第四章

系統安全之保障 -- Kerberos 系統之建置

王達明

國立中央大學 大氣科學系學士

t2300030@sparc10.ncu.edu.tw

劉嘉均

國立中央大學 資訊管理研究所研究生

benjamin@im.mgt.ncu.edu.tw

4、1節前言

美國麻省理工學院 (M.I.T.) 在1983 年與 IBM 及 DEC 公司合作，從事一個爲期八年的計劃，稱爲雅典娜 (Athena) 計劃。 此計劃之目的在於整合學院內各式各樣的電腦資源。

在計劃裡，他們發現在開放式的網路環境上，會有很多網路上的安全問題出現。 例如，(1) 他們沒有辦法去防止學生非法地偷竊網路上傳輸的明文資料(plaintext)；(2) 他們幾乎不可能防止學生去破解超級使用者的密碼 (Superuser password) 及防止學生重新啓動 (reboot) 工作站使之成爲單一使用者模式 (single-user mode)。這些問題，促使雅典娜計劃最終設計出 Kerberos 辨識系統來解決網路安全上的問題。

Kerberos 實際上是一個公信的第三者辨識服務系統 (authentication service)。 它提供証明身份的証書 (Credential) 讓工作站與伺服器 (Server) 作爲相互辨識之用。 Kerberos 主要的功用是保証工作站在要求

服務以前，要先向它索取証書；當工作站取得証書以後，再把証書及服務要求一起送給伺服器，此時伺服器利用証書來辨識使用者的眞正身份， 並決定是否提供服務。此與一般工作站要求服務的差別在於：一般工作站都是直接向伺服器提出服務的要求，而且伺服器並沒有被賦予辨識使用者的能力。

4、2節工作原理

在說明 Kerberos 辨識協定以前，先說明以下符號的定義以方便底下的說明，圖4－1：

c	→	client
s	→	伺服器 (server)
u	→	使用者名稱 (username)
addr	→	client 的網路位址
life	→	票証之有效期 (lifetime)
tgs, TGS	→	票証許可伺服器 (ticket-granting server)
tg	→	票証許可服務 (ticket-granting service)
AS	→	辨識伺服器 (authentication server)
K(x)	→	x 的私人鑰匙 (private key)
K(x,y)	→	供 x 及 y 二者使用的會期鑰匙 (session key)
{abc}K(x)	→	把 abc 用 x 的鑰匙編密 (encrypted)
T(x,y)	→	利用 x 的票証去使用 y 這項服務
A(x)	→	給 x 的辨識子 (authenticator)
time	→	時間標記 (timestamp)

圖4－1：說明 Kerberos 系統時所用到的符號

在 Kerberos 系統裡用以証明身份的証書 (Credentials) 有兩種——票証 (tickets) 及辨識子(authenticator)。兩者都是基於以私人鑰匙 (private key) 來編密的資料，但它們利用不同的鑰匙來對資料加以編密。 在此，票証的功用是在辨識伺服器及工作站二者眞正的身份，以

便安全地傳送使用者的身份，這張票証的內容包括工作站名稱、工作站的 Internet 位址、時間、有效期及隨機產生的會期鑰匙 (random session key)，此票証可以重覆使用直到票証使用時間期滿爲止。

{s, c, addr, time, life, *K(s,c)*} *Ks*

相反的，辨識子 (authenticator) 只可以被使用一次。 當一個工作站需要利用服務時，必須重新再產生一個辨識子，此辨識子是由工作站（或者應該說是在該工作站上的應用程式來得更加的貼切）自已所產生。辨識子的內容包括工作站的名稱、工作站的 IP 位址、及工作站當時的時間。

{c, addr, time} *K(s,c)*

Kerberos 辨識協定可以分成三個階段 (phase)，分述如下：

(1) 索取起始票証 (Getting the Initial Ticket)

(1-1) 在工作站 (Client) 輸入使用者名稱

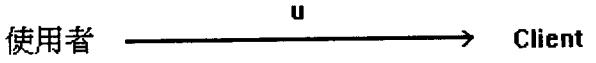


圖４－２ 使用者鍵入使用者名稱（帳號名稱）

使用者在 login 提示符號 (prompt)下鍵入使用者名稱 (圖４－2)。

(1-2) 索取票証許可伺服器 (TGS) 之票証

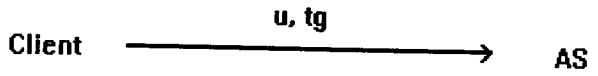
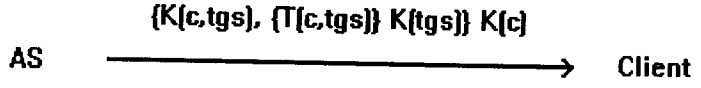


圖４－３ Client 端送出服務要求到辨識伺服器上

當使用者名稱鍵入完畢，Client會送一個要求 (request) 到辨識伺服器(Authentication Server)上 (圖４－3)，其內容包括使用者名稱及一個特別服務，票証許可服務 (ticket-granting service)， 的名稱。

(1-3) 傳回 TGS 之票証



where T[c,tgs] = {c, tgs, time, life, K{c,tgs}}

圖４－４辨識伺服器所傳回來的票証

當要求到達辨識伺服器後，伺服器會核對是否認識提出服務要求的工作站，也就是要辨識這部工作站的身份； 若爲合法的使用者，則伺服器會產生一隨機的會期鑰匙 (random session key)，此鑰匙 K(c,tgs) 是供往後工作站及辨識伺服器兩者間溝通之用。然後，伺服器會產生一票証 T(c,tgs) 給票証許可伺服器，此票証是被另一只有票証許可伺服器 (TGS) 及辨識伺服器 (AS) 才認識的鑰匙 K(tgs) 所編密的。

最後， AS 會把票証、隨機會期鑰匙及一些附加的資訊傳回給 Client (圖４－４)，這些資料都是被只有辨識伺服器及 Client 才認識的 Client 私人鑰匙 K(c) 所編密，此 K(c) 是從使用者密碼所產生的。

(1-4) 輸入使用者密碼 (password)

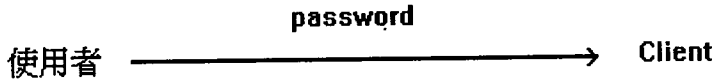


圖４－５ 使用者在工作站前輸入密碼

當 Client 接收到辨識伺服器 (AS) 所傳回的資料時，Client 會要求使用者鍵入使用者密碼 (password) (圖４－５)，此密碼會用來計算出 K(c)，此 K(c) 會做解密處理，把傳回票証內的 K(c,tgs) 及 {T(c,tgs)}K(tgs) 讀出，並保留以作後用(參見下文(2-1)之說明)。此時若使用者輸入的密碼不正確，則會因爲得不到正確的 K(c) 而無法解密得到資料。

(2) 索取服務票証 (Getting Server Tickets)

(2-1) 向票証許可伺服器 (TGS) 索取票証

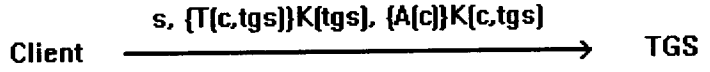


圖４－６ 向票証許可伺服器（TGS）索取其他服務的票証

當 Client 要求某項服務之前，它必須得到該服務的票証。此時 Client 會送出一個信息給票証許可伺服器 TGS (圖４－６)，其信息內容包括提供該服務的伺服器名稱 (s)、票証許可票証 T(c,tgs)、及辨識子 A(c)。在此，辨識子 A(c) 是由 Client 所產生，其內容包括 Client 的名稱、網路 IP 位址、及時間，A(c) 是由辨識伺服器 (AS) 得到的 K(c,tgs)

