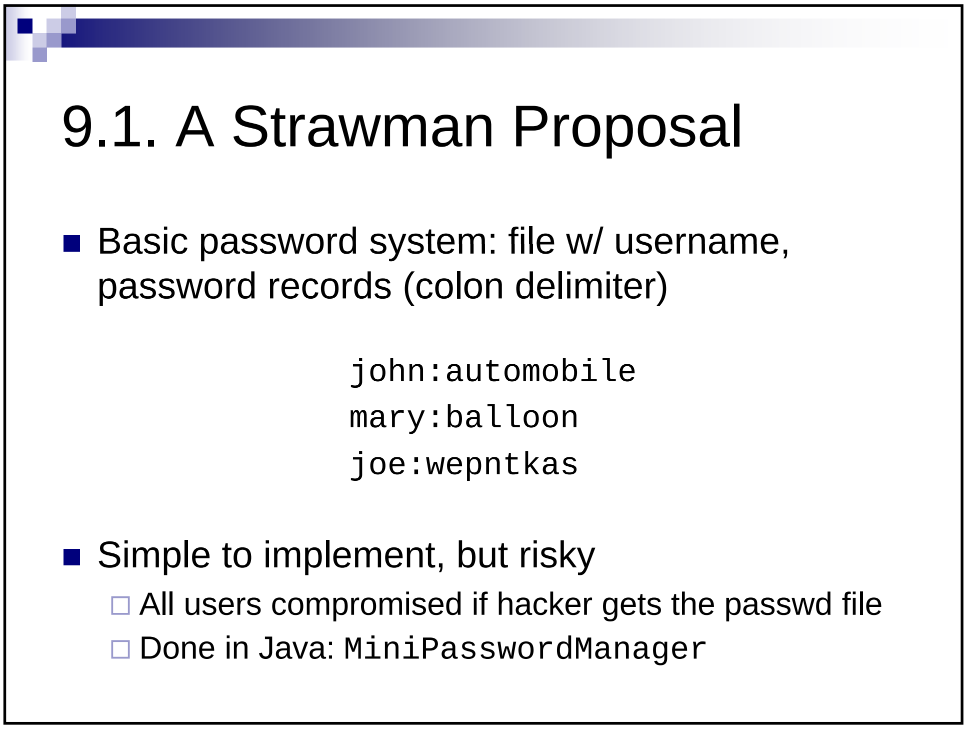
Password Security

Slides adapted from "Foundations of Security: What Every Programmer Needs To Know" by Neil Daswani, Christoph Kern, and Anita Kesavan (ISBN 1590597842; http://www.foundationsofsecurity.com). Except as otherwise noted, the content of this presentation is licensed under the Creative Commons 3.0 License.

Agenda

- Password systems ubiquitous, vulnerable
- Early password security studies (1979) - Morris, Thompson: 86% of passwords can be cracked
- Threats: Online & Offline Dictionary Attacks
- Solutions: Hashing & Salting

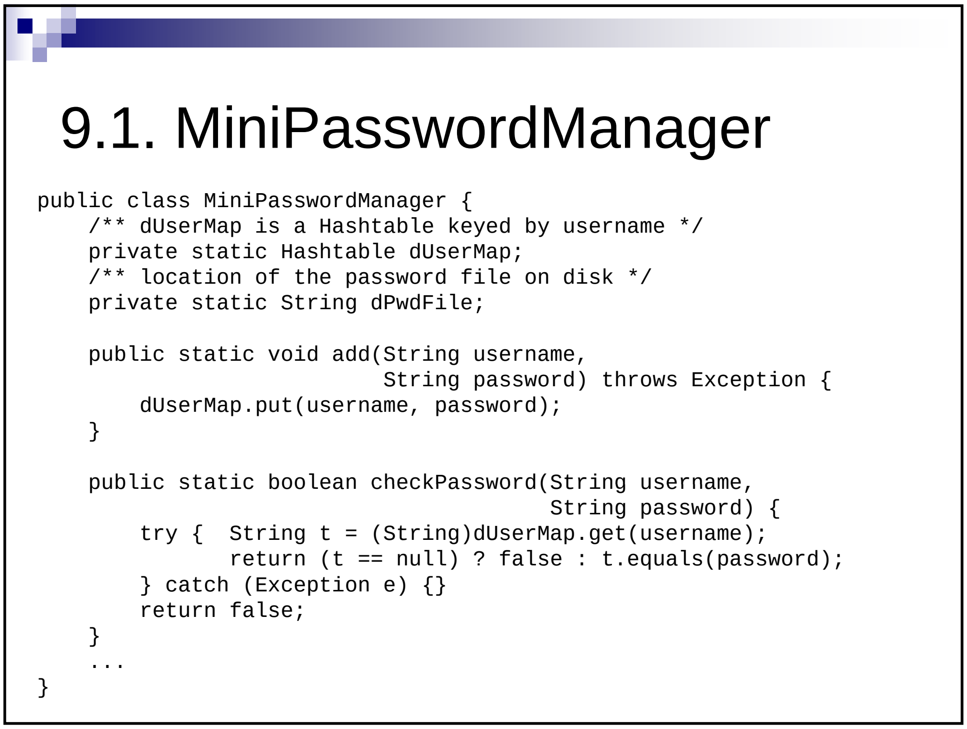


9.1. A Strawman Proposal

- Basic password system: file w/ username, password records (colon delimiter)

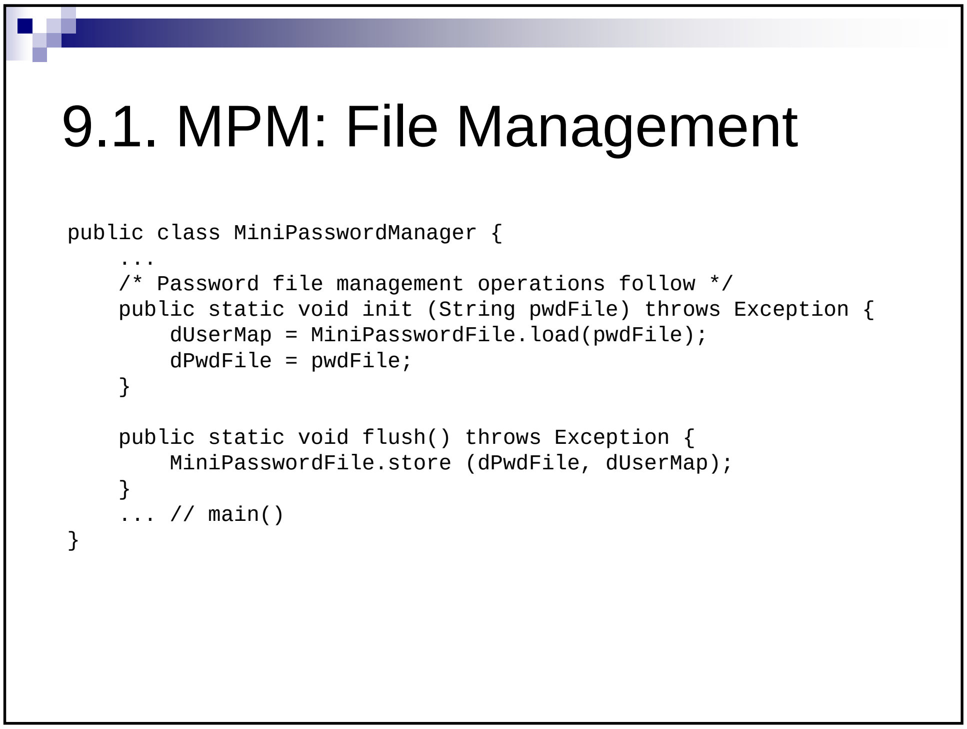
```
john:automobile
mary:balloon
joe:wepntkas
```

- Simple to implement, but risky
  - All users compromised if hacker gets the passwd file
  - Done in Java: `MiniPasswordManager`



