PROGRAMMING GOOFS

THAT WILL HOSE

YOUR SYSTEM SECURITY

(a purely personal viewpoint)

ALEC MUFFETT

http://www.users.dircon.co.uk/~alecm/

Muffett's Observation:

"Frequently the most important or critical applications in a network are run on the least secure machines, due to lack of upgrades/patches, mandated by the very criticality of the application..."

Statements for discussion:

"99.9% of bugs are avoidable"
(sacrifice the remaining 0.1% to Goedel)

"most of these are due to sloppy programming"

"we do not learn the lessons of security,
even with hindsight and in the aftermath
of really major security incidents..."

"amongst the prime causes of this are
commercial O/Ses, legacy apps, and ignorance"

The really irritating thing about computer security:

THE SAME PROBLEMS COME UP

AGAIN AND AGAIN

AND AGAIN AND AGAIN

AND AGAIN AND AGAIN AND AGAIN

The same attacks on networked hosts that were used in the 70s, 80s and early 90s are still in use today

and moreover get conceptually re-used to attack new protocols (gopher, http, ???) in the same way as older ones (smtp, ftp)

WHY?

Because:

- programmers are ignorant when leaving college

- companies can sell widgets better than security to the marketplace

- legacy apps hamper us (try to convince a vendor to drop sendmail)

- legislation ties up technologies that can help (eg: US crypto export)

...AND...

(#pragma personal_cynicism 1)

I strongly suspect that nobody really cares*

(*except for the people who have to clear up the mess)

So what are the problems which keep returning?

- **viruses (not dealt with by me)**
- **stack overwriting**
- **trusting insanitary data**
- **authentication spoofing (direct or indirect)**
- **OVERPOWERFUL SOFTWARE RUNNING WITH EXCESS PRIVILEGE**

...and poor encryption session key generation not covered in this presentation 1st rev.

Viruses

- not really my forte

- possibly the one form of security bug that is more "social" than "erroneous" in nature

- like life: so long as there is exchange of data there will be the possibility that something nasty is piggybacking a ride, inside

Stack Overruns

- blame squarely on the head of the programmer

- can cause:
    - denial of service
    - system crash (at protocol level)
    - hacker infestation

