

第二十章

本章內容：

- * `checkcompat()` 的工作原理
- * 食譜
- * 依字母排序的 V8.8 副常式

checkcompat () 食譜

在 `sendmail` 的內部有個經常被忽略的 `checkcompat()` 常式。這個常式自從第三版就開始存在了，其目的是讓網站管理員能夠接收、拒絕與登錄遞送郵件的企圖。它的用途，也正是我們想要解說的內容，如下所示：

- 拒絕充當郵件開道器。拒絕處理任何送件人與收件人同時隸屬本地地址以外的郵件。
- 限制問候訪客 (`guest`) 之訊息的大小。依據使用者的 `uid` 篩選訊息，並當訊息超過所指定的大小時，拒絕接受訪客類型的使用者。
- 驗證 `identd(8)` 的資訊，比較在 `$s` 與 `$_` 中的主機資訊，如果兩者不同就登錄一個警告訊息。
- 在防火牆上刪剪 `Received:` 標頭。讓所有寄出的郵件在經過防火牆後，都像是從內部網路寄出的一樣。
- 拒絕來自垃圾郵件 (`spamming`) 或郵件炸彈 (`mail-bombing`) 網站的郵件。在一個外部資料庫中查詢寄件人，如果寄件人被註記為 `bad` (不守規矩的)，就退回他所寄來的郵件。

`checkcompat()` 常式放在 `src/conf.c` 中。【編註】該檔內含註解，說明了撰寫 `checkcompat()` 常式的方法。本章我們將要告訴你另一種方法。

`checkcompat()` 與生俱來就是在內部使用的常式，這表示它必須瞭解可能改變的內部資料結構【註】。既然你正要修改原始程式碼，那麼你一定已經讀過原始程式碼。本章我們將會提供一些 `checkcompat()` 的使用範例。請注意，它們只是範例而已，你還需俱備 C 語言的程式設計技巧，才能將它們擴展到真實世界的層面。

當每一次企圖遞送郵件給收件人時，`sendmail` 都會叫用一次 `checkcompat()` 常式。請注意，當你自行設計 `checkcompat()` 常式時，假使稍有不慎，則可能產生一連串的錯誤。舉例而言，所登錄的警告訊息乃是依據收件人而產生的，如果有多位收件人，則會產生許多的警告訊息。

20.1 `checkcompat()` 的工作原理

當 `sendmail` 準備遞送郵件時，它會先檢查郵件訊息的大小，如果超過 `M=` 遞送代理程式等式（參見 § 30.4.7）所設定的值，便會退回（bounce）該郵件。如果是 V8.8 `sendmail`，還會接著叫用 `check_compat` 規則集（參見 § 29.10.4）。然後，所有版本的 `sendmail` 都會叫用 `checkcompat()` 常式。

`checkcompat()` 常式放在 `sendmail` 程式碼中一個獨一無二的位置上。在那個位置上可以同時取得寄件人與收件人的地址。由於它執行在實際遞送進行之前，因此你可以檢查遞送所需的所有資訊。

如果 `checkcompat()` 所傳回的值是 `<sys/exit.h>` 所定義的 `EX_OK`，則郵件訊息會被視為沒問題而遞送出去；否則該訊息會被退回。如果你希望的是郵件被重新放入佇列，而不是被退回，你可以傳回 `EX_TEMPFAIL`。

再強調一次，`sendmail` 對每一位收件人都會叫用一次 `checkcompat()` 常式。

編註 對 V8.10.2 `sendmail` 而言 `checkcompat()` 常式放在 `sendmail/conf.c` 中。

註 由於 V8.8 `sendmail` 還提供有 `check_compat` 規則集（參見 § 29.10.4），因此可以在規則集的層級上執行 `checkcompat()` 常式的部份功能。這是一種可以避免撰寫 C 程式碼的方法。

20.1.1 傳給 checkcompat() 的引數

你可以在 `src/conf.c` 這個 C 語言的原始碼檔中找到 `checkcompat()`。在該檔案中你可以找到 `checkcompat()` 的宣告：

```
checkcompat(to, e)
    register ADDRESS *to;
    register ENVELOPE *e;
```

其中，`to` 是個指標，指向以 `typedef` 定義的 `ADDRESS` 結構，`ADDRESS` 內含收件人的資訊。而 `e` 也是一個指標，指向以 `typedef` 定義的 `ENVELOPE` 結構（實際上是一個鏈結串列（linked list）的結構），`ENVELOPE` 內含目前信封的資訊。

`ADDRESS *to` 結構中的成員顯示在表 20-1 中。請注意，這些成員只有在 V8.8 的 `sendmail` 中才是正確的。另外也請注意，表中只顯示 `checkcompat()` 常式可能會用到的成員（參見 `sendmail.h` 關於 `*to` 的其他成員）。

