状态码按如下方式返回：

```
OM_uint32 major_status ;    /* status returned by GSS-API */

major_status = gss_generic_function(arg1, arg2 ...);
```

  主状态返回码的处理方式与任何其他 OM_uint32 的处理方式相同。例如，请考虑以下代码：

```
OM_uint32 maj_stat;

maj_sta = gss_generic_function(arg1, arg2 ...);

if (maj_stat == GSS_CREDENTIALS_EXPIRED)
        <do something...>
```

  主状态码可以通过 GSS_ROUTINE_ERROR()、GSS_CALLING_ERROR() 和 GSS_SUPPLEMENTARY_INFO() 宏进行处理。第 203 页中的 “GSS-API 状态码”介绍了如何读取主状态码并提供了 GSS-API 状态码的列表。

- **次状态码**－基础机制返回的代码。这些代码未在本手册中具体说明。

  每个 GSS-API 函数都将次状态码的 OM_uint32 类型作为第一个参数。该函数返回到调用函数时，次状态码将存储到 OM_uint32 参数中。请考虑以下代码：

```
OM_uint32 *minor_status ;    /* status returned by mech */

major_status = gss_generic_function(&minor_status, arg1, arg2 ...);
```

即使返回了致命的主状态码错误，*minor_status* 参数也始终由 GSS-API 例程设置。请注意，大多数其他输出参数均可以保持未设置状态。但是，需要将应当返回特定指针的输出参数设置为 NULL，这些指针指向该例程已分配的存储空间。NULL 表示并未实际分配存储空间。正如在 `gss_buffer_desc` 结构中一样，与此类指针相关联的任何长度字段均设置为零。在此类情况下，应用程序无需释放这些缓冲区。

GSS-API 令牌

GSS-API 中的基本“流通”(currency) 单位是**令牌** (token)。使用 GSS-API 的应用程序可以借助令牌来相互通信。使用令牌可以交换数据并创建安全布局。令牌声明为 `gss_buffer_t` 数据类型。令牌对于应用程序是不透明的。

令牌的类型包括**上下文级别的令牌**和**每消息令牌**两种。上下文级别的令牌主要在建立（即启动和接受）上下文时使用。另外，还可以在以后通过传递上下文级别的令牌来管理上下文。

每消息令牌在建立了上下文之后使用。每消息令牌用于为数据提供保护服务。例如，请考虑一个希望将消息发送到另一个应用程序的应用程序。该应用程序可能会使用 GSS-API 来生成随该消息传递的加密标识符。该标识符可存储在令牌中。

可以考虑按如下方式针对消息使用每消息令牌。**消息**是应用程序发送到其对等应用程序的一段数据。例如，`ls` 命令可以是发送到 `ftp` 服务器的消息。**每消息令牌**是 GSS-API 为该消息生成的对象。每消息令牌可以是加密标记，也可以是消息的加密形式。请注意，最后一个示例不太准确。加密消息仍是消息，而不是令牌。令牌**仅**是指 GSS-API 生成的信息。但是，在非正式情况下，**消息**和**每消息**令牌通常可以互换使用。

应用程序负责进行以下活动：

1. 收发令牌。开发者通常需要编写一般性的读写函数来执行这些操作。`send_token()` 和 `recv_token()` 函数在第 195 页中的“各种 GSS-API 样例函数”中进行介绍。
2. 区分不同类型的令牌并相应地处理这些令牌。

   由于令牌对于应用程序来说是不透明的，因此应用程序不会对不同的令牌加以区分。如果不知道令牌的内容，则应用程序必须能够区分令牌的类型才能将令牌传递到相应的 GSS-API 函数。应用程序可以通过以下方法来区分令牌类型：

   - 按状态。通过程序的控制流程。例如，等待接受上下文的应用程序可能会假设收到的任何令牌都与所建立的上下文有关。对等应用程序应当等到上下文完全建立之后才发送消息令牌（即数据）。建立上下文之后，正在等待接受上下文的应用程序会假设新令牌即是消息令牌。这是处理令牌的相当常见的方法。本书中的样例程序会使用此方法。
   - 按标志。例如，如果某个应用程序包含用于向对等应用程序发送令牌的函数，则该应用程序可以引入一个标志，用于指示令牌类型。请考虑以下代码：

```
gss_buffer_t token;     /* declare the token */
OM_uint32 token_flag       /* flag for describing the type of token */
```

<get token from a GSS-API function>

```
token_flag = MIC_TOKEN;     /* specify what kind of token it is */
send_a_token(&token, token_flag);
```

接收应用程序可包含用于检查 *token_flag* 参数的接收函数，例如 `get_a_token()`。

- 通过显式使用标记。应用程序可以使用**元令牌**。元令牌是一种用户定义的结构，其中包含从 GSS-API 函数收到的令牌。元令牌包括用户定义的字段，这些字段指示如何使用 GSS-API 提供的令牌。

GSS-API 中的进程间令牌

GSS-API 允许在多进程应用程序中的进程之间传递安全上下文。通常，应用程序已经接受了客户机的上下文。然后，应用程序将在其进程之间共享该上下文。有关多进程应用程序的信息，请参见第 74 页中的 “在 GSS-API 中导出和导入上下文”。

`gss_export_context()` 函数可用于创建进程间令牌。使用此令牌中包含的信息，另一个进程可以重建该上下文。应用程序负责在进程之间传递进程间令牌。这与应用程序负责将令牌传递到其他应用程序情况相似。

进程间令牌可能包含密钥或其他敏感信息。并非所有的 GSS-API 实现都以加密方式保护进程间令牌。因此，在进行交换之前，应用程序必须保护进程间令牌。这种保护可能涉及到使用 `gss_wrap()` 对令牌进行加密（如果加密机制可用）。

注 – 请勿假设进程间令牌可以在不同的 GSS-API 实现之间传送。

开发使用 GSS-API 的应用程序

本节说明如何使用 GSS-API 来实现安全的数据交换。本节将重点介绍那些对于使用 GSS-API 至关重要的函数。有关更多信息，请参见附录 B，GSS-API 参考，其中包含所有 GSS-API 函数、状态码和数据类型的列表。要查找有关任何 GSS-API 函数的更多信息，请检查相应的手册页。

本手册中的示例遵循一个简单的模型。客户机应用程序将数据直接发送到远程服务器。无需通过传输协议层（如 RPC）进行中间调用。

GSS-API 的一般用法

下面是使用 GSS-API 的一般步骤：

1. 每个应用程序（无论是发送者还是接受者）都显式获取凭证，除非已经自动获取凭证。
2. 发送者启动一个安全上下文。接受者接受该上下文。
3. 发送者向要传送的数据应用安全保护机制。发送者会对消息进行加密或者使用标识标记对数据进行标记。发送者随后将传送受保护的消息。

注 – 发送者可以选择不应用安全保护机制，在这种情况下，消息仅具有缺省的 GSS-API 安全服务，即验证。

---

4. 接受者根据需要对消息进行解密，并在适当的情况下对消息进行验证。
5. （可选）接受者将标识标记返回到发送者进行确认。
6. 这两个应用程序都会销毁共享的安全上下文。如有必要，分配功能还可以取消分配其余任何 GSS-API 数据。

注意 – 调用应用程序负责释放已经分配的所有数据空间。

使用 GSS-API 的应用程序需要**包含**文件 `gssapi.h`。

在 GSS-API 中使用凭证

凭证是指向主体名称提供应用程序声明证明的数据结构。应用程序使用凭证来建立其全局标识。此外，凭证还可用于确认实体的特权。

GSS-API 不提供凭证。凭证由那些以 GSS-API 为基础的安全机制在调用 GSS-API 函数之前创建。在许多情况下，用户会在登录时接收凭证。

指定的 GSS-API 凭证对于单个主体有效。单个凭证可以包含该主体的多个元素，每个元素由不同的机制创建。如果某个凭证是在包含多个安全机制的计算机上获取的，则该凭证在传输到包含其中部分机制的计算机上时有效。GSS-API 通过 `gss_cred_id_t` 结构访问凭证。此结构称为**凭证句柄**。凭证对于应用程序是不透明的。因此，应用程序无需知道指定凭证的具体信息。

凭证采用以下三种形式：

- `GSS_C_INITIATE`－标识仅启动安全上下文的应用程序
- `GSS_C_ACCEPT`－标识仅接受安全上下文的应用程序
- `GSS_C_BOTH`－标识可启动或接受安全上下文的应用程序

在 GSS-API 中获取凭证

服务器和客户机都必须获取各自的凭证，然后才能建立安全上下文。应用程序可以在凭证到期之前重用它，在失效之后必须重新获取凭证。客户机使用的凭证与服务器使用的凭证可以具有不同的生命周期。

基于 GSS-API 的应用程序可以通过以下两种方法获取凭证：

- 使用 `gss_acquire_cred()` 或 `gss_add_cred()` 函数
- 在建立上下文时指定值 `GSS_C_NO_CREDENTIAL`，该值表示使用缺省凭证

大多数情况下，只有上下文接受器（即服务器）才能调用 `gss_acquire_cred()`。上下文启动器（即客户机）通常在登录时接收凭证。因此，客户机通常会指定缺省凭证。服务器也可以跳过 `gss_acquire_cred()` 而使用其缺省凭证。

客户机的凭证用于向其他进程证明该客户机的身份。服务器在获取凭证后即可接受安全上下文。因此，如果某个客户机向服务器发出 `ftp` 请求，则表明该客户机可能已经在登录时获取了凭证。客户机尝试启动上下文时，GSS-API 会自动检索凭证。但是，服务器程序可以显式获取所请求服务 (`ftp`) 的凭证。

如果 `gss_acquire_cred()` 成功完成，将返回 `GSS_S_COMPLETE`。如果不能返回有效的凭证，将返回 `GSS_S_NO_CRED`。有关其他错误代码，请参见 `gss_acquire_cred(3GSS)` 手册页。有关示例，请参见第 6 章，GSS-API 服务器示例中的第 104 页中的 “获取凭证”。

`gss_add_cred()` 与 `gss_acquire_cred()` 类似。但是，`gss_add_cred()` 允许应用程序使用现有的凭证来创建新句柄或者添加新的凭证元素。如果将 `GSS_C_NO_CREDENTIAL` 指定为现有的凭证，则 `gss_add_cred()` 会根据缺省行为创建新凭证。有关更多信息，请参见 `gss_add_cred(3GSS)` 手册页。

在 GSS-API 中使用上下文

在提供安全性方面，GSS-API 最重要的两个任务就是创建安全上下文和保护数据。应用程序获取必要的凭证之后，必须建立安全上下文。建立上下文时会涉及到两个应用程序，一个应用程序（通常是客户机）用于启动上下文，另一个应用程序（通常是服务器）用于接受上下文。允许在对等应用程序之间使用多个上下文。

相互通信的应用程序通过交换验证令牌来建立共同的安全上下文。安全上下文是一对 GSS-API 数据结构，其中包含要在两个应用程序之间共享的信息。这些信息从安全的角度描述了每个应用程序的状态。安全上下文是保护数据所必需的。

在 GSS-API 中启动上下文

`gss_init_sec_context()` 函数用于启动应用程序和远程服务器之间的安全上下文。如果成功，该函数将返回一个**上下文句柄**以建立上下文，还将返回一个要发送到接受器的上下文级别的令牌。调用 `gss_init_sec_context()` 之前，客户机应当执行以下任务：

1. 使用 gss_acquire_cred() 获取凭证（如有必要）。通常，客户机会在登录时接收凭证。gss_acquire_cred() 只能从正在运行的操作系统中检索初始凭证。
2. 使用 gss_import_name() 以 GSS-API 内部格式导入服务器的名称。有关名称和 gss_import_name() 的更多信息，请参见第 56 页中的 “GSS-API 中的名称”。

调用 gss_init_sec_context() 时，客户机通常会传递以下参数值：

- GSS_C_NO_CREDENTIAL（用于 *cred_handle* 参数），用于表示缺省凭证
- GSS_C_NULL_OID（用于 *mech_type* 参数），用于表示缺省机制
- GSS_C_NO_CONTEXT（用于 *context_handle* 参数），用于表示初始的空上下文。由于 gss_init_sec_context() 通常会循环调用，因此后续的调用会传递以前的调用所返回的上下文句柄
- GSS_C_NO_BUFFER（用于 *input_token* 参数），用于表示最初为空的令牌。或者，应用程序可以传递一个指向 gss_buffer_desc 对象的指针，该对象的长度字段已经设置为零
- 使用 gss_import_name() 以 GSS-API 内部格式导入的服务器名称。

应用程序不一定要使用这些缺省值。此外，客户机还可以使用 *req_flags* 参数指定其他安全参数的要求。下面描述了完整的 gss_init_sec_context() 参数集。

上下文接受器可能需要多次握手才能建立上下文。也就是说，接受器会要求启动器先发送多段上下文信息，然后再建立完整的上下文。因此，为了实现可移植性，应始终在检查上下文是否已完全建立的循环过程中启动上下文。

如果上下文不完整，则 gss_init_sec_context() 将返回一个主状态码 GSS_C_CONTINUE_NEEDED。因此，循环应当使用 gss_init_sec_context() 的返回值来测试是否继续执行启动循环操作。

客户机将上下文信息以**输出令牌**形式传递到服务器，输出令牌通过 gss_init_sec_context() 返回。客户机可接收从服务器返回的**输入令牌**形式的信息。以后调用 gss_init_sec_context() 时，输入令牌可以作为参数传递。但是，如果所收到的输入令牌的长度为零，则表明服务器不再需要其他输出令牌。

因此，除了检查 gss_init_sec_context() 的返回状态以外，该循环还应当检查输入令牌的长度。如果长度是非零值，则需要向服务器发送其他令牌。开始循环之前，应将输入令牌的长度初始化为零。请将输入令牌设置为 GSS_C_NO_BUFFER 或者将结构的长度字段设置为零值。

下面的伪代码举例说明如何从客户端建立上下文：

```
context = GSS_C_NO_CONTEXT
input token = GSS_C_NO_BUFFER

do
```

```
call gss_init_sec_context(credential, context, name, input token,
                          output token, other args...)

if (there's an output token to send to the acceptor)
    send the output token to the acceptor
    release the output token

if (the context is not complete)
    receive an input token from the acceptor

if (there's a GSS-API error)
    delete the context

until the context is complete
```

错误检查范围越大，实际循环将越完整。有关此类上下文启动循环的实际示例，请参见第 91 页中的 “建立与服务器的安全上下文”。此外，`gss_init_sec_context(3GSS)` 手册页还提供了比较特殊的示例。

通常，在上下文未完全建立时返回的参数值即是那些会在上下文完整建立时返回的值。有关更多信息，请参见 `gss_init_sec_context`(3GSS) 手册页。

如果 `gss_init_sec_context()` 成功完成，将返回 `GSS_S_COMPLETE`。如果对等应用程序需要上下文建立令牌，将返回 `GSS_S_CONTINUE_NEEDED`。如果出现错误，则将返回 `gss_init_sec_context`(3GSS) 手册页中所示的错误代码。

如果上下文启动失败，则客户机会断开与服务器的连接。

在 GSS-API 中接受上下文

建立上下文的另一方面是接受上下文，这可通过 `gss_accept_sec_context()` 函数来完成。通常情况下，服务器接受客户机使用 `gss_init_sec_context()` 已启动的上下文。

`gss_accept_sec_context()` 的主要输入是来自启动器的输入令牌。启动器会返回一个上下文句柄，以及一个要返回到启动器的输出令牌。但是，服务器应当首先获取客户机所请求服务的凭证，然后才能调用 `gss_accept_sec_context()`。服务器使用 `gss_acquire_cred()` 函数获取这些凭证。或者，服务器可以指定在调用 `gss_accept_sec_context()` 时的缺省凭证（即 `GSS_C_NO_CREDENTIAL`），从而无需显式获取凭证。

调用 `gss_accept_sec_context()` 时，服务器可以按如下方式设置以下参数：

- *cred_handle*－`gss_acquire_cred()` 返回的凭证句柄。或者，可以使用 `GSS_C_NO_CREDENTIAL` 来表示缺省凭证。

- *context_handle*－GSS_C_NO_CONTEXT 表示最初为空的上下文。由于 gss_init_sec_context() 通常会循环调用，因此后续的调用会传递以前的调用所返回的上下文句柄。
- *input_token*－从客户机接收的上下文令牌。

以下几段将介绍完整的 gss_accept_sec_context() 参数集。

建立安全上下文可能需要几次握手。启动器和接受器通常需要先发送多段上下文信息，然后才能建立完整的上下文。因此，为了实现可移植性，应始终在检查上下文是否已完全建立的循环过程中接受上下文。如果上下文尚未建立，则 gss_accept_sec_context() 将返回一个主状态码 GSS_C_CONTINUE_NEEDED。因此，循环应当使用 gss_accept_sec_context() 返回的值来测试是否继续执行接受循环操作。

上下文接受器将上下文信息以输出令牌形式返回到启动器，输出令牌通过 gss_accept_sec_context() 返回。以后，接受器可以从启动器接收输入令牌形式的其他信息。以后调用 gss_accept_sec_context() 时，输入令牌将作为参数传递。gss_accept_sec_context() 不再向启动器发送令牌时，将返回一个长度为零的输出令牌。除了检查返回状态 gss_accept_sec_context() 以外，循环还应当检查输出令牌的长度，查看是否必须发送其他令牌。开始循环之前，应将输出令牌的长度初始化为零。请将输出令牌设置为 GSS_C_NO_BUFFER 或者将结构的长度字段设置为零值。

下面的伪代码举例说明如何从服务器端建立上下文：

```
context = GSS_C_NO_CONTEXT
output token = GSS_C_NO_BUFFER

do

   receive an input token from the initiator

   call gss_accept_sec_context(context, cred handle, input token,
                               output token, other args...)

   if (there's an output token to send to the initiator)
      send the output token to the initiator
      release the output token

   if (there's a GSS-API error)
      delete the context

until the context is complete
```

错误检查范围越大，实际循环将越完整。有关此类上下文接受循环的实际示例，请参见第 91 页中的 “建立与服务器的安全上下文”。此外，gss_accept_sec_context(3GSS) 手册页还提供了一个示例。

同样，GSS-API 不会收发令牌。令牌必须由应用程序进行处理。有关令牌传送函数的示例可以在第 195 页中的 “各种 GSS-API 样例函数”中找到。

`gss_accept_sec_context()` 在成功完成时将返回 `GSS_S_COMPLETE`。如果上下文不完整，该函数将返回 `GSS_S_CONTINUE_NEEDED`。如果出现错误，该函数将返回错误代码。有关更多信息，请参见 `gss_accept_sec_context`(3GSS) 手册页。

在 GSS-API 中使用其他上下文服务

`gss_init_sec_context()` 函数允许应用程序请求超出所建立基本上下文范围的其他数据保护服务。可通过 `gss_init_sec_context()` 的 *req_flags* 参数来请求这些服务。

并非所有的机制都提供所有这些服务。`gss_init_sec_context()` 的 *ret_flags* 参数用于指示指定上下文中可用的服务。同样，上下文接受器也会通过检查 `gss_accept_sec_context()` 返回的 *ret_flags* 值来确定可用的服务。这些服务将在以下几节中进行介绍。

在 GSS-API 中委托凭证

如果允许的话，上下文启动器可以请求由上下文接受器充当代理。在此类情况下，接受器可以代表启动器启动进一步的上下文。

假设计算机 A 上的某个用户希望通过 `rlogin` 登录到计算机 B，然后通过 `rlogin` 从计算机 B 登录到计算机 C。根据所使用的机制，所授予的凭证会将 B 标识为 A，或者将 B 标识为 A 的代理。

如果允许委托的话，可以将 *ret_flags* 设置为 `GSS_C_DELEG_FLAG`。接受器可接收委托凭证并将其作为 `gss_accept_sec_context()` 的 *delegated_cred_handle* 参数。委托凭证不同于导出上下文。请参见第 74 页中的 “在 GSS-API 中导出和导入上下文”。二者的区别是应用程序可以将其凭证同时委托多次，而上下文一次只能由一个进程持有。

在 GSS-API 中的对等应用程序之间执行相互验证

将文件传送到 `ftp` 站点的用户通常无需站点标识的证明。另一方面，需要向应用程序提供信用卡号的用户则需要接收者身份的确切证明。在此类情况下，需要进行**相互验证**。上下文启动器和上下文接受器都需要证明各自的身份。

上下文启动器可以通过将 `gss_init_sec_context()` *req_flags* 参数设置为值 `GSS_C_MUTUAL_FLAG` 来请求相互验证。如果已经对相互验证进行了授权，则函数会通过将 *ret_flags* 参数设置为该值来表示授权。如果所请求的相互验证不可用，则启动应用程序会负责进行相应的响应。在此情况下 GSS-API 不会自动终止上下文。另外，即使没有特定的请求，某些机制也会始终执行相互验证。

在 GSS-API 中执行匿名验证

正常使用 GSS-API 时，启动器的身份会在建立上下文的过程中提供给接受器。但是，上下文启动器会请求不将其身份显示给上下文接受器。

例如，请考虑提供对医疗数据库进行无限制访问的应用程序。此类服务的客户机可能需要对该服务进行验证。此方法会信任从该数据库中检索的所有信息。例如，出于保密性方面的考虑，客户机可能不希望暴露其身份。

要请求进行匿名验证，请将 `gss_init_sec_context()` 的 *req_flags* 参数设置为 `GSS_C_ANON_FLAG`。要检验匿名验证是否可用，请检查 `gss_init_sec_context()` 或 `gss_accept_sec_context()` 的 *ret_flags* 参数，查看是否返回了 `GSS_C_ANON_FLAG`。

如果匿名验证有效，则针对 `gss_accept_sec_context()` 或 `gss_inquire_context()` 返回的客户机名称调用 `gss_display_name()` 时会生成通用的匿名名称。

注 – 应用程序负责在所请求的匿名验证不被许可时采取相应的措施。在此类情况下，GSS-API 不会终止上下文。

在 GSS-API 中使用通道绑定

对于许多应用程序来说，建立基本的上下文足以确保对上下文启动器进行正确的验证。如果需要实现额外的安全性，则可以使用 GSS-API 所提供的**通道绑定**功能。通道绑定是用于标识所使用的特定数据通道的标记。具体来说，通道绑定可标识起点和终点，即上下文的启动器和接受器。由于标记特定于始发者应用程序和接受者应用程序，因此此类标记可为有效身份提供更多证明。

`gss_channel_bindings_t` 数据类型是一个指针，它指向如下所示的作为通道绑定的 `gss_channel_bindings_struct` 结构。

```
typedef struct gss_channel_bindings_struct {
OM_uint32       initiator_addrtype;
gss_buffer_desc initiator_address;
OM_uint32       acceptor_addrtype;
gss_buffer_desc acceptor_address;
gss_buffer_desc application_data;
} *gss_channel_bindings_t;
```

第一个字段表示地址类型，用于标识所发送的启动器地址采用的格式。第二个字段表示启动器的地址。例如，*initiator_addrtype* 可能会发送到 `GSS_C_AF_INET`，表示 *initiator_address* 采用的是 Internet 地址（即 IP 地址）形式。同样，第三和第四个字段分别表示接受器的地址类型和接受器的地址。应用程序可以根据需要使用最后一个字段 *application_data*。如果不打算使用 *application_data*，请将 *application_data* 设置为 `GSS_C_NO_BUFFER`。如果没有为某个应用程序指定地址，则应当将该应用程序的地址类型字段设置为 `GSS_C_AF_NULLADDR`。第 209 页中的 “通道绑定的地址类型”一节提供了有效地址类型值的列表。

地址类型用于表示地址族，而不是表示特定的寻址格式。对于包含多种替换地址形式的地址族，*initiator_address* 和 *acceptor_address* 字段中必须包含足够的信息才能确定所使用的形式。除非另行指定，否则应当以网络字节顺序（即地址族的本机字节排序）指定地址。

要建立使用通道绑定的上下文，`gss_init_sec_context()` 的 *input_chan_bindings* 参数应当指向所分配的通道绑定结构。该结构的字段将连接到一个八位字节字符串，并将派生一个 MIC。该 MIC 随后会绑定到输出令牌。然后，应用程序会将该令牌发送到上下文接受器。接受器在收到该令牌之后将调用 `gss_accept_sec_context()`。有关更多信息，请参见第 70 页中的 “在 GSS-API 中接受上下文”。`gss_accept_sec_context()` 会针对所收到的通道绑定计算 MIC。如果 MIC 不匹配，`gss_accept_sec_context()` 随后将返回 `GSS_C_BAD_BINDINGS`。

由于 `gss_accept_sec_context()` 会返回已传送的通道绑定，因此接受器可以使用这些值来执行安全检查。例如，接受器可以针对保留在安全数据库中的代码字检查 *application_data* 的值。

注 – 基础机制可能不会为通道绑定信息提供保密性。因此，除非确保能够实现保密性，否则应用程序不应当在通道绑定中包括敏感信息。要测试保密性，应用程序可以检查 `gss_init_sec_context()` 或 `gss_accept_sec_context()` 的 *ret_flags* 参数。值 `GSS_C_CONF_FLAG` 和 `GSS_C_PROT_READY_FLAG` 可用于指示保密性。有关 *ret_flags* 的信息，请参见第 68 页中的 “在 GSS-API 中启动上下文” 或第 70 页中的 “在 GSS-API 中接受上下文”。

单个机制可以针对出现在通道绑定中的地址和地址类型施加额外的约束。例如，一个机制可能会检验通道绑定的 *initiator_address* 字段是否返回到 `gss_init_sec_context()`。因此，具有可移植性的应用程序应当为地址字段提供正确的信息。如果无法确定正确的信息，则应当将 `GSS_C_AF_NULLADDR` 指定为地址类型。

在 GSS-API 中导出和导入上下文

GSS-API 提供了用于导出和导入上下文的方法。此方法允许多进程应用程序（通常是上下文接受器）在进程之间传送上下文。例如，接受器可以有两个进程，一个进程侦听上下文启动器，另一个进程使用在上下文中发送的数据。第 114 页中的 “使用 `test_import_export_context()` 函数”一节说明了如何使用这些函数来保存和恢复上下文。

`gss_export_sec_context()` 函数可创建进程间令牌，其中包含有关所导出上下文的信息。有关更多信息，请参见第 66 页中的 “GSS-API 中的进程间令牌”。调用 `gss_export_sec_context()` 之前，应当将接收令牌的缓冲区设置为 `GSS_C_NO_BUFFER`。

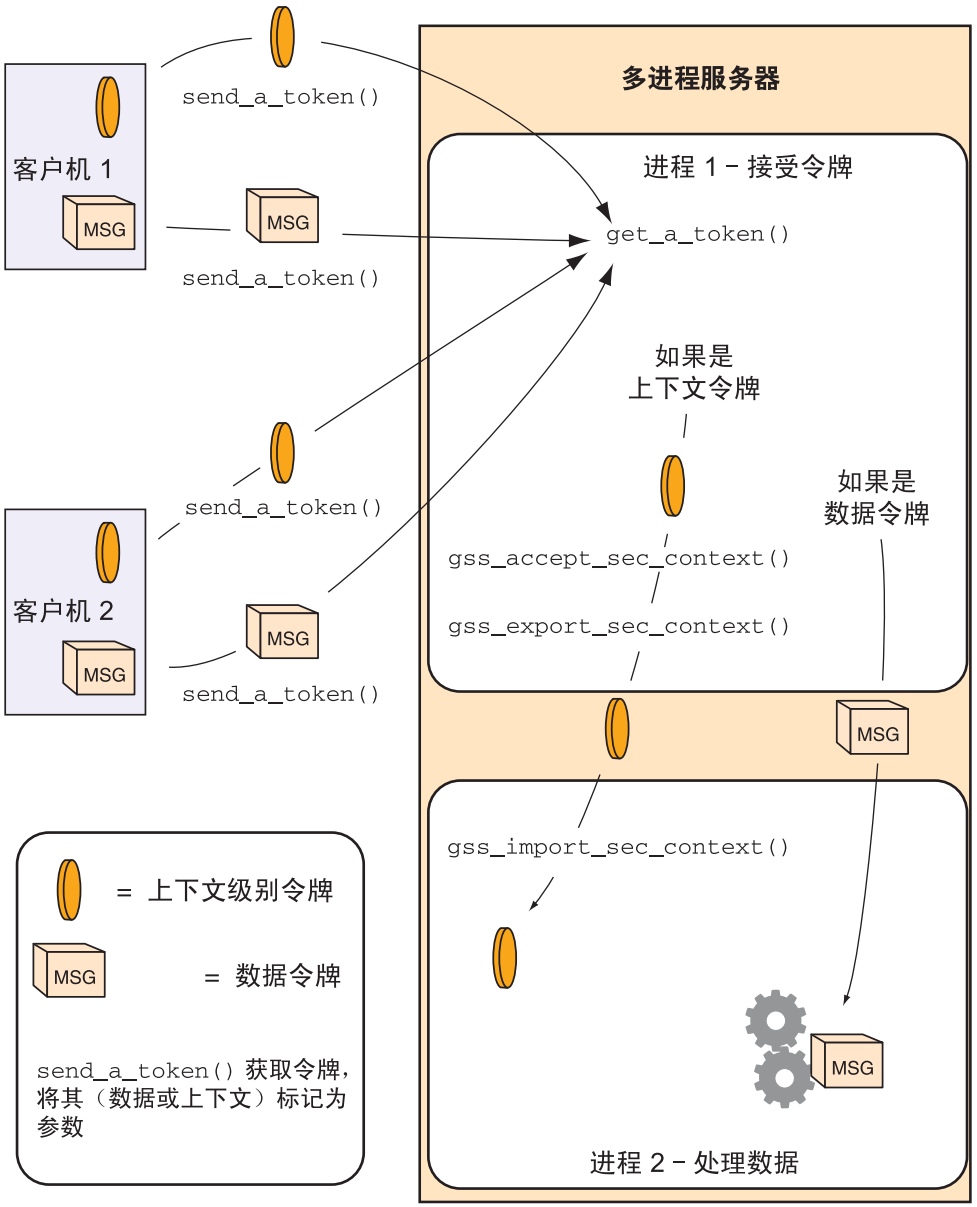
应用程序随后会将令牌传递到另一个进程。新进程将接受该令牌并将其传递到 `gss_import_sec_context()`。用于在应用程序之间传递令牌的函数通常也可用于在进程之间传递令牌。

一次只能存在一个安全进程实例。`gss_export_sec_context()` 可取消激活所导出的上下文并将上下文句柄设置为 `GSS_C_NO_CONTEXT`。`gss_export_sec_context()` 还可取消分配与该上下文相关联的任何进程范围内的资源。如果无法完成对上下文的导出，`gss_export_sec_context()` 会保持现有的安全上下文不变，并且不返回进程间令牌。

并非所有机制都允许导出上下文。应用程序可以通过检查 `gss_accept_sec_context()` 或 `gss_init_sec_context()` 的 *ret_flags* 参数来确定上下文是否可以导出。如果此标志设置为 `GSS_C_TRANS_FLAG`，则可以导出上下文。（请参见第 70 页中的 “在 GSS-API 中接受上下文”和第 68 页中的 “在 GSS-API 中启动上下文”。）

图 4–6 说明多进程接受器如何将上下文导出到多个任务。在这种情况下，进程 1 接收和处理令牌。此步骤会将上下文级别的令牌与数据令牌分开，并将令牌传递到进程 2。进程 2 按照应用程序特定的方式处理数据。在该图中，客户机已经从 `gss_init_sec_context()` 获取了导出令牌。客户机将令牌传递到用户定义的函数 `send_a_token()`，该函数指示要传送的令牌是上下文级别的令牌还是消息令牌。`send_a_token()` 将这些令牌传送到服务器。`send_a_token()` 可能会用于在线程之间传递令牌，但是该图中未显示这一点。

图 4–6 导出上下文：多线程接受器示例



在 GSS-API 中获取上下文信息

GSS-API 提供了一个 gss_inquire_context(3GSS) 函数，该函数可用于获取有关指定的安全上下文的信息。请注意，上下文无需是完整的上下文。如果提供了上下文句柄，则 gss_inquire_context() 将提供以下有关上下文的信息：

- 上下文启动器的名称
- 上下文接受器的名称
- 上下文保持有效的秒数
- 要用于上下文的安全机制
- 若干个上下文参数标志。这些标志与 gss_accept_sec_context(3GSS) 函数的 *ret_flags* 参数相同。这些标志涉及委托、相互验证等。请参见第 70 页中的 “在 GSS-API 中接受上下文”。
- 指示查询应用程序是否为上下文启动器的标志
- 指示上下文是否已完全建立的标志

在 GSS-API 中发送受保护的数据

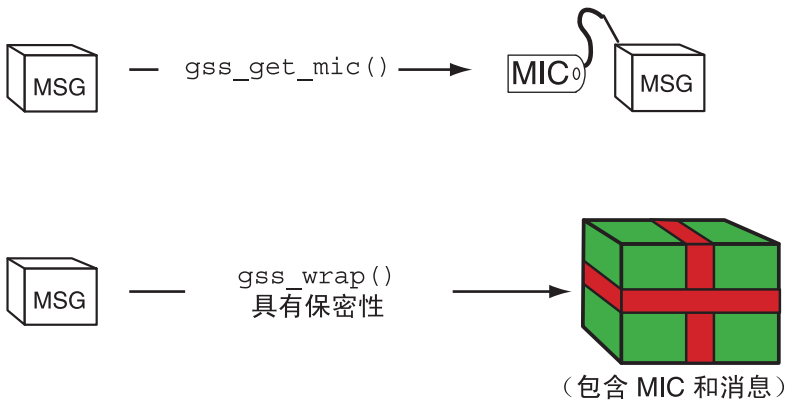
在对等应用程序之间建立上下文之后，即可在发送消息之前对其进行保护。

建立上下文时使用的只是最基本的 GSS-API 保护：**验证**。根据基础安全机制，GSS-API 提供了另外两个保护级别：

- **完整性**－消息的消息完整性代码 (Message Integrity Code, MIC) 由 gss_get_mic() 函数生成。接受者通过检查 MIC 来确保所收到的消息与所发送的消息完全相同。
- **保密性**－除了使用 MIC 以外，还会对消息进行加密。GSS-API 函数 gss_wrap() 可用于执行加密。

gss_get_mic() 和 gss_wrap() 之间的区别如下图所示：使用 gss_get_mic()，接收者可获取用于指示消息是否保持不变的标记；使用 gss_wrap()，接收者除了获取标记外还可获取经过加密的消息。

图 4–7 gss_get_mic() 与 gss_wrap()



要使用哪个函数取决于具体情况。由于 gss_wrap() 包括完整性服务，因此许多程序都使用 gss_wrap()。程序可以测试保密性服务的可用性。随后程序可以根据保密性服务的可用情况，以应用或不应用保密性的方式调用 gss_wrap()。第 96 页中的“包装和发送消息”即是一个示例。但是，由于无需展开使用 gss_get_mic() 的消息，因此所使用的 CPU 周期比 gss_wrap() 所使用的要少。所以，不需要保密性的程序可能会使用 gss_get_mic() 来保护消息。

使用 gss_get_mic() 标记消息

程序可以使用 gss_get_mic() 向消息添加加密 MIC。接受者可以通过调用 gss_verify_mic() 检查消息的 MIC。

与 gss_wrap() 相比，gss_get_mic() 会针对消息和 MIC 生成单独的输出。这意味着发送者应用程序必须既能发送消息又能发送随附的 MIC。更重要的是，接受者必须能够区分消息和 MIC。以下方法可确保对消息和 MIC 进行正确的处理：

- 通过程序控制（即状态）。接受者应用程序可能知道对接收函数调用了两次，第一次用于获取消息，第二次用于获取消息的 MIC。
- 通过标志。发送者和接收者可以标记所包含令牌的类型。
- 通过用户定义的同时包含消息和 MIC 的令牌结构。

gss_get_mic() 在成功完成时将返回 GSS_S_COMPLETE。如果指定的 QOP 无效，则将返回 GSS_S_BAD_QOP。有关更多信息，请参见 gss_get_mic(3GSS)。

使用 gss_wrap() 包装消息

可以使用 gss_wrap() 函数来包装消息。与 gss_get_mic() 一样，gss_wrap() 也提供了 MIC。如果基础机制允许使用所请求的保密性，则 gss_wrap() 也会对指定的消息进行加密。消息的接收者通过使用 gss_unwrap() 来展开消息。

与 gss_get_mic() 不同，gss_wrap() 会将消息和 MIC 一起包装到传出消息中。用于传送该包的函数只需调用一次。在另一端，可以使用 gss_unwrap() 提取消息。MIC 对于应用程序是不可见的。

如果消息已成功包装，则 gss_wrap() 将返回 GSS_S_COMPLETE。如果所请求的 QOP 无效，则将返回 GSS_S_BAD_QOP。有关 gss_wrap() 的示例，请参见第 96 页中的 “包装和发送消息”。

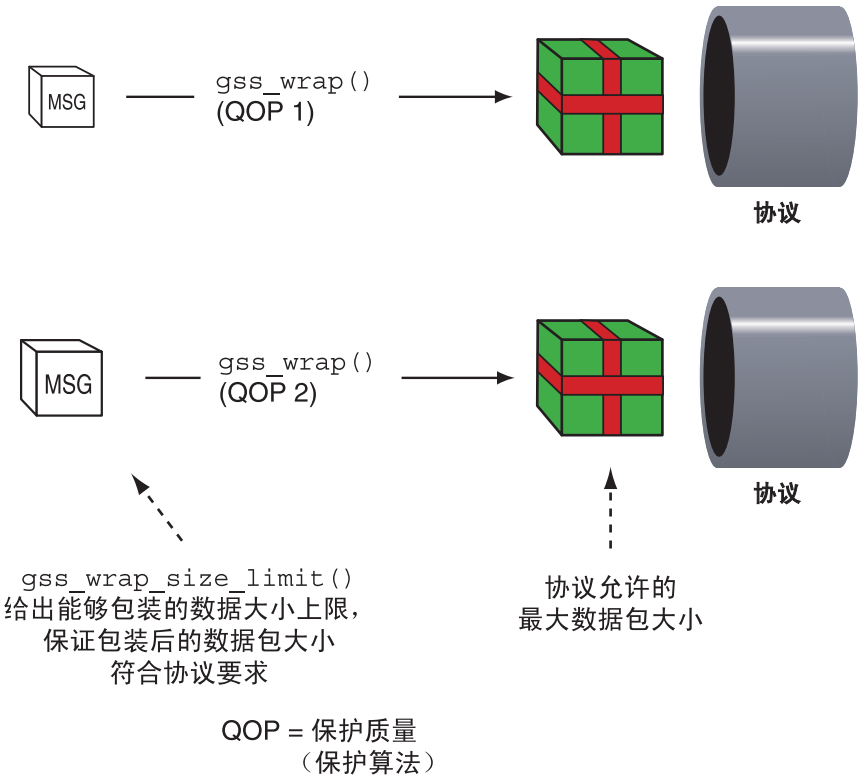
在 GSS-API 中处理包装大小问题

使用 gss_wrap() 来包装消息会增加要发送的数据量。由于所保护的消息包需要适应指定的传输协议，因此 GSS-API 提供了 gss_wrap_size_limit() 函数。gss_wrap_size_limit() 用于计算可以包装的消息的最大大小，使其不至于因变得过大而不适用于该协议。应用程序可以在调用 gss_wrap() 之前截断超出该大小的消息。在实际包装消息之前，请始终检查包装大小限制。

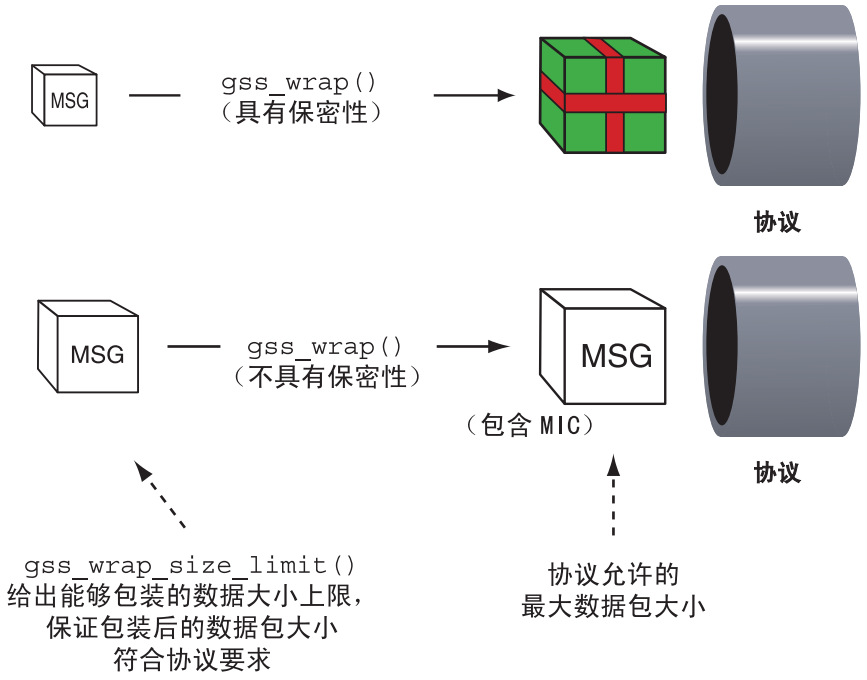
包装大小的增加取决于以下两个因素：

- 用于进行转换的 QOP 算法
- 是否调用了保密性

缺省的 QOP 会因不同的 GSS-API 实现而异。因此，即使指定了缺省的 QOP，所包装消息的大小也会有所不同。下图说明了这种可能性：



无论是否应用了保密性，`gss_wrap()` 仍然会增加消息的大小。`gss_wrap()` 可将 MIC 嵌入到所传送的消息中。但是，对消息进行加密会进一步增加消息的大小。此过程如下图所示。



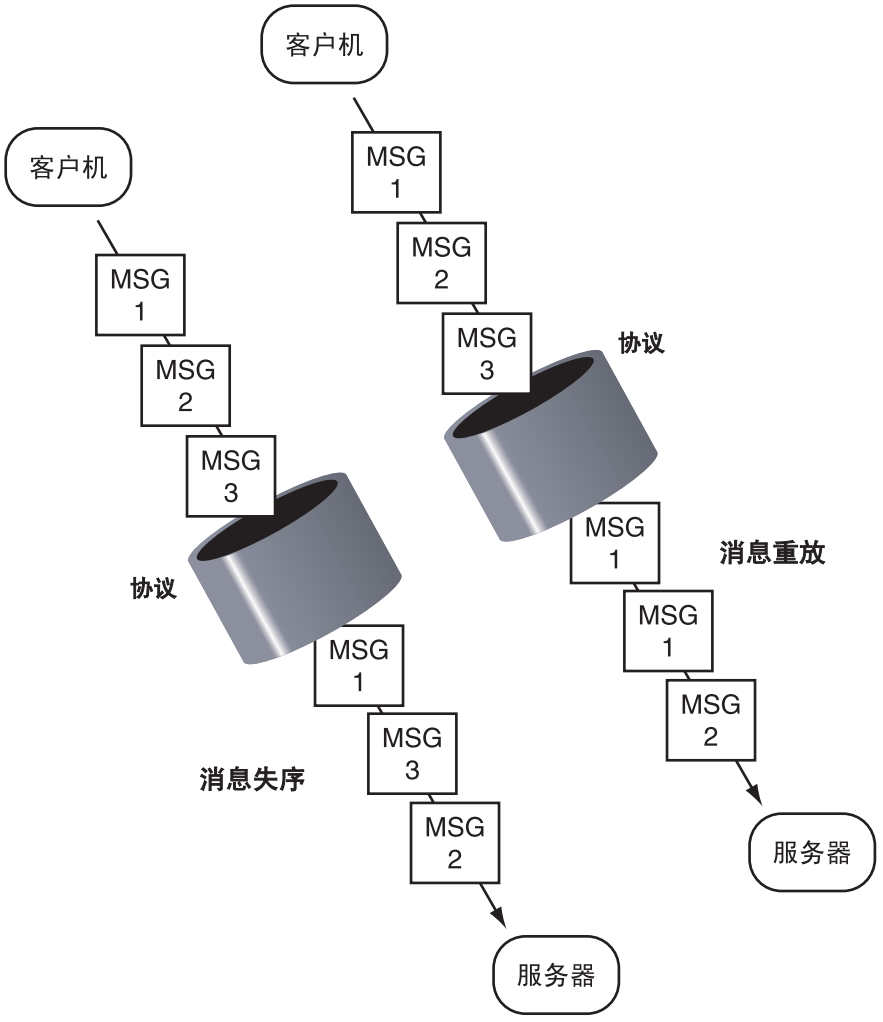
`gss_wrap_size_limit()` 在成功完成时将返回 `GSS_S_COMPLETE`。如果指定的 QOP 无效，则将返回 `GSS_S_BAD_QOP`。第 96 页中的“包装和发送消息”中提供了一个说明如何使用 `gss_wrap_size_limit()` 返回原始消息最大大小的示例。