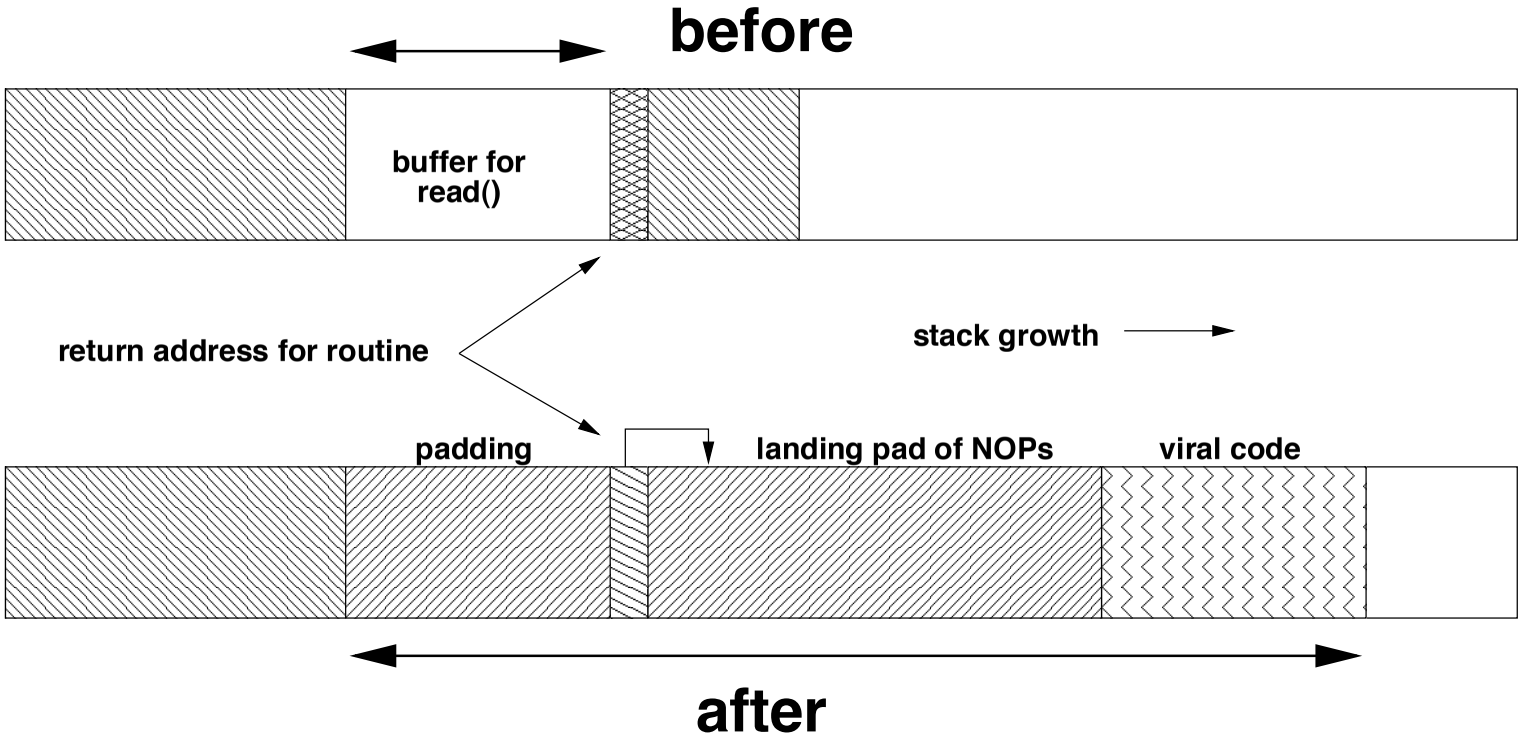
Stack Overruns

- common causes:

    - gets() (Morris Worm)
    - sprintf()
    - strcat()
    - strcpy()
    - insanitary calls to read()

...into small/undersized memory buffers

Stack Overruns



Diagram

Stack Overflows

- require certain creative bent to programming

  - viral payload usually hand-tooled assembler code

  - circumstances may dictate that payload contains no NLs, CRs, NULs, etc... can lead to very creative solutions

  - ...but any moron can execute one that is packaged up adequately.

Stack Overflows

- instances:

  Morris Worm: unbounded gets() on socket

  Sendmail: syslog() routine called strcat()
  on unbounded data read from socket

  Ping: NIS+ host resolver library did sprintf()
  on argv[1] from command line; instant
  SUID hack, no network involved.
  (nb: made more subtle as required DLLs)

Stack Overflows

Probably the most straightforward

of the major holes that

we will be looking at today

Insanitary Data

- Far more subtle class of bugs

- generally due to meddling/trusting
  things that are beyond your control
  in the first case...

- so what *is* under your control?

Under your control?

A good question, nearly metaphysical:

- files/filestore?

- input streams?

- environment variables?

- executable code?

Files under your control?

Maybe, but watch out for:

- user-provided filenames
  direct input or thru env vars
  (PATH, termcap/terminfo, "at")
- fixed filenames
  directory perms, time races in code
  ("ps", "mail", ...)
- filestore perms holding config files
  or parent directories thereof.
  ("chmod 777 /", GID of "/etc")

Environment under your control?

No!

- Do not expect contents of an env var to be sane
- Remember that env vars will propagate to child processes
- Be suspicious of your ability to unset a variable before forking a child

```
PATH=/bin:/usr/bin:...
IFS=/
IFS=/          (multiple instance)
...
```

Environment under your control?

Only sane way to approach env vars:

1) do not trust anything

2) do not propagate anything that you did not create

"everything is forbidden except that which is explicitly permitted"

Input under your control?

No!

- Data servers that are subvertable
  (DNS, NIS, NFS, Kerberos)
- old days: TIOCSTI
- new days: TCP segment injection/spoof
- inbound spams (see further down)

"who knows what's coming down the pipe next?"

Cinderella Attack

- forge (eg:) poorly-authenticated NTP packets.

- use this method to wind the clock on the target host forward to yr 2000-odd

- software licenses for security software on target machine expire

- firewall bastion host turns into pumpkin

- network turns into pumpkin pie.

Code under your control?

Alas, probably not.

- stack overflows/buffer spams
- new dynamism:
  - shared libraries
    (LD_PRELOAD, LC_COLLATE, runpath,
    LD_LIBRARY_PATH, ...)
  - ever since we gave users dl_open()
    or similar...

