Matthew Broersma, Techworld, 2004.11.23:

"WinAmp blows another security fuse: the third major vulnerability this year

"For those enterprise IT managers who've been eagerly anticipating the next major WinAmp security flaw, the wait is over. Brett Moore of Security-Assessment.com on Tuesday published details of a security hole allowing attackers to take over a PC when a user visits a specially crafted Web page.

"The bug, a boundary error in the 'IN_CDDA.dll' file, is the latest in a string of serious vulnerabilities in WinAmp, including an August flaw in the handling of ‘skin’ files which attackers began to exploit before it had been discovered by researchers. The new bug, the skin file flaw and an April flaw in in the handling of ‘.xm’ files could all be exploited by luring an affected user to a website containing a specific type of file, which would then be automatically downloaded and executed.

“This week’s bug can be exploited in a number of ways, the most dangerous being via an ‘.m3u’ playlist file, according to Moore. ‘When hosted on a website, these files will be automatically downloaded and opened in winamp without any user interaction,’ he wrote in

Security-Assessment.com's advisory. 'This is enough to cause the overflow that would allow a malicious playlist to overwrite EIP and execute arbitrary code.' "

File-editing security holes

User edits a file for an hour:

```
    emacs fancy.c
```

When `emacs` starts,
it copies `fancy.c` into memory.

When `emacs` finishes,
it rewrites `fancy.c`
with the new contents.

Security hole 1:
Attacker fills up disk.
`emacs` truncates `fancy.c`
and fails to rewrite it.

Many programs have these bugs.
Example from `xchat`:

```
    fp=fopen("servlist.conf","w");
    fprintf(fp,...);
```

Fix: `emacs` uses `rename`
for safe rewriting,
so it doesn't lose old version;
and notifies user that
new data is not yet saved.

Security hole 2:
Attacker fills up memory.
Overcommitment forces
`emacs` to die.

Fix: `emacs` tells user
to save to disk frequently,
for example using the
`emacs` auto-save option.

Somewhat annoying but standard
security policy: it's okay
for a few minutes of work
to be lost, if user is notified
that the work has been lost.

World-writable directories

DavFS2 bug fixed 2004.10.30:

```
    f = concat("/tmp",d,".pid",0);
    fp = fopen(f,"w");
```

Impact: Any user can destroy
any file on the system.
Possibly worse effects.

This is a very common problem.

Fix: Replace `/tmp` with `/var/run`.

`/tmp` and `/var/tmp` are directories with permissions 01777.

Any user can create and remove files in these directories:

```
    echo hello > /tmp/test
    cat /tmp/test
    rm /tmp/test
```

These directories are sticky, so a user cannot remove another user's files.

A minor historical side note

The first UNIX security hole
I ever found:

ULTRIX 2.0 failed to check
sticky bit during `rename()`, so
`rename("/tmp/evil","/tmp/victim")`
would replace `/tmp/victim`
owned by another user.
Could easily steal mail etc.

Sysadmin Kevin R. Perry, 1988.10.05:
“My God, you’re right. . . .
Try not to abuse this
for the next couple of days,
until I figure out the fix.”

So what's wrong with DavFS2?

`fopen()` ends up calling

```
open("/tmp/whatever.pid",
  O_WRONLY|O_TRUNC|O_CREAT,0644).
```

Before this happens, attacker guesses name `/tmp/whatever.pid`, and creates `/tmp/whatever.pid` as a **symbolic link**:

```
   ln -s /etc/passwd \
   /tmp/whatever.pid
```

`open()` sees that `/tmp/whatever.pid` links to `/etc/passwd`, so it opens `/etc/passwd`, also truncating `/etc/passwd`.

A different attack: attacker
guesses name `/tmp/whatever.pid`
and creates `/tmp/whatever.pid`
as a world-writable file.

DavFS2 opens that file.
Attacker still owns the file
and changes its contents.
DavFS2 later reads the file
and gets attacker's information.

Partial fix: Use `O_EXCL`,
making `open()` fail if
file already exists
(or if it is a symbolic link).

Attacker can still create file,
making DavFS2 fail to create it.

Fix: Choose random filename;
create with `O_EXCL`;
if that fails, try again
with another random filename.
Use unpredictable source
of random numbers,
such as `/dev/urandom`.

Much better fix:
Never use `/tmp`, `/var/tmp`, etc.
Figure out non-world-writable
locations for all files.

In some languages (e.g., `/bin/sh`), no easy way to use `O_EXCL`.

“Who cares?” programmer says. “I can check for myself whether file already exists. Much easier than giving up on `/tmp`.”

```
while :
do
  x=‘jot -r 1 1 1000000‘
  test -e /tmp/$x || break
done
echo mydata > /tmp/$x
...
rm /tmp/$x
```

`while : do ... done`
is an infinite loop.

`jot -r 1 1 1000000`
prints random 6-digit number.

`` x=`...` `` puts the
random number into `$x`.

`test -e /tmp/$x || break`
quits loop if `/tmp/$x`
does not exist.

e.g. Number is 769488;
`/tmp/769488` does not exist;
script then writes to `/tmp/769488`.

“TOCTOU gap”: the script’s
time of checking `/tmp/769488`
is not the same as the script’s
time of using `/tmp/769488`.
The file may be created
after the check and before the use.

Attacker guesses the number
and creates `/tmp/769488`
as a world-writable file
or a symbolic link.

“Race condition”: the attacker
and the script are in a race.
Race starts when the script
checks `/tmp/769488`;
race ends when file is created.

Another type of race condition:

```
    fopen("mydata","w");
    chmod("mydata",0600);
```

Suppose umask is 022;
most common setting.

`mydata` has mode 0644
until `chmod` changes it to 0600.

Attacker races to open `mydata`
after `fopen`, before `chmod`.
Attacker can then read `mydata`,
including information written later.
Presumably this is unauthorized.