表 20-1：ADDRESS *to 的成員

型別	成員	說明
char *	q_paddr	此地址具有適於列印的形式
char *	q_user	從規則集 0（參見 § 29.6）所得的使用者部份（\$）
char *	q_ruser	使用者的登入名稱（如果知道的話）
char *	q_host	從規則集 0（參見 § 29.6）所得的主機部份（\$@）
struct mailer *	q_mailer	從規則集 0（參見 § 29.6）所得的遞送代理程式（\$#）
u_long	q_flags	狀態旗標（參見 § 37.3.1 的表 37-1）
uid_t	q_uid	q_ruser 的 uid（如果知道的話）
gid_t	q_guid	q_ruser 的 gid（如果知道的話）
char *	q_home	使用者的登入目錄（home directory），只適用在本地的遞送
char *	q_fullname	q_ruser 的（geos）完整名稱（如果知道的話）
struct address *	q_next	鏈結到資料鏈（chain）中的下一個 ADDRESS
struct address *	q_alias	產生這個地址的別名
char *	q_owner	q_alias 的擁有人

而 `ENVELOPE *e` 結構中的成員則顯示在表 20-2 中。請注意，這些成員只有在 V8.8 `sendmail` 中才是正確的。另外也請注意，表中只顯示 `checkcompat()` 常式可能會用到的成員（參見 `sendmail.h` 關於 `*e` 的其他成員）。

表 20-2：ENVELOPE *e 的成員

型別	成員	說明
HDR *	<code>e_header</code>	標頭的鏈結串列
<code>time_t</code>	<code>e_ctime</code>	首次放入佇列的時間訊息
ADDRESS	<code>e_from</code>	寄件人
ADDRESS *	<code>e_sendqueue</code>	收件人的鏈結串列
<code>long</code>	<code>e_msgsize</code>	訊息的大小，以位元組為單位
<code>long</code>	<code>e_flags</code>	信封旗標（參見 § 37.5.12 的表 37-3）
<code>int</code>	<code>e_nrcpts</code>	收件人的數目
<code>short</code>	<code>e_hopcount</code>	郵件訊息所經過的網站計次（hop count）

`checkcompat()` 在 `sendmail` 中是個功能強大的內部工具。由於它是如此接近 `sendmail` 的內部而且功能強大，因此如果你夠聰明的話，你甚至可以在執行時期透過 `checkcompat()` 來變更改寫規則（雖然讓人有點提心吊膽，但卻是可能的）。

20.1.2 全域變數

V8.8 `sendmail` 使用的全域變數超過 100 個。那些變數全部列在 `sendmail.h` 與 `conf.c` 中，並附有簡單的註解。全域變數存放了許多資訊，例如：`sendmail` 的選項值、檔案代碼（file descriptor）的值、巨集的值、類型清單以及資料庫存取資訊。每個全域變數都可以透過 `checkcompat()` 來進行修改，但在修改之前，請先研究一下 `sendmail` 的 C 原始程式碼，預先考慮任何非預期的副作用。

一般而言，當你設計自己的 `checkcompat()` 常式時，你幾乎可以使用任何的全域變數。其中最讓人感興趣的是下面四個：

RealHostAddr

寄件主機的 IP 位址。它是一個 union 型別的變數，內含多個 sockaddr_ 型別，視你選用的協定類型而異。對於本地遞送的郵件，這個變數的值可以是零。

RealHostName

這是個字串內，含寄件主機最可靠的正規名稱。如果無法解析此機器名稱，則它會以文字型態存儲主機的 IP 位址，並且是放在中括號 ([]) 之中。

LogLevel

這個變數決定了 sendmail 應該登錄的訊息數量。它的初始值是由 LogLevel (L) 選項設定的 (參見 § 34.8.33)。你可能會使用 checkcompat() 偵測有問題的網路連線，如果找到了，就將 LogLevel 的值增加到 12。這樣便可將連線兩端之後的每個 SMTP 連線都登錄下來。

MatchGecos

在本地搜尋的名稱，不論是否相符，都會透過 MatchGECOS (G) 選項 (參見 § 34.8.34) 的控制，在 passwd (5) 檔中進行搜尋。因為這樣的搜尋是很昂貴的，你可能只希望非商業運作的時間開啟它。解決的方法之一是修改 checkcompat() 中的 MatchGecos 變數。

20.2 食譜

本節展示了幾個範例，說明 checkcompat() 常式的可能用法。我們將說明下面幾個項目：

- 只接受從自己網域寄出的郵件 (參見 § 20.2.1)。
- 避免讓你的工作站成為一個郵件閘道器 (參見 § 20.2.2)。
- 限制 guest 帳號的訊息大小 (參見 § 20.2.3)。
- 驗證 identd 資訊是否正確 (參見 § 20.2.4)。
- 在防火牆處修剪 Received: 標頭 (參見 § 20.2.5)。

