AUTHENTICATION

1. Group presentation
2. Notes from reviews:
    a. How authentication works
    b. Known plain text attacks
    c. Preventing replay attacks
    d. How verify protocols?
    e. Can a single AS handle a network?
    f. Use of caching?
    g. Seriation – providing order to a set of things
    h. Multiple servers
    i. How avoid spoofing AS?
        i. Must know key Ka
    j. Malicious AS?
3. Authentication procedure
    a. Identifying a user to the computer
    b. Essentials:
        i. What do you know?
            1. Information only correct user would have
            2. A password
            3. Your dog's name
            4. ISSUES: human's are limited in what they can remember
        ii. What do you have?
            1. A smart card
            2. A metal key
            3. ISSUES:
                a. Limited size
                b. Users may forget to bring it
        iii. Who are you?
            1. Biometrics
            2. ISSUES:
                a. People who lack that item
                b. Can be copied with silicon gel
                c. How do you get the information to the computer?
    c. Issues: must communicate information from user to computer
        i. Channel may not be trust worthy
            1. Cameras on keyboards
            2. Keystroke loggers

            3. Network sniffers
        ii. QUESTION: how impact biometrics?
        iii. SOLUTION:
            1. Encryption. Where do you do it?
                a. On password database (unix): doesn't help sniffers, but allows accidental release
                b. On terminal (Windows): doesn't help untrusted workstations
                c. On smart card: user controls a small computer that does the verification
    d. Issues: why should user trust computer?
        i. QUESTION: How can we solve this?
        ii. ANSWER: computer must provide information
            1. E.g. Phishing prevention by bank providing information it knows about you
            2. Smart card verifies password of computer
4. PASSWORDS
    a. QUESTION: what is threat model for passwords?
        i. Physical access to machine?
        ii. Access to access links?
        iii. Examples: Ashley Madison, etc.
    b. Solution 1: Password file, protected by ACLS
        i. QUESTION: why is it a problem?
            1. A: hard to get perfect (depends on unbuggy software, brittle)
            2. A: hard to synchronize public info (name, UID) with private info
            3. EXAMPLES:
                a. Editors confusing temp files
                b. Backup / restore
                c. Boot alternate OS, look at file
        ii. Problems:
            1. Some systems check 1 character at a time (e.g. strcmp)
            2. Could break by placing address at page boundary, seeing whether fault occurred.
    c. Solution 2: cryptographic hash passwords and store in file
        i. BENEFITS:
            1. Can make file world-read (can't use information)
                a. QUESTION: Why can't you use the pwd information?
        ii. PROBLEMS:

            1. Vulnerable to precomputed brute-force attacks
                a. Try all short strings
                b. Try words in dictionaries, dates, etc.
                c. Can tell when two people have same password
    d. Solution 2a: assign random passwords
        i. Problem: need good source of randomness. Clock is not good enough (e.g. 16 bits)
        ii. Netscape SSL implementation had this problem. Used clock to seed random number generator. How many possible values for a key created today? (out of 128 bits in key)
    e. Solution 3: slow encryption function + salt
        i. Salt: append public number to pwd before hashing
            1. Login program reads salt, does hash
        ii. BENEFITS:
            1. Blows up size of precomputed dictionary
            2. Prevents seeing if two users have same PWD, or same user on 2 systems has same PWD
    f. PROBLEMS:
        i. How can you use unix password file over a network?
    g. Microsoft solution:
        i. 1-way function + protection
5. Problems statement
    a. Basic problem: you want to know who is talking to you over a network
        i. QUESTION: who should know this? Client, server or both?
        ii. Client authentication: common in LAN; server identifies client (e.g. file server)
            1. Important before releasing sensitive data
        iii. Server authentication : common on web: client identifies server (e.g., SSL)
            1. Important before storing or passing on sensitive data
        iv. Mutual authentication: both
    b. Assumptions: QUESTION: what are they
        i. **Need threat model;** what you are protecting against.
        ii. Network totally untrustworthy
            1. Can sniff messages
            2. Can inject messages
            3. Can replay messages
            4. Can forge addresses (but harder)

        iii. Trust local hardware to perform correctly
        iv. Trust the people you trust not to divulge secrets
        v. **QUESTION: What not concerned about?**
            1. Traffic analysis
            2. Reversing cryptography
            3. Known plain-text
                a. Used to defeat Enigma
    c. Constraints
        i. Large, scalable system → not share passwords with everyone
        ii. Low overhead → cache information as much as possible, few round trips & messages
        iii. Use long-term keys as little as possible
    d. Question: can you prove a protocol secure?
        i. Unlike proving a protocol correct – you have to enumerate all the threats & beliefs
        ii. Many people have come up with frameworks, hard to use

