CryptoWorks21

Fundamentals of Network Security

5. Access control • Authentication • Web Application Security

CryptoWorks21 • July 16, 2021

Dr Douglas Stebila

UNIVERSITY OF WATERLOO

https://www.douglas.stebila.ca/teaching/cryptoworks21

Fundamentals of Network Security

- Basics of Information Security
  - Security architecture and infrastructure; security goals (confidentiality, integrity, availability, and authenticity); threats/vulnerabilities/attacks; risk management
- Cryptographic Building Blocks
  - Symmetric crypto: ciphers (stream, block), hash functions, message authentication codes, pseudorandom functions
  - Public key crypto: public key encryption, digital signatures, key agreement
- Network Security Protocols & Standards
  - Overview of networking and PKI
  - Transport Layer Security (TLS) protocol
  - Overview: SSH, IPsec, Wireless (Tool: Wireshark)
- Offensive and defensive network security
  - Offensive: Pen-tester/attack sequence: reconnaissance; gaining access; maintaining access (Tool: nmap)
    - Supplemental material: denial of service attacks
  - Defensive: Firewalls and intrusion detection
- **Access Control & Authentication; Web Application Security**
  - **Access control: discretionary/mandatory/role-based; phases**
  - **Authentication: something you know/have/are/somewhere you are**
  - **Web security: cookies, SQL injection**
  - **Supplemental material: Passwords**

Network security vs. computer security

Network security	Computer security
Controlling access to network resources	Controlling access to computer resources
Awareness of services running on a network	Awareness of software running on a computer
Concerned about misconfigurations, violations of access policy	Concerned about misconfigurations, software bugs

But…

- Modern computers and applications are very network-dependent
- Modern network devices are small computers
- Web-based applications are widespread

Assignment 3

3a) Password hash cracking

- Use various techniques to crack password hashes
- Estimate the difficulty of password hash cracking

3b) 2-factor authentication

- Investigate 2-factor authentication options in an online service you use

Assignment 0
Downloading and installing VirtualBox and Kali Linux

https://www.douglas.stebila.ca/teaching/cryptoworks21/

ACCESS CONTROL

Access control

- Controlling or restricting the use of information assets or resources
  - Recall our security goals were all about actions by authorised/unauthorised users

Terminology

Subjects

- Entities requesting access to a resource
  - Examples: Person (User), Process, Device
- This is an active role:
  - Entity initiates access request and is user of information/resource

Objects

- Resources or entities which contain information
  - Examples: Disks, files, records, directories
- This is a passive role
  - Object is repository for information or the resources that a subject tries to access

Access control terminology

Access modes / permissions / rights

- Which actions a subject can perform on an object
  - Create
  - Read
  - Write: observe and alter
  - Execute: neither observe nor alter
  - Append: limited type of alteration
  - Search
  - Destroy

Owner

- In some approaches we distinguish the subject who created or has primary control of the object as the "owner" who gets to make decisions about who else can access it
- In other approaches we don't distinguish the owner

Common principles

- <u>Blacklists</u>: access generally **permitted** unless expressly forbidden
- <u>Whitelists</u>: access generally **forbidden** unless expressly permitted
- <u>Principle of least privilege</u>: restrict access to minimum needed to perform day-to-day job ("need to know principle")
- <u>Separation of duties</u>: for critical tasks, divide task into steps that must be performed by different entities

Access control process

1. **Policy administration:** privilege is allocated and administered
   a) **Define** the authorisation policy for subjects and objects
   b) **Distribute** access credentials/token to subject
   c) **Change/revoke** authorisation whenever necessary

2. **Policy enforcement:** privilege is required to gain access
   a) **Identify** the subject
   b) **Authenticate** subject
   c) **Check policy** and then grant access
   – Also need to monitor access

Types of access control policies

How will access control decisions be made?

Discretionary access control

- Decision at the discretion of some individual, possibly the information asset owner

Mandatory access control

- System wide set of rules applied

Role-based access control

- Access permissions based on the role of the individual, rather than the identity (user, administrator, student, etc)

Discretionary access control

- Access rights to an object or resource **are granted at the discretion of the owner**
  – For example, the security administrator, the owner of the resource, or the person who created the asset

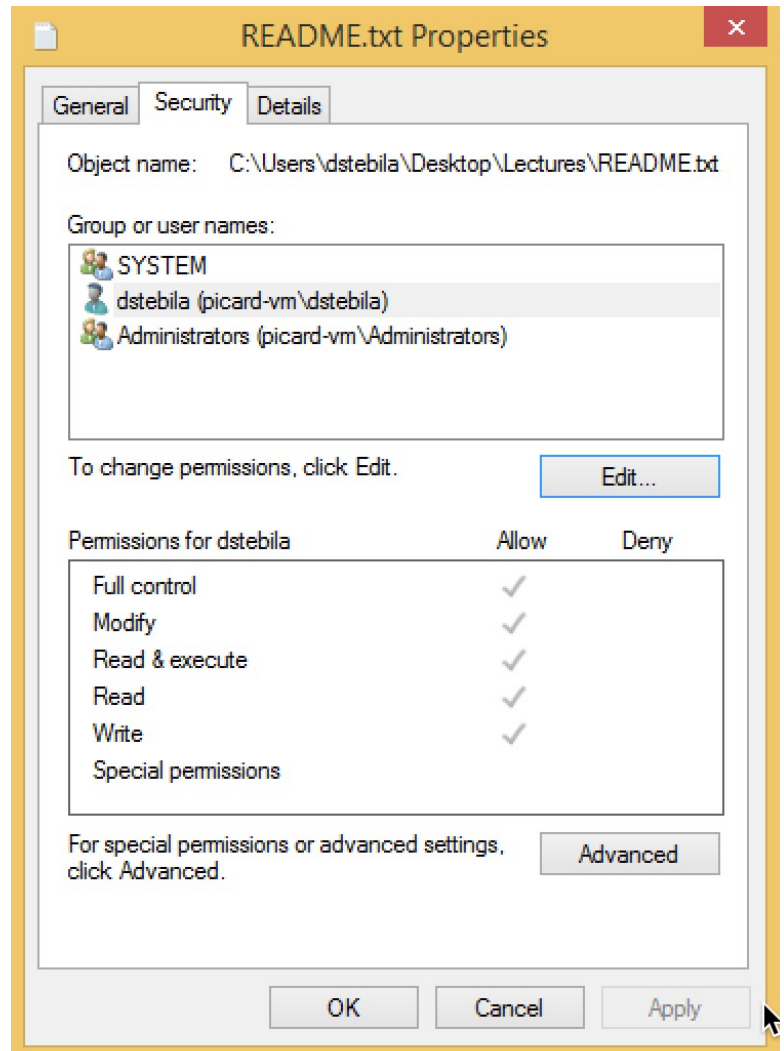
- Often implemented via **access control lists (ACLs)**
- Popular operating systems use DAC with access control lists.

