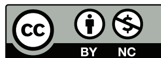
User Authentication



User Authentication

- Types
- Server issues
- Client issues

Threat-Driven Analysis

- Defense is always a response to offense: you don't need to defend against something that can't happen
- Do you need to defend against things that some adversaries can do but not all?
- Who is your enemy? Can they develop more sophisticated attacks?

Forms of User Authentication

- Something you know
- Something you have
- Something you are

Forms of User Authentication

- Something you know: **passwords**
- Something you have: e.g., **smart card**
- Something you are: e.g., **fingerprint**

Forms of User Authentication

- Something you know: **passwords**
- Something you have: e.g., **smart card**
- Something you are: e.g., **fingerprint**
- The latter two are a response to the weaknesses of passwords

Something You Know

- Ancient: “what’s the secret word? (Supposedly dates to at least Roman times.)
- Modern incarnation: passwords
- Most common form of authentication

Passwords

- Everyone understands the concept
- Passwords should be sufficient
- Not really. . .

Passwords are Really Bad

- Guessable
- Forgettable
- Enumerable
- Eavesdroppable (but that isn't a word...)
- Replayable
- Reusable
- Leakable
- Probably a lot more reasons not to use them

Threat: Password File Compromised

How Should Passwords be Stored?

- Not in plaintext
  - Administrator can see them
  - Can be stolen from backup media (or recycled disk drives...)
  - Editor bugs can leak them
  - ☞ Something that doesn't exist can't be stolen!
- Use a one-way hash; compare stored hash with hash of entered password
- Read-protect the hashed passwords anyway—why?

Guessable Passwords

- People tend to pick bad passwords
- Their own name, phone number, spouse's name, kids' names, etc.
- Easy to write password-guessing program (Morris and Thompson, CACM, Nov. 1979))

Guessable Passwords

- People tend to pick bad passwords
- Their own name, phone number, spouse's name, kids' names, etc.
- Easy to write password-guessing program (Morris and Thompson, CACM, Nov. 1979)
- Take careful note of that year...

Password-Guessing Programs

- Try likely words: names, dictionaries, etc.
  Use specialized dictionaries, too: other languages, science fiction terms, etc.
- Try variants: “password” → “passw0rd” or “Password”
- Use specialized, optimized algorithm
- In uncontrolled environments, at least 40-50% of people will have guessable passwords

Guessing Mechanisms

- Online: try to log in as the user
- Offline: steal a copy of the password file and try on your own machine (or on many compromised machines—including their GPUs)
- Note: that's why we read-protect the hashed passwords

Defenses

- Rate-limit online guesses
- Perhaps lock out the account—but that leaves you vulnerable to DoS attacks

Defenses

- Rate-limit online guesses
- Perhaps lock out the account—but that leaves you vulnerable to DoS attacks
- Make password-guessing inherently slow: use a slow algorithm

The Classic Unix Password-Hashing Algorithm

- Use DES, an encryption algorithm with 56-bit keys in 8 bytes. (In 1979, there were no cryptographic hash functions)
- Don't encrypt the password, encrypt a constant (all 0s) using the password as the key

☞ This is where the 8-character limit comes from

- Why not encrypt the password?

The Classic Unix Password-Hashing Algorithm

- Use DES, an encryption algorithm with 56-bit keys in 8 bytes. (In 1979, there were no cryptographic hash functions)
- Don’t encrypt the password, encrypt a constant (all 0s) using the password as the key

- Why not encrypt the password?
- The system would have to have the encryption and decryption keys available—and hence stealable
- Any decent cryptosystem can resist finding the key, given the plaintext and ciphertext
- Iterate 25 times, to really frustrate an attacker
- Guard against specialized hardware attacks by using the “salt” to modify the DES algorithm

Salt

- Pick a random number—12 bits, way back when—and use it to modify the password-hashing algorithm
- Store the salt (unprotected) with the hashed password
- Prevent the same password from hashing to the same value on different machines or for different users
- Makes dictionary of precomputed hashed passwords much more expensive
- Doesn't make the attack on a single password harder; makes attacks trying to find *some* password 4096× harder
- Today, we use much longer salts—why?

Salt

- Pick a random number—12 bits, way back when—and use it to modify the password-hashing algorithm
- Store the salt (unprotected) with the hashed password
- Prevent the same password from hashing to the same value on different machines or for different users
- Makes dictionary of precomputed hashed passwords much more expensive
- Doesn't make the attack on a single password harder; makes attacks trying to find *some* password 4096× harder
- Today, we use much longer salts—why?
- There are many more password files, and they're much larger

Salt

- Pick a random number—12 bits, way back when—and use it to modify the password-hashing algorithm
- Store the salt (unprotected) with the hashed password
- Prevent the same password from hashing to the same value on different machines or for different users
- Makes dictionary of precomputed hashed passwords much more expensive
- Doesn't make the attack on a single password harder; makes attacks trying to find *some* password 4096× harder
- Today, we use much longer salts—why?
- There are many more password files, and they're much larger
- How long should the salt be?

