CS3235
Eighth set of lecture slides

Hugh Anderson

National University of Singapore
School of Computing

October, 2007
Quantum effects...

Where am I...?
Or what is my momentum...?
Or where am I...?

Oh hell...! Why worry about all that again...? I'm not even sure if I'm a wave or a particle!

Photon self-identity problems.
Outline

1 Random stuff
   - Different amounts of change

2 Quantum effects
   - Quantum computing
   - Quantum cryptography

3 Case studies - password security
   - Unix password systems
   - Microsoft password security
Changing/Flipping 0% of the bits

Each bit has been selected and changed 0% of the time...
Randomly Changing/Flipping 25% of the bits

Each bit has been selected and changed 25% of the time...
Randomly Changing/Flipping 50% of the bits

Each bit has been selected and changed on the flip of a coin.
Changing/Flipping 100% of the bits

Each bit has been selected and changed...
Changing/Flipping every second bit (50% changed)...

Every second bit has been selected and changed...
First interest: Quantum computing...

1. Quantum *computers* may be able to compute **HARD** problems quickly (such as factorizing large composites).

How? The underlying data elements are quantum bits (qubits), not limited to just 0,1 states - instead considered to be a **superposition** of states. An **operation** performed on a qubit is **performed** on all the states simultaneously. See Shor’s algorithm...

It is likely that no effective quantum computer has yet been built.
Quantum cryptography

Second interest: Quantum cryptography

2. Quantum *cryptography* uses laws of quantum mechanics - Heisenberg Uncertainty applies to some pairs (of properties) of (atomic) particles. Measuring one property affects another.

A snooper is easily detected, and there are various protocols for using quantum effects to share keys.

Alice randomly chooses one of four polarizations: rectilinear: 0, 90, or diagonal: 45, 135 (degrees).
Alice transmits 10000 photons

Encoded using different polarizations:

![Diagram showing quantum cryptography](image)
Alice and Bob’s protocol

A protocol to choose bits - no reveal...

1. Alice records what she has sent. Bob randomly chooses polarizations, and for each one reads the resultant value. (If he chooses correctly gets a valid 1 or 0, if not gets a random value)

2. Bob tells Alice the polarizations he has used: diag, diag, rectilinear, diag...

3. Alice replies by telling Bob which ones were correct. (1, 3, 4, 8, 9, 10, 12, 17...

4. They now have 5000 (approx) bits in common.
Harry the hacker listens in, but...

Harry has a problem

1. If **Harry** the hacker senses (some of) the photons, he must choose which polarization to use, and **will affect the photon**.

2. **Bob and Alice compare** a subset of the bits that they think they know to **detect snooping**.

3. If no snooping, then rest of bits are likely to be OK.

Quantum cryptography systems are now commercially available, operating over reasonably long (40km) fibre. Note the probabilistic nature of the algorithm. By choosing bit length can get any degree of assurance.
Password security

UNIX systems are traditionally open systems, given their background in university environments. The security on UNIX accounts is often minimal. It is common for UNIX accounts to be made available relatively freely.

- Morris and Thompson article:
  
  http://citeseer.ist.psu.edu/morris79password.html

- Computer generated passwords more predictable than user ones.
- At the MIT Media lab\(^a\) all computers have been password-free until recently...

\(^a\)MIT - home of Kerberos!
UNIX password security

Vulnerabilities

- UNIX systems are **vulnerable** to a wide range of attacks, particularly internal attacks.
- All Unix systems have a **root** account.
- This account has a UID and GID of zero, and once **root access** is obtained on a UNIX system, there is very little that **cannot** be done.
- Account passwords are constructed to meet the following requirements:
  - Each password has **at least six** characters.
  - Only the **first eight**\(^a\) characters are **significant**.

\(^a\)Why eight???
UNIX accounts

Other accounts...
There are many other accounts found on Unix systems, not just those for clients:

- `sysadm` - A System V administration account, and
- `daemon` - A daemon process account, and
- `uucp` - The UUCP owner, and
- `lp` - The print spooler owner.

When protecting a UNIX system, we must protect all these accounts - not just `root`. 
The password file

- Account information is kept in a file called /etc/passwd.
- It normally consists of seven colon-delimited fields, and may look like the following:

```
hugo:aAbBcJJJx2:501:100:My Account:/home/hugo:/bin/csh
```

hugo: The account or user name.
aAbBcJJJx2: A one-way encrypted (hashed) password
501: The UID - unique user number
100: The GID - group number for user.
My Account: Account information.
/home/hugo: The account’s home directory
/bin/csh: A program to run when you log in
Configuration in text files...

