

# Unix System Administration

**Chris Schenk**

**Lecture 15 – Thursday Mar 08**

**CSCI 4113, Spring 2007**

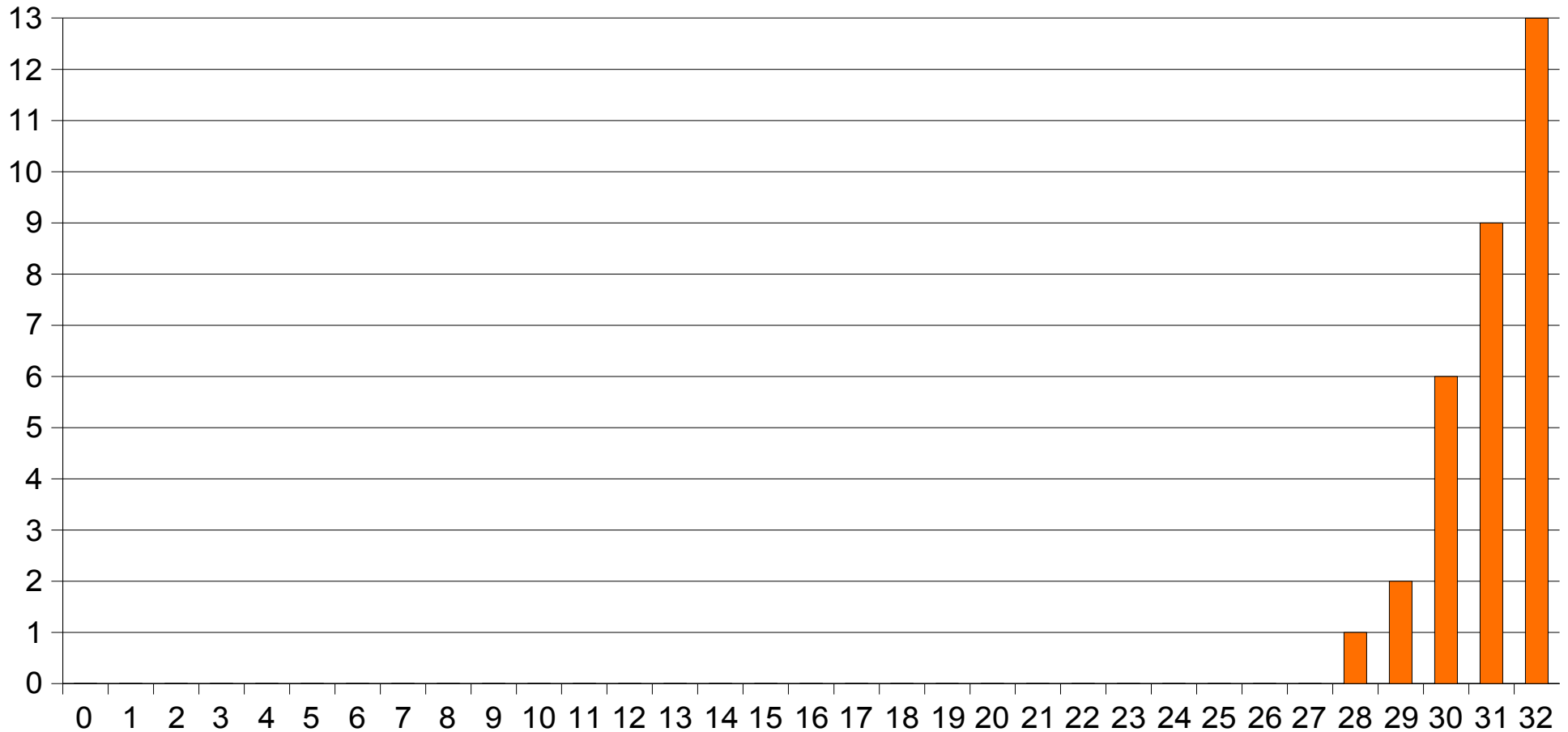
# Logistics

- More BIND config info in lecture 14 slides
  - We didn't get a chance to go through all of them
- DNS Lab05 is involved!
  - Don't start late
- All grades are up to date
  - Stop by if you want to find out how you're doing

# Quiz 01

- Out of 32, Average 31, 96.88%

Quiz 01 Scores



# Web – Apache Web Server

- Apache runs as `httpd` or `apache2` daemon
  - depending on the distro
  - 'apache2' under Ubuntu
- Highly configurable
  - Allow/disallow dynamic content with PHP, Perl, Python, etc
  - Fine-grained permissions and options per directory
  - User-defined web pages from `public_html` dirs
  - SSL support enabled with a module