Salt

- Pick a random number—12 bits, way back when—and use it to modify the password-hashing algorithm
- Store the salt (unprotected) with the hashed password
- Prevent the same password from hashing to the same value on different machines or for different users
- Makes dictionary of precomputed hashed passwords much more expensive
- Doesn't make the attack on a single password harder; makes attacks trying to find *some* password $4096\times$ harder
- Today, we use much longer salts—why?
- There are many more password files, and they're much larger
- How long should the salt be?
- The birthday paradox gives the answer!

Examples of Salting

```
$ python3
>>> import crypt
>>> crypt.mksalt(method=crypt.METHOD_SHA256)
'$5$ps0ZRhwC0PmXxARk'
>>> crypt.crypt('password', salt='$5$ps0ZRhwC0PmXxARk')
'$5$ps0ZRhwC0PmXxARk$Kj0euOWYOVut3iNWJwloJFNGmtn7C6xRN.ifJQORQj1'
>>> crypt.crypt('password', salt='$5$ps0ZRhwC0PmXxARK')
'$5$ps0ZRhwC0PmXxARK$mMQfyYdG3ZZo0Kqeq6klqQmyHru9YL/TCncOteqosXD'
```

Note that a trivial change in the salt creates an entirely different hashed password

Also note that the salt is stored with the hashed password

Why Does Password-Guessing Work?

- People are predictable
- Passwords don't have much *information*
- According to Shannon, an 8-character word has 2.3 bits/character of information, or a total of 19 bits
- Empircally, the set of first names in the AT&T online phonebook had only 7.8 bits of information in the whole name
- 2^{19} isn't very many words to try...

Can We Lengthen Passwords?

- Today's password-hashing algorithms are based on cryptographic hash functions, which take unlimited lengths of input strings
- Are long passphrases guessable?
- Running English text has low entropy—but no one has built a guessing program to exploit that
- No one knows if it's even possible to exploit it

Password Change Requirements

- Most security specialists agree that mandatory password changes are a bad idea
- People just add punctuation or counters to their current passwords: `MyStrong*Pw` becomes `MyStrong*Pw1` or some such
- Algorithms have been developed that can predict password changes!
- Current NIST standards agree: it's a bad idea

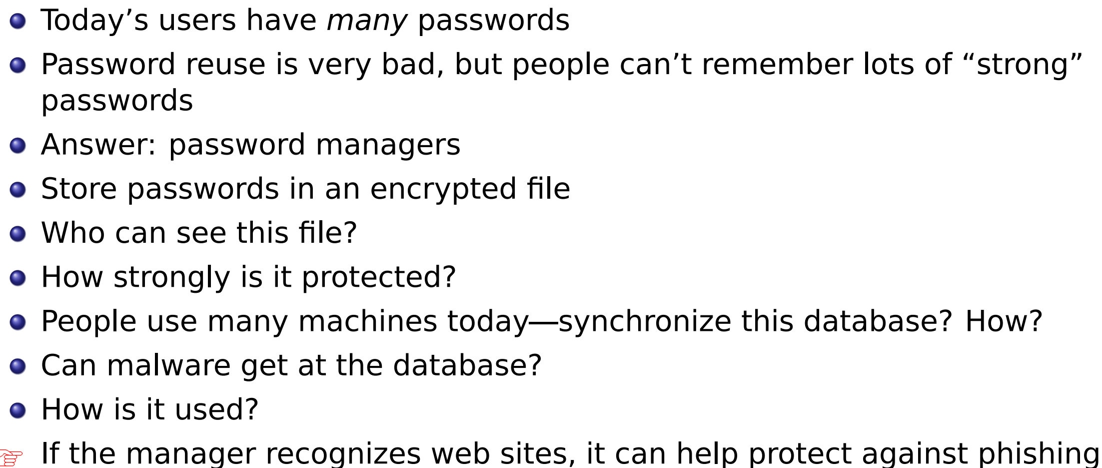
Forgettable Passwords

- People forget seldom-used passwords (or ones they've just changed to. . .)
- What should the server do?
- Email them? Many web sites do that
- ☞ What if someone can read your email? Your email password is the most valuable one you have!
- Reset them?
- ☞ How do you authenticate the requester?
- Password hints?
- Is it bad to write down passwords? If your threat model is electronic-only, it's a fine thing to do. If your threat model is physical, forget it. (See the movie "Ghost")
- Primary physical threats: domestic partners, relatives, friends, and—for some people—law enforcement

Reusable Passwords

- People tend to reuse the same passwords in different places
- If one site is compromised, the password can be stolen and used elsewhere
- At the root of “phishing” attacks
- ☞ A fraud incident on Stubhub is believed to have used passwords stolen from Adobe.com.
- Password reuse is a *very* serious threat

Password Managers



- Today's users have *many* passwords
- Password reuse is very bad, but people can't remember lots of "strong" passwords
- Answer: password managers
- Store passwords in an encrypted file
- Who can see this file?
- How strongly is it protected?
- People use many machines today—synchronize this database? How?
- Can malware get at the database?
- How is it used?

