Feb 7, 2005 -- Lecture 9

22C:169
Computer Security

Douglas W. Jones

Department of Computer Science

Viruses

Self Reproducing Code

Simple to write in reflexive languages

eg: assembly language

Difficult in compiled languages (eg C)

```
main(a)
{
   a="main(a){a=%c%s%c;printf(a,34,a,34);}";
   printf(a,34,a,34);
}

#define q(k)main(){puts(#k"\nq("#k")");}
q(#define q(k)main(){puts(#k"\nq("#k")");})
```

Self Reproducing Code

Easier to understand in LISP

```
( ( lambda (x)
    (list x (list (quote quote) x))
  )
  ( quote
    (lambda (x)
            (list x (list (quote quote) x))
) ) )
```

A Virus is:

A self-reproducing code fragment
That attaches itself to other programs
    instead of merely outputting itself

Therefore, it must contain code to
    Search out targets
    Edit targets
        In addition to basic self-reproduction

Successful viruses:

Attach to files likely to be exported

- *MS-Word documents*
- *Games*

Evade notice

- *No obvious side effects*
- *No heavy disk usage*
- *No huge file-size increment*

Antivirus measures

A virus cannot infect a passive document

Think twice before allowing active content in files that don't need it

A virus must be able to act on other files

If active content is supported, limit domain of action

Build firewalls around applications

Build embedded language sandbox

Detect viral code

Does P include code that does X

Generally:

Equivalent to Halting Problem

We rely on approximations

Large catalogs of known viruses

Patterns of "dangerous operations"

Either miss some viruses
or prevent some legitimate operation

Worms

John Bruner's *Shockwave Rider*, 1975
First Implemented, Xerox PARC, 1978

Self reproducing code

Spreads between network hosts
Spread via network links

Requirements

Read from link executes code

Deliberately or not

Deliberate worm

```
# Unix shell script in file f
setenv  host `randomhost`
rcp     f $(host):f
rsh     $(host) f
# insert payload here
rm      f
```

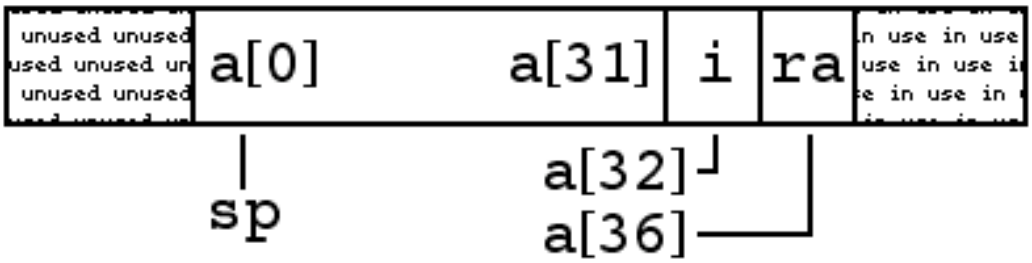
How can worms invade?

Error in network interface that allows
- *injection of code where data intended*
  - Buffer Overflow Attack

Debugging interfaces left in place
- *Beware: Sensible development tools can be dangerous in production*

Buffer Overflow Vulnerability:

```
int f( int i )
{
   char a[32];
   gets( a );
   return lookup(a);
}
```



Buffer Overflow Attack