Text files for configuration... good thing or bad thing?
A system administrator on the CTSS system at MIT was editing the password file and another system administrator was editing the daily message that is printed on everyone’s terminal on login.
Due to a software design error, the temporary editor files of the two users were interchanged and thus, for a time, the password file was printed on every terminal when it was logged in.

(Robert Morris and Ken Thompson, Password Security: A Case History)
UNIX passwords

When you log in with your account name and password

- the password is encrypted and the resulting hash is compared to the hash stored in the password file.
- If they are equal, the system accepts that you’ve typed in the correct password and grants you access.
- UNIX originally used a DES-like algorithm to calculate the encrypted password. (Now use MD5...)
  - The password is used as the DES key (eight 7-bit characters make a 56 bit DES key) to encrypt a block of binary zeroes.
  - The result of this encryption is the hash value.

Note: the password is not encrypted, it is the key used to perform the encryption!
UNIX salt

UNIX introduces two random characters...

- Two equal passwords result in different hashes.
- From viewing the UNIX password file you can not tell if two persons have the same password.
- To prevent crackers from simply encrypting an entire dictionary and then looking up the hash, the salt was added to the algorithm to create a possible 4096 different hashes for a particular password.
- This lengthens the cracking time because it becomes a little harder to store an encrypted dictionary online as the encrypted dictionary now would have to take up 4096 times the disk space.
- Password cracking not harder, just more time consuming.
Sample crypt code from LINUX uClibc
The code has the following structure:

```c
extern char * crypt(const char *key, const char *salt) {
    /* Are we using the MD5 replacement */
    /* instead of DES... */
    if (salt[0]=='$' && salt[1]=='1' && salt[2]=='$')
        return __md5_crypt(key, salt);
    else
        return __des_crypt(key, salt);
}
```
Cracking

Brute force cracking...

- It is very time consuming, but given enough time, brute force cracking will get the password.
- The hashed passwords are compared with the entry in the /etc/passwd file.
- BTW - You cannot try to log in using all the possible passwords, as UNIX systems enforce 10 second timeouts after three consecutive login failures.
Dictionary cracking

Dictionary password cracking is the most popular method for cracking Unix passwords

- The cracking program will take a word list, and one at a time try to crack one or all of the passwords listed in the password file.
- Some password crackers will filter and/or mutate:
  - substitute numbers for certain letters,
  - add prefixes or suffixes,
  - or switch case or order of letters.
Dictionary cracking

Crack software

- A popular cracking utility is called Crack.
- Crack can use user-definable rules for word manipulation or mutation to maximize dictionary effectiveness.
- Crack merges dictionaries, turns the password files into a sorted list, and generates lists of possible passwords from the merged dictionary or from information gleaned about users from the password file.
Solution is to not reveal the encrypted passwords

Once the password hashes are moved to the shadow file, its permissions are changed as follows:

```
opo 35# ls -l /etc/shadow
-r-------- 1 root sys 3429 Aug 20 14:46 /etc/shadow
opo 36#
```

These permissions ensure that ordinary users are unable to look at the password hashes, and hence are unable to try dictionary attacks.
Microsoft password security

Windows carries history as well

Two one-way password hashes are stored on NT systems:

- a **LanManager** hash, and
- a **Windows NT** hash.

The LanManager hash supports the older LanManager protocol originally used in Windows and OS/2. In an all-NT environment it is desirable to turn off LanManager passwords, as it is easier to crack. The NT method uses a **stronger algorithm** and allows mixed-cased passwords.
Microsoft password security

Database for hashes...

- The database containing these hashes on an NT system is called the **SAM** (Security Access Manager)
- If you have administrative access\(^a\), the program `pwdump` can extract the hashes.
- Microsoft does not **salt** during hash generation, so once a potential password has generated a hash it can be checked against **all** accounts.
- The cracking software takes advantage of this.