# Apache – Ubuntu File Locations

- Any of the following locations is configurable
  - I generally like to use the defaults, however
- `/etc/apache2/apache2.conf` – Main config
- `/var/www` – Default location for webserver files
- `/usr/sbin/apache2` – Daemon
- `/var/log/apache2` – Log file location
  - Many different logs exist in this directory
- `/etc/init.d/apache2` – Startup script

# Apache – Server Directives

- Tons of different options, we'll cover important ones
  - `ServerRoot` – Location of configs/modules
  - `DocumentRoot` – Location of webserver files
  - `Listen <port>` – Bind port for apache (80)
  - `LoadModule` – Loads extra useful modules
  - `DirectoryIndex` – Default files to load in a directory (index.html, index.php, etc)
  - `User/Group` – What users to run as non-privileged
  - `ServerAdmin` – Contact email address (usually webmaster)

# Apache – Directory Permissions

- Set with the `Directory` directive
- ```
<Directory />  
    Options None                #Turn all features off  
    AllowOverride none          #disable .htaccess  
    Order allow,deny  
    Deny from all               #deny access to files  
</Directory>
```
- The above disables all options and denies all access to /
  - This is root on the FILESYSTEM, not the webserver docroot!
  - Useful to set restrictive permissions on a site

# Apache – Directory Permissions (cont)

- Now we can set the permissions on the document root
- ```
<Directory /var/www>  
    #set default options  
    Options Indexes SymLinksIfOwnerMatch  
    #allow .htaccess to override default options  
    AllowOverride All  
    Order allow,deny  
    Allow from all  
</Directory>
```
- This allows the site to function for files under `/var/www`
  - Because of the `Allow from all` directive

# SSH – Secure Shell

- Built on asymmetric & symmetric key encryption
- Provides secure communication
- Designed to replace insecure tools under unix
  - ssh (telnet, rsh), scp , sftp (ftp)
- Other utilities included
  - ssh-agent, ssh-keysign, ssh-keygen, and others
- Tunneling for other protocols, X windows
  - Provides encryption for other programs

# SSH – RSA Keys

- Asymmetric key relationship
  - Built on premise - factoring large numbers is difficult
- RSA provides an initial communication
  - Which is VERY slow compared to symmetric keys
  - Allows a secure transfer of symmetric keys
- Server always provides the public key for communication
  - “RSA key fingerprint is XX:XX...  
Are you sure you want to continue connecting  
(yes/no)?”

# SSH – RSA Keys (cont)

- Keys split into 'public' and 'private' halves
  - Each half is used for communication in a specific direction
- A message encrypted with the 'public' key can only be decrypted with the 'private' key
  - and vice-versa
- SSH server uses the 'private' key to send data to client
  - Client decodes with the public key originally presented

# SSH – Symmetric Keys

- Asymmetric keys are butt slow
  - Symmetric keys takeover after initial RSA exchange
  - Encryption algorithms (block ciphers) use a single shared key
- Key size depends on algorithm used
  - AES supports 128, 192, 256 bit key lengths
- Block size is the max length of input to the block cipher
  - Messages are split into parts equal to the block size
  - This is NOT the same as key size!

# SSH – Large Numbers

- Remember each bit doubles the keyspace
- Years till next ice age  $2^{14}$
- Age of the universe  $2^{34}$
- Number of atoms in the planet  $2^{170}$
- Number of atoms in the sun  $2^{190}$
- Number of atoms in the galaxy  $2^{223}$
- Number of atoms in the universe  $2^{265}$
- Number of bits in an RSA key  $2^{1024}$

# SSH – Block Ciphers

- A Block Cipher is an encryption algorithm
  - Takes a fixed sized input
  - Produces an output of EQUAL size to the input
- A “good” block cipher creates output that is seemingly indistinguishable from random
  - Output is based on the **input and the key**
  - What does “good” mean?
- Many different block ciphers available
  - 3DES, **AES**, Blowfish, and others

# SSH – Modes of Operation

- A block cipher must be reversible
  - Have to get the message back somehow!
  - encryption/decryption uses the shared key
  - The input message *ALWAYS* produces the same ciphertext
- A “mode of operation” is needed for security
  - Provides a way of further randomizing the ciphertext
  - Prevents frequency pattern attacks
- Common modes – CBC (cipher block chaining) and CTR (counter)