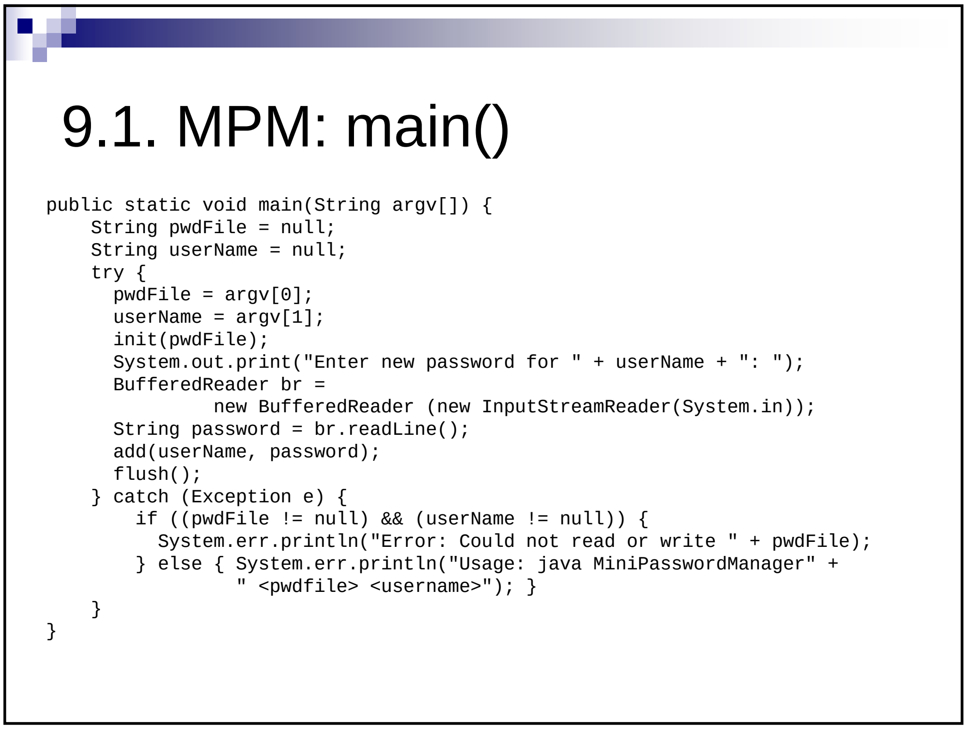
9.1. MiniPasswordManager

```
public class MiniPasswordManager {
    /** dUserMap is a Hashtable keyed by username */
    private static Hashtable dUserMap;
    /** location of the password file on disk */
    private static String dPwdFile;

    public static void add(String username,
                           String password) throws Exception {
        dUserMap.put(username, password);
    }

    public static boolean checkPassword(String username,
                                        String password) {
        try {  String t = (String)dUserMap.get(username);
               return (t == null) ? false : t.equals(password);
        } catch (Exception e) {}
        return false;
    }
    ...
}
```



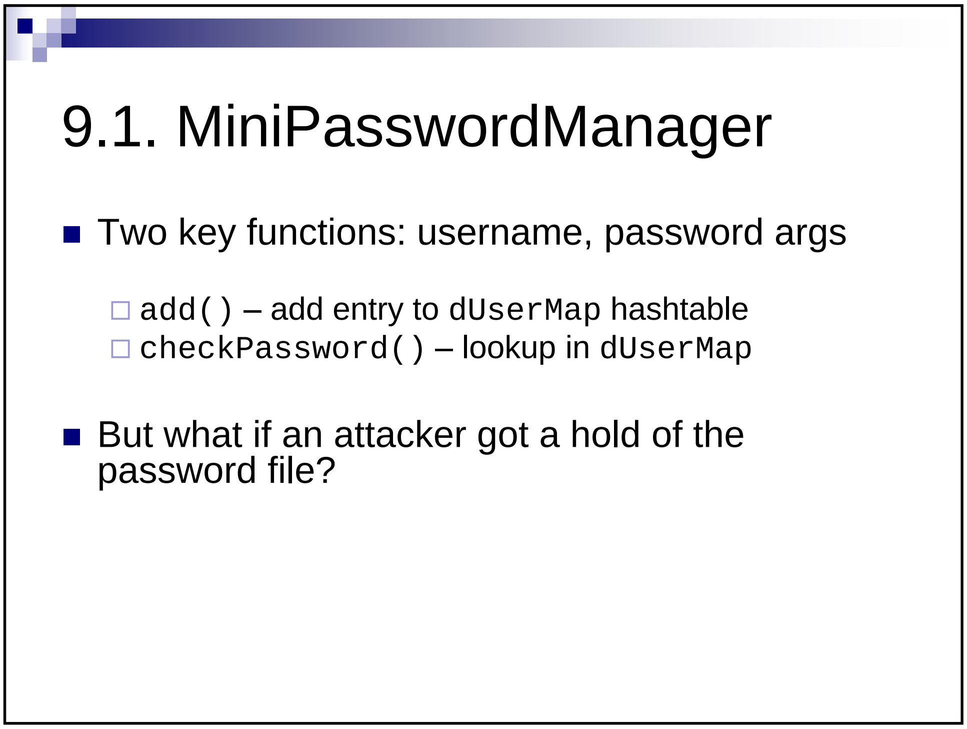
9.1. MPM: File Management

```
public class MiniPasswordManager {
    ...
    /* Password file management operations follow */
    public static void init (String pwdFile) throws Exception {
        dUserMap = MiniPasswordFile.load(pwdFile);
        dPwdFile = pwdFile;
    }

    public static void flush() throws Exception {
        MiniPasswordFile.store (dPwdFile, dUserMap);
    }
    ... // main()
}
```



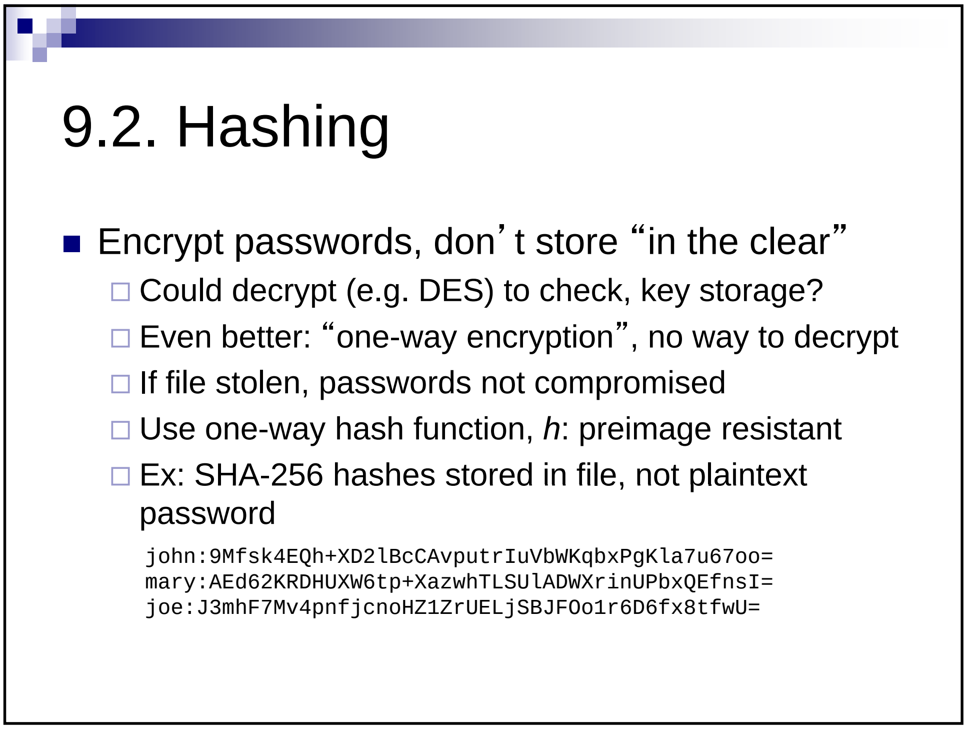
9.1. MPM: main()

```
public static void main(String argv[]) {
    String pwdFile = null;
    String userName = null;
    try {
      pwdFile = argv[0];
      userName = argv[1];
      init(pwdFile);
      System.out.print("Enter new password for " + userName + ": ");
      BufferedReader br =
               new BufferedReader (new InputStreamReader(System.in));
      String password = br.readLine();
      add(userName, password);
      flush();
    } catch (Exception e) {
        if ((pwdFile != null) && (userName != null)) {
          System.err.println("Error: Could not read or write " + pwdFile);
        } else { System.err.println("Usage: java MiniPasswordManager" +
                 " <pwdfile> <username>"); }
    }
}
```



9.1. MiniPasswordManager

- Two key functions: username, password args
  - □ `add()` – add entry to `dUserMap` hashtable
  - □ `checkPassword()` – lookup in `dUserMap`
- But what if an attacker got a hold of the password file?