成功完成此调用并不一定保证 `gss_wrap()` 可以保护长度为 *max-input-size* 个字节的消息。能否保护消息取决于调用 `gss_wrap()` 时系统资源是否可用。有关更多信息，请参见 `gss_wrap_size_limit(3GSS)` 手册页。

在 GSS-API 中检测顺序问题

由于上下文启动器会向接受器传送顺序的数据包，因此某些机制允许上下文接受器检查顺序是否正确。这些检查包括数据包是否按正确的顺序到达以及是否存在不需要的数据包重复。请参见下图。接受器可以在检验数据包和展开数据包的过程中检查这两种情况。有关更多信息，请参见第 113 页中的“展开消息”。

图 4–8 消息重放和消息失序



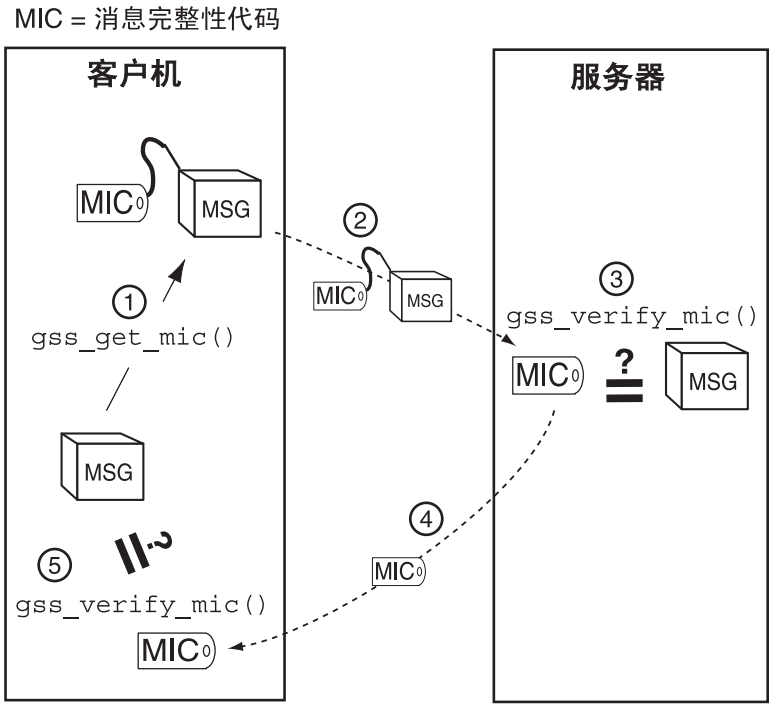
启动器可以使用 `gss_init_sec_context()` 来检查顺序，方法是通过 `GSS_C_REPLAY_FLAG` 或 `GSS_C_SEQUENCE_FLAG` 对 *req_flags* 参数应用逻辑 `OR`。

在 GSS-API 中确认消息传送

接受者展开或检验所传送的消息之后，会向发送者返回确认信息。这意味着会发回该消息的 MIC。请考虑以下情况：消息没有由发送者进行包装，而只是通过 `gss_get_mic()` 使用 MIC 进行了标记。图 4–9 中说明的过程如下所示：

1. 启动器使用 `gss_get_mic()` 对消息进行标记。
2. 启动器将该消息和 MIC 发送到接受器。
3. 接受器使用 `gss_verify_mic()` 检验该消息。
4. 接受器将该 MIC 发回到启动器。
5. 启动器使用 `gss_verify_mic()` 针对原始消息检验所收到的 MIC。

图 4–9 确认 MIC 数据



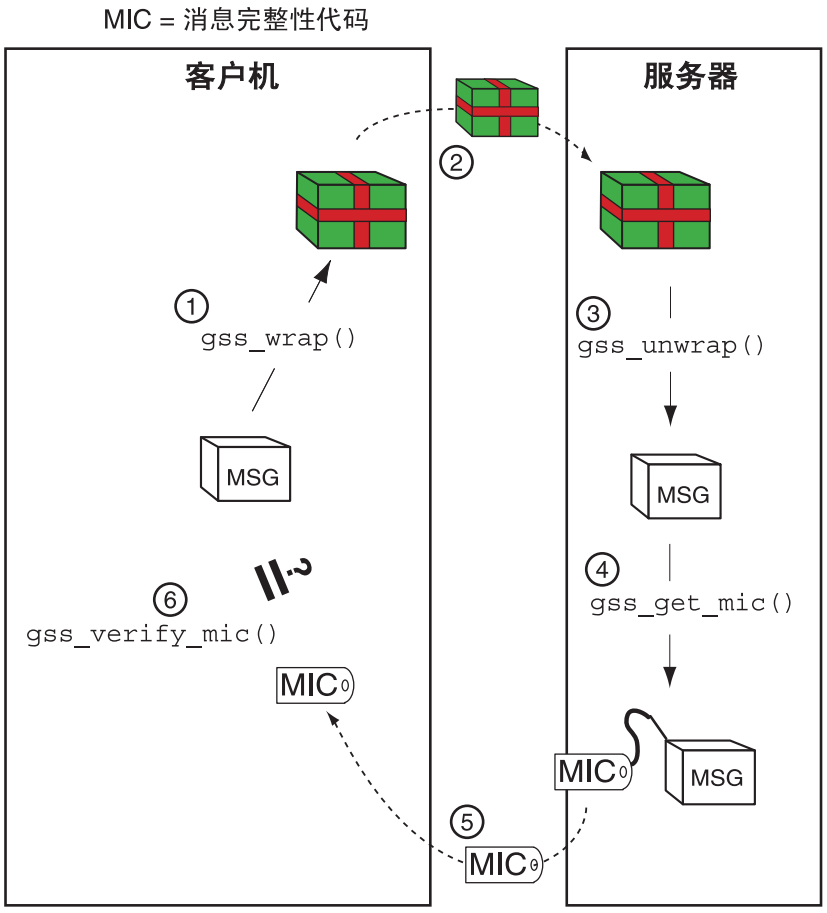
对于已包装的数据，`gss_unwrap()` 函数从不生成单独的 MIC，因此，接受者必须根据所收到的未包装的消息生成它。图 4–10 中说明的过程如下所示：

1. 启动器使用 `gss_wrap()` 包装消息。

2. 启动器发送已包装的消息。
3. 接受器使用 `gss_unwrap()` 展开该消息。
4. 接受器调用 `gss_get_mic()` 以生成未包装的消息的 MIC。
5. 接受器将派生的 MIC 发送到启动器。
6. 启动器使用 `gss_verify_mic()` 将所收到的 MIC 与原始消息进行比较。

对于已经为 GSS-API 数据分配的任何数据空间，应用程序应当取消对其进行分配。相关函数包括 `gss_release_buffer(3GSS)`、`gss_release_cred(3GSS)`、`gss_release_name(3GSS)` 和 `gss_release_oid_set(3GSS)`。

图 4–10 确认已包装的数据



清除 GSS-API 会话

最后，发送和接收所有的消息，之后启动器应用程序和接受器应用程序就将完成操作。此时，这两个应用程序都应当调用 `gss_delete_sec_context()` 以销毁共享的上下文。`gss_delete_sec_context()` 可删除与该上下文相关的局部数据结构。

对于已经为 GSS-API 数据分配的任何数据空间，确保应用程序取消对其进行分配是不错的方法。这可以通过
`gss_release_buffer()`、`gss_release_cred()`、`gss_release_name()` 和 `gss_release_oid_set()` 函数来完成。

◆ ◆ ◆ 第 5 章

GSS-API 客户机示例

本章将对典型的 GSS-API 客户机应用程序进行详细介绍，本章包含以下主题：

GSSAPI 客户机示例概述

样例客户端程序 `gss-client` 可创建与服务器之间的安全上下文，建立安全参数，并将**消息**字符串发送到服务器。此程序使用基于 TCP 的简单套接字连接机制来建立连接。

以下几节将提供有关 `gss-client` 工作方式的逐步说明。由于 `gss-client` 是一个用于说明 GSSAPI 功能的样例程序，因此仅详细讨论该程序的相关部分。这两个应用程序的完整源代码可在附录中找到，也可以从以下网址下载：

```
http://www.oracle.com/
technetwork/indexes/downloads/sdlc-decommission-333274.html
```

GSSAPI 客户机示例结构

`gss-client` 应用程序执行以下步骤：

1. 解析命令行。
2. 如果指定了机制，则为机制创建一个对象 ID (Object ID, OID)。否则，使用缺省机制，此为最常见的情况。

3. 创建与服务器的连接。
4. 建立安全上下文。
5. 包装和发送消息。
6. 验证服务器是否已对消息进行正确的“签名”。
7. 删除安全上下文。

运行 GSSAPI 客户机示例

gss-client 示例在命令行中采用以下形式：

```
gss-client [-port port] [-d] [-mech mech] host service-name [-f] msg
```

- *port*－用于与 *host* 所指定的远程计算机建立连接的端口号。
- -d 标志－用于将安全凭证委托给服务器。具体来说，*deleg-flag* 变量会设置为 GSS-API 值 GSS_C_DELEG_FLAG。否则，*deleg-flag* 设置为零。
- *mech*－要使用的安全机制的名称，如 Kerberos v5。如果未指定任何机制，GSS-API 将使用缺省机制。
- *host*－服务器的名称。
- *service-name*－客户机所请求的网络服务的名称。telnet、ftp 和 login 服务便是一些典型的示例。
- *msg*－作为受保护的数据发送到服务器的字符串。如果指定了 -f 选项，则 *msg* 是指从中读取字符串的文件名。

客户机应用程序的典型命令行可能如以下示例所示：

```
% gss-client -port 8080 -d -mech kerberos_v5 erebos.eng nfs "ls"
```

以下示例不指定机制、端口或委托：

```
% gss-client erebos.eng nfs "ls"
```

GSSAPI 客户机示例：main() 函数

与所有 C 程序一样，该程序的外部 shell 也包含在入口点函数 main() 中。main() 可执行以下四种功能：

- 解析命令行参数并为变量指定参数。
- 如果要使用缺省机制以外的机制，则调用 parse_oid() 来创建 GSS-API OID（即对象标识符）。对象标识符来自安全机制的名称，前提是已经提供了机制名称。
- 调用 call_server()，这会实际创建上下文并发送数据。
- 发送数据之后，根据需要释放 OID 的存储空间。

main() 例程的源代码如以下示例所示。

示例 5–1 gss-client 示例：main()

```
int main(argc, argv)
     int argc;
     char **argv;
{
     char *msg;
     char service_name[128];
     char hostname[128];
     char *mechanism = 0;
     u_short port = 4444;
     int use_file = 0;
     OM_uint32 deleg_flag = 0, min_stat;

     display_file = stdout;

     /* Parse command-line arguments. */

        argc--; argv++;
     while (argc) {
          if (strcmp(*argv, "-port") == 0) {
               argc--; argv++;
               if (!argc) usage();
               port = atoi(*argv);
          } else if (strcmp(*argv, "-mech") == 0) {
               argc--; argv++;
               if (!argc) usage();
               mechanism = *argv;
          } else if (strcmp(*argv, "-d") == 0) {
               deleg_flag = GSS_C_DELEG_FLAG;
          } else if (strcmp(*argv, "-f") == 0) {
               use_file = 1;
          } else
               break;
          argc--; argv++;
     }
     if (argc != 3)
          usage();

     if (argc > 1) {
                strcpy(hostname, argv[0]);
        } else if (gethostname(hostname, sizeof(hostname)) == -1) {
                       perror("gethostname");
                       exit(1);
        }


     if (argc > 2) {
        strcpy(service_name, argv[1]);
        strcat(service_name, "@");
        strcat(service_name, hostname);

     }

      msg = argv[2];
```

示例 5–1 gss-client 示例：main() （续）

```
     /* Create GSSAPI object ID. */
     if (mechanism)
         parse_oid(mechanism, &g_mechOid);

     /* Call server to create context and send data. */
     if (call_server(hostname, port, g_mechOid, service_name,
                  deleg_flag, msg, use_file) < 0)
          exit(1);

     /* Release storage space for OID, if still allocated  */
     if (g_mechOid != GSS_C_NULL_OID)
         (void) gss_release_oid(&min_stat, &gmechOid);

     return 0;
}
```

打开与服务器的连接

call_server() 函数使用以下代码与服务器建立连接：

```
     if ((s = connect_to_server(host, port)) < 0)
          return -1;
```

s 是一个最初通过调用 socket() 返回的 int 类型的文件描述符。

connect_to_server() 是 GSS-API 外部的一个简单函数，它使用套接字来创建连接。connect_to_server() 的源代码如以下示例所示。

示例 5–2 connect_to_server() 函数

```
int connect_to_server(host, port)
     char *host;
     u_short port;
{
     struct sockaddr_in saddr;
     struct hostent *hp;
     int s;

     if ((hp = gethostbyname(host)) == NULL) {
          fprintf(stderr, "Unknown host: %s\n", host);
          return -1;
     }

     saddr.sin_family = hp->h_addrtype;
     memcpy((char *)&saddr.sin_addr, hp->h_addr, sizeof(saddr.sin_addr));
     saddr.sin_port = htons(port);

     if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
          perror("creating socket");
          return -1;
     }
```

示例 5–2 connect_to_server() 函数 （续）

```
     if (connect(s, (struct sockaddr *)&saddr, sizeof(saddr)) < 0) {
          perror("connecting to server");
          (void) close(s);
          return -1;
     }

     return s;
}
```

建立与服务器的安全上下文

建立连接之后，call_server() 会使用 client_establish_context() 函数来创建安全上下文，如下所示：

```
if (client_establish_context(s, service-name, deleg-flag, oid, &context,
                                  &ret-flags) < 0) {
          (void) close(s);
          return -1;
     }
```

- *s* 是一个文件描述符，表示通过 connect_to_server() 建立的连接。
- *service-name* 是所请求的网络服务。
- *deleg-flag* 用于指定服务器是否可以充当客户机的代理。
- *oid* 表示机制。
- *context* 是要创建的上下文。
- *ret-flags* 是一个 int，用于指定 GSS-API 函数 gss_init_sec_context() 所返回的任何标志。

client_establish_context() 执行以下任务：

- 将服务名称转换为 GSSAPI 内部格式
- 在客户机和服务器之间循环执行令牌交换，直到建立完整的安全上下文为止

将服务名称转换为 GSS-API 格式

client_establish_context() 执行的首个任务是使用 gss_import_name() 将服务名称字符串转换为 GSS-API 内部格式。

示例 5–3 client_establish_context() 一转换服务名称

```
     /*
     * Import the name into target_name.  Use send_tok to save
     * local variable space.
     */
```

示例 5–3 client_establish_context()－转换服务名称 （续）

```
    send_tok.value = service_name;
    send_tok.length = strlen(service_name) + 1;
    maj_stat = gss_import_name(&min_stat, &send_tok,
                       (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &target_name);
    if (maj_stat != GSS_S_COMPLETE) {
         display_status("parsing name", maj_stat, min_stat);
         return -1;
    }
```

gss_import_name() 提取存储在不透明 GSS_API 缓冲区 send_tok 中的服务名称，然后将服务名称字符串转换为 GSS_API 内部名称 target_name。send_tok 用于节省空间，而不是用于声明新的 gss_buffer_desc。第三个参数类型为 gss_OID，用于指明 send_tok 名称的格式。此示例使用 GSS_C_NT_HOSTBASED_SERVICE，这表示服务采用 *service@host* 格式。有关此参数的其他可能值，请参见第 208 页中的“名称类型”。

为 GSS-API 建立安全上下文

将服务转换为 GSS-API 内部格式之后即可建立上下文。为了尽可能提高可移植性，应当始终循环建立上下文。

进入循环之前，client_establish_context() 会初始化上下文和 *token_ptr* 参数。使用 *token_ptr* 时需要进行选择。*token_ptr* 可以指向 *send_tok*（发送到服务器的令牌）或 *recv_tok*（服务器发回的令牌）。

在循环内部，需要检查两项：

- gss_init_sec_context() 返回的状态

  返回状态可捕捉任何可能要求循环异常中止的错误。当且仅当服务器还要发送另一个令牌时，gss_init_sec_context() 才会返回 GSS_S_CONTINUE_NEEDED。

- gss_init_sec_context() 所生成的要发送到服务器的令牌的大小

  如果令牌大小为零，则表示不再有可以发送到服务器的信息并且可以退出循环。令牌大小可根据 *token_ptr* 来确定。

以下伪代码对该循环进行了说明：

```
do
   gss_init_sec_context()
   if no context was created
      exit with error;
   if the status is neither "complete" nor "in process"
      release the service namespace and exit with error;
   if there is a token to send to the server, that is, the size is nonzero
```

```
    send the token;
    if sending the token  fails,
        release the token and service namespaces.Exit with error;
    release the namespace for the token that was just sent;
  if  the context is not completely set up
    receive a token from the server;
while the context is not complete
```

该循环从调用 `gss_init_sec_context()` 开始，该函数使用以下参数：

- 要由基础机制设置的状态码。
- 凭证句柄。此示例使用 `GSS_C_NO_CREDENTIAL` 充当缺省主体。
- 要创建的上下文句柄。
- 上下文接受器的名称。
- 所需机制的对象 ID。
- 请求标志。在本示例中，客户机请求执行以下操作：服务器对自身进行验证，打开消息重复功能，服务器根据请求充当代理。
- 对于上下文无时间限制。
- 没有请求使用通道绑定。
- 要从对等应用程序接收的令牌。
- 服务器实际使用的机制。机制在此处设置为 `NULL`，这是因为应用程序不会使用该值。
- `gss_init_sec_context()` 所创建的要发送到对等应用程序的令牌。
- 返回标志。设置为 `NULL`，这是因为在此示例中忽略了这些标志。

注 – 客户机在启动上下文之前无需获取凭证。在客户端，凭证管理通过 GSS-API 以透明方式处理。即，GSS-API **知道**如何获取此机制为该主体创建的凭证。因此，应用程序可以向 `gss_init_sec_context()` 传递缺省凭证。但是在服务器端，服务器应用程序在接受上下文之前必须明确获取服务的凭证。请参见第 104 页中的 “获取凭证”。

检查了上下文或其一部分是否存在，以及 `gss_init_sec_context()` 是否返回有效状态之后，`connect_to_server()` 会检查 `gss_init_sec_context()` 是否提供了要发送到服务器的令牌。如果不存在任何令牌，表明服务器已经指示不需要其他任何令牌。如果提供了令牌，必须将该令牌发送到服务器。如果发送该令牌失败，则无法确定该令牌和服务的名称空间，并且 `connect_to_server()` 将退出。以下算法通过查看令牌的长度来检查令牌是否存在：

```
if (send_tok_length != 0) {
     if (send_token(s, &send_tok) < 0) {
          (void) gss_release_buffer(&min_stat, &send_tok);
          (void) gss_release_name(&min_stat, &target_name);
```

```
            return -1;
        }
}
```

send_token() 不是 GSS-API 函数，需要由用户进行编写。send_token() 函数可将令牌写入到文件描述符中。send_token() 在成功时返回 0，在失败时返回 –1。GSS-API 本身不收发令牌。调用应用程序负责收发 GSS-API 已创建的任何令牌。

下面提供了用于建立上下文循环的源代码。

示例 5–4 用于建立上下文的循环

```
/*
 * Perform the context establishment loop.
 *
 * On each pass through the loop, token_ptr points to the token
 * to send to the server (or GSS_C_NO_BUFFER on the first pass).
 * Every generated token is stored in send_tok which is then
 * transmitted to the server; every received token is stored in
 * recv_tok, which token_ptr is then set to, to be processed by
 * the next call to gss_init_sec_context.
 *
 * GSS-API guarantees that send_tok's length will be non-zero
 * if and only if the server is expecting another token from us,
 * and that gss_init_sec_context returns GSS_S_CONTINUE_NEEDED if
 * and only if the server has another token to send us.
 */

token_ptr = GSS_C_NO_BUFFER;
*gss_context = GSS_C_NO_CONTEXT;
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456

do {
    maj_stat =
        gss_init_sec_context(&min_stat, GSS_C_NO_CREDENTIAL,
        gss_context, target_name, oid,
        GSS_C_MUTUAL_FLAG | GSS_C_REPLAY_FLAG | deleg_flag,
        0, NULL,                        /* no channel bindings */
        token_ptr, NULL,                /* ignore mech type */
        &send_tok, ret_flags, NULL);  /* ignore time_rec */
    if (gss_context == NULL){
        printf("Cannot create context\n");
        return GSS_S_NO_CONTEXT;
    }
    if (token_ptr != GSS_C_NO_BUFFER)
        (void) gss_release_buffer(&min_stat, &recv_tok);
    if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
        display_status("initializing context", maj_stat, min_stat);
        (void) gss_release_name(&min_stat, &target_name);
        return -1;
    }

    if (send_tok.length != 0){
        fprintf(stdout, "Sending init_sec_context token (size=%ld)...",
            send_tok.length);
        if (send_token(s, &send_tok) < 0) {
            (void) gss_release_buffer(&min_stat, &send_tok);
```

示例 5–4 用于建立上下文的循环 （续）

```
            (void) gss_release_name(&min_stat, &target_name);
            return -1;
        }
    }
    (void) gss_release_buffer(&min_stat, &send_tok);

    if (maj_stat == GSS_S_CONTINUE_NEEDED) {
        fprintf(stdout, "continue needed...");
        if (recv_token(s, &recv_tok) < 0) {
            (void) gss_release_name(&min_stat, &target_name);
            return -1;
        }
        token_ptr = &recv_tok;
    }
    printf("\n");
} while (maj_stat == GSS_S_CONTINUE_NEEDED);
```

有关 `send_token()` 和 `recv_token()` 工作方式的更多信息，请参见第 195 页中的 “各种 GSS-API 样例函数”。

客户端上的各种 GSSAPI 上下文操作

作为样例程序，`gss-client` 所执行的一些功能用于说明。以下源代码不是执行基本任务所必需的，之所以提供它是为了说明以下其他操作：

- 保存和恢复上下文
- 显示上下文标志
- 获取上下文状态

这些操作的源代码如以下示例所示。

示例 5–5 gss-client： `call_server()` 建立上下文

```
/* Save and then restore the context */
     maj_stat = gss_export_sec_context(&min_stat,
                                       &context,
                                       &context_token);
     if (maj_stat != GSS_S_COMPLETE) {
             display_status("exporting context", maj_stat, min_stat);
             return -1;
     }
     maj_stat = gss_import_sec_context(&min_stat,
                                       &context_token,
                                       &context);
     if (maj_stat != GSS_S_COMPLETE) {
        display_status("importing context", maj_stat, min_stat);
        return -1;
     }
     (void) gss_release_buffer(&min_stat, &context_token);
```

示例 5–5 gss-client：call_server() 建立上下文 （续）

```
    /* display the flags */
    display_ctx_flags(ret_flags);

    /* Get context information */
    maj_stat = gss_inquire_context(&min_stat, context,
                                   &src_name, &targ_name, &lifetime,
                                   &mechanism, &context_flags,
                                   &is_local,
                                   &is_open);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("inquiring context", maj_stat, min_stat);
        return -1;
    }

    if (maj_stat == GSS_S_CONTEXT_EXPIRED) {
    printf(" context expired\n");
        display_status("Context is expired", maj_stat, min_stat);
        return -1;
    }
```

包装和发送消息

gss-client 应用程序必须首先包装（即加密）数据，然后才能将其发送。应用程序通过执行以下步骤来包装消息：

- 确定包装大小限制。此过程可确保协议提供经过包装的消息。
- 获取源和目标的名称。将名称从对象标识符转换为字符串。
- 获取机制名称的列表。将名称从对象标识符转换为字符串。
- 将消息插入到缓冲区中并对其进行包装。
- 将消息发送到服务器。

以下源代码可用于包装消息。

示例 5–6 gss-client 示例：call_server() — 包装消息

```
/* Test gss_wrap_size_limit */
maj_stat = gss_wrap_size_limit(&min_stat, context, conf_req_flag,
    GSS_C_QOP_DEFAULT, req_output_size, &max_input_size);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("wrap_size_limit call", maj_stat, min_stat);
} else
    fprintf (stderr, "gss_wrap_size_limit returned "
        "max input size = %d \n"
        "for req_output_size = %d with Integrity only\n",
         max_input_size , req_output_size , conf_req_flag);

conf_req_flag = 1;
maj_stat = gss_wrap_size_limit(&min_stat, context, conf_req_flag,
    GSS_C_QOP_DEFAULT, req_output_size, &max_input_size);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("wrap_size_limit call", maj_stat, min_stat);
```

示例 5–6 gss-client示例： call_server()－包装消息 （续）

```
} else
    fprintf (stderr, "gss_wrap_size_limit returned "
        " max input size = %d \n" "for req_output_size = %d with "
        "Integrity & Privacy \n", max_input_size , req_output_size );

maj_stat = gss_display_name(&min_stat, src_name, &sname, &name_type);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("displaying source name", maj_stat, min_stat);
    return -1;
}

maj_stat = gss_display_name(&min_stat, targ_name, &tname,
    (gss_OID *) NULL);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("displaying target name", maj_stat, min_stat);
    return -1;
 }
fprintf(stderr, "\"%.*s\" to \"%.*s\", lifetime %u, flags %x, %s, %s\n",
    (int) sname.length, (char *) sname.value, (int) tname.length,
    (char *) tname.value, lifetime, context_flags,
    (is_local) ? "locally initiated" : "remotely initiated",
    (is_open) ? "open" : "closed");

(void) gss_release_name(&min_stat, &src_name);
(void) gss_release_name(&min_stat, &targ_name);
(void) gss_release_buffer(&min_stat, &sname);
(void) gss_release_buffer(&min_stat, &tname);

maj_stat = gss_oid_to_str(&min_stat, name_type, &oid_name);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("converting oid->string", maj_stat, min_stat);
    return -1;
}
fprintf(stderr, "Name type of source name is %.*s.\n", (int) oid_name.length,
    (char *) oid_name.value);
(void) gss_release_buffer(&min_stat, &oid_name);

/* Now get the names supported by the mechanism */
maj_stat = gss_inquire_names_for_mech(&min_stat, mechanism, &mech_names);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("inquiring mech names", maj_stat, min_stat);
    return -1;
}

maj_stat = gss_oid_to_str(&min_stat, mechanism, &oid_name);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("converting oid->string", maj_stat, min_stat);
    return -1;
}
mechStr = (char *)__gss_oid_to_mech(mechanism);
fprintf(stderr, "Mechanism %.*s (%s) supports %d names\n", (int) oid_name.length,
    (char *) oid_name.value, (mechStr == NULL ? "NULL" : mechStr),
    mech_names->count);
(void) gss_release_buffer(&min_stat, &oid_name);

for (i=0; i < mech_names->count; i++) {
```

示例 5–6 gss-client 示例：call_server() — 包装消息 （续）

```
    maj_stat = gss_oid_to_str(&min_stat, &mech_names->elements[i], &oid_name);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("converting oid->string", maj_stat, min_stat);
        return -1;
    }
    fprintf(stderr, "  %d: %.*s\n", i, (int) oid_name.length, (
    char *) oid_name.value);

    (void) gss_release_buffer(&min_stat, &oid_name);
    }
(void) gss_release_oid_set(&min_stat, &mech_names);

if (use_file) {
    read_file(msg, &in_buf);
} else {
    /* Wrap the message */
    in_buf.value = msg;
    in_buf.length = strlen(msg) + 1;
}

if (ret_flag & GSS_C_CONF_FLAG) {
    state = 1;
else
    state = 0;
}
maj_stat = gss_wrap(&min_stat, context, 1, GSS_C_QOP_DEFAULT, &in_buf,
    &state, &out_buf);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("wrapping message", maj_stat, min_stat);
    (void) close(s);
    (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
    return -1;
} else if (! state) {
    fprintf(stderr, "Warning!  Message not encrypted.\n");
}

/* Send to server */
if (send_token(s, &out_buf) < 0) {
    (void) close(s);
    (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
    return -1;
}
(void) gss_release_buffer(&min_stat, &out_buf);
```

读取和验证 GSS-API 客户机中的签名块

现在，gss-client 程序可以测试已发送消息的有效性。服务器会针对已发送的消息返回 MIC。可以通过 recv_token() 检索此消息。

然后，使用 gss_verify_mic() 函数验证消息的**签名**（即 MIC）。gss_verify_mic() 用于将收到的 MIC 与未包装的原始消息进行比较。收到的 MIC 来自服务器的令牌，该令牌存储在 *out_buf* 中。来自未包装版本的消息的 MIC 存放在 *in_buf* 中。如果这两个 MIC 匹配，系统便会验证此消息。客户机随后会为所收到的令牌释放缓冲区 *out_buf*。

以下源代码说明了读取和验证签名块的过程。

示例 5–7 gss-client 示例－读取和验证签名块

```
/* Read signature block into out_buf */
     if (recv_token(s, &out_buf) < 0) {
          (void) close(s);
          (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
          return -1;
     }

/* Verify signature block */
     maj_stat = gss_(&min_stat, context, &in_buf,
                                &out_buf, &qop_state);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("verifying signature", maj_stat, min_stat);
          (void) close(s);
          (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
          return -1;
     }
     (void) gss_release_buffer(&min_stat, &out_buf);

     if (use_file)
         free(in_buf.value);

     printf("Signature verified.\n");
```

删除安全上下文

`call_server()` 函数通过删除上下文并返回到 `main()` 函数来结束操作。

示例 5–8 gss-client 示例： `call_server()` －删除上下文

```
/* Delete context */
     maj_stat = gss_delete_sec_context(&min_stat, &context, &out_buf);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("deleting context", maj_stat, min_stat);
          (void) close(s);
          (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
          return -1;
     }

     (void) gss_release_buffer(&min_stat, &out_buf);
     (void) close(s);
     return 0;
```

◆ ◆ ◆ 第 6 章

GSS-API 服务器示例

本章将对 gss-server 样例程序的源代码进行详细介绍，本章包含以下主题：

GSSAPI 服务器示例概述

样例服务器端程序 gss-server 可以与上一章中介绍的 gss-client 结合使用。gss-server 的基本用途是从 gssapi-client 接收和返回经过包装的消息并对其进行签名。

以下几节将提供有关 gss-server 工作方式的逐步说明。由于 gss-server 是一个用于说明 GSSAPI 功能的样例程序，因此仅详细讨论该程序的相关部分。这两个应用程序的完整源代码可在附录中找到，也可以从以下网址下载：

GSSAPI 服务器示例结构

gss-structure 应用程序执行以下步骤：

1. 解析命令行。
2. 如果指定了机制，请将该机制的名称转换为内部格式。
3. 获取调用者的凭证。
4. 查看用户是否指定了要使用 inetd 守护进程进行连接。
5. 与客户机建立连接。
6. 从客户机接收数据。
7. 对数据进行签名并返回数据。

8. 释放名称空间并退出。

运行 GSSAPI 服务器示例

gss-server 在命令行中采用以下形式：

```
gss-server [-port port] [-verbose] [-inetd] [-once] [-logfile file] \
           [-mech mechanism] service-name
```

- *port* 是要侦听的端口号。如果未指定任何端口，则程序使用端口 4444 作为缺省端口。
- -verbose 会导致在运行 gss-server 时显示消息。
- -inetd 表示程序应当使用 inetd 守护进程来侦听端口。-inetd 使用 stdin 和 stdout 连接到客户机。
- -once 表示仅建立单实例连接。
- *mechanism* 是要使用的安全机制的名称，如 Kerberos v5。如果未指定任何机制，GSS-API 将使用缺省机制。
- *service-name* 是客户机所请求的网络服务的名称，如 telnet、ftp 或登录服务。

典型的命令行可能如以下示例所示：

```
% gss-server -port 8080 -once -mech kerberos_v5 erebos.eng nfs "hello"
```

GSSAPI 服务器示例：main() 函数

gss-server main() 函数可执行以下任务：

- 解析命令行参数并为变量指定参数
- 获取与机制对应的服务的凭证
- 调用 sign_server() 函数，该函数会执行涉及到对消息进行签名和返回消息等工作
- 释放已经获取的凭证
- 释放机制的 OID 名称空间
- 在连接仍处于打开状态时关闭连接

示例 6–1 gss-server 示例：main()

```
int
main(argc, argv)
     int argc;
     char **argv;
{
     char *service_name;
     gss_cred_id_t server_creds;
     OM_uint32 min_stat;
     u_short port = 4444;
```

示例6–1 gss-server示例： main() （续）

```
int s;
int once = 0;
int do_inetd = 0;

log = stdout;
display_file = stdout;

/* Parse command-line arguments. */
argc--; argv++;
while (argc) {
if (strcmp(*argv, "-port") == 0) {
     argc--; argv++;
     if (!argc) usage();
     port = atoi(*argv);
} else if (strcmp(*argv, "-verbose") == 0) {
     verbose = 1;
} else if (strcmp(*argv, "-once") == 0) {
     once = 1;
} else if (strcmp(*argv, "-inetd") == 0) {
     do_inetd = 1;
} else if (strcmp(*argv, "-logfile") == 0) {
     argc--; argv++;
     if (!argc) usage();
     log = fopen(*argv, "a");
     display_file = log;
     if (!log) {
     perror(*argv);
     exit(1);
     }
} else
     break;
argc--; argv++;
}
if (argc != 1)
     usage();

if ((*argv)[0] == '-')
     usage();

service_name = *argv;

/* Acquire service credentials. */
if (server_acquire_creds(service_name, &server_creds) < 0)
     return -1;

if (do_inetd) {
     close(1);
     close(2);
     /* Sign and return message. */
     sign_server(0, server_creds);
     close(0);
} else {
     int stmp;

     if ((stmp = create_socket(port)) >= 0) {
         do {
```

示例 6–1 gss-server 示例：main() （续）

```
                /* Accept a TCP connection */
                if ((s = accept(stmp, NULL, 0)) < 0) {
                    perror("accepting connection");
                    continue;
                }
                /* This return value is not checked, because there is
                   not really anything to do if it fails. */
                sign_server(s, server_creds);
                close(s);
            } while (!once);

            close(stmp);
        }
    }

    /* Close down and clean up. */
    (void) gss_release_cred(&min_stat, &server_creds);

    /*NOTREACHED*/
    (void) close(s);
    return 0;
}
```

获取凭证

凭证由基础机制创建，而不是由客户机应用程序、服务器应用程序或 GSS-API 创建。客户机程序通常具有在登录时获取的凭证。服务器需要始终明确获取凭证。

gss-server 程序包含 server_acquire_creds() 函数，该函数用于获取要提供的服务的凭证。server_acquire_creds() 将该服务的名称和要使用的安全机制用作输入。然后，server_acquire_creds() 返回该服务的凭证。

server_acquire_creds() 使用 GSS-API 函数 gss_acquire_cred() 来获取服务器所提供的服务的凭证。server_acquire_creds() 访问 gss_acquire_cred() 之前，server_acquire_creds() 必须执行以下两个任务：

1. 检查机制列表并将该列表缩减成单个机制以获取凭证。

   如果多个机制可以共享单个凭证，gss_acquire_cred() 函数将返回所有这些机制的凭证。因此，gss_acquire_cred() 会将**一组**机制用作输入。（请参见第 67 页中的“在 GSS-API 中使用凭证”。）但是，在大多数情况（包括此情况）下，单个凭证可能并不适用于多个机制。gss-server 程序中会在命令行上指定单个机制或使用缺省机制。因此，第一个任务是确保传递给 gss_acquire_cred() 的那组机制中包含单个机制（缺省机制或其他机制），如下所示：

   ```
   if (mechOid != GSS_C_NULL_OID) {
        desiredMechs = &mechOidSet;
        mechOidSet.count = 1;
        mechOidSet.elements = mechOid;
   } else
        desiredMechs = GSS_C_NULL_OID_SET;
   ```

GSS_C_NULL_OID_SET 表示应当使用缺省机制。

2. 将服务名称转换为 GSS-API 格式。

   由于 gss_acquire_cred() 会接受 gss_name_t 结构形式的服务名称作为参数，因此必须将服务名称转换为这种格式。可使用 gss_import_name() 函数执行此转换。与所有 GSS-API 函数一样，此函数要求参数采用 GSS-API 类型，因此必须首先将服务名称复制到 GSS-API 缓冲区，如下所示：

```
name_buf.value = service_name;
name_buf.length = strlen(name_buf.value) + 1;
maj_stat = gss_import_name(&min_stat, &name_buf,
           (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &server_name);
if (maj_stat != GSS_S_COMPLETE) {
     display_status("importing name", maj_stat, min_stat);
     if (mechOid != GSS_C_NO_OID)
           gss_release_oid(&min_stat, &mechOid);
     return -1;
}
```

   另请注意非标准函数 gss_release_oid() 的用法。

   输入是 *name_buf* 中字符串形式的服务名称。输出是一个指向 gss_name_t 结构 *server_name* 的指针。第三个参数 GSS_C_NT_HOSTBASED_SERVICE 是 *name_buf* 中字符串的名称类型。在这种情况下，名称类型表示应当将字符串解释为 *service@host* 格式的服务。

执行这些任务之后，服务器程序可以调用 gss_acquire_cred()：

```
maj_stat = gss_acquire_cred(&min_stat, server_name, 0,
                            desiredMechs, GSS_C_ACCEPT,
                            server_creds, NULL, NULL);
```

- *min_stat* 是函数返回的错误代码。
- *server_name* 是服务器的名称。
- 0 表示程序无需知道凭证的最长生命周期。
- *desiredMechs* 是一组应用该凭证的机制。
- GSS_C_ACCEPT 表示凭证只能用于接受安全上下文。
- *server_creds* 是该函数返回的凭证句柄。
- NULL，NULL 表示该程序无需知道所使用的特定机制或凭证将保持有效的时间长度。

以下源代码说明了 server_acquire_creds() 函数。

示例 6–2 server_acquire_creds() 函数的样例代码

```
/*
 * Function: server_acquire_creds
 *
 * Purpose: imports a service name and acquires credentials for it
 *
 * Arguments:
 *
```

示例 6–2 server_acquire_creds() 函数的样例代码 （续）

```
 *      service_name    (r) the ASCII service name
        mechType        (r) the mechanism type to use
 *      server_creds    (w) the GSS-API service credentials
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * The service name is imported with gss_import_name, and service
 * credentials are acquired with gss_acquire_cred.  If either operation
 * fails, an error message is displayed and -1 is returned; otherwise,
 * 0 is returned.
 */
int server_acquire_creds(service_name, mechOid, server_creds)
     char *service_name;
     gss_OID mechOid;
     gss_cred_id_t *server_creds;
{
     gss_buffer_desc name_buf;
     gss_name_t server_name;
     OM_uint32 maj_stat, min_stat;
     gss_OID_set_desc mechOidSet;
     gss_OID_set desiredMechs = GSS_C_NULL_OID_SET;

     if (mechOid != GSS_C_NULL_OID) {
               desiredMechs = &mechOidSet;
               mechOidSet.count = 1;
               mechOidSet.elements = mechOid;
     } else
               desiredMechs = GSS_C_NULL_OID_SET;

     name_buf.value = service_name;
     name_buf.length = strlen(name_buf.value) + 1;
     maj_stat = gss_import_name(&min_stat, &name_buf,
               (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &server_name);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("importing name", maj_stat, min_stat);
          if (mechOid != GSS_C_NO_OID)
                gss_release_oid(&min_stat, &mechOid);
          return -1;
     }

     maj_stat = gss_acquire_cred(&min_stat, server_name, 0,
                                 desiredMechs, GSS_C_ACCEPT,
                                 server_creds, NULL, NULL);

     if (maj_stat != GSS_S_COMPLETE) {
          display_status("acquiring credentials", maj_stat, min_stat);
          return -1;
     }

     (void) gss_release_name(&min_stat, &server_name);

     return 0;
}
```

检查 inetd

获取服务的凭证之后，可以使用 gss-server 来查看用户是否指定了 inetd。main 函数按如下方式检查 inetd：

```
if (do_inetd) {
     close(1);
     close(2);
```

如果用户指定了使用 inetd，则程序将关闭标准输出和标准错误。然后，gss-server 会针对标准输入调用 sign_server()，inetd 将使用该函数传递连接。否则，gss-server 会创建一个套接字，使用 TCP 函数 accept() 接受该套接字的连接，然后针对 accept() 返回的文件描述符调用 sign_server()。

如果未使用 inetd，程序会创建连接和上下文，直到终止程序为止。但是，如果用户指定了 -once 选项，则循环将在首次连接之后终止。

从客户机接收数据

检查 inetd 之后，gss-server 程序会调用 sign_server()，该函数用于执行程序的主要工作。sign_server() 首先通过调用 server_establish_context() 来建立上下文。

sign_server() 可执行以下任务：

- 接受上下文
- 展开数据
- 对数据进行签名
- 返回数据

这些任务将在以下各节中进行介绍。以下源代码说明了 sign_server() 函数。

示例 6–3 sign_server() 函数

```
int sign_server(s, server_creds)
     int s;
     gss_cred_id_t server_creds;
{
     gss_buffer_desc client_name, xmit_buf, msg_buf;
     gss_ctx_id_t context;
     OM_uint32 maj_stat, min_stat;
     int i, conf_state, ret_flags;
     char    *cp;

     /* Establish a context with the client */
     if (server_establish_context(s, server_creds, &context,
                  &client_name, &ret_flags) < 0)
     return(-1);

     printf("Accepted connection: \"%.*s\"\n",
         (int) client_name.length, (char *) client_name.value);
```

示例 6–3 sign_server() 函数 （续）

```
    (void) gss_release_buffer(&min_stat, &client_name);

    for (i=0; i < 3; i++)
        if (test_import_export_context(&context))
            return -1;

    /* Receive the sealed message token */
    if (recv_token(s, &xmit_buf) < 0)
        return(-1);

    if (verbose && log) {
        fprintf(log, "Sealed message token:\n");
        print_token(&xmit_buf);
    }

    maj_stat = gss_unwrap(&min_stat, context, &xmit_buf, &msg_buf,
                &conf_state, (gss_qop_t *) NULL);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("unsealing message", maj_stat, min_stat);
        return(-1);
    } else if (! conf_state) {
        fprintf(stderr, "Warning!  Message not encrypted.\n");
    }

    (void) gss_release_buffer(&min_stat, &xmit_buf);

    fprintf(log, "Received message: ");
    cp = msg_buf.value;
    if ((isprint(cp[0]) || isspace(cp[0])) &&
        (isprint(cp[1]) || isspace(cp[1]))) {
        fprintf(log, "\"%.*s\"\n", msg_buf.length, msg_buf.value);
    } else {
        printf("\n");
        print_token(&msg_buf);
    }

    /* Produce a signature block for the message */
    maj_stat = gss_get_mic(&min_stat, context, GSS_C_QOP_DEFAULT,
                &msg_buf, &xmit_buf);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("signing message", maj_stat, min_stat);
        return(-1);
    }

    (void) gss_release_buffer(&min_stat, &msg_buf);

    /* Send the signature block to the client */
    if (send_token(s, &xmit_buf) < 0)
        return(-1);

    (void) gss_release_buffer(&min_stat, &xmit_buf);

    /* Delete context */
    maj_stat = gss_delete_sec_context(&min_stat, &context, NULL);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("deleting context", maj_stat, min_stat);
```

示例 6–3 sign_server() 函数 （续）

```
        return(-1);
    }

    fflush(log);

    return(0);
}
```

接受上下文

建立上下文通常涉及到在客户机和服务器之间执行一系列令牌交换。为了保持程序的可移植性，应当在循环中同时执行上下文接受和上下文初始化。用于接受上下文的循环与用于建立上下文的循环非常相似，但是二者的方向相反。请与第 91 页中的 “建立与服务器的安全上下文”进行比较。

以下源代码说明了 server_establish_context() 函数。

示例 6–4 server_establish_context() 函数

```
/*
 * Function: server_establish_context
 *
 * Purpose: establishes a GSS-API context as a specified service with
 * an incoming client, and returns the context handle and associated
 * client name
 *
 * Arguments:
 *
 *      s               (r) an established TCP connection to the client
 *      service_creds   (r) server credentials, from gss_acquire_cred
 *      context         (w) the established GSS-API context
 *      client_name     (w) the client's ASCII name
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * Any valid client request is accepted.  If a context is established,
 * its handle is returned in context and the client name is returned
 * in client_name and 0 is returned.  If unsuccessful, an error
 * message is displayed and -1 is returned.
 */
int server_establish_context(s, server_creds, context, client_name, ret_flags)
     int s;
     gss_cred_id_t server_creds;
     gss_ctx_id_t *context;
     gss_buffer_t client_name;
     OM_uint32 *ret_flags;
{
     gss_buffer_desc send_tok, recv_tok;
     gss_name_t client;
     gss_OID doid;
```