(參見上文(1-3)及(1-4)) 所編密。此外給 K(tgs) 所編密的 T(c,tgs) 也是從 AS 所得到的。

(2-2) 傳回票証許可票証 *(Ticket-granting ticket)*

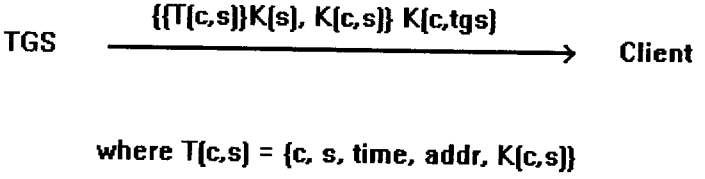


圖４－７ 票証許可伺服器所傳回來的其他服務的票証

當票証許可伺服器 (TGS) 收到 Client 送來的信息後，TGS 會利用私人鑰匙 K(tgs) 對票証 {T(c,tgs)}K(tgs) 做解密，得出包含在 T(c,tgs) 內的會期鑰匙 K(c,tgs)，然後此 K(c,tgs) 會用來對辨識子 {A(c)}K(c,tgs) 做解密，之後 TGS 會查核票証之有效期 (lifetime) 並比較 T(c,tgs) 及 A(c) 內的 Client 名稱及網路位址是否相同。若核對無誤，TGS 將會如上文 (1-3) 步驟一樣產生一個隨機的會期鑰匙 K(c,s)，並利用預先存放在 TGS 的 K(s) (Client要求的服務所持有的私人鑰匙)對票証 T(c,s) 做加密產生 {T(c,s)}K(s)，此時 TGS 將 {T(c,s)}K(s) 及 K(c,s) 一起用 K(c,tgs) 編密送回給 Client (圖４－７)。

(3) 要求一個服務 (Requesting a Service)

(3-1) 向伺服器提出服務要求

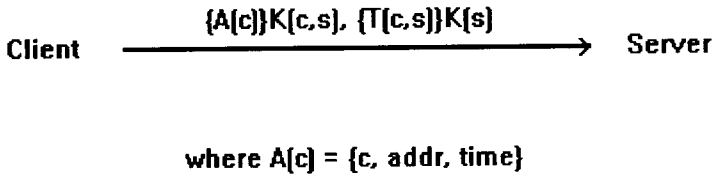


圖４－８ 向所要求的服務伺服器提出服務的要求

Client 在收到票証許可伺服器 (TGS) 傳回的資料後，便會用鑰匙 K(c,tgs) 解密並讀出 {T(c,s)}K(s) 及 K(c,s)。此時以與上文 (2-1) 步驟相同之方式利用 K(c,s) 產生辨識子 {A(c)}K(c,s) ，再將 {A(c)}K(c,s) 與 T(c,s) 一起用 K(s) 編密傳送給伺服器 (Server) (圖４－８)。

當伺服器接收到這些資料後，便會以與上文 (2-2) 步驟相同之方式做使用者辨識工作，依辨識結果決定是否對該使用者提供服務。

4、3節Kerberos系統的安裝

目前，我們所安裝的Kerberos系統爲4.0版，讀者可以在底下的網路節點裡找到所有的原始程式：

athena-dist.mit.edu: \pub\kerberos\dist\921209\ksrc.tar.Z

此已是patchlevel 10的版本，解壓縮後不需用patch指令做修補。但要視乎本身機器環境做適當的修改。請參見以下說明及本章節附錄4A。同時請注意：建議在進入此ftpsite後，用cd \pub\kerberos\dist\921209直接進入此子目錄，否則在進入dist子目錄以後，將無法進入921209子目錄。（請參看\pub\kerberos\README_KRB4)

安裝的程序如下：

在安裝Kerberos之前，必須先進人超級使用者(Superuser)帳號。若在FreeBSD-1.0-release上安裝，請先按照附錄4A把檔案做適當的修改，然後再按下列步驟進行安裝：

（1）首先把ksrc.tar.Z拷貝到一專爲Kerberos原始檔案而設的子目錄下，建議自建一子目錄**/mit/kerberos/src**（下文我們將會利用**[SOURCE_DIR]**來代表此原始檔目錄）並把ksrc.tar.Z拷貝到此目錄下，利用uncompress及tar -xvf把此壓縮檔解開。

（2）再建立一用來放object files的子目錄（下文以**[OBJ_DIR]**代表）**/mit/kerberos/obj**。

（3）改變目錄到**[OBJ_DIR]**。在此目錄下鍵入以下指令，此指令之用處爲在**[OBJ_DIR]**下建立子目錄及安裝 Makefiles：

make -f **[SOURCE_DIR]**/tools/makeconfig SRCDIR=**[SOURCE_DIR]**

（4）改變目錄到util/imake.includes。用文書處理器把在此目錄下的config.Imakefile檔從頭到尾看一遍，把適當flags加入及改變。

特別要注意 SRCTOP 及 SITE_KRB_REALM 此二 flags，SRCTOP應與 **[SOURCE_TOP]** 一致、SITE_KRB_REALM 應改爲你所要用的 Realm Name。（請參考本章節附錄有關 config.Imakefile之修補）。

（5）檢查一下你的機器類型 (machine type) 是否有定義在 include/osconf.h 及其他相關的檔案下。我們這次安裝的機器是 I386 (即PC)，在此檔案內已經有定義了。若您發現您的機器沒有在此檔案內定義，可嘗試自行加入。

（6）檢查一下是否有 /usr/tmp 此子目錄，若沒有請自行建立。

（7）改變目錄到 **[OBJ_DIR]**。以下此指令爲根據 config.Imakefile 內的設定來建立新的 Makefiles 及建立系統：

make world

若你需要把輸出的信息引導到一個檔案並用背景作業，請用：

make world >& WORLDLOG_891201 &

若你修改了一些 Kerberos 的原始程式，而沒有修改過 config.Imakefile 或任何 Imakefiles 或 Makefiles，你可利用：

make all

否則 (例如修改過 config.Imakefile... 時)，則必須利用 make world。

（8）若看見像 verify 指令所輸出的資料(參見下文測試部份)，則表示建立完成。然後請在 **[OBJ_DIR]** 子目錄用以下指令來安裝：

make install

（9）請把在 **[SOURCE_DIR]**/prototypes/ 目錄下之 etc.krb.conf 檔案拷貝到 /etc/ 目錄下，並把檔案改名爲 krb.conf。然後用文書處理器

編輯此 krb.conf 檔案裡的 realm_name 及 master_server_name，此檔案的格式如下：

realm_name
realm_name master_server_name admin server

若你準備要在此機器上跑 administration server ，'admin server'二字必須要在 master_server_name 名字的旁邊。

例如，你的 Realm Name 爲 net.ncu.edu.tw 而 master server 是 athena.net.ncu.edu.tw，那麼 krb.conf 檔案將包括以下文字：

NET.NCU.EDU.TW
NET.NCU.EDU.TW ATHENA.NET.NCU.EDU.TW admin server

注意字母的大小寫要與在 **[OBJ_DIR]**/util/imake.includes/ 目錄下 config.Imakefile 檔案內 SITE_KRB_REALM 的定義一致。(參見本章節附錄 config.Imakefile 的修改)

測試：

（１）利用 verify 指令對DES library工具做測試，請用以下指令：

[OBJ_DIR]/lib/des/verify

若有類似以下資料產生，則建立成功。否則 Kerberos 系統將無法正常工作：

```
Examples per FIPS publication 81, keys ivs and cipher
in hex.  These are the correct answers, see below for
the actual answers.
Examples per Davies and Price.
EXAMPLE ECB     key = 08192a3b4c5d6e7f
        clear = 0
        cipher = 25 dd ac 3e 96 17 64 67
```

```
ACTUAL ECB
    clear ""
    cipher  = (low to high bytes)
        3e ac dd 25 67 64 17 96
EXAMPLE ECB    key = 0123456789abcdef
    clear = "Now is the time for all "
    cipher = 3f a4 0e 8a 98 4d 48 15 ...
ACTUAL ECB
    clear "Now is the time for all "
    cipher  = (low to high bytes)
        ae 7a f9 d6 94 06 e6 a1

EXAMPLE CBC    key = 0123456789abcdef  iv = 1234567890abcdef
    clear = "Now is the time for all "
    cipher =   e5 c7 cd de 87 2b f2 7c
               43 e9 34 00 8c 38 9c 0f
               68 37 88 49 9a 7c 05 f6
ACTUAL CBC
    clear "Now is the time for all "
    ciphertext = (low to high bytes)
        af 76 10 25 5a a3 df 88
        5a 20 69 36 4a dc 7b d5
        e1 1b a6 1a 21 a3 55 87
        00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00
    decrypted clear_text = "Now is the time for all "
EXAMPLE CBC checksum    key =  0123456789abcdef iv =
1234567890abcdef
    clear =         "7654321 Now is the time for "
    checksum       58 d2 e7 7e 86 06 27 33, or some part thereof
ACTUAL CBC checksum
        encrypted cksum = (low to high bytes)
        16 6c 90 6a fd d6 5a 11
```