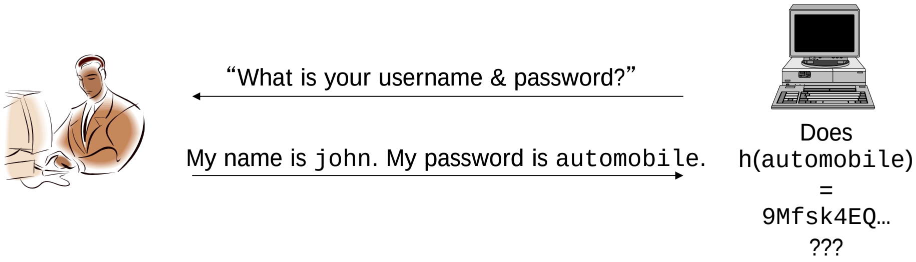


9.2. Hashing

- Encrypt passwords, don't store "in the clear"
  - □ Could decrypt (e.g. DES) to check, key storage?
  - □ Even better: "one-way encryption", no way to decrypt
  - □ If file stolen, passwords not compromised
  - □ Use one-way hash function, *h*: preimage resistant
  - □ Ex: SHA-256 hashes stored in file, not plaintext password

```
john:9Mfsk4EQh+XD2lBcCAvputrIuVbWKqbxPgKla7u67oo=
mary:AEd62KRDHUXW6tp+XazwhTLSUlADWXrinUPbxQEfnsI=
joe:J3mhF7Mv4pnfjcnoHZ1ZrUELjSBJFOo1r6D6fx8tfwU=
```

9.2. Hashing Example


“What is your username & password?”

My name is john. My password is automobile.

Does h(automobile) = 9Mfsk4EQ… ???

- Hash: “One-way encryption”
  - No need to (can’t) decrypt
  - Just compare hashes
  - Plaintext password not in file, not “in the clear”

9.2. Hashing MPM Modifications

```
public static void add(String username,
                       String password) throws Exception {
    dUserMap.put(username,computeSHA(password));
}

public static boolean checkPassword(String username,
                                    String password) {
    try { String t = (String)dUserMap.get(username);
          return (t == null) ? false :
                               t.equals(computeSHA(password));
    } catch (Exception e) {}
    return false;
}

private static String computeSHA (String preimage) throws Exception {
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    md.update(preimage.getBytes("UTF-8"));
    byte raw[] = md.digest();
    return (new sun.misc.BASE64Encoder().encode(raw));
}
```

9.3. Off-line Dictionary Attacks

- *Offline*: attacker steals file and tries combos
- *Online*: try combos against live system

Attacker Obtains Password File:

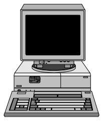
```
joe     9Mfsk4EQ...
mary    AEd62KRD...
john    J3mhF7Mv...
```

mary has password balloon!

Attacker

Attacker computes possible password hashes (using words from dictionary)

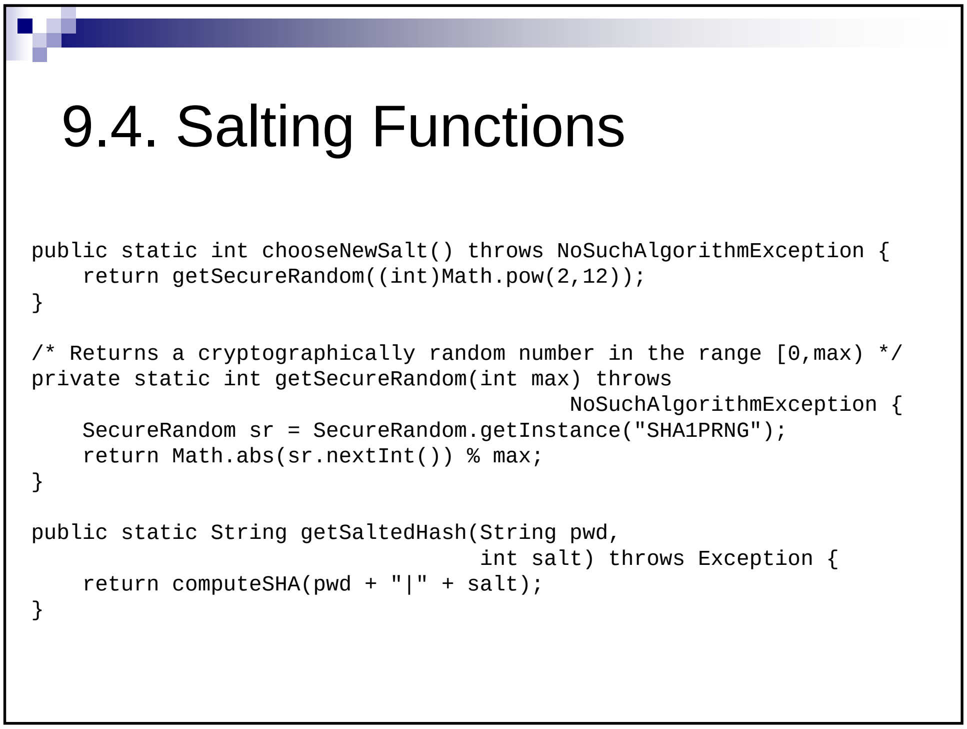
```
h(automobile)  = 9Mfsk4EQ...
h(aardvark)    = z5wcuJWE...
h(balloon)     = AEd62KRD...
h(doughnut)    = tvj/d6R4
```



9.4. Salting

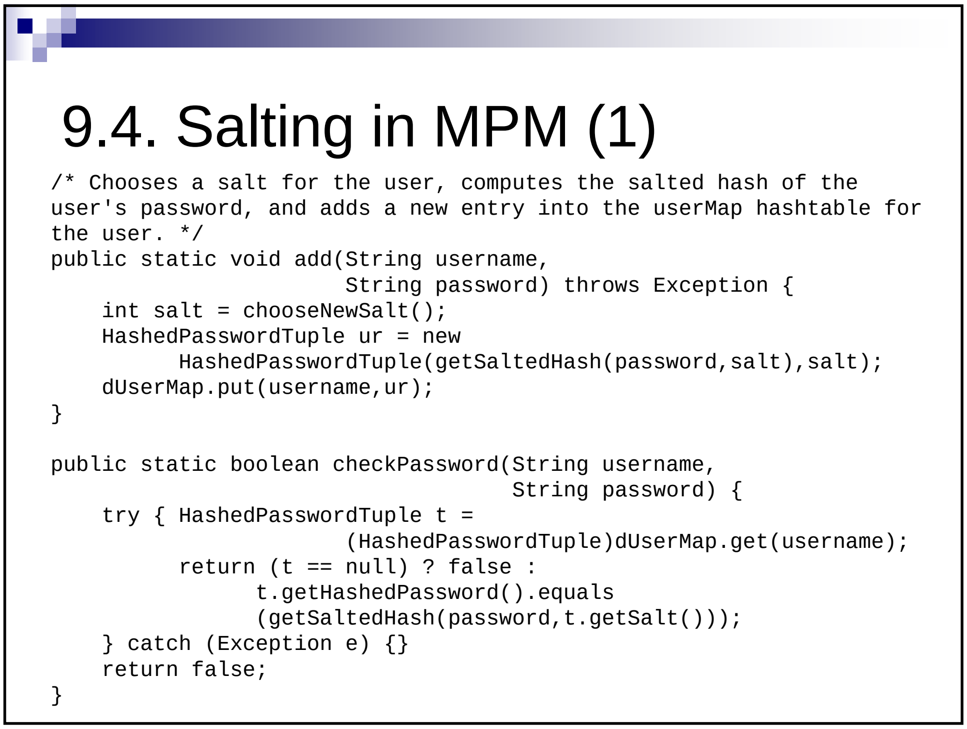
- *Salting* – include additional info in hash
- Add third field to file storing random # (*salt*)
- Example Entry: `john` with password `automobile`

  `john:ScF5GDhWeHr2q5m7mSDuGPVasV2NHz4kuu5n5eyuMbo=:1515`
- Hash of password concatenated with salt: *h(*`automobile|1515`*)* = `ScF5GDhW...`