示例 6–4 server_establish_context() 函数 （续）

```
OM_uint32 maj_stat, min_stat, acc_sec_min_stat;
gss_buffer_desc    oid_name;

*context = GSS_C_NO_CONTEXT;

do {
     if (recv_token(s, &recv_tok) < 0)
          return -1;

     if (verbose && log) {
         fprintf(log, "Received token (size=%d): \n", recv_tok.length);
         print_token(&recv_tok);
     }

     maj_stat =
          gss_accept_sec_context(&acc_sec_min_stat,
                                 context,
                                 server_creds,
                                 &recv_tok,
                                 GSS_C_NO_CHANNEL_BINDINGS,
                                 &client,
                                 &doid,
                                 &send_tok,
                                 ret_flags,
                                 NULL,     /* ignore time_rec */
                                 NULL);    /* ignore del_cred_handle */

     (void) gss_release_buffer(&min_stat, &recv_tok);

     if (send_tok.length != 0) {
          if (verbose && log) {
               fprintf(log,
                     "Sending accept_sec_context token (size=%d):\n",
                     send_tok.length);
               print_token(&send_tok);
          }
          if (send_token(s, &send_tok) < 0) {
               fprintf(log, "failure sending token\n");
               return -1;
          }

          (void) gss_release_buffer(&min_stat, &send_tok);
     }
     if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
          display_status("accepting context", maj_stat,
                         acc_sec_min_stat);
          if (*context == GSS_C_NO_CONTEXT)
                  gss_delete_sec_context(&min_stat, context,
                                         GSS_C_NO_BUFFER);
          return -1;
     }

     if (verbose && log) {
         if (maj_stat == GSS_S_CONTINUE_NEEDED)
             fprintf(log, "continue needed...\n");
         else
```

示例 6–4 server_establish_context() 函数 （续）

```
                fprintf(log, "\n");
            fflush(log);
        }
    } while (maj_stat == GSS_S_CONTINUE_NEEDED);

    /* display the flags */
    display_ctx_flags(*ret_flags);

    if (verbose && log) {
        maj_stat = gss_oid_to_str(&min_stat, doid, &oid_name);
        if (maj_stat != GSS_S_COMPLETE) {
            display_status("converting oid->string", maj_stat, min_stat);
            return -1;
        }
        fprintf(log, "Accepted connection using mechanism OID %.*s.\n",
                (int) oid_name.length, (char *) oid_name.value);
        (void) gss_release_buffer(&min_stat, &oid_name);
    }

    maj_stat = gss_display_name(&min_stat, client, client_name, &doid);
    if (maj_stat != GSS_S_COMPLETE) {
         display_status("displaying name", maj_stat, min_stat);
         return -1;
    }
    maj_stat = gss_release_name(&min_stat, &client);
    if (maj_stat != GSS_S_COMPLETE) {
         display_status("releasing name", maj_stat, min_stat);
         return -1;
    }
    return 0;
}
```

sign_server() 函数使用以下源代码，通过调用 server_establish_context() 来接受上下文。

```
/* Establish a context with the client */
    if (server_establish_context(s, server_creds, &context,
                &client_name, &ret_flags) < 0)
   return(-1);
```

server_establish_context() 函数首先查找客户机在初始化上下文的过程中发送的令牌。由于 GSS-API 本身不会收发令牌，因此程序必须各自具有用于执行这些任务的例程。服务器使用 recv_token() 来接收令牌：

```
    do {
         if (recv_token(s, &recv_tok) < 0)
              return -1;
```

接下来，server_establish_context() 会调用 GSS-API 函数 gss_accept_sec_context()：

```
maj_stat = gss_accept_sec_context(&min_stat,
                                  context,
                                  server_creds,
                                  &recv_tok,
                                  GSS_C_NO_CHANNEL_BINDINGS,
                                  &client,
                                  &doid,
                                  &send_tok,
                                  ret_flags,
                                  NULL,     /* ignore time_rec */
                                  NULL);    /* ignore del_cred_handle */
```

- *min_stat* 是基础机制返回的错误状态。
- *context* 是所建立的上下文。
- *server_creds* 是要提供的服务的凭证（请参见第 104 页中的 “获取凭证”）。
- *recv_tok* 是 `recv_token()` 从客户机接收的令牌。
- `GSS_C_NO_CHANNEL_BINDINGS` 是一个标志，表示不使用通道绑定（请参见第 73 页中的 “在 GSS-API 中使用通道绑定”）。
- *client* 是客户机的 ASCII 名称。
- *oid* 表示机制（采用 OID 格式）。
- *send_tok* 是要发送到客户机的令牌。
- *ret_flags* 是用于指明上下文是否支持指定选项的各种标志，如 `message-sequence-detection`。
- 两个 NULL 参数表示程序无需知道上下文将保持有效的时间长度或者服务器是否可以充当客户机的代理。

只要 `gss_accept_sec_context()` 将 *maj_stat* 设置为 `GSS_S_CONTINUE_NEEDED`，接受循环就将继续并且不会遇到任何错误。如果 *maj_stat* 与该值或者 `GSS_S_COMPLETE` 不等，则表明存在问题，循环将退出。

无论是否存在要发送回客户机的令牌，`gss_accept_sec_context()` 都会返回一个表示 *send_tok* 长度的正值。下一步是查看是否存在要发送的令牌，如果存在，则发送该令牌：

```
if (send_tok.length != 0) {
     . . .
     if (send_token(s, &send_tok) < 0) {
          fprintf(log, "failure sending token\n");
          return -1;
     }

     (void) gss_release_buffer(&min_stat, &send_tok);
     }
```

展开消息

接受上下文之后，sign_server() 将接收客户机已发送的消息。由于 GSS-API 不提供用于接收令牌的函数，因此程序将使用 recv_token() 函数：

```
if (recv_token(s, &xmit_buf) < 0)
     return(-1);
```

由于消息可能已经过加密，因此程序将使用 GSS-API 函数 gss_unwrap() 来展开消息：

```
maj_stat = gss_unwrap(&min_stat, context, &xmit_buf, &msg_buf,
                           &conf_state, (gss_qop_t *) NULL);
     if (maj_stat != GSS_S_COMPLETE) {
        display_status("unwrapping message", maj_stat, min_stat);
        return(-1);
     } else if (! conf_state) {
        fprintf(stderr, "Warning!  Message not encrypted.\n");
     }

     (void) gss_release_buffer(&min_stat, &xmit_buf);
```

gss_unwrap() 会提取 recv_token() 已经放在 *xmit_buf* 中的消息，转换该消息并将结果放在 *msg_buf* 中。请注意 gss_unwrap() 的两个参数。*conf_state* 标志用于指明是否已经向该消息应用了保密性（即加密）。最后一个 NULL 表示程序无需知道用于保护该消息的 QOP。

消息的签名和返回

此时，sign_server() 函数需要对消息进行签名。对消息进行签名需要将消息的消息完整性代码（即 MIC）返回到客户机。返回消息可证明消息已成功发送和展开。为了获取 MIC，sign_server() 将使用 gss_get_mic() 函数：

```
maj_stat = gss_get_mic(&min_stat, context, GSS_C_QOP_DEFAULT,
                            &msg_buf, &xmit_buf);
```

gss_get_mic() 在 *msg_buf* 中查找该消息，生成 MIC，并将该 MIC 存储到 *xmit_buf* 中。然后，服务器会使用 send_token() 将该 MIC 发回到客户机。客户机会使用 gss_verify_mic() 来验证该 MIC。请参见第 98 页中的 “读取和验证 GSS-API 客户机中的签名块”。

最后，sign_server() 会执行一些清除操作。sign_server() 会使用 gss_release_buffer() 来释放 GSS-API 缓冲区 *msg_buf* 和 *xmit_buf*。然后，sign_server() 会使用 gss_delete_sec_context() 来销毁上下文。

使用 test_import_export_context() 函数

通过 GSS-API 可导出和导入上下文。使用这些活动可以在多进程程序中的不同进程之间共享上下文。sign_server() 包含一个概念证明函数 test_import_export_context()，用于说明如何导出和导入上下文。test_import_export_context() 不在进程之间传递上下文。test_import_export_context() 而是会显示导出所需的时间，然后导入上下文。虽然 test_import_export_context() 是一个虚构的函数，但可用于指示如何使用 GSS-API 导入和导出函数。test_import_export_context() 还可以指示如何使用时间戳来处理上下文。

test_import_export_context() 的源代码如以下示例所示。

示例 6–5 test_import_export_context()

```
int test_import_export_context(context)
        gss_ctx_id_t *context;
{
        OM_uint32       min_stat, maj_stat;
        gss_buffer_desc context_token, copied_token;
        struct timeval tm1, tm2;

        /*
         * Attempt to save and then restore the context.
         */
        gettimeofday(&tm1, (struct timezone *)0);
        maj_stat = gss_export_sec_context(&min_stat, context, &context_token);
        if (maj_stat != GSS_S_COMPLETE) {
                display_status("exporting context", maj_stat, min_stat);
                return 1;
        }
        gettimeofday(&tm2, (struct timezone *)0);
        if (verbose && log)
                fprintf(log, "Exported context: %d bytes, %7.4f seconds\n",
                        context_token.length, timeval_subtract(&tm2, &tm1));
        copied_token.length = context_token.length;
        copied_token.value = malloc(context_token.length);
        if (copied_token.value == 0) {
            fprintf(log, "Couldn't allocate memory to copy context token.\n");
            return 1;
        }
        memcpy(copied_token.value, context_token.value, copied_token.length);
        maj_stat = gss_import_sec_context(&min_stat, &copied_token, context);
        if (maj_stat != GSS_S_COMPLETE) {
                display_status("importing context", maj_stat, min_stat);
                return 1;
        }
        free(copied_token.value);
        gettimeofday(&tm1, (struct timezone *)0);
        if (verbose && log)
                fprintf(log, "Importing context: %7.4f seconds\n",
                        timeval_subtract(&tm1, &tm2));
        (void) gss_release_buffer(&min_stat, &context_token);
        return 0;
}
```

在 GSSAPI 服务器示例中清除

返回到 main() 函数中，应用程序会使用 gss_release_cred() 来删除服务凭证。如果已经为机制指定了 OID，则程序会使用 gss_release_oid() 删除该 OID 并退出。

```
(void) gss_release_cred(&min_stat, &server_creds);
```

◆ ◆ ◆ 第 7 章

7 编写使用 SASL 的应用程序

SASL（Simple Authentication and Security Layer，简单验证和安全层）是一个安全框架。SASL（发音为 "sassel"）为基于连接的协议提供验证服务以及完整性和保密性服务（可选）。本章包含以下主题：

简单验证和安全层 (Simple Authentication and Security Layer, SASL) 介绍

SASL 为应用程序和共享库的开发者提供了用于验证、数据完整性检查和加密的机制。开发者可通过 SASL 对通用 API 进行编码。此方法避免了对特定机制的依赖性。SASL 特别适用于使用 IMAP、SMTP、ACAP 和 LDAP 协议的应用程序，因为这些协议全都支持 SASL。RFC 2222 中对 SASL 进行了介绍。

SASL 库基础

SASL 库称为 `libsasl`。`libsasl` 是一个框架，允许正确编写的 SASL 使用者应用程序使用系统中可用的所有 SASL 插件。术语**插件**是指为 SASL 提供服务的对象。插件位于 `libsasl` 的外部。SASL 插件可用于验证和安全性、名称标准化以及辅助属性（如口令）的查找。加密算法存储在插件中，而不是 `libsasl` 中。

`libsasl` 为使用者应用程序和库提供应用编程接口 (Application Programming Interface, API)。服务提供者接口 (Service Provider Interface, SPI) 是为插件提供的，用于为 `libsasl` 提供服务。`libsasl` 不能识别网络或协议。相应地，应用程序必须负责在客户机与服务器之间发送和接收数据。

SASL 对用户使用两个重要的标识符。**验证** *ID* (`authid`) 是用于验证用户的用户 ID。验证 ID 授予用户系统访问权限。**授权** *ID* (`userid`) 用于检查是否允许用户使用特定选项。

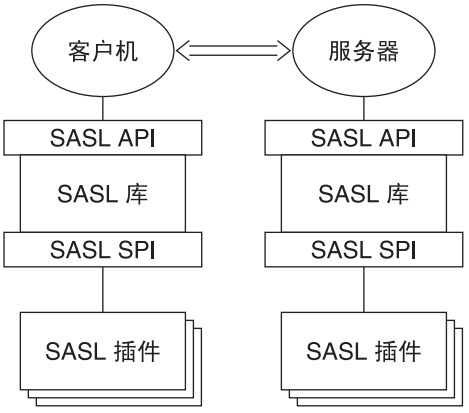
SASL 客户机应用程序和 SASL 服务器应用程序将协商公用的 SASL 机制和安全级别。通常，SASL 服务器应用程序会将其可接受的验证机制的列表发送给客户机。随后 SASL 客户机应用程序便可决定哪种验证机制最能满足其要求。此后，客户机与服务器使用双方同意的验证机制，对它们之间交换的由 SASL 提供的验证数据进行验证。此交换将持续下去，直到验证成功完成、失败或被客户机或服务器中止。

在验证过程中，SASL 验证机制可以协商安全层。如果已选择安全层，则必须在 SASL 会话期间使用该层。

SASL 体系结构

下图显示了基本的 SASL 体系结构。

图 7–1 SASL 体系结构



客户机与服务器应用程序通过 SASL API 调用 `libsasl` 的本地副本。`libsasl` 通过 SASL 服务提供者接口 (Service Provider Interface, SPI) 与 SASL 机制进行通信。

安全机制

安全机制插件为 `libsasl` 提供安全服务。以下是安全机制提供的一些典型功能：

- 在客户端进行验证
- 在服务器端进行验证
- 完整性，即检查传输的数据是否保持不变
- 保密性，即对传输的数据进行加密和解密

SASL 安全强度因子

SSF（即安全强度因子）指示 SASL 保护的强度。如果该机制支持安全层，则客户机与服务器会协商 SSF。SSF 的值基于执行 SASL 协商之前指定的安全属性。如果协商结果是非零 SSF，则验证完成后，客户机和服务器都需要使用该机制的安全层。SSF 由具有以下值之一的整数表示：

- 0 － 无保护。
- 1 － 仅限于完整性检查。
- >1 － 支持验证、完整性和保密性。数字表示加密密钥长度。

保密性和完整性操作是通过安全机制执行的。`libsasl` 可以协调这些请求。

注 – 在协商过程中，SASL 客户机会选择具有最大 SSF 的机制。但是，实际所选的 SASL 机制可能随后会协商较小的 SSF。

SASL 中的通信

应用程序通过 `libsasl` API 与 `libsasl` 进行通信。`libsasl` 可通过应用程序注册的回调方式请求其他信息。应用程序不会直接调用插件，而只是通过 `libsasl` 进行调用。一般情况下，插件会调用 `libsasl` 框架的插件，随后这些插件调用应用程序的回调。SASL 插件还可以直接调用应用程序，不过应用程序不知道调用来自插件还是来自 `libsasl`。

回调在以下几个方面非常有用。

- `libsasl` 可以使用回调获取完成验证所需的信息。
- `libsasl` 使用者应用程序可以使用回调更改插件和配置数据的搜索路径、验证文件以及更改各种缺省行为。
- 服务器可以使用回调来更改授权策略、提供不同的口令验证方法以及获取口令更改信息。
- 客户机和服务器可以使用回调来指定错误消息的语言。

应用程序可以注册两种回调：全局回调和会话回调。此外，`libsasl` 定义了大量用于为不同种类的回调注册的回调标识符。如果未注册给定类型的回调，则 `libsasl` 将执行缺省操作。

会话回调将覆盖全局回调。如果为给定 ID 指定了会话回调，则不会为该会话调用全局回调。某些回调必须是全局的，因为这些回调发生在会话之外。以下实例要求使用全局回调：

- 确定要装入的插件的搜索路径
- 验证插件
- 定位配置数据
- 记录错误消息
- 对 `libsasl` 或其插件的其他全局配置

可以使用给定 SASL 回调 ID 的 NULL 回调函数来注册 SASL 回调。NULL 回调函数指示配备客户机的目的是为了提供所需的数据。所有的 SASL 回调 ID 都以前缀 SASL_CB_ 开头。

SASL 提供以下可供客户机或服务器使用的回调：

SASL_CB_GETOPT
: 获取 SASL 选项。选项用于修改 libsasl(3LIB) 和相关插件的行为。可由客户机或服务器使用。

SASL_CB_LOG
: 设置 libsasl 及其插件的日志记录函数。缺省行为是使用 syslog。

SASL_CB_GETPATH
: 获取以冒号分隔的 SASL 插件搜索路径列表。缺省路径取决于以下体系结构：
  - 32 位 SPARC 体系结构：/usr/lib/sasl
  - 32 位 x86 体系结构：/usr/lib/sasl
  - 64 位 SPARC 体系结构：/usr/lib/sasl/sparcv9
  - x64 体系结构：/usr/lib/sasl/amd64

SASL_CB_GETCONF
: 获取 SASL 服务器的配置目录的路径。缺省设置为 /etc/sasl。

SASL_CB_LANGUAGE
: 为客户机和服务器错误消息和客户机提示指定以逗号分隔的 RFC 1766 语言代码列表（按优先级顺序）。缺省设置为 i-default。

SASL_CB_VERIFYFILE
: 验证配置文件和插件文件。

SASL 提供以下仅限客户机使用的回调：

SASL_CB_USER
: 获取客户机用户名。用户名与授权 ID 相同。LOGNAME 环境变量为缺省设置。

SASL_CB_AUTHNAME
: 获取客户机验证名称。

SASL_CB_PASS
: 获取基于客户机口令短语的机密。

SASL_CB_ECHOPROMPT
: 获取给定质询提示的结果。可以回显来自客户机的输入。

SASL_CB_NOECHOPROMPT
: 获取给定质询提示的结果。不应回显来自客户机的输入。

SASL_CB_GETREALM
: 设置用于验证的领域。

SASL 提供以下仅限服务器使用的回调：

SASL_CB_PROXY_POLICY
: 检查是否授权经过验证的用户代表指定用户执行操作。如果未注册此回调，则经过验证的用户与要授权的用户必须是同一用户。如果这些 ID 不同，则验证将失败。请使用服务器应用程序来维护非标准授权策略。

SASL_CB_SERVER_USERDB_CHECKPASS
　针对调用方提供的用户数据库验证纯文本口令。

SASL_CB_SERVER_USERDB_SETPASS
　在用户数据库中存储纯文本口令。

SASL_CB_CANON_USER
　调用应用程序提供的用户标准化函数。

首次初始化 SASL 库时，服务器和客户机会声明所有必要的全局回调。执行 SASL 会话之前或期间可以使用全局回调。初始化之前，回调将执行诸如装入插件、记录数据和读取配置文件之类的任务。SASL 会话开始时，可以声明其他回调。这类回调可以根据需要覆盖全局回调。

SASL 连接上下文

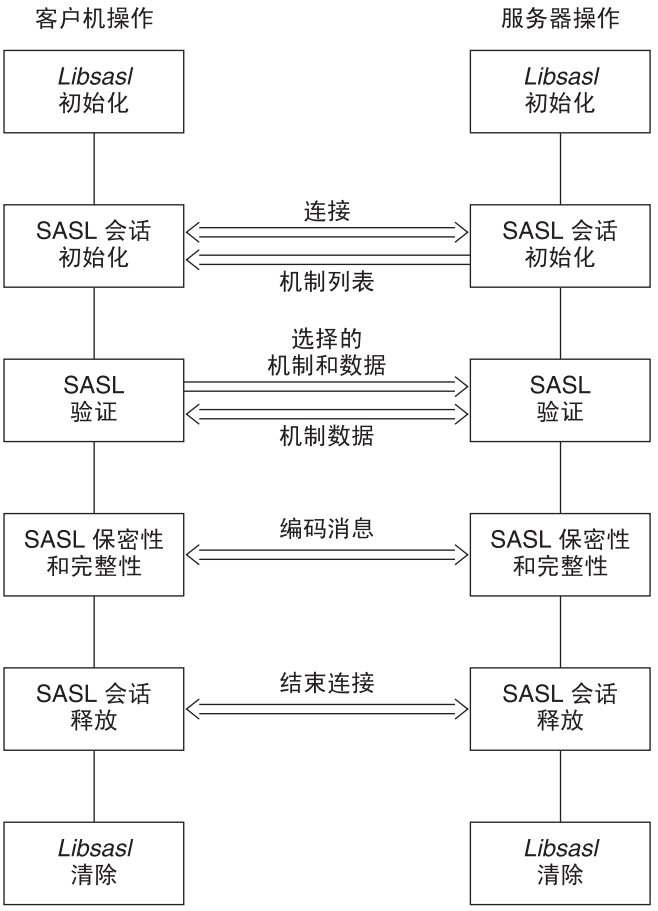
`libsasl` 使用 SASL 连接**上下文**维护 SASL 客户机和 SASL 服务器的每个 SASL 会话的状态。每个上下文一次只能用于一个验证和安全会话。维护的状态包括以下信息：

- 连接信息，如服务、命名和地址信息以及协议标志
- 特定于连接的回调
- 用于协商 SASL SSF 的安全属性
- 验证状态以及安全层信息

SASL 周期中的步骤

下图显示了 SASL 生命周期中的步骤。客户机操作显示在图的左侧，服务器操作显示在右侧。中间的箭头显示客户机与服务器之间通过外部连接执行的交互操作。

图 7–2 SASL生命周期



以下各节说明了生命周期中的步骤。

libsasl 初始化

客户机调用 sasl_client_init() 来初始化 libsasl 以供客户机使用。服务器调用 sasl_server_init() 来初始化 libsasl 以供服务器使用。

运行 sasl_client_init() 时，将装入 SASL 客户机、该客户机的机制以及该客户机的标准化插件。同样，调用 sasl_server_init() 时，将装入 SASL 服务器、该服务器的机制、该服务器的标准化插件以及该服务器的 auxprop 插件。调用 sasl_client_init() 后，可以使用 sasl_client_add_plugin() 和 sasl_canonuser_add_plugin() 来添加其他客户机插件。在服务器端，调用 sasl_server_init() 后，可以通过

`sasl_server_add_plugin()`、`sasl_canonuser_add_plugin()` 和 `sasl_auxprop_add_plugin()` 来添加其他服务器插件。依据体系结构，我们在 Solaris 软件的以下目录中提供了 SASL 机制：

- 32 位 SPARC 体系结构：`/usr/lib/sasl`
- 32 位 x86 体系结构：`/usr/lib/sasl`
- 64 位 SPARC 体系结构：`/usr/lib/sasl/sparcv9`
- x64 体系结构：`/usr/lib/sasl/amd64`

可以使用 SASL_CB_GETPATH 回调覆盖缺省位置。

此时，可以设置所有必需的全局回调。SASL 客户机和服务器可能包括以下回调：

- SASL_CB_GETOPT
- SASL_CB_LOG
- SASL_CB_GETPATH
- SASL_CB_VERIFYFILE

此外，SASL 服务器还可能包括 SASL_CB_GETCONF 回调。

SASL 会话初始化

服务器和客户机通过协议建立连接。要使用 SASL 执行验证，服务器和客户机可以分别使用 `sasl_server_new()` 和 `sasl_client_new()` 来创建 SASL 连接上下文。SASL 客户机和服务器可以使用 `sasl_setprop()` 来设置对机制强制执行安全限制的属性。此方法使 SASL 使用者应用程序可以决定指定 SASL 连接上下文的最小 SSF、最大 SSF 和安全属性。

```
#define SASL_SEC_NOPLAINTEXT            0x0001
#define SASL_SEC_NOACTIVE               0x0002
#define SASL_SEC_NODICTIONARY           0x0004
#define SASL_SEC_FORWARD_SECRECY        0x0008
#define SASL_SEC_NOANONYMOUS            0x0010
#define SASL_SEC_PASS_CREDENTIALS       0x0020
#define SASL_SEC_MUTUAL_AUTH            0x0040
```

注 – 验证和安全层可由客户机/服务器协议提供，也可由 `libsasl` 外部的某些其他机制提供。在这类情况下，可以使用 `sasl_setprop()` 来设置外部验证 ID 或外部 SSF。例如，请考虑协议对服务器结合使用 SSL 和客户机验证的情况。在这种情况下，外部验证标识可以为客户机的主题名称。外部 SSF 可以为密钥大小。

对于服务器，`libsasl` 可以依据安全属性和外部 SSF 来确定可用的 SASL 机制。客户机可以通过协议从 SASL 服务器获取可用的 SASL 机制。

为使 SASL 服务器可以创建 SASL 连接上下文，服务器应该调用 `sasl_server_new()`。可以重复使用不再使用的现有 SASL 连接上下文。但是，可能需要重置以下参数：

```
#define SASL_DEFUSERREALM 3     /* default realm passed to server_new or set with setprop */
#define SASL_IPLOCALPORT 8      /* iplocalport string passed to server_new */
#define SASL_IPREMOTEPORT 9     /* ipremoteport string passed to server_new */
#define SASL_SERVICE    12      /* service passed to sasl_*_new */
#define SASL_SERVERFQDN 13      /* serverFQDN passed to sasl_*_new */
```

可以修改 `sasl_client_new()` 和 `sasl_server_new()` 的任何参数，但回调和协议标志除外。

服务器和客户机还可以使用 `sasl_setprop()` 来指定以下属性，从而建立安全策略并设置连接特定参数：

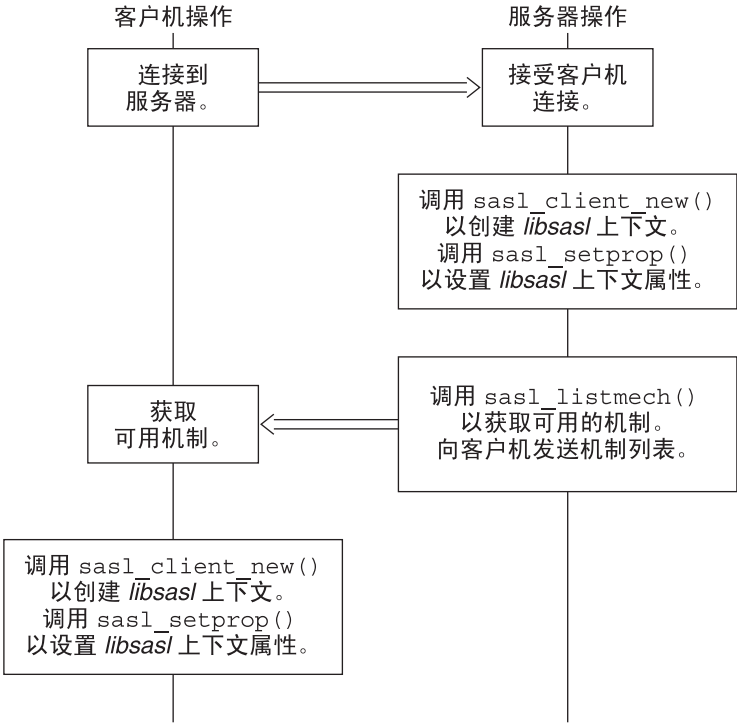
```
#define SASL_SSF_EXTERNAL 100 /* external SSF active (sasl_ssf_t *) */
#define SASL_SEC_PROPS 101 /* sasl_security_properties_t */
#define SASL_AUTH_EXTERNAL 102 /* external authentication ID (const char *)
 */
```

- SASL_SSF_EXTERNAL－用于设置强度因子，即密钥中的位数
- SASL_SEC_PROPS－用于定义安全策略
- SASL_AUTH_EXTERNAL－外部验证 ID

服务器可以调用 `sasl_listmech()` 来获取满足安全策略的可用 SASL 机制的列表。一般情况下，客户机可以采用与协议相关的方式从服务器获取可用机制的列表。

下图说明了 SASL 会话的初始化过程。在此图和后续的图中，为了简单起见，省略了通过协议进行传输后的数据检查。

图 7–3 SASL 会话初始化



SASL 验证

验证根据使用的安全机制采用不同数量的客户机和服务器步骤。SASL 客户机将调用 `sasl_client_start()` 和要使用的安全机制列表。此列表通常来自服务器。`libsasl` 将依据可用机制和客户机的安全策略选择要用于此 SASL 会话的最佳机制。客户机的安全策略控制允许的机制。选定的机制由 `sasl_client_start()` 返回。有时，客户机的安全机制需要其他验证信息。对于已注册的回调，`libsasl` 将调用指定的回调，除非回调函数为 NULL。如果回调函数为 NULL，则 `libsasl` 将返回 SASL_INTERACT 和对所需信息的请求。如果返回 SASL_INTERACT，则应使用请求的信息调用 `sasl_client_start()`。

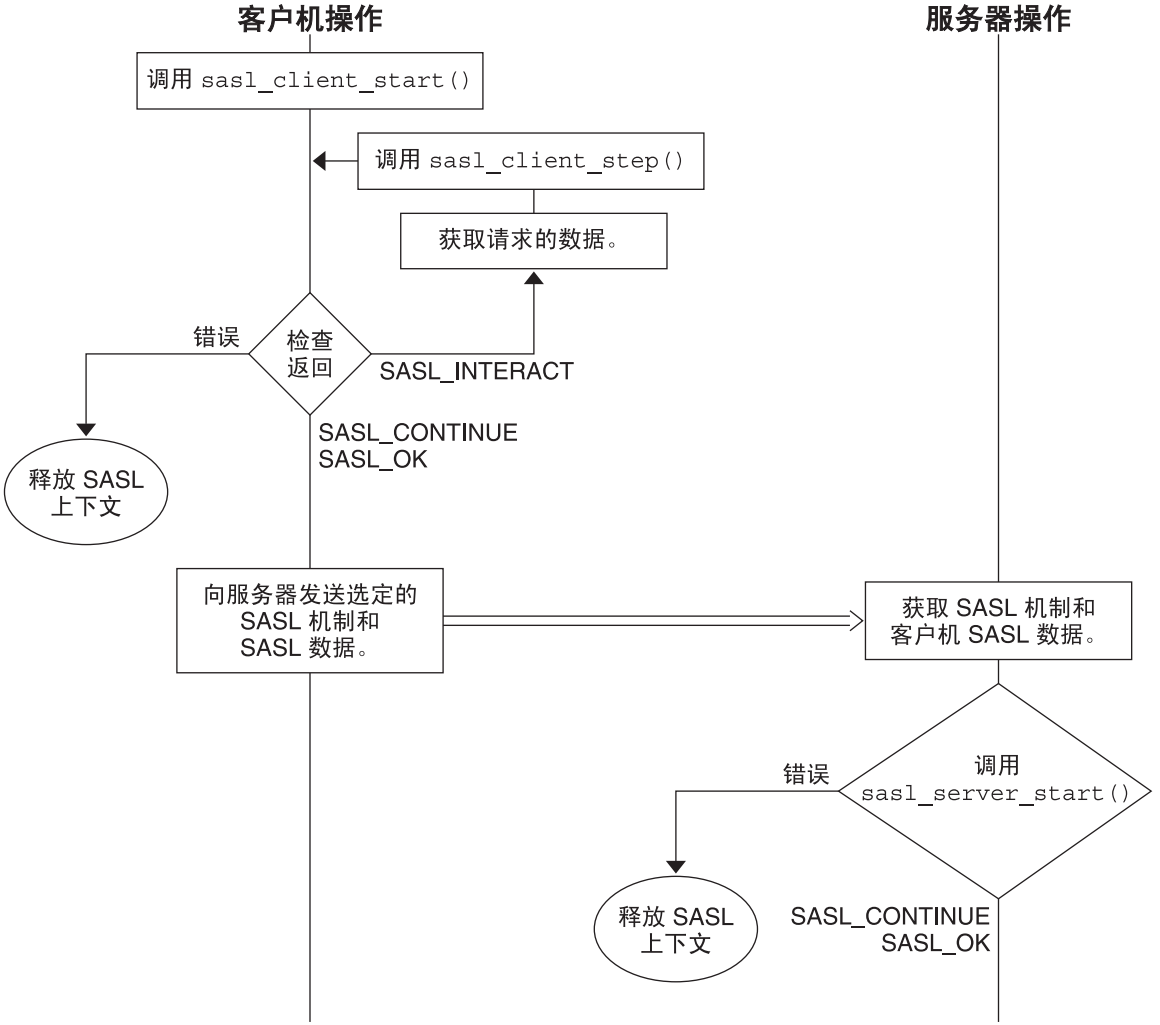
如果 `sasl_client_start()` 返回 SASL_CONTINUE 或 SASL_OK，则客户机应向服务器发送包含生成的所有验证数据的选定机制。如果返回任何其他值，则表示出现了错误。例如，任何机制可能都不可用。

服务器将接收客户机选定的机制，以及所有的验证数据。随后，服务器将调用 `sasl_server_start()` 来初始化此会话的机制数据。`sasl_server_start ()` 还将处理所有的验证数据。如果 `sasl_server_start()` 返回 SASL_CONTINUE 或 SASL_OK，则服

务器将发送验证数据。如果 sasl_server_start() 返回任何其他值，则表示出现了错误，如不可接受的机制或验证失败。必须中止验证。应该释放或重复使用 SASL 上下文。

下图说明了此部分的验证过程。

图 7–4 SASL 验证：发送客户机数据

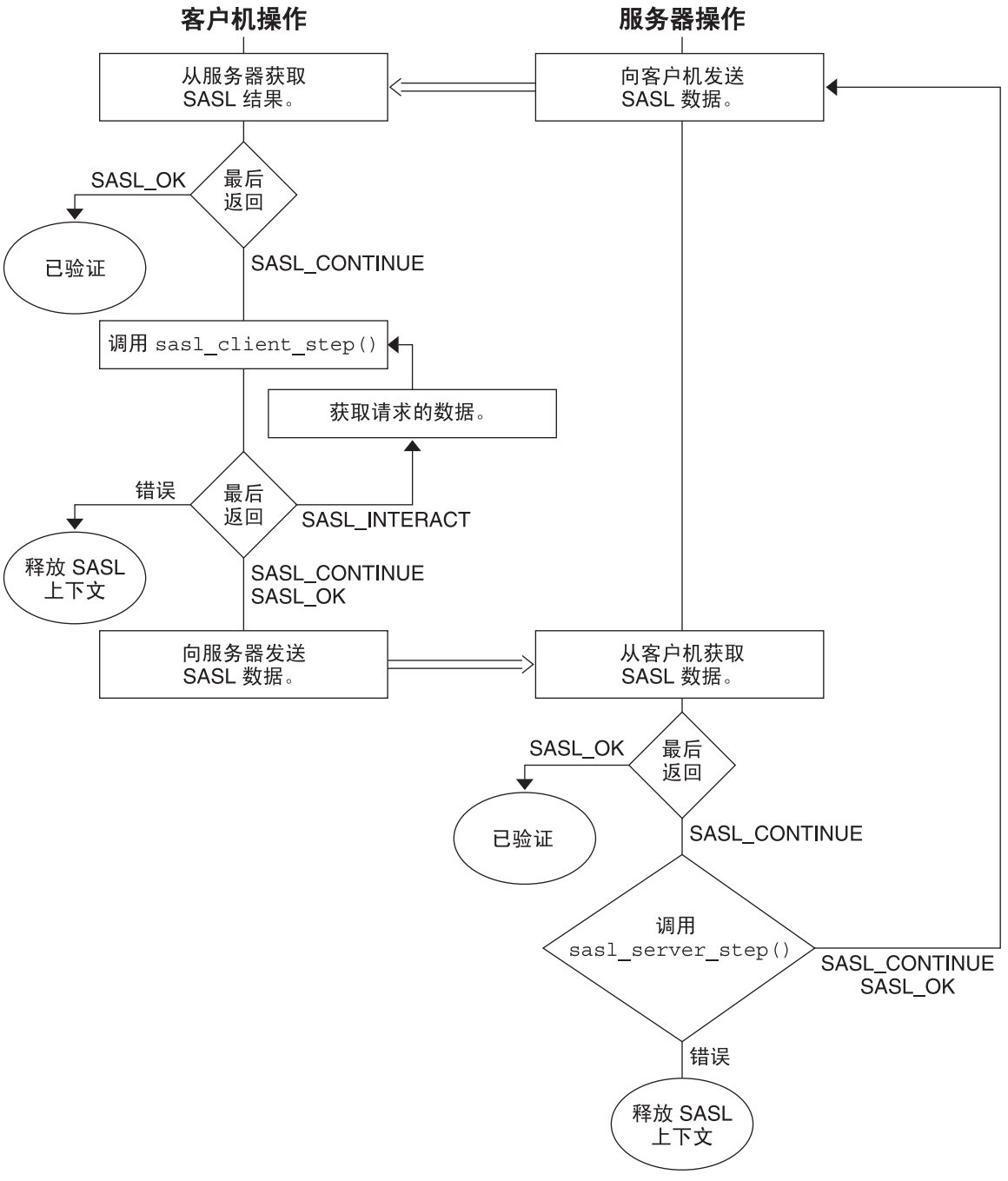


如果服务器对 sasl_server_start() 的调用返回 SASL_CONTINUE，则服务器将继续与客户机进行通信，以获取所有必要的验证信息。后续步骤的数目取决于机制。如果需

要，客户机可以调用 `sasl_client_step()` 来处理来自服务器的验证数据并生成回复。同样，服务器可以调用 `sasl_server_step()` 来处理来自客户机的验证并相应地生成回复。此交换将持续进行下去，直到验证完成或出现错误为止。如果返回 SASL_OK，则指示对客户机或服务器的验证已成功完成。SASL 机制可能仍然具有要发送给另一端的其他数据，因此另一端可以完成验证。在两端都完成验证后，服务器和客户机便可查询彼此的属性。

下图显示了服务器与客户机之间为传输其他验证数据所执行的交互。

图 7–5 SASL 验证：处理服务器数据



SASL 保密性和完整性

要检查安全层，请使用 sasl_getprop(3SASL) 函数查看安全强度因子 (Security Strength Factor, SSF) 的值是否大于 0。如果已协商安全层，则成功验证后客户机和服务器必须使用生成的 SSF。在客户机与服务器之间交换数据的方式与验证的方式相似。通过协议向客户机或服务器发送数据之前，将 sasl_encode() 应用于数据。在接收端，使用 sasl_decode() 对数据进行解码。如果未协商安全层，则无需 SASL 连接上下文。随后，即可处理或重复使用该上下文。

释放 SASL 会话

只有在不会重复使用会话时，才应释放 SASL 连接上下文。sasl_dispose() 将释放 SASL 连接上下文以及所有关联的资源和机制。调用 sasl_done() 之前，必须处理 SASL 连接上下文。sasl_done() 不负责释放 SASL 连接的上下文资源。请参见第 129 页中的 “libsasl 清除”。

释放 SASL 会话时，将通知关联的机制所有状态均可释放。只有在不会重复使用 SASL 会话时，才应释放该会话。否则，其他会话可以重复使用 SASL 状态。客户机和服务器都使用 sasl_dispose() 释放 SASL 连接上下文。

libsasl 清除

此步骤可释放 SASL 库和插件中的所有资源。客户机和服务器可以调用 sasl_done() 来释放 libsasl() 资源并卸载所有的 SASL 插件。sasl_done() 不会释放 SASL 连接上下文。请注意，如果应用程序同时为 SASL 客户机和 SASL 服务器，则 sasl_done() 将同时释放 SASL 客户机和 SASL 服务器资源。您不能仅释放客户机或服务器的资源。

注意 – 库不应调用 sasl_done()。应用程序在调用 sasl_done() 时务必要谨慎，避免与可能使用 libsasl 的任何库发生干扰。

SASL 示例

本节说明客户机应用程序与服务器应用程序之间的典型 SASL 会话。该示例包含以下步骤：

1. 客户机应用程序可以初始化 libsasl 并设置以下全局回调：
   - SASL_CB_GETREALM
   - SASL_CB_USER
   - SASL_CB_AUTHNAME
   - SASL_CB_PASS
   - SASL_CB_GETPATH
   - SASL_CB_LIST_END
2. 服务器应用程序可以初始化 libsasl 并设置以下全局回调：

- SASL_CB_LOG
- SASL_CB_LIST_END

3. 客户机将创建 SASL 连接上下文，设置安全属性并从服务器请求可用机制的列表。
4. 服务器将创建 SASL 连接上下文，设置安全属性，获取适当 SASL 机制的列表并向客户机发送该列表。
5. 客户机将接收可用机制的列表，选择一种机制，并向服务器发送所选择的机制以及所有验证数据。
6. 随后，客户机和服务器将交换 SASL 数据，直到验证和安全层协商完成为止。
7. 验证完成后，客户机和服务器将确定是否已协商安全层。客户机将对测试消息进行编码。然后，会将该消息发送给服务器。服务器也会确定经过验证的用户的用户名和该用户的领域。
8. 服务器将接收、解码和显示编码的消息。
9. 客户机将调用 `sasl_dispose()` 以释放客户机的 SASL 连接上下文。随后，客户机将调用 `sasl_done()` 以释放 `libsasl` 资源。
10. 服务器将调用 `sasl_dispose()` 以释放客户机连接上下文。

以下是客户机与服务器之间的对话。执行调用时，会显示对 `libsasl` 的每个调用。每次数据传输都由发送者和接收者指明。数据采用编码的形式显示，并在前面加上表示来源的字符：`C:` 表示来自于客户机，`S:` 表示来自于服务器。附录 D，SASL 示例的源代码中同时提供了两种应用程序的源代码。

客户机

```
% doc-sample-client
*** Calling sasl_client_init() to initialize libsasl for client use ***
*** Calling sasl_client_new() to create client SASL connection context ***
*** Calling sasl_setprop() to set sasl context security properties ***
Waiting for mechanism list from server...
```

服务器

```
% doc-sample-server digest-md5
*** Calling sasl_server_init() to initialize libsasl for server use ***
*** Calling sasl_server_new() to create server SASL connection context ***
*** Calling sasl_setprop() to set sasl context security properties ***
Forcing use of mechanism digest-md5
Sending list of 1 mechanism(s)
S: ZGlnZXN0LW1kNQ==
```

客户机

```
S: ZGlnZXN0LW1kNQ==
received 10 byte message
got 'digest-md5'
Choosing best mechanism from: digest-md5
*** Calling sasl_client_start() ***
Using mechanism DIGEST-MD5
Sending initial response...
C: RElHRVNULU1ENQ==
Waiting for server reply...
```

服务器

```
C: RElHRVNULU1ENQ==
got 'DIGEST-MD5'
*** Calling sasl_server_start() ***
Sending response...
S: bm9uY2U9IklicGxhRHJZNE4Z1gyVm5lQzl5MTZOYWxUOVcvanUrcmp5YmRqaHM\
sbT0iam0xMTQxNDIiLHFvcD0iYXV0aCxhdXRoLWludCxhdXRoLWNvbmYiLGNpcGhlcj0ic\
QwLHJjNC01NixyYzQiLG1heGJ1Zj0yMDQ4LGNoYXJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1k\
XNz
Waiting for client reply...
```

客户机

```
S: bm9uY2U9IklicGxhRHJZNE4Z1gyVm5lQzl5MTZOYWxUOVcvanUrcmp5YmRqaHM\
sbT0iam0xMTQxNDIiLHFvcD0iYXV0aCxhdXRoLWludCxhdXRoLWNvbmYiLGNpcGhlcj0ic\
QwLHJjNC01NixyYzQiLG1heGJ1Zj0yMDQ4LGNoYXJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1k\
XNz
received 171 byte message
got 'nonce="IbplaDrY4N4szhgX2VneC9y16NalT9W/ju+rjybdjhs=",\
realm="jm114142",qop="auth,auth-int,auth-conf",cipher="rc4-40,rc4-56,\
rc4",maxbuf=2048,charset=utf-8,algorithm=md5-sess'
*** Calling sasl_client_step() ***
Please enter your authorization name : zzzz
Please enter your authentication name : zzzz
Please enter your password : zz
*** Calling sasl_client_step() ***
Sending response...
C: dXNlcm5hbWU9Inp6enoiLHJlYWxtPSJqbTExNDE0MiIsbm9uY2U9IklicGxhRHJZNE4\
yVm5lQzl5MTZOYWxUOVcvanUrcmp5YmRqaHM9Iixjbm9uY2U9InlqZ2hMVmhjRFJMa0Fob\
tDS0p2WVUxMUM4V1NycjJVWm5IR2Vkclk9IixuYz0wMDAwMDAwMSxxb3A9YXV0aC1jb25m\
Ghlcj0icmM0IixtYXhidWY9MjA0OCxkaWdlc3QtdXJpPSJyY21kLyIscmVzcG9uc2U9OTY\
ODI1MmRmNzY4YTJjYzkxYjJjZDMyYTk0ZWM=
Waiting for server reply...
```

服务器

```
C: dXNlcm5hbWU9Inp6enoiLHJlYWxtPSJqbTExNDE0MiIsbm9uY2U9IklicGxhRHJZNE4\
yVm5lQzl5MTZOYWxUOVcvanUrcmp5YmRqaHM9Iixjbm9uY2U9InlqZ2hMVmhjRFJMa0Fob\
tDS0p2WVUxMUM4V1NycjJVWm5IR2Vkclk9IixuYz0wMDAwMDAwMSxxb3A9YXV0aC1jb25m\
Ghlcj0icmM0IixtYXhidWY9MjA0OCxkaWdlc3QtdXJpPSJyY21kLyIscmVzcG9uc2U9OTY\
ODI1MmRmNzY4YTJjYzkxYjJjZDMyYTk0ZWM=
got 'username="zzzz",realm="jm114142",\
nonce="IbplaDrY4N4szhgX2VneC9y16NalT9W/ju+rjybdjhs=",\
cnonce="yjghLVhcDRLkAhoirwKCKJvYU11C8WSrr2UZnHGedrY=", \
nc=00000001,qop=auth-conf,cipher="rc4",maxbuf=2048,digest-uri="rcmd/",\
response=966e978252df768a2cc91b2cd32a94ec'
*** Calling sasl_server_step() ***
Sending response...
S: cnNwYXV0aD0yYjEzMzRjYzU4NTE4MTEwOWM3OTdhMjUwYjkwMzk3OQ==
Waiting for client reply...
```

