CSE 127
Computer Security

Fall 2015

Lecture #6
User Authentication

Stefan Savage
How do you authenticate a human?

So far we’ve talked about authenticating computers… but what about people?

How do you know I am who I say I am?
- If you haven’t met me?
- If you already know me?
Key issue today

How do you authenticate a human to a computer?

What evidence can you provide that you are who you say you are?

- Something you have
- Something you are
- Something you know
Text Passwords

Shared code/phrase
Client sends to authenticate
Simple, right?

How do you...

- Establish them to begin with?
- Make them easy to remember?
- Make them hard to guess?
- Keep them from leaking/being stolen?
Prime Mover Issues

How to establish first password?
- Piggyback on existing identifier
  - E-mail address (e.g. hotmail), Triton card
  - But where does the chain stop?
    Triton Card -> drivers license -> birth certificate -> ?

How to communicate password
- Default to existing identifier (e.g. SSN)
- Out of band (e.g., mail)
- In person

Who selects password
- User vs service provider?
Password choice issues?

Easy to memorize?
Easy to enter?
Hard to guess?
Updated frequently?

How hard can this be?
Why is there a book on this?!?
Aside: classic usability paper

Adams and Sasse, “Users are not the enemy”, CACM 49(12), Dec 1999

Key points

- Users lack security knowledge
- Users will circumvent restrictions that get in their way (security is secondary task)
- Doing this lowers respect for security mechanisms; repeat
Unintended consequences of reasonable ideas

Different password for each account/principal
- Users use single password

Require long passwords
- Users add padding

Require random passwords
- Users write down passwords

Use letters and numbers
- Users add digit (“1”) to password

Change passwords regularly
- Users append month to password

“Reset password” functionality
- Used regularly as primary means of login
How hard to just guess repeatedly?

Username design
- The "no such user" mistake
- The "here's who we are" mistake
- Last names

Passwords
- Common words, phrases for passwords
- Null passwords, "password", username, backwards, etc.
- Dictionary attacks
No constraints on choice of password

- 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

Modern cracker: http://www.openwall.com/john/

Passwords typically had lifetime > 30 days

Heuristics
- Letter substitutions (e>3, l>1), words backwards, common names, patterns, etc.

Recent studies (2012)
- Only 20 bits of entropy against dictionary attacks
Leaks

Social engineering
- Targeted: “Hello, this is ACS… we need to you login to foo.bar.com and change your password”
  Mass target: Phishing

Shared passwords
- Perhaps not great idea

Writing the password down on paper
- Why? Where?

Technical attacks
- Fool the user (fake login program)
- Password files
- Passwords in memory, in transit
Fake authentication programs

Easy to make screen that looks legitimate
Pass login info to real login program

Windows solution
- Special key sequence (CTRL-ALT-DEL)
- Can’t be trapped
- Launches real login program

Phishing attacks

Real or fake?
Case Study:
Tenex password scheme

Password stored in the clear (only readable by OS)
OS validates password as follows:

```c
CheckPasswd (char *InputPassword) {
    for (i=0; i <= strlen(RealPassword); i++) {
        if (RealPassword[i] != InputPassword[i]) {
            return -1;
        }
    }
    return 0;
}
```

Problem?
Breaking Tenex in Linear Time

Recall that virtual memory is broken into pages (e.g. 4kbytes each on x86)

Put first character of InputPassword on the last byte of a VM Page

Arrange for next page to be unmapped
  ◆ Unallocated, protected, whatever…

Pick a letter for the first character then call CheckPassword
  ◆ If you get a page fault then you picked the right letter, else pick a different letter

Repeat for the second character…
Passwords in transit

Clear-text passwords

Keystroke logger

API hooker
  * HTTPSendRequest(*)
Passwords in memory
[Chow et al, 04,05]

When you login your password is stored in a buffer, which may be copied and persist for some time

Question: how many copies of your plaintext password are out

Methodology

- Capture buffer that holds password (e.g. login screen, ssh)
- Instrument system to track all places “tainted” by that data (i.e. copies of password)

Example: you login to Hotmail via Firefox

- Copies of password data in
  - X windows event queue
  - Socket buffer (for keystrokes)
  - Strings classes (long lifetime)
  - Kernel tty/terminal buffers
  - Etc…
- Can potentially last for hours/day
Case study: Unix password scheme

Store hash of password
- Encrypt string of 0’s repeatedly with password as key [crypt(3): DES]
- Password file (/etc/passwd) world readable

Problem: identical passwords...

Problem: offline dictionary attack
- Grab password file and take it home
- Compute hashes for all likely passwords (e.g. all dictionary words)
- Check if any match the password file
Solution: 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.
- Also makes dictionary attacks more expensive.