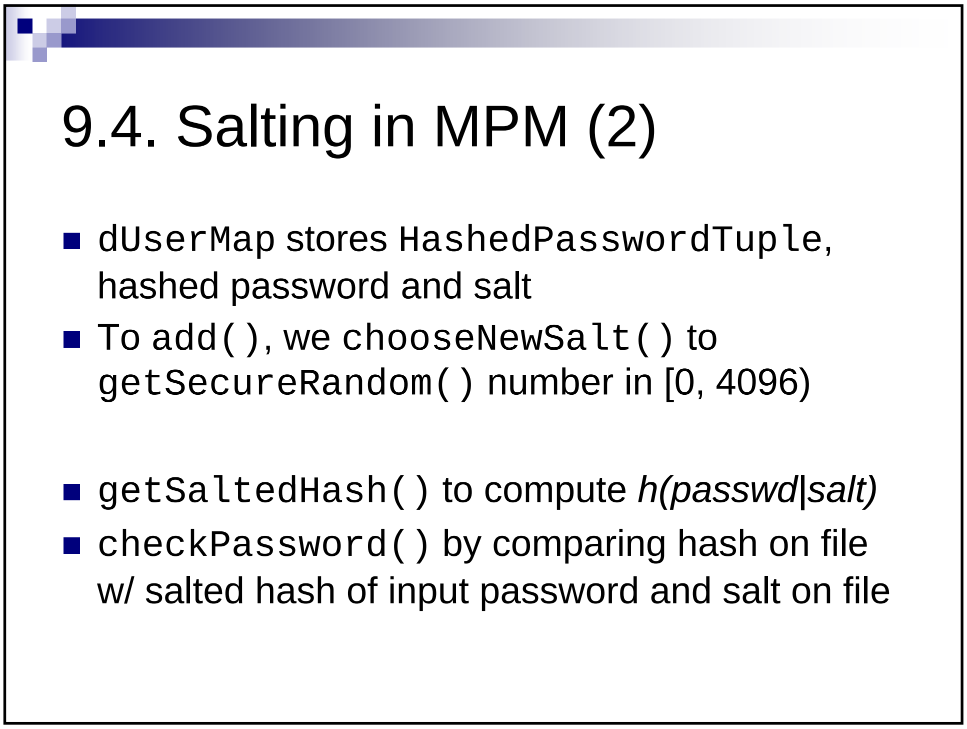
9.4. Salting Functions

```
public static int chooseNewSalt() throws NoSuchAlgorithmException {
    return getSecureRandom((int)Math.pow(2,12));
}

/* Returns a cryptographically random number in the range [0,max) */
private static int getSecureRandom(int max) throws
                                          NoSuchAlgorithmException {
    SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
    return Math.abs(sr.nextInt()) % max;
}

public static String getSaltedHash(String pwd,
                                   int salt) throws Exception {
    return computeSHA(pwd + "|" + salt);
}
```



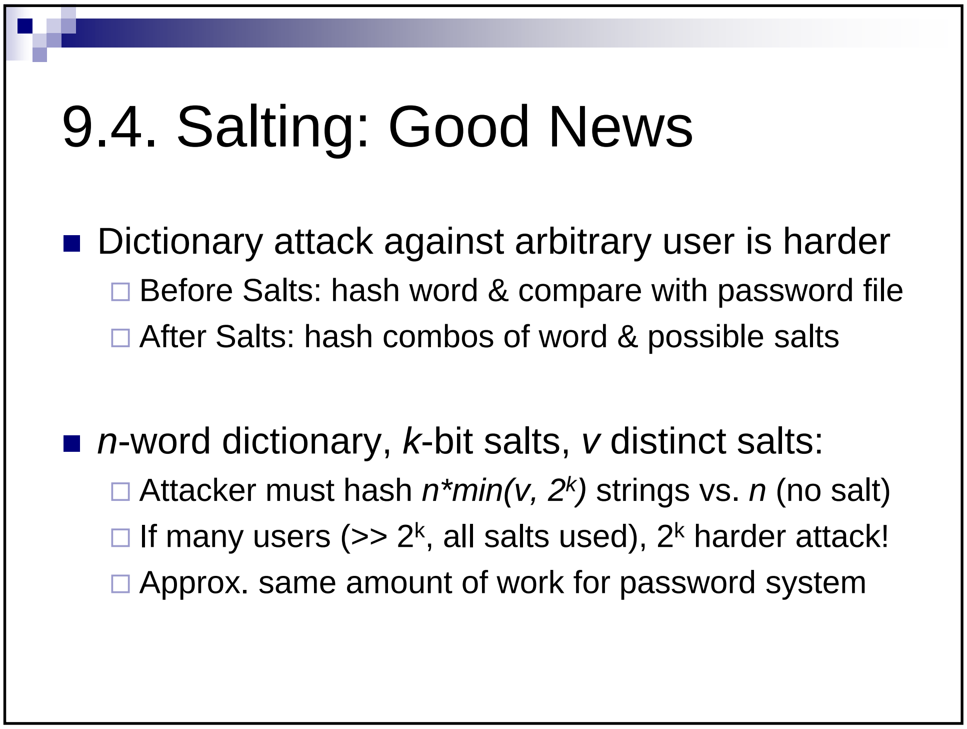
9.4. Salting in MPM (1)

```
/* Chooses a salt for the user, computes the salted hash of the
user's password, and adds a new entry into the userMap hashtable for
the user. */
public static void add(String username,
                       String password) throws Exception {
    int salt = chooseNewSalt();
    HashedPasswordTuple ur = new
          HashedPasswordTuple(getSaltedHash(password,salt),salt);
    dUserMap.put(username,ur);
}

public static boolean checkPassword(String username,
                                    String password) {
    try { HashedPasswordTuple t =
                       (HashedPasswordTuple)dUserMap.get(username);
          return (t == null) ? false :
                t.getHashedPassword().equals
                (getSaltedHash(password,t.getSalt()));
    } catch (Exception e) {}
    return false;
}
```



9.4. Salting in MPM (2)

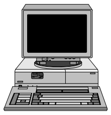
- `dUserMap` stores `HashedPasswordTuple`, hashed password and salt
- To `add()`, we `chooseNewSalt()` to `getSecureRandom()` number in [0, 4096)
- `getSaltedHash()` to compute *h(passwd*|*salt)*
- `checkPassword()` by comparing hash on file w/ salted hash of input password and salt on file



9.4. Salting: Good News

- Dictionary attack against arbitrary user is harder
  - □ Before Salts: hash word & compare with password file
  - □ After Salts: hash combos of word & possible salts
- *n*-word dictionary, *k*-bit salts, *v* distinct salts:
  - □ Attacker must hash $n*min(v, 2^k)$ strings vs. *n* (no salt)
  - □ If many users (>> 2^k, all salts used), 2^k harder attack!
  - □ Approx. same amount of work for password system

9.4. Off-line Dictionary Attack Foiled!



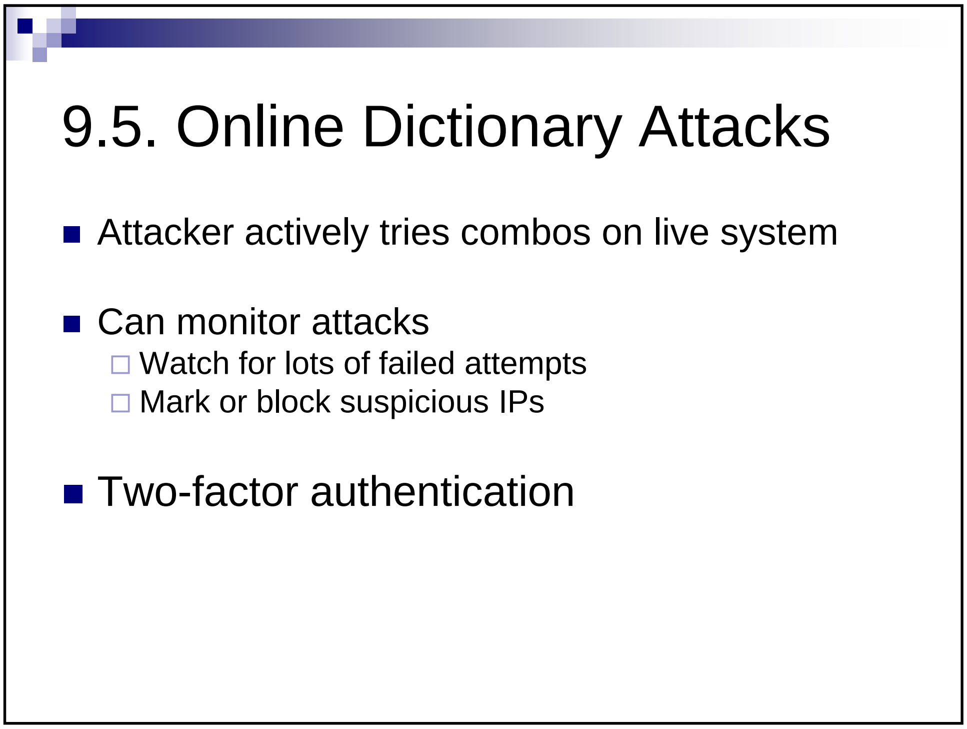
```
h(automobile2975) = KNVXKOHBDEBKOURX
h(automobile1487) = ZNBXLPOEWNVDEJOG
h(automobile2764) = ZMCXOSJNFKOFJHKDF
h(automobile4012) = DJKOINSLOKDKOLJUS
h(automobile3912) = CNVIUDONSOUIEPQN
...Etc...
h(aardvark2975)   = DKOUOXKOUDJWOIQ
h(aardvark1487)   = PODNJUIHDJSHYEJNU
...Etc...
```

```
/etc/passwd:
john   LPINSFRABXJYWONF    2975
mary   DOIIDBQBZIDRWNKG    1487
joe    LDHNSUNELDUALKDY    2764
```

Too many combinations!!! Attack is Foiled!