客户机

```
S: cnNwYXV0aD0yYjEzMzRjYzU4NTE4MTEwOWM3OTdhMjUwYjkwMzk3OQ==
received 40 byte message
got 'rspauth=2b1334cc585181109c797a250b903979'
*** Calling sasl_client_step() ***
C:
```

```
Negotiation complete
*** Calling sasl_getprop() ***
Username: zzzz
SSF: 128
Waiting for encoded message...
```

服务器

```
Waiting for client reply...
C: got '' *** Calling sasl_server_step() ***
Negotiation complete
*** Calling sasl_getprop() to get username, realm, ssf ***
Username: zzzz
Realm: 22c38
SSF: 128
*** Calling sasl_encode() *** sending encrypted message 'srv message 1'
S: AAAAHvArjnAvDFuMBqAAxkqdumzJB6VD1oajiwABAAAAAA==
```

客户机

```
S: AAAAHvArjnAvDFuMBqAAxkqdumzJB6VD1oajiwABAAAAAA==
received 34 byte message
got ''
*** Calling sasl_decode() ***
received decoded message 'srv message 1'
*** Calling sasl_encode() ***
sending encrypted message 'client message 1'
C: AAAAIRdkTEMYOn9X4NXkxPc3OTFvAZUnLbZANqzn6gABAAAAAA==
*** Calling sasl_dispose() to release client SASL connection context ***
*** Calling sasl_done() to release libsasl resources ***
```

服务器

```
Waiting for encrypted message...
C: AAAAIRdkTEMYOn9X4NXkxPc3OTFvAZUnLbZANqzn6gABAAAAAA==
got ''
*** Calling sasl_decode() ***
received decoded message 'client message 1'
*** Calling sasl_dispose() to release client SASL connection context ***
```

服务提供者的 SASL

本节介绍如何创建用于为 SASL 应用程序提供机制和其他服务的插件。

注 – 由于导出规程，对于非 Solaris 客户机/服务器机制插件，Solaris SASL SPI 不支持安全层。因此，非 Solaris 客户机/服务器机制插件不能提供完整性和保密性服务。Solaris 客户机/服务器机制插件没有此限制。

SASL 插件概述

SASL 服务提供者接口 (Service Provider Interface, SPI) 可实现插件与 `libsasl` 库之间的通信。SASL 插件通常作为共享库来实现。单个共享库可以具有一个或多个不同类型的 SASL 插件。位于共享库中的插件是由 `libsasl` 通过 `dlopen(3C)` 函数动态打开的。

还可以将插件静态绑定至调用 `libsasl` 的应用程序。使用 `sasl_client_add_plugin()` 函数或 `sasl_server_add_plugin()` 函数装入这些种类的插件，具体视应用程序是客户机还是服务器而定。

Solaris 操作系统中的 SASL 插件具有以下要求：

- 共享库中的插件必须位于有效的可执行对象文件（文件扩展名最好是 `.so`）中。
- 插件必须位于可验证的位置中。SASL_CB_VERIFYFILE 回调用于验证插件。
- 插件必须包含正确的入口点。
- SASL 客户机的插件版本必须与 SASL 服务器的对应插件版本匹配。
- 插件需要能够成功初始化。
- 二进制类型的插件必须与 `libsasl` 的二进制类型匹配。

SASL 插件分为四种类别：

- 客户机机制插件
- 服务器机制插件
- 标准化插件
- `auxprop` 插件

`sasl_client_init()` 函数导致 SASL 客户机装入所有可用的客户机插件。`sasl_server_init()` 函数导致 SASL 服务器装入服务器插件、标准化插件和 `auxprop` 插件。调用 `sasl_done()` 时将卸载所有插件。

为查找插件，`libsasl` 使用 SASL_CB_GETPATH 回调函数或缺省路径。SASL_CB_GETPATH 返回以冒号分隔的目录列表，以供在其中查找插件。如果 SASL 使用者指定了 SASL_CB_GETPATH 回调，则 `libsasl` 将使用返回的搜索路径。否则，SASL 使用者可以使用与二进制类型对应的缺省路径：

- 32 位 SPARC 体系结构：`/usr/lib/sasl`
- 32 位 x86 体系结构：`/usr/lib/sasl`
- 64 位 SPARC 体系结构：`/usr/lib/sasl/sparcv9`
- x64 体系结构：`/usr/lib/sasl/amd64`

在装入过程中，`libsasl` 将调用最新支持的插件版本。插件将返回该版本以及描述该插件的结构。如果该版本合格，则 `libsasl` 即可装入该插件。当前版本号 SASL_UTILS_VERSION 为 4。

初始化插件后，插件与 `libsasl` 之间的后续通信通过必须建立的结构执行。插件使用 `sasl_utils_t` 结构来调用 `libsasl`。`libsasl` 使用以下结构中的入口点与插件进行通信：

- sasl_out_params_t
- sasl_client_params_t
- sasl_server_params_t
- sasl_client_plug_t
- sasl_server_plug_t
- sasl_canonuser_plug_t
- sasl_auxprop_plug_t

可以在 SASL 头文件中找到这些结构的源代码。这些结构将在下一节中进行介绍。

SASL 插件的重要结构

libsasl 与插件之间的通信是通过以下结构完成的：

- sasl_utils_t－sasl_utils_t 结构包含大量实用程序函数以及三种上下文：

  此结构包含大量可为插件编写人员提供方便的实用程序函数。许多函数是指向 libsasl 中的公共接口的指针。插件不需要直接调用 libsasl，除非出于某种原因，插件需要是 SASL 使用者。

  libsasl 将为 sasl_utils_t 创建三种上下文。

  - sasl_conn_t *conn
  - sasl_rand_t *rpool
  - void *getopt_context

  在某些情况（如装入插件）下，sasl_utils_t 中的 conn 变量实际上与连接没有关联。在另外一些情况下，conn 是 SASL 使用者的 SASL 连接上下文。rpool 变量用于随机数生成函数。getopt_context 是应该与 getopt() 函数结合使用的上下文。

  sasl_getopt_t(3SASL)、sasl_log_t(3SASL) 和 sasl_getcallback_t(3SASL)
- sasl_out_params_t－libsasl 用于创建 sasl_out_params_t 结构并将该结构传递给客户机或服务器中的 mech_step()。此结构可以将以下信息传达给 libsasl：验证状态、authid、authzid、maxbuf、协商的 ssf 以及数据编码和解码信息。
- sasl_client_params_t－libsasl 使用 sasl_client_params_t 结构将客户机状态传递给 SASL 客户机机制。客户机机制的 mech_new()、mech_step() 和 mech_idle() 入口点用于发送此状态数据。canon_user_client() 入口点还需要同时传递客户机状态。
- sasl_server_params_t－sasl_server_params_t 结构在服务器端执行和 sasl_client_params_t 类似的功能。

客户机插件

客户机插件用于管理 SASL 协商的客户端。客户机插件通常随对应的服务器插件一同打包。客户机插件包含一种或多种客户端 SASL 机制。每种 SASL 客户机机制都支持验证、完整性和保密性（后两者是可选的）。每种机制都提供有关该机制的功能的信息：

- 最大 SSF
- 最大安全标志
- 插件功能
- 用于使用插件的回调和提示 ID

客户机插件必须导出 sasl_client_plug_init()。libsasl 将调用 sasl_client_plug_init() 来初始化客户机的插件。插件将返回 sasl_client_plug_t 结构。sasl_client_plug_t 将为 libsasl 提供以下用于调用机制的入口点：

- mech_new()－客户机通过调用 sasl_client_start()（使用 mech_new()）来启动连接。mech_new() 将执行特定于机制的初始化。如有必要，将分配连接上下文。
- mech_step()－mech_step() 可由 sasl_client_start() 和 sasl_client_step() 进行调用。调用 mech_new() 后，mech_step() 将对客户端执行验证。如果验证成功，mech_step() 将返回 SASL_OK。如果需要更多数据，则将返回 SASL_CONTINUE。如果验证失败，则将返回 SASL 错误代码。如果出现错误，则将调用 seterror()。如果验证成功，则 mech_step() 必须返回包含相关安全层信息和回调的 sasl_out_params_t 结构。canon_user() 函数是此结构的一部分。客户机收到验证和授权 ID 时，必须调用 canon_user()。
- mech_dispose()－mech_dispose() 是在安全关闭上下文时进行调用的。mech_dispose() 由 sasl_dispose() 进行调用。
- mech_free()－mech_free() 是在 libsasl 关闭时进行调用的。插件的所有其余全局状态都是通过 mech_free() 释放的。

服务器插件

服务器插件用于管理 SASL 协商的服务器端。服务器插件通常随对应的客户机插件一同打包。服务器插件包含一种或多种服务器端 SASL 机制。每种 SASL 服务器机制都支持验证、完整性和保密性（后两者是可选的）。每种机制都提供有关该机制的功能的信息：

- 最大 SSF
- 最大安全标志
- 插件功能
- 用于使用插件的回调和提示 ID

服务器插件必须导出 sasl_server_plug_init()。libsasl 将调用 sasl_server_plug_init() 来初始化服务器的插件。插件将返回 sasl_server_plug_t 结构。sasl_server_plug_t 将为 libsasl 提供以下用于调用机制的入口点：

- mech_new()－服务器通过调用 sasl_server_start()（使用 mech_new()）来启动连接。mech_new() 将执行特定于机制的初始化。如有必要，mech_new() 将分配连接上下文。

- mech_step() – mech_step() 可由 sasl_server_start() 和 sasl_server_step() 进行调用。调用 mech_new() 后，mech_step() 将对服务器端执行验证。如果验证成功，mech_step() 将返回 SASL_OK。如果需要更多数据，则将返回 SASL_CONTINUE。如果验证失败，则将返回 SASL 错误代码。如果出现错误，则将调用 seterror()。如果验证成功，则 mech_step() 必须返回包含相关安全层信息和回调的 sasl_out_params_t 结构。canon_user() 函数是此结构的一部分。服务器收到验证和授权 ID 时，必须调用 canon_user()。调用 canon_user() 函数将导致 propctx 被填充。标准化验证之前，应执行所有必需的辅助属性请求。标准化验证后，应执行授权 ID 查找。

  返回 SASL_OK 之前，mech_step() 函数必须填充所有相关的 sasl_out_params_t 字段。这些字段可以执行以下函数：

  - doneflag – 指示完整的交换
  - maxoutbuf – 指示安全层的最大输出大小
  - mech_ssf – 为安全层提供 SSF
  - encode() – 由 sasl_encode()、sasl_encodev() 和 sasl_decode() 进行调用
  - decode() – 由 sasl_encode()、sasl_encodev() 和 sasl_decode() 进行调用
  - encode_context() – 由 sasl_encode()、sasl_encodev() 和 sasl_decode() 进行调用
  - decode_context() – 由 sasl_encode()、sasl_encodev() 和 sasl_decode() 进行调用
- mech_dispose() – mech_dispose() 是在安全关闭上下文时进行调用的。mech_dispose() 由 sasl_dispose() 进行调用。
- mech_free() – mech_free() 是在 libsasl 关闭时进行调用的。插件的所有其余全局状态都是通过 mech_free() 释放的。
- setpass() 用于设置用户的口令。setpass() 使机制可以具有内部口令。
- mech_avail() 由 sasl_listmech () 进行调用，以检查机制是否可用于给定用户。mech_avail() 可以创建新的上下文，进而避免对 mech_new() 的调用。只要性能不受影响，就应使用此方法创建上下文。

用户标准化插件

标准化插件为客户端和服务器端的验证和授权名称的备用标准化提供支持。sasl_canonuser_plug_init() 用于装入标准化插件。标准化插件具有以下要求：

- 必须将标准化的名称复制到输出缓冲区。
- 可以将同一输入缓冲区用作输出缓冲区。
- 当仅存在一个验证 ID 或授权 ID 时，标准化插件必须发挥作用。

用户标准化插件必须导出 sasl_canonuser_init() 函数。sasl_canonuser_init() 函数必须返回 sasl_canonuser_plug_t 以建立必要的入口点。用户标准化插件必须至少实现 sasl_canonuser_plug_t 结构的一个 canon_user_client() 成员或 canon_user_server() 成员。

辅助属性 (auxprop) 插件

auxprop 插件为查找 SASL 服务器的 authid 和 authzid 的辅助属性提供支持。例如，应用程序可能需要查找内部验证的用户口令。sasl_auxprop_plug_init() 函数用于初始化 auxprop 插件并返回 sasl_auxpropr_plug_t 结构。

要成功实现 auxprop 插件，必须实现 sasl_auxprop_plug_t 结构的 auxprop_lookup 成员。标准化用户名后，应使用标准化的用户名来调用 auxprop_lookup() 函数。随后，插件即可执行请求的辅助属性所需的所有查询。

注 – Oracle Solaris 当前不提供 auxprop 插件。

SASL 插件开发指南

本节提供了一些用于开发 SASL 插件的其他建议。

SASL 插件中的错误报告

适当的错误报告可以帮助跟踪验证问题和其他调试问题。建议插件开发者使用 sasl_utils_t 结构中的 sasl_seterror() 回调为给定连接提供详细的错误信息。

SASL 插件中的内存分配

在 SASL 中分配内存的一般规则是在不再需要已分配的任何内存时将其释放。遵循此规则可以提高性能和可移植性，而且可以避免内存泄漏。

设置 SASL 协商顺序

插件机制可以设置客户机和服务器通过以下标志执行 SASL 会话的顺序：

- SASL_FEAT_WANT_CLIENT_FIRST－客户端将开始交换。
- SASL_FEAT_WANT_SERVER_LAST－服务器将最终数据发送到客户机。

如果未设置任何标志，则机制插件将在内部设置顺序。在这种情况下，机制必须同时检查客户机和服务器中需要发送的数据。请注意，只有在协议允许初始响应时，才有可能出现客户机首先发送的情况。

服务器最后发送的情况要求在步骤函数返回 SASL_OK 时插件可设置 *serverout。永远不让服务器最后发送的那些机制必须将 *serverout 设置为 NULL。始终让服务器最后发送的那些机制需要将 *serverout 指向成功数据。

◆ ◆ ◆ 第 8 章

8 Oracle Solaris 加密框架介绍

Oracle Solaris 加密框架是一种体系结构，可使 Oracle Solaris 操作系统中的应用程序使用或提供加密服务。与框架的所有交互都基于 RSA Security Inc. 的 PKCS#11 加密令牌接口 (Cryptographic Token Interface, Cryptoki)。PKCS#11 是 RSA Laboratories（RSA Security Inc. 的研究机构）开发的一种产品。本章介绍有关 Oracle Solaris 加密框架的以下主题：

Oracle Solaris 加密术语

获取加密服务的应用程序、库或内核模块称作**使用者**。通过框架向使用者提供加密服务的应用程序称作**提供者**，或称作**插件**。用来实现加密操作的软件称作**机制**。机制不只是算法，还包括算法的应用方式。例如，应用于验证机制的 DES（数据加密标准）算法被视为一个单独的机制，应用于逐块加密的 DES 则是另一个机制。

令牌是可以执行加密的设备的抽象术语。此外，令牌可以存储要用在加密操作中的信息。一个令牌可以支持一个或多个机制。令牌可以代表硬件，例如加速器板中的部件。仅代表软件的令牌称作**软令牌**。令牌可以**插入**到**插槽**中，插槽也是一种物理比喻。插槽是使用加密服务的应用程序的连接点。

除了提供者的特定插槽以外，Solaris 实现还提供一个名为**元插槽**的特殊插槽。元插槽是 Solaris 加密框架库 (`libpkcs11.so`) 的一个组件。metaslot 起着一个虚拟插槽的作用，该插槽具有已经安装在框架中的所有令牌和插槽的组合功能。使用 metaslot，应用程序可以通过单个插槽与任何可用加密服务之间有效地实现透明连接。当应用程序请求加密服务时，元插槽将指向最适合的插槽，这简化了插槽的选择过程。在某些情况下可能需要一个不同的插槽，此时，应用程序必须显式地执行单独的搜索。元插槽由系统自动启用，而且只能由系统管理员通过显式的操作来禁用。

会话是令牌与使用加密服务的应用程序之间的连接。PKCS #11 标准使用两种对象：令牌对象和会话对象。**会话对象**是暂时的对象，即它只在会话期间存在。在会话结束之后仍存在的对象称作**令牌对象**。

令牌对象的缺省位置是 `$HOME/.sunw/pkcs11_softtoken`。或者，可以将令牌对象存储到 `$SOFTTOKEN_DIR/pkcs11_softtoken` 中。专用令牌对象由个人识别码 (personal identification number, PIN) 保护。要创建或更改令牌对象，需要对用户进行验证，除非用户访问的是专用令牌对象。

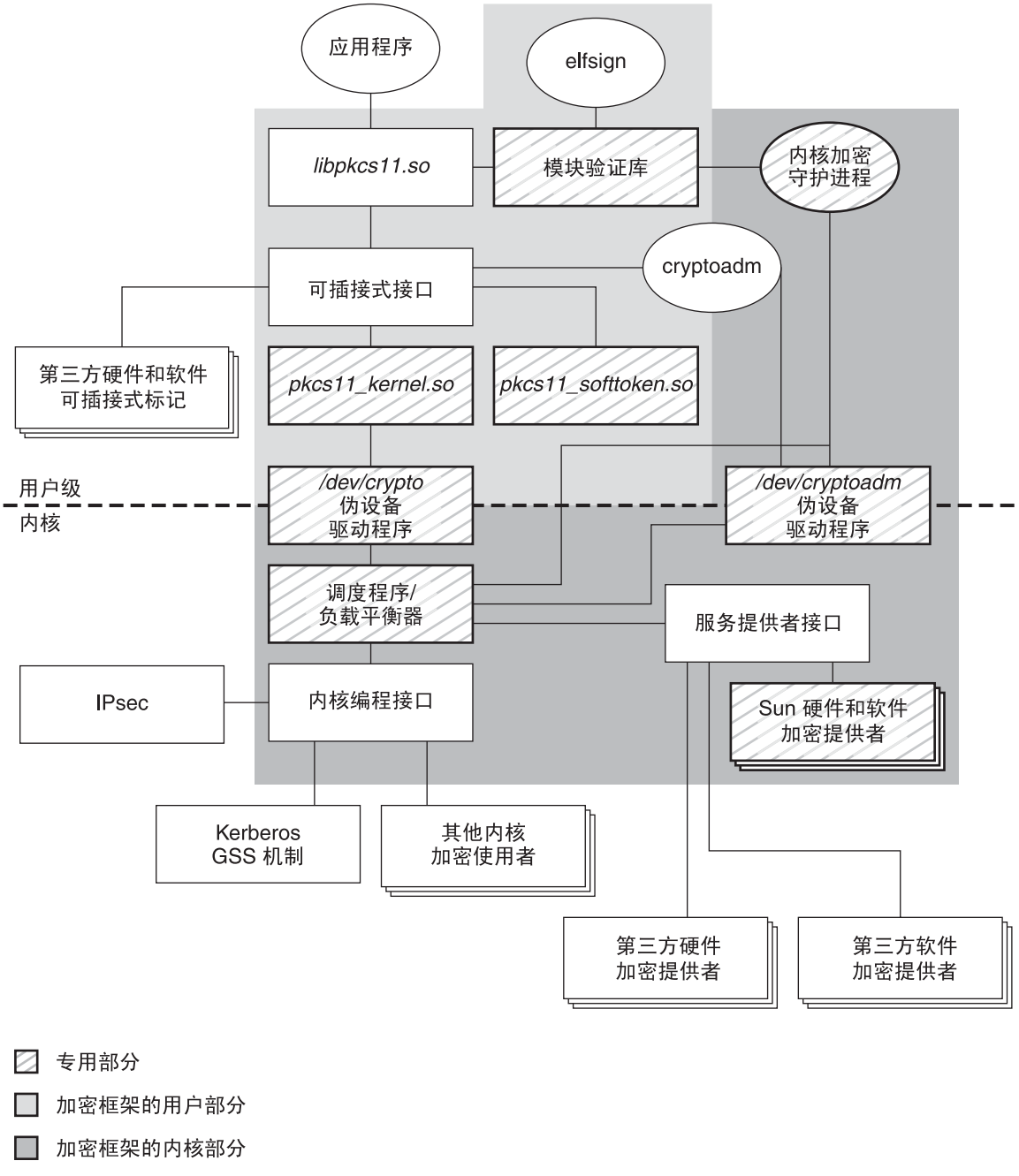
加密框架概述

加密框架是 Oracle Solaris OS 的一部分，它提供来自 Oracle Corporation 和第三方供应商的加密 服务。加密框架提供以下各种服务：

- 消息加密和消息摘要
- 消息验证代码 (Message Authentication Code, MAC)
- 数字签名
- 用来访问加密服务的应用编程接口 (Application Programmer Interface, API)
- 用来提供加密服务的服务提供者接口 (Service Provider Interface, SPI)
- 用来管理加密资源的管理命令

下图提供了加密框架的概述。该图中的浅灰色阴影表示加密框架的用户级部分，深灰色阴影表示加密框架的内核级部分，带有斜条纹的背景表示专用软件部分。

图 8–1 Oracle Solaris 加密框架概述



加密框架的组件

加密框架的组件如下所述：

- `libpkcs11.so`－该框架通过 RSA Security Inc. 的 PKCS#11 加密令牌接口 (Cryptographic Token Interface, Cryptoki) 提供访问权限。需要将应用程序链接到 `libpkcs11.so` 库，该库实现 RSA PKCS#11 v2.11 标准。
- **可插接式接口**－可插接式接口是 &PK 的服务提供者接口 (Service Provider Interface, SPI)，这些服务由 Oracle Corporation 和第三方开发者提供。提供者是用户级库，可以通过能够从硬件或软件使用的加密服务实现。
- `pkcs11_softtoken.so`－一个专用共享目标文件，其中包含由 Oracle Corporation 提供的用户级加密机制。`pkcs11_softtoken(5)` 库可用来实现 RSA PKCS#11 v2.11 规范。
- `pkcs11_kernel.so`－用来访问内核级加密机制的专用共享目标文件。`pkcs11_kernel(5)` 可用来实现 RSA PKCS#11 v2.11 规范。`pkcs11_kernel.so` 为插入到内核的服务提供者接口中的加密服务提供一个 PKCS#11 用户接口。
- `/dev/crypto` **伪设备驱动程序**－使用内核级加密机制的专用伪设备驱动程序。提供此信息的目的在于避免意外删除伪设备驱动程序。
- **调度程序/负载平衡器**－一种内核软件，负责协调对加密服务请求的使用并对这些请求进行负载平衡和分发。
- **内核编程接口**－用于加密服务的内核级使用者的接口。IPSec（Internet 协议安全）协议和 Kerberos GSS 机制是典型的加密使用者。

注 – 只有与 Oracle Corporation 签订了特殊的合同才能使用此接口。有关更多信息，请发电子邮件至 `solaris-crypto-req_ww@oracle.com`。

- **Oracle HW 和 SW 加密提供者**－由 Oracle Corporation 提供的内核级加密服务。HW 是指硬件加密服务（如加速板）。SW 是指提供加密服务的内核模块（如加密算法的实现）。
- **内核加密框架守护进程**－一种专用守护进程，负责管理用于加密操作的系统资源。还负责验证加密提供者。
- **模块验证库**－一种专用库，用于验证 Solaris 加密框架所导入的所有库的完整性和真实性。
- `elfsign`－提供给加密服务的第三方提供者的实用程序。`elfsign` 用于从 Oracle Corporation 申请 证书。`elfsign` 还允许提供者对插入到 Oracle Solaris 加密框架中的二进制文件（即 elf 目标文件）进行实际签名。
- `/dev/cryptoadm` **伪设备驱动程序**－一种专用的伪设备驱动程序，由 `cryptoadm`(1M) 用来管理内核级加密机制。提供此信息的目的在于避免意外删除伪设备驱动程序。
- `cryptoadm`－可供管理员管理加密服务的用户级命令。使用 `cryptoadm` 执行的典型任务就是列出加密提供者及其功能，使用 `cryptoadm` 还可以根据安全策略禁用和启用加密机制。

加密开发者需要了解的内容

本节介绍可插入到 Oracle Solaris 加密框架中的三种应用程序的开发要求。

用户级使用者的开发要求

要开发用户级使用者，开发者需要牢记以下几点：

- 包括 `<security/cryptoki.h>`。
- 所有的调用都只通过 PKCS #11 接口执行。
- 链接到 `libpkcs11.so`。
- 库不应当调用 `C_Finalize()` 函数。

有关更多信息，请参见第 9 章，编写用户级加密应用程序和提供者。

用户级提供者的开发要求

要开发用户级提供者，开发者需要牢记以下几点：

- 设计要独立使用的提供者。尽管由提供者共享的目标文件不必是应用程序所链接到的具有完整功能的库，但是提供者中必须存在所有必需的符号。假设提供者将要在 RTLD_GROUP 和 RTLD_NOW 模式下由 `dlopen(3C)` 打开。
- 在共享目标文件中创建 PKCS #11 Cryptoki 实现。此共享目标文件中应当包括必需的符号，而不要依赖于使用者应用程序。
- 强烈建议（但并非必需）为数据清除提供 `_fini()` 例程。如果应用程序或共享库同时装入 `libpkcs11` 和其他提供者库，则需要使用此方法来避免在 `C_Finalize()` 调用之间产生冲突。请参见第 144 页中的“避免在用户级提供者中出现数据清除冲突”。
- 从 Oracle Corporation 申请证书。请参见第 257 页中的“申请提供者签署证书”。
- 使用证书和 `elfsign` 对二进制文件进行签名。请参见第 258 页中的“签署提供者”。
- 按照 Oracle 约定对共享目标文件进行打包。请参见附录 F，打包和签署加密提供者。

内核级使用者的开发要求

要开发内核级使用者，开发者需要牢记以下几点：

- 包括 `<sys/crypto/common.h>` 和 `<sys/crypto/api.h>`。
- 所有的调用都通过内核编程接口执行。

避免在用户级提供者中出现数据清除冲突

插入到加密框架中的用户级库应当提供 _fini () 函数。卸载用户级库时，装入器会调用 _fini() 函数。_fini() 函数是确保所有的清除操作都在恰当的时间正确完成所必需的。系统不假设那些使用 libpkcs11 的库调用 C_Finalize()，因为 libpkcs11 是共享库，有可能正由应用程序使用。

要提供 _fini() 函数，需要在可重定位对象的程序数据部分中创建一个 .fini 部分。.fini 部分提供运行时终止代码块。请参见《链接程序和库指南》。以下代码样例演示了如何设计 .fini 部分。

示例 8–1 向 PKCS #11 库提供 _fini()

```
#pragma fini(pkcs11_fini)
static void pkcs11_fini();

/* [... (other library code omitted)] */

static void
pkcs11_fini()
{
        (void) pthread_mutex_lock(&pkcs11mutex);

        /* If CRYPTOKI is not initialized, do not clean up */
        if (!initialized) {
                (void) pthread_mutex_unlock(&pkcs11mutex);
                return;
        }

        (void) finalize_routine(NULL_PTR);

        (void) pthread_mutex_unlock(&pkcs11mutex);
}
```

◆ ◆ ◆ 第 9 章

9 编写用户级加密应用程序和提供者

本章介绍如何开发使用 PKCS #11 函数进行加密的用户级应用程序和提供者。本章包含以下主题：

有关加密框架的更多信息，请参见第 8 章，Oracle Solaris 加密框架介绍。

Cryptoki 库概述

Oracle Solaris 加密框架中的用户级应用程序通过 `libpkcs11.so` 模块中所提供的 `cryptoki` 库来访问 PKCS #11 函数。`pkcs11_softtoken.so` 模块是由 Oracle Corporation 提供的 PKCS #11 软令牌实现，用于提供加密机制。软令牌插件是缺省的机制源。加密机制还可以通过第三方插件提供。

本节列出了软令牌所支持的 PKCS #11 函数和返回值，返回代码根据插入到框架中的提供者而异。本节还介绍了一些常见的函数。有关 `cryptoki` 库中所有元素的完整说明，请参阅 `libpkcs11(3LIB)` 或者 RSA Laboratories Web 站点上的 PKCS #11: Cryptographic Token Interface Standard。

PKCS #11 函数列表

下面列出了 Solaris 加密框架中的 pkcs11_softtoken.so 所支持的 PKCS #11 函数类别以及相关函数：

- **通用**－C_Initialize()、C_Finalize()、C_GetInfo() 和 C_GetFunctionList()
- **会话管理**
  －C_OpenSession()、C_CloseSession()、C_GetSessionInfo()、C_CloseAllSessions()、C_Login() 和 C_Logout()
- **插槽和令牌管理**
  －C_GetSlotList()、C_GetSlotInfo()、C_GetMechanismList()、C_GetMechanismInfo() 和 C_SetPIN()
- **加密和解密**
  －C_EncryptInit()、C_Encrypt()、C_EncryptUpdate()、C_EncryptFinal()、C_DecryptInit()、C_Decrypt()、C_DecryptUpdate() 和 C_DecryptFinal()
- **消息摘要**－C_DigestInit()、C_Digest()、C_DigestKey()、C_DigestUpdate() 和 C_DigestFinal()
- **MAC 的签名和应用**
  －C_Sign()、C_SignInit()、C_SignUpdate()、C_SignFinal()、C_SignRecoverInit() 和 C_SignRecover()
- **签名验证**－C_Verify()、C_VerifyInit()、C_VerifyUpdate()、C_VerifyFinal()、C_VerifyRecoverInit() 和 C_VerifyRecover()
- **双重用途加密函数**
  －C_DigestEncryptUpdate()、C_DecryptDigestUpdate()、C_SignEncryptUpdate() 和 C_DecryptVerifyUpdate()
- **随机数生成**－C_SeedRandom() 和 C_GenerateRandom()
- **对象管理**
  －C_CreateObject()、C_DestroyObject()、C_CopyObject()、C_FindObjects()、C_FindObjectsInit()、C_FindObjectsFinal()、C_GetAttributeValue() 和 C_SetAttributeValue()
- **密钥管理**－C_GenerateKey()、C_GenerateKeyPair() 和 C_DeriveKey()

使用 PKCS #11 的函数

本节提供了以下使用 PKCS #11 的函数的说明：

- 第 147 页中的 “PKCS #11 函数： C_Initialize()”
- 第 147 页中的 “PKCS #11 函数： C_GetInfo()”
- 第 148 页中的 “PKCS #11 函数： C_GetSlotList()”
- 第 148 页中的 “PKCS #11 函数： C_GetTokenInfo()”
- 第 149 页中的 “PKCS #11 函数： C_OpenSession()”

- 第 150 页中的 “PKCS #11 函数： C_GetMechanismList()”

注 – 所有的 PKCS #11 函数都可以从 libpkcs11.so 库中获取，不必使用 C_GetFunctionList() 函数来获取可用函数的列表。

PKCS #11 函数： C_Initialize()

C_Initialize() 可用于初始化 PKCS #11 库。C_Initialize() 语法如下：

```
C_Initialize(CK_VOID_PTR pInitArgs);
```

pInitArgs 是空值 NULL_PTR 或是指向 CK_C_INITIALIZE_ARGS 结构的指针。通过 NULL_PTR，该库可以将 Solaris 互斥锁用作锁定原语，在多个线程之间仲裁对内部共享结构的访问。请注意，Solaris 加密框架不接受互斥锁。由于 cryptoki 库的此实现可以安全高效地处理多线程，因此建议使用 NULL_PTR。应用程序还可以使用 *pInitArgs* 来设置诸如 CKF_LIBRARY_CANT_CREATE_OS_THREADS 之类的标志。C_Finalize() 表示应用程序使用 PKCS #11 库结束会话。

注 – C_Finalize() 绝不应当通过库进行调用。按照惯例，应用程序负责调用 C_Finalize() 来关闭会话。

除了 CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY 和 CKR_OK 以外，C_Initialize() 还会使用以下返回值：

- CKR_ARGUMENTS_BAD
- CKR_CANT_LOCK
- CKR_CRYPTOKI_ALREADY_INITIALIZED－此错误不是致命的。

PKCS #11 函数： C_GetInfo()

C_GetInfo() 使用的是有关 cryptoki 库的制造商和版本信息。C_GetInfo() 使用以下语法：

```
C_GetInfo(CK_INFO_PTR pInfo);
```

C_GetInfo() 会返回以下值：

- cryptokiVersion = 2, 11
- manufacturerID = Oracle Corporation

除了 CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY 和 CKR_OK 以外，C_GetInfo() 还可以获取以下返回值：

- CKR_ARGUMENTS_BAD
- CKR_CRYPTOKI_NOT_INITIALIZED

PKCS #11 函数：C_GetSlotList()

C_GetSlotList() 使用的是可用插槽的列表。如果除了 pkcs11_softtoken.so 以外尚未安装任何其他加密提供者，则 C_GetSlotList() 仅返回缺省插槽。C_GetSlotList() 使用以下语法：

```
C_GetSlotList(CK_BBOOL tokenPresent, CK_SLOT_ID_PTR pSlotList,
CK_ULONG_PTR pulCount);
```

如果 *tokenPresent* 设置为 TRUE，则会将搜索限制在那些存在令牌的插槽。

如果 *pSlotList* 设置为 NULL_PTR，则 C_GetSlotlist() 仅返回插槽的数量。*pulCount* 是指向用于接收插槽计数的位置的指针。

如果 *pSlotList* 指向用于接收插槽的缓冲区，则 **pulCount* 将设置为 CK_SLOT_ID 元素的最大预期数量。在返回时，**pulCount* 将设置为 CK_SLOT_ID 元素的实际数量。

通常，PKCS #11 应用程序会调用 C_GetSlotList() 两次。第一次调用 C_GetSlotList() 用于获取进行内存分配的插槽数量，第二次调用 C_GetSlotList() 用于检索插槽。

注 – 不能保证插槽的顺序。插槽的顺序会因 PKCS #11 库的每次装入而异。

除了 CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY 和 CKR_OK 以外，C_GetSlotlist() 还可以获取以下返回值：

- CKR_ARGUMENTS_BAD
- CKR_BUFFER_TOO_SMALL
- CKR_CRYPTOKI_NOT_INITIALIZED

PKCS #11 函数：C_GetTokenInfo()

C_GetTokenInfo() 可用于获取有关特定令牌的信息。C_GetTokenInfo() 使用以下语法：

```
C_GetTokenInfo(CK_SLOT_ID slotID, CK_TOKEN_INFO_PTR pInfo);
```

slotID 用于标识令牌的插槽。*slotID* 必须是由 C_GetSlotList () 返回的有效 ID。*pInfo* 是指向用于接收令牌信息的位置的指针。

如果 pkcs11_softtoken.so 是所安装的唯一提供者，C_GetTokenInfo() 将返回以下字段和值：

- 标签－Sun Software PKCS#11 软令牌。
- 标志－CKF_DUAL_CRYPTO_OPERATIONS、CKF_TOKEN_INITIALIZED、CKF_RNG、CKF_USER_PIN_INITIALIZED 和 CKF_LOGIN_REQUIRED，这些标志设置为 1。
- ulMaxSessionCount－设置为 CK_EFFECTIVELY_INFINITE。
- ulMaxRwSessionCount－设置为 CK_EFFECTIVELY_INFINITE。

- ulMaxPinLen－设置为 256。
- ulMinPinLen－设置为 1。
- ulTotalPublicMemory－设置为 CK_UNAVAILABLE_INFORMATION。
- ulFreePublicMemory－设置为 CK_UNAVAILABLE_INFORMATION。
- ulTotalPrivateMemory－设置为 CK_UNAVAILABLE_INFORMATION。
- ulFreePrivateMemory－设置为 CK_UNAVAILABLE_INFORMATION。

除了 CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY 和 CKR_OK 以外，C_GetSlotlist() 还可以获取以下返回值：

- CKR_ARGUMENTS_BAD
- CKR_BUFFER_TOO_SMALL
- CKR_CRYPTOKI_NOT_INITIALIZED
- CKR_SLOT_ID_INVALID

以下返回值与具有硬件令牌的插件相关：

- CKR_DEVICE_ERROR
- CKR_DEVICE_MEMORY
- CKR_DEVICE_REMOVED
- CKR_TOKEN_NOT_PRESENT
- CKR_TOKEN_NOT_RECOGNIZED

PKCS #11 函数：C_OpenSession()

应用程序可使用 C_OpenSession() 来启动特定插槽中具有特定令牌的加密会话。C_OpenSession() 语法如下：

```
C_OpenSession(CK_SLOT_ID slotID, CK_FLAGS flags, CK_VOID_PTR pApplication,
CK_NOTIFY Notify, CK_SESSION_HANDLE_PTR phSession);
```

slotID 用于标识插槽。*flags* 用于指示会话是可读写的还是只读的。*pApplication* 是应用程序所定义的用于回调的指针。*Notify* 用于存放可选回调函数的地址。*phSession* 是指向会话句柄的位置的指针。

除了 CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY 和 CKR_OK 以外，C_OpenSession() 还可以获取以下返回值：

- CKR_ARGUMENTS_BAD
- CKR_CRYPTOKI_NOT_INITIALIZED
- CKR_SLOT_ID_INVALID
- CKR_TOKEN_WRITE_PROTECTED－随受写保护的令牌出现。

以下返回值与具有硬件令牌的插件相关：

- CKR_DEVICE_ERROR
- CKR_DEVICE_MEMORY

- CKR_DEVICE_REMOVED
- CKR_SESSION_COUNT
- CKR_SESSION_PARALLEL_NOT_SUPPORTED
- CKR_SESSION_READ_WRITE_SO_EXISTS
- CKR_TOKEN_NOT_PRESENT
- CKR_TOKEN_NOT_RECOGNIZED

PKCS #11 函数：C_GetMechanismList()

C_GetMechanismList() 用于获取指定令牌所支持的机制类型的列表。C_GetMechanismList() 使用以下语法：

```
C_GetMechanismList(CK_SLOT_ID slotID, CK_MECHANISM_TYPE_PTR pMechanismList,
CK_ULONG_PTR pulCount);
```

slotID 用于标识令牌的插槽。*pulCount* 是指向用于接收机制数量的位置的指针。如果 *pMechanismList* 设置为 NULL_PTR，则 **pulCount* 将返回机制的数量。否则，必须将 **pulCount* 设置为列表的大小，*pMechanismList* 必须指向用于存放列表的缓冲区。

如果已插入 PKCS #11 软令牌，C_GetMechanismList() 将返回以下列出的支持的机制：

- CKM_AES_CBC
- CKM_AES_CBC_PAD
- CKM_AES_ECB
- CKM_AES_KEY_GEN
- CKM_DES_CBC
- CKM_DES_CBC_PAD
- CKM_DES_ECB
- CKM_DES_KEY_GEN
- CKM_DES_MAC
- CKM_DES_MAC_GENERAL
- CKM_DES3_CBC
- CKM_DES3_CBC_PAD
- CKM_DES3_ECB
- CKM_DES3_KEY_GEN
- CKM_DH_PKCS_DERIVE
- CKM_DH_PKCS_KEY_PAIR_GEN
- CKM_DSA
- CKM_DSA_KEY_PAIR_GEN
- CKM_DSA_SHA_1
- CKM_MD5
- CKM_MD5_KEY_DERIVATION
- CKM_MD5_RSA_PKCS
- CKM_MD5_HMAC
- CKM_MD5_HMAC_GENERAL
- CKM_PBE_SHA1_RC4_128

- CKM_PKCS5_PBKD2
- CKM_RC4
- CKM_RC4_KEY_GEN
- CKM_RSA_PKCS
- CKM_RSA_X_509
- CKM_RSA_PKCS_KEY_PAIR_GEN
- CKM_SHA_1
- CKM_SHA_1_HMAC_GENERAL
- CKM_SHA_1_HMAC
- CKM_SHA_1_KEY_DERIVATION
- CKM_SHA_1_RSA_PKCS
- CKM_SSL3_KEY_AND_MAC_DERIVE
- CKM_SSL3_MASTER_KEY_DERIVE
- CKM_SSL3_MASTER_KEY_DERIVE_DH
- CKM_SSL3_MD5_MAC
- CKM_SSL3_PRE_MASTER_KEY_GEN
- CKM_SSL3_SHA1_MAC
- CKM_TLS_KEY_AND_MAC_DERIVE
- CKM_TLS_MASTER_KEY_DERIVE
- CKM_TLS_MASTER_KEY_DERIVE_DH
- CKM_TLS_PRE_MASTER_KEY_GEN

除了 CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY 和 CKR_OK 以外，`C_GetSlotlist()` 还使用以下返回值：

- CKR_ARGUMENTS_BAD
- CKR_BUFFER_TOO_SMALL
- CKR_CRYPTOKI_NOT_INITIALIZED
- CKR_SLOT_ID_INVALID

以下返回值与具有硬件令牌的插件相关：

- CKR_DEVICE_ERROR
- CKR_DEVICE_MEMORY
- CKR_DEVICE_REMOVED
- CKR_TOKEN_NOT_PRESENT
- CKR_TOKEN_NOT_RECOGNIZED

扩展的 PKCS #11 函数

除了标准的 PKCS #11 函数以外，Oracle Solaris 加密框架还附带了两个便利函数：

- 第 152 页中的 “扩展的 PKCS #11 函数： `SUNW_C_GetMechSession()()`”
- 第 152 页中的 “扩展的 PKCS #11 函数： SUNW_C_KeyToObject”

扩展的 PKCS #11 函数：SUNW_C_GetMechSession()()

SUNW_C_GetMechSession() 是一个便利函数，用于初始化 Solaris 加密框架。该函数随后会使用指定的机制启动会话。SUNW_C_GetMechSession() 使用以下语法：

```
SUNW_C_GetMechSession(CK_MECHANISM_TYPE mech, C\
K_SESSION_HANDLE_PTR hSession)
```

mech 参数用于指定要使用的机制。*hSession* 是指向会话位置的指针。

SUNW_C_GetMechSession() 在内部调用 C_Initialize() 以初始化 cryptoki 库。SUNW_C_GetMechSession() 接着会使用指定的机制调用 C_GetSlotList() 和 C_GetMechanismInfo()，在可用插槽中搜索令牌。如果找到了机制，SUNW_C_GetMechSession() 会调用 C_OpenSession() 来打开会话。

SUNW_C_GetMechSession() 只需要调用一次。不过，多次调用 SUNW_C_GetMechSession() 也不会造成任何问题。

扩展的 PKCS #11 函数：SUNW_C_KeyToObject

SUNW_C_KeyToObject() 用于创建私钥对象。调用程序必须指定要使用的机制以及原始密钥数据。SUNW_C_KeyToObject() 可在内部确定指定机制的密钥类型。通用密钥对象是通过 C_CreateObject() 创建的。SUNW_C_KeyToObject() 接着会调用 C_GetSessionInfo() 和 C_GetMechanismInfo() 来获取插槽和机制。C_SetAttributeValue() 随后会根据机制的类型为密钥对象设置属性标志。

用户级加密应用程序示例

本节包含以下示例：

消息摘要示例

此示例使用 PKCS #11 函数通过输入文件创建摘要。此示例执行以下步骤：

1. 指定摘要机制。

   此示例中使用的是 CKM_MD5 摘要机制。

2. 查找适用于指定摘要算法的插槽。

   此示例使用 Orace Solaris 的便利函数 SUNW_C_GetMechSession()。SUNW_C_GetMechSession() 用于打开 cryptoki 库，该库用于存放 Oracle Solaris 加密框架中所使用的全部 PKCS #11 函

数。SUNW_C_GetMechSession() 随后使用所需的机制来查找插槽。然后，将会启动会话。这个便利函数可有效地替换 C_Initialize() 调用、C_OpenSession() 调用以及查找支持指定机制的插槽所需的任何代码。

3. 获取 cryptoki 信息。

   本部分实际上不是创建消息摘要所必需的，之所以将其包括在内是为了说明 C_GetInfo() 函数的用法。此示例中将获取制造商 ID。其他信息选项用于检索版本和库数据。

4. 针对插槽执行摘要操作。

   此任务中的消息摘要可通过以下几个步骤创建：

   a. 打开输入文件。
   b. 通过调用 C_DigestInit() 来初始化摘要操作。
   c. 使用 C_DigestUpdate() 逐段处理数据。
   d. 使用 C_DigestFinal() 获取完整的摘要，从而结束摘要操作过程。

5. 结束会话。

   程序使用 C_CloseSession() 关闭会话，使用 C_Finalize() 关闭库。

以下示例中显示了消息摘要示例的源代码。

示例 9–1 使用 PKCS #11 函数创建消息摘要

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <security/cryptoki.h>
#include <security/pkcs11.h>