圖 4—9 verify 程式測試的結果

（２）關於其他部份的執行程式測試，請參考本章４、４節 Kerberos 系統的測試以及操作部份。

4、4節 Kerberos 系統的測試以及操作

這裡假設讀者已經參考了 Kerberos 安裝指引把 Kerberos 系統成功地安裝在系統上。就如同在那份指引裡頭的慣例一樣，我們把 Kerberos 可執行檔所在的目錄稱爲： **[OBJ_DIR]**。在參考這份指引的時候，最好讀者本身就是系統管理者，否則，建議讀者在操作本指引的時候能有一位系統管理者在旁協助。

1.Kerberos 是如何運作的？ 一個整體的概括性描述

這節將爲使用者如何與 Kerberos 系統產生互動 (interaction) 的關係做一個很簡單的描述。這兩者之間的互動是完全“透通的“(transparence)——使用者一點也感覺不到，但是一個 Kerberos 系統的管理者卻會發現：這樣的一個簡單的描述是很有用的。底下的討論將會對 Kerberos 系統的細節做一個總覽；如果要更多的資訊，請參考底下的這篇指引：

An Authentication Service for Open Network Systems。

1.1 網路的服務和他們的委託程式(Client Programs)

在一個提供網路服務的環境裡，我們可以利用委託程式來向伺服程式(Server Programs) 要求做一些服務，而這些伺服程式也是在網路上的某些節點上。假設你想要簽入到一部工作站，然後想利用 rlogin 來遠程簽入到其他的主機上的時候，你便需要利用 rlogin 這支委託程式來連結到所想要登錄的主機上的 rlogin 伺服程式。

1.2 Kerberos 票証 (Kerberos Tickets)

在 Kerberos 環境底下，只要委託程式能夠在向伺服程式提出要求的時候附上一張“票証“(ticket)，則伺服程式便會接受這樣的要求。這張票証的目的在於辨識使用委託程式的使用者的身份。

1.3 Kerberos 的主資料庫 (Master Database)

只要你在 Kerberos 伺服器上的主資料庫有你相關的訊息的時候，Kerberos 便會發給你一張票証。資料庫中記載著你的 Kerberos 使用者名稱 (principal name) 以及你的 Kerberos 密碼。每一個 Kerberos 的使用者在這個主資料庫中都要有一筆記錄。

1.4 領取票証的票証 (Ticket-Granting Ticket)

kinit 這個程式會要求使用者輸入使用者名稱和密碼；如果你的輸入正確的話，你就會得到一張"可以領取票証的票証"。在底下我們會說明的是：一個使用者便是利用這張“可以領取票証的票証“來向 Kerberos 系統要求領取其他服務的票証。

1.5 網路服務和 Kerberos 主資料庫

Kerberos 主資料庫裡也包含著所有需要做到辨識的網路伺服程式的記錄。假設你所在的環境裡有一部稱作 laughter 的主機，它要求 Kerberos 替它辨識每一位想要使用 rlogin 伺服程式遠端簽入 (remote login) 到本系統的每一位使用者，那麼這支伺服程式 rlogind 便需要在 Kerberos 主資料庫中有一筆記錄；這筆記錄記載著這個服務程式的名稱 (principal name) 以及實例名稱(instance)。

實例名稱就是這個伺服程式所在主機的主機名稱；在上述的例子中，這項服務的實例名稱就是 laughter。Kerberos 系統就是利用實例名稱來辨別不同主機上所提供的相同服務。（例如，幾乎所有的主機都有提供 rlogin 這支伺服程式)

1.6 使用者與 Kerberos 系統的互動 (interaction)

假設你是一個普通的使用者，當你走到一部工作站的前面而想要使用它來遠程簽入到 laughter 這部主機上。底下就是整體系統的一連串動作：

1. 首先你要簽入到你面前的系統，並且使用 kinit 程式來獲取一張"領取票証的票証"。kinit 會要求你輸入你的使用者名稱 (username)，以及你的 Kerberos 密碼。

a. kinit 程式會把你的要求送到 Kerberos 的主要伺服主機去。Kerberos 伺服程式便會在主資料庫中尋找這項要求中的使用者名稱。

b. 如果這筆資料存在，Kerberos 伺服程式便會利用你的密碼資料來將"領取票証的票証"做編密的動作，之後再把這編密過後的資料送回到你面前的工作站。如果，kinit 程式能利用你在終端機前所鍵入的密碼來把從 Kerberos 伺服程式送回來的資料成功地解密的話，它便會把這張"領取票証的票証"存成一個檔案，以供你在以後想要使用其他服務時的票証記錄。這個票証檔的名稱可以利用 KRBTKFILE 這個環境變數來指定。如果這個環境變數沒有被指定的話，則這個票証檔案的名稱會是：/tmp/tkt_uid，uid 是你的使用者編號。

2. 現在你使用 rlogin 委託程式來遠程簽入到 laughter 這部主機上。

host% rlogin laughter

a. rlogin 委託程式將會檢查你是否有向 laughter 這部主機要求遠程服務的票証。當然，在這個時候你還沒有這樣的票証，所以，rlogin 委託程式便利用"領取票証的票証"向 Kerberos 的伺服程式要求要領取這項 rlogin 服務的票証。

b. "發放票証"的服務 (Ticket-Granting Service) 在接受到這樣的要求的時候，便會到 Kerberos 的主資料庫中尋找是否有這樣的服務的記錄，也

就是這部主機上是否已經被設定成有提供 rlogin 的服務。如果確實有這樣的記錄存在，“發放票証“的伺服程式便會發給回給你一張 rlogin 的使用票証，而這張票証便保留在你的票証檔中。

c. 此時，你面前這部工作站的 rlogin 委託程式便利用這張票証來向位於 laughter 這部主機上的 rlogin 伺服程式提出要求。如果証實你的票証是正確的話，laughter 上的 rlogin 服務程式便會允許你的遠程簽入的要求。

2. 設立並且測試 Kerberos 伺服程式

設立並且測試 Kerberos 伺服程式的步驟如下：

1. 使用 kdb_init 指令來啓始化 Kerberos 主資料庫。
2. 使用 kdb_edit 指令來把你的使用者名稱加入到 Kerberos 主資料庫中。
3. 啓動 Kerberos 伺服程式。
4. 使用 kinit 指令來索取“可以領取票証的票証。
5. 使用 klist 指令來測試 kinit 程式已經成功地向 Kerberos 伺服程式辨識你的身份，也就是說，在這個時候你應該要有這張“可以領取票証的票証“了。

2.1 啓始化 Kerberos 的主資料庫

簽入到 Kerberos 的伺服主機上，使用 su 指令來使自己成爲超級使用者(Superuser)。如果你在安裝 Kerberos 時候都是採用 安裝程式裡的預設值(default value)的話，那與 Kerberos 操作有關的指令就會放在 /usr/etc/ 這個標準目錄下。從現在開始，我們都稱這個目錄名稱爲：**[ADMIN_DIR]**。

kdb_init 指令會產生一個新的 Kerberos 主資料庫並且會對它做啓始化的動作。它會要求你輸入這部伺服主機的區域名稱 (realm name)以及主資料庫的密碼。千萬不要忘了這個密碼。如果你忘了的話，整個資料庫就變得一點用處都沒有了。使用 kdb_init 的程序如下：

```
host# [ADMIN_DIR]/kdb_init
Realm name (default XXX): [REALMNAME]
                                    <- 鍵入你的系統區域名稱
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter Kerberos master key:          <- 鍵入你的密碼
```

圖 4－10 kdb_init 指令的使用範例

2.2 儲存 Kerberos 主資料庫的密碼

kstash 指令會把 Kerberos 的主資料庫密碼存在 /.k 這個檔案中，以便 Kerberos 伺服程式可以自動的啓動而不需要要求管理者輸入密碼。其他一些相關的 Kerberos 管理指令便可以利用這個檔案中的資料來直接存取主資料庫中的資料而不需要管理者還要手動地輸入這個密碼。如果你在使用這些管理指令的時候想要輸入密碼的話，那就不要使用 kstash 指令。