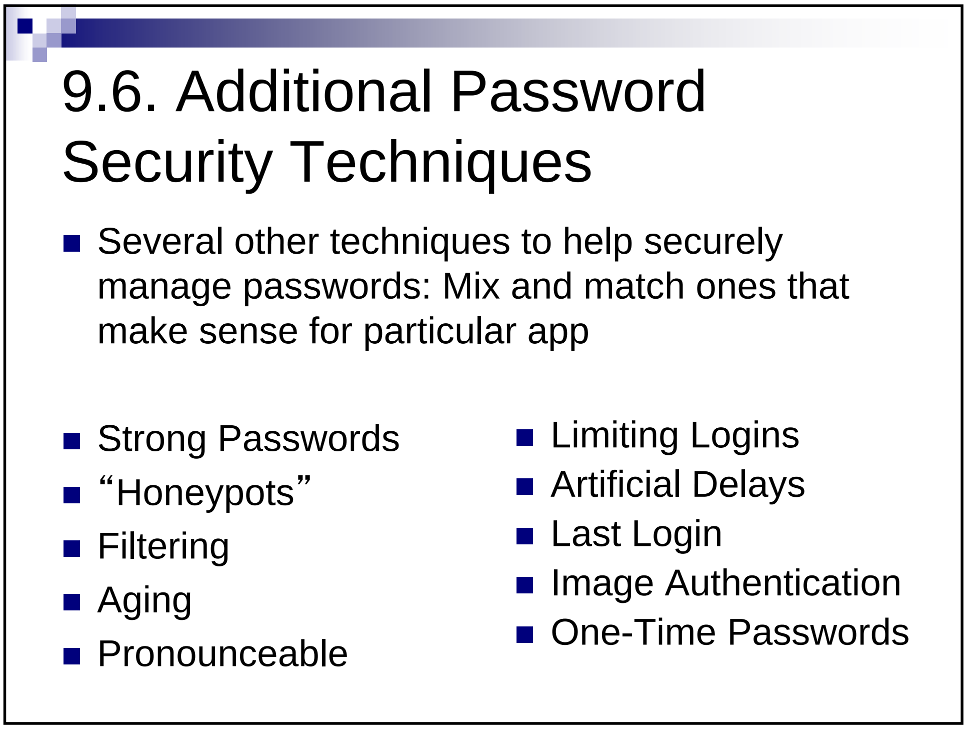
9.4. Salting: Bad News

- Ineffective against chosen-victim attack
  - □ Attacker wants to compromise particular account
  - □ Just hash dictionary words with victim's salt
- Attacker's job harder, not impossible
  - □ Easy for attacker to compute 2^kn hashes?
  - □ Then offline dictionary attack still a threat.



9.5. Online Dictionary Attacks

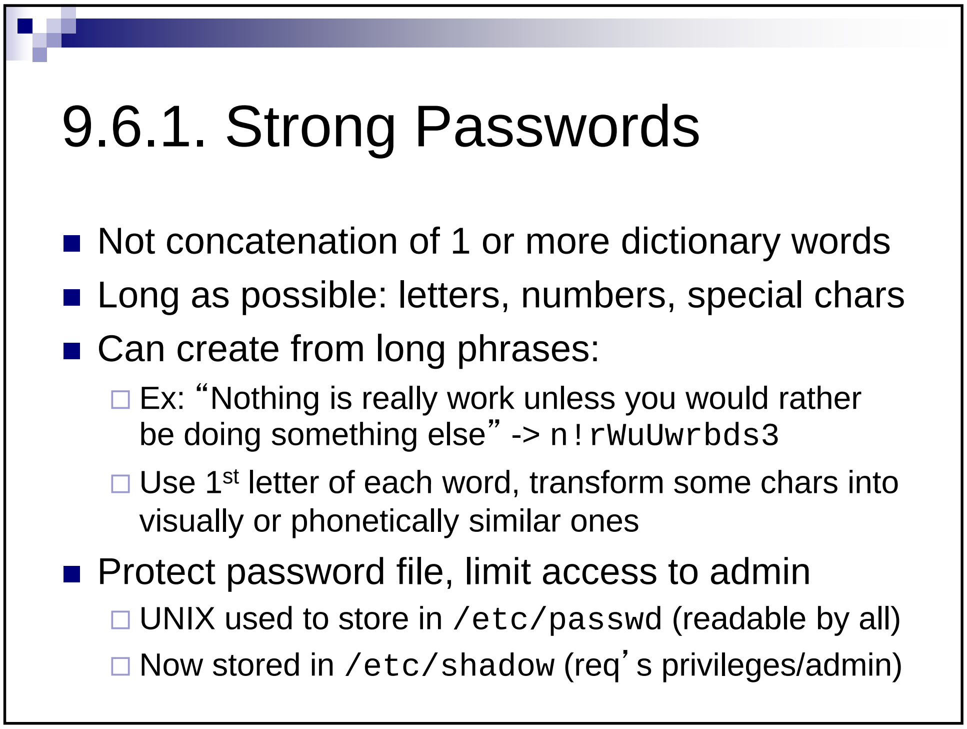
- Attacker actively tries combos on live system
- Can monitor attacks
  - □ Watch for lots of failed attempts
  - □ Mark or block suspicious IPs
- Two-factor authentication



9.6. Additional Password Security Techniques

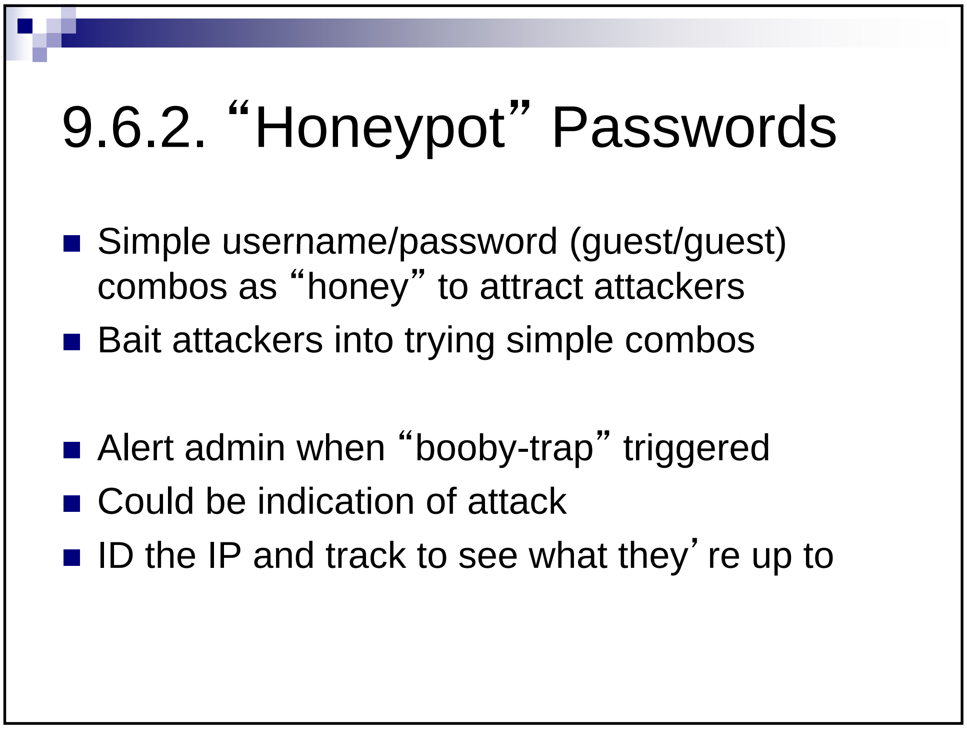
- Several other techniques to help securely manage passwords: Mix and match ones that make sense for particular app