#define BUFFERSIZ    8192
#define MAXDIGEST    64

/* Calculate the digest of a user supplied file. */
void
main(int argc, char **argv)
{
    CK_BYTE digest[MAXDIGEST];
    CK_INFO info;
    CK_MECHANISM mechanism;
    CK_SESSION_HANDLE hSession;
    CK_SESSION_INFO Info;
    CK_ULONG ulDatalen = BUFFERSIZ;
    CK_ULONG ulDigestLen = MAXDIGEST;
    CK_RV rv;
    CK_SLOT_ID SlotID;

    int i, bytes_read = 0;
    char inbuf[BUFFERSIZ];
    FILE *fs;
    int error = 0;

    /* Specify the CKM_MD5 digest mechanism as the target */
```

示例 9–1 使用 PKCS #11 函数创建消息摘要 （续）

```
mechanism.mechanism = CKM_MD5;
mechanism.pParameter = NULL_PTR;
mechanism.ulParameterLen = 0;

/* Use SUNW convenience function to initialize the cryptoki
 * library, and open a session with a slot that supports
 * the mechanism we plan on using. */
rv = SUNW_C_GetMechSession(mechanism.mechanism, &hSession);
if (rv != CKR_OK) {
    fprintf(stderr, "SUNW_C_GetMechSession: rv = 0x%.8X\n", rv);
    exit(1);
}

/* Get cryptoki information, the manufacturer ID */
rv = C_GetInfo(&info);
if (rv != CKR_OK) {
    fprintf(stderr, "WARNING: C_GetInfo: rv = 0x%.8X\n", rv);
}
fprintf(stdout, "Manufacturer ID = %s\n", info.manufacturerID);

/* Open the input file */
if ((fs = fopen(argv[1], "r")) == NULL) {
    perror("fopen");
    fprintf(stderr, "\n\tusage: %s filename>\n", argv[0]);
    error = 1;
    goto exit_session;
}

/* Initialize the digest session */
if ((rv = C_DigestInit(hSession, &mechanism)) != CKR_OK) {
    fprintf(stderr, "C_DigestInit: rv = 0x%.8X\n", rv);
    error = 1;
    goto exit_digest;
}

/* Read in the data and create digest of this portion */
while (!feof(fs) && (ulDatalen = fread(inbuf, 1, BUFFERSIZ, fs)) > 0) {
    if ((rv = C_DigestUpdate(hSession, (CK_BYTE_PTR)inbuf,
                ulDatalen)) != CKR_OK) {
        fprintf(stderr, "C_DigestUpdate: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_digest;
    }
    bytes_read += ulDatalen;
}
fprintf(stdout, "%d bytes read and digested!!!\n\n", bytes_read);

/* Get complete digest */
ulDigestLen = sizeof (digest);
if ((rv = C_DigestFinal(hSession, (CK_BYTE_PTR)digest,
            &ulDigestLen)) != CKR_OK) {
    fprintf(stderr, "C_DigestFinal: rv = 0x%.8X\n", rv);
    error = 1;
    goto exit_digest;
}
```

示例 9–1 使用 PKCS #11 函数创建消息摘要 （续）

```
    /* Print the results */
    fprintf(stdout, "The value of the digest is: ");
    for (i = 0; i < ulDigestLen; i++) {
        fprintf(stdout, "%.2x", digest[i]);
    }
    fprintf(stdout, "\nDone!!!\n");

exit_digest:
    fclose(fs);

exit_session:
    (void) C_CloseSession(hSession);

exit_program:
    (void) C_Finalize(NULL_PTR);

    exit(error);

}
```

对称加密示例

示例 9–2 在 CBC（Cipher Block Chaining，密码块链接）模式下使用 DES 算法为加密创建了密钥对象。此源代码执行以下步骤：

1. 声明密钥材料。

   定义 DES 和初始化向量。以静态方式声明的初始化向量仅用于说明，初始化向量应始终以动态方式定义并且永远不会重用。

2. 定义密钥对象。

   对于此任务，必须为密钥设置模板。

3. 查找适用于指定加密机制的插槽。

   此示例使用 Oracle Solaris 的便利函数 `SUNW_C_GetMechSession()`。`SUNW_C_GetMechSession()` 用于打开 `cryptoki` 库，该库用于存放 Oracle Solaris 加密框架中所使用的全部 PKCS #11 函数。`SUNW_C_GetMechSession()` 随后使用所需的机制来查找插槽。然后，将会启动会话。这个便利函数可有效地替换 `C_Initialize()` 调用、`C_OpenSession()` 调用以及查找支持指定机制的插槽所需的任何代码。

4. 在插槽中执行加密操作。

   此任务中的加密可通过以下几个步骤执行：

   a. 打开输入文件。
   b. 创建密钥的对象句柄。
   c. 使用机制结构将加密机制设置为 CKM_DES_CBC_PAD。
   d. 调用 `C_EncryptInit()` 来初始化加密操作。
   e. 使用 `C_EncryptUpdate()` 逐段处理数据。

f. 使用 C_EncryptFinal() 获取最后一部分加密数据，从而结束加密过程。

5. 在插槽中执行解密操作。

   此任务中的解密可通过以下几个步骤执行。提供解密的目的仅是为了进行测试。

   a. 调用 C_DecryptInit() 来初始化解密操作。
   b. 使用 C_Decrypt() 处理整个字符串。

6. 结束会话。

   程序使用 C_CloseSession() 关闭会话，使用 C_Finalize() 关闭库。

以下示例中显示了对称加密示例的源代码。

示例 9–2 使用 PKCS #11 函数创建加密密钥对象

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <security/cryptoki.h>
#include <security/pkcs11.h>

#define BUFFERSIZ    8192

/* Declare values for the key materials. DO NOT declare initialization
 * vectors statically like this in real life!! */
uchar_t des_key[] = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef};
uchar_t des_cbc_iv[] = { 0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xcd, 0xef};

/* Key template related definitions. */
static CK_BBOOL truevalue = TRUE;
static CK_BBOOL falsevalue = FALSE;
static CK_OBJECT_CLASS class = CKO_SECRET_KEY;
static CK_KEY_TYPE keyType = CKK_DES;


/* Example encrypts and decrypts a file provided by the user. */
void
main(int argc, char **argv)
{
    CK_RV rv;
    CK_MECHANISM mechanism;
    CK_OBJECT_HANDLE hKey;
    CK_SESSION_HANDLE hSession;
    CK_ULONG ciphertext_len = 64, lastpart_len = 64;
    long ciphertext_space = BUFFERSIZ;
    CK_ULONG decrypttext_len;
    CK_ULONG total_encrypted = 0;
    CK_ULONG ulDatalen = BUFFERSIZ;

    int i, bytes_read = 0;
    int error = 0;
    char inbuf[BUFFERSIZ];
    FILE *fs;
    uchar_t ciphertext[BUFFERSIZ], *pciphertext, decrypttext[BUFFERSIZ];

    /* Set the key object */
```

示例 9–2 使用 PKCS #11 函数创建加密密钥对象 （续）

```
    CK_ATTRIBUTE template[] = {
        {CKA_CLASS, &class, sizeof (class) },
        {CKA_KEY_TYPE, &keyType, sizeof (keyType) },
        {CKA_TOKEN, &falsevalue, sizeof (falsevalue) },
        {CKA_ENCRYPT, &truevalue, sizeof (truevalue) },
        {CKA_VALUE, &des_key, sizeof (des_key) }
    };

    /* Set the encryption mechanism to CKM_DES_CBC_PAD */
    mechanism.mechanism = CKM_DES_CBC_PAD;
    mechanism.pParameter = des_cbc_iv;
    mechanism.ulParameterLen = 8;

    /* Use SUNW convenience function to initialize the cryptoki
     * library, and open a session with a slot that supports
     * the mechanism we plan on using. */
    rv = SUNW_C_GetMechSession(mechanism.mechanism, &hSession);

    if (rv != CKR_OK) {
        fprintf(stderr, "SUNW_C_GetMechSession: rv = 0x%.8X\n", rv);
        exit(1);
    }

    /* Open the input file */
    if ((fs = fopen(argv[1], "r")) == NULL) {
        perror("fopen");
        fprintf(stderr, "\n\tusage: %s filename>\n", argv[0]);
        error = 1;
        goto exit_session;
    }

    /* Create an object handle for the key */
    rv = C_CreateObject(hSession, template,
        sizeof (template) / sizeof (CK_ATTRIBUTE),
        &hKey);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_CreateObject: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_session;
    }


    /* Initialize the encryption operation in the session */
    rv = C_EncryptInit(hSession, &mechanism, hKey);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_EncryptInit: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_session;
    }

    /* Read in the data and encrypt this portion */
    pciphertext = &ciphertext[0];
    while (!feof(fs) && (ciphertext_space > 0) &&
        (ulDatalen = fread(inbuf, 1, ciphertext_space, fs)) > 0) {
```

示例 9–2 使用 PKCS #11 函数创建加密密钥对象 （续）

```
        ciphertext_len = ciphertext_space;

        /* C_EncryptUpdate is only being sent one byte at a
         * time, so we are not checking for CKR_BUFFER_TOO_SMALL.
         * Also, we are checking to make sure we do not go
         * over the alloted buffer size.  A more robust program
         * could incorporate realloc to enlarge the buffer
         * dynamically.     */
        rv = C_EncryptUpdate(hSession, (CK_BYTE_PTR)inbuf, ulDatalen,
            pciphertext, &ciphertext_len);
        if (rv != CKR_OK) {
            fprintf(stderr, "C_EncryptUpdate: rv = 0x%.8X\n", rv);
            error = 1;
            goto exit_encrypt;
        }
        pciphertext += ciphertext_len;
        total_encrypted += ciphertext_len;
        ciphertext_space -= ciphertext_len;
        bytes_read += ulDatalen;
    }

    if (!feof(fs) || (ciphertext_space < 0)) {
        fprintf(stderr, "Insufficient space for encrypting the file\n");
        error = 1;
        goto exit_encrypt;
    }

    /* Get the last portion of the encrypted data */
    lastpart_len = ciphertext_space;
    rv = C_EncryptFinal(hSession, pciphertext, &lastpart_len);
    if (rv != CKR_OK) {
        fprintf(stderr, "C_EncryptFinal: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_encrypt;
    }
    total_encrypted += lastpart_len;

    fprintf(stdout, "%d bytes read and encrypted. Size of the "
        "ciphertext: %d!\n\n", bytes_read, total_encrypted);

    /* Print the encryption results */
    fprintf(stdout, "The value of the encryption is:\n");
    for (i = 0; i < ciphertext_len; i++) {
        if (ciphertext[i] < 16)
            fprintf(stdout, "0%x", ciphertext[i]);
        else
            fprintf(stdout, "%2x", ciphertext[i]);
    }

    /* Initialize the decryption operation in the session */
    rv = C_DecryptInit(hSession, &mechanism, hKey);

    /* Decrypt the entire ciphertext string */
    decrypttext_len = sizeof (decrypttext);
    rv = C_Decrypt(hSession, (CK_BYTE_PTR)ciphertext, total_encrypted,
        decrypttext, &decrypttext_len);
```

示例 9–2 使用 PKCS #11 函数创建加密密钥对象 （续）

```
    if (rv != CKR_OK) {
        fprintf(stderr, "C_Decrypt: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_encrypt;
    }

    fprintf(stdout, "\n\n%d bytes decrypted!!!\n\n", decrypttext_len);

    /* Print the decryption results */
    fprintf(stdout, "The value of the decryption is:\n%s", decrypttext);

    fprintf(stdout, "\nDone!!!\n");

exit_encrypt:
    fclose(fs);

exit_session:
    (void) C_CloseSession(hSession);

exit_program:
    (void) C_Finalize(NULL_PTR);
    exit(error);
}
```

签名和验证示例

本节中的示例生成了一个 RSA 密钥对。该密钥对用于对简单字符串进行签名和验证。此示例执行以下步骤：

1. 定义密钥对象。
2. 设置公钥模板。
3. 设置私钥模板。
4. 创建样例消息。
5. 指定用于生成密钥对的 `genmech` 机制。
6. 指定用于对密钥对进行签名的 `smech` 机制。
7. 初始化 `cryptoki` 库。
8. 通过用于生成和验证密钥对并对其进行签名的机制来查找插槽。此任务将使用一个名为 `getMySlot()` 的函数来执行以下步骤：
   a. 调用 `C_GetSlotList()` 函数以获取可用插槽的列表。

      与 PKCS #11 约定中所建议的一样，`C_GetSlotList()` 需要调用两次。第一次调用 `C_GetSlotList()` 用于获取进行内存分配的插槽数量，第二次调用 `C_GetSlotList()` 用于检索插槽。
   b. 查找可以提供所需机制的插槽。

对于每个插槽，该函数都会调用 GetMechanismInfo() 以查找可用于生成密钥对并对其进行签名的机制。如果插槽不支持这些机制，则 GetMechanismInfo() 将返回错误。如果 GetMechanismInfo() 成功返回，将检查机制标志，以确保这些机制可以执行所需的操作。

9. 调用 C_OpenSession() 来打开会话。
10. 使用 C_GenerateKeyPair () 来生成密钥对。
11. 使用 C_GetAttributeValue() 显示公钥－仅用于说明。
12. 签名以 C_SignInit() 开始，以 C_Sign() 结束。
13. 验证以 C_VerifyInit() 开始，以 C_Verify() 结束。
14. 关闭会话。

    程序使用 C_CloseSession() 关闭会话，使用 C_Finalize() 关闭库。

下面是签名和验证示例的源代码。

示例 9–3 使用 PKCS #11 函数对文本进行签名和验证

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <security/cryptoki.h>
#include <security/pkcs11.h>

#define BUFFERSIZ    8192

/* Define key template */
static CK_BBOOL truevalue = TRUE;
static CK_BBOOL falsevalue = FALSE;
static CK_ULONG modulusbits = 1024;
static CK_BYTE public_exponent[] = {3};

boolean_t GetMySlot(CK_MECHANISM_TYPE sv_mech, CK_MECHANISM_TYPE kpgen_mech,
    CK_SLOT_ID_PTR pslot);

/* Example signs and verifies a simple string, using a public/private
 * key pair. */
void
main(int argc, char **argv)
{
    CK_RV   rv;
    CK_MECHANISM genmech, smech;
    CK_SESSION_HANDLE hSession;
    CK_SESSION_INFO sessInfo;
    CK_SLOT_ID slotID;
    int error, i = 0;

    CK_OBJECT_HANDLE privatekey, publickey;

    /* Set public key. */
    CK_ATTRIBUTE publickey_template[] = {
        {CKA_VERIFY, &truevalue, sizeof (truevalue)},
        {CKA_MODULUS_BITS, &modulusbits, sizeof (modulusbits)},
```

示例 9–3 使用 PKCS #11 函数对文本进行签名和验证 （续）

```
        {CKA_PUBLIC_EXPONENT, &public_exponent,
            sizeof (public_exponent)}
    };

    /* Set private key. */
    CK_ATTRIBUTE privatekey_template[] = {
        {CKA_SIGN, &truevalue, sizeof (truevalue)},
        {CKA_TOKEN, &falsevalue, sizeof (falsevalue)},
        {CKA_SENSITIVE, &truevalue, sizeof (truevalue)},
        {CKA_EXTRACTABLE, &truevalue, sizeof (truevalue)}
    };

    /* Create sample message. */
    CK_ATTRIBUTE getattributes[] = {
        {CKA_MODULUS_BITS, NULL_PTR, 0},
        {CKA_MODULUS, NULL_PTR, 0},
        {CKA_PUBLIC_EXPONENT, NULL_PTR, 0}
    };

    CK_ULONG messagelen, slen, template_size;

    boolean_t found_slot = B_FALSE;
    uchar_t *message = (uchar_t *)"Simple message for signing & verifying.";
    uchar_t *modulus, *pub_exponent;
    char sign[BUFFERSIZ];
    slen = BUFFERSIZ;

    messagelen = strlen((char *)message);

    /* Set up mechanism for generating key pair */
    genmech.mechanism = CKM_RSA_PKCS_KEY_PAIR_GEN;
    genmech.pParameter = NULL_PTR;
    genmech.ulParameterLen = 0;

    /* Set up the signing mechanism */
    smech.mechanism = CKM_RSA_PKCS;
    smech.pParameter = NULL_PTR;
    smech.ulParameterLen = 0;

    /* Initialize the CRYPTOKI library */
    rv = C_Initialize(NULL_PTR);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_Initialize: Error = 0x%.8X\n", rv);
        exit(1);
    }

    found_slot = GetMySlot(smech.mechanism, genmech.mechanism, &slotID);

    if (!found_slot) {
        fprintf(stderr, "No usable slot was found.\n");
        goto exit_program;
    }

    fprintf(stdout, "selected slot: %d\n", slotID);
```

示例 9–3 使用 PKCS #11 函数对文本进行签名和验证 （续）

```
    /* Open a session on the slot found */
    rv = C_OpenSession(slotID, CKF_SERIAL_SESSION, NULL_PTR, NULL_PTR,
        &hSession);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_OpenSession: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_program;
    }

    fprintf(stdout, "Generating keypair....\n");

    /* Generate Key pair for signing/verifying */
    rv = C_GenerateKeyPair(hSession, &genmech, publickey_template,
        (sizeof (publickey_template) / sizeof (CK_ATTRIBUTE)),
        privatekey_template,
        (sizeof (privatekey_template) / sizeof (CK_ATTRIBUTE)),
        &publickey, &privatekey);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_GenerateKeyPair: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_session;
    }

    /* Display the publickey. */
    template_size = sizeof (getattributes) / sizeof (CK_ATTRIBUTE);

    rv = C_GetAttributeValue(hSession, publickey, getattributes,
        template_size);

    if (rv != CKR_OK) {
        /* not fatal, we can still sign/verify if this failed */
        fprintf(stderr, "C_GetAttributeValue: rv = 0x%.8X\n", rv);
        error = 1;
    } else {
        /* Allocate memory to hold the data we want */
        for (i = 0; i < template_size; i++) {
            getattributes[i].pValue =
                malloc (getattributes[i].ulValueLen *
                sizeof(CK_VOID_PTR));
            if (getattributes[i].pValue == NULL) {
                int j;
                for (j = 0; j < i; j++)
                    free(getattributes[j].pValue);
                goto sign_cont;
            }
        }

        /* Call again to get actual attributes */
        rv = C_GetAttributeValue(hSession, publickey, getattributes,
            template_size);

        if (rv != CKR_OK) {
            /* not fatal, we can still sign/verify if failed */
            fprintf(stderr,
```

示例 9–3 使用 PKCS #11 函数对文本进行签名和验证 （续）

```
                "C_GetAttributeValue: rv = 0x%.8X\n", rv);
            error = 1;
        } else {
            /* Display public key values */
            fprintf(stdout, "Public Key data:\n\tModulus bits: "
                "%d\n",
                *((CK_ULONG_PTR)(getattributes[0].pValue)));

            fprintf(stdout, "\tModulus: ");
            modulus = (uchar_t *)getattributes[1].pValue;
            for (i = 0; i < getattributes[1].ulValueLen; i++) {
                fprintf(stdout, "%.2x", modulus[i]);
            }

            fprintf(stdout, "\n\tPublic Exponent: ");
            pub_exponent = (uchar_t *)getattributes[2].pValue;
            for (i = 0; i< getattributes[2].ulValueLen; i++) {
                fprintf(stdout, "%.2x", pub_exponent[i]);
            }
            fprintf(stdout, "\n");
        }
    }

sign_cont:
    rv = C_SignInit(hSession, &smech, privatekey);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_SignInit: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_session;
    }

    rv = C_Sign(hSession, (CK_BYTE_PTR)message, messagelen,
        (CK_BYTE_PTR)sign, &slen);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_Sign: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_session;
    }

    fprintf(stdout, "Message was successfully signed with private key!\n");

    rv = C_VerifyInit(hSession, &smech, publickey);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_VerifyInit: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_session;
    }

    rv = C_Verify(hSession, (CK_BYTE_PTR)message, messagelen,
        (CK_BYTE_PTR)sign, slen);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_Verify: rv = 0x%.8X\n", rv);
```

示例 9–3 使用 PKCS #11 函数对文本进行签名和验证 （续）

```
        error = 1;
        goto exit_session;
    }

    fprintf(stdout, "Message was successfully verified with public key!\n");

exit_session:
    (void) C_CloseSession(hSession);

exit_program:
    (void) C_Finalize(NULL_PTR);

    for (i = 0; i < template_size; i++) {
        if (getattributes[i].pValue != NULL)
            free(getattributes[i].pValue);
    }

    exit(error);

}

/* Find a slot capable of:
 * . signing and verifying with sv_mech
 * . generating a key pair with kpgen_mech
 * Returns B_TRUE when successful. */
boolean_t GetMySlot(CK_MECHANISM_TYPE sv_mech, CK_MECHANISM_TYPE kpgen_mech,
    CK_SLOT_ID_PTR pSlotID)
{
    CK_SLOT_ID_PTR pSlotList = NULL_PTR;
    CK_SLOT_ID SlotID;
    CK_ULONG ulSlotCount = 0;
    CK_MECHANISM_INFO mech_info;
    int i;
    boolean_t returnval = B_FALSE;

    CK_RV rv;

    /* Get slot list for memory alloction */
    rv = C_GetSlotList(0, NULL_PTR, &ulSlotCount);

    if ((rv == CKR_OK) && (ulSlotCount > 0)) {
        fprintf(stdout, "slotCount = %d\n", ulSlotCount);
        pSlotList = malloc(ulSlotCount * sizeof (CK_SLOT_ID));

        if (pSlotList == NULL) {
            fprintf(stderr, "System error: unable to allocate "
                "memory\n");
            return (returnval);
        }

        /* Get the slot list for processing */
        rv = C_GetSlotList(0, pSlotList, &ulSlotCount);
        if (rv != CKR_OK) {
            fprintf(stderr, "GetSlotList failed: unable to get "
                "slot count.\n");
            goto cleanup;
```

示例 9–3 使用 PKCS #11 函数对文本进行签名和验证 （续）

```
        }
    } else {
        fprintf(stderr, "GetSlotList failed: unable to get slot "
            "list.\n");
        return (returnval);
    }

    /* Find a slot capable of specified mechanism */
    for (i = 0; i < ulSlotCount; i++) {
        SlotID = pSlotList[i];

        /* Check if this slot is capable of signing and
         * verifying with sv_mech. */
        rv = C_GetMechanismInfo(SlotID, sv_mech, &mech_info);

        if (rv != CKR_OK) {
            continue;
        }

        if (!(mech_info.flags & CKF_SIGN &&
            mech_info.flags & CKF_VERIFY)) {
            continue;
        }

        /* Check if the slot is capable of key pair generation
         * with kpgen_mech. */
        rv = C_GetMechanismInfo(SlotID, kpgen_mech, &mech_info);

        if (rv != CKR_OK) {
            continue;
        }

        if (!(mech_info.flags & CKF_GENERATE_KEY_PAIR)) {
            continue;
        }

        /* If we get this far, this slot supports our mechanisms. */
        returnval = B_TRUE;
        *pSlotID = SlotID;
        break;
    }

cleanup:
    if (pSlotList)
        free(pSlotList);
    return (returnval);
}
```

随机字节生成示例

示例 9–4 说明了如何使用可以生成随机字节的机制来查找插槽。此示例执行以下步骤：

1. 初始化 `cryptoki` 库。

2. 调用 GetRandSlot()，以便使用可以生成随机字节的机制来查找插槽。

   插槽查找任务执行以下步骤：

   a. 调用 C_GetSlotList() 函数以获取可用插槽的列表。

      与 PKCS #11 约定中所建议的一样，C_GetSlotList() 需要调用两次。第一次调用 C_GetSlotList() 用于获取进行内存分配的插槽数量，第二次调用 C_GetSlotList() 用于检索插槽。

   b. 查找可以生成随机字节的插槽。

      对于每个插槽，该函数都可以使用 GetTokenInfo() 来获取令牌信息，并在设置了 CKF_RNG 标志的情况下检查匹配项。如果找到设置了 CKF_RNG 标志的插槽，则 GetRandSlot() 函数将返回。

3. 使用 C_OpenSession() 来打开会话。
4. 使用 C_GenerateRandom() 来生成随机字节。
5. 结束会话。

   程序使用 C_CloseSession() 关闭会话，使用 C_Finalize() 关闭库。

随机数生成样例的源代码如以下示例所示。

示例 9–4 使用 PKCS #11 函数生成随机数

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <security/cryptoki.h>
#include <security/pkcs11.h>

#define RANDSIZE 64

boolean_t GetRandSlot(CK_SLOT_ID_PTR pslot);

/* Example generates random bytes. */
void
main(int argc, char **argv)
{
    CK_RV   rv;
    CK_MECHANISM mech;
    CK_SESSION_HANDLE hSession;
    CK_SESSION_INFO sessInfo;
    CK_SLOT_ID slotID;
    CK_BYTE randBytes[RANDSIZE];

    boolean_t found_slot = B_FALSE;
    int error;
    int i;

    /* Initialize the CRYPTOKI library */
    rv = C_Initialize(NULL_PTR);

    if (rv != CKR_OK) {
```

示例 9–4 使用 PKCS #11 函数生成随机数 （续）

```
        fprintf(stderr, "C_Initialize: Error = 0x%.8X\n", rv);
        exit(1);
    }

    found_slot = GetRandSlot(&slotID);

    if (!found_slot) {
        goto exit_program;
    }

    /* Open a session on the slot found */
    rv = C_OpenSession(slotID, CKF_SERIAL_SESSION, NULL_PTR, NULL_PTR,
        &hSession);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_OpenSession: rv = 0x%.8x\n", rv);
        error = 1;
        goto exit_program;
    }

    /* Generate random bytes */
    rv = C_GenerateRandom(hSession, randBytes, RANDSIZE);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_GenerateRandom: rv = 0x%.8x\n", rv);
        error = 1;
        goto exit_session;
    }

    fprintf(stdout, "Random value: ");
    for (i = 0; i < RANDSIZE; i++) {
        fprintf(stdout, "%.2x", randBytes[i]);
    }

exit_session:
    (void) C_CloseSession(hSession);

exit_program:
    (void) C_Finalize(NULL_PTR);
    exit(error);

}

boolean_t
GetRandSlot(CK_SLOT_ID_PTR pslot)
{
    CK_SLOT_ID_PTR pSlotList;
    CK_SLOT_ID SlotID;
    CK_TOKEN_INFO tokenInfo;
    CK_ULONG ulSlotCount;
    CK_MECHANISM_TYPE_PTR pMechTypeList = NULL_PTR;
    CK_ULONG ulMechTypecount;
    boolean_t result = B_FALSE;
    int i = 0;

    CK_RV rv;
```

示例 9–4 使用 PKCS #11 函数生成随机数 （续）

```
    /* Get slot list for memory allocation */
    rv = C_GetSlotList(0, NULL_PTR, &ulSlotCount);

    if ((rv == CKR_OK) && (ulSlotCount > 0)) {
        fprintf(stdout, "slotCount = %d\n", (int)ulSlotCount);
        pSlotList = malloc(ulSlotCount * sizeof (CK_SLOT_ID));

        if (pSlotList == NULL) {
            fprintf(stderr,
                "System error: unable to allocate memory\n");
            return (result);
        }

        /* Get the slot list for processing */
        rv = C_GetSlotList(0, pSlotList, &ulSlotCount);
        if (rv != CKR_OK) {
            fprintf(stderr, "GetSlotList failed: unable to get "
                "slot list.\n");
            free(pSlotList);
            return (result);
        }
    } else {
        fprintf(stderr, "GetSlotList failed: unable to get slot"
            " count.\n");
        return (result);
    }

    /* Find a slot capable of doing random number generation */
    for (i = 0; i < ulSlotCount; i++) {
        SlotID = pSlotList[i];

        rv = C_GetTokenInfo(SlotID, &tokenInfo);

        if (rv != CKR_OK) {
            /* Check the next slot */
            continue;
        }

        if (tokenInfo.flags & CKF_RNG) {
            /* Found a random number generator */
            *pslot = SlotID;
            fprintf(stdout, "Slot # %d supports random number "
                "generation!\n", SlotID);
            result = B_TRUE;
            break;
        }
    }

    if (pSlotList)
        free(pSlotList);

    return (result);

}
```

◆ ◆ ◆ 第 10 章

10 使用智能卡框架

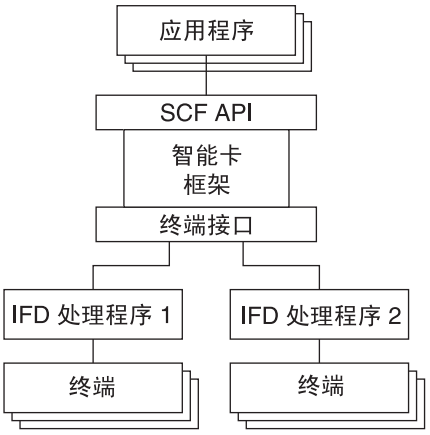
智能卡是包含微处理器和内存的便携式计算机。智能卡的形状和大小通常与信用卡相同。智能卡为可通过验证和加密保护的机密信息提供高度安全存储。

本章包含以下主题：

Oracle Solaris 智能卡框架概述

在 Oracle Solaris 操作系统中，智能卡框架用于将使用者应用程序与智能卡终端连接起来。使用者应用程序可以调用智能卡框架 (Smart Card Framework, SCF) API。智能卡终端通过接口设备 (Interface Device, IFD) 处理程序（本质上为设备驱动程序）与使用者应用程序进行通信。IFD 处理程序通过终端接口连接至框架。请参见下图。

图 10–1 智能卡框架



Oracle Solaris 操作系统在专用文件中存储智能卡配置信息。与此相反，Linux 实现通常使用 /etc/reader.conf。要更改配置文件中的项，请使用命令 smartcard(1M)。

目前，智能卡框架与 Oracle Solaris 加密框架无关。

开发智能卡使用者应用程序

SCF API 提供了一组用于访问智能卡的接口。这些接口采用低级应用程序协议数据单元 (Application Protocol Data Unit, APDU) 形式提供与卡之间的通信。C 和 Java 中都提供了这些接口。这些接口与 Solaris 操作系统支持的所有读取器以及与 APDU 通信的任何智能卡协同工作。SCF API 基于以下组件：

- **会话对象** – 每个单个线程的常规上下文，以便避免冲突。
- **终端对象** – 物理智能卡终端的抽象术语。此对象可以检测出是否存在、插入或移除了卡。
- **卡对象** – 表示在终端插入的智能卡。该对象可以采用 APDU 格式将信息发送给物理智能卡。该对象还可以调节互斥锁，以使应用程序可具有对卡进行独占访问的权限。
- **侦听器对象** – 接收事件通知的对象。

SCF API 提供以下领域的功能：

- 检查读取器中是否存在智能卡。
- 接收智能卡移动（即插入和移除）通知。
- 与智能卡交换数据。
- 检索有关会话、终端和智能卡的信息。
- 锁定和解除锁定要独占访问的智能卡。

以下各节提供有关特定 SCF 接口的信息。

SCF 会话接口

以下函数用于 SCF 会话。

SCF_Session_getSession(3SMARTCARD)
使用系统的智能卡框架建立会话。打开会话后，可以将该会话与 SCF_Session_getTerminal(3SMARTCARD) 一同使用，以访问智能卡终端。

SCF_Session_close(3SMARTCARD)
释放打开会话时分配的资源。另外，关闭与该会话关联的所有终端或卡。

SCF_Session_getInfo(3SMARTCARD)
获取有关会话的信息。

SCF_Session_freeInfo(3SMARTCARD)
取消从 SCF_Session_getInfo(3SMARTCARD) 返回的存储的分配。

SCF_Session_getTerminal(3SMARTCARD)
使用特定的智能卡终端在会话中建立上下文。终端对象用于检测卡移动（即插入或移除）。终端对象还用于创建用于访问特定卡的卡对象。

SCF 终端接口

以下函数用于访问 SCF 终端。

SCF_Terminal_close(3SMARTCARD)
释放打开终端时分配的资源。该函数还可以关闭所有与终端关联的卡。

SCF_Terminal_getInfo(3SMARTCARD)
获取有关终端的信息。

SCF_Terminal_freeInfo(3SMARTCARD)
取消从 SCF_Terminal_getInfo(3SMARTCARD) 返回的存储的分配。

SCF_Terminal_waitForCardPresent(3SMARTCARD)
阻塞并等待，直到指定终端中出现卡为止。

SCF_Terminal_waitForCardAbsent(3SMARTCARD)
阻塞并等待，直到指定终端中移除卡为止。

SCF_Terminal_addEventListener(3SMARTCARD)
在终端发生事件时，允许程序接收回调通知。此概念与信号处理程序类似。发生事件时，服务线程将执行提供的回调函数。

SCF_Terminal_updateEventListener(3SMARTCARD)
更新与此终端关联的指定事件侦听器。

SCF_Terminal_removeEventListener(3SMARTCARD)
从与此终端关联的侦听器列表中删除指定的事件侦听器。

SCF_Terminal_getCard(3SMARTCARD)
使用特定的智能卡在终端中建立上下文。可以使用卡对象借助 SCF_Card_exchangeAPDU(3SMARTCARD) 将 APDU 发送给卡。

SCF 卡和各种接口

以下函数用于访问智能卡及获取状态。

SCF_Card_close(3SMARTCARD)
释放打开卡时分配的资源（如内存和线程）。另外，还释放该卡持有的锁定。

SCF_Card_getInfo(3SMARTCARD)
获取有关卡的信息。

SCF_Card_freeInfo(3SMARTCARD)
取消从 SCF_Card_getInfo(3SMARTCARD) 返回的存储的分配。

SCF_Card_lock(3SMARTCARD)
获取对特定卡的锁定。此函数允许应用程序执行多重 APDU 事务，而不会受到其他智能卡应用程序的干扰。

SCF_Card_unlock(3SMARTCARD)
从特定卡中移除锁定。

SCF_Card_exchangeAPDU(3SMARTCARD)
将命令 APDU 发送给卡并读取卡的响应。

SCF_Card_waitForCardRemoved(3SMARTCARD)
检查是否已移除特定卡。如果插入了其他卡或重新插入了同一个卡，则该函数将报告旧卡已被移除。

SCF_Card_reset(3SMARTCARD)
重置特定卡。

SCF_strerror(3SMARTCARD)
获取描述状态代码的字符串。

为智能卡终端开发 IFD 处理程序

为 Oracle Solaris OS 开发的智能卡终端使用的 API 集合与 Linux 智能卡终端所使用的 API 集合完全相同。如果您以前未开发过 IFD 处理程序，则可访问一个提供 IFD 源代码的 Linux 环境 Web 站点，如 http://www.musclecard.com/drivers.html。要为 Solaris 操作系统中的智能卡终端开发 IFD 处理程序，需要包括 `/usr/include/smartcard/ifdhandler.h` 并实现以下接口：

- IFDHCreateChannelByName(3SMARTCARD)－使用指定的智能卡终端打开信道。此接口在最新版本的 MUSCLE IFD 规范中是新增的内容。因此，`IFDHCreateChannelByName()` 在其他 IFD 处理程序中可能不可用。在 Solaris 软件

中，使用 IFDHCreateChannelByName()，而不使用 IFDHCreateChannel(3SMARTCARD) 函数。

- IFDHICCPresence(3SMARTCARD)－检查逻辑单元号 (Logical Unit Number, LUN) 指定的读取器或插槽中是否存在 ICC（即智能卡）。
- IFDHPowerICC(3SMARTCARD)－控制电源和重置 ICC 的信号。
- IFDHCloseChannel(3SMARTCARD)－关闭由 LUN 指定的 IFD 的通信通道。
- IFDHGetCapabilities(3SMARTCARD)－返回指定智能卡、IFD 处理程序或智能卡终端的功能。
- IFDHSetProtocolParameters(3SMARTCARD)－为特定插槽或卡设置协议类型选择 (Protocol Type Selection, PTS)。在 ISO 7816 标准中查找 PTS 值。尽管此函数可能不是由框架调用，但还是应该执行此函数。使用 IFDHSetProtocolParameters() 可以确保各种卡都能与框架进行通信。
- IFDHTransmitToICC(3SMARTCARD)－由框架调用，用于与智能卡进行通信。

注 – 当前不使用 IFDHCreateChannel()、IFDHSetCapabilities() 和 IFDHControl()，但未来的发行版中可能需要这些接口。

IFDHICCPresence() 和 IFDHPowerICC() 函数对于测试非常有用。例如，可以使用 IFDHICCPresence() 函数来测试插槽中是否存在卡。检查智能卡电源是否正常工作的一种方式是使用 IFDHPowerICC() 函数。此函数可获取已插入智能卡的重置应答 (Answer to Reset, ATR) 值。

安装智能卡终端

Solaris 智能卡框架不支持可热插拔的终端，如 USB 终端。请使用以下方法连接和安装智能卡终端：

1. 在终端与系统之间建立物理连接。
2. 将 IFD 处理程序的共享库复制到系统中。
3. 使用 smartcard(1M) 在框架中注册终端的 IFD 处理程序。

◆ ◆ ◆ 附 录 A

基于 C 的 GSS-API 样例程序

本附录给出了使用 GSS-API 建立安全网络连接的两个样例应用程序的源代码。第一个应用程序为典型客户机。第二个应用程序说明服务器如何在 GSS-API 中工作。在运行的过程中，这两个程序都会显示基准测试程序。因此，用户可以查看使用中的 GSS-API。此外，还提供了可供客户机和服务器应用程序使用的各种特定函数。

- 第 175 页中的 “客户端应用程序”
- 第 185 页中的 “服务器端应用程序”
- 第 195 页中的 “各种 GSS-API 样例函数”

第 5 章，GSS-API 客户机示例和第 6 章，GSS-API 服务器示例中详细检查了这些程序。

客户端应用程序

以下示例提供了客户端程序 `gss_client` 的源代码。

示例 A–1 `gss-client.c` 样例程序的完整列表

```
/*
 * Copyright 1994 by OpenVision Technologies, Inc.
 *
 * Permission to use, copy, modify, distribute, and sell this software
 * and its documentation for any purpose is hereby granted without fee,
 * provided that the above copyright notice appears in all copies and
 * that both that copyright notice and this permission notice appear in
 * supporting documentation, and that the name of OpenVision not be used
 * in advertising or publicity pertaining to distribution of the software
 * without specific, written prior permission. OpenVision makes no
 * representations about the suitability of this software for any
 * purpose.  It is provided "as is" without express or implied warranty.
 *
 * OPENVISION DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
 * INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
 * EVENT SHALL OPENVISION BE LIABLE FOR ANY SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
 * USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
```

示例 A–1 gss-client.c样例程序的完整列表 （续）

```
 * OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 * PERFORMANCE OF THIS SOFTWARE.
 */

#if !defined(lint) && !defined(__CODECENTER__)
static char *rcsid = \
"$Header: /cvs/krbdev/krb5/src/appl/gss-sample/gss-client.c,\
v 1.16 1998/10/30 02:52:03 marc Exp $";
#endif

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>

#include <gssapi/gssapi.h>
#include <gssapi/gssapi_ext.h>
#include <gss-misc.h>

void usage()
{
     fprintf(stderr, "Usage: gss-client [-port port] [-d] host service \
msg\n");
     exit(1);
}

/*
 * Function: connect_to_server
 *
 * Purpose: Opens a TCP connection to the name host and port.
 *
 * Arguments:
 *
 *      host            (r) the target host name
 *      port            (r) the target port, in host byte order
 *
 * Returns: the established socket file descriptor, or -1 on failure
 *
 * Effects:
 *
 * The host name is resolved with gethostbyname(), and the socket is
 * opened and connected.  If an error occurs, an error message is
 * displayed and -1 is returned.
 */
int connect_to_server(host, port)
     char *host;
     u_short port;
{
```

示例 A–1 gss-client.c样例程序的完整列表 （续）

```
     struct sockaddr_in saddr;
     struct hostent *hp;
     int s;

     if ((hp = gethostbyname(host)) == NULL) {
          fprintf(stderr, "Unknown host: %s\n", host);
          return -1;
     }

     saddr.sin_family = hp->h_addrtype;
     memcpy((char *)&saddr.sin_addr, hp->h_addr, sizeof(saddr.sin_addr));
     saddr.sin_port = htons(port);

     if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
          perror("creating socket");
          return -1;
     }
     if (connect(s, (struct sockaddr *)&saddr, sizeof(saddr)) < 0) {
          perror("connecting to server");
          (void) close(s);
          return -1;
     }
     return s;
}

/*
 * Function: client_establish_context
 *
 * Purpose: establishes a GSS-API context with a specified service and
 * returns the context handle
 *
 * Arguments:
 *
 *      s               (r) an established TCP connection to the service
 *      service_name    (r) the ASCII service name of the service
 *      context         (w) the established GSS-API context
 *      ret_flags       (w) the returned flags from init_sec_context
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * service_name is imported as a GSS-API name and a GSS-API context is
 * established with the corresponding service; the service should be
 * listening on the TCP connection s.  The default GSS-API mechanism
 * is used, and mutual authentication and replay detection are
 * requested.
 *
 * If successful, the context handle is returned in context.  If
 * unsuccessful, the GSS-API error messages are displayed on stderr
 * and -1 is returned.
 */
int client_establish_context(s, service_name, deleg_flag, oid,
                             gss_context, ret_flags)
     int s;
     char *service_name;
```

示例 A–1 gss-client.c样例程序的完整列表 （续）

```
     gss_OID oid;
     OM_uint32 deleg_flag;
     gss_ctx_id_t *gss_context;
     OM_uint32 *ret_flags;
{
     gss_buffer_desc send_tok, recv_tok, *token_ptr;
     gss_name_t target_name;
     OM_uint32 maj_stat, min_stat, init_sec_min_stat;

     /*
      * Import the name into target_name.  Use send_tok to save
      * local variable space.
      */
     send_tok.value = service_name;
     send_tok.length = strlen(service_name) + 1;
     maj_stat = gss_import_name(&min_stat, &send_tok,
         (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &target_name);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("parsing name", maj_stat, min_stat);
          return -1;
     }

     /*
      * Perform the context-establishement loop.
      *
      * On each pass through the loop, token_ptr points to the token
      * to send to the server (or GSS_C_NO_BUFFER on the first pass).
      * Every generated token is stored in send_tok which is then
      * transmitted to the server; every received token is stored in
      * recv_tok, which token_ptr is then set to, to be processed by
      * the next call to gss_init_sec_context.
      *
      * GSS-API guarantees that send_tok's length will be non-zero
      * if and only if the server is expecting another token from us,
      * and that gss_init_sec_context returns GSS_S_CONTINUE_NEEDED if
      * and only if the server has another token to send us.
      */

     token_ptr = GSS_C_NO_BUFFER;
     *gss_context = GSS_C_NO_CONTEXT;

     do {
          maj_stat =
               gss_init_sec_context(&init_sec_min_stat,
                                    GSS_C_NO_CREDENTIAL,
                                    gss_context,
                                    target_name,
                                    oid,
                                    GSS_C_MUTUAL_FLAG | GSS_C_REPLAY_FLAG |
                                                        deleg_flag,
                                    0,
                                    NULL,       /* no channel bindings */
                                    token_ptr,
                                    NULL,       /* ignore mech type */
                                    &send_tok,
                                    ret_flags,
```

示例 A–1 gss-client.c样例程序的完整列表 （续）

```
                                    NULL);      /* ignore time_rec */

          if (token_ptr != GSS_C_NO_BUFFER)
               (void) gss_release_buffer(&min_stat, &recv_tok);

          if (send_tok.length != 0) {
               printf("Sending init_sec_context token (size=%d)...",
                     send_tok.length);
               if (send_token(s, &send_tok) < 0) {
                    (void) gss_release_buffer(&min_stat, &send_tok);
                    (void) gss_release_name(&min_stat, &target_name);
                    return -1;
               }
          }
          (void) gss_release_buffer(&min_stat, &send_tok);

          if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
               display_status("initializing context", maj_stat,
                              init_sec_min_stat);
               (void) gss_release_name(&min_stat, &target_name);
               if (*gss_context == GSS_C_NO_CONTEXT)
                       gss_delete_sec_context(&min_stat, gss_context,
                                              GSS_C_NO_BUFFER);
               return -1;
          }

          if (maj_stat == GSS_S_CONTINUE_NEEDED) {
               printf("continue needed...");
               if (recv_token(s, &recv_tok) < 0) {
                    (void) gss_release_name(&min_stat, &target_name);
                    return -1;
               }
               token_ptr = &recv_tok;
          }
          printf("\n");
     } while (maj_stat == GSS_S_CONTINUE_NEEDED);

     (void) gss_release_name(&min_stat, &target_name);
     return 0;
}

void read_file(file_name, in_buf)
    char               *file_name;
    gss_buffer_t        in_buf;
{
    int fd, bytes_in, count;
    struct stat stat_buf;

    if ((fd = open(file_name, O_RDONLY, 0)) < 0) {
        perror("open");
        fprintf(stderr, "Couldn't open file %s\n", file_name);
        exit(1);
    }
    if (fstat(fd, &stat_buf) < 0) {
        perror("fstat");
        exit(1);
```