Discretionary access control

In Windows 8:

- Right-click a file
- -> Properties
- -> Security

- ACL lists
  - groups or users with access permission
  - the type of permission granted



Discretionary access control

In Unix command line:

- `ls -l`
  - Object (file/directory) on each line
  - 3 groups of 3 letters
  - Permissions indicated for: Owner, Group and Other
  - Type of permissions: r read, w write and x execute

Lectures — fish /Users/dstebila/Dropbox/Documents/School/Teaching/CS 3IS3/Lectures — -fish — 103×14

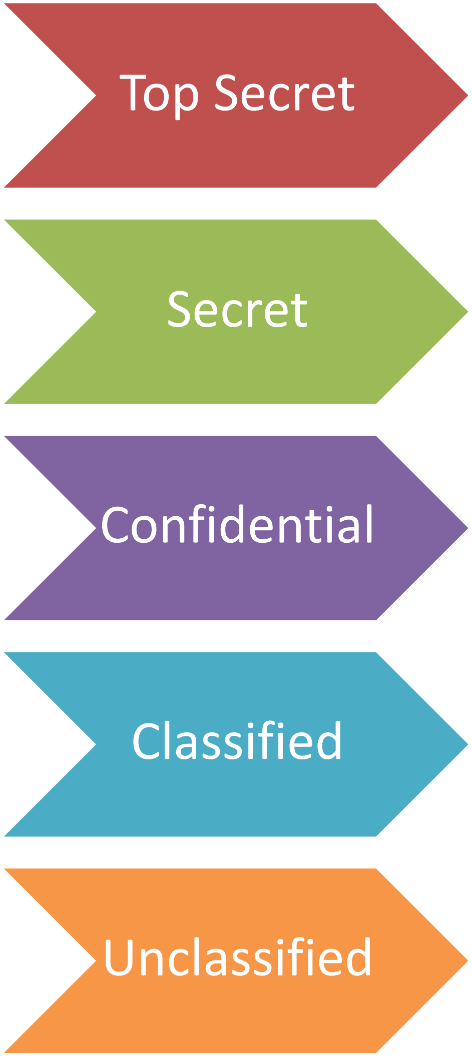
```
dstebila@picard ~/D/D/S/T/C/Lectures> ls -laF
total 23448
drwxr-xr-x@ 8 dstebila  staff      272  7 Sep 22:48 ./
drwxr-xr-x@ 9 dstebila  staff      306  7 Sep 22:40 ../
-rw-r-----@ 1 dstebila  staff   329947  7 Sep 22:06 Lecture 2 - Intro.pptx
-rw-r-----@ 1 dstebila  staff  1223608  7 Sep 22:41 Week 0 - Course information.pdf
-rw-r-----@ 1 dstebila  staff  3209378  7 Sep 22:41 Week 0 - Course information.pptx
-rw-r-----@ 1 dstebila  staff  2694557  4 Sep 21:26 Week 1 - Security principles.pdf
-rw-r-----@ 1 dstebila  staff  3120919  7 Sep 22:34 Week 1 - Security principles.pptx
-rw-r-----@ 1 dstebila  staff  1418159  7 Sep 23:29 Week 1-2 - Access Control.pptx
dstebila@picard ~/D/D/S/T/C/Lectures>
```

The owner (dstebila) can read/write, anyone in the group (staff) can read, other users cannot do anything.

Mandatory access control

- **A central authority assigns attributes to objects and to subjects**
- For example:
  - subjects assigned clearance levels,
  - objects assigned classification levels
- Have a system-wide set of rules relating attributes of the objects and subjects to the modes of access that are permitted
- MAC is mandatory in the sense that entities are not able to decide which other entities they want to allow to access resources, the system rules apply
  - the system denies users full control over access to the resources they create

Mandatory access control
Example – Security level hierarchy



Role-based access control

- Access rights are based on the **role of the subject**, rather than the subject's individual identity
- A **role** is some abstract collection of procedures that many subjects need to perform
  - Often associated to a job type
  - Examples:
    - In education: instructor, TA, student
    - In finance: approver, submitter, administrator
- A subject could have more than one role
  - Example: Tutor may be a student and also a staff member
  - But can only be acting in one role at any particular time
- More than one subject could have the same role
  - Example: Lots of students!

2) Policy enforcement

Identify the subject

Who are you claiming to be?