- 拒絕接收來自垃圾郵件網站或郵件炸彈網站的郵件（參見 § 20.2.6）。

請注意，在下面所有的範例中，每一行最左邊的編號僅供討論說明之用，並非程式碼的一部份。

20.2.1 只接受自己網域所寄出的郵件

如果你的網站位於防火牆【註】之後，你可能會透過 `checkcompat()` 來調校位於內部的 `sendmail`，讓它只接受本地所產生的郵件。外部的 `sendmail`（在防火牆之外，或位在防火牆上）扮演的是 `proxy`（網路代理人）的角色。也就是說，它接收由外面要寄到內部的郵件，並將之轉送給內部的 `sendmail`。由於外部的 `sendmail` 仍是本地網域之一部份，因此它的信封看起來總像是來自本地網域。任何企圖繞過防火牆的外部郵件都應該被退回。若想達到這樣的功能，你可以像這樣來設計 `checkcompat()` 中的程式碼：

```
1  # define OUR_NET_IN_HEX 0x7b2d4300 /* 以十六進位數值表示 123.45.67.0 */
2  # define OUR_NETMASK 0xffffffff00
3
4  checkcompat(to, e)
5      register ADDRESS *to;
6      register ENVELOPE *e;
7  {
8      if (tTd(49, 1))
9          printf("checkcompat(to=%s, from=%s)\n",
10              to->q_paddr, e->e_from.q_paddr);
11
12
13      if (RealHostAddr.sa.sa_family == 0)
14      {
15          /* 這是本地所提交的信息 */
16          return EX_OK;
17      }
18      if (RealHostAddr.sa.sa_family != AF_INET ||
19          (RealHostAddr.sin.sin_addr.s_addr & OUR_NETMASK) != OUR_NET_IN_HEX)
20      {
```

註 防火牆是一部位於本地網路與外部網路之間的機器。它的功能為攔截與篩選網路流量，以及排除任何不適當的封包。

```

21     usrrerr("553 End run mail not allowed");
22     e->e_flags |= EF_NO_BODY_RETN;
23     to->q_status = "5.7.1";
24     return (EX_UNAVAILABLE);
25 }
26 return (EX_OK);
27 }

```

usrrerr() 常式 (第 21 行) 會在寄出郵件的網站上印出一條警告訊息，而所傳回的 EX_UNAVAILABLE (第 24 行) 則會使得該郵件被退回。所退回的郵件會被送回給原寄件人。另外還可以傳送一份副本給本地的 postmaster，這端視 PostmasterCopy (P) 選項的設定與否 (參見 § 34.8.46)。

EF_NO_BODY_RETN (第 22 行) 會造成只將郵件的標頭，而非原始的郵件本體，放到被退回的郵件中。其他較令人感興趣的信封旗標可在 § 37.5.12 的表 37-3 中找到。

to->q_status (第 23 行) 會在被退回的郵件訊息中傳達 DSN 的錯誤狀態 (參見 RFC 1893)。此處的 5.7.1 指出這是「政策狀態」(7) 下的「永久性失敗」(5)，表示該遞送並未獲得認可，因而「拒絕接受」(1) 所遞送的郵件訊息。

另外請注意，這個程式碼範例只是個建議而已，它並沒有考慮到 RealHostAddr 的值可能為 0x7f000001 (127.0.0.1 即 localhost)。

20.2.2 拒絕讓工作站成為郵件閘道器

如果你已經花費了數個月的時間才將你的工作站設定好，而它也非常完美地運作著，此時你或許不希望讓外面的人以它做為郵件的轉發伺服器。若想防止這種不當的使用方式，則可以設定 conf.c 檔中的 checkcompat()，讓工作站拒絕接受送件人來自你機器以外之網站而且收件人亦為你機器以外網站的郵件。這樣做還有個不錯的副作用：可以避免郵件從外面直接寄給你內部的郵件名單。

```

1  checkcompat(to, e)
2      register ADDRESS *to;
3      register ENVELOPE *e;
4  {

```

```

5
6     if (tTd(49, 1))
7         printf("checkcompat(to=%s, from=%s)\n",
8             to->q_paddr, e->e_from.q_paddr);
9
10        if (RealHostAddr.sa.sa_family == 0)
11        {
12            /* 這是本地所提交的信息 */
13            return (EX_OK);
14        }
15        /* 只接受從外地寄給本地的郵件 */
16        if (!bitnset(M_LOCALMAILER, to->q_mailer->m_flags))
17        {
18            userrr("553 External gateway use prohibited");
19            e->e_flags |= EF_NO_BODY_RETN;
20            to->q_status = "5.7.1";
21            return (EX_UNAVAILABLE);
22        }
23        return (EX_OK);
24    }

```

雖然 `to` (第 16 行) 事實上是一個收件人的鏈結串列，但我們只檢查目前的收件人，以避開偽造的警告訊息。之所以這樣做是因為 `sendmail` 對每一位收件人都會叫用一次 `checkcompat()`。第 16 行的檢查是要看看 `F=1` 的遞送代理程式旗標設定了沒有 (參見 § 30.8.28)，如果沒有設定，就表示收件人不在本地。

請注意，這種拒絕接受郵件訊息的形式並不適用於 `mail hub` (郵件集散中心)。在 `mail hub` 上需要更複雜的檢查方法。其內容如下所示：

- 檢查你網站所有的 IP 網域，如果只有一個，則適用 § 20.2.1 所說的檢查方式。如果有好幾個 (如同 C 等級網域的分類方式)，則檢查的工作就會比較複雜。如果連線的主機位於你的網域中，或是在你所屬之許多網域中的一個，你都應該接受該郵件。
- 應該要檢查信封上寄件人的主機 (`e->e_from->q_host`)，看看它是否存在於 `$=w` 類型中 (參見 § 32.5.8)。你可以使用 `wordinclass()` 常式 (參見 § 20.3.8) 進行搜尋。如果它存在於 `$=w` 之中，你就應該接受該訊息。這樣便可避免郵件被工作站轉送出去。