把這個密碼儲存在磁碟機上的這種做法可能不太能被接受，但是如果不這樣使用的話，那日後每次要啓動 Kerberos 伺服程式的時候便需要有管理者來輸入密碼。這個指令會要求你鍵入兩次的密碼：

```
host# [ADMIN_DIR]/kstash
Enter Kerberos master key:          <- 鍵入你的密碼
Current Kerberos master key version is 1.
Master key entered   BEWARE!
```

圖 4－11 kstash 指令的使用範例

要注意的一點是：如果這個密碼被他人得知的話，整個資料庫中的資料就赤裸裸地展現出來，系統也就沒有了安全性可言。密碼一定要保持安全。

2.3 使用 kdb_edit 指令來加入一位使用者到主資料庫中

kdb_edit 指令可以用來把一位使用者或是伺服程式的資料加到主資料庫中，並且也可以修改資料庫中現有的資料。它會要求你輸入名稱(principal name)以及實例名稱 (instance)。

這裡所指的名稱就是使用者的名稱或是一個伺服程式的名稱。實例名稱更進一步地指明是哪一個名稱。如果名稱所指的是一個伺服程式的名稱，那它相對應的實例名稱就是這個伺服程式所在的主機名稱。如果名稱指的是一般使用者的名稱，那其相對應的實例名稱通常都是設定成空值(NULL)。

底下假設利用 kdb_edit 指令來把 benjamin 這位使用者加入到 Kerberos 的主資料庫中。

```
host# [ADMIN_DIR]/kdb_edit

Opening database...

Enter Kerberos master key: ..... <- 鍵入 Kerberos 主資料庫的密碼
Verifying, please re-enter
Enter Kerberos master key:...   <- 鍵入 Kerberos 主資料庫的密碼
Current Kerberos master key version is 1

Master key entered. BEWARE!
Previous or default values are in [brackets],
enter return to leave the same, or new value.

Principal name: benjamin        <- 鍵入使用者名稱
```

```
Instance:                          <- 鍵入一個空值的實例名稱

<Not found>, Create [y] ? y        <- 按 y 來確定要加入這位使用者

Principal: benjamin  Instance:  m_key_v: 1
New Password:....                  <- 鍵入使用者的密碼
Verifying, please re-enter
New Password:....                  <- 鍵入使用者的密碼
Principal's new key version = 1
Expiration date (enter dd-mm-yy) [15/03/94]?
                                   <- 按 return 鍵來接受預設值
Max ticket lifetime (*5 minutes) [255]?
                                   <- 按 return 鍵來接受預設值

Attributes [0]?                    <- 鍵入 return 來接受預設值
Edit O.K.

Principal name:                    <- 鍵入 return 來離開 kdb_edit
```

圖４－１２ kdb_edit 指令的使用範例

kdb_edit 指令把你自己的資料加到 Kerberos 的主資料庫中。

2.4 啓動 Kerberos 伺服程式

目錄到預設的目錄： /usr/etc， 在背景裡啓動伺服程式：

```
host# ./kerberos&
```

果你曾使用過 kstash 指令來儲存主資料庫的密碼，那伺服程式便會自動地啓動而不需要你輸入密碼。如果你不想使用 kstash 的資料的話，鍵入下列的命令：

host# ./kerberos -m

服程式會要求你輸入密碼後才眞正地啓動運作。

2.5 測試 Kerberos 伺服程式

退出超級使用者的權限並且使用 kinit 指令來得到一張“可以領取票証的票証“，這個指令會爲你產生一個票証檔。如果你在安裝的時候所採用的都是預設值的話，那這些與使用者有關的指令都會放在 /usr/athena 這個目錄底下。底下我們將稱這個目錄爲：**[K_USER]**。

用 kinit 指令：

```
t% [K_USER]/kinit
 Project Athena
beros Initialization
beros name:        <- 輸入你的使用者名稱
sword:             <- 輸入你的密碼
```

圖４－１３ kinit 指令的使用範例

用 klist 指令來列出票証檔中的資料：

```
host% [K_USER]/klist
```

個指令所印出的訊息應該像是：

```
Ticket file:    /tmp/tkt100
Principal:      yourusername@REALMNAME
Issued          Expires          Principal
Mar15 7:20:17  Mar15 15:20:17krbtgt.REALMNAME@REALMNAME
```

圖４－１４ 利用 klist 指令來顯示 Kerberos 票証的內容

如果在這個時候發生了問題，可以參考 /kerberos/kerberos.log 檔，來看看到底是發生了什麼樣的錯誤。

3.設立並且測試管理伺服程式

設立並且測試管理伺服程式的步驟如下：

1. 使用 kdb_edit 指令來把你自己的使用者名稱加入到主資料庫中。記得要附上一個管理者的實例名稱。
2. 在伺服主機上修改存取控制串列(access control lists)。
3. 啓動 Kerberos 管理伺服程式。
4. 使用 kpasswd 指令來修改你的密碼。
5. 使用 kadmin 指令來把新的記錄加到主資料庫中。
6. 使用 kinit 指令來測試 kadmin 指令是否已經成功地把新使用者的資料加到主資料庫中。

3.1 爲管理者加上一個管理者的實例名稱

登錄到 Kerberos 伺服主機上並且使用 su 指令來成爲超級使用者。利用 kdb_edit 指令來加入一個新的使用者到主資料庫中，而這個使用者的實例名稱是：admin。

```
host# [ADMIN_DIR]/kdb_edit

Opening database...

Enter Kerberos master key:      <- 鍵入 Kerberos 主資料庫的密碼
Verifying, please re-enter
Enter Kerberos master key:          <- 鍵入 Kerberos 主資料庫的密
碼

Current Kerberos master key version is 1

Master key entered.  BEWARE!
Previous or difault values are in [brackets],
enter return to leave the same, or new value.

Principal name: benjamin
Instance: admin

<Not found>, Create [y] ?
Principal: benjamin        Instance: admin m_key_v: 1
```

```
New Password:              <-鍵入使用者的密碼
Verifying, please re-enter
New Password:              <-鍵入使用者的密碼
Principal's new key version = 1
Expiration date (enter dd-mm-yy) [15/03/94] ?
                           <- 按 return 來接受預設值
Max ticket lifetime (*5 minutes) [255] ?
                           <-  按 return 來接受預設值
Attributes [0] ?           <-  按 return 來接受預設值
Edit O.K.
Principal name:            <-  按 return 來離開
```

圖４－１５ 利用 kdb_edit 指令來加入一個 kerberos 的管理者帳號

3.2 存取控制串列 (Access Control Lists)

Kerberos 管理伺服程式使用了三個存取控制串列來決定哪些使用者有權要求一些特殊的處理。在 Kerberos 伺服器上，存取控制串列的所在位置和 Kerberos 主資料庫的所在位置是一樣的，也就是：/kerberos。這些存取控制串列是簡單的 ASCII 文字檔，其中，每一行都記錄著某一位使用者具有什麼權限來執行什麼樣的系統功能。爲了使得多位的使用者能夠執行相同的功能，可以把這些使用者的名稱放在同一個檔案中的不同行裡。

第一個串列：/kerberos/admin_acl.mod 記載著有哪些使用者可以有權限來修改 Kerberos 主資料庫中的資料。爲了使得使用者，benjamin，有權限修改區域名稱爲 TIM.EDU 的主資料庫裡的資料，你應該把下列這行資料放到 /kerberos/admin_acl.mod 這個資料檔中。

benjamin.admin@TIM.EDU

第二個串列：/kerberos/admin_acl.get 記載著有哪些使用者可以有權限來讀取主資料庫中的資料。

第三個串列： /kerberos/admin_acl.add 記載著有哪些使用者可以有權限來加入新的使用者到主資料庫中。

3.3 啓動管理伺服程式

更改目錄到管理相關指令目錄的所在，預設值是： /usr/etc，並且在背景中啓動伺服程式：

```
host# ./kadmind -n&
```

如果你曾經使用過 kstash 指令來存放 Kerberos 的密碼，那麼這個管理伺服程式會自動地啓動。如果你不想使用 /.k 中的資料，則鍵入下列的指令：

```
host# ./kadmind
```