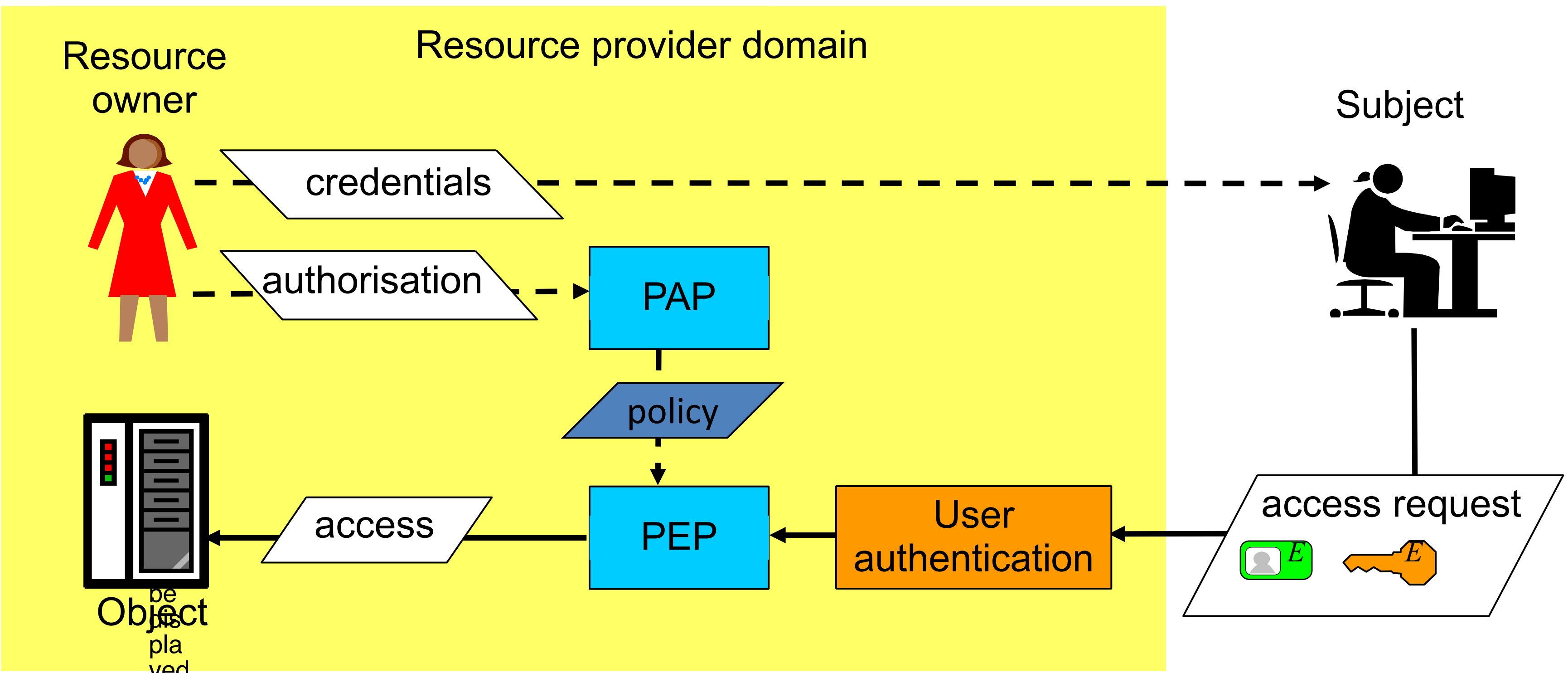
Authenticate the subject

Provide evidence that you are who you claim to be

Check policy then grant access

System checks that you are permitted to access resource in the manner requested, or prevents access if unauthorised