示例 A–1 gss-client.c样例程序的完整列表 （续）

```
    }
    in_buf->length = stat_buf.st_size;

    if (in_buf->length == 0) {
        in_buf->value = NULL;
        return;
    }

    if ((in_buf->value = malloc(in_buf->length)) == 0) {
        fprintf(stderr, \
            "Couldn't allocate %d byte buffer for reading file\n",
            in_buf->length);
        exit(1);
    }

    /* this code used to check for incomplete reads, but you can't get
       an incomplete read on any file for which fstat() is meaningful */

    count = read(fd, in_buf->value, in_buf->length);
    if (count < 0) {
        perror("read");
        exit(1);
    }
    if (count < in_buf->length)
        fprintf(stderr, "Warning, only read in %d bytes, expected %d\n",
                count, in_buf->length);
}

/*
 * Function: call_server
 *
 * Purpose: Call the "sign" service.
 *
 * Arguments:
 *
 *      host            (r) the host providing the service
 *      port            (r) the port to connect to on host
 *      service_name    (r) the GSS-API service name to authenticate to
 *      msg             (r) the message to have "signed"
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * call_server opens a TCP connection to <host:port> and establishes a
 * GSS-API context with service_name over the connection.  It then
 * seals msg in a GSS-API token with gss_seal, sends it to the server,
 * reads back a GSS-API signature block for msg from the server, and
 * verifies it with gss_verify.  -1 is returned if any step fails,
 * otherwise 0 is returned.  */
int call_server(host, port, oid, service_name, deleg_flag, msg, use_file)
     char *host;
     u_short port;
     gss_OID oid;
     char *service_name;
     OM_uint32 deleg_flag;
```

示例 A–1 gss-client.c样例程序的完整列表 （续）

```
     char *msg;
     int use_file;
{
     gss_ctx_id_t context;
     gss_buffer_desc in_buf, out_buf;
     int s, state;
     OM_uint32 ret_flags;
     OM_uint32 maj_stat, min_stat;
     gss_name_t         src_name, targ_name;
     gss_buffer_desc    sname, tname;
     OM_uint32          lifetime;
     gss_OID            mechanism, name_type;
     int                is_local;
     OM_uint32          context_flags;
     int                is_open;
     gss_qop_t          qop_state;
     gss_OID_set        mech_names;
     gss_buffer_desc    oid_name;
     size_t     i;

     /* Open connection */
     if ((s = connect_to_server(host, port)) < 0)
          return -1;

     /* Establish context */
     if (client_establish_context(s, service_name, deleg_flag, oid,
         &context, &ret_flags) < 0) {
         (void) close(s);
         return -1;
     }

     /* display the flags */
     display_ctx_flags(ret_flags);

     /* Get context information */
     maj_stat = gss_inquire_context(&min_stat, context,
                                    &src_name, &targ_name, &lifetime,
                                    &mechanism, &context_flags,
                                    &is_local,
                                    &is_open);
     if (maj_stat != GSS_S_COMPLETE) {
         display_status("inquiring context", maj_stat, min_stat);
         return -1;
     }

     maj_stat = gss_display_name(&min_stat, src_name, &sname,
                                 &name_type);
     if (maj_stat != GSS_S_COMPLETE) {
         display_status("displaying source name", maj_stat, min_stat);
         return -1;
     }
     maj_stat = gss_display_name(&min_stat, targ_name, &tname,
                                 (gss_OID *) NULL);
     if (maj_stat != GSS_S_COMPLETE) {
         display_status("displaying target name", maj_stat, min_stat);
         return -1;
```

示例 A–1 gss-client.c样例程序的完整列表 （续）

```
    }
    fprintf(stderr, "\"%.*s\" to \"%.*s\", lifetime %d, flags %x, %s,
            %s\n", (int) sname.length, (char *) sname.value,
            (int) tname.length, (char *) tname.value, lifetime,
            context_flags,
            (is_local) ? "locally initiated" : "remotely initiated",
            (is_open) ? "open" : "closed");

    (void) gss_release_name(&min_stat, &src_name);
    (void) gss_release_name(&min_stat, &targ_name);
    (void) gss_release_buffer(&min_stat, &sname);
    (void) gss_release_buffer(&min_stat, &tname);

    maj_stat = gss_oid_to_str(&min_stat,
                              name_type,
                              &oid_name);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("converting oid->string", maj_stat, min_stat);
        return -1;
    }
    fprintf(stderr, "Name type of source name is %.*s.\n",
            (int) oid_name.length, (char *) oid_name.value);
    (void) gss_release_buffer(&min_stat, &oid_name);

    /* Now get the names supported by the mechanism */
    maj_stat = gss_inquire_names_for_mech(&min_stat,
                                          mechanism,
                                          &mech_names);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("inquiring mech names", maj_stat, min_stat);
        return -1;
    }

    maj_stat = gss_oid_to_str(&min_stat,
                              mechanism,
                              &oid_name);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("converting oid->string", maj_stat, min_stat);
        return -1;
    }
    fprintf(stderr, "Mechanism %.*s supports %d names\n",
            (int) oid_name.length, (char *) oid_name.value,
            mech_names->count);
    (void) gss_release_buffer(&min_stat, &oid_name);

    for (i=0; i<mech_names->count; i++) {
        maj_stat = gss_oid_to_str(&min_stat,
                                  &mech_names->elements[i],
                                  &oid_name);
        if (maj_stat != GSS_S_COMPLETE) {
            display_status("converting oid->string", maj_stat, min_stat);
            return -1;
        }
        fprintf(stderr, "  %d: %.*s\n", i,
                (int) oid_name.length, (char *) oid_name.value);
```

示例 A–1 gss-client.c样例程序的完整列表 （续）

```
        (void) gss_release_buffer(&min_stat, &oid_name);
    }
    (void) gss_release_oid_set(&min_stat, &mech_names);

    if (use_file) {
        read_file(msg, &in_buf);
    } else {
        /* Seal the message */
        in_buf.value = msg;
        in_buf.length = strlen(msg);
    }

    maj_stat = gss_wrap(&min_stat, context, 1, GSS_C_QOP_DEFAULT,
                        &in_buf, &state, &out_buf);
    if (maj_stat != GSS_S_COMPLETE) {
         display_status("sealing message", maj_stat, min_stat);
         (void) close(s);
         (void) gss_delete_sec_context(&min_stat, &context,
             GSS_C_NO_BUFFER);
         return -1;
    } else if (! state) {
         fprintf(stderr, "Warning!  Message not encrypted.\n");
    }

    /* Send to server */
    if (send_token(s, &out_buf) < 0) {
         (void) close(s);
         (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
         return -1;
    }
    (void) gss_release_buffer(&min_stat, &out_buf);

    /* Read signature block into out_buf */
    if (recv_token(s, &out_buf) < 0) {
         (void) close(s);
         (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
         return -1;
    }

    /* Verify signature block */
    maj_stat = gss_verify_mic(&min_stat, context, &in_buf,
                              &out_buf, &qop_state);
    if (maj_stat != GSS_S_COMPLETE) {
         display_status("verifying signature", maj_stat, min_stat);
         (void) close(s);
         (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
         return -1;
    }
    (void) gss_release_buffer(&min_stat, &out_buf);

    if (use_file)
        free(in_buf.value);

    printf("Signature verified.\n");

    /* Delete context */
```

示例 A–1 gss-client.c样例程序的完整列表 （续）

```
     maj_stat = gss_delete_sec_context(&min_stat, &context, &out_buf);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("deleting context", maj_stat, min_stat);
          (void) close(s);
          (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
          return -1;
     }

     (void) gss_release_buffer(&min_stat, &out_buf);
     (void) close(s);
     return 0;
}

static void parse_oid(char *mechanism, gss_OID *oid)
{
    char        *mechstr = 0, *cp;
    gss_buffer_desc tok;
    OM_uint32 maj_stat, min_stat;

    if (isdigit(mechanism[0])) {
        mechstr = malloc(strlen(mechanism)+5);
        if (!mechstr) {
            printf("Couldn't allocate mechanism scratch!\n");
            return;
        }
        sprintf(mechstr, "{ %s }", mechanism);
        for (cp = mechstr; *cp; cp++)
            if (*cp == '.')
                *cp = ' ';
        tok.value = mechstr;
    } else
        tok.value = mechanism;
    tok.length = strlen(tok.value);
    maj_stat = gss_str_to_oid(&min_stat, &tok, oid);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("str_to_oid", maj_stat, min_stat);
        return;
    }
    if (mechstr)
        free(mechstr);
}

int main(argc, argv)
     int argc;
     char **argv;
{
     char *service_name, *server_host, *msg;
     char *mechanism = 0;
     u_short port = 4444;
     int use_file = 0;
     OM_uint32 deleg_flag = 0, min_stat;
     gss_OID oid = GSS_C_NULL_OID;

     display_file = stdout;

     /* Parse arguments. */
```

示例 A–1 gss-client.c样例程序的完整列表 （续）

```
     argc--; argv++;
     while (argc) {
          if (strcmp(*argv, "-port") == 0) {
               argc--; argv++;
               if (!argc) usage();
               port = atoi(*argv);
          } else if (strcmp(*argv, "-mech") == 0) {
               argc--; argv++;
               if (!argc) usage();
               mechanism = *argv;
          } else if (strcmp(*argv, "-d") == 0) {
               deleg_flag = GSS_C_DELEG_FLAG;
          } else if (strcmp(*argv, "-f") == 0) {
               use_file = 1;
          } else
               break;
          argc--; argv++;
     }
     if (argc != 3)
          usage();

     server_host = *argv++;
     service_name = *argv++;
     msg = *argv++;

     if (mechanism)
         parse_oid(mechanism, &oid);

     if (call_server(server_host, port, oid, service_name,
                     deleg_flag, msg, use_file) < 0)
          exit(1);

     if (oid != GSS_C_NULL_OID)
         (void) gss_release_oid(&min_stat, &oid);

     return 0;
}
```

服务器端应用程序

以下示例提供了服务器端程序 gss_server 的源代码。

示例 A–2 gss-server.c样例程序的完整代码列表

```
/*
 * Copyright 1994 by OpenVision Technologies, Inc.
 *
 * Permission to use, copy, modify, distribute, and sell this software
 * and its documentation for any purpose is hereby granted without fee,
 * provided that the above copyright notice appears in all copies and
 * that both that copyright notice and this permission notice appear in
 * supporting documentation, and that the name of OpenVision not be used
 * in advertising or publicity pertaining to distribution of the software
```

示例 A–2 gss-server.c样例程序的完整代码列表 （续）

```
 * without specific, written prior permission. OpenVision makes no
 * representations about the suitability of this software for any
 * purpose.  It is provided "as is" without express or implied warranty.
 *
 * OPENVISION DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
 * INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
 * EVENT SHALL OPENVISION BE LIABLE FOR ANY SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
 * USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
 * OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 * PERFORMANCE OF THIS SOFTWARE.
 */

#if !defined(lint) && !defined(__CODECENTER__)
static char *rcsid = \
"$Header: /cvs/krbdev/krb5/src/appl/gss-sample/gss-server.c, \
    v 1.21 1998/12/22 \
04:10:08 tytso Exp $";
#endif

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdlib.h>
#include <ctype.h>

#include <gssapi/gssapi.h>
#include <gssapi/gssapi_ext.h>
#include <gss-misc.h>

#include <string.h>

void usage()
{
     fprintf(stderr, "Usage: gss-server [-port port] [-verbose]\n");
     fprintf(stderr, "       [-inetd] [-logfile file] [service_name]\n");
     exit(1);
}

FILE *log;

int verbose = 0;

/*
 * Function: server_acquire_creds
 *
 * Purpose: imports a service name and acquires credentials for it
 *
 * Arguments:
 *
 *      service_name    (r) the ASCII service name
 *      server_creds    (w) the GSS-API service credentials
 *
```

示例 A–2 gss-server.c样例程序的完整代码列表 （续）

```
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * The service name is imported with gss_import_name, and service
 * credentials are acquired with gss_acquire_cred.  If either operation
 * fails, an error message is displayed and -1 is returned; otherwise,
 * 0 is returned.
 */
int server_acquire_creds(service_name, server_creds)
     char *service_name;
     gss_cred_id_t *server_creds;
{
     gss_buffer_desc name_buf;
     gss_name_t server_name;
     OM_uint32 maj_stat, min_stat;

     name_buf.value = service_name;
     name_buf.length = strlen(name_buf.value) + 1;
     maj_stat = gss_import_name(&min_stat, &name_buf,
         (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &server_name);
     if (maj_stat != GSS_S_COMPLETE) {
         display_status("importing name", maj_stat, min_stat);
         return -1;
     }

     maj_stat = gss_acquire_cred(&min_stat, server_name, 0,
                                 GSS_C_NULL_OID_SET, GSS_C_ACCEPT,
                                 server_creds, NULL, NULL);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("acquiring credentials", maj_stat, min_stat);
          return -1;
     }

     (void) gss_release_name(&min_stat, &server_name);

     return 0;
}

/*
 * Function: server_establish_context
 *
 * Purpose: establishes a GSS-API context as a specified service with
 * an incoming client, and returns the context handle and associated
 * client name
 *
 * Arguments:
 *
 *      s               (r) an established TCP connection to the client
 *      service_creds   (r) server credentials, from gss_acquire_cred
 *      context         (w) the established GSS-API context
 *      client_name     (w) the client's ASCII name
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
```

示例 A–2 gss-server.c样例程序的完整代码列表 （续）

```
 *
 * Any valid client request is accepted.  If a context is established,
 * its handle is returned in context and the client name is returned
 * in client_name and 0 is returned.  If unsuccessful, an error
 * message is displayed and -1 is returned.
 */
int server_establish_context(s, server_creds, context, client_name, \
     ret_flags)

     int s;
     gss_cred_id_t server_creds;
     gss_ctx_id_t *context;
     gss_buffer_t client_name;
     OM_uint32 *ret_flags;
{
     gss_buffer_desc send_tok, recv_tok;
     gss_name_t client;
     gss_OID doid;
     OM_uint32 maj_stat, min_stat, acc_sec_min_stat;
     gss_buffer_desc    oid_name;

     *context = GSS_C_NO_CONTEXT;

     do {
          if (recv_token(s, &recv_tok) < 0)
               return -1;

          if (verbose && log) {
              fprintf(log, "Received token (size=%d): \n", recv_tok.length);
              print_token(&recv_tok);
          }

          maj_stat =
               gss_accept_sec_context(&acc_sec_min_stat,
                                      context,
                                      server_creds,
                                      &recv_tok,
                                      GSS_C_NO_CHANNEL_BINDINGS,
                                      &client,
                                      &doid,
                                      &send_tok,
                                      ret_flags,
                                      NULL,     /* ignore time_rec */
                                      NULL);    /* ignore del_cred_handle */

          (void) gss_release_buffer(&min_stat, &recv_tok);

          if (send_tok.length != 0) {
              if (verbose && log) {
                  fprintf(log,
                          "Sending accept_sec_context token (size=%d):\n",
                          send_tok.length);
                  print_token(&send_tok);
              }
              if (send_token(s, &send_tok) < 0) {
                  fprintf(log, "failure sending token\n");
```

示例 A–2 gss-server.c样例程序的完整代码列表 （续）

```
                return -1;
            }

            (void) gss_release_buffer(&min_stat, &send_tok);
        }
        if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
            display_status("accepting context", maj_stat,
                           acc_sec_min_stat);
            if (*context == GSS_C_NO_CONTEXT)
                    gss_delete_sec_context(&min_stat, context,
                                           GSS_C_NO_BUFFER);
            return -1;
        }

        if (verbose && log) {
            if (maj_stat == GSS_S_CONTINUE_NEEDED)
                fprintf(log, "continue needed...\n");
            else
                fprintf(log, "\n");
            fflush(log);
        }
    } while (maj_stat == GSS_S_CONTINUE_NEEDED);

    /* display the flags */
    display_ctx_flags(*ret_flags);

    if (verbose && log) {
        maj_stat = gss_oid_to_str(&min_stat, doid, &oid_name);
        if (maj_stat != GSS_S_COMPLETE) {
            display_status("converting oid->string", maj_stat, min_stat);
            return -1;
        }
        fprintf(log, "Accepted connection using mechanism OID %.*s.\n",
                (int) oid_name.length, (char *) oid_name.value);
        (void) gss_release_buffer(&min_stat, &oid_name);
    }

    maj_stat = gss_display_name(&min_stat, client, client_name, &doid);
    if (maj_stat != GSS_S_COMPLETE) {
         display_status("displaying name", maj_stat, min_stat);
         return -1;
    }
    maj_stat = gss_release_name(&min_stat, &client);
    if (maj_stat != GSS_S_COMPLETE) {
         display_status("releasing name", maj_stat, min_stat);
         return -1;
    }
    return 0;
}

/*
 * Function: create_socket
 *
 * Purpose: Opens a listening TCP socket.
 *
 * Arguments:
```

示例 A–2 gss-server.c样例程序的完整代码列表 （续）

```
 *
 *      port            (r) the port number on which to listen
 *
 * Returns: the listening socket file descriptor, or -1 on failure
 *
 * Effects:
 *
 * A listening socket on the specified port is created and returned.
 * On error, an error message is displayed and -1 is returned.
 */
int create_socket(port)
     u_short port;
{
     struct sockaddr_in saddr;
     int s;
     int on = 1;

     saddr.sin_family = AF_INET;
     saddr.sin_port = htons(port);
     saddr.sin_addr.s_addr = INADDR_ANY;

     if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
          perror("creating socket");
          return -1;
     }
     /* Let the socket be reused right away */
     (void) setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *)&on,
          sizeof(on));
     if (bind(s, (struct sockaddr *) &saddr, sizeof(saddr)) < 0) {
          perror("binding socket");
          (void) close(s);
          return -1;
     }
     if (listen(s, 5) < 0) {
          perror("listening on socket");
          (void) close(s);
          return -1;
     }
     return s;
}

static float timeval_subtract(tv1, tv2)
        struct timeval *tv1, *tv2;
{
        return ((tv1->tv_sec - tv2->tv_sec) +
                ((float) (tv1->tv_usec - tv2->tv_usec)) / 1000000);
}

/*
 * Yes, yes, this isn't the best place for doing this test.
 * DO NOT REMOVE THIS UNTIL A BETTER TEST HAS BEEN WRITTEN, THOUGH.
 *                                          -TYT
 */
int test_import_export_context(context)
        gss_ctx_id_t *context;
{
```

示例 A–2 gss-server.c样例程序的完整代码列表 （续）

```
        OM_uint32       min_stat, maj_stat;
        gss_buffer_desc context_token, copied_token;
        struct timeval tm1, tm2;

        /*
         * Attempt to save and then restore the context.
         */
        gettimeofday(&tm1, (struct timezone *)0);
        maj_stat = gss_export_sec_context(&min_stat, context, \
            &context_token);
        if (maj_stat != GSS_S_COMPLETE) {
                display_status("exporting context", maj_stat, min_stat);
                return 1;
        }
        gettimeofday(&tm2, (struct timezone *)0);
        if (verbose && log)
                fprintf(log, "Exported context: %d bytes, %7.4f seconds\n",
                        context_token.length, timeval_subtract(&tm2, &tm1));
        copied_token.length = context_token.length;
        copied_token.value = malloc(context_token.length);
        if (copied_token.value == 0) {
            fprintf(log, "Couldn't allocate memory to copy context \
                token.\n");
            return 1;
        }
        memcpy(copied_token.value, context_token.value, \
            copied_token.length);
        maj_stat = gss_import_sec_context(&min_stat, &copied_token, \
            context);
        if (maj_stat != GSS_S_COMPLETE) {
                display_status("importing context", maj_stat, min_stat);
                return 1;
        }
        free(copied_token.value);
        gettimeofday(&tm1, (struct timezone *)0);
        if (verbose && log)
                fprintf(log, "Importing context: %7.4f seconds\n",
                        timeval_subtract(&tm1, &tm2));
        (void) gss_release_buffer(&min_stat, &context_token);
        return 0;
}

/*
 * Function: sign_server
 *
 * Purpose: Performs the "sign" service.
 *
 * Arguments:
 *
 *      s               (r) a TCP socket on which a connection has been
 *                      accept()ed
 *      service_name    (r) the ASCII name of the GSS-API service to
 *                      establish a context as
 *
 * Returns: -1 on error
 *
```

示例 A–2 gss-server.c样例程序的完整代码列表 （续）

```
 * Effects:
 *
 * sign_server establishes a context, and performs a single sign request.
 *
 * A sign request is a single GSS-API sealed token.  The token is
 * unsealed and a signature block, produced with gss_sign, is returned
 * to the sender.  The context is then destroyed and the connection
 * closed.
 *
 * If any error occurs, -1 is returned.
 */
int sign_server(s, server_creds)
     int s;
     gss_cred_id_t server_creds;
{
     gss_buffer_desc client_name, xmit_buf, msg_buf;
     gss_ctx_id_t context;
     OM_uint32 maj_stat, min_stat;
     int i, conf_state, ret_flags;
     char       *cp;

     /* Establish a context with the client */
     if (server_establish_context(s, server_creds, &context,
                                  &client_name, &ret_flags) < 0)
        return(-1);

     printf("Accepted connection: \"%.*s\"\n",
            (int) client_name.length, (char *) client_name.value);
     (void) gss_release_buffer(&min_stat, &client_name);

     for (i=0; i < 3; i++)
             if (test_import_export_context(&context))
                     return -1;

     /* Receive the sealed message token */
     if (recv_token(s, &xmit_buf) < 0)
        return(-1);

     if (verbose && log) {
        fprintf(log, "Sealed message token:\n");
        print_token(&xmit_buf);
     }

     maj_stat = gss_unwrap(&min_stat, context, &xmit_buf, &msg_buf,
                           &conf_state, (gss_qop_t *) NULL);
     if (maj_stat != GSS_S_COMPLETE) {
        display_status("unsealing message", maj_stat, min_stat);
        return(-1);
     } else if (! conf_state) {
        fprintf(stderr, "Warning!  Message not encrypted.\n");
     }

     (void) gss_release_buffer(&min_stat, &xmit_buf);

     fprintf(log, "Received message: ");
     cp = msg_buf.value;
```

示例 A–2 gss-server.c样例程序的完整代码列表 （续）

```
     if ((isprint(cp[0]) || isspace(cp[0])) &&
         (isprint(cp[1]) || isspace(cp[1]))) {
        fprintf(log, "\"%.*s\"\n", msg_buf.length, msg_buf.value);
     } else {
        printf("\n");
        print_token(&msg_buf);
     }

     /* Produce a signature block for the message */
     maj_stat = gss_get_mic(&min_stat, context, GSS_C_QOP_DEFAULT,
                            &msg_buf, &xmit_buf);
     if (maj_stat != GSS_S_COMPLETE) {
        display_status("signing message", maj_stat, min_stat);
        return(-1);
     }

     (void) gss_release_buffer(&min_stat, &msg_buf);

     /* Send the signature block to the client */
     if (send_token(s, &xmit_buf) < 0)
        return(-1);

     (void) gss_release_buffer(&min_stat, &xmit_buf);

     /* Delete context */
     maj_stat = gss_delete_sec_context(&min_stat, &context, NULL);
     if (maj_stat != GSS_S_COMPLETE) {
        display_status("deleting context", maj_stat, min_stat);
        return(-1);
     }

     fflush(log);

     return(0);
}

int
main(argc, argv)
     int argc;
     char **argv;
{
     char *service_name;
     gss_cred_id_t server_creds;
     OM_uint32 min_stat;
     u_short port = 4444;
     int s;
     int once = 0;
     int do_inetd = 0;

     log = stdout;
     display_file = stdout;
     argc--; argv++;
     while (argc) {
          if (strcmp(*argv, "-port") == 0) {
               argc--; argv++;
               if (!argc) usage();
```

示例 A–2 gss-server.c样例程序的完整代码列表 （续）

```
            port = atoi(*argv);
        } else if (strcmp(*argv, "-verbose") == 0) {
            verbose = 1;
        } else if (strcmp(*argv, "-once") == 0) {
            once = 1;
        } else if (strcmp(*argv, "-inetd") == 0) {
            do_inetd = 1;
        } else if (strcmp(*argv, "-logfile") == 0) {
            argc--; argv++;
            if (!argc) usage();
            log = fopen(*argv, "a");
            display_file = log;
            if (!log) {
                perror(*argv);
                exit(1);
            }
        } else
             break;
        argc--; argv++;
    }
    if (argc != 1)
        usage();

    if ((*argv)[0] == '-')
        usage();

    service_name = *argv;

    if (server_acquire_creds(service_name, &server_creds) < 0)
        return -1;

    if (do_inetd) {
        close(1);
        close(2);

        sign_server(0, server_creds);
        close(0);
    } else {
        int stmp;

        if ((stmp = create_socket(port)) >= 0) {
            do {
                /* Accept a TCP connection */
                if ((s = accept(stmp, NULL, 0)) < 0) {
                    perror("accepting connection");
                    continue;
                }
                /* this return value is not checked, because there's
                   not really anything to do if it fails */
                sign_server(s, server_creds);
                close(s);
            } while (!once);

            close(stmp);
        }
    }
```

示例 A–2 gss-server.c 样例程序的完整代码列表 （续）

```
     (void) gss_release_cred(&min_stat, &server_creds);

     /*NOTREACHED*/
     (void) close(s);
     return 0;
}
```

各种 GSS-API 样例函数

要使客户机和服务器程序按所示方式工作，还需要大量其他函数。这些函数用于显示值。在其他情况下不需要这些函数。以下是此类别的函数：

- `send_token()` – 将令牌和消息传输给收件人
- `recv_token()` – 接受来自发件人的令牌和消息
- `display_status()` – 显示上次调用 GSS-API 函数时返回的状态
- `write_all()` – 将缓冲区中的信息写入文件
- `read_all()` – 将文件读入到缓冲区中
- `display_ctx_flags()` – 以可读格式显示有关当前上下文的信息，如是否允许保密性或相互验证
- `print_token()` – 输出令牌的值

以下示例中给出了这些函数的代码。

示例 A–3 各种 GSS-API 函数的代码列表

```
/*
 * Copyright 1994 by OpenVision Technologies, Inc.
 *
 * Permission to use, copy, modify, distribute, and sell this software
 * and its documentation for any purpose is hereby granted without fee,
 * provided that the above copyright notice appears in all copies and
 * that both that copyright notice and this permission notice appear in
 * supporting documentation, and that the name of OpenVision not be used
 * in advertising or publicity pertaining to distribution of the software
 * without specific, written prior permission. OpenVision makes no
 * representations about the suitability of this software for any
 * purpose.  It is provided "as is" without express or implied warranty.
 *
 * OPENVISION DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
 * INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
 * EVENT SHALL OPENVISION BE LIABLE FOR ANY SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
 * USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
 * OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 * PERFORMANCE OF THIS SOFTWARE.
 */
```

示例 A–3 各种 GSS-API 函数的代码列表 （续）

```
#if !defined(lint) && !defined(__CODECENTER__)
static char *rcsid = "$Header: /cvs/krbdev/krb5/src/appl/gss-sample/\
    gss-misc.c, v 1.15 1996/07/22 20:21:20 marc Exp $";
#endif

#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>

#include <gssapi/gssapi.h>
#include <gssapi/gssapi_ext.h>
#include <gss-misc.h>

#include <stdlib.h>

FILE *display_file;

static void display_status_1
        (char *m, OM_uint32 code, int type);

static int write_all(int fildes, char *buf, unsigned int nbyte)
{
     int ret;
     char *ptr;

     for (ptr = buf; nbyte; ptr += ret, nbyte -= ret) {
          ret = write(fildes, ptr, nbyte);
          if (ret < 0) {
               if (errno == EINTR)
                    continue;
               return(ret);
          } else if (ret == 0) {
               return(ptr-buf);
          }
     }

     return(ptr-buf);
}

static int read_all(int fildes, char *buf, unsigned int nbyte)
{
     int ret;
     char *ptr;

     for (ptr = buf; nbyte; ptr += ret, nbyte -= ret) {
          ret = read(fildes, ptr, nbyte);
          if (ret < 0) {
               if (errno == EINTR)
                    continue;
               return(ret);
          } else if (ret == 0) {
               return(ptr-buf);
```

示例 A–3 各种GSS-API函数的代码列表 （续）

```
          }
     }

     return(ptr-buf);
}

/*
 * Function: send_token
 *
 * Purpose: Writes a token to a file descriptor.
 *
 * Arguments:
 *
 *      s               (r) an open file descriptor
 *      tok             (r) the token to write
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * send_token writes the token length (as a network long) and then the
 * token data to the file descriptor s.  It returns 0 on success, and
 * -1 if an error occurs or if it could not write all the data.
 */
int send_token(s, tok)
     int s;
     gss_buffer_t tok;
{
     int len, ret;

     len = htonl(tok->length);

     ret = write_all(s, (char *) &len, 4);
     if (ret < 0) {
          perror("sending token length");
          return -1;
     } else if (ret != 4) {
         if (display_file)
             fprintf(display_file,
                     "sending token length: %d of %d bytes written\n",
                     ret, 4);
          return -1;
     }

     ret = write_all(s, tok->value, tok->length);
     if (ret < 0) {
          perror("sending token data");
          return -1;
     } else if (ret != tok->length) {
         if (display_file)
             fprintf(display_file,
                     "sending token data: %d of %d bytes written\n",
                     ret, tok->length);
         return -1;
     }
```

示例 A–3 各种 GSS-API 函数的代码列表 （续）

```
     return 0;
}

/*
 * Function: recv_token
 *
 * Purpose: Reads a token from a file descriptor.
 *
 * Arguments:
 *
 *      s               (r) an open file descriptor
 *      tok             (w) the read token
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * recv_token reads the token length (as a network long), allocates
 * memory to hold the data, and then reads the token data from the
 * file descriptor s.  It blocks to read the length and data, if
 * necessary.  On a successful return, the token should be freed with
 * gss_release_buffer.  It returns 0 on success, and -1 if an error
 * occurs or if it could not read all the data.
 */
int recv_token(s, tok)
     int s;
     gss_buffer_t tok;
{
     int ret;

     ret = read_all(s, (char *) &tok->length, 4);
     if (ret < 0) {
          perror("reading token length");
          return -1;
     } else if (ret != 4) {
         if (display_file)
             fprintf(display_file,
                     "reading token length: %d of %d bytes read\n",
                     ret, 4);
         return -1;
     }

     tok->length = ntohl(tok->length);
     tok->value = (char *) malloc(tok->length);
     if (tok->value == NULL) {
         if (display_file)
             fprintf(display_file,
                     "Out of memory allocating token data\n");
          return -1;
     }

     ret = read_all(s, (char *) tok->value, tok->length);
     if (ret < 0) {
          perror("reading token data");
          free(tok->value);
          return -1;
```

示例 A–3 各种GSS-API函数的代码列表 （续）

```
     } else if (ret != tok->length) {
          fprintf(stderr, "sending token data: %d of %d bytes written\n",
                  ret, tok->length);
          free(tok->value);
          return -1;
     }

     return 0;
}

static void display_status_1(m, code, type)
     char *m;
     OM_uint32 code;
     int type;
{
     OM_uint32 maj_stat, min_stat;
     gss_buffer_desc msg;
     OM_uint32 msg_ctx;

     msg_ctx = 0;
     while (1) {
          maj_stat = gss_display_status(&min_stat, code,
                                        type, GSS_C_NULL_OID,
                                        &msg_ctx, &msg);
          if (display_file)
              fprintf(display_file, "GSS-API error %s: %s\n", m,
                      (char *)msg.value);
          (void) gss_release_buffer(&min_stat, &msg);

          if (!msg_ctx)
               break;
     }
}

/*
 * Function: display_status
 *
 * Purpose: displays GSS-API messages
 *
 * Arguments:
 *
 *      msg             a string to be displayed with the message
 *      maj_stat        the GSS-API major status code
 *      min_stat        the GSS-API minor status code
 *
 * Effects:
 *
 * The GSS-API messages associated with maj_stat and min_stat are
 * displayed on stderr, each preceded by "GSS-API error <msg>: " and
 * followed by a newline.
 */
void display_status(msg, maj_stat, min_stat)
     char *msg;
     OM_uint32 maj_stat;
     OM_uint32 min_stat;
{
```

示例 A–3 各种 GSS-API 函数的代码列表 （续）

```
     display_status_1(msg, maj_stat, GSS_C_GSS_CODE);
     display_status_1(msg, min_stat, GSS_C_MECH_CODE);
}

/*
 * Function: display_ctx_flags
 *
 * Purpose: displays the flags returned by context initiation in
 *          a human-readable form
 *
 * Arguments:
 *
 *      int             ret_flags
 *
 * Effects:
 *
 * Strings corresponding to the context flags are printed on
 * stdout, preceded by "context flag: " and followed by a newline
 */

void display_ctx_flags(flags)
     OM_uint32 flags;
{
     if (flags & GSS_C_DELEG_FLAG)
          fprintf(display_file, "context flag: GSS_C_DELEG_FLAG\n");
     if (flags & GSS_C_MUTUAL_FLAG)
          fprintf(display_file, "context flag: GSS_C_MUTUAL_FLAG\n");
     if (flags & GSS_C_REPLAY_FLAG)
          fprintf(display_file, "context flag: GSS_C_REPLAY_FLAG\n");
     if (flags & GSS_C_SEQUENCE_FLAG)
          fprintf(display_file, "context flag: GSS_C_SEQUENCE_FLAG\n");
     if (flags & GSS_C_CONF_FLAG )
          fprintf(display_file, "context flag: GSS_C_CONF_FLAG \n");
     if (flags & GSS_C_INTEG_FLAG )
          fprintf(display_file, "context flag: GSS_C_INTEG_FLAG \n");
}

void print_token(tok)
     gss_buffer_t tok;
{
    int i;
    unsigned char *p = tok->value;

    if (!display_file)
        return;
    for (i=0; i < tok->length; i++, p++) {
        fprintf(display_file, "%02x ", *p);
        if ((i % 16) == 15) {
            fprintf(display_file, "\n");
        }
    }
    fprintf(display_file, "\n");
    fflush(display_file);
}
```

◆ ◆ ◆ 附 录 B

GSS-API 参考

本附录包括以下几节：

- 第 201 页中的 “GSS-API 函数”提供了 GSS-API 函数表。
- 第 203 页中的 “GSS-API 状态码”讨论 GSS-API 函数返回的状态码，并提供这些状态码的列表。
- 第 207 页中的 “GSS-API 数据类型和值”讨论 GSS-API 使用的各种数据类型。
- 第 210 页中的 “GSS-API 中特定于实现的功能”介绍 Oracle Solaris GSS-API 实现独有的功能。
- 第 212 页中的 “Kerberos v5 状态码”列出了 Kerberos v5 机制可以返回的状态码。

其他 GSS-API 定义可在 `gssapi.h` 文件中找到。

GSS-API 函数

Oracle Solaris 软件实现了 GSS-API 函数。有关每个函数的更多信息，请参见其手册页。另请参见第 203 页中的 “早期 GSS-API 版本中的函数”。

`gss_acquire_cred()`	通过获取预先存在凭证的 GSS-API 凭证句柄来建立全局标识
`gss_add_cred()`	以增量方式构造凭证
`gss_inquire_cred()`	获取有关凭证的信息
`gss_inquire_cred_by_mech()`	获取有关凭证的每机制信息
`gss_release_cred()`	放弃凭证句柄
`gss_init_sec_context()`	使用对等应用程序启动安全上下文
`gss_accept_sec_context()`	接受对等应用程序启动的安全上下文
`gss_delete_sec_context()`	放弃安全上下文
`gss_process_context_token()`	在对等应用程序的安全上下文中处理令牌

`gss_context_time()`	确定上下文处于有效状态的时间
`gss_inquire_context()`	获取有关安全上下文的信息
`gss_wrap_size_limit()`	确定上下文中 `gss_wrap()` 的令牌大小限制
`gss_export_sec_context()`	将安全上下文传输给其他进程
`gss_import_sec_context()`	导入传输的上下文
`gss_get_mic()`	计算消息的加密消息完整性代码 (Message Integrity Code, MIC)
`gss_verify_mic()`	根据消息检查 MIC，以验证收到消息的完整性
`gss_wrap()`	将 MIC 附加到消息中，并有选择地加密消息内容
`gss_unwrap()`	使用附加的 MIC 验证消息。解密消息内容（如果必要）
`gss_import_name()`	将连续的字符串名称转换为内部格式名称
`gss_display_name()`	将内部格式名称转换为文本
`gss_compare_name()`	比较两个内部格式名称
`gss_release_name()`	放弃内部格式名称
`gss_inquire_names_for_mech()`	列出指定机制支持的名称类型
`gss_inquire_mechs_for_name()`	列出支持指定名称类型的机制
`gss_canonicalize_name()`	将内部名称转换为机制名称 (Mechanism Name, MN)
`gss_export_name()`	将 MN 转换为导出格式
`gss_duplicate_name()`	创建内部名称的副本
`gss_add_oid_set_member()`	向组中添加对象标识符
`gss_display_status()`	将 GSS-API 状态码转换为文本
`gss_indicate_mechs()`	确定可用的基础验证机制
`gss_release_buffer()`	放弃缓冲区
`gss_release_oid_set()`	放弃一组对象标识符
`gss_create_empty_oid_set()`	创建不含对象标识符的组
`gss_test_oid_set_member()`	确定对象标识符是否为组的成员

早期 GSS-API 版本中的函数

本节说明了早期 GSS-API 版本中包含的函数。

用于处理 OID 的函数

Oracle Solaris 的 GSS-API 实现提供了下列函数，这些函数满足了便捷性和向后兼容性的需要。但是，其他 GSS-API 实现可能不支持这些函数。

- `gss_delete_oid()`
- `gss_oid_to_str()`
- `gss_str_to_oid()`

尽管可以将机制的名称从字符串转换为 OID，但程序员应尽可能地使用缺省 GSS-API 机制。

重命名的函数

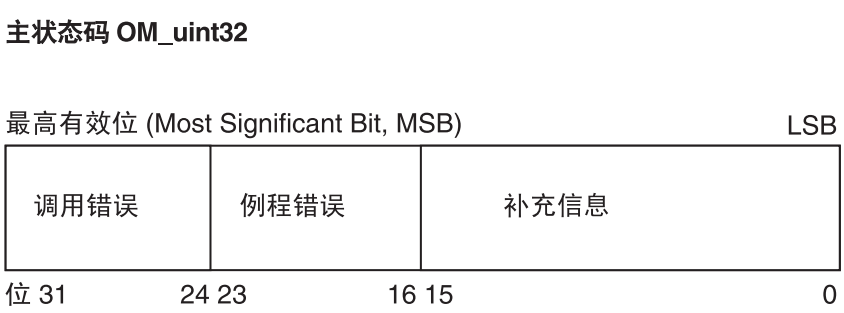
以下函数已被更新的函数所替代。在所有情况下，新函数的功能与较早的函数等效。尽管支持较早的函数，但开发者应尽可能地将这些函数替换为更新的函数。

- `gss_sign()` 已被替换为 `gss_get_mic ()`。
- `gss_verify()` 已被替换为 `gss_verify_mic ()`。
- `gss_seal()` 已被替换为 `gss_wrap ()`。
- `gss_unseal()` 已被替换为 `gss_unwrap ()`。

GSS-API 状态码

主状态码是按下图所示方式在 `OM_uint32` 中进行编码的。

图 B–1 主状态编码



如果 GSS-API 例程返回的 GSS 状态码的高 16 位包含非零值，则调用失败。如果调用错误字段为非零值，则应用程序的例程调用是错误的。表 B–1 中列出了**调用错误**。如果例程错误字段为非零值，则说明该例程因**例程特定错误**（在表 B–2 中列出）而失败。无论高 16 位指示失败还是成功，都可以设置状态码的补充信息字段中的位。表 B–3 中列出了各个位的含义。

GSS-API 主状态码值

下表列出了 GSS-API 返回的调用错误。这些错误特定于特定语言绑定（在本例中为 C）。

表 B–1 GSS-API 调用错误

错误	字段中的值	含义
GSS_S_CALL_INACCESSIBLE_READ	1	不能读取所需的输入参数
GSS_S_CALL_INACCESSIBLE_WRITE	2	不能写入所需的输出参数
GSS_S_CALL_BAD_STRUCTURE	3	参数格式错误

下表列出了 GSS-API 例程错误，即 GSS-API 函数返回的一般错误。

表 B–2 GSS-API 例程错误

错误	字段中的值	含义
GSS_S_BAD_MECH	1	请求不受支持的机制。
GSS_S_BAD_NAME	2	提供无效名称。

表 B–2 GSS-API 例程错误 （续）

错误	字段中的值	含义
GSS_S_BAD_NAMETYPE	3	提供的名称属于不受支持的类型。
GSS_S_BAD_BINDINGS	4	提供不正确的通道绑定。
GSS_S_BAD_STATUS	5	提供无效状态码。
GSS_S_BAD_MIC、GSS_S_BAD_SIG	6	令牌具有无效的 MIC。
GSS_S_NO_CRED	7	凭证不可用、不可访问或不受支持。
GSS_S_NO_CONTEXT	8	未建立上下文。
GSS_S_DEFECTIVE_TOKEN	9	令牌无效。
GSS_S_DEFECTIVE_CREDENTIAL	10	凭证无效。
GSS_S_CREDENTIALS_EXPIRED	11	引用的凭证已到期。
GSS_S_CONTEXT_EXPIRED	12	上下文已到期。
GSS_S_FAILURE	13	各种故障。基础机制检测到未定义特定 GSS–API 状态码的错误。机制特定状态码（即次状态码）提供了有关错误的更多详细信息。
GSS_S_BAD_QOP	14	无法提供请求的保护质量。
GSS_S_UNAUTHORIZED	15	本地安全策略禁止该操作。
GSS_S_UNAVAILABLE	16	操作或选项不可用。
GSS_S_DUPLICATE_ELEMENT	17	请求的凭证元素已存在。
GSS_S_NAME_NOT_MN	18	提供的名称不是机制名称 (Mechanism Name, MN)。

名称 `GSS_S_COMPLETE`（零值）指示缺少 API 错误或补充信息位。

下表列出了 GSS-API 函数返回的补充信息值。

表 B–3 GSS-API 补充信息代码

Code（代码）	位数	含义
GSS_S_CONTINUE_NEEDED	0 (LSB)	仅由 `gss_init_sec_context()` 或 `gss_accept_sec_context ()` 返回。必须再次调用例程才能完成其函数。
GSS_S_DUPLICATE_TOKEN	1	该令牌是早期令牌的副本。
GSS_S_OLD_TOKEN	2	令牌的有效期已到期。

表 B–3 GSS-API 补充信息代码 （续）

Code（代码）	位数	含义
GSS_S_UNSEQ_TOKEN	3	已处理后面的令牌。
GSS_S_GAP_TOKEN	4	未收到预期的每条消息令牌。

有关状态码的更多信息，请参见第 64 页中的 “GSS-API 状态码”。

显示状态码

函数 gss_display_status() 将 GSS-API 状态码转换为文本格式。采用此格式，可以向用户显示代码或将代码置于文本日志中。gss_display_status() 一次仅显示一个状态码，而且某些函数可以返回多个状态条件。因此，应该将 gss_display_status() 作为循环的一部分进行调用。如果 gss_display_status() 指示非零状态码，则函数可以提取其他状态码。

示例 B–1 使用 gss_display_status() 显示状态码

```
OM_uint32 message_context;
OM_uint32 status_code;
OM_uint32 maj_status;
OM_uint32 min_status;
gss_buffer_desc status_string;

...

message_context = 0;

do {

     maj_status = gss_display_status(
               &min_status,
               status_code,
               GSS_C_GSS_CODE,
               GSS_C_NO_OID,
               &message_context,
               &status_string);

     fprintf(stderr, "%.*s\n", \
               (int)status_string.length, \
               (char *)status_string.value);

     gss_release_buffer(&min_status, &status_string,);

} while (message_context != 0);
```

状态码宏

宏 GSS_CALLING_ERROR()、GSS_ROUTINE_ERROR () 和 GSS_SUPPLEMENTARY_INFO() 使用 GSS 状态码。这些宏可以删除相关字段以外的所有信息。例如，可以将 GSS_ROUTINE_ERROR() 应用于状态码，以删除调用错误和补充信息字段。此操作仅保留例程错误字段。可以将这些宏提供的值与相应类型的 GSS_S_*xxx* 符号直接进行比较。如果状态码指示调用错误或例程错误，则宏 GSS_ERROR() 返回非零值，否则将返回零值。由 GSS-API 定义的所有宏一次可以评估多个参数。

GSS-API 数据类型和值

本节介绍各种类型的 GSS-API 数据类型和值。某些数据类型（如 gss_cred_id_t 或 gss_name_t）对用户是不透明的。这些数据类型无需介绍。本节介绍以下主题：