12 bit salt results in 4,096 versions of each password.

/etc/passwd entry:

```
user_id | salt_u | Hash(salt_u + passwd_u) | ...
```

How to decide if this is a good solution?
Well... it depends

How much time is enough?

- VAX 11/780 could do crypt(3) 3.6 times/sec
- 2.4Ghz Pentium 250,000/sec, modern proc close to 1M/sec
- GPUs even faster (B’s/sec on simpler hashes)
- EFF DES Hardware could potentially do \(88 \times 10^9\)/sec
- Distributed offline attacks
- SDSC has a disk with the crypt(3) hashes for 50M common passwords

Key point: difficulty can’t be statically set, needs to be scaled over time

Second point: don’t give people access to the password file

- Most modern implementations use so-called shadow password files (/etc/shadow) that aren’t world readable.
- Tricky… what about backup?
Case Study: Windows passwords

Windows 95/98: LanMan hash
- Split password into two 7 character strings, pad out and convert to uppercase
- Use each half for keyed hash, concatenate hashes

Problem?
- Much easier to crack two 7 character passwords than one 14 character password (39B times easier)
- Conversion to uppercase makes it even easier
- Google Lophtcrack sometime… (5.3M/sec)
  » Also Oeschslin’s Crypto 2003 paper on time/memory tradeoffs

Windows NT/2000/XP
- MD4 hash across entire password (case sensitive)
- No salt
Misc practical issues

Unchanged default password
- E.g. Linksys -> admin
No default password
No way to reset password
Retry counters

Terminal shell v1.0
Cayman-DSL Model 3220-H, DMT-ADSL (Alcatel) plus 4-port hub
Running GatorSurf version 5.6.1 (build R0)
(d completed login: user level)

Cayman-DSL1663970>
Password reset

In-person request
Send new password via e-mail and force change

issues?

Associated challenges

- What is your mother’s maiden name?
- What is your favorite color?
- Who is your favorite football team?
- Choose your own question…
- Issues?
Graphical passwords
[Jermyn et al, 99]

User draws figure for input
Use stroke order, pen up/down direction & position to identify
Large space of memorable figures
  - But people don’t choose from whole space

Potentially easier to observe
Recognition-based graphical passwords

Déjà vu (Perrig & Dhamija ’00)
- User image portfolio
- Challenge set of images
- User picks subset in portfolio

Passfaces
- 8 decoys, one passface in random positions in grid
- Users selects face and process repeats
Passwords are imperfect...
Where does this lead us?

Turn humans into machines?
◆ We want long, random, non-language passwords for security
◆ But we can't remember them

Something you have
◆ One time passwords
◆ Physical tokens

Something you are
◆ Biometric authentication
One Time Passwords

Shared lists

Sequentially updated

One-time password sequences based on a one-way (hash) function
Hash-chain 1-time Passwords

Alice identifies herself to verifier Bob 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 Bob
  \[ H(H(H(...(H(w))...))) \]
  \[ \text{t times} \]
One-time passwords: $i^{th}$ authentication

Alice

- Compute $w' = H(t-i)(w)$ and transmit the value to Bob
- Bob 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 Alice (as part of session $i-1$)
- Bob saves $w'$ and $i$ for use in the next session
Why This 1-time Password Works

It’s hard to compute x from H(x) – one-way
- Even though attacker gets to see H^{(t-i)}(x), they can’t guess the next message H^{(t-(i+1))}(x)

Used in practice: e.g. SKey
Also, some cryptographic tokens (i.e. cards)

Vulnerable to man-in-the-middle attack
- Zurg intercepts Alice’s authentication attempt and delivers it himself
Other kinds of cryptographic tokens

Challenge-response cards
- User activates card and tries to login
- Server sends challenge $C$ to user
  - Random #, timestamp, counter, etc
- Card computes response $R = f(C,K)$ where $K$ is some shared secret

Time-sync cards
- Similar but use current time instead of $C$
  - Require good clock synchronization

Can use PIN to limit time that $K$ can be used; erase $K$ after multiple bad attempts

Some variant of these – typically as a smart phone app – in most two-factor authentication schemes
But depends on the security of the shared secret...
Key idea: use some unique identifying characteristic to authenticate user (or create credentials)

- Physical feature: fingerprint, iris print
- Behavioral characteristic: handwriting, typing
- Combination thereof: voice, gait

Big pluses:
- Nothing to remember, same everywhere
- Passive
- Can’t be delegated
- Can be very strong differentiators (unique-ish)
Real-world biometric ids

- Face recognition
- Voice recognition
- Handwriting
- Fingerprint: first to be systematized
Fingerprint history

Identified by multiple people in late 1800s
- Highly distinct, phenotypic
- Change little over time