- Strong Passwords
- “Honeypots”
- Filtering
- Aging
- Pronounceable
- Limiting Logins
- Artificial Delays
- Last Login
- Image Authentication
- One-Time Passwords



9.6.1. Strong Passwords

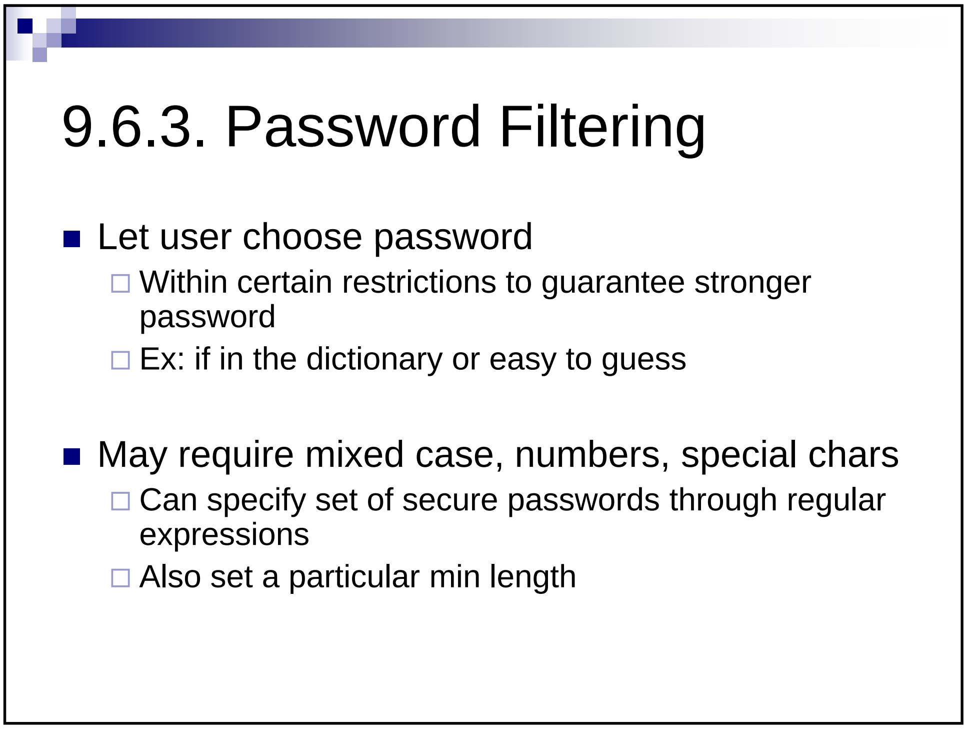
- Not concatenation of 1 or more dictionary words
- Long as possible: letters, numbers, special chars
- Can create from long phrases:
  - Ex: “Nothing is really work unless you would rather be doing something else” -> `n!rWuUwrbds3`
  - Use 1st letter of each word, transform some chars into visually or phonetically similar ones
- Protect password file, limit access to admin
  - UNIX used to store in `/etc/passwd` (readable by all)
  - Now stored in `/etc/shadow` (req’s privileges/admin)



9.6.2. “Honeypot” Passwords

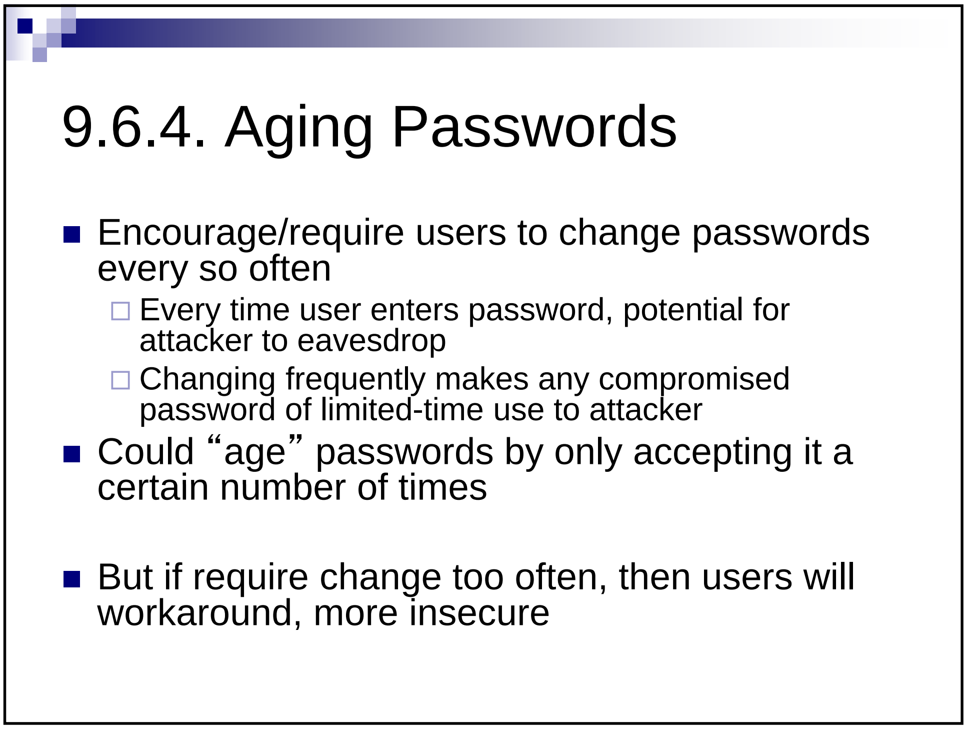
- Simple username/password (guest/guest) combos as “honey” to attract attackers
- Bait attackers into trying simple combos

- Alert admin when “booby-trap” triggered
- Could be indication of attack
- ID the IP and track to see what they’re up to



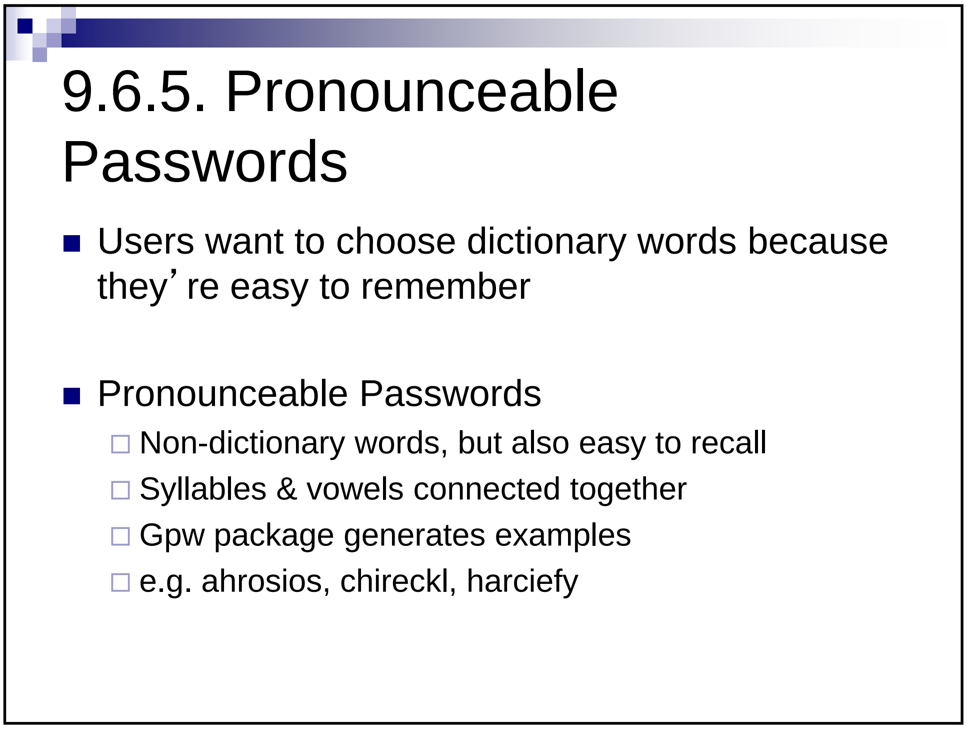
9.6.3. Password Filtering

- Let user choose password
  - □ Within certain restrictions to guarantee stronger password
  - □ Ex: if in the dictionary or easy to guess
- May require mixed case, numbers, special chars
  - □ Can specify set of secure passwords through regular expressions
  - □ Also set a particular min length