Access control process - conceptual diagram



Legend PAP: Policy Administration Point

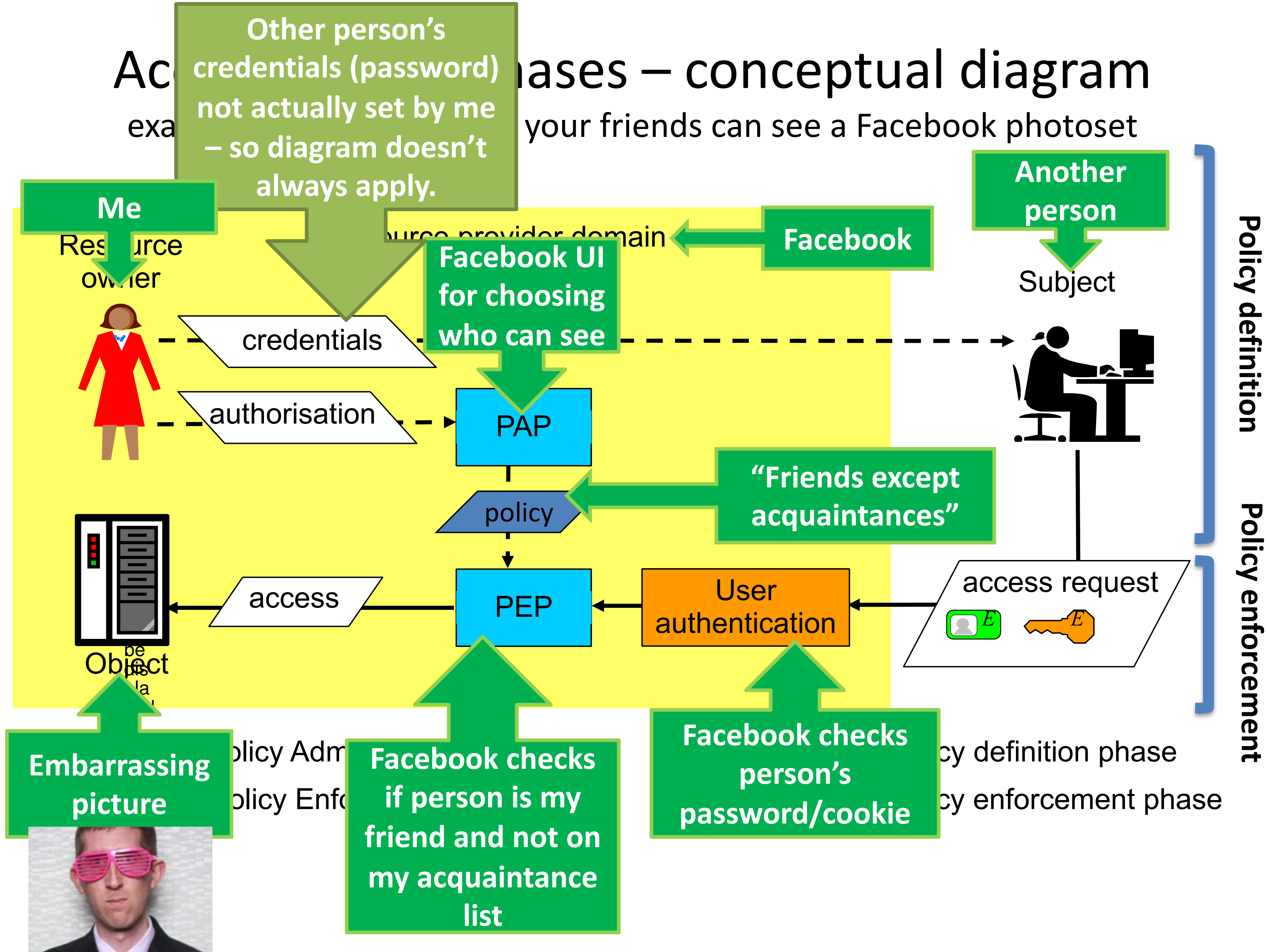
PEP: Policy Enforcement Point

- - - - ▸ AC policy definition phase

◂——— AC policy enforcement phase

Access control phases – conceptual diagram

example: limiting which of your friends can see a Facebook photoset


Ac ases – conceptual diagram
exa your friends can see a Facebook photoset
Other person's credentials (password) not actually set by me – so diagram doesn't always apply.
Me
Resource owner
Facebook
Another person
Subject
Facebook UI for choosing who can see
credentials
authorisation
PAP
policy
"Friends except acquaintances"
Policy definition
Policy enforcement
access request
User authentication
PEP
access
Object
Embarrassing picture
Facebook checks if person is my friend and not on my acquaintance list
Facebook checks person's password/cookie
olicy Adm
olicy Enf
cy definition phase
cy enforcement phase

AUTHENTICATION

Access control process

1. **Policy administration:** privilege is allocated and administered
   a) **Define** the authorisation policy for subjects and objects
   b) **Distribute** access credentials/token to subject
   c) **Change/revoke** authorisation whenever necessary

2. **Policy enforcement:** privilege is required to gain access
   a) **Identify** the subject
   b) **Authenticate** subject
   c) **Check policy** and then grant access
   – Also need to monitor access

User authentication

- Authenticators can be categorised as:
  - **Knowledge-Based** (Something you know)
  - **Object-Based** (Something you have)
  - **ID-Based** (Something you are)
  - **Location-based** (Somewhere you are)

- **Multi-factor authentication** uses combinations from multiple different categories of authenticators

Knowledge-based authentication: Something you know: passwords

- **Passwords** are human-memorizable strings that are used for authentication.

- **Threats** against passwords:
  - brute-force online/offline guessing
  - stealing the password
  - stealing a database of passwords (or password verifiers)
  - hard-coded passwords