- 如果收件人的遞送代理程式是 `*include*`，則表示該郵件的目的地是郵件名單。你可能希望在這個時候做進一步的篩選。

20.2.3 限制訪客訊息的大小

假設你的網站保留了 900 到 999 的 uid 編號給訪客 (guest user) 使用。由於訪客有時比較不會顧慮到別人，因此你可能會想要限制他們所送出之訊息的大小，以及他們同時能指定的收件人數目。若想達到這樣的功能，你可以像這樣來設計 `checkcompat()` 中的程式碼：

```

1  #define MAXGUESTSIZE 8000
2  #define MAXGUESTNRCP 4
3
4  checkcompat(to, e)
5      register ADDRESS *to;
6      register ENVELOPE *e;
7  {
8
9      if (tTd(49, 1))
10         printf("checkcompat(to=%s, from=%s)\n",
11             to->q_paddr, e->e_from.q_paddr);
12
13         /* q_uid 內含有效的 uid? -- 非來自外地 */
14         if (! bisset(QGOODUID, e->e_from.q_flags))
15             return (EX_OK);
16         if (e->e_from.q_uid < 900 || e->e_from.q_uid > 999)
17             return (EX_OK);
18         if (e->e_msgsize > MAXGUESTSIZE)
19             {
20                 syslog(LOG_NOTICE,
21                     "Guest %s attempted to send %d size",
22                     e->e_from.q_user, e->e_msgsize);
23                 usrrerr("553 Message too large, %d max", MAXGUESTSIZE);
24                 e->e_flags |= EF_NO_BODY_RETN;
25                 to->q_status = "5.7.1";
26                 return (EX_UNAVAILABLE);

```

```

27     }
28     if (e->e_nrcpts > MAXGUESTNRCP)
29     {
30         syslog(LOG_NOTICE,
31             "Guest %s attempted to send %d recipients",
32             e->e_from.q_user, e->e_nrcpts);
33         usrrrr("553 Too many recipients for guest, %d max",
34             MAXGUESTNRCP);
35         e->e_flags &= ~EF_NO_BODY_RETN;
36         to->q_status = "5.7.1";
37         return (EX_UNAVAILABLE);
38     }
39     return (EX_OK);
40 }

```

請注意，只有當寄件人位於本地時（第 14 行），`q_uid` 才會擁有一個有效的 `uid`（而且會設定成 `QGOODUID`）。當外部的郵件進來時，`QGOODUID` 將會被清除掉。

另外也請注意，如果郵件被退回是因為郵件太大了（第 24 行），則我們會指定不要將郵件的本體一併傳回。但如果是因為收件人太多而發生退件的情況（第 35 行），則我們才會傳回郵件的本體。其他令人感興趣的信封旗標可在 § 37.5.12 的表 37-3 中找到。

20.2.4 確認 `identd` 資訊

當一部外面的主機透過 `SMTP` 與本地的 `sendmail` 連線時，其主機名稱將會存放在 `$_s` 巨集中（參見 § 31.10.33）。如果 `Timeout.ident` 選項（參見 § 34.8.70）的值不為零，`sendmail` 會使用 `RFC 1413` 的驗證協定記錄另一端主機的身份（`identity`），也就是說，它會記錄進行連線之主機的身份。該身份資料將會記錄在 `$_` 巨集中（參見 § 31.10.1）。

如果你對其他主機的身份特別吹毛求疵，你可能會想要確認 `$_s` 與 `$_` 中主機的一致性。若想執行這樣的檢查，你可以像這樣來設計 `checkcompat()` 常式：

```

1     checkcompat(to, e)
2         register ADDRESS *to;

```

```
3     register ENVELOPE *e;
4     {
5     char *s, *u, *v;
6     int len;
7     static char old_s[MAXHOSTNAMELEN];
8
9     if (tTd(49, 1))
10        printf("checkcompat(to=%s, from=%s)\n",
11             to->q_paddr, e->e_from.q_paddr);
12
13     /* 若 $s 是 localhost 或列在 $=w 中, 則接受此郵件 */
14     if ((s = macvalue('s', e)) == NULL)
15         return (EX_OK);
16     if (strncasecmp(s, old_s, MAXHOSTNAMELEN-1) == 0)
17         return (EX_OK);
18     else
19         (void)sprintf(old_s, "%.s", MAXHOSTNAMELEN-1, s);
20     if (strcasecmp(s, "localhost") == 0)
21         return (EX_OK);
22     if (wordinclass(s, 'w') == TRUE)
23         return (EX_OK);
24
25     if ((u = macvalue('_', e)) == NULL)
26         return (EX_OK);
27     if ((u = strchr(u, '@')) == NULL)
28         return (EX_OK);
29     if ((v = strchr(u, ' ')) != NULL)
30         *v = '\0';
31     len = strlen(u);
32     if (v != NULL)
33         *v = ' ';
34
35     if (strncasecmp(s, u, len) != 0)
36     {
37         auth_warning(e, "$s=%s doesn't match $_=%.s", s, len, u);
38     }
39     return (EX_OK);
40 }
```