- 第 207 页中的 “基本 GSS-API 数据类型”－给出 OM_uint32、gss_buffer_desc、gss_OID_desc、gss_OID_set_desc_struct 和 gss_channel_bindings_struct 数据类型的定义。
- 第 208 页中的 “名称类型”－给出为指定名称由 GSS-API 识别的各种名称格式。
- 第 209 页中的 “通道绑定的地址类型”－给出可用作 gss_channel_bindings_t 结构的 *initiator_addrtype* 和 *acceptor_addrtype* 字段的各种值。

基本 GSS-API 数据类型

本节介绍 GSS-API 使用的数据类型。

OM_uint32

OM_uint32 是与平台无关的 32 位无符号整数。

gss_buffer_desc

包含 gss_buffer_t 指针的 gss_buffer_desc 的定义采用以下格式：

```
typedef struct gss_buffer_desc_struct {
        size_t length;
        void *value;
} gss_buffer_desc, *gss_buffer_t;
```

gss_OID_desc

包含 gss_OID 指针的 gss_OID_desc 的定义采用以下格式：

```
typedef struct gss_OID_desc_struct {
        OM_uint32 length;
        void*elements;
} gss_OID_desc, *gss_OID;
```

gss_OID_set_desc

包含 gss_OID_set 指针的 gss_OID_set_desc 的定义采用以下格式：

```
typedef struct gss_OID_set_desc_struct  {
        size_t  count;
        gss_OID elements;
} gss_OID_set_desc, *gss_OID_set;
```

gss_channel_bindings_struct

gss_channel_bindings_struct 结构和 gss_channel_bindings_t 指针的定义的格式如下：

```
typedef struct gss_channel_bindings_struct {
        OM_uint32 initiator_addrtype;
        gss_buffer_desc initiator_address;
        OM_uint32 acceptor_addrtype;
        gss_buffer_desc acceptor_address;
        gss_buffer_desc application_data;
} *gss_channel_bindings_t;
```

名称类型

名称类型指示关联名称的格式。有关名称和名称类型的更多信息，请参见第 56 页中的 “GSS-API 中的名称”和第 62 页中的 “GSS-API OID”。GSS-API 支持下表中的 gss_OID 名称类型。

GSS_C_NO_NAME
: 建议将符号名称 GSS_C_NO_NAME 作为参数值，以指示在名称传输中未提供任何值。

GSS_C_NO_OID
: 此值对应于空输入值，而不是实际的对象标识符。如果指定了该值，则该值基于机制特定缺省可列显语法指示关联名称的解释。

GSS_C_NT_ANONYMOUS
: 标识匿名名称的方式。可以比较此值，以与机制无关的方式确定名称是否引用匿名主体。

GSS_C_NT_EXPORT_NAME
: 使用 gss_export_name() 函数导出的名称。

GSS_C_NT_HOSTBASED_SERVICE
: 用于表示与主机关联的服务。此名称格式是按照以下方式使用两个元素（服务和主机名）构造的：*service@hostname*。

GSS_C_NT_MACHINE_UID_NAME
: 用于指示与本地系统中的用户对应的数字用户标识符。该值的解释特定于 OS。`gss_import_name()` 函数将此 UID 解析为用户名，之后该 UID 就表示为用户名形式。

GSS_C_NT_STRING_STRING_UID_NAME
: 用于指示一个数字字符串，该字符串表示本地系统中的用户的数字用户标识符。该值的解释特定于 OS。此名称类型与计算机 UID 格式类似，不同的是缓冲区包含表示用户 ID 的字符串。

GSS_C_NT_USER_NAME
: 本地系统中的命名用户。该值的解释特定于 OS。该值采用以下格式：*username*。

通道绑定的地址类型

下表给出了 `gss_channel_bindings_struct` 结构的 *initiator_addrtype* 和 *acceptor_addrtype* 字段的可能值。这些字段指示名称可以采用的格式，例如 ARPAnet IMP 地址或 AppleTalk 地址。通道绑定将在第 73 页中的 “在 GSS-API 中使用通道绑定”中讨论。

表 B–4 通道绑定地址类型

字段	值（十进制）	地址类型
GSS_C_AF_UNSPEC	0	未指定的地址类型
GSS_C_AF_LOCAL	1	本地主机
GSS_C_AF_INET	2	Internet 地址类型，例如 IP
GSS_C_AF_IMPLINK	3	ARPAnet IMP
GSS_C_AF_PUP	4	pup 协议，例如 BSP
GSS_C_AF_CHAOS	5	MIT CHAOS 协议
GSS_C_AF_NS	6	XEROX NS
GSS_C_AF_NBS	7	nbs
GSS_C_AF_ECMA	8	ECMA
GSS_C_AF_DATAKIT	9	Datakit 协议
GSS_C_AF_CCITT	10	CCITT
GSS_C_AF_SNA	11	IBM SNA
GSS_C_AF_DECnet	12	DECnet
GSS_C_AF_DLI	13	直接数据链接接口
GSS_C_AF_LAT	14	LAT

表 B–4 通道绑定地址类型 （续）

字段	值（十进制）	地址类型
GSS_C_AF_HYLINK	15	NSC 超级通道
GSS_C_AF_APPLETALK	16	AppleTalk
GSS_C_AF_BSC	17	BISYNC
GSS_C_AF_DSS	18	分布式系统服务
GSS_C_AF_OSI	19	OSI TP4
GSS_C_AF_X25	21	X.25
GSS_C_AF_NULLADDR	255	未指定任何地址

GSS-API 中特定于实现的功能

在 API 的实现之间 GSS-API 的某些方面可能有所不同。大多数情况下，实现之间的差异对程序只有很小的影响。在所有情况下，开发者都可以不依赖任何特定于给定实现（包括 Oracle Solaris 实现）的操作来最大化可移植性。

特定于 Oracle Solaris 的函数

Oracle 实现中没有定制的 GSS-API 函数。

人工可读的名称语法

GSS-API 实现在与名称对应的可列显语法中可能有所不同。对于可移植性，应用程序不应该比较使用人工可读（即可列显）格式的名称。相反，这些应用程序应该使用 `gss_compare_name()` 来确定内部格式名称是否与任何其他名称匹配。

Oracle Solaris `gss_display_name()` 实现按以下方式显示名称。如果 *input_name* 参数表示用户主体，则 `gss_display_name()` 将返回 `user_principal@realm` 作为 *output_name_buffer*，返回 `gss_OID` 值作为 *output_name_type*。如果 Kerberos v5 是基础机制，则 `gss_OID` 为 `1.2.840.11354.1.2.2`。

如果 `gss_display_name()` 接收到的名称是 `gss_import_name()` 使用 `GSS_C_NO_OID` 名称类型创建的，则 `gss_display_name()` 将在 *output_name_type* 参数中返回 `GSS_C_NO_OID`。

匿名名称的格式

`gss_display_name()` 函数将输出字符串 "`<anonymous>`"，以指示匿名 GSS-API 主体。与此名称关联的名称类型 OID 为 `GSS_C_NT_ANONYMOUS`。Oracle Solaris 实现支持的其他有效可列显名称不应以尖括号 (<>) 括起。

实现选定数据类型

以下数据类型已作为指针实现，但某些实现可能将这些类型指定为算术类型：gss_cred_t、gss_ctx_id_t 和 gss_name_t。

删除上下文和存储数据

如果上下文建立失败，则 Oracle Solaris 实现不会自动删除部分生成的上下文。因此，应用程序应该通过使用 gss_delete_sec_context() 删除上下文来处理此事件。

此实现将通过内存管理自动释放存储的数据（如内部名称）。但是，当不再需要数据元素时，应用程序仍然应该调用相应的函数（如 gss_release_name()）。

保护通道绑定信息

对通道绑定的支持随机制而变化。Diffie-Hellman 机制与 Kerberos v5 机制都支持通道绑定。

开发者应该假设通道绑定数据没有保密性保护。尽管 Kerberos v5 机制提供此保护，但通道绑定数据的保密性对于 Diffie-Hellman 机制不可用。

上下文导出和进程间令牌

Oracle Solaris 实现检测并拒绝相同上下文的多次尝试导入。

支持的凭证类型

Oracle Solaris 的 GSS-API 实现支持通过 gss_acquire_cred() 获取 GSS_C_INITIATE、GSS_C_ACCEPT 和 GSS_C_BOTH 凭证。

凭证到期

Oracle Solaris 的 GSS-API 实现支持凭证到期。因此，程序员可以在函数（如 gss_acquire_cred() 和 gss_add_cred()）中使用与凭证生命周期相关的参数。

上下文到期

Oracle Solaris 的 GSS-API 实现支持上下文到期。因此，程序员可以在函数（如 gss_init_sec_context() 和 gss_inquire_context()）中使用与上下文生命周期相关的参数。

回绕大小限制和 QOP 值

Oracle Solaris 的 GSS-API 实现（与任何基础机制相反）不对 `gss_wrap()` 处理的消息强加最大大小。应用程序可以使用 `gss_wrap_size_limit()` 确定最大消息大小。

调用 `gss_wrap_size_limit()` 时，Oracle Solaris 的 GSS-API 实现可以检测无效 QOP 值。

使用 *minor_status* 参数

在 Oracle Solaris 的 GSS-API 实现中，函数在 *minor_status* 参数中仅返回特定于机制的信息。其他实现可能在返回的次状态码中包括特定于实现的返回值。

Kerberos v5 状态码

每个 GSS-API 函数都会返回两个状态码：**主状态码**和**次状态码**。主状态码与 GSS-API 的行为相关。例如，如果应用程序尝试在安全上下文到期后传输消息，则 GSS-API 将返回 `GSS_S_CONTEXT_EXPIRED` 的主状态码。第 203 页中的 “GSS-API 状态码”中列出了主状态码。

次状态码是由给定的 GSS-API 实现支持的基础安全机制返回的。每个 GSS-API 函数都采用 *minor_status* 或 *minor_stat* 参数作为第一个变量。无论函数是否成功返回，应用程序都可以检查此参数，以了解基础机制返回的状态。

下表列出了 *minor_status* 参数中的 Kerberos v5 返回的状态消息。有关 GSS-API 状态码的更多信息，请参见第 64 页中的 “GSS-API 状态码”。

Kerberos v5 中状态码 1 的返回消息

下表列出了 Kerberos v5 中状态码 1 的返回的次状态消息。

表 B–5 Kerberos v5 状态码 1

次状态	值	含义
KRB5KDC_ERR_NONE	-1765328384L	无错误
KRB5KDC_ERR_NAME_EXP	-1765328383L	数据库中的客户机项已到期
KRB5KDC_ERR_SERVICE_EXP	-1765328382L	数据库中的服务器项已到期
KRB5KDC_ERR_BAD_PVNO	-1765328381L	请求的协议版本不受支持
KRB5KDC_ERR_C_OLD_MAST_KVNO	-1765328380L	客户机的密钥用旧的主密钥加密

表 B–5 Kerberos v5 状态码 1 （续）

次状态	值	含义
KRB5KDC_ERR_S_OLD_MAST_KVNO	-1765328379L	服务器的密钥用旧的主密钥加密
KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN	-1765328378L	在 Kerberos 数据库中找不到客户机
KRB5KDC_ERR_S_PRINCIPAL_UNKNOWN	-1765328377L	在 Kerberos 数据库中找不到服务器
KRB5KDC_ERR_PRINCIPAL_NOT_UNIQUE	-1765328376L	主体具有 Kerberos 数据库中的多项
KRB5KDC_ERR_NULL_KEY	-1765328375L	客户机或服务器具有空密钥
KRB5KDC_ERR_CANNOT_POSTDATE	-1765328374L	票证的生效期不能延后
KRB5KDC_ERR_NEVER_VALID	-1765328373L	请求的有效生命周期为负数或太短
KRB5KDC_ERR_POLICY	-1765328372L	KDC 策略拒绝请求
KRB5KDC_ERR_BADOPTION	-1765328371L	KDC 无法实现请求的选项
KRB5KDC_ERR_ETYPE_NOSUPP	-1765328370L	KDC 不支持加密类型
KRB5KDC_ERR_SUMTYPE_NOSUPP	-1765328369L	KDC 不支持校验和类型
KRB5KDC_ERR_PADATA_TYPE_NOSUPP	-1765328368L	KDC 不支持 padata 类型
KRB5KDC_ERR_TRTYPE_NOSUPP	-1765328367L	KDC 不支持传输的类型
KRB5KDC_ERR_CLIENT_REVOKED	-1765328366L	已撤销客户机的凭证
KRB5KDC_ERR_SERVICE_REVOKED	-1765328365L	已撤销服务器的凭证

Kerberos v5 中状态码 2 的返回消息

下表列出了 Kerberos v5 中状态码 2 的返回的次状态消息。

表 B–6 Kerberos v5 状态码 2

次状态	值	含义
KRB5KDC_ERR_TGT_REVOKED	-1765328364L	已撤销 TGT
KRB5KDC_ERR_CLIENT_NOTYET	-1765328363L	客户机尚无效，请稍后重试

表 B–6 Kerberos v5 状态码 2 （续）

次状态	值	含义
KRB5KDC_ERR_SERVICE_NOTYET	-1765328362L	服务器尚无效，请稍后重试
KRB5KDC_ERR_KEY_EXP	-1765328361L	口令已到期
KRB5KDC_ERR_PREAUTH_FAILED	-1765328360L	预验证失败
KRB5KDC_ERR_PREAUTH_REQUIRED	-1765328359L	需要其他预验证
KRB5KDC_ERR_SERVER_NOMATCH	-1765328358L	请求的服务器和票证不匹配
KRB5PLACEHOLD_27 到 KRB5PLACEHOLD_30	-1765328357L 到 -1765328354L	KRB5 错误码 27 到 30（保留）
KRB5KRB_AP_ERR_BAD_INTEGRITY	-1765328353L	解密完整性检查失败
KRB5KRB_AP_ERR_TKT_EXPIRED	-1765328352L	票证已到期
KRB5KRB_AP_ERR_TKT_NYV	-1765328351L	票证尚无效
KRB5KRB_AP_ERR_REPEAT	-1765328350L	请求为重放
KRB5KRB_AP_ERR_NOT_US	-1765328349L	票证不是供我们使用
KRB5KRB_AP_ERR_BADMATCH	-1765328348L	票证/验证者不匹配
KRB5KRB_AP_ERR_SKEW	-1765328347L	时钟相位差太大
KRB5KRB_AP_ERR_BADADDR	-1765328346L	网络地址不正确
KRB5KRB_AP_ERR_BADVERSION	-1765328345L	协议版本不匹配
KRB5KRB_AP_ERR_MSG_TYPE	-1765328344L	消息类型无效
KRB5KRB_AP_ERR_MODIFIED	-1765328343L	消息流已经过修改
KRB5KRB_AP_ERR_BADORDER	-1765328342L	消息无序
KRB5KRB_AP_ERR_ILL_CR_TKT	-1765328341L	非法的跨领域票证
KRB5KRB_AP_ERR_BADKEYVER	-1765328340L	密钥版本不可用

Kerberos v5 中状态码 3 的返回消息

下表列出了 Kerberos v5 中状态码 3 的返回的次状态消息。

表 B–7 Kerberos v5 状态码 3

次状态	值	含义
KRB5KRB_AP_ERR_NOKEY	-1765328339L	服务密钥不可用
KRB5KRB_AP_ERR_MUT_FAIL	-1765328338L	相互验证失败
KRB5KRB_AP_ERR_BADDIRECTION	-1765328337L	消息方向不正确
KRB5KRB_AP_ERR_METHOD	-1765328336L	需要替换验证方法
KRB5KRB_AP_ERR_BADSEQ	-1765328335L	消息中的序列号不正确
KRB5KRB_AP_ERR_INAPP_CKSUM	-1765328334L	消息中的校验和类型不适合
KRB5PLACEHOLD_51 到 KRB5PLACEHOLD_59	-1765328333L 到 -1765328325L	KRB5 错误码 51 到 59（保留）
KRB5KRB_ERR_GENERIC	-1765328324L	一般性错误
KRB5KRB_ERR_FIELD_TOOLONG	-1765328323L	字段对于此实现太长
KRB5PLACEHOLD_62 到 KRB5PLACEHOLD_127	-1765328322L 到 -1765328257L	KRB5 错误码 62 到 127（保留）
未返回值	-1765328256L	**仅供内部使用**
KRB5_LIBOS_BADLOCKFLAG	-1765328255L	文件锁定模式的标志无效
KRB5_LIBOS_CANTREADPWD	-1765328254L	无法读取口令
KRB5_LIBOS_BADPWDMATCH	-1765328253L	口令不匹配
KRB5_LIBOS_PWDINTR	-1765328252L	口令读取中断
KRB5_PARSE_ILLCHAR	-1765328251L	组件名称存在非法字符
KRB5_PARSE_MALFORMED	-1765328250L	主体的错误格式表示
KRB5_CONFIG_CANTOPEN	-1765328249L	无法打开/找到 Kerberos `/etc/krb5/krb5` 配置文件
KRB5_CONFIG_BADFORMAT	-1765328248L	Kerberos `/etc/krb5/krb5` 配置文件的格式不正确
KRB5_CONFIG_NOTENUFSPACE	-1765328247L	空间不足，无法返回完整信息
KRB5_BADMSGTYPE	-1765328246L	为编码指定了无效的消息类型
KRB5_CC_BADNAME	-1765328245L	凭证高速缓存名称格式错误

Kerberos v5 中状态码 4 的返回消息

下表列出了 Kerberos v5 中状态码 4 的返回的次状态消息。

表 B–8 Kerberos v5 状态码 4

次状态	值	含义
KRB5_CC_UNKNOWN_TYPE	-1765328244L	未知的凭证高速缓存类型
KRB5_CC_NOTFOUND	-1765328243L	未找到匹配的凭证
KRB5_CC_END	-1765328242L	到达凭证高速缓存的结尾
KRB5_NO_TKT_SUPPLIED	-1765328241L	请求未提供票证
KRB5KRB_AP_WRONG_PRINC	-1765328240L	请求中的主体错误
KRB5KRB_AP_ERR_TKT_INVALID	-1765328239L	票证具有无效的标志集
KRB5_PRINC_NOMATCH	-1765328238L	请求的主体和票证不匹配
KRB5_KDCREP_MODIFIED	-1765328237L	KDC 回复与预期情况不匹配
KRB5_KDCREP_SKEW	-1765328236L	KDC 回复中的时钟相位差太大
KRB5_IN_TKT_REALM_MISMATCH	-1765328235L	初始票证请求中的客户机/服务器领域不匹配
KRB5_PROG_ETYPE_NOSUPP	-1765328234L	程序缺少加密类型支持
KRB5_PROG_KEYTYPE_NOSUPP	-1765328233L	程序缺少密钥类型支持
KRB5_WRONG_ETYPE	-1765328232L	消息中未使用请求的加密类型
KRB5_PROG_SUMTYPE_NOSUPP	-1765328231L	程序缺少校验和类型支持
KRB5_REALM_UNKNOWN	-1765328230L	找不到请求领域的 KDC
KRB5_SERVICE_UNKNOWN	-1765328229L	Kerberos 服务未知
KRB5_KDC_UNREACH	-1765328228L	无法访问请求领域的任何 KDC
KRB5_NO_LOCALNAME	-1765328227L	未找到主体名称的本地名称
KRB5_MUTUAL_FAILED	-1765328226L	相互验证失败
KRB5_RC_TYPE_EXISTS	-1765328225L	已注册重放高速缓存类型

表 B–8 Kerberos v5 状态码 4 （续）

次状态	值	含义
KRB5_RC_MALLOC	-1765328224L	无更多内存可供分配（用重放高速缓存代码）
KRB5_RC_TYPE_NOTFOUND	-1765328223L	重放高速缓存类型未知

Kerberos v5 中状态码 5 的返回消息

下表列出了 Kerberos v5 中状态码 5 的返回的次状态消息。

表 B–9 Kerberos v5 状态码 5

次状态	值	含义
KRB5_RC_UNKNOWN	-1765328222L	一般性未知 RC 错误
KRB5_RC_REPLAY	-1765328221L	消息为重放
KRB5_RC_IO	-1765328220L	重放 I/O 操作失败
KRB5_RC_NOIO	-1765328219L	重放高速缓存类型不支持非易失性存储
KRB5_RC_PARSE	-1765328218L	重放高速缓存名称解析和格式错误
KRB5_RC_IO_EOF	-1765328217L	重放高速缓存 I/O 的文件结束
KRB5_RC_IO_MALLOC	-1765328216L	无更多内存可供分配（用重放高速缓存 I/O 代码）
KRB5_RC_IO_PERM	-1765328215L	重放高速缓存代码中权限被拒绝
KRB5_RC_IO_IO	-1765328214L	重放高速缓存 I/O 代码中的 I/O 错误
KRB5_RC_IO_UNKNOWN	-1765328213L	一般性未知 RC/IO 错误
KRB5_RC_IO_SPACE	-1765328212L	存储重放信息的系统空间不足
KRB5_TRANS_CANTOPEN	-1765328211L	无法打开/找到领域转换文件
KRB5_TRANS_BADFORMAT	-1765328210L	领域转换文件的格式不正确

表 B–9 Kerberos v5 状态码 5 （续）

次状态	值	含义
KRB5_LNAME_CANTOPEN	-1765328209L	无法打开或找到 lname 转换数据库
KRB5_LNAME_NOTRANS	-1765328208L	不能对请求的主体进行转换
KRB5_LNAME_BADFORMAT	-1765328207L	转换数据库项的格式不正确
KRB5_CRYPTO_INTERNAL	-1765328206L	密码系统内部错误
KRB5_KT_BADNAME	-1765328205L	密钥表名称格式错误
KRB5_KT_UNKNOWN_TYPE	-1765328204L	未知的密钥表类型
KRB5_KT_NOTFOUND	-1765328203L	未找到密钥表项
KRB5_KT_END	-1765328202L	到达密钥表的结尾
KRB5_KT_NOWRITE	-1765328201L	无法写入指定的密钥表

Kerberos v5 中状态码 6 的返回消息

下表列出了 Kerberos v5 中状态码 6 的返回的次状态消息。

表 B–10 Kerberos v5 状态码 6

次状态	值	含义
KRB5_KT_IOERR	-1765328200L	写入密钥表时出错
KRB5_NO_TKT_IN_RLM	-1765328199L	找不到请求领域的票证
KRB5DES_BAD_KEYPAR	-1765328198L	DES 密钥包含错误的奇偶校验
KRB5DES_WEAK_KEY	-1765328197L	DES 密钥为弱密钥
KRB5_BAD_ENCTYPE	-1765328196L	错误的加密类型
KRB5_BAD_KEYSIZE	-1765328195L	密钥大小与加密类型不兼容
KRB5_BAD_MSIZE	-1765328194L	消息大小与加密类型不兼容
KRB5_CC_TYPE_EXISTS	-1765328193L	已注册凭证高速缓存类型
KRB5_KT_TYPE_EXISTS	-1765328192L	已注册密钥表类型

表 B–10 Kerberos v5 状态码 6 （续）

次状态	值	含义
KRB5_CC_IO	-1765328191L	凭证高速缓存 I/O 操作失败
KRB5_FCC_PERM	-1765328190L	凭证高速缓存文件权限不正确
KRB5_FCC_NOFILE	-1765328189L	未找到凭证高速缓存文件
KRB5_FCC_INTERNAL	-1765328188L	内部文件凭证高速缓存错误
KRB5_CC_WRITE	-1765328187L	写入凭证高速缓存文件时出错
KRB5_CC_NOMEM	-1765328186L	没有更多内存可供分配（用凭证高速缓存代码）
KRB5_CC_FORMAT	-1765328185L	凭证高速缓存中的格式错误
KRB5_INVALID_FLAGS	-1765328184L	无效的 KDC 选项组合，即内部库错误
KRB5_NO_2ND_TKT	-1765328183L	请求缺失的第二个票证
KRB5_NOCREDS_SUPPLIED	-1765328182L	未向库例程提供凭证
KRB5_SENDAUTH_BADAUTHVERS	-1765328181L	发送了错误的 sendauth 版本
KRB5_SENDAUTH_BADAPPLVERS	-1765328180L	sendauth 发送了错误的应用程序版本
KRB5_SENDAUTH_BADRESPONSE	-1765328179L	错误响应（sendauth 交换期间）
KRB5_SENDAUTH_REJECTED	-1765328178L	服务器在执行 sendauth 交换期间拒绝验证

Kerberos v5 中状态码 7 的返回消息

下表列出了 Kerberos v5 中状态码 7 的返回的次状态消息。

表 B–11 Kerberos v5 状态码 7

次状态	值	含义
KRB5_PREAUTH_BAD_TYPE	-1765328177L	预验证类型不受支持

表 B–11 Kerberos v5 状态码 7 （续）

次状态	值	含义
KRB5_PREAUTH_NO_KEY	-1765328176L	未提供所需的预验证密钥
KRB5_PREAUTH_FAILED	-1765328175L	一般性预验证故障
KRB5_RCACHE_BADVNO	-1765328174L	重放高速缓存的格式版本号不受支持
KRB5_CCACHE_BADVNO	-1765328173L	凭证高速缓存格式版本号不受支持
KRB5_KEYTAB_BADVNO	-1765328172L	密钥表格式的版本号不受支持
KRB5_PROG_ATYPE_NOSUPP	-1765328171L	程序缺少地址类型支持
KRB5_RC_REQUIRED	-1765328170L	消息重放检测需要 rcache 参数
KRB5_ERR_BAD_HOSTNAME	-1765328169L	主机名无法规范
KRB5_ERR_HOST_REALM_UNKNOWN	-1765328168L	主机名无法规范
KRB5_SNAME_UNSUPP_NAMETYPE	-1765328167L	未针对名称类型定义转换到服务主体
KRB5KRB_AP_ERR_V4_REPLY	-1765328166L	最初的票证响应似乎为版本 4 错误
KRB5_REALM_CANT_RESOLVE	-1765328165L	无法解析请求领域的 KDC
KRB5_TKT_NOT_FORWARDABLE	-1765328164L	请求票证无法获取可转发票证
KRB5_FWD_BAD_PRINCIPAL	-1765328163L	尝试转发凭证时的错误主体名称
KRB5_GET_IN_TKT_LOOP	-1765328162L	在 krb5_get_in_tkt 中检测到循环
KRB5_CONFIG_NODEFREALM	-1765328161L	配置文件 `/etc/krb5/krb5.conf` 未指定缺省领域
KRB5_SAM_UNSUPPORTED	-1765328160L	obtain_sam_padata 中的 SAM 标志错误
KRB5_KT_NAME_TOOLONG	-1765328159L	密钥表名字太长
KRB5_KT_KVNONOTFOUND	-1765328158L	密钥表中的主体的密钥版本号不正确

表 B–11 Kerberos v5 状态码 7 （续）

次状态	值	含义
KRB5_CONF_NOT_CONFIGURED	-1765328157L	未配置 Kerberos `/etc/krb5/krb5.conf` 配置文件
ERROR_TABLE_BASE_krb5	-1765328384L	缺省值

◆ ◆ ◆ 附 录 C

指定 OID

应尽可能使用 GSS-API 所提供的缺省 QOP 和机制。请参见第 62 页中的 “GSS-API OID”。但是，您可能会出于自己的考虑指定 OID。本附录介绍了如何指定 OID。本章包含以下主题：

包含 OID 值的文件

为方便起见，GSS-API 允许以可读方式显示机制和 QOP。在 Solaris 系统上，有两个文件（`/etc/gss/mech` 和 `/etc/gss/qop`）中包含有关可用机制和可用 QOP 的信息。如果无权访问这些文件，则必须提供来自其他源代码的串文字。针对该机制或 QOP 发布的 Internet 标准应当可以实现此目的。

`/etc/gss/mech` 文件

`/etc/gss/mech` 文件列出了可用的机制。`/etc/gss/mech` 包含数值和字母两种格式的名称。`/etc/gss/mech` 将显示以下格式的信息：

- ASCII 形式的机制名称
- 机制的 OID
- 用于实现此机制所提供的服务的共享库
- 也可以是用于实现服务的内核模块

样例 `/etc/gss/mech` 可能如示例 C–1 所示。

示例 C–1 `/etc/gss/mech` 文件

```
#
# Copyright 2003 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
```

示例 C–1 /etc/gss/mech 文件 （续）

```
#
#ident    "@(#)mech    1.12    03/10/20 SMI"
#
# This file contains the GSS-API based security mechanism names,
# the associated object identifiers (OID) and a shared library that
# implements the services for the mechanisms under GSS-API.
#
# Mechanism Name    Object Identifier    Shared Library    Kernel Module
[Options]
#
kerberos_v5        1.2.840.113554.1.2.2    mech_krb5.so kmech_krb5
spnego            1.3.6.1.5.5.2        mech_spnego.so.1 [msinterop]
diffie_hellman_640_0    1.3.6.4.1.42.2.26.2.4    dh640-0.so.1
diffie_hellman_1024_0    1.3.6.4.1.42.2.26.2.5    dh1024-0.so.1
```

/etc/gss/qop 文件

对于安装的所有机制，/etc/gss/qop 文件存储每个机制所支持的所有 QOP，这些机制均以 ASCII 字符串和对应的 32 位整数两种形式提供。样例 /etc/gss/qop 可能如以下示例所示：

示例 C–2 /etc/gss/qop 文件

```
#
# Copyright (c) 2000, by Sun Microsystems, Inc.
# All rights reserved.
#
#ident  "@(#)qop 1.3     00/11/09 SMI"
#
# This file contains information about the GSS-API based quality of
# protection (QOP), its string name and its value (32-bit integer).
#
# QOP string                    QOP Value       Mechanism Name
#
GSS_KRB5_INTEG_C_QOP_DES_MD5    0               kerberos_v5
GSS_KRB5_CONF_C_QOP_DES         0               kerberos_v5
```

gss_str_to_oid() 函数

为了与早期版本的 GSS-API 向下兼容，此 GSS-API 实现支持函数 gss_str_to_oid()。gss_str_to_oid() 可用于将表示机制或 QOP 的字符串转换为 OID。字符串可以采用数字或字的形式。

注意 – 某些 GSS-API 实现不支持 gss_str_to_oid()、gss_oid_to_str() 和 gss_release_oid()，因此不建议使用显式的非缺省机制和 QOP。

机制字符串可以硬编码到应用程序中，也可以来自用户输入的内容。但是，并非所有的 GSS-API 实现都支持 gss_str_to_oid()，因此应用程序不应当依赖此函数。

表示机制的数字可以采用两种不同的格式。第一种格式 { 1 2 3 4 } 是 GSS-API 规范要求使用的正式格式。第二种格式 1.2.3.4 的使用更为广泛，但不是正式的标准格式。gss_str_to_oid() 应当使用第一种格式的机制数字，因此，在调用 gss_str_to_oid() 之前，必须对采用第二种格式的字符串进行转换。示例 C–3 中显示了 gss_str_to_oid() 的示例。如果机制无效，gss_str_to_oid() 将返回 GSS_S_BAD_MECH。

由于 gss_str_to_oid() 会分配 GSS-API 数据空间，因此在您完成操作时会提供已存在的 gss_release_oid() 函数来删除所分配的 OID。与 gss_str_to_oid() 一样，gss_release_oid() 也不是广泛支持的函数，因此对于希望具有通用可移植性的程序来说，不应依赖此函数。

构造机制 OID

由于无法始终使用 gss_str_to_oid()，因此提供了多种用于查找和选择机制的备用方法。一种方法是手动构造机制 OID，然后将该机制与一组可用的机制进行比较。另一种方法是获取一组可用机制并从其中选择一个机制。

gss_OID 类型的格式如下：

```
typedef struct gss_OID_desc struct {
     OM_uint32 length;
     void           *elements;
} gss_OID_desc, *gss_OID;
```

其中，此结构的 *elements* 字段指向八位字节字符串的第一个字节，该字符串中包含 gss_OID 的常规 BER TLV 编码的值部分的 ASN.1 BER 编码。*length* 字段包含该值中的字节数。例如，对应于 DASS X.509 验证机制的 gss_OID 值包含一个值为 7 的 *length* 字段和一个指向以下八位字节值的 *elements* 字段：53,14,2,207,163,7,5。

构造机制 OID 的一种方法是声明 gss_OID，然后手动初始化表示指定机制的元素。如上所述，*elements* 值的输入可以是从表中获取的硬编码值，也可以由用户输入。此方法比使用 gss_str_to_oid() 稍微麻烦些，但是二者的效果相同。

然后，可以将所构造的 gss_OID 与 gss_indicate_mechs() 或 gss_inquire_mechs_for_name() 函数已返回的一组可用机制进行比较。应用程序可以使用 gss_test_oid_set_member() 函数来检查这组可用机制中是否存在所构造的机制 OID。如果 gss_test_oid_set_member() 没有返回错误，则表明可以将所构造的 OID 用作 GSS-API 事务的机制。

构造预设 OID 的另一种方法是使应用程序使用 gss_indicate_mechs() 或 gss_inquire_mechs_for_name() 来获取可用机制的 gss_OID_set。gss_OID_set 具有以下格式：

```
typedef struct gss_OID_set_desc_struct {
     OM_uint32 length;
     void           *elements;
} gss_OID_set_desc, *gss_OID_set;
```

其中，每个元素都是一个表示相应机制的 gss_OID。应用程序随后会解析每个机制并显示其数值表示形式。用户可以使用所显示的数值来选择机制。然后，应用程序会将该机制设置为 gss_OID_set 的相应成员。应用程序还可以将所需的机制与首选机制的列表进行比较。

createMechOid() 函数

说明此函数的目的是为了保持代码完整。通常，应当使用 GSS_C_NULL_OID 所指定的缺省机制。

示例 C–3 createMechOid() 函数

```
gss_OID createMechOid(const char *mechStr)
{
        gss_buffer_desc mechDesc;
        gss_OID mechOid;
        OM_uint32 minor;

        if (mechStr == NULL)
                return (GSS_C_NULL_OID);

        mechDesc.length = strlen(mechStr);
        mechDesc.value = (void *) mechStr;

        if (gss_str_to_oid(&minor, &mechDesc, &mechOid) !
            = GSS_S_COMPLETE) {
                fprintf(stderr, "Invalid mechanism oid specified <%s>",
                                mechStr);
                return (GSS_C_NULL_OID);
        }

        return (mechOid);
}
```

指定非缺省机制

parse_oid() 可用于将命令行中的安全机制名称转换为兼容的 OID。

示例 C–4 parse_oid() 函数

```
static void parse_oid(char *mechanism, gss_OID *oid)
{
    char        *mechstr = 0, *cp;
    gss_buffer_desc tok;
    OM_uint32 maj_stat, min_stat;
```

示例 C–4 parse_oid() 函数 （续）

```
    if (isdigit(mechanism[0])) {
        mechstr = malloc(strlen(mechanism)+5);
        if (!mechstr) {
            printf("Couldn't allocate mechanism scratch!\n");
            return;
        }
        sprintf(mechstr, "{ %s }", mechanism);
        for (cp = mechstr; *cp; cp++)
            if (*cp == '.')
                *cp = ' ';
        tok.value = mechstr;
    } else
        tok.value = mechanism;
    tok.length = strlen(tok.value);
    maj_stat = gss_str_to_oid(&min_stat, &tok, oid);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("str_to_oid", maj_stat, min_stat);
        return;
    }
    if (mechstr)
        free(mechstr);
}
```

◆ ◆ ◆ 附 录 D

SASL 示例的源代码

本附录包含第 129 页中的 “SASL 示例”中所介绍示例的源代码。本附录包含以下主题：

- 第 229 页中的 “SASL 客户机示例”
- 第 237 页中的 “SASL 服务器示例”
- 第 245 页中的 “通用代码”

SASL 客户机示例

以下是第 129 页中的 “SASL 示例”中所介绍样例客户机的代码列表。

```
#pragma ident    "@(#)client.c    1.4    03/04/07 SMI"
/* $Id: client.c,v 1.3 2002/09/03 15:11:59 rjs3 Exp $ */
/*
 * Copyright (c) 2001 Carnegie Mellon University.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The name "Carnegie Mellon University" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For permission or any other legal
 *    details, please contact
 *      Office of Technology Transfer
 *      Carnegie Mellon University
 *      5000 Forbes Avenue
 *      Pittsburgh, PA  15213-3890
 *      (412) 268-4387, fax: (412) 268-7395
```

```
 *      tech-transfer@andrew.cmu.edu
 *
 * 4. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by Computing Services
 *     at Carnegie Mellon University (http://www.cmu.edu/computing/)."
 *
 * CARNEGIE MELLON UNIVERSITY DISCLAIMS ALL WARRANTIES WITH REGARD TO
 * THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS, IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE
 * FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN
 * AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
 * OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

#include <config.h>

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <ctype.h>
#include <errno.h>
#include <string.h>

#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#ifdef _SUN_SDK_
#include <sysexits.h>
#endif /* _SUN_SDK_ */

#include <assert.h>

#include <sasl.h>

#include "common.h"

/* remove \r\n at end of the line */
static void chop(char *s)
{
    char *p;

    assert(s);
    p = s + strlen(s) - 1;
    if (p[0] == '\n') {
    *p-- = '\0';
    }
    if (p >= s && p[0] == '\r') {
    *p-- = '\0';
    }
}

static int getrealm(void *context __attribute__((unused)),
```

```
            int id,
            const char **availrealms,
            const char **result)
{
    static char buf[1024];

    /* Double-check the ID */
    if (id != SASL_CB_GETREALM) return SASL_BADPARAM;
    if (!result) return SASL_BADPARAM;

    printf("please choose a realm (available:");
    while (*availrealms) {
    printf(" %s", *availrealms);
    availrealms++;
    }
    printf("): ");

    fgets(buf, sizeof buf, stdin);
    chop(buf);
    *result = buf;

    return SASL_OK;
}

static int simple(void *context __attribute__((unused)),
          int id,
          const char **result,
          unsigned *len)
{
    static char buf[1024];

    /* Double-check the connection */
    if (! result)
    return SASL_BADPARAM;

    switch (id) {
    case SASL_CB_USER:
    printf("please enter an authorization id: ");
    break;
    case SASL_CB_AUTHNAME:
    printf("please enter an authentication id: ");
    break;
    default:
    return SASL_BADPARAM;
    }

    fgets(buf, sizeof buf, stdin);
    chop(buf);
    *result = buf;
    if (len) *len = strlen(buf);

    return SASL_OK;
}

#ifndef HAVE_GETPASSPHRASE
static char *
getpassphrase(const char *prompt)
{
  return getpass(prompt);
```

```
}
#endif /* ! HAVE_GETPASSPHRASE */

static int
getsecret(sasl_conn_t *conn,
      void *context __attribute__((unused)),
      int id,
      sasl_secret_t **psecret)
{
    char *password;
    size_t len;
    static sasl_secret_t *x;

    /* paranoia check */
    if (! conn || ! psecret || id != SASL_CB_PASS)
    return SASL_BADPARAM;

    password = getpassphrase("Password: ");
    if (! password)
    return SASL_FAIL;

    len = strlen(password);

    x = (sasl_secret_t *) realloc(x, sizeof(sasl_secret_t) + len);

    if (!x) {
    memset(password, 0, len);
    return SASL_NOMEM;
    }

    x->len = len;
#ifdef _SUN_SDK_
    strcpy((char *)x->data, password);
#else
    strcpy(x->data, password);
#endif /* _SUN_SDK_ */
    memset(password, 0, len);

    *psecret = x;
    return SASL_OK;
}

static int getpath(void * context __attribute__((unused)),
    const char **path)
{
    *path = getenv("SASL_PATH");

    if (*path == NULL)
    *path = PLUGINDIR;

    return SASL_OK;
}

/* callbacks we support */
static sasl_callback_t callbacks[] = {
  {
    SASL_CB_GETREALM, &getrealm, NULL
  }, {
    SASL_CB_USER, &simple, NULL
```

```
  }, {
    SASL_CB_AUTHNAME, &simple, NULL
  }, {
    SASL_CB_PASS, &getsecret, NULL
  }, {
    SASL_CB_GETPATH, &getpath, NULL
  }, {
    SASL_CB_LIST_END, NULL, NULL
  }
};

int getconn(const char *host, const char *port)
{
    struct addrinfo hints, *ai, *r;
    int err, sock = -1;

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = PF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    if ((err = getaddrinfo(host, port, &hints, &ai)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(err));
    exit(EX_UNAVAILABLE);
    }

    for (r = ai; r; r = r->ai_next) {
    sock = socket(r->ai_family, r->ai_socktype, r->ai_protocol);
    if (sock < 0)
        continue;
    if (connect(sock, r->ai_addr, r->ai_addrlen) >= 0)
        break;
    close(sock);
    sock = -1;
    }

    freeaddrinfo(ai);
    if (sock < 0) {
    perror("connect");
    exit(EX_UNAVAILABLE);
    }

    return sock;
}

char *mech;

int mysasl_negotiate(FILE *in, FILE *out, sasl_conn_t *conn)
{
    char buf[8192];
    const char *data;
    const char *chosenmech;
#ifdef _SUN_SDK_
    unsigned len;
#else
    int len;
#endif /* _SUN_SDK_ */
    int r, c;

    /* get the capability list */
```

```
dprintf(0, "receiving capability list... ");
len = recv_string(in, buf, sizeof buf);
dprintf(0, "%s\n", buf);

if (mech) {
/* make sure that 'mech' appears in 'buf' */
if (!strstr(buf, mech)) {
    printf("server doesn't offer mandatory mech '%s'\n", mech);
    return -1;
}
} else {
mech = buf;
}

r = sasl_client_start(conn, mech, NULL, &data, &len, &chosenmech);
if (r != SASL_OK && r != SASL_CONTINUE) {
saslerr(r, "starting SASL negotiation");
printf("\n%s\n", sasl_errdetail(conn));
return -1;
}

dprintf(1, "using mechanism %s\n", chosenmech);

/* we send up to 3 strings;
   the mechanism chosen, the presence of initial response,
   and optionally the initial response */
send_string(out, chosenmech, strlen(chosenmech));
if(data) {
send_string(out, "Y", 1);
send_string(out, data, len);
} else {
send_string(out, "N", 1);
}

for (;;) {
dprintf(2, "waiting for server reply...\n");

c = fgetc(in);
switch (c) {
case 'O':
    goto done_ok;

case 'N':
    goto done_no;

case 'C': /* continue authentication */
    break;

default:
    printf("bad protocol from server (%c %x)\n", c, c);
    return -1;
}
len = recv_string(in, buf, sizeof buf);

r = sasl_client_step(conn, buf, len, NULL, &data, &len);
if (r != SASL_OK && r != SASL_CONTINUE) {
    saslerr(r, "performing SASL negotiation");
    printf("\n%s\n", sasl_errdetail(conn));
    return -1;
```

```
    }

    if (data) {
        dprintf(2, "sending response length %d...\n", len);
        send_string(out, data, len);
    } else {
        dprintf(2, "sending null response...\n");
        send_string(out, "", 0);
    }
    }

 done_ok:
    printf("successful authentication\n");
    return 0;

 done_no:
    printf("authentication failed\n");
    return -1;
}

#ifdef _SUN_SDK_
void usage(const char *s)
#else
void usage(void)
#endif /* _SUN_SDK_ */
{
#ifdef _SUN_SDK_
    fprintf(stderr, "usage: %s [-p port] [-s service] [-m mech] host\n", s);
#else
    fprintf(stderr, "usage: client [-p port] [-s service] \
        [-m mech] host\n");
#endif /* _SUN_SDK_ */
    exit(EX_USAGE);
}

int main(int argc, char *argv[])
{
    int c;
    char *host = "localhost";
    char *port = "12345";
    char localaddr[NI_MAXHOST + NI_MAXSERV],
    remoteaddr[NI_MAXHOST + NI_MAXSERV];
    char *service = "rcmd";
    char hbuf[NI_MAXHOST], pbuf[NI_MAXSERV];
    int r;
    sasl_conn_t *conn;
    FILE *in, *out;
    int fd;
    int salen;
    struct sockaddr_storage local_ip, remote_ip;

    while ((c = getopt(argc, argv, "p:s:m:")) != EOF) {
    switch(c) {
    case 'p':
        port = optarg;
        break;

    case 's':
        service = optarg;
```

```
        break;

    case 'm':
        mech = optarg;
        break;

    default:
#ifdef _SUN_SDK_
        usage(argv[0]);
#else
        usage();
#endif /* _SUN_SDK_ */
        break;
    }
    }

    if (optind > argc - 1) {
#ifdef _SUN_SDK_
    usage(argv[0]);
#else
    usage();
#endif /* _SUN_SDK_ */
    }
    if (optind == argc - 1) {
    host = argv[optind];
    }

    /* initialize the sasl library */
    r = sasl_client_init(callbacks);
    if (r != SASL_OK) saslfail(r, "initializing libsasl");

    /* connect to remote server */
    fd = getconn(host, port);

    /* set ip addresses */
    salen = sizeof(local_ip);
    if (getsockname(fd, (struct sockaddr *)&local_ip, &salen) < 0) {
    perror("getsockname");
    }

    getnameinfo((struct sockaddr *)&local_ip, salen,
        hbuf, sizeof(hbuf), pbuf, sizeof(pbuf),
#ifdef _SUN_SDK_    /* SOLARIS doesn't support NI_WITHSCOPEID */
        NI_NUMERICHOST | NI_NUMERICSERV);
#else
        NI_NUMERICHOST | NI_WITHSCOPEID | NI_NUMERICSERV);
#endif
    snprintf(localaddr, sizeof(localaddr), "%s;%s", hbuf, pbuf);

    salen = sizeof(remote_ip);
    if (getpeername(fd, (struct sockaddr *)&remote_ip, &salen) < 0) {
    perror("getpeername");
    }

    getnameinfo((struct sockaddr *)&remote_ip, salen,
        hbuf, sizeof(hbuf), pbuf, sizeof(pbuf),
#ifdef _SUN_SDK_    /* SOLARIS doesn't support NI_WITHSCOPEID */
        NI_NUMERICHOST | NI_NUMERICSERV);
#else
```

```
        NI_NUMERICHOST | NI_WITHSCOPEID | NI_NUMERICSERV);
#endif
    snprintf(remoteaddr, sizeof(remoteaddr), "%s;%s", hbuf, pbuf);

    /* client new connection */
    r = sasl_client_new(service, host, localaddr, remoteaddr, NULL,
        0, &conn);
    if (r != SASL_OK) saslfail(r, "allocating connection state");

    /* set external properties here
       sasl_setprop(conn, SASL_SSF_EXTERNAL, &extprops); */

    /* set required security properties here
       sasl_setprop(conn, SASL_SEC_PROPS, &secprops); */

    in = fdopen(fd, "r");
    out = fdopen(fd, "w");

    r = mysasl_negotiate(in, out, conn);
    if (r == SASL_OK) {
    /* send/receive data */


    }

    printf("closing connection\n");
    fclose(in);
    fclose(out);
    close(fd);
    sasl_dispose(&conn);

    sasl_done();

    return 0;
}
```