DON'T TRUST ANYTHING

(and yes, your code really *does* matter, it *is* important to know this)

Alec Muffett

Inbound Record Delimiters

- one of the great, perpetual mistakes

- totally obvious when it is explained, but re-occurs a lot; either programmers forget that the problem exists, or become blithe in their trust of some other service which leaves them open to subversion.

Inbound record delimiters bug, 1970s

IFS variable; field separators define notion of "whitespace", in a shellscript...

IFS=/ ; /bin/ls -> "bin" "ls"

so, create /tmp/bin that does something nasty, and:

```
export PATH=/tmp:$PATH
export IFS=/
suidscriptname # calls /bin/ls, invokes "/tmp/bin"
```

...works for any char, eg: "IFS=n" -> "/bi" "/ls"

Inbound record delimiters bug, 1980s

DNS reverse lookup hostname set to:

```
\nR"l/bin/sed -e ’1,/^$/d’l/bin/sh"\nHxx:
```

Text interpolates into Sendmail’s control file:

```
HReceived-from: HOSTNAME.site.domain
```

becomes:

```
HReceived-from:
R"l/bin/sed -e ’1./^$/d’l/bin/sh"
Hxx: .site.domain
```

...makes bogus recipient record in config,
due to lack of checking for newlines in input.

Viral input bug, 1980s

- Log into NIC to do "whois" query...

@ whois ‘/bin/sh < /dev/tty >/dev/tty 2>&1‘

...escapes from captive environment.

Viral input bug, 1990s

http://site/cgi-bin/foo?%60rm+%2Drf+%2F%60

(‘rm -rf /‘ gets eval’ed by poor CGI script)

...worse still...

http://site/cgi-bin/perl?...

Authentication Spoofing

- What does this mean?

  Broad definition:

  - meddling with an established communications channel

  - forging credentials to lie about who you are

  - cheating an authentication process

Authentication Spoofing

Examples:

- sniffing/guessing reusable passwords
- replaying authentication cookies
  eg: HTML document passwords ==
  b64encode("username:password")
- pre-empting challenge/response schemes
  eg: hijacking S/Key sessions
  (aka: "beat the clock")
- TCP stream hijacking or resetting
  thru forged addresses or sequence nos

TCP/IP
IS NOT FIT FOR USE
AS AN
AUTHENTICATOR

SO WHY DO PEOPLE PERSIST IN USING IT AS IF IT WERE?

By now, you should be able to tell me.

8-)

Spoofing Example

- How many people know that "#"
  is not a legal character in a .rhosts file?

- Tweak DNS:
  #.foo.ac.uk 28800 CNAME host.foo.ac.uk.

  $ ping #
  host.foo.ac.uk is alive

- Go one step further, set "#" as reverse
  A-record, and log into any host with
  a bad .rhosts file...

Spoofing Examples

...but that's HARD compared to just plain lying.

- "+" in hosts.equiv, "my name is 'root'... honest"
- forged "admind" requests from ‘‘localhost’’
- source routed NFS traffic to implement a VPN
- forged TCP RSTs to disconnect sessions
- SYN flooding probably fits this category, too

Excess Privilege

- The BANE of our LIVES
- Problem cuts both ways:
    - not only use of root permissions for programs that do not require them...
    - ...but also excessive promiscuity of data that shouldn’t really be public

Excess Privilege

THE EXAMPLE
"sendmail"

Why run as root?

- TCP port 25 access? Use inetd/fd-passing
- "chown" mailboxes to users? Use groups.
- protect intermediate files? Unix fileperms.
- odds and sods? Use SUID modules.

What is there about a mail daemon that requires root?

Excess Privilege

Data users don't need to see,
and data users don't need to be able to modify.

- **Encrypted ciphertexts**
  (how many years before shadow passwords gained common acceptance?)
- **syslog data, etc...**
- **world writable tty's, /dev/console, etc...**
- **lots of stupid little things, but...**

SECURITY IS HOLISTIC

Irritations of excess privilege:

- perms on "/etc", rwxrwxr-x, uid=root gid=bin
  therefore anyone who can get "bin"
  can get root.

- ownership on older /var/spool dirs =uucp
  therefore anyone who can get "uucp"
  can get root (eg: forge a sendmail queuefile)

- ...and so forth.