9.6.4. Aging Passwords

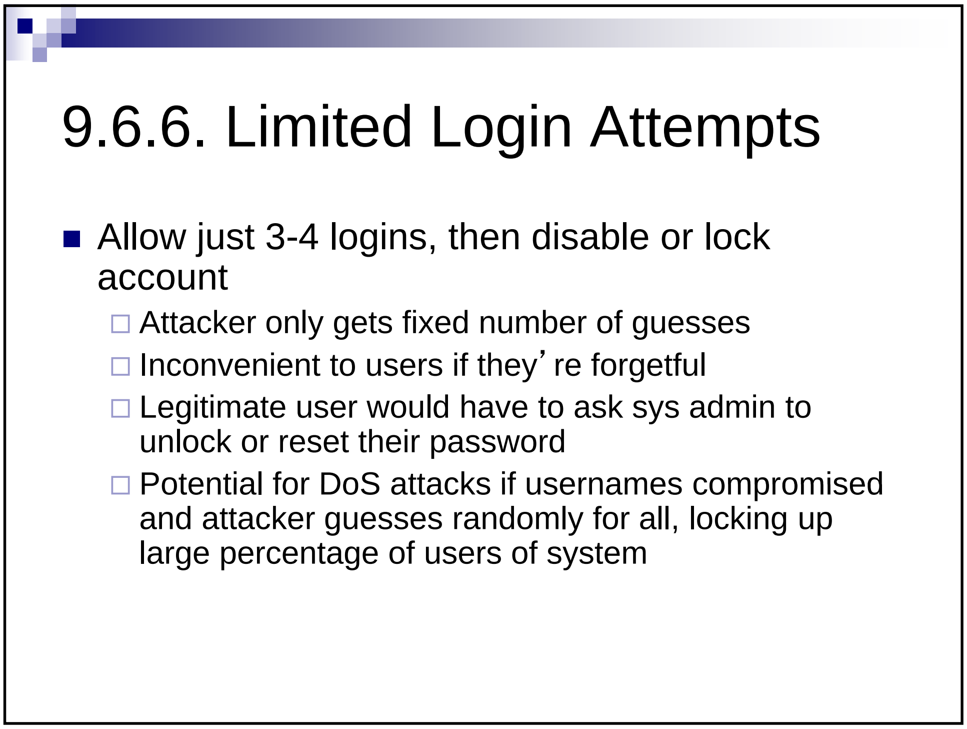
- Encourage/require users to change passwords every so often
  - □ Every time user enters password, potential for attacker to eavesdrop
  - □ Changing frequently makes any compromised password of limited-time use to attacker
- Could “age” passwords by only accepting it a certain number of times
- But if require change too often, then users will workaround, more insecure



9.6.5. Pronounceable Passwords

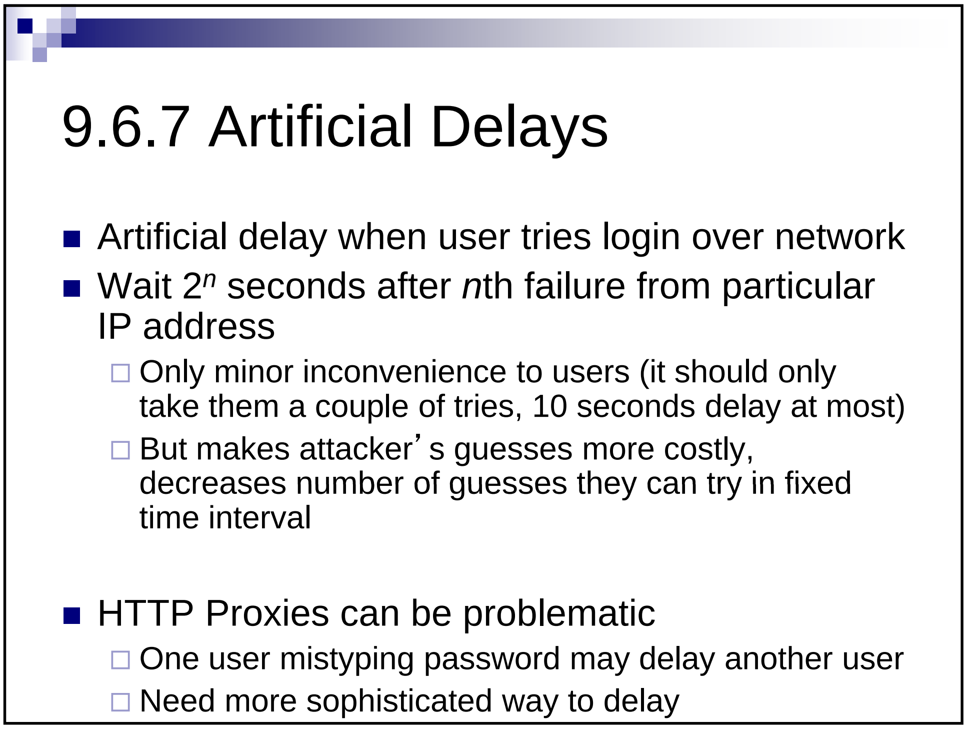
- Users want to choose dictionary words because they’ re easy to remember

- Pronounceable Passwords
  - □ Non-dictionary words, but also easy to recall
  - □ Syllables & vowels connected together
  - □ Gpw package generates examples
  - □ e.g. ahrosios, chireckl, harciefy



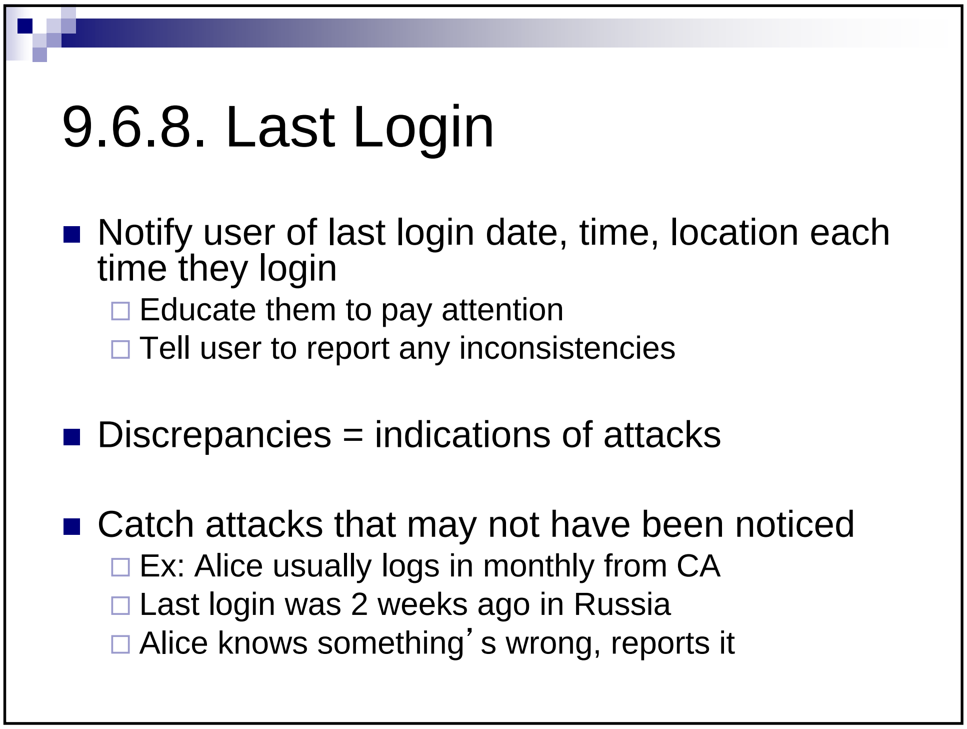
9.6.6. Limited Login Attempts

- Allow just 3-4 logins, then disable or lock account
  - □ Attacker only gets fixed number of guesses
  - □ Inconvenient to users if they’ re forgetful
  - □ Legitimate user would have to ask sys admin to unlock or reset their password
  - □ Potential for DoS attacks if usernames compromised and attacker guesses randomly for all, locking up large percentage of users of system