伺服程式將會要求你輸入密碼；啓動了伺服程式之後，你應該把它暫停住(suspend)並且把它放到背景中執行。（鍵入 Control-Z 以及 bg 指令）

3.4 測試 kpasswd

爲了測試管理伺服程式，你可以試著利用 kpasswd 來更改你的密碼，並且你也應該試著利用 kadmin 這個指令來加入一個使用者到主資料庫中；預設上，這兩個指令都存放在預設的目錄： /usr/athena。測試之前，請退出超級使用者的權限。

```
host% [K_USER]/kpasswd
Old password for benjamin@TIM.EDU:      <- 鍵入原來的密碼
New password for benjamin@TIM.EDU:      <- 鍵入新的密碼
Verifying, please re-enter New Password for benjamin@TIM.EDU:
                                        <- 鍵入新的密碼
Password changed.
```

圖４－１６ 利用 kpasswd 指令來更改 kerberos 系統中的密碼

一旦你更改了密碼之後，請使用上述的 kinit 指令來驗証你的密碼已經被成功地更改過了。（附帶說明的一點是：由於受到美國對於編密／解密程式模組的限制出境，在我們所抓取回來的原始程式中並沒有這樣的模組，而更改密碼便會牽涉到編密的模組，所以我們並沒有做這方面的測試）

3.5 測試 kadmin

你可以藉由加入一個新的使用者到主資料庫中來測試 kadmin 這個程式：

```
host% [K_USER]/kadmin
Welcome to the Kerberos Administration Program, version 2
Type "help" if you need it.
admin: ank username     "ank" 代表著：add new key
Admin password:         <- 鍵入管理者的密碼
Password for username: <- 鍵入使用者的密碼
Verifying, please re-enter Password for username:
                        <- 鍵入使用者的密碼
username added to database.
admin: quit
Cleaning up and exiting.
```

圖４－１７ 使用 kadmin 指令來加入一個新的使用者

3.6 利用 kinit 來驗証

一旦你已經加入了一個新的使用者，你應該試著使用 kinit 指令來驗証，並且試著能否得到正確的票証：

```
host% [K_USER]/kinit username
MIT Project Athena
Kerberos Initialization for "usrname@TIM.EDU"
Password: <- 鍵入使用者的密碼
host% [K_USER]/klist
Ticket file:    /tmp/tkt_100
Principal:      username@TIM.MIT.EDU
Issued              Expires             Principal
Nov 20 15:58:52     Nov 20 23:58:52     krbtgt.TIM.EDU@TIM.EDU
```

圖４－１８ 利用 kinit 指令來獲取啓始的票証

如果你遇到了任何的問題，你可以檢視 /kerberos/kerberos.log 這個檔案，看看到底是發生了哪一類的錯誤問題。

附錄４Ａ　在FreeBSD-1.0-RELEASE上安裝Kerberos VI *(patchlevel 10)* 的修改

格式說明：

（１）以下檔案行號後之括號中：**added** 表示加入、**marked** 表示該段程式碼被加入註解符號 /* 及 */ (即該段程式碼將不會被編譯器所編譯)、**changed**表該段程式碼以 -> 符號之下的程式碼所取代。

（２）行號以 *斜體* 字型表示之。

（３）原程式碼以 標準 字型表示之。

（４）修改過後的程式碼與原程式碼不同之處則以 **粗體** 字型表之。

由於需要做適當修改的檔案非常的多，爲了讀者的方便起見，我們整理了下列的表格，讓讀者能夠方便地找到需要修改的檔案。 請參考下頁的列表。

檔案的所在目錄	需要修改的檔案	頁數
obj/appl/bsd/	login.c	4-27
	rcp.c	4-29
	rlogin.c	4-30
	rlogind.c	4-32
obj/appl/knetd/	knetd.c	4-33
obj/appl/rkinit/lib/	rk_krb.c	4-34
	rk_rpc.c	4-35
	rk_util.c	4-35
obj/appl/rkinit/rkinit/	rkinit.c	4-36
obj/appl/rkinit/rkinitd/	rpc.c	4-36
obj/appl/sample/	sample_client.c	4-37
obj/appl/tftp	tftp.c	4-37
	upd.c	4-38
obj/include/	conf.h	4-38
obj/kadmin/	admin_server.c	4-39
	get_srvtab.c	4-40
	kadm_funcs.c	4-40
obj/lib/des/	des.c	4-40
	make_p.c	4-41
	new_rnd_key.c	4-41
	quad_cksum.c	4-42
	random_key.c	4-42
obj/lib/kadm/	kadm_cli_wrap.c	4-43
obj/lib/krb/	in_tkt.c	4-44
	log.c	4-44
	util.c	4-44
obj/server/	kerberos.c	4-45
obj/slave/	kprop.c	4-47
	kpropd.c	4-48
obj/util/et/	com_err.c	4-49
	compile_et.c	4-50
	error_table.y	4-50
	internal.h	4-51
obj/util/imake.includes/	config.Imakefile	4-51
obj/util/makedepend/	main.c	4-53
obj/util/ss/	help.c	4-54
	list_rqs.c	4-54
	listen.c	4-55
	mk_cmds.c	4-56

==========

(1) 目錄：obj/appl/bsd/

==========

(1-1) 檔案：login.c

---

line *60* (added):

```
#include <sys/ttydefaults.h>
```

line *61* (added):

```
#include <sys/ioctl_compat.h>
```

line *62* (added):

```
#include <sys/termios.h>
```

line *65* (marked):

```
#if !defined(_AIX)
#include <lastlog.h>
#endif
->
/*
#if !defined(_AIX)
#include <lastlog.h>
#endif
*/
```

line *149* (changed) -- 參考Unix系統目錄/usr/include/下utmp.h檔案UT_HOSTSIZE之定義。

```
#define UT_HOSTSIZE     sizeof(((struct utmp *)0)->ut_host)
->
```

```
#define UT_HOSTSIZE    16
```

line *151* (changed) -- 參考Unix系統目錄/usr/include/下utmp.h檔案對UT_NAMESIZE之定義。

```
#define UT_NAMESIZE    sizeof(((struct utmp *)0)->ut_name)
->
#define UT_NAMESIZE    8
```

line *226* (marked):

```
off_t lseek();
->
/*
off_t lseek();
*/
```

line *231* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
(void)signal(SIGALRM, timedout);
->
(void)signal(SIGALRM, (void (*)())timedout);
```

line *953* (changed) -- signal函數及其第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
oldint = signal(SIGINT, sigint);
->
oldint = (sigtype (*)())signal(SIGINT, (void (*)())sigint);
```

line *957* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
(void)signal(SIGINT, oldint);
->
(void)signal(SIGINT, (void (*)())oldint);
```

line *1236* (changed) -- wait函數第一參數資料形態的定義與系統有別，請用Unix系統指令man wait查看其定義並更改之。

```
 while(wait((union wait *)0) != child)
->
 while(wait((int *)0) != child)
```

(1-2) 檔案：rcp.c

---

line *252* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
 (void) signal(SIGPIPE, lostconn);
->
 (void) signal(SIGPIPE, (void (*)())lostconn);
```

line *574* (changed) -- signal函數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
 istat = signal(SIGINT, SIG_IGN);
 qstat = signal(SIGQUIT, SIG_IGN);
->
 istat = (int (*)())signal(SIGINT, SIG_IGN);
 qstat = (int (*)())signal(SIGQUIT, SIG_IGN);
```

line *580* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
 (void) signal(SIGINT, istat);
 (void) signal(SIGQUIT, qstat);
->
 (void) signal(SIGINT, (void (*)())istat);
 (void) signal(SIGQUIT, (void (*)())qstat);
```

(1-3) 檔案：rlogin.c

---

line *355* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
    (void) signal(SIGPIPE, lostpeer);
  ->
    (void) signal(SIGPIPE, (void (*)())lostpeer);
```

line *577* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
    (void) signal(SIGURG, copytochild);
    (void) signal(SIGUSR1, writeroob);
  ->
    (void) signal(SIGURG, (void (*)())copytochild);
    (void) signal(SIGUSR1, (void (*)())writeroob);
```

line *580* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
    (void) signal(SIGCHLD, catchild);
  ->
    (void) signal(SIGCHLD, (void (*)())catchild);
```

line *594* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
    if (signal(sig, act) == SIG_IGN)
  ->
    if (signal(sig, (void (*)())act) == SIG_IGN)
```

line *639* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
    (void) signal(SIGWINCH, sigwinch);
```


097

->

(void) signal(SIGWINCH, **(void (*)())**sigwinch);

line *654* (changed) -- wait3函數第一參數資料形態的定義與系統有別，請用Unix系統指令man wait查看其定義並更改之。

pid = wait3(&status, WNOHANG|WUNTRACED, (struct rusage *)0);

->

pid = wait3(**(int *)**&status, WNOHANG|WUNTRACED, (struct rusage *)0);

line *808* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

(void) signal(SIGCHLD, catchild);

->

(void) signal(SIGCHLD, **(void (*)())**catchild);

line *863* (changed) -- 參考Unix系統目錄/usr/include/下fcntl.h檔案對FWRITE之定義。

int out = FWRITE;

->

int out = **0x0002**;

line *986* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

(void) signal(SIGURG, oob);

->

(void) signal(SIGURG, **(void (*)())**oob);

(1-4) 檔案：rlogind.c

---

line *212* (changed) -- gethostbyaddr函數第一參數資料形態的定義與系統有別，請用Unix系統指令man gethostbyaddr查看其定義並更改之。