Irritations of excess privilege

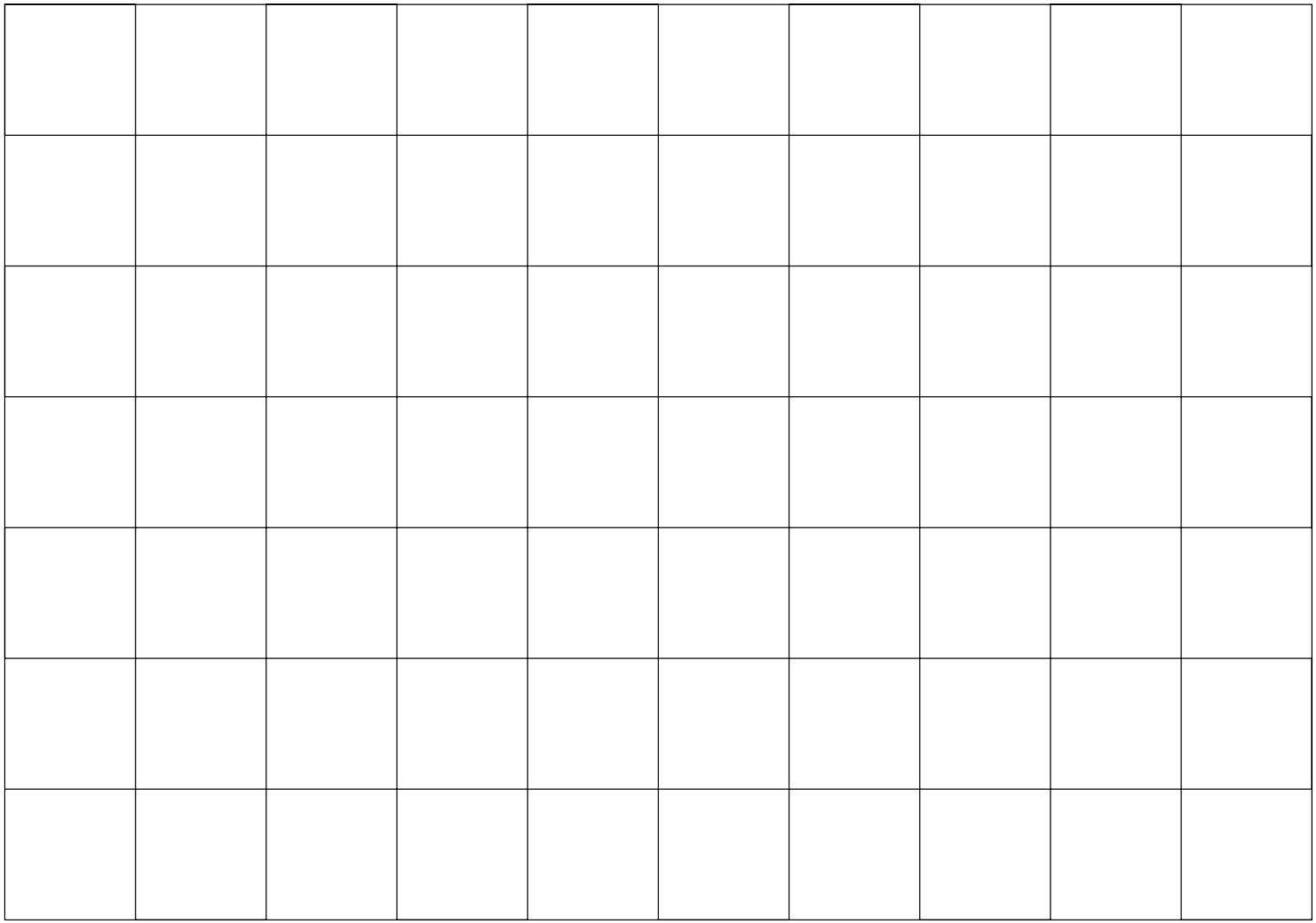
- Attitude amongst O/S designers often is:
  "files executed by root may be owned
  by anyone at all..."

- Attitude should be:
  "As much as possible should be root-owned
  but almost nothing should be root-executed
  since this automatically limits damage..."

The principle of least privilege:

Design your software such that it runs without requiring privileges that are unavailable to normal users.

Try not to f*ck up.



Alec Muffett

Programming Holes