\(^a\)Originally, *anyone* could extract the hashed passwords from the SAM, as Microsoft believed that “if they didn’t tell anyone the algorithms they used, no-one could discover what they had done”. Security through obscurity is not a safe strategy, and Jeremy Allison was able to de-obfuscate the SAM entries relatively quickly.
LanManager/Windows NT encryption

The hashed passwords

- LanManager encryption is created by taking the user's plaintext password, capitalising it, and either truncating to 14 bytes, or padding to 14 bytes with null bytes.
- This 14 byte value is used as two 56-bit DES keys to encrypt an eight byte value, forming a 16 byte value which is stored by the server and client.
- This value is known as the **hashed password**.
- Windows NT encryption is a higher quality mechanism, consisting of doing an **MD4 hash** on a Unicode version of the user's password.
- This also produces a **16 byte hash value** that is non-reversible.
NT Password security

Is it safe?

- Note that the LANManager hash is similar to UNIX level of cryptography
- The NT hash is better
- But... neither use strong encryption, and
- the network login mechanism has some problems.
Challenge response

Challenge response protocol used for password checking...

![Diagram of challenge response protocol](image)
Challenge-response protocol

When a client wishes to use a resource...

- ... it requests a connection and negotiates the protocol.
- In the reply to this request the server generates and appends an 8 byte, random value - this is stored in the server after the reply is sent and is known as the challenge - different for every client connection.
- The client then uses the hashed password (16 byte values described above), appended with 5 null bytes, as three 56 bit DES keys, to encrypt the challenge 8 byte value, forming a 24 byte value known as the response.
- This calculation is done on both hashes of the user’s password, and both responses are returned to the server, giving two 24 byte values.
The server then...

- ... reproduces the above calculation, using its own value of the 16 byte hashed password and the challenge value that it kept during the initial protocol negotiation.
- It then checks to see if the 24 byte value it calculates matches the 24 byte value returned to it from the client.
- If these values match exactly, then the client knew the correct password and is allowed access.
Challenge-response protocol

There are good points about this:

- The server never knows or stores the cleartext of the users password - just the 16 byte hashed values derived from it.
- The cleartext password or 16 byte hashed values are never transmitted over the network - thus increasing security.
Challenge-response protocol

However, there is also a bad side:

- The 16 byte hashed values are a "password equivalent". You cannot derive the users password from them, but they can be used in a modified client to gain access to a server.

- The initial protocol negotiation is generally insecure, and can be hijacked in a range of ways. One common hijack involves convincing the server to allow clear-text passwords.

- Despite functionality added to NT to protect unauthorized access to the SAM, the mechanism is trivially insecure.

- Both the hashed values can be retrieved using the network sniffer mentioned before, and they are as-good-as passwords.
Attacks on Windows (NT) systems rely on flawed mechanism...

- Even *without* network access, it is possible by various means to access the SAM password hashes, and *with* network access it is easy.

- The hashed values are password equivalents, and may be used directly if you have modified client software.

- The attack considered here is the use of either a dictionary, or brute force attack directly on the password hashes (which must be first collected somehow).
Attack

A tool

**L0phtCrack** is a tool for turning Microsoft Lan Manager and NT password hashes back into the original clear text passwords. It may be configured to run in different ways.

**Dictionary cracking:** L0phtCrack running on a Pentium Pro 200 checked a password file with 100 passwords against a 8 Megabyte (about 1,000,000 word) dictionary file in under one minute.

**Brute force:** L0phtCrack running on a Pentium Pro 200 checked a password file with 10 passwords using the alpha character set (A-Z) in 26 hours.
**Attack time**

Comparison of attack times...

<table>
<thead>
<tr>
<th>Character set size</th>
<th>Size of computation</th>
<th>Relative time taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>$8.353 \times 10^9$</td>
<td>1.00</td>
</tr>
<tr>
<td>36</td>
<td>$8.060 \times 10^{10}$</td>
<td>9.65</td>
</tr>
<tr>
<td>46</td>
<td>$4.455 \times 10^{11}$</td>
<td>53.33</td>
</tr>
<tr>
<td>68</td>
<td>$6.823 \times 10^{12}$</td>
<td>816.86</td>
</tr>
</tbody>
</table>

So if 26 characters takes 26 hours to complete, a worst-case scenario for 36 characters (A-Z,0-9) would take 250 hours or 10.5 days. A password such as `take2asp1r1n` would probably be computed in about 7 days.
Microsoft base security fix

How to minimize risk...

1. **Disable** the use of **LanManager** passwords.
2. Don’t log in over network as administrator
3. **Encrypt** all network traffic
4. Use **long** passwords, and all allowable characters
5. Use an **alternative** login system
6. Use an **unsniffable network** cabling system.