首先（第 16 行）我們要先確定是否已經檢查過 `$s` 的值。如果是的話，則我們就不再檢查，因為 `sendmail` 對每個收件人都會叫用一次 `checkcompat()`。如果 `$s` 的值是新的，則我們會複製一份它的值，以供下一次使用。

然後我們要確定本地主機（不管其名稱為何）是可以接受的（第 20 與 22 行）。如果這是一個在外地的（`offsite`）主機，則我們就會比較 `$s` 的值與 `$_` 的主機部份（第 35 行）。如果兩者不相符，我們會插入一個 `X-Authentication-Warning:` 標頭（第 37 行）。這個警告訊息受到 `PrivacyOptions.authwarning(p)` 選項（參見 § 34.8.47）的控制。

20.2.5 在防火牆處刪剪 Received: 標頭

當郵件繞經防火牆向外傳送時（參見 § 20.2.1），最好能夠以一個主標頭（`master header`）來取代所有內部的 `Received:` 標頭。若想達成此功能，你可以像這樣來設計 `checkcompat()` 常式：

```

1  # define OUR_NET_IN_HEX 0x7b2d4300 /* 以十六進位數值表示 123.45.67.0 */
2  # define OUR_NETMASK 0xffffffff
3  # define LOOP_CHECK "X-Loop-Check"
4
5  checkcompat(to, e)
6      register ADDRESS *to;
7      register ENVELOPE *e;
8  {
9      HDR *h;
10     int cnt;
11
12     if (RealHostAddr.sa.sa_family == 0)
13     {
14         /* 這是本地所提交的信息 */
15         return EX_OK;
16     }
17     if (RealHostAddr.sa.sa_family != AF_INET ||
18         (RealHostAddr.sin.sin_addr.s_addr & OUR_NETMASK) != OUR_NET_IN_HEX)
19     {
20         /* 所收到的信息並非來自內部網路 */

```

```

21     return EX_OK;
22 }
23 if (hvalue(LOOP_CHECK, e->e_header) != NULL)
24 {
25     /* 我們已經將標頭移除過一次了 */
26     return EX_OK;
27 }
28 addheader(LOOP_CHECK, "", &e->e_header);
29
30 for (cnt = 0, h = e->e_header; h != NULL; h = h->h_link)
31 {
32     if (strcasecmp(h->h_field, "received") != 0)
33         continue;
34     if (cnt++ == 0)
35         continue;
36     clrbitmap(h->h_mflags);
37     h->h_flags |= H_ACHECK;
38 }
39 return (EX_OK);
40 }

```

由於我們要移除 `Received:` 標頭中的訊息，因此我們得要特別注意。如果郵件最初是從防火牆的機器產生的，我們就不該這麼做（第 12 行）。如果郵件最初是從內部（防火牆）以外的網路產生的，我們也不該這麼做（第 17 與 18 行）。為了避免產生不幸的郵件迴圈，我們還要檢查是否有個特殊的標頭（第 23 行），如果有的話，就得跳過移除 `Received:` 標頭訊息的動作。然後我們會加上那個特殊的標頭（第 28 行），以防止這封郵件再度流回到該防火牆。

如果以上的過程都很順利，我們會掃描所有的標頭（第 30 行），尋找所有的 `Received:` 標頭（第 32 行）。我們不會刪除第一個 `Received:` 標頭，因為那是由防火牆放進去的（第 34 行）。我們會刪除其他所有的 `Received:` 標頭，使用的方法是清除它們的 `?flags?` 位元（第 36 行）以及設定 `H_ACHECK` 旗標（第 37 行）。請參見 § 20.3.3 關於這項技術的一般性討論。

請注意，這只是一種可能的實作方法，端視網際網路上其他主機對這封郵件處理的方式，這個迴圈偵測可能會失敗。另一個較安全但較為困難的實作方法是改寫 `Received:` 標頭，並且將其中敏感的資訊隱藏起來。

20.2.6 拒絕接收來自垃圾郵件網站或郵件炸彈網站的郵件

隨著 Internet 的增長，你的網站漸漸成為廣告推銷的對象，也越來越容易受到外來的報復性攻擊。廣告的攻擊稱為「垃圾郵件」(spam)，是由發出廣告的人將許多廣告的副本透過你內部的郵件名單送出，或寄給你網站上的許多使用者。報復性的攻擊稱為「郵件炸彈」(mail bomb)，通常會讓你的郵件暫存目錄塞滿來自單一寄件人的大量郵件【註 1】。

要降低這類事件（以及未來人們還可能發明的其他類似事件）所造成的傷害，你可以結合資料庫與 `checkcompt()`，對外來的郵件進行篩選。首先我們會告訴你如何建立這樣的資料庫，然後我們會展示一個能夠使用這個資料庫的 `checkcompt()` 常式【註 2】。

這個資料庫的原始檔案如下所示：

```
user@spam.host      spam
user@bomb.host      bomb
```

其中，每個左手邊的項目都是個電子郵件地址，內含一個使用者部份、一個 @ 以及一個主機部份。我們會根據個別的寄件人地址篩選郵件，而不是以網站的層級進行篩選。而右手邊則以 `spam` 這個字來表示一個垃圾郵件的寄件人，或是 `bomb` 這個字來表示一個郵件炸彈的寄件人。