```
    hp = gethostbyaddr(&fromp->sin_addr, sizeof (struct in_addr),
        fromp->sin_family);
->
    hp = gethostbyaddr((char *)&fromp->sin_addr, sizeof (struct in_addr),
        fromp->sin_family);
```

line *262* (changed) -- malloc函數資料形態的定義與系統有別，請用Unix系統指令 man malloc查看其定義並更改之。

```
    line = malloc(16);
->
    line = (char *)malloc(16);
```

line *423* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
    signal(SIGCHLD, cleanup);
->
    signal(SIGCHLD, (void (*)())cleanup);
```

line *292* (changed) -- 同上。

```
    signal(SIGINT, sendsig);
->
    signal(SIGINT, (void (*)())sendsig);
```

line *294* (changed) -- 同上。

```
    signal(SIGQUIT, sendsig);
->
    signal(SIGQUIT, (void (*)())sendsig);
```

line *296* (changed) -- 同上。

```
    signal(SIGTERM, sendsig);
  ->
    signal(SIGTERM, (void (*)())sendsig);
```

==========

(2) 目錄：obj/appl/knetd/

==========

(2-1) 檔案：knetd.c

---

line *140* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
    (void) signal(SIGHUP, configure_it);
    (void) signal(SIGCHLD, reapchild);
    (void) signal(SIGTERM, closedown);
  ->
    (void) signal(SIGHUP, (void (*)())configure_it);
    (void) signal(SIGCHLD, (void (*)())reapchild);
    (void) signal(SIGTERM, (void (*)())closedown);
```

line *157* (changed) -- bind函數第二參數資料形態的定義與系統有別，請用Unix系統指令man bind查看其定義並更改之。

```
    if (bind(tcp_socket, &tcp_saddr, sizeof(tcp_saddr))) {
  ->
    if (bind(tcp_socket, (struct sockaddr *)&tcp_saddr, sizeof(tcp_saddr)))
{
```

line *402* (changed) -- wait函數第一參數資料形態的定義與系統有別，請用Unix系統指令man wait查看其定義並更改之。

```
(void) wait((union wait *)0);
->
(void) wait((int *)0);
```

==========

(3) 目錄： obj/appl/rkinit/lib/

==========

(3-1) 檔案：rk_krb.c

---

line *235* (changed) -- strcpy函數第二參數資料形態的定義與系統有別，請用Unix系統指令man strcpy查看其定義並更改之。

```
strcpy(phost, krb_get_phost(host));
->
strcpy(phost, (const char *)krb_get_phost(host));
```

line *302* (changed) -- signal函數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
old_sigfunc[i] = signal(i,sig_restore);
->
old_sigfunc[i] = (int (*)())signal(i,sig_restore);
```

line *309* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
signal(i,old_sigfunc[i]);
->
signal(i,(void (*)())old_sigfunc[i]);
```

(3-2)檔案：rk_rpc.c

---

line *198* (changed) -- connect函數第二參數資料形態的定義與系統有別，請用Unix系統指令man connect查看其定義並更改之。

```
    if (connect(sock, (char *)&saddr, sizeof (saddr)) < 0) {
->
    if (connect(sock, (struct sockaddr *)&saddr, sizeof (saddr)) < 0) {
```

line *341* (changed) -- getsockname函數第二參數資料形態的定義與系統有別，請用Unix系統指令man getsockname查看其定義並更改之。

```
    if (getsockname(sock, caddrp, &addrlen) < 0) {
->
    if (getsockname(sock, (struct sockaddr *)caddrp, &addrlen) < 0) {
```

(3-3) 檔案：rk_util.c

---

line *200* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
    return((int (*)()) signal(SIGALRM, rki_timeout));
->
    return((int (*)()) signal(SIGALRM, (void (*)())rki_timeout));
```

line *211* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
    (void) signal(SIGALRM, old_alrm);
->
    (void) signal(SIGALRM, (void (*)())old_alrm);
```

==========

(4) 目錄： obj/appl/rkinit/rkinit/

==========

(4-1) 檔案：rkinit.c

---

line *156* (changed) -- strcpy函數第二參數資料形態的定義與系統有別，請用Unix系統指令man strcpy查看其定義並更改之。

```
    strcpy(r_krealm, krb_realmofhost(hp->h_name));
  ->
    strcpy(r_krealm, (const char *)krb_realmofhost(hp->h_name));
```

==========

(5) 目錄： obj/appl/rkinit/rkinitd/

==========

(5-1) 檔案：rpc.c

---

line *87* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
    signal(SIGALRM, timeout);
  ->
    signal(SIGALRM, (void (*)())timeout);
```

==========

(6) 目錄： obj/appl/sample/

==========

(6-1) 檔案：sample_client.c

---

line *116* (changed) -- connect函數第二參數資料形態的定義與系統有別，請用Unix系統指令man connect查看其定義並更改之。

```
    if (connect(sock, &sin, sizeof(sin)) < 0) {
->
    if (connect(sock, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
```

==========

(7) 目錄： obj/appl/tftp/

==========

(7-1) 檔案：tftp.c

---

line *105* (changed) -- 以雙引號取代單引號。

```
    if (!succ && strcmp(argv[2], '-') != 0)
->
    if (!succ && strcmp(argv[2], "-") != 0)
```

(7-2) 檔案：udp.c

---

line *114* (changed) -- return函數第五參數資料形態的定義與系統有別，請用Unix系統指令man return查看其定義並更改之。

```
    return(recvfrom(s, buf, len, 0, fhost, &fhlen));
  ->
    return(recvfrom(s, buf, len, 0, (struct sockaddr *)fhost, &fhlen));
```

==========

(8) 目錄： obj/include/

==========

(8-1) 檔案：conf.h

---

line *65* (marked) -- 此句應是程式的註解

```
    Error: byte order not defined.
  ->
    /* Error: byte order not defined. */
```

line *72* (marked) -- 同上

```
    Error: how big is this machine anyways?
  ->
    /* Error: how big is this machine anyways? */
```

==========

(9) 目錄： obj/kadmin/

==========

(9-1) 檔案：admin_server.c

---

line *200* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
  (void) signal(SIGINT, doexit);
  (void) signal(SIGTERM, doexit);
  (void) signal(SIGHUP, doexit);
  (void) signal(SIGQUIT, doexit);
->
  (void) signal(SIGINT, (void (*)())doexit);
  (void) signal(SIGTERM, (void (*)())doexit);
  (void) signal(SIGHUP, (void (*)())doexit);
  (void) signal(SIGQUIT, (void (*)())doexit);
```

line *205* (changed) -- 同上

```
  (void) signal(SIGALRM, doexit);
  (void) signal(SIGCHLD, do_child);
->
  (void) signal(SIGALRM, (void (*)())doexit);
  (void) signal(SIGCHLD, (void (*)())do_child);
```

line *410* (changed) -- wait函數第一參數資料形態的定義與系統有別，請用Unix系統指令man wait查看其定義並更改之。

```
  pid = wait(&status);
->
  pid = wait((int *)&status);
```

(9-2) 檔案：get_srvtab.c

---

line *250* (changed) -- strcpy函數第二參數資料形態的定義與系統有別，請用Unix 系統指令man strcpy查看其定義並更改之。

```
strcpy(my_hostname, krb_get_phost(my_hostname));
->
strcpy(my_hostname, (const char *)krb_get_phost(my_hostname));
```

(9-3) 檔案：kadm_funcs.c

---

line *613* (changed)