# SSH – sshd Configuration

- Usually found under `/etc/ssh/sshd_config`
  - Sometimes `/etc/openssh/sshd_config`
  - Server keys also found in this location
- Many different options for how the server behaves
  - Disallow root access, Disallow empty passwords
  - Allow only client ssh keypairs (no password authentication)
  - Set a login banner
  - And more

# SSH – ssh-keygen

- Create an SSH keypair
- % `ssh-keygen -t <type> [-f filename] [-b bits]`
  - `ssh-keygen -t rsa`
  - If options are omitted you may be prompted
  - Default keysize is 1024 bits (2<sup>1024</sup>)
- Optionally you can enter a passphrase
  - Stronger protection of your logins, highly recommended
- View a key's fingerprint (very useful)
  - `ssh-keygen -l ~/.ssh/id_rsa.pub`

# SSH – ssh-agent

- Keeps track of keypairs in memory
  - You open your ssh keys once at the beginning with passphrase
  - All passphrases tracked thereafter for the duration of ssh-agent
  - Useful if you ssh to one machine more than once
    - And want to keep access very secure!
- ssh-agent runs in the background
  - Any new shells you spawn are tracked by ssh-agent
    - May not be true for older versions

# SSL – Secure Sockets Layer

- Originally developed by Netscape
- Uses public-key cryptography to transmit data
  - Originally RSA was not included in SSL because of patent laws
  - Non-commercial use or full blown licenses for RSA
  - RSA patent expired in 2000, now included in OpenSSL
- Trust established with SSL Certificates
  - Trust is managed by a Certificate Authority (CA)
  - Solves a huge scalability problem with public keypairs

# Public Keys – The Scaling Problem

- We use public keys to talk securely with servers
- 'First try risk' exists with accepting a public key
  - Seen in SSH, but would also be true with browsers
    - Without VALID certificates, that is
- To avoid the 'first try risk' problem, we can include public keys in all web browsers
  - This is impossible with the number of servers on the web
  - We need a way to allow our public key to be accepted
    - And still maintain trust with the client

# SSL Certificates

- Enter certificates to solve the scaling problem
  - Embed a small(er) set of public keys in browsers
    - Instead of every web server's public key embedded in the browser (which would be hundreds of thousands)
  - An authority deems set of public keys to be trusted
- Each site manages their own certificate that is signed by the CA
  - The client checks if the CA has signed the key
  - If signature is valid, accept the public key for use

# OpenSSL – Command Line Tools

- Curiously enough, it's called `openssl`
  - Tons of different commands available in the library
- Used to do a number of things
  - Encrypt and decrypt files with a specific algorithm
    - e.g. Use RSA or DES3 to manage private data
  - Calculate cryptographic checksums
  - Create and manipulate X.509 certificates
    - This is the primary use of OpenSSL for the web

# OpenSSL – RSA Key Generation

- Use 'genrsa' option in openssl to create keys
  - schenkc@schenktop:~\$ **openssl genrsa -aes128 -out chris-priv.pem 1024**  
Generating RSA private key, 1024 bit long modulus  
.....++++++  
.....++++++  
e is 65537 (0x10001)  
Enter pass phrase for chris-priv.pem:  
Verifying - Enter pass phrase for chris-priv.pem:  
schenkc@schenktop:~\$
- You must have a block cipher to privatize key!
  - Otherwise the key is left completely unprotected
  - We use aes128 in the above example

# OpenSSL – Key File

- What does our secret key look like?

3 B G3IN R RI T K Y  
3 T 4 NCRY T D  
D K Inf : 128 CBC,974F4776 18993 47368C14F3D B4D38

6m8D kM B 9IDg mQT v MFT CKUF LmCTI5L55B8b=F7=Jl4 b/2K  
O MMl kL On8.5 =F1 vm3/+f kmE C9j bu /VT9j DKh/u=NMCv+ L  
Oiu=g Hj5Q JnK u v J Q0j HM T mFF5D G/F+HOMD GgM+I C B  
+O72K Ulv f Gh=J 6D 6 mvG bCi3f6 5IT2 o U Ou v CROg9 Tliuf  
3D=/DR7Dk =UIbnv 8 DFv GI/LUU mY 7 3FIH46GRk6 mN1u B  
UYfT m i21 //5J7 f 2+f Cf00gGUF nB hkium u g Bgg 3F7b f  
u6B7 h m+g OT Y YT5 hQ=MT0LF mKKiO GFui 2nYG IC+Rjf DYBLf  
B GuM DDQ TY C 1 M 7C 105 2ufRvM I 7 +3B KQ 7 H k2+ G6u /BGG  
Ilg U 4 Nm h7 N9j OIR k/7 H7 7 gO9n OYBD0 9IU Q HL5 j /7 5  
h f /G Dn U lhdDUL l 1 C1F0NH4 / Omjm5l4 b3kgJ+ L 7 R 1  
nvyh+ugFj 3IL+ DuM5 299 k /vCm G6k7 i5l2gg+=1 7 91Ygm b2  
E4 DT 8uYY +b F D 3 M h=0 1u8 0ni4mDC 4lQ9 H O0CD 1 5 Kfl  
2lGI L hn hKl2R BII89JC u u GGDfkD+G7i nNT74L UYM On Y 424Q  
ND R RI T K Y