如果原始檔案是 `/etc/mail/blockusers`，則產生資料庫的方法如下：

```
% makemap hash /etc/mail/blockusers.db < /etc/mail/blockusers
```

這樣便可以產生一個 `hash db` 樣式的資料庫。其他可以使用的樣式請參考 § 33.2。

註 1 這通常是針對你網站上某個使用者寄出的攻擊性垃圾郵件所做的回應。

註 2 你也可以使用新的 V8.8 `check_mail` 規則集（參見 § 29.10.1），篩選 SMTP MAIL 命令中的寄件人地址。雖然設計 `check_mail` 規則比較容易，但 `checkcompt()` 卻有比較強大的功能。

一旦資料庫準備就緒，接下來還得讓組態檔知道該資料庫的存在。要達到這個目的，你可以使用 `k` 組態命令（參見 § 33.3）：

```
Kbadusers hash -o /etc/mail/blockusers.db
```

就 `m4` 的組態技術而言，你可以將這個宣告放在 `mc` 檔中 `LOCAL_CONFIG` 設定行之下。（參見 § 19.6.30）

若想達成此功能，你可以像這樣來設計 `checkcompat()` 常式：

```

1  checkcompat(to, e)
2      register ADDRESS *to;
3      register ENVELOPE *e;
4  {
5      STAB *map;
6      char *p;
7      int ret = 0;
8
9      map = stab("badusers", ST_MAP, ST_FIND);
10     if (map == (STAB *)NULL)
11         return (EX_OK);
12     p = (*map->s_map.map_class->map_lookup)
13         (&map->s_map, e->e_from.q_paddr, NULL, &ret);
14     if (p == NULL)
15         return (EX_OK);
16
17     if (strcasecmp(p, "spam") == 0)
18     {
19         usrrerr("553 Spamming mail rejected from %s",
20             e->e_from.q_paddr);
21         to->q_status = "5.7.1";
22         return (EX_UNAVAILABLE);
23     }
24     if (strcasecmp(p, "bomb") == 0)
25     {
26         usrrerr("553 Message rejected from mail-bomber %s",
27             e->e_from.q_paddr);
28         e->e_flags &= ~EF_NO_BODY_RETN;

```

```

29         to->q_status = "5.7.1";
30         return (EX_UNAVAILABLE);
31     }
32     return (EX_OK);
33 }

```

其中，我們先在符號表格中搜尋名為 `badusers` 的資料庫（第 9 行）。如果該資料庫不存在就沒有問題（第 11 行）。如果資料庫存在，我們便在資料庫中搜尋寄件人的地址（第 12 行）。如果沒找到該地址，一切也就都沒問題（第 15 行）。

如果在資料庫中找到了該地址，寄件人可能就是個壞人。所以我們先檢查該地址是否被標示為 `spam`（第 17 行）。如果是，就將郵件退回，並附上一個適當的錯誤訊息（第 19 行）。

如果是個郵件炸彈，我們也會將郵件退回（第 24 行）。然而這樣做充滿了風險。退回的郵件可能會塞滿寄出郵件的佇列，這樣只不過是以另一種形式達成了轟炸者的目的。另一個比較好的方法是在最底層將郵件清除掉（參見 `envelope.c` 檔中的 `dropenvelo()` 常式），但我們將這個工作留給讀者自行練習。

20.3 依字母排序的 V8.8 副常式

在 `sendmail` 內部有許多現成的副常式，當你撰寫自己的 `checkcompat()` 常式時，這些副常式十分有用。請一定要先嘗試利用 `sendmail` 的內部副常式，而不要重新設計你自己的副常式，因為 `sendmail` 所提供的副常式是經許多熟練的程式設計人員篩選與測試過的程式碼。

本節列出了我們認為最有用處的幾個副常式，並說明如何善用它們（參見表 20-3）。

表 20-3：可供 `checkcompat()` 使用的副常式

副常式	節次	說明
<code>adheader</code>	20.3.1	在標頭清單中增加一個新的標頭
<code>defne</code>	20.3.2	定義一個巨集

副常式	節次	說明	(續)
	20.3.3	如何移除一個標頭	
hvalue	20.3.4	依名稱搜尋標頭的欄位	
macid	20.3.5	將文字字串的巨集名稱轉換成整數	
macvalue	20.3.6	取出巨集的值	
-> map_lookup	20.3.7	在外部資料庫中搜尋鍵值	
wordinclass	20.3.8	在類型中搜尋一個字	

請注意，這些副程式的說明都是針對 V 8.8 sendmail。如果你使用的是不同版本的 sendmail，在使用這些副程式之前，一定得先瞭解其原始程式。不相符的參數可能會引發難以診斷的錯誤。

addheader() (void)

20.3.1 在標頭清單中增加一個新的標頭

透過 addheader() 常式可以在已經存在的標頭清單中增加一個新的標頭。請注意，因為它並不會檢查標頭是否重複，所以你應該在增加標頭前先檢查該標頭是否已經存在（除非那個標頭可以重複出現）：

```

1 char value[MAXLINE];
2
3 (void)sprintf(value, "Screened by checkcompat() for %s", q_user);
4 (void)addheader("x-screened", value, e->e_header);
```