SASL服务器示例

以下是第 129 页中的 “SASL 示例”中所介绍样例服务器的代码列表。

```
#pragma ident    "@(#)server.c    1.3    03/04/07 SMI"
/* $Id: server.c,v 1.4 2002/10/07 05:04:05 rjs3 Exp $ */
/*
 * Copyright (c) 2001 Carnegie Mellon University.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
```

```
 *    distribution.
 *
 * 3. The name "Carnegie Mellon University" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For permission or any other legal
 *    details, please contact
 *      Office of Technology Transfer
 *      Carnegie Mellon University
 *      5000 Forbes Avenue
 *      Pittsburgh, PA  15213-3890
 *      (412) 268-4387, fax: (412) 268-7395
 *      tech-transfer@andrew.cmu.edu
 *
 * 4. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by Computing Services
 *     at Carnegie Mellon University (http://www.cmu.edu/computing/)."
 *
 * CARNEGIE MELLON UNIVERSITY DISCLAIMS ALL WARRANTIES WITH REGARD TO
 * THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS, IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE
 * FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN
 * AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
 * OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

#include <config.h>

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <ctype.h>
#include <errno.h>
#include <string.h>

#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#ifdef _SUN_SDK_
#include <sysexits.h>
#endif /* _SUN_SDK_ */

#include <sasl.h>

#include "common.h"

#if !defined(IPV6_BINDV6ONLY) && defined(IN6P_IPV6_V6ONLY)
#define IPV6_BINDV6ONLY IN6P_BINDV6ONLY
#endif
#if !defined(IPV6_V6ONLY) && defined(IPV6_BINDV6ONLY)
#define    IPV6_V6ONLY    IPV6_BINDV6ONLY
#endif
#ifndef IPV6_BINDV6ONLY
```

```
#undef      IPV6_V6ONLY
#endif

static int getpath(void * context __attribute__((unused)),
    const char **path)
{
    *path = getenv("SASL_PATH");

    if (*path == NULL)
    *path = PLUGINDIR;

    return SASL_OK;
}

/* callbacks we support */
static sasl_callback_t callbacks[] = {
  {
    SASL_CB_GETPATH, &getpath, NULL
  }, {
    SASL_CB_LIST_END, NULL, NULL
  }
};

/* create a socket listening on port 'port' */
/* if af is PF_UNSPEC more than one socket might be returned */
/* the returned list is dynamically allocated, so caller needs to free it */
int *listensock(const char *port, const int af)
{
    struct addrinfo hints, *ai, *r;
    int err, maxs, *sock, *socks;
    const int on = 1;

    memset(&hints, 0, sizeof(hints));
    hints.ai_flags = AI_PASSIVE;
    hints.ai_family = af;
    hints.ai_socktype = SOCK_STREAM;
    err = getaddrinfo(NULL, port, &hints, &ai);
    if (err) {
    fprintf(stderr, "%s\n", gai_strerror(err));
    exit(EX_USAGE);
    }

    /* Count max number of sockets we can open */
    for (maxs = 0, r = ai; r; r = r->ai_next, maxs++)
    ;
    socks = malloc((maxs + 1) * sizeof(int));
    if (!socks) {
    fprintf(stderr, "couldn't allocate memory for sockets\n");
    freeaddrinfo(ai);
    exit(EX_OSERR);
    }

    socks[0] = 0;    /* num of sockets counter at start of array */
    sock = socks + 1;
    for (r = ai; r; r = r->ai_next) {
    fprintf(stderr, "trying %d, %d, %d\n",r->ai_family, r->ai_socktype,
     r->ai_protocol);
    *sock = socket(r->ai_family, r->ai_socktype, r->ai_protocol);
    if (*sock < 0) {
```

```
            perror("socket");
            continue;
        }
        if (setsockopt(*sock, SOL_SOCKET, SO_REUSEADDR,
                   (void *) &on, sizeof(on)) < 0) {
            perror("setsockopt(SO_REUSEADDR)");
            close(*sock);
            continue;
        }
#if defined(IPV6_V6ONLY) && !(defined(__FreeBSD__) && __FreeBSD__ < 3)
        if (r->ai_family == AF_INET6) {
            if (setsockopt(*sock, IPPROTO_IPV6, IPV6_BINDV6ONLY,
                   (void *) &on, sizeof(on)) < 0) {
            perror("setsockopt (IPV6_BINDV6ONLY)");
            close(*sock);
            continue;
            }
        }
#endif
        if (bind(*sock, r->ai_addr, r->ai_addrlen) < 0) {
            perror("bind");
            close(*sock);
            continue;
         }

         if (listen(*sock, 5) < 0) {
             perror("listen");
             close(*sock);
             continue;
         }

         socks[0]++;
         sock++;
        }

        freeaddrinfo(ai);

        if (socks[0] == 0) {
         fprintf(stderr, "Couldn't bind to any socket\n");
         free(socks);
        exit(EX_OSERR);
        }

        return socks;
}

#ifdef _SUN_SDK_
void usage(const char *s)
#else
void usage(void)
#endif /* _SUN_SDK_ */
{
#ifdef _SUN_SDK_
        fprintf(stderr, "usage: %s [-p port] [-s service] [-m mech]\n", s);
#else
        fprintf(stderr, "usage: server [-p port] [-s service] [-m mech]\n");
#endif /* _SUN_SDK_ */
        exit(EX_USAGE);
}
```

```
/* Globals are used here, but local variables are preferred */
char *mech;

/* do the sasl negotiation; return -1 if it fails */
int mysasl_negotiate(FILE *in, FILE *out, sasl_conn_t *conn)
{
    char buf[8192];
    char chosenmech[128];
    const char *data;
#ifdef _SUN_SDK_
    unsigned len;
#else
    int len;
#endif /* _SUN_SDK_ */
    int r = SASL_FAIL;
    const char *userid;

    /* generate the capability list */
    if (mech) {
    dprintf(2, "forcing use of mechanism %s\n", mech);
    data = strdup(mech);
    } else {
    int count;

    dprintf(1, "generating client mechanism list... ");
    r = sasl_listmech(conn, NULL, NULL, " ", NULL,
              &data, &len, &count);
    if (r != SASL_OK) saslfail(r, "generating mechanism list");
    dprintf(1, "%d mechanisms\n", count);
    }

    /* send capability list to client */
    send_string(out, data, len);

    dprintf(1, "waiting for client mechanism...\n");
    len = recv_string(in, chosenmech, sizeof chosenmech);
    if (len <= 0) {
    printf("client didn't choose mechanism\n");
    fputc('N', out); /* send NO to client */
    fflush(out);
    return -1;
    }

    if (mech && strcasecmp(mech, chosenmech)) {
    printf("client didn't choose mandatory mechanism\n");
    fputc('N', out); /* send NO to client */
    fflush(out);
    return -1;
    }

    len = recv_string(in, buf, sizeof(buf));
    if(len != 1) {
    saslerr(r, "didn't receive first-send parameter correctly");
    fputc('N', out);
    fflush(out);
    return -1;
    }
```

```
if(buf[0] == 'Y') {
    /* receive initial response (if any) */
    len = recv_string(in, buf, sizeof(buf));

    /* start libsasl negotiation */
    r = sasl_server_start(conn, chosenmech, buf, len,
                &data, &len);
} else {
r = sasl_server_start(conn, chosenmech, NULL, 0,
                &data, &len);
}

if (r != SASL_OK && r != SASL_CONTINUE) {
saslerr(r, "starting SASL negotiation");
fputc('N', out); /* send NO to client */
fflush(out);
return -1;
}

while (r == SASL_CONTINUE) {
if (data) {
    dprintf(2, "sending response length %d...\n", len);
    fputc('C', out); /* send CONTINUE to client */
    send_string(out, data, len);
} else {
    dprintf(2, "sending null response...\n");
    fputc('C', out); /* send CONTINUE to client */
    send_string(out, "", 0);
}

dprintf(1, "waiting for client reply...\n");
len = recv_string(in, buf, sizeof buf);
if (len < 0) {
    printf("client disconnected\n");
    return -1;
}

r = sasl_server_step(conn, buf, len, &data, &len);
if (r != SASL_OK && r != SASL_CONTINUE) {
    saslerr(r, "performing SASL negotiation");
    fputc('N', out); /* send NO to client */
    fflush(out);
    return -1;
}
}

if (r != SASL_OK) {
saslerr(r, "incorrect authentication");
fputc('N', out); /* send NO to client */
fflush(out);
return -1;
}

fputc('O', out); /* send OK to client */
fflush(out);
dprintf(1, "negotiation complete\n");

r = sasl_getprop(conn, SASL_USERNAME, (const void **) &userid);
printf("successful authentication '%s'\n", userid);
```

```
    return 0;
}

int main(int argc, char *argv[])
{
    int c;
    char *port = "12345";
    char *service = "rcmd";
    int *l, maxfd=0;
    int r, i;
    sasl_conn_t *conn;

    while ((c = getopt(argc, argv, "p:s:m:")) != EOF) {
    switch(c) {
    case 'p':
        port = optarg;
        break;

    case 's':
        service = optarg;
        break;

    case 'm':
        mech = optarg;
        break;

    default:
#ifdef _SUN_SDK_
        usage(argv[0]);
#else
        usage();
#endif /* _SUN_SDK_ */
        break;
    }
    }

    /* initialize the sasl library */
    r = sasl_server_init(callbacks, "sample");
    if (r != SASL_OK) saslfail(r, "initializing libsasl");

    /* get a listening socket */
    if ((l = listensock(port, PF_UNSPEC)) == NULL) {
    saslfail(SASL_FAIL, "allocating listensock");
    }

    for (i = 1; i <= l[0]; i++) {
       if (l[i] > maxfd)
           maxfd = l[i];
    }

    for (;;) {
    char localaddr[NI_MAXHOST | NI_MAXSERV],
         remoteaddr[NI_MAXHOST | NI_MAXSERV];
    char myhostname[1024+1];
    char hbuf[NI_MAXHOST], pbuf[NI_MAXSERV];
    struct sockaddr_storage local_ip, remote_ip;
    int salen;
    int nfds, fd = -1;
```

```
    FILE *in, *out;
    fd_set readfds;

    FD_ZERO(&readfds);
    for (i = 1; i <= l[0]; i++)
        FD_SET(l[i], &readfds);

    nfds = select(maxfd + 1, &readfds, 0, 0, 0);
    if (nfds <= 0) {
        if (nfds < 0 && errno != EINTR)
        perror("select");
        continue;
    }

       for (i = 1; i <= l[0]; i++)
           if (FD_ISSET(l[i], &readfds)) {
               fd = accept(l[i], NULL, NULL);
               break;
           }

    if (fd < 0) {
        if (errno != EINTR)
        perror("accept");
        continue;
    }

    printf("accepted new connection\n");

    /* set ip addresses */
    salen = sizeof(local_ip);
    if (getsockname(fd, (struct sockaddr *)&local_ip, &salen) < 0) {
        perror("getsockname");
    }
    getnameinfo((struct sockaddr *)&local_ip, salen,
            hbuf, sizeof(hbuf), pbuf, sizeof(pbuf),
#ifdef _SUN_SDK_    /* SOLARIS doesn't support NI_WITHSCOPEID */
            NI_NUMERICHOST | NI_NUMERICSERV);
#else
            NI_NUMERICHOST | NI_WITHSCOPEID | NI_NUMERICSERV);
#endif
    snprintf(localaddr, sizeof(localaddr), "%s;%s", hbuf, pbuf);

    salen = sizeof(remote_ip);
    if (getpeername(fd, (struct sockaddr *)&remote_ip, &salen) < 0) {
        perror("getpeername");
    }

    getnameinfo((struct sockaddr *)&remote_ip, salen,
            hbuf, sizeof(hbuf), pbuf, sizeof(pbuf),
#ifdef _SUN_SDK_    /* SOLARIS doesn't support NI_WITHSCOPEID */
            NI_NUMERICHOST | NI_NUMERICSERV);
#else
            NI_NUMERICHOST | NI_WITHSCOPEID | NI_NUMERICSERV);
#endif
    snprintf(remoteaddr, sizeof(remoteaddr), "%s;%s", hbuf, pbuf);

    r = gethostname(myhostname, sizeof(myhostname)-1);
    if(r == -1) saslfail(r, "getting hostname");
```

```
    r = sasl_server_new(service, myhostname, NULL, localaddr, remoteaddr,
                NULL, 0, &conn);
    if (r != SASL_OK) saslfail(r, "allocating connection state");

    /* set external properties here
       sasl_setprop(conn, SASL_SSF_EXTERNAL, &extprops); */

    /* set required security properties here
       sasl_setprop(conn, SASL_SEC_PROPS, &secprops); */

    in = fdopen(fd, "r");
    out = fdopen(fd, "w");

    r = mysasl_negotiate(in, out, conn);
    if (r == SASL_OK) {
        /* send/receive data */


    }

    printf("closing connection\n");
    fclose(in);
    fclose(out);
    close(fd);
    sasl_dispose(&conn);
    }

    sasl_done();
}
```

通用代码

以下代码样例包含各种 SASL 函数的列表。

```
#pragma ident    "@(#)common.c    1.1    03/03/28 SMI"
/* $Id: common.c,v 1.3 2002/09/03 15:11:59 rjs3 Exp $ */
/*
 * Copyright (c) 2001 Carnegie Mellon University.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The name "Carnegie Mellon University" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For permission or any other legal
 *    details, please contact
```

```
 *      Office of Technology Transfer
 *      Carnegie Mellon University
 *      5000 Forbes Avenue
 *      Pittsburgh, PA  15213-3890
 *      (412) 268-4387, fax: (412) 268-7395
 *      tech-transfer@andrew.cmu.edu
 *
 * 4. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by Computing Services
 *     at Carnegie Mellon University (http://www.cmu.edu/computing/)."
 *
 * CARNEGIE MELLON UNIVERSITY DISCLAIMS ALL WARRANTIES WITH REGARD TO
 * THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS, IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE
 * FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN
 * AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
 * OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

#include <config.h>

#include <stdio.h>
#include <ctype.h>
#include <stdarg.h>
#ifdef _SUN_SDK_
#include <sysexits.h>
#endif /* _SUN_SDK_ */

#include <sasl.h>

/* send/recv library for IMAP4 style literals.

   really not important; just one way of doing length coded strings */

int send_string(FILE *f, const char *s, int l)
{
    int al;

    al = fprintf(f, "{%d}\r\n", l);
    fwrite(s, 1, l, f);
    fflush(f);

    printf("send: {%d}\n", l);
    while (l--) {
    if (isprint((unsigned char) *s)) {
        printf("%c", *s);
    } else {
        printf("[%X]", (unsigned char) *s);
    }
    s++;
    }
    printf("\n");

    return al;
}

int recv_string(FILE *f, char *buf, int buflen)
```

```
{
    int c;
    int len, l;
    char *s;

    c = fgetc(f);
    if (c != '{') return -1;

    /* read length */
    len = 0;
    c = fgetc(f);
    while (isdigit(c)) {
    len = len * 10 + (c - '0');
    c = fgetc(f);
    }
    if (c != '}') return -1;
    c = fgetc(f);
    if (c != '\r') return -1;
    c = fgetc(f);
    if (c != '\n') return -1;

    /* read string */
    if (buflen <= len) {
    fread(buf, buflen - 1, 1, f);
    buf[buflen - 1] = '\0';
    /* discard oversized string */
    len -= buflen - 1;
    while (len--) (void)fgetc(f);

    len = buflen - 1;
    } else {
    fread(buf, len, 1, f);
    buf[len] = '\0';
    }

    l = len;
    s = buf;
    printf("recv: {%d}\n", len);
    while (l--) {
    if (isprint((unsigned char) *s)) {
        printf("%c", *s);
    } else {
        printf("[%X]", (unsigned char) *s);
    }
    s++;
    }
    printf("\n");

    return len;
}

int debuglevel = 0;

int dprintf(int lvl, const char *fmt, ...)
{
    va_list ap;
    int ret = 0;

    if (debuglevel >= lvl) {
```

```
    va_start(ap, fmt);
    ret = vfprintf(stdout, fmt, ap);
    va_end(ap);
    }

    return ret;
}

void saslerr(int why, const char *what)
{
  fprintf(stderr, "%s: %s", what, sasl_errstring(why, NULL, NULL));
}

void saslfail(int why, const char *what)
{
    saslerr(why, what);
    exit(EX_TEMPFAIL);
}
```

◆ ◆ ◆ 附录 E

SASL 参考表

本附录提供了 SASL（Simple Authentication and Security Layer，简单验证和安全层）的参考信息。

SASL 接口摘要

下表提供了一些 SASL 接口的简要说明。

表 E–1 通用于客户机和服务器的 SASL 函数

功能	说明
sasl_version	获取 SASL 库的版本信息。
sasl_done	释放所有的 SASL 全局状态。
sasl_dispose	完成连接以后，处置 `sasl_conn_t`。
sasl_getprop	获取属性，例如用户名、安全层信息。
sasl_setprop	设置 SASL 属性。
sasl_errdetail	根据上次出现的连接错误生成字符串。
sasl_errstring	将 SASL 错误代码转换为字符串。
sasl_encode	使用安全层对要发送的数据进行编码。
sasl_encodev	对通过安全层传输的数据块进行编码。使用 `iovec *` 作为输入参数。
sasl_listmech	创建可用机制的列表。
sasl_global_listmech	返回所有可能机制的数组。请注意，此接口已过时。
sasl_seterror	设置将由 `sasl_errdetail()` 返回的错误字符串。

表 E–1 通用于客户机和服务器的 SASL 函数 （续）

功能	说明
sasl_idle	配置 saslib 以便在空闲期间或网络往返期间执行计算。
sasl_decode	对使用安全层接收的数据进行解码。

表 E–2 仅限于客户机的基本 SASL 函数

功能	说明
sasl_client_init	最初调用一次，以装入和初始化客户机插件。
sasl_client_new	初始化客户机连接。设置 sasl_conn_t 上下文。
sasl_client_start	选择连接机制。
sasl_client_step	执行一个验证步骤。

表 E–3 基本的 SASL 服务器函数（客户机可选的）

功能	说明
sasl_server_init	最初调用一次，以装入和初始化服务器插件。
sasl_server_new	初始化服务器连接。设置 sasl_conn_t 上下文。
sasl_server_start	开始验证交换。
sasl_server_step	执行一个验证交换步骤。
sasl_checkpass	检查纯文本口令短语。
sasl_checkapop	检查 APOP 质询/响应。使用类似于 CRAM-MD5 机制的伪 APOP 机制（可选）。请注意，此接口已过时。
sasl_user_exists	检查用户是否存在。
sasl_setpass	更改口令。（可选）添加用户项。
sasl_auxprop_request	请求辅助属性。
sasl_auxprop_getctx	获取连接的辅助属性上下文。

表 E–4 用于配置基本服务的 SASL 函数

功能	说明
sasl_set_alloc	指定内存分配函数。请注意，此接口已过时。
sasl_set_mutex	指定互斥函数。请注意，此接口已过时。
sasl_client_add_plugin	添加客户机插件。

表 E–4 用于配置基本服务的 SASL 函数 （续）

功能	说明
sasl_server_add_plugin	添加服务器插件。
sasl_canonuser_add_plugin	添加用户标准化插件。
sasl_auxprop_add_plugin	添加辅助属性插件。

表 E–5 SASL 实用程序函数

功能	说明
sasl_decode64	使用 base64 解码。
sasl_encode64	使用 base64 编码。
sasl_utf8verify	验证字符串是否为有效的 UTF-8。
sasl_erasebuffer	删除与安全性相关的缓冲区或口令。实现可能会使用阻止恢复的删除逻辑。

表 E–6 SASL 属性函数

功能	说明
`prop_clear()`	清除值和（可选）来自属性上下文的请求
`prop_dispose()`	处置属性上下文
`prop_dup()`	创建新的 `propctx`，以复制现有 `propctx` 的内容
`prop_erase()`	删除属性的值
`prop_format()`	将请求的属性名称格式化为字符串
`prop_get()`	从上下文中返回 `propval` 结构的数组
`prop_getnames()`	填充 `struct propval` 的数组（在给定属性名称列表时）
`prop_new()`	创建属性上下文
`prop_request()`	向请求中添加属性名称
`prop_set()`	向上下文中添加属性值
`prop_setvals()`	为属性设置值
`sasl_auxprop_getctx()`	获取连接的辅助属性上下文
`sasl_auxprop_request()`	请求辅助属性

表 E–7 回调数据类型

回调	说明
sasl_getopt_t	获取选项值。客户机和服务器均可使用。
sasl_log_t	记录消息处理程序。客户机和服务器均可使用。
sasl_getpath_t	获取用于搜索机制的路径。客户机和服务器均可使用。
sasl_verifyfile_t	验证由 SASL 使用的文件。客户机和服务器均可使用。
sasl_canon_user_t	用户名标准化函数。客户机和服务器均可使用。
sasl_getsimple_t	获取用户和语言列表。仅由客户机使用。
sasl_getsecret_t	获取验证机密。仅由客户机使用。
sasl_chalprompt_t	显示质询和响应提示。仅由客户机使用。
sasl_getrealm_t	获取验证领域。仅由客户机使用。
sasl_authorize_t	授权策略回调。仅由服务器使用。
sasl_server_userdb_checkpass_t	验证纯文本口令。仅由服务器使用。
sasl_server_userdb_setpass_t	设置纯文本口令。仅由服务器使用。

表 E–8 SASL 头文件

头文件	注释
`sasl/saslplug.h`	
`sasl/sasl.h`	开发插件时需要此文件
`sasl/saslutil.h`	
`sasl/prop.h`	

表 E–9 SASL 返回码：常规

返回码	说明
SASL_BADMAC	完整性检查失败
SASL_BADVERS	机制版本之间不匹配
SASL_BADPARAM	提供的参数无效
SASL_BADPROT	错误的协议，取消操作
SASL_BUFOVER	缓冲区溢出
SASL_CONTINUE	验证中需要其他步骤

表 E–9 SASL 返回码：常规 （续）

返回码	说明
SASL_FAIL	一般性失败
SASL_NOMECH	不支持机制
SASL_NOMEM	内存不足，无法完成操作
SASL_NOTDONE	无法请求信息，直到以后交换时才能实现
SASL_NOTINIT	SASL 库未初始化
SASL_OK	成功的结果
SASL_TRYAGAIN	瞬态失败，例如弱密钥

表 E–10 SASL 返回码：仅限客户机

功能	说明
SASL_BADSERV	服务器无法执行相互验证步骤
SASL_INTERACT	需要用户交互
SASL_WRONGMECH	机制不支持请求的功能

表 E–11 SASL 返回码：仅限服务器

功能	说明
SASL_BADAUTH	验证失败
SASL_BADVERS	版本与插件不匹配
SASL_DISABLED	帐户已被禁用
SASL_ENCRYPT	使用机制时需要加密
SASL_EXPIRED	口令短语已到期，需要重置
SASL_NOAUTHZ	授权失败
SASL_NOUSER	找不到用户
SASL_NOVERIFY	用户存在，但没有检验器
SASL_TOOWEAK	对于此用户而言，机制太弱
SASL_TRANS	一次性使用纯文本口令即可为用户启用请求的机制
SASL_UNAVAIL	远程验证服务器不可用

表 E–12 SASL 返回码－口令操作

功能	说明
SASL_NOCHANGE	请求的更改不需要
SASL_NOUSERPASS	不允许用户提供的口令
SASL_PWLOCK	口令短语已锁定
SASL_WEAKPASS	对于安全策略而言，口令短语太弱

◆ ◆ ◆ 附 录 F

打包和签署加密提供者

本附录介绍如何对 Solaris 加密提供者应用程序和模块进行打包。本章包含以下主题：

对加密提供者应用程序和模块打包

在 Solaris 操作系统中，应用程序软件是在称为**软件包**的单元中提供的。软件包是分发和安装软件产品所需的文件集合。软件包通常是在完成应用程序代码开发后由应用程序开发者设计和生成的。有关软件应用程序打包的一般信息，请参见《应用程序包开发者指南》。

加密提供者打包有两个附加要求：

- 开发者必须提供输入文件，这些文件用于将应用程序添加到管理加密框架的配置文件中。
- 开发者必须提供 X.509 证书，以表明符合美国政府的出口法。为了进行测试，可以在获得美国政府批准之前先生成证书。软件包必须包含许可证和已签名提供者才能发运。

符合美国政府出口法

美国政府限制出口开放式加密接口（也称为**有漏洞加密**）。由于此限制，所有的提供者供应商都必须获得美国政府的出口批准。供应商需要向 Oracle Corporation 申请证书，以表明符合出口法。随后，供应商即可通过电子方式签署提供者并提供包含证书的软件。

在出口审批过程中，加密的强度决定可以使用软件的国家/地区。美国政府为美国制造的加密产品定义了两种出口类别：

- **零售加密产品** – 允许将零售加密产品发往所有国家/地区（被认为会带来安全威胁的指定国家/地区除外）。
- **非零售加密产品** – 规定非零售加密产品仅限国内使用，或供美国政府已经豁免的国家/地区使用。

如果您的提供者已获非零售批准，即说明该提供者符合零售批准条件。通过禁止某些调用程序（如 IPsec）使用提供者，可以获得零售批准。在这种情况下，Oracle 为受限使用和无限制使用提供了两种不同的证书。可以在证书申请过程（第 257 页中的 “申请提供者签署证书”）中指明这种情况。此外，还必须生成和签署特殊的激活文件，并将其与提供者一同提供。请参见第 260 页中的 “针对零售出口生成激活文件”。

对用户级提供者应用程序打包

用户级加密提供者应用程序的第三方开发者需要完成以下过程：

1. 从 Oracle Corporation 获取证书。然后在库上签名。请参见第 257 页中的 “向提供者中添加签名”。
2. 随软件包一起提供证书。证书必须置于 `/etc/crypto/certs` 目录中。
3. 将 `pkcs11conf` 类添加到 `pkginfo` 文件的 CLASSES 字符串中。应该添加以下行：

   ```
   CLASS=none pkcs11conf
   ```

4. 在 `etc/crypto` 目录中创建输入文件 `pkcs11.conf`。

   用户级提供者的输入文件名为 `pkcs11.conf`。此文件指定提供者的路径。`pkcs11.conf` 对该项使用以下语法：

   filename

   该项为文件的绝对路径，如 `/opt/lib/$ISA/myProviderApp.so`。运行 `pkgadd` 时会将此文件添加到配置文件中。请注意路径名中的 `$ISA` 表达式。根据需要，`$ISA` 指向应用程序的 32 位版本或 64 位版本。

5. 将以下行添加到软件包的原型文件中：

   ```
   e pkcs11conf etc/crypto/pkcs11conf 0644 root sys
   ```

对内核级提供者模块打包

内核级加密提供者模块的第三方开发者需要完成以下过程：

1. 从 Oracle Corporation 获取证书。然后，在内核软件模块或设备驱动程序上签名。请参见第 257 页中的 “向提供者中添加签名”。
2. 随软件包一起提供证书。证书应该置于 `/etc/crypto/certs` 目录中。
3. 将 `kcfconf` 类添加到 `pkginfo` 文件的 CLASSES 字符串中。应该添加以下行：

   ```
   CLASS=none kcfconf
   ```

4. 在 /etc/crypto 目录中创建输入文件 kcf.conf。此文件用于将软件和硬件插件添加到内核配置文件中。
   - 如果提供者是具有加密机制的内核软件模块，请对该项使用以下语法：

     provider-name:supportedlist=*mech1*,*mech2*,...

     provider-name　内核软件模块的基本名称

     *mech**　列表中的加密机制的名称

     以下项是内核软件模块的示例：

     ```
     des:supportedlist=CKM_DES_CBC,CKM_DES_ECB,CKM_DES_CFB
     ```
   - 如果提供者是加密机制（如加速器卡）的设备驱动程序，则对该项使用以下语法：

     driver_names=*devicedriver1*,*devicedriver2*,...

     *devicedriver**　加密设备的设备驱动程序名称。

     以下项是设备驱动程序的示例：

     ```
     driver_names=dca
     ```

向提供者中添加签名

本节介绍了如何向提供者中添加数字签名，以使提供者可以在框架内工作。本节还介绍如何验证是否已正确签署了提供者。提供者可以为以下对象之一：PKCS #11 库、执行算法的可装入内核模块或硬件加速器的设备驱动程序。

▼ 申请提供者签署证书

通常，提供者的开发者需要申请证书。不过，作为站点安全策略的一部分，将指派系统管理员处理此申请。

1　使用 elfsign request 命令向 Oracle 申请证书。

该命令在生成证书申请的同时会生成一个私钥。

```
% elfsign request -k private-keyfile -r certificate-request
```

private-keyfile　指向私钥位置的路径。随后，系统管理员针对 Solaris 加密框架签署提供者时需要此密钥。该目录应该是安全的。请使用与包含 Oracle 证书的目录不同的目录。

certificate-request　证书申请的路径。

以下示例说明如何将典型申请提交给 Oracle：

```
% elfsign request \
-k /securecrypt/private/MyCompany.private.key \
-r /reqcrypt/MyCompany.certrequest

Enter Company Name / Stock Symbol or some other globally unique identifier.
This will be the prefix of the Certificate DN:MYCORP

The government of the United States of America restricts the export of
"open cryptographic interfaces", also known as "crypto-with-a-hole".
Due to this restriction, all providers for the Solaris cryptographic
framework must be signed, regardless of the country of origin.

The terms "retail" and "non-retail" refer to export classifications
for products manufactured in the USA.  These terms define the portion of the
world where the product may be shipped.  Roughly speaking, "retail" is
worldwide (minus certain excluded nations) and "non-retail" is domestic
only (plus some highly favored nations).  If your provider is subject to
USA export control, then you must obtain an export approval (classification)
from the government of the USA before exporting your provider.  It is
critical that you specify the obtained (or expected, when used during
development) classification to the following questions so that your provider
will be appropriately signed.

Do you have retail export approval for use without restrictions based
on the caller (for example, IPsec)? [Yes/No] N

If you have non-retail export approval for unrestricted use of your
provider by callers, are you also planning to receive retail approval
restricting which export sensitive callers (for example, IPsec) may
use your provider? [Y/N] Y
```

私钥置于指定的文件名（例如 `/etc/crypto/private/MyCompany.private.key` 文件）中。证书申请也置于指定的文件名（例如 `/reqcrypt/MyCompany.certrequest` 文件）中。

2 **将证书申请提交给 Oracle。**

将证书申请发送到以下电子邮件地址：`solaris-crypto-req_ww@oracle.com`

Oracle 将根据证书申请文件生成证书。然后将证书的副本发送回来。

3 **将从 Oracle 收到的证书存储在 `/etc/crypto/certs` 目录中。**

为了安全起见，应该将私钥和证书申请存储在另外的目录中。

▼ 签署提供者

通常，提供者的开发者需要签署提供者。不过，作为站点安全策略的一部分，将指派系统管理员签署此开发者的二进制文件。

- **签署提供者。使用 `elfsign sign` 命令、Oracle 颁发的证书，以及用于从 Oracle 申请证书的私钥。**

  `% elfsign sign -k` *private-keyfile* `-c` *Sun-certificate* `-e` *provider-object*

`-k`	包含用于生成证书申请（已发送给 Oracle）的私钥的文件。
`-c`	Oracle 根据证书申请所颁发的证书的路径。
`-e`	指向待签名的、将在 Solaris 加密框架中使用的提供者或二进制文件的路径。

  以下示例说明如何签署提供者。

```
% elfsign sign \
-k /securecrypt/private/MyCompany.private.key \
-c /etc/crypto/certs/MyCompany
-e /path/to/provider.object
```

  请注意，使用 `elfsign sign` 会更改指定位置中的对象。如果需要该对象的未签名版本，则在应用 `elfsign sign` 之前应该将该对象复制到其他位置。

▼ 验证是否已签署提供者

1. **收集 Oracle 颁发的证书和已签名提供者的路径。**

2. **使用 `elfsign verify` 命令验证是否已正确签署提供者。**

   以下示例说明假设证书位于缺省目录 `/etc/crypto/certs/MyCompany` 中时进行的验证。

```
% elfsign verify \
-e /path/to/MyProvider.so.1
elfsign: verification of /path/to/MyProvider.so.1 passed
```

   以下示例说明如何将证书存储在非缺省目录中。

```
% elfsign verify \
-c /path/to/MyCerts  \
-e /path/to/MyProvider.so.1
elfsign: verification of /path/to/MyProvider.so.1 passed
```

   以下示例说明如何验证使用受限制的证书签署的提供者。

```
% elfsign verify \
-e /path/to/MyRestrictedProvider.so.1
elfsign: verification of /path/to/MyRestrictedProvider.so.1 passed, \
but restricted.
```

▼ 针对零售出口生成激活文件

在针对国内使用和受限制的国际使用提供同一提供者时，此过程非常有用。可以使用对所有客户都有使用限制的证书密钥来签署提供者。对于使用提供者时不受基于调用程序限制的那些客户而言，可以生成和包括允许与 IPsec 一同使用的特殊激活文件。激活文件应该与提供者位于相同的目录中。激活文件的命名约定是将驱动程序名称与扩展名 .esa 组合起来，例如 /kernel/drv/vca.esa。

- **签署提供者。使用 elfsign sign 命令、Oracle 颁发的证书，以及用于从 Oracle 申请证书的私钥。**

  ```
  % elfsign sign -a -k private-keyfile -c Sun-certificate -e provider-object
  ```

-a	生成已签署的 ELF 签名激活 (.esa) 文件。当加密提供者需要非零售出口批准和零售批准时，可以使用此选项。通过限制与出口有关的调用程序（如 IPsec），可以获得零售批准。此选项假设以前已使用受限制的证书签署了提供者二进制文件。
-k	包含用于生成证书申请（已发送给 Oracle Corporation）的私钥的文件。
-c	Oracle 根据证书申请所颁发的证书的路径。
-e	指向待签名的、将在 Solaris 加密框架中使用的提供者或二进制文件的路径。

  以下示例说明如何签署提供者。

  ```
  % elfsign sign \
  -a \
  -k /securecrypt/private/MyCompany.private.key \
  -c /etc/crypto/certs/MyCompany
  -e /path/to/provider.object
  ```

词汇表

Access Control List, ACL（访问控制列表） 包含具有特定访问权限的主体列表的文件。通常，服务器通过查看访问控制列表来验证客户机是否有权使用其服务。请注意，如果没有 ACL 的允许，则仍会拒绝 GSS-API 所验证的主体使用服务。

authentication（验证） 用于检验所声明的主体身份的安全服务。

authorization（授权） 用于确定以下情况的过程：主体是否可以使用服务、允许主体访问哪些对象以及允许针对每个对象执行的访问类型。

client（客户机） 狭义上讲，是指代表用户使用网络服务的进程，例如使用 `rlogin` 的应用程序。在某些情况下，服务器本身即可是其他某个服务器或服务的客户机。非正式地讲，是指使用服务的主体。

confidentiality（保密性） 用于对数据进行加密的安全服务。保密性还包括完整性和验证服务。另请参见 authentication（验证）、integrity（完整性）和 service（服务）。

consumer（使用者） 使用系统服务的应用程序、库或内核模块。

context-level token（上下文级别的令牌） 请参见 token（令牌）。

context（上下文） 两个应用程序之间的信任状态。在两个对等应用程序之间成功建立了上下文之后，上下文接收器可以识别上下文启动者即是所声明的主体，并且可以对发送到上下文接收器的消息进行验证和解密。如果上下文中包括相互验证，则启动器便可知道接收器的标识是否有效，并且还可以对来自接收器的消息进行验证和解密。

credential cache（凭证高速缓存） 包含通过指定机制存储的凭证的存储空间（通常是文件）。

credential（凭证） 用于标识主体和主体身份的信息包。凭证用于指定主体以及主体通常具有的特权。凭证通过安全机制生成。

data replay（数据重放） 消息流中的一条消息被多次接收到。许多安全机制都支持数据重放检测。必须在建立上下文时请求重放检测（如果可用）。

data type（数据类型） 指定数据段所采用的格式，例如 `int`、`string`、`gss_name_t` 结构或者 `gss_OID_set` 结构。

delegation（委托） 如果基础安全机制允许的话，主体（通常是上下文启动器）可以通过将其凭证**委托**给对等主体（通常是上下文接受器）来将对等主体指定为代理。接受者可以使用所委托的凭证来代表最初的主体发出请求，当主体在一台计算机上使用 `rlogin` 登录到另一台计算机时可能会出现这种情况。

exported name（导出名称） 已通过 `gss_export_name()` 从 GSS-API 内部名称格式转换为 GSS-API 导出名称格式的机制名称。可以使用 `memcmp()` 将导出名称与那些非 GSS-API 字符串格式的名称进行比较。另请参见 mechanism name, MN（机制名称）和 name（名称）。

flavor（特性） 以前，**安全方式**和**验证方式**是等效的术语，都用于表示验证类型（如 AUTH_UNIX、AUTH_DES 和 AUTH_KERB）的方式。RPCSEC_GSS 也是一种安全方式，虽然其除了验证之外还提供完整性和保密性服务。

GSS-API Generic Security Service Application Programming Interface（通用安全服务应用编程接口）。即为各种模块化安全服务提供支持的网络层。GSS-API 可提供安全验证、完整性和保密性服务，并允许应用程序在安全性方面获得最大的可移植性。另请参见 authentication（验证）、confidentiality（保密性）和 integrity（完整性）。

host（主机） 可通过网络访问的计算机。

integrity（完整性） 一种安全服务，除了用于进行用户验证之外，还用于通过加密标记来为所传送的数据提供有效性证明。另请参见 authentication（验证）、confidentiality（保密性）和 message integrity code, MIC（消息完整性代码）。

mechanism name, MN（机制名称） GSS-API 内部格式名称的特殊实例。常规的 GSS-API 内部格式名称中可以包含名称的多个实例，每个实例都采用基础机制格式。但是，机制名称对于特定的机制是唯一的。机制名称通过 `gss_canonicalize_name()` 生成。

mechanism（机制） 指定加密技术以实现数据验证或保密性的软件包。Kerberos v5 和 Diffie-Hellman 公钥即是此类示例。

message integrity code, MIC（消息完整性代码） 附加到所传送数据的加密标记，用于确保数据的有效性。数据的接收者会生成另一个 MIC，并将该 MIC 与已发送的 MIC 进行比较。如果这两个 MIC 相等，则表明消息有效。某些 MIC（如那些通过 `gss_get_mic()` 生成的 MIC）对于应用程序可见，而其他 MIC（如那些通过 `gss_wrap()` 或 `gss_init_sec_context()` 生成的 MIC）则对于应用程序不可见。

message-level token（消息级令牌） 请参见 token（令牌）。

message（消息） 采用 `gss_buffer_t` 对象格式的数据，该对象从基于 GSS-API 的应用程序发送到对等应用程序。发送到远程 `ftp` 服务器的 "`ls`" 便是一个消息示例。

消息中不仅仅包含用户所提供的数据。例如，`gss_wrap()` 可提取未包装的消息并生成要发送的已包装消息。经过包装的消息同时包括原始消息和随附的 MIC。GSS-API 所生成的不包括消息的信息是一个**令牌**。请参见 token（令牌）。

MIC 请参见 message integrity code, MIC（消息完整性代码）。

MN 请参见 mechanism name, MN（机制名称）。

mutual authentication（相互验证） 建立了上下文之后，上下文启动器必须向上下文接收器验证其身份。在某些情况下，启动器可能会请求接收器对其进行验证。如果接受器执行此操作，则可以说启动器和接收器进行**相互验证**。

name type（名称类型） 指定了名称的特定形式。名称类型以 gss_OID 类型存储，用于指明名称所使用的格式。例如，user@machine 的名称类型可以是 GSS_C_NT_HOSTBASED_SERVICE。另请参见 exported name（导出名称）、mechanism name, MN（机制名称）和 name（名称）。

name（名称） 主体的名称，如 user@machine。GSS-API 中的名称通过 gss_name_t 结构进行处理，该结构对于应用程序是不透明的。另请参见 exported name（导出名称）、mechanism name, MN（机制名称）、name type（名称类型）和 principal（主体）。

opaque（不透明） 应用于一段数据，其值或格式通常对于使用它的函数不可见。例如，gss_init_sec_context() 的 *input_token* 参数对于应用程序是不透明的，但是对于 GSS-API 则至关重要。同样，gss_wrap() 的 *input_message* 参数对于 GSS-API 也是不透明的，但是对于进行包装的应用程序至关重要。

out-of-sequence detection（失序检测） 许多安全机制都可以检测消息流中的消息是否未按正确的顺序接收。必须在建立上下文时请求消息检测（如果可用）。

per-message token（每消息令牌） 请参见 token（令牌）。

principal（主体） 参与网络通信并且具有唯一名称的客户机/用户或服务器/服务实例；基于 GSS–API 的事务涉及主体之间的交互。主体名称的示例包括：

- user
- user@machine
- nfs@machine
- 123.45.678.9
- ftp://ftp.company.com

另请参见 name（名称）和 name type（名称类型）。

privacy（保密性） 请参见 confidentiality（保密性）。

provider（提供者） 用于为使用者提供服务的应用程序、库或内核模块。

Quality of Protection, QOP（保护质量） 用于选择与完整性服务或保密性服务结合使用的加密算法的参数。如果与完整性服务结合使用，则 QOP 会指定用于生成消息完整性代码 (Message Integrity Code, MIC) 的算法。如果与保密性服务结合使用，则 QOP 会指定用于对 MIC 和消息均进行加密的算法。

replay detection（重放检测） 许多安全机制都可以检测是否错误重复了消息流中的消息。必须在建立上下文时请求消息重放检测（如果可用）。

security flavor（安全特性） 请参见 flavor（特性）。

security mechanism（安全机制） 请参见 mechanism（机制）。

security service（安全服务） 请参见 service（服务）。

server（服务器） 为网络客户机提供资源的主体。例如，如果使用 rlogin 登录到计算机 boston.eng.acme.com，则该计算机即是提供 rlogin 服务的服务器。

service（服务）

1.（也称为**网络服务**）提供给网络客户机的资源，通常由多台服务器提供。例如，如果使用 `rlogin` 登录到计算机 `boston.eng.acme.com`，则该计算机即是提供 `rlogin` 服务的服务器。

2. **安全服务**可以是完整性服务或保密性服务，提供除验证之外的保护级别。另请参见 authentication（验证）、integrity（完整性）和 confidentiality（保密性）。

token（令牌）

采用 GSS-API `gss_buffer_t` 结构形式的数据包。令牌通过 GSS-API 函数生成，用于传送到对等应用程序。

令牌具有以下两种类型：**上下文级别的令牌**，包含用于建立或管理安全上下文的信息。例如，`gss_init_sec_context()` 可将上下文启动器的凭证句柄、目标计算机的名称、所请求的各种服务的标志以及可能的其他项捆绑到要发送到上下文接收器的令牌中。

消息令牌（又称为**每消息令牌**或**消息级别的令牌**），包含 GSS-API 函数根据发送到对等应用程序的消息所生成的信息。例如，`gss_get_mic()` 可为指定的消息生成一个标识加密标记，并将其存储到随该消息发送到对等应用程序的令牌中。从技术上来说，令牌与消息是不同的，这就是为什么称 `gss_wrap()` 生成的是 *output_message* 而不是 *output_token*。

另请参见 message（消息）。

索引

G

Q

R

S

T

U

X

安

包

保

标

插