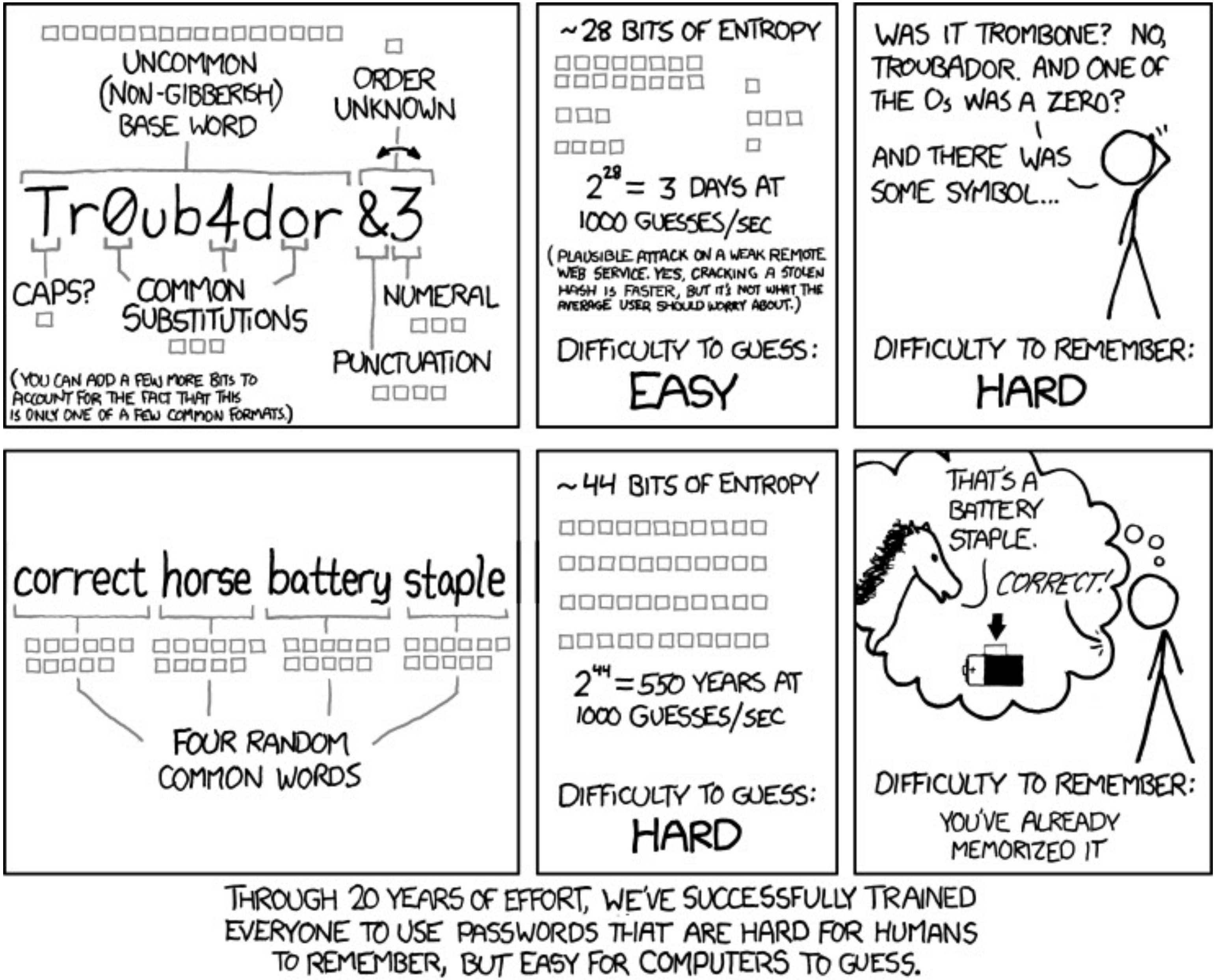
Generating passwords

User-selected passwords

- Use a 'strong' password
  - Aspects include minimum length, character set, prohibiting use of identifiers or known associated items as passwords, limitation on length of time before change required
- Store password securely
  - Not on a post-it note on your monitor (?)
- Don't share password with other entities
  - Colleagues, friends, family, etc.
- Don't use same password for multiple systems
  - Different unrelated passwords for work/study, online banking, social media, etc.

Computer-generated passwords

- Should be high entropy
- From a cryptographically strong source of (pseudo)randomness
- Challenges with usability



https://xkcd.com/936/

Storing passwords

- Server databases are regularly compromised.
- Good practices involve **not storing the raw password** in the database.
- Instead, store a **hash** of the password.
- Even better: store a **hash** of the password combined with a **salt.**

Hashing passwords

Instead of storing the user's password "123456", store the hash of the password:
SHA-1("123456")=7c4a8d09ca3762af61e59520943dc26494f8941b.

At login time:

- take the password the user typed,
- hash it,
- see if it matches the hash stored in the database.

Benefits:

- compromise of the database doesn't reveal the user's password
- almost no overhead for storage and login

Drawbacks:

- can't recover passwords for users who forget
- attackers could create a table of password hashes to compare against database
  - Demo

Salting

- We can defeat tables of hashes by **salting** the password:
  1. For each user, pick a random *k*-bit string, say *k*=80, called the **salt**.
  2. Store H(*salt*, *password*) and the *salt*.

- When the attempts to login with *password'*:
  1. Lookup the salt for that user.
  2. Compute H(*salt*, *password'*).
  3. See if it matches the stored hash value.

Password hardening

- You can slow down brute-force attacks even more by **hashing the password multiple times**.
- Instead of storing
  H(salt, password)
  store
  H(H(H(…H(salt, password)))
  with 10000 hash function applications.

- My computer can apply SHA1 3190046 times per second
- So 10000 times only takes in 0.003 seconds

- Doesn’t slow down login much.
- But it does slow down brute-force attacks by a factor of 10000.

- PBKDF2 (2000) (widely used; fairly secure); bcrypt; scrypt; Argon2 (2015) (best available approach)

Object-based authentication: something you have

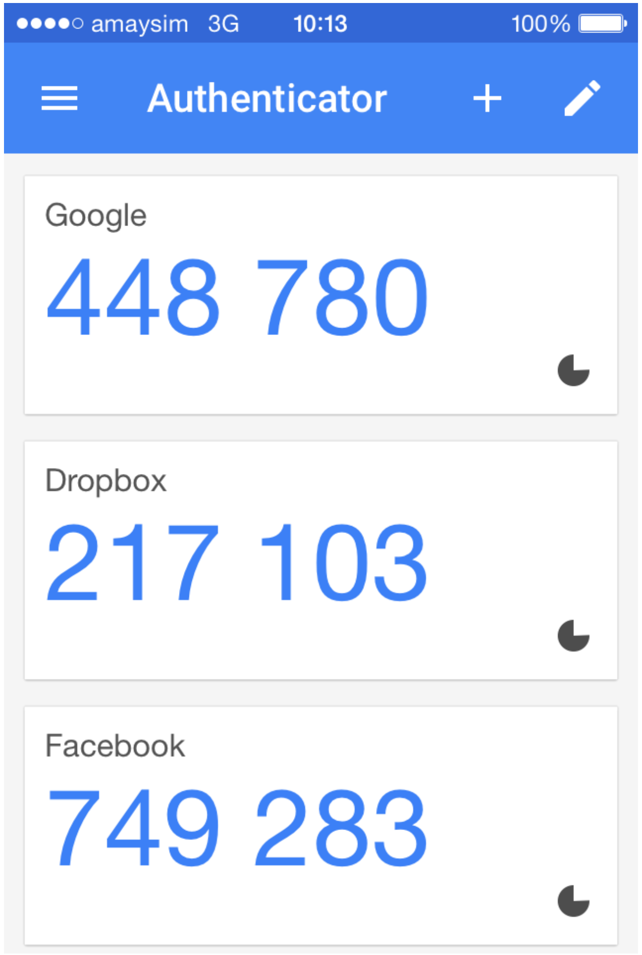
- Characterized by (exclusive) physical possession of a token.
- Examples:
  - Physical key
  - Magnetic swipe card
  - Phone that can receive SMS messages (?)
  - Token used for generating access codes
- Advantages:
  - Difficult to share (effort required to make a copy)
  - If lost, the owner may realise - sees evidence of the loss
- Disadvantages:
  - If lost, the finder can make use of the token