1. Encryption overview
    a. Symmetric Key / Shared Key – fast (us)
        i. both sides know the key
        ii. Use same key for encrypt / decrypt
        iii. You have an account with a key, and you see a message (M)ka. Who could have signed it?
            1. A: user A or password database server
    b. Public key / asymmetric key – slow (ms)
        i. Two different keys. If one encrypts, other can decrypt
        ii. ((M)PK)SK) = M
        iii. ((M)PKA = secret for A, only A can decrypt but anybody can generate
        iv. ((M)SKA = signed by A – only A can generate but anybody can decrypt
        v. ((M)SKA)PKB = message signed by A that only B can decrypt
    c. Hash functions – one way functions
        i. Easy to generate, impossible to invert or find a collision
        ii. Used to avoid expensive operations on whole message
        iii. M,(H(M))SKA = M signed by SKA
        iv. Used for keyed hash: HMAC = H(M . PWD)
        v. **Good for integrity – detect modification**

    d. Issues:
        i. Speed: hash < secret < public (factor of 10)

6. Basic protocol for shared secrets
    a. Challenge response
        i. Client requests challenge
            1. C->S: ReqC
        ii. Server sends challenge
            1. S->C: C
        iii. Client encrypts with pwd, sends to server
            1. C->S: (C)Ka
        iv. Server verifies
    b. Idea: client proves it knows something without exposing what it is.
    c. Idea: based on shared knowledge of client and server
    d. Problems
        i. Man in the middle
            1. If you can make client connect to you, you can forward messages to server
        ii. Requires server & client to have a shared secret
            1. Used in Windows up to NT 4
            2.
7. How do you think about protocols: beliefs
    a. What does the client believe at each point
        i. E.g. nonce/challenge is fresh
        ii. Things that are encrypted imply other side knows the key
8. **Scaling** protocol: what if have star-formation, a server knows all the keys
    a. Needham & Schroeder
        i. Assumptions: clients, servers share a key with authentication server not known to anyone else
        ii. Protocol:
            1. A → AS: A,B, Nonce-a
            2. AS → A: (Nonce-a, B, CK, (CK,A)KB)KA
            3. A → B: (CK,A)KB
            4. B → A: (Nonce-B)CK
            5. A → B: (Nonce-B – 1) CK
        iii. points:
            1. Essence: server performs a **ticket protocol** as compared to a list protocol, AS does a list protocol.
                a. Ticket protocol: just verify one thing, no lookups in list

b. List protocol: lookup something in a list (e.g. name)

2. **Nonce: preserves freshness – can tie response to request**
3. **3.** When things sent in clear, response sends back same value encrypted (or decremented) to indicate it got through (e.g. message 2)
4. **4.** When ambiguous who a message came from, must include name (e.g. message 3)
5. (Nonce-B)CK is an **authenticator** – it demonstrates that B can decrypt CK, and is used to verify that A knows CK as well
6. Using CK has benefits – can now be used to encrypt additional data between A and B

iv. QUESTION: What do you learn at the end?
   1. Is the user allowed to access the service?
   2. What the name of the user is?

v. **QUESTION: do you need to encryption Nonce-A and B in message 2?**
   1. No; just need integrity. Could use HMAC with KA or CK

vi. **NOTE: can reuse authenticator**
   1. Can just do steps 3,4,5 multiple times
   2. As CK is not fresh (message 4 could be replayed, want to modify:
      a. 3': (CK,A)KB, (nonce-a)CK
      b. 4' (nonce-b, nonce-a – 1) CK
   3.

vii. Problems
   1. A must enter password each time it talks to a new server
   2. No limit on how long message 3 is good for – if can break in and steal CK
   3. Fixed in Kerberos
   4.