```
strcpy(principal.name, values->name, ANAME_SZ);
strcpy(principal.instance, values->instance, INST_SZ);
->
strncpy(principal.name, values->name, ANAME_SZ);
strncpy(principal.instance, values->instance, INST_SZ);
```

==========

(10) 目錄： obj/lib/des/

==========

(10-1) 檔案：des.c

---

line *88* (marked) -- 此句應是程式註解。

```
    dunno how to do this machine type, you lose;
  ->
    /*  dunno how to do this machine type, you lose; */
```

(10-2) 檔案：make_p.c

---

line *21* (added) -- i 的資料形態忽略定義。

```
    void gen(stream)
        FILE *stream;
    {
  ->
    void gen(stream)
        FILE *stream;
    {
        int i;
```

(10-3) 檔案：new_rnd_key.c

---

line *71* (added) -- #define BSDUNIX。

```
    #ifndef BSDUNIX
    you lose...  (aka, you get to implement an analog of this for your
                system...)
    #else
  ->
    #define BSDUNIX

    #ifndef BSDUNIX
    you lose...  (aka, you get to implement an analog of this for your
                system...)
```

#else

line *113* (changed) -- gettimeofday函數第二參數資料形態的定義與系統有別請用Unix系統指令man gettimeofday查看其定義並更改之。

```
gettimeofday(&time, (struct timeval *)0);
->
gettimeofday(&time, (struct timezone *)0);
```

(10-4) 檔案：quad_cksum.c

---

line *84* (added) -- #define LSBFIRST

```
#ifdef LSBFIRST
#ifdef MUSTALIGN
static unsigned long vaxtohl();
static unsigned short vaxtohs();
->
#define LSBFIRST

#ifdef LSBFIRST
#ifdef MUSTALIGN
static unsigned long vaxtohl();
static unsigned short vaxtohs();
```

(10-5) 檔案：random_key.c

---

line *105* (marked) -- 此句應爲程式註解

```
dont know how to do random numbers for this machine;
->
/* dont know how to do random numbers for this machine; */
```

==========

(11) 目錄： obj/lib/kadm/

==========

(11-1) 檔案：kadm_cli_wrap.c

---

line *559* (changed) -- signal函數資料形態的定義與系統有別，請用Unix系統指令 man signal查看其定義並更改之。

```
    opipe = signal(SIGPIPE, SIG_IGN);
  ->
    opipe = (sigtype (*)())signal(SIGPIPE, SIG_IGN);
```

line *566* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

```
    (void) signal(SIGPIPE, opipe);
  ->
    (void) signal(SIGPIPE, (void (*)())opipe);
```

line *573* (changed) -- 同上。

```
    (void) signal(SIGPIPE, opipe);
  ->
    (void) signal(SIGPIPE, (void (*)())opipe);
```

line *582* (changed) -- 同上。

```
    (void) signal(SIGPIPE, opipe);
  ->
    (void) signal(SIGPIPE, (void (*)())opipe);
```

==========

(12) 目錄： obj/lib/krb/

==========

(12-1) 檔案：in_tkt.c

---

line *40* (changed) -- creat()函數資料形態的定義與系統有別，請用Unix系統指令man creat查看其定義。

```
    int tktfile, creat();
->
    int tktfile;
```

(12-2) 檔案：log.c

---

line *59* (marked):

```
    char *month_sname();
->
    /* char *month_sname(); */
```

(12-3) 檔案：util.c

---

line *42* (added):

```
    printf("\n%s %s %s %s flags %u cksum 0x%X\n\ttkt_tm 0x%X
sess_key",
            x->pname, x->pinst, x->prealm,
->
    struct in_addr temp;
    temp.s_addr = x->address;
```

```
    printf("\n%s %s %s %s flags %u cksum 0x%X\n\ttkt_tm 0x%X
sess_key",
            x->pname, x->pinst, x->prealm,
```

line *44* (changed):

```
    inet_ntoa(x->address), x->k_flags,
  ->
    inet_ntoa(temp), x->k_flags,
```

(12-4) 在此目錄下，請把底下的檔案名稱予以更動。因爲在 FreeBSD 1.0 版裡，一個目的檔的檔案名稱其字母數字最多不可以超過 16 個字母。否則 ar 程式所製造出來的函數庫檔案會產生： bad magic number 的錯誤訊息。而整個編譯的動作就會停止下來。

create_auth_reply.c	->	**crea_au_rep.c**
create_death-packet.c	->	**crea_dea_pkt.c**
extract_ticket.c	->	**ext_tkt.c**
get_svc_in_tkt.c	->	**get_s_i_tkt.c**
get_tf_fullname.c	->	**get_tf_fname.c**
krb_get_in_tkt.c	->	**krb_g_i_tkt.c**
read_service_key.c	->	**read_sev_key.c**
save_credentials.c	->	**save_cred.c**

==========

(13) 目錄： **obj/server/**

==========

(13-1) 檔案：kerberos.c

---

line *39* (changed) -- sin變數與內存數學函數sin名字相同，故更改名字。

```
    struct sockaddr_in sin = {AF_INET};
  ->
    struct sockaddr_in ncu_sin = {AF_INET};
```

line *227* (changed) -- 同上

```
    sin.sin_port = sp->s_port;
  ->
    ncu_sin.sin_port = sp->s_port;
```

line *236* (changed) -- bind函數第二參數資料形態的定義與系統有別，請用Unix系統指令man bind查看其定義並更改之。

```
    if (bind(f, &sin, S_AD_SZ, 0) < 0) {
  ->
    if (bind(f, (struct sockaddr *)&ncu_sin, S_AD_SZ) < 0) {
```

line *297* (changed) -- recvfrom函數第五參數資料形態的定義與系統有別，請用Unix系統指令man recvfrom查看其定義並更改之。

```
    n = recvfrom(f, pkt->dat, MAX_PKT_LEN, 0, &from, &fromlen);
  ->
    n = recvfrom(f, pkt->dat, MAX_PKT_LEN, 0, (struct sockaddr
*)&from, &fromlen);
```

line *481* (changed) -- sendto函數第五參數資料形態的定義與系統有別，請用Unix系統指令man sendto查看其定義並更改之。

```
    sendto(f, rpkt->dat, rpkt->length, 0, client, S_AD_SZ);
  ->
    sendto(f, rpkt->dat, rpkt->length, 0, (const struct sockaddr *)client,
S_AD_SZ);
```

line *596* (changed) -- 同上。

```
    sendto(f, rpkt->dat, rpkt->length, 0, client, S_AD_SZ);
```

->

sendto(f, rpkt->dat, rpkt->length, 0, **(const struct sockaddr *)**client, S_AD_SZ);

line *675* (changed) -- 同上。

sendto(f, e_pkt->dat, e_pkt->length, 0, client, S_AD_SZ);

->

sendto(f, e_pkt->dat, e_pkt->length, 0, **(const struct sockaddr *)**client, S_AD_SZ);

==========

(14) 目錄： obj/slave/

==========

(14-1) 檔案：kprop.c

---

line *105* (changed):

char *malloc(), *strcat(), *strcpy();

->

char ***strcpy();**

line *306* (changed) -- connect函數第二參數資料形態的定義與系統有別，請用Unix系統指令man connect查看其定義並更改之。

if (connect(s, &sin, sizeof sin) < 0) {

->

if (connect(s, **(struct sockaddr *)**&sin, sizeof sin) < 0) {

line *316* (changed) -- getsockname函數第二參數資料形態的定義與系統有別，請用Unix系統指令man getsockname查看其定義並更改之。

```
    if (getsockname (s, &my_sin, &n) != 0) {
  ->
    if (getsockname (s, (struct sockaddr *)&my_sin, &n) != 0) {
```

line *495* (changed) -- strcat函數資料形態的定義與系統有別，請用Unix系統指令man strcat查看其定義並更改之。

```
    strcat(strcpy(pc, cs->name), "-last-prop");
  ->
    (char *)strcat(strcpy(pc, cs->name), "-last-prop");
```

(14-2) 檔案：kpropd.c

---

line *171* (changed) -- bind函數第二參數資料形態的定義與系統有別，請用Unix系統指令man bind查看其定義並更改之。

```
    if (bind(s, &sin, sizeof sin) < 0) {
  ->
    if (bind(s, (struct sockaddr *)&sin, sizeof sin) < 0) {
```

line *198* (changed) -- accept函數第二參數資料形態的定義與系統有別，請用Unix系統指令man accept查看其定義並更改之。