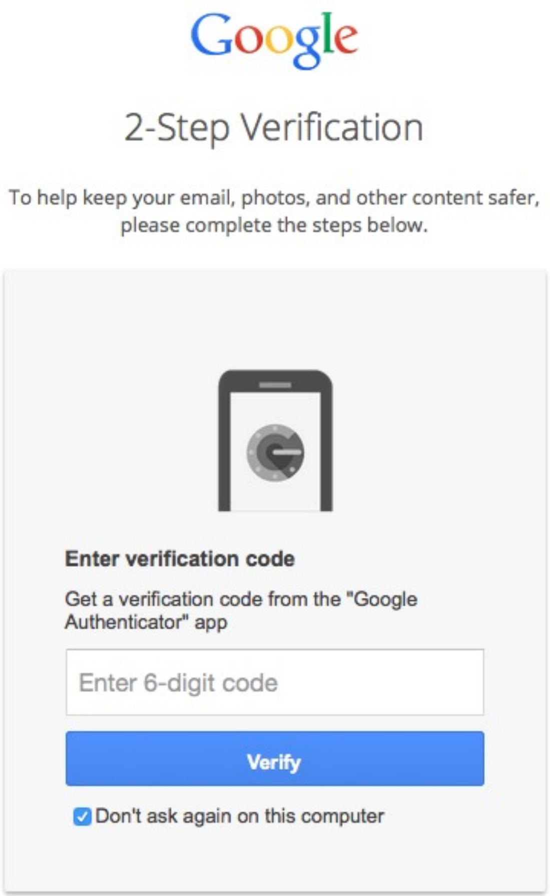
One-time password tokens

- Physical device that generates a sequence of one-time passwords
- Need to have password generators in the token and at the host system that are synchronized to produce the same sequence of random passwords
- Two general methods:
  - Clock-based tokens
  - Counter-based tokens

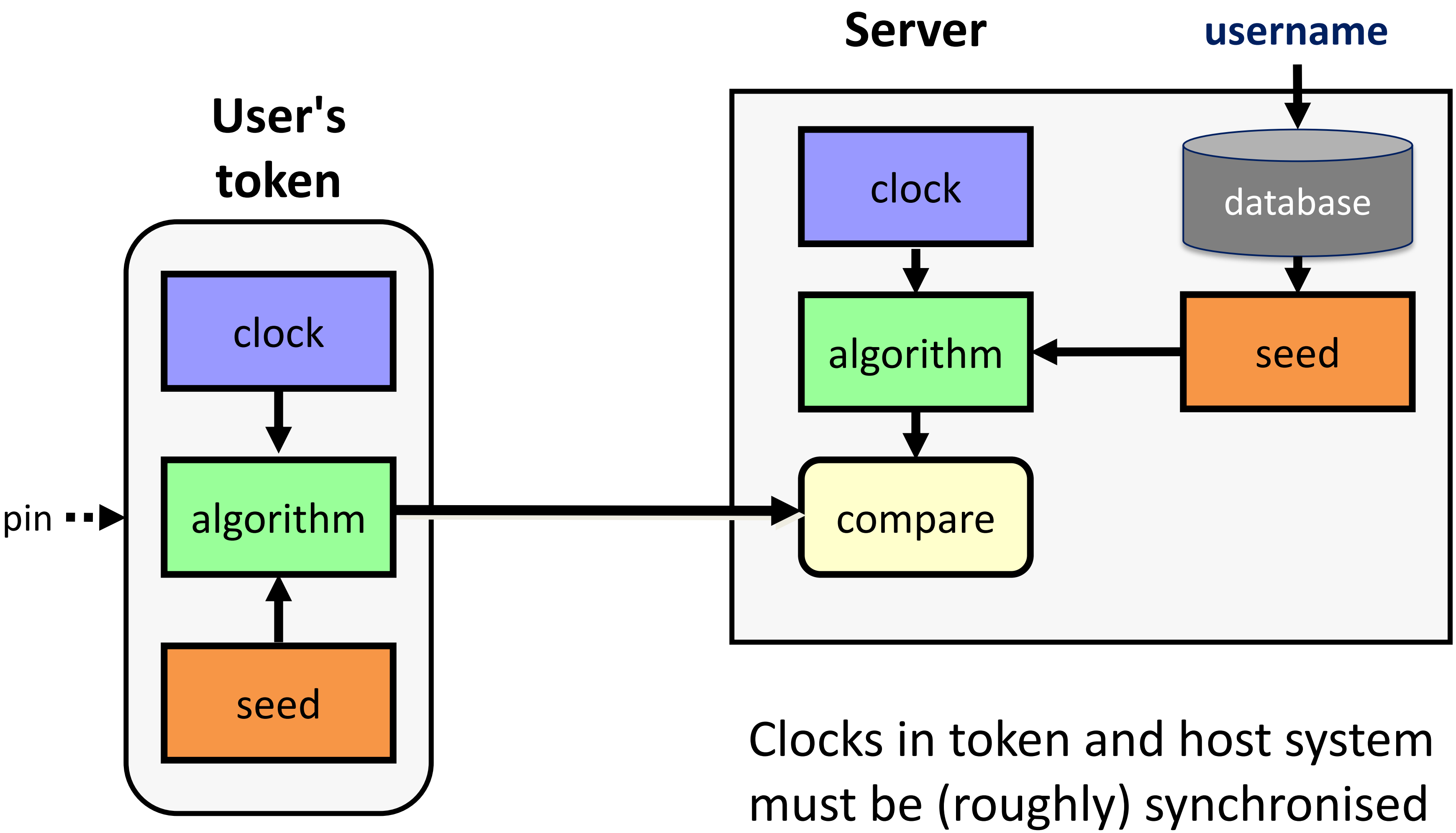
Clock-based tokens – TOTP

Time-based One-Time Password (RFC 6238)





Clock-based one-time tokens



Clocks in token and host system must be (roughly) synchronised

Phones as "something you have"

Idea for one-time passwords:

1. Register your mobile phone number with the server
2. Server sends you a text message with a one-time password
3. Use that one-time password during login

Is this secure?