Quickly implemented by law enforcement
- Manual card catalogs, sorted across values in 10 points
- Search problem scaling becomes critical in late 1970’s

US LEA’s switch to computerized system (AFIS)
- Consequence: cold cases solved via comprehensive search (forensic use)
Technology issues

As a rule for all biometrics

- Analog sample -> set of digital features
- Use enrolls feature set; saved and associated
- On registration, do approximate match against saved features

Challenges

- Accuracy
- Easy of use (particularly enrollment)
- User acceptance
- Feature stability
Fingerprint scanners

Typically optical or capacitive sensors
- Produces “feature list” for finger
- Typically 2D geometry (some 3D)

“Liveness” checks
- Thermal sensors
- Electrical resistance
- Pulse, Perspiration

Good accuracy, so-so registration, ok maintenance (oily build up)
Public suspicion, but declining (largely due to iphone)
Hand scanners

Geometry
- Finger lengths/widths
- Gaps between fingers
- Span, etc
- Also versions that use two fingers only (V)

Time-consuming to train, but highly accurate, good registration and relatively low maintenance
High “comfort level”
Eye scanners

Retina
- 2D Geometry of Blood Vessels
  - Either optical or infrared
  - Blood warmer than tissue
- Stability issues

Iris Pattern
Highly accurate, easy enrollment/registration
Low maintenance
But, expensive and awkward to use… also issues with some contact lenses
Others

Voiceprint (usually used for forensic purposes)
Vein (vascular pattern in back of hand)
Face recognition (passive vs active)
Tissue organization (usually in hand)
Signature (special pen)
Typing (timing between character sequences)
Gait recognition
DNA
Enrollment issues

Unlike passwords, hard to pre-enroll user
Users must be enrolled interactively
For many biometrics, getting good accuracy requires multiple readings

- Build templates and test against registration
- Repeat
- Some templates simply tough (e.g., smooth fingerprint); “goats”
How strong is a biometric?

Non-adversarial
- False accept rate
- False reject rate

Adversarial
- Intercept
- Spoofing
Non-adversarial testing

False accept rate
- How many random trials before expectation of false accept > 0.5

False reject rate
- How many random trials before expectation of false reject > 0.5

Lower FAR = less tolerant of close matches
- Harder to attack
- Necessarily increases FRR

Lower FRR = more tolerant of close matches
- Easier to use
- Necessarily increases FAR

Since match is approximate can almost always tune for one or other

Equal error rate: point where FAR = FRR

Note, huge difference between a single false accept and system-wide false accept (more templates means more things you can accept against)
Interception

Typically used in distributed systems, so biometric template must be saved

Challenges

- Intercept template on network
- Inject forged template on network

Requires biometric scanner to be trusted and to have secure channel (authenticity, privacy, integrity, no-replay) to server... rare in practice.

Can one reconstruct biometric from template? (unclear, controversial)
Spoofing

Biometrics are private, but not secret
Users expose biometric instances everywhere
- Fingerprints, hand geometry, face, handwriting, iris, etc

Allows attacker to create biometric forgery

Making the problem worse
- Very hard to revoke a biometric
Latent fingerprints

Last user leaves residue of oils on scanner surface

- If you can make oils opaque can forge last finger

Real world strategies that work (Thalheim et al)

- Breathe on reader
- Press plastic bag with warm water to scanner
- Dust with graphite and apply transparent tape
Define latent fingerprint with superglue 
& take photo
- Could also use tape/graphite

Create mold using std photoresist techniques for making circuit boards to create mold
Making a Gummy Finger from a Latent Print

Gelatin Liquid

Drip the liquid onto the mold.

Put this mold into a refrigerator to cool, and then peel carefully.

40wt.%

From Matsumoto, ITU-T Workshop
Finally...

What if they just cut your finger off?

[Live Image] [Cadaver Image]

Shuckers et al
Anti-spoof techniques

Latent registration
- Save previous image and reject if identical
- Tricky: can pick up print and rotate to fool

Attack mold precision
- Pore detection (can capture with more porous material)
- Perspiration detection (issues w/temperature/users)

“Liveness” detection
- Temperature (put gummy in microwave)
- Pulse-detection (FRR and time issues)
- Capacitive load on ridges – capillary action of water on ridges
  » Monitor spread to defeat wet gummy fingers (need multiple registrations over a few secs)
There are spoofing techniques for virtually all biometrics
One approach

Multi-factor biometrics

Multiple biometrics
- Longer registration, more management

Biometric plus password

Biometric, plus token
- Loses convenience
Social issues

Biometric identifier can track your physical activities as well as your virtual activities
Many have legal standing not associated with virtual tokens
Easy to match (even if can’t spoof)
For next time...

Side Channels