☞ If the manager recognizes web sites, it can help protect against phishing

Eavesdroppable

- Wiretapping the net isn't hard, especially if wireless links are used
- Done on the Internet backbone in 1993-4; see CERT Advisory CA-1994-01
- Install a keystroke logger on the client
- Install a password capture device on the server
- Play games with the DNS or routing to divert the login traffic

Stealable

- Shoulder-surfing
- Bribery—trade a password for a candy bar (http://news.bbc.co.uk/2/hi/technology/3639679.stm)

The Fundamental Problems

- Passwords have to be human-usable
- Passwords are static, and hence can be replayed

Tokens: Something You Have

- Many forms of tokens
- Time-based cards
- USB widgets (“dongles”)
- Challenge/response calculators
- Mobile phones
- Smart cards
- NFC tokens
- More

Disadvantages of Tokens

- They can be lost or stolen
- Lack of hardware support on many machines
- Lack of software support on many machines
- Inconvenient to use
- Cost

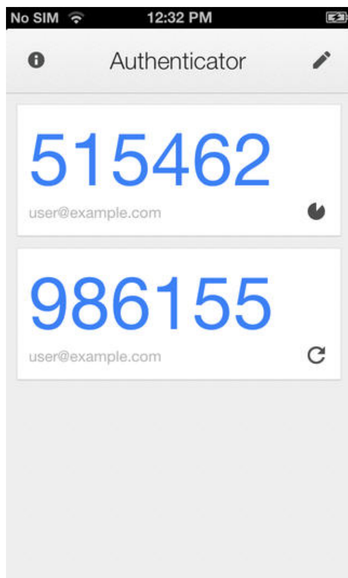
Two-Factor Authentication

- Two of the three types of authentication technology
- Use second factor to work around limitations of first
- Example: SecurID card *plus* PIN

SecurID Tokens

A SecurID token on two successive time cycles. The bars on the left of the second picture indicate how many 10-second ticks remain before the display changes, in this case about a minute. In essence, the display shows $H_k(T)$, where T is the time and H_k is a keyed hash function.
Generic name: TOTP (Time-based One-Time Passwords)

Soft Tokens



- Phone apps can do the same things as dedicated tokens (CU uses Duosec)
- The partially-filled circle shows the time left for that code; there's a refresh button to generate a new one
- But—is the cryptographic secret protected as well as on dedicated tokens? There are hardware and software attacks possible now

Eavesdropping Again

- Can’t someone eavesdrop on a token-based or two-factor exchange?
- Sure!
- Must use other techniques as well: encryption and/or replay protection

Replay Protection

- SecurID: code changes every minute; database prevents replay during that minute
- Challenge/response: server picks a unique number; client encrypts it
- Cryptographic protocols

Cryptographic Authentication

- Use cryptographic techniques to authenticate
- Simultaneously, negotiate a key to use to protect the session
- But where do the original cryptographic keys come from?

Cryptographic Keys are Long

- An AES key is at least 128 bits. Care to remember 32 hex digits as your password?
- An RSA key is at least 2048 bits. Care to remember 512 hex digits as your password?
- Solution 1: store the key on a token
- Solution 2: store the key on a computer, but encrypted

Storing Keys on Tokens

- The most secure approach
- Proper integration with host software can be tricky
- Generally want two-factor approach: use a password to unlock the token
- Ideally, the token is tamper-resistant

Storing Keys on Hosts

- Software-only approach is useful for remote logins
- *Must* use passphrase to encrypt key
- Not very resistant to capture of encrypted key—we're back to offline password guessing
- Can you trust the host to protect your key?

Use a Passphrase as a Key?

- Convert the user's passphrase to a key, and use it directly
- (Remember PBKDF2)
- Remember the low information content of passphrases. . .
- Attack: eavesdrop on an encrypted message; guess at passphrases; see which one yields a sensible decryption

Standardized Tokens: FIDO2

- Industry-standard token
- Many vendors, many interface technologies: USB-A and USB-C, NFC, Bluetooth
- Via browser integration, the tokens cryptographically verify the remote site's certificate
- Completely prevents phishing attacks
- Some FIDO2 tokens are unlocked via fingerprints

☞ FIDO2 tokens are the best general-purpose login technique currently available and should be used far more widely, especially for email and enterprise logins

Why Should Tokens be Tamper-Resistant?