b. Needham & Schroeder public key
   i. Public key overview
      1. Authentication server only stores public keys
      2. Akin to a phone book, directory assistance
      3. Assume Client has PKAS – public key for server preinstalled
         a. Comes with browsers and operating systems currently
   ii. Protocol

1. A → AS: B
2. AS → A: (PKB, B) skas
    a. This is a certificate – signed by AS
3. A → B (Nonce-a, A)pkb
    a. Send a nonce and name encrypted with pkb
    b. B can decrypt this directly, but can't reply yet
4. B → AS: A
5. AS → B: (PKA, A) skas
    a. Again, a certificate. **Note: client could send this with request if wanted**
6. B → A: (Nonce-A, Nonce-B) pka
7. A → B: (nonce-B)

iii. Bugs:
1. Messages 2 and 5 could be replayed – no freshness here
2. Can do man-in-the-middle: I is intruder
    a. A → I: (Na,A)pki
    b. I → B : (Na,A)pkb
        i. Can be generated because knows ski, pkb
    c. B → I (Na, Nb)pka
        i. B sends back wrong value
    d. I → A (Na, Nb)pka
        i. I forwards it along
    e. A → I (Nb)pki
        i. A decrypts Nb on I's behalf
    f. I → B (Nb)pkb
        i. B now trusts I
        ii. I knows Na, Nb – what was considered a shared secret
3. Solution: include names when encrypting
    a. Change message 6 from
        i. B → A: (Nonce-A, Nonce-B)pka to:
        ii. B → A: (Nonce-A, Nonce-B, B)pka

iv. Compared to private key protocol
1. AS does less work: no encryption, just database lookup
2. AS must have strong integrity: can't let database be corrupted

v. Reality: public key encryption slow

            1. Encrypting with 1024 bit key takes ~ 1ms (1 million cycles)
            2. AES – 200 cycles
            3. People only encrypt secret session keys with public keys

9. One-way communication
    a. Motivation:
        i. Secure email.
        ii. Want privacy: message can't be seen
        iii. Want integrity: nobody else could have written message
        iv. Want replay protection: can't send message twice
    b. Conventional algorithm:
        i. A → B: (CK, A)kb , (message)ck
            1. Need timestamps to avoid replay
    c. Public key
        i. A → B: (A, I, (B)ska)pkb, (message+I)PKB
            1. Notes: nobody can see I, so can't be replaced
        ii. Problems
            1. I doesn't tightly bind message to crypto, could replace message and leave I
            2. Solution: use I as a crypto key: (message)i
            3. Remember I so can't be reused
10. Digital Signatures
    a. Motivation: provide evidence to a third party that the sender did something
    b. Conventional
        i. A → AS: A, (hash)ka
        ii. AS → A: (A, hash) kas
            1. AS encrypts hash and A's name –binding them together
        iii. B → AS: B, (A,hash)kas
        iv. AS → B : (A, hash)kb
    c. Public key
        i. A → B: ((text)ska)pkb
            1. Encrypt with pkb: only pkb can read it
            2. Encrypt with ska: only ska could write it
            3. Reality: text ((hash)ska)pkb for speed
11. Public key infrastructure
    a. Certificate servers
        i. Produce signed sertificates
            1. (A, pka)scs
        ii. Maintain revocation lists
            1. If A loses her secret key

    iii. Can produce certificate for other certificate servers

12. Decision: how do you choose what to use?
    a. Storing keys on client:
        i. Secret key / hash: can use passwords
        ii. Public key: must store private key somewhere
    b. Storing keys on server:
        i. Secret key / hash: data must be kept super-secret
        ii. Public key: data is not secret, but must not be modified
        iii. Server can be offline (not needed synchronously except for revocation)
    c. Key lifetime:
        i. Secret key: fairly short (minutes, months)
        ii. Public key: fairly long (need not be remembered)
    d. Verification:
        i. Secret key / hash: authentication server generally can get involved in verification (because it has to write the tickets)
        ii. Public key: end node does verification (because auth server doesn't generate anything)
    e. Direction:
        i. Secret key fine if you can establish keys
        ii. Public key good if you can't; you can bootstrap from one "trusted" value using PKI
        iii. Public key good for authenticating servers; users don't need their own key