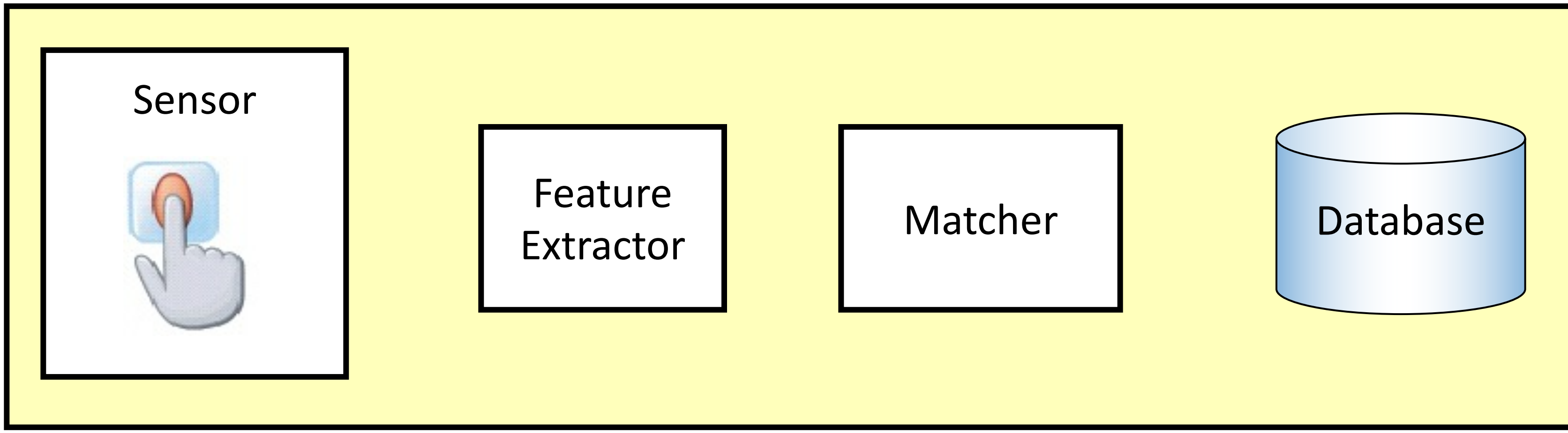
- Yes
  - Only you have your phone (or it's locked) so no one else can access the OTP you received
- No
  - Can an attacker change the mobile number associated with your account?
  - Can an attacker change the SIM card associated with your mobile number?
  - Can SMS messages be intercepted?

ID-based authentication: something you are

- Characterized by uniqueness to one person.
- Examples:
  - **Biometrics** such as fingerprint, eye scan, face scan, voiceprint, signature

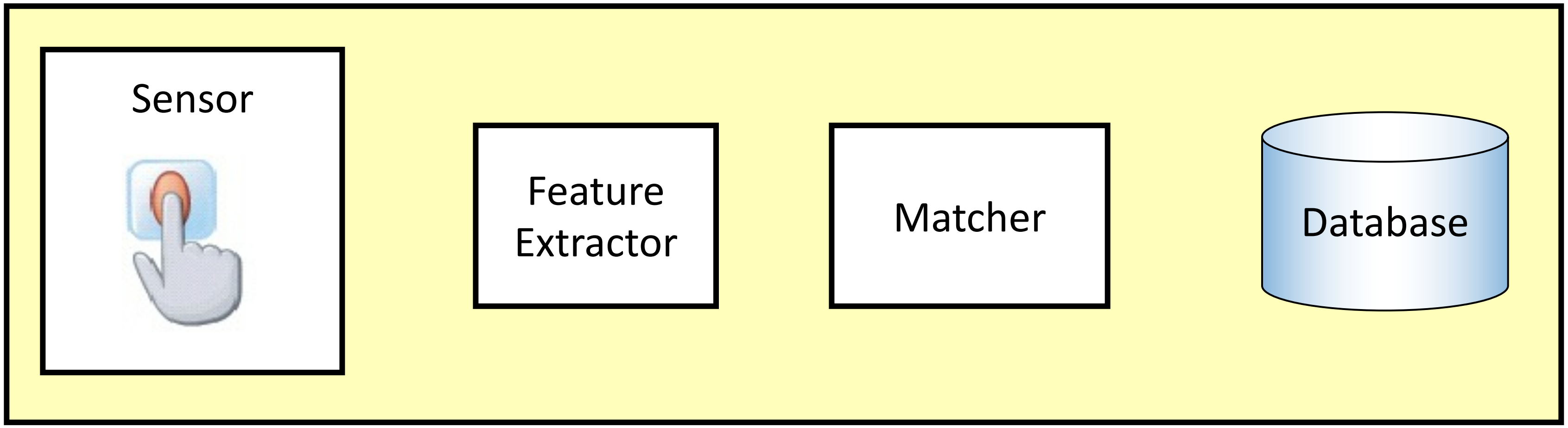
- Advantages:
  - Characteristic can't be forgotten or lost
  - May be difficult to copy, share or distribute
  - Should require the person being authenticated to be present at the time and point of authentication
- Disadvantages:
  - Harder to replace a compromised biometric authenticator, than to replace passwords or tokens

Biometric authentication systems



- Sensor: captures readings of the biometric signal of an individual
  - Example: camera that reads a fingerprint
- Feature extractor: processes the acquired biometric signal to extract a set of discriminatory features
  - Example: software that extracts positions and lengths of ridges and whorls from a fingerprint image