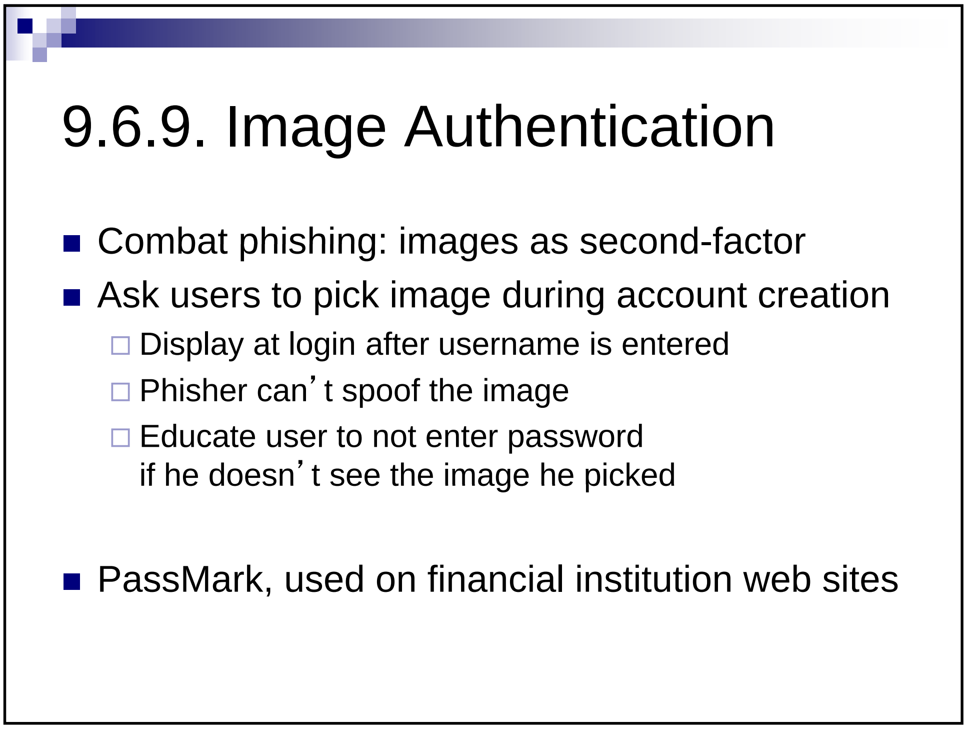
9.6.7 Artificial Delays

- Artificial delay when user tries login over network
- Wait 2^n seconds after *n*th failure from particular IP address
  - □ Only minor inconvenience to users (it should only take them a couple of tries, 10 seconds delay at most)
  - □ But makes attacker's guesses more costly, decreases number of guesses they can try in fixed time interval
- HTTP Proxies can be problematic
  - □ One user mistyping password may delay another user
  - □ Need more sophisticated way to delay



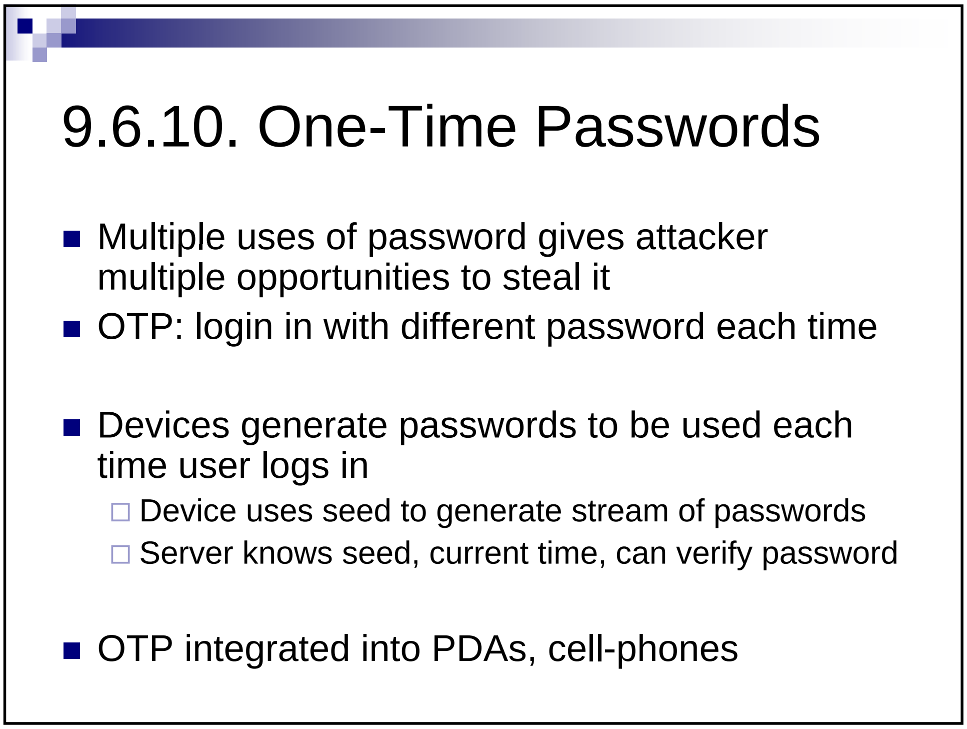
9.6.8. Last Login

- Notify user of last login date, time, location each time they login
  - □ Educate them to pay attention
  - □ Tell user to report any inconsistencies
- Discrepancies = indications of attacks
- Catch attacks that may not have been noticed
  - □ Ex: Alice usually logs in monthly from CA
  - □ Last login was 2 weeks ago in Russia
  - □ Alice knows something's wrong, reports it



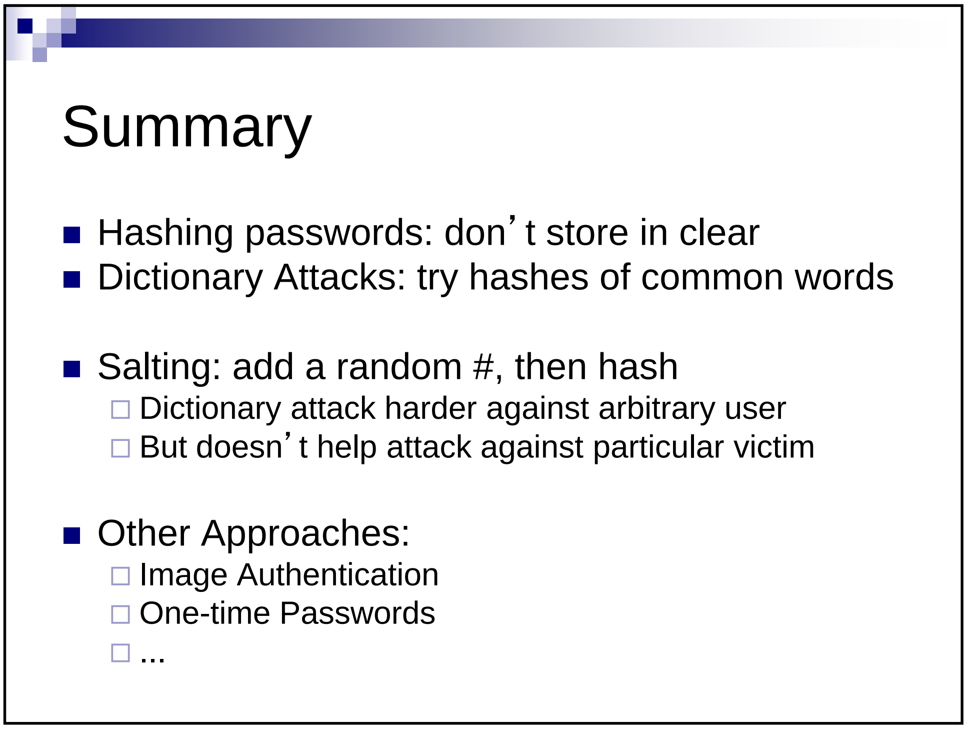
9.6.9. Image Authentication

- Combat phishing: images as second-factor
- Ask users to pick image during account creation
  - □ Display at login after username is entered
  - □ Phisher can't spoof the image
  - □ Educate user to not enter password if he doesn't see the image he picked
- PassMark, used on financial institution web sites



9.6.10. One-Time Passwords

- Multiple uses of password gives attacker multiple opportunities to steal it
- OTP: login in with different password each time
- Devices generate passwords to be used each time user logs in
  - □ Device uses seed to generate stream of passwords
  - □ Server knows seed, current time, can verify password
- OTP integrated into PDAs, cell-phones



Summary

- Hashing passwords: don’t store in clear
- Dictionary Attacks: try hashes of common words

- Salting: add a random #, then hash
  - □ Dictionary attack harder against arbitrary user
  - □ But doesn’t help attack against particular victim

- Other Approaches:
  - □ Image Authentication
  - □ One-time Passwords
  - □ ...