- Not very readable, is it?

# OpenSSL – Key File (cont)

- To read our key, we need to run openssl again

```
3  @ :~$ openssl rsa -in chris-priv.pem -text -noout
-----
Key: (1024 bit)
-----
m7  ulu  :
00:f0:5b:90:9:3:97:5:6b:03:8b:b:87:63:2:
3  f:13:3 :87:1:3 :5f:2:51:20:b :5:82:2: ...
3  ubli  3  n n 65537 (010001)
iv  7  n n
0 :f :6:8:8:34:f2:8 :0b:37:b:31:63:6f:38:
3  f9:97:4:05:2:8f:1 b:57:4f:34:70:20:f:7 : ...
3  m 1: ...
3  m 2: ...
3  3 7  n n : ...
3  7  n n 2: ...
3  7  fff i n : ...
```

- All fields here are used in RSA encryption/decryption

# OpenSSL – CSR: Certificate Request

- Certificate not valid until signed by an authority
  - Usually a company (Verisign) will do this for cash
    - Large number of CAs around to take your money
- To obtain a certificate, first create a 'request'
  - Has your name, email, other info, your public key and signature
  - CA will sign this request if properly formatted and your information matches who you are
  - The CA supposedly checks to see if you are a valid business
    - And checks periodically as well

# OpenSSL – Creating a CSR

- You use your private key to create the request

```
chris@chris:~$ openssl req -key chris-priv.pem -new -out chris-req.pem
```

...

```
C=un N m (2 l U :US  
vin N m (full n m) :Colorado  
L=li N m (g, i) :Boulder  
O=gn i n N m (g, m n) In i gi L :University of Colorado  
O=gn i n I Uni N m (g, n) :Department of Computer Science  
C=mm n N m (g, YOUR n m) :schenktop.cs.colorado.edu  
m il :Christopher.Schenk@Colorado.EDU
```

```
chris@chris:~$
```

# OpenSSL – Viewing a CSR

- Very similar to viewing your key

```
chank@chank:~$ openssl req -in chris-req.pem -text -noout
Certificate Request
Data:
  version: 0 (0x0)
  subject: C=UK, T=City, L=Building, O=University of City, OU=Department
  CN=chank, email=chank@city.uk
  subjectPublicKeyInfo:
    publicKey: (1024 bit)
    modulus: (1024 bit)
      00:f0:5b:90:93:97:56:b0:38:b1:87:63:2:
      f:13:87:1:3:5f:2:51:20:b5:82:2: ...
  signature:
    algorithm: sha1withRSA
      3b:07:65:b9:96:b3:b35:59:42:f1:51:7b: ...
```

# OpenSSL - CSRs

- A CSR is self-signed (signed by you). Why?
  - Ensures that the CSR author (you) has the private key corresponding to the public key in the CSR
    - If this wasn't enforced, I could grab anyone else's public key to use
- Why does the CA sign your public key?
  - Well, this is the whole point
  - The authority will only sign your key if you're trustworthy
    - For credit card handling, personal data, etc
- What happens if a CA does not sign your key?

# OpenSSL – Signed Certificates

- What do we have so far?
  - An X.509 certificate signed by the CA
    - Contains our public key, name, email, and other info
  - A private key in a password-protected file
    - Don't lose the password or your certificate may become useless
- What else do we need?
  - We need to be able to verify the CA's signature on the cert
    - Which means we need the CA's verification key

# OpenSSL – CA Verification Key

- CA's verification key IS a certificate
  - CA generates a self-signed 'root' certificate
  - Certificate has the verification key (aka public key)
  - This certificate is embedded in your web browser!
  - Used to validate public keys from other sources
- Root certificate is freely distributed to everyone else
  - You can add in other CA certificates to your browser if you want
    - “Unable to verify certificate...do you want to accept?”