- Prevent extraction of key if stolen
- Note: recovery of login key *may* permit decryption of old conversations
- Prevent authorized-but-unfaithful user from *giving* away the secret—you can't give it away and still have use of it yourself.
- Folks have pointed cameras at their tokens and OCRed the digits. . . http://smallhacks.wordpress.com/2012/11/11/reading-codes-from-rsa-secureid-token/

Mobile Phones

- Use a phone as a token: send an SMS challenge to the phone
- Indepedent failure mode: will the attacker who has planted a keystroke logger on a computer also have access to the owner's phone?
- Are there privacy risks from everyone having your mobile number?
- What about malware on the phone?

☞ But: attackers have learned to cope with SMS

Other Threats

- Bogus SIM cards, with the help of a deluded carrier
- An attacker who can inject certain messages into the phone network
- An attacker who controls the phone network
- Inceasing linkage between hosts and phones reduces the second factor: it's no longer independent

Federated Authentication

- Log in—via strong-but-inconvenient authentication—to Facebook, Google, etc.
- These sites vouch for your identity to other sites
- What about privacy? (Mozilla's solution tries to solve this.)
- Do you trust some other site to vouch for your users? Your employees?

Today's Status

- Strong passwords are rarely the solution—but sites can enforce them
- The Morris and Thompson paper is from 1979—users had no local storage, no local computing power, and very few logins
- Password-guessing typically represents the *second* failure, after a site is hacked (but you still need rate-limiting)
- Password reuse is a much bigger sin
- Use FIDO2 tokens!

Biometrics

- Something you are
- A characteristic of the body
- Presumed unique and invariant over time

Metanote: biometrics is an area of rapid progress; some of the limitations I describe here are likely to change in the near future. Exercise: which of the problems are likely to remain difficult issues for system designers?

Common Biometrics

- Fingerprint—common on phones
- Facial recognition—also common on phones
- Iris scan
- Retinal scan
- Hand geometry

Advantages of Biometrics

- You can't forget your fingers
- You can't lend your eyes to a friend
- You can't fake a fingerprint
- Why aren't they used more?
- Maybe they're not that secure...

Some Problems with Biometrics

- False accept rate
- False reject rate
- Fake (or “detached”) body parts
- “Bit replay”
- Non-reproducibility
- Many biometrics are *public*

False Accept Rate

- No biometric system is perfect
- Reducing false accept rate increases false reject rate
- Usual metric: what is the true accept rate for a given false accept rate?
- Substantial difference between different products
- Dramatic improvements in facial recognition over the last several years, as hard-coded algorithms were replaced by machine learning
- All systems work much better for one-to-one match than “does this biometric match something in the database?”

Why is One-to-One Match Better?

- Suppose that the false positive on a 1-1 match is F
- Assume that the database has N entries
- False positive probability on one-to-many match is $1-(1-F)^N$
- For $F=10^{-6}, N=1000000$, that's 63%

False Reject Rate

- People change, including aging
- Cuts, scars, glasses, colds, bandages, etc.
- Problems in original image acquisition

Capture Quality

- Quality of the captured data, for both initial enrollment and checking, is crucial
- Facial recognition *can* work well, but only under good circumstances, including lighting, angle, obscuring details (e.g., a hat or sunglasses), etc.
- Bias issues: facial recognition works best on white males; more problems with women's faces, darker-skinned faces, etc.

Storing Biometrics

- Store a template, not the actual picture
- Essentially, a one-way hash
- But are these templates reversible?
- It’s hard to change a fingerprint. . .

Bit Replay

- Ultimately, a biometric translates to a string of bits
- If the biometric sensor is remote from the accepting device, someone can inject a replayed bit stream
- What if someone hacks a server and steals a biometric? You can't change your fingerprints. . .

☞ Note: this happened with the OPM database breach

- Encryption helps; so does tamper-resistance
- Relying on human observation may help even more

Using Biometrics

- Biometrics work best in public places or under observation
- Remote verification is difficult, because verifier doesn't know if it's really a biometric or a bit stream replay
- Local verification is often problematic, because of the difficulty of passing the match template around
- Users don't want to rely on remote databases, because of the risk of compromise and the difficulty of changing one's body
- Best solution: use a biometric to unlock a local tamper-resistant token or chip; store keys there

This is what the iPhone does

Questions?

(Red-tailed hawk, Central Park, July 16, 2019)