由於存放新標頭名稱與值的記憶體空間是由 addheader() 內部所配置的，因此可以很安全地傳遞自動變數。請注意，addheader() 的第一個引數（第 4 行）是標頭的名稱。如果你不小心在該引數中放入了一個冒號，則該名稱在儲存時也會含有冒號，而稍後在送出標頭時會不正確地變成兩個冒號。

請注意，addheader 不會檢查其引數的正確性。如果你傳給它 NULL 或一個不合法的地址，可能會讓 sendmail 產生 core-dump（核心傾印）而導致執行失敗。由於此處會動到 sendmail 的內部，因此請特別小心。

`define()` (void)

20.3.2 定義一個巨集

在 `checkcompat()` 中使用 `define()` 常式可以定義或重新定義巨集。方法如下：

```
define( macid, value, envel);
```

其中，若巨集為單一字元的名稱（例如 A）則 `macid` 是個純文字的字元；若巨集為多字元的名稱則 `macid` 是個的識別值（稍後可叫用 `macid()` 來取得），`value` 是指定給巨集的值。（注意，如果巨集已經有值，原有的值會被覆蓋掉。）`envel` 則是一個指向信封的指標（在 `checkcompat()` 中它是 `e`）。

為了清楚說明，考慮下面的程式碼片段，它可以將一個新的值指定給巨集 `$A` 與 `#{foo}`：

```
5  int mid;
6
7  /*
8   * A single-character name is given a value that is the address
9   * of a character constant.
10  */
11  define( 'A', "some text as a value", e);
12
13  /*
14   * A multi-character name is given a value that is in malloc()d
15   * memory space.
16  */
17  mid = macid("#{foo}", NULL);
18  define(mid, newstr(to->q_paddr), e);
```

在這兩種情況底下，`value` 都是存放在永久的記憶體空間中。你絕不可以將一個指標傳遞到自動的或暫時的記憶體空間中，因為 `define()` 只會儲存指標，而不是複製其值【註】。在第二個例子中（第 18 行），字串的記憶體配置是透過 `newstr()` 來完成的，它是另一個內部常式，可以為字串與字串結尾的零字元配置存放空間，將字串複製到配置的空間中，然後傳回一個指向新字串的指標。但要小心地記錄你所做的事。如果在使用 `newstr()` 時不小心，可能會產生嚴重的記憶體流失現象。

註 就技術上而言，你不應該使用字串常數，因為字串常數是存放在唯讀的程式空間裡，且在執行時期通常無法改變其值，例如，當稍後需要適當地修改一個巨集的值時會有困難。

macid() 的第一個引數 (第 17 行) 是一個字串, 內含不帶有 \$ 字首的巨集文字名稱。對多字元的名稱而言, 則必須要有一對大括號 (參見 § 31.4.2)。

請參考 -d35 除錯開關 (§ 37.5.120), 它提供了觀察定義巨集或重新定義巨集的方法。另外如果你要從其他的文字中解析出巨集的文字名稱, 也請參考 readcf.c 中的 munchstring()。

20.3.3 如何移除一個標頭

使用 sendmail 程式的內部常式, 將無法從標頭清單 (鏈結串列) 中移除一個標頭, 但若藉著修改旗標的方式, 則可以防止 sendmail 將該標頭放入遞送的訊息中。

現在考慮將自訂之 X-Accounting: 標頭隱藏起來的需求:

```

1  HDR *h;
2
3  for (h = e->e_header; h != NULL; h = h->h_link)
4  {
5      if (strcasecmp(h->h_field, "x-accounting") != 0)
6          continue;
7      clrbitmap(h->h_mflags);
8      h->h_flags |= H_ACHECK;
9  }
```

其中, 我們必須手動掃描標頭清單 (鏈結串列) (第 3 行)。當找到了要搜尋的標頭時 (注意這項檢查與字母大小寫無關, 第 5 行), 我們便將之刪除。其作法是先將標頭所有的 ?flags? 設定為零 (第 7 行)。然後加上 H_ACHECK 旗標 (第 8 行), 以確保在遞送時會跳過該標頭。請參考 § 35.5 關於 H_ACHECK 旗標效果的討論。

hvalue() 依名稱搜尋標頭的欄位

你可以透過 hvalue() 常式來搜尋標頭並按照自己的意思修改它, 或是在缺少該標頭時將之插入郵件中。它的用法如下所示:

```

1 char *field;
2
3 field = hvalue("message-id", e->e_header)
4 if (field == NULL)
5 {
6     /* 在信息中若不存在 Message-ID: 標頭, 則插入此標頭 */
7 }

```

請注意，`hvalue()` 的第一個引數就是所要搜尋的標頭名稱（第 3 行），該名稱並不含有結尾的冒號。如果你不慎將冒號包含在其中，則會導致搜尋失敗。

`hvalue()` 常式會傳回一個字串，其內容是所搜尋到之標頭的欄位（參見 § 35.3）。如果某個特別的標頭名稱可以出現許多次（例如 `Received:`），則 `hvalue()` 會傳回所找到的第一個標頭。如果你需要處理多次出現的所有標頭，則你得手動進行掃描。

macid()