```
    if ((s2 = accept(s, &from, &from_len)) < 0) {
  ->
    if ((s2 = accept(s, (struct sockaddr *)&from, &from_len)) < 0) {
```

line *205* (changed) -- gethostbyaddr函數第一參數資料形態的定義與系統有別，請用Unix系統指令man gethostbyaddr查看其定義並更改之。

```
if ((hp = gethostbyaddr(&(from.sin_addr.s_addr), from_len, AF_INET)) ==
NULL) {
```

->

if ((hp = gethostbyaddr((**char ***)&(from.sin_addr.s_addr), from_len, AF_INET)) == NULL) {

line *216* (changed) -- getsockname函數第二參數資料形態的定義與系統有別，請用Unix系統指令man getsockname查看其定義並更改之。

```
    if (getsockname (s2, &sin, &n) != 0) {
->
    if (getsockname (s2, (struct sockaddr *)&sin, &n) != 0) {
```

==========

(15) 目錄： obj/util/et/

==========

(15-1) 檔案：com_err.c

---

line *19* (marked) -- 系統內有vfprintf此函數，故無需定義。

```
    #define vfprintf(stream,fmt,args) _doprnt(fmt,args,stream)
->
    /* #define vfprintf(stream,fmt,args) _doprnt(fmt,args,stream) */
```

line *28* (marked) -- 同上。

```
    #   undef vfprintf
    #   define vfprintf(stream,fmt,args) _doprnt(fmt,args,stream)
->
    /*
    #   undef vfprintf
    #   define vfprintf(stream,fmt,args) _doprnt(fmt,args,stream)
    */
```

line *44* (marked) -- 同上。

```
#undef vfprintf
#define vfprintf(stream,fmt,args) _doprnt(fmt,args,stream)
->
/*
#undef vfprintf
#define vfprintf(stream,fmt,args) _doprnt(fmt,args,stream)
*/
```

(15-2) 檔案：compile_et.c

---

line *43* (changed):

```
extern int yylineno;
->
int yylineno;
```

(15-3) 檔案：error_table.y

---

line *3* (changed) -- malloc函數資料形態的定義與系統有別，請用Unix系統指令man malloc查看其定義並更改之。

```
char *str_concat(), *ds(), *quote(), *malloc(), *realloc();
->
char *str_concat(), *ds(), *quote(), *realloc();
void *malloc()
```

line *79* (changed) -- 同上。

```
char *malloc(), *realloc();
```

->

char ***realloc();**

void *malloc();

(15-4) 檔案：internal.h

---

line *11* (changed):

extern char const * const sys_errlist[];

->

extern **char** *sys_errlist[];

==========

(16) 目錄： obj/util/imake.includes/

==========

(16-1) 檔案：config.Imakefile *(注意：本檔案須視本身環境來作修改)*

---

line *75* (changed) -- 僅供參考，必須與[SOURCE_DIR]一致。

SRCTOP=/afs/net/project/krb4/src

->

SRCTOP=**/mit/kerberos/src**

line *112* (changed) -- 僅供參考，必須爲你所使用之REALM NAME。

SITE_KRB_REALM=ATHENA.MIT.EDU

->

SITE_KRB_REALM=**NET.NCU.EDU.TW**

line *232* (marked) -- 若有利用Hesiod，請忽略此修改，即保持define。*(有關 Hesiod的說明，請參考本報告之第？章)*

```
    #define USE_HESIOD
  ->
    /* #define USE_HESIOD */
```

line *255* (changed) -- 僅供參考，必須爲各下組織的名字。

```
    ORGANIZATION=MIT Athena
  ->
    ORGANIZATION=NCU Athena
```

line *313* (marked) -- 若有lint在系統，請忽略此修改。*(lint爲一C語言程式原始碼的檢查程式)*

```
    #if (defined(vax) && !defined(ultrix)) || (defined(ibm032)) ||
defined(sun)
    LINTLIBFLAG=-C
    #else
    LINTLIBFLAG=-o
    #endif
    #if defined(_AIX) && (AIXV > 30) || defined(_AUX_SOURCE)
  ->
    /*
    #if (defined(vax) && !defined(ultrix)) || (defined(ibm032)) ||
defined(sun)
    LINTLIBFLAG=-C
    #else
    LINTLIBFLAG=-o
    #endif
    #if defined(_AIX) && (AIXV > 30) || defined(_AUX_SOURCE)
    */
```

line *320* (marked) -- 若有lint在系統，請忽略此修改。

```
    #endif
```

->
/* #endif */

line *418* (marked):

#define PROFILED_LIBS
->
/* #define PROFILED_LIBS */

line *515* (changed) -- 僅供參考，必須爲系統內某一使用者。

DEF_UID = 113
->
DEF_UID = **athena**

==========
(17) 目錄： obj/util/makedepend/
==========

(17-1) 檔案：main.c
---

line *84* (changed) -- catch函數資料形態的定義與系統有別，請用Unix系統指令man catch查看其定義並更改之。

catch,
->
(void (*)())catch,

line *297* (changed) -- fprintf函數第二參數資料形態的定義與系統有別，請用Unix系統指令man fprintf查看其定義並更改之。

fprintf(stderr, x0,x1,x2,x3,x4,x5,x6,x7,x8,x9);
->
fprintf(stderr, **(const char *)**x0,x1,x2,x3,x4,x5,x6,x7,x8,x9);

==========

(18) 目錄： **obj/util/ss/**

==========

(18-1) 檔案：help.c

---

line *77* (changed) -- wait函數第一參數資料形態的定義與系統有別，請用Unix系統指令man wait查看其定義並更改之。

```
    while (wait((union wait *)NULL) != child) {
->
    while (wait((int *)NULL) != child) {
```

(18-2) 檔案：list_rqs.c

---

line *37* (changed) -- 參考下文line *87* 。

```
    int (*func)();
->
    void (*func)();
```

line *85* (changed) -- wait函數第一參數資料形態的定義與系統有別，請用Unix系統指令man wait查看其定義並更改之。

```
    wait(&waitb);
->
    wait((int *)&waitb);
```

line *87* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

(void) signal(SIGINT, func);

->

(void) signal(SIGINT, **(void (*)())**func);

(18-3) 檔案：listen.c

---

line *77* (changed) -- signal函數及其第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

sig_int = signal(SIGINT, listen_int_handler);

->

sig_int = **(int (*)())**signal(SIGINT, **(void (*)())**listen_int_handler);

line *84* (changed) -- signal函數及其第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

sig_cont = signal(SIGCONT, print_prompt);

->

sig_cont = **(int (*)())**signal(SIGCONT, **(void (*)())**print_prompt);

line *97* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

(void) signal(SIGCONT, sig_cont);

->

(void) signal(SIGCONT, **(void (*)())**sig_cont);

line *119* (changed) -- signal函數第二參數資料形態的定義與系統有別，請用Unix系統指令man signal查看其定義並更改之。

(void) signal(SIGINT, sig_int);

->

(void) signal(SIGINT, **(void (*))**sig_int);

(18-4) 檔案：mk_cmds.c

---

line *27* (changed):

```
    extern int yylineno;
->
    int yylineno;
```

附錄４Ｂ　參考文獻

（１）Steiner J. G., C. Neuman, and J. I. Schiller, "*Kerberos: An Authentication Service for Open Network Systems,*" pp. 191-202in Usenix Conference Proceedings, Dallas, Texas (February, 1988).

Ftpsite: athena-dist.mit.edu:\pub\kerberos\doc\usenix.PS

（２）吳瑞明，"*網路安全技術與應用*　，" pp. 54-72 in 網路通訊雜誌，Taiwan (September, 1992).

（３）Bryant B., J. Steiner, and J. Kohl,"*Kerberos Install —ation Notes DRAFT*　," M.I.T. Project Athena (1989).

Ftpsite: athena-dist.mit.edu:\pub\kerberos\doc\installation.PS

（４）Bryant B., J. Kohl, "*Kerberos Operation Notes DRAFT,*" M.I.T.Project Athena (1989).

Ftpsite: athena-dist.mit.edu:\pub\kerberos\doc\operation.PS

（５）"*README.KRB4,*" M.I.T. Project Athena

Ftpsite: athena-dist.mit.edu:\pub\kerberos\README.KRB4