20.3.5 將文字字串的巨集名稱轉換成整數

巨集、類型名稱與規則集名稱可以是單一字元或多字元的形式。`macid()` 常式可將巨集的名稱從文字字串形式（完全依照字面的形式）轉換成整數。所得的整數稍後可供其他常式使用，例如 `macvalue()` 與 `wordinclass()`。

為了清楚地說明，請考慮 `A` 與 `{foo}` 這兩個名稱：

```

1 int mid;
2
3 mid = macid("A", NULL);
4
5 mid = macid("{foo}", NULL);

```

傳給 `macid()` 的第一個引數必須是個字串，內含需要轉換的名稱。對單一字元的名稱（第 3 行）而言，其名稱會被轉換成第一個字母的字元常數值（character-constant value）（`"A"` 變成 `'A'`）。至於多字元的名稱（第 5 行）一定得放在一對大括號中。該名稱會被轉換成一個唯一的整數，它可能是一個新的值，也可能是先前叫用 `macid()` 所得的值。

在本例中傳給 `macid` 的第二個引數是 `NULL`，因為第一個引數中的字串只含有一個名稱。如果該字串中有其他的文字，你可以傳遞非 `null` 的第二個引數：

```

1  char *str = "localhost is ${lhost}, so there.";
2  char *cp, *ep;
3  int mid;
4
5  if ((cp = strchr(str, '$')) != NULL)
6  {
7      mid = macid( cp+1, &ep );
8  }
```

第二個引數（第 7 行）是一個字元指標的位址。叫用過 `macid()` 後，該字元指標（`ep`）將會含有名稱之後第一個字元的位址，也就是 `str` 中的點號（第 1 行）。

macvalue()

20.3.6

取出一個巨集的值

存放在巨集中的值可以透過 `macvalue()` 常式來取得。例如：

```

1  char *val;
2  int mid;
3
4  val = macvalue( 'A', e);
5
6  mid = macid( "{foo}", NULL);
7  val = macvalue( mid, e);
```

第一個例子（第 4 行）展示了透過 `macvalue()` 取得存放在單一字元巨集名稱（`A`）中的值。第二個例子（第 6 行）則展示了一種技術，對於多字元的名稱以及當你必須處理一個字串中任意的名稱時，它將會很有用。`macid()` 常式會將一個巨集的文字名稱轉換成可供識別之用的整數。然後 `macvalue()` 會搜尋該整數，並將其值傳回（第 7 行）。

如果要尋找的巨集尚未定義，`macvalue()` 會傳回 `NULL`。

->map_lookup()

20.3.7

在外部資料庫中搜尋鍵值

透過宣告外部資料庫而間接建立的常式，可以在該資料庫中搜尋任意的鍵值。舉例而言，考慮一個 dbm 形式的資料庫，其鍵值為登入名稱，而對映值為 ok 或 drop。你會先使用 makemap 程式（參見 § 33.2）處理原始的文字檔，接著你會在組態檔中宣告該資料庫：

```
Kbouncelist dbm -o /etc/bouncelist
```

這個 K 組態命令（參見 § 33.3）會將 bouncelist 這個內部符號名稱提供給資料庫使用。

現在可以使用下面的程式碼片段搜尋想要的值：

```
1  STAB *map;
2  char *p;
3  int  r = 0;
4
5  map = stab("bouncelist", ST_MAP, ST_FIND);
6  if (map == (STAB *)NULL)
7      return (EX_OK);
8  p = (*map->s_map.map_class->map_lookup(&map->s_map, to->q_ruser, NULL, &r);
9  if (p == NULL)
10     return (EX_OK);
11
12  if (strcasecmp(p, "drop") == 0)
13     /* drop the message on the floor */
```

搜尋的動作是藉由呼叫存放在 map_lookup 中的位址來完成的（第 8 行）。第二個引數是要搜尋的鍵值（我們這裡使用了 to->q_ruser，也就是收件人的登入名稱，做為鍵值）。

錯誤碼會放在 r 中傳回。我們在本例中忽略了那些錯誤，但如果你想要的話，你可以使用它們。它們將會對映到 <sys/exit.h> 中所列的錯誤值。但請注意，並非所有的搜尋都會改變 r（參見 map.c 中相關的細節）。

`wordinclass()`

20.3.8

搜尋類型所表列的文字

類型所表列的文字（參見 § 32.1.1）是以符號的形式存放在符號表中（參見 § 32.2.4）。若想檢查一文字字串是否包含在特定的類型中，可由 `wordinclass()` 常式來完成。它可以搜尋單一字元的類型名稱與多字元的類型名稱：

```
1  int mid;
2
3  if (wordinclass( "localhost", 'w') == TRUE)
4      /* 找到了 */
5
6  mid = macid("{foo}", NULL);
7  if (wordinclass( "localhost", mid) == FALSE)
8      /* 沒找到 */
```

傳給 `macid` 的第一個引數是個字串，其中含有想要搜尋的文字。請務必要小心，確定第一個引數是個合法的地址，而且不是 `NULL`，因為 `wordinclass()` 並不會進行這方面的檢查。

第二個引數可以是一個整數型態的字元常數（例如，第 3 行的 `'w'`），或者是一個由 `macid()` 所傳回的整數識別碼（第 7 行）。