Biometric authentication systems

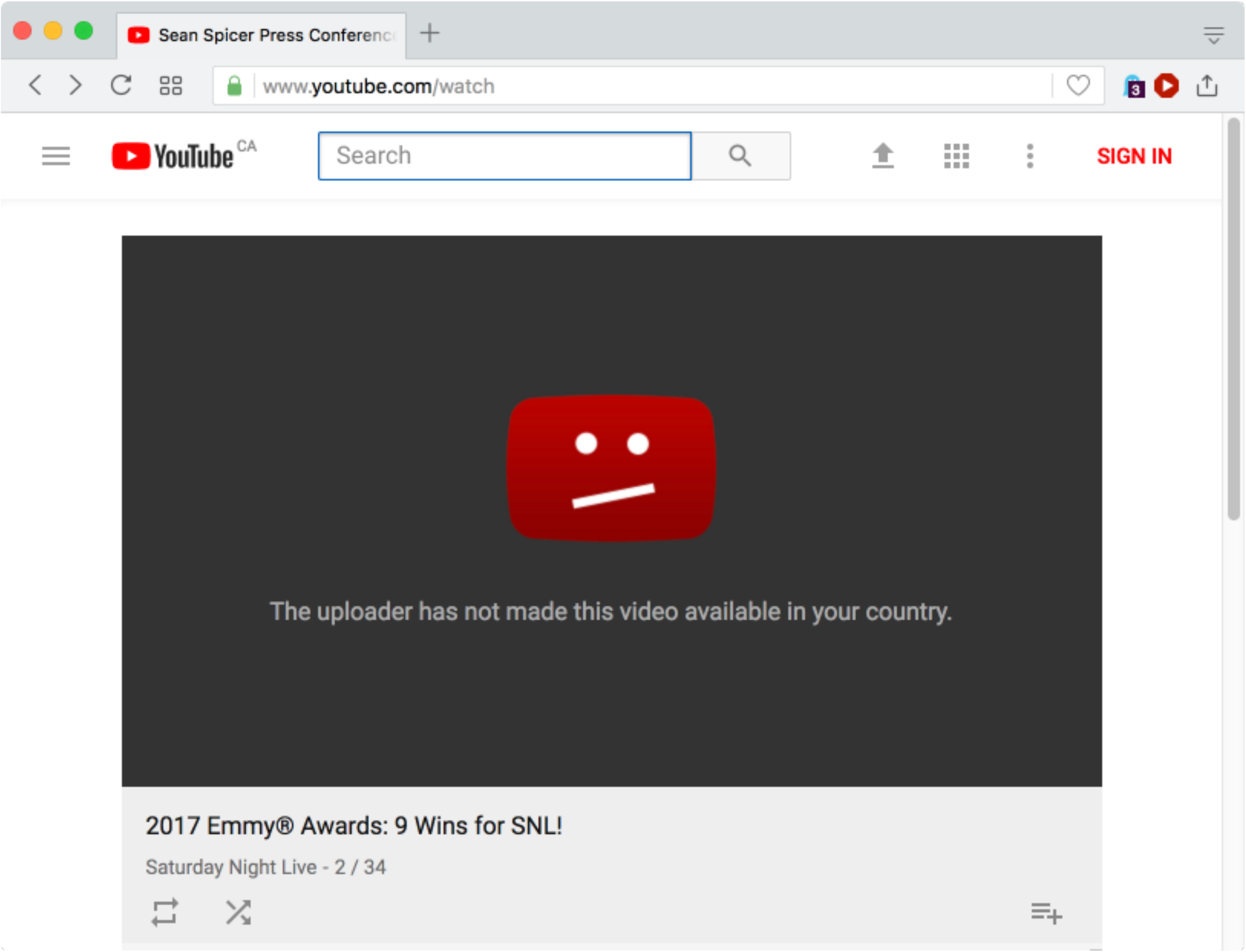


- Database: stores biometric template(s) for each user
- Matcher: compares values captured during identification/verification with values stored during enrolment

Location-based authentication: somewhere you are

- Characterized by your location in space and/or time
- Examples:
  - Triangulation of cell-phone signals
  - GPS tracker
  - IP address -> database/range of IP addresses
  - Link location to time
  - Are you in the exam room?

Location-based authentication: geo-blocking



Assignment 3

3a) Password hash cracking

- Use various techniques to crack password hashes
- Estimate the difficulty of password hash cracking
- Read the supplemental material at the end of these slides

3b) 2-factor authentication

- Investigate 2-factor authentication options in an online service you use

Assignment 0
Downloading and installing VirtualBox and Kali Linux

https://www.douglas.stebila.ca/teaching/cryptoworks21/

Session management and cookies

SQL injection attacks

WEB APPLICATIONS

Why web application security?

- More and more applications are getting web-enabled or converted to web apps.

- Blocking traffic at network layer doesn't work as all traffic flows through port 80/443 (or what the web server is configured on)
- Firewalls don't filter application level traffic

OWASP Top Ten (2017 Edition)

http://www.owasp.org/

OWASP
The Open Web Application Security Project

My classification:

- Web-specific issue
- General programming issue
- General security management issue

A1: Injection	A2: Broken Authentication and Session Management
A3: Sensitive Data Exposure	A4: XML External Entity (XXE)
A5: Broken Access Control	A6: Security Misconfiguration
A7: Cross-Site Script (XSS)	A8: Insecure Deserialization
A9: Using Components with Known Vulnerabilities	A10: Insufficient Logging & Monitoring

https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf.pdf

IETF Internet Protocol suite

Layer	Examples
Application	web (HTTP, HTTPS) email (SMTP, POP3, IMAP) login (SSH, Telnet)
Transport	connection-oriented (TCP) connectionless (UDP)
Internet	addressing and routing: • IPv4, IPv6 control (ICMP) security (IPsec)
Link	packet framing (Ethernet) physical connection • WLAN (WEP, WPA) • ADSL • GSM/3G

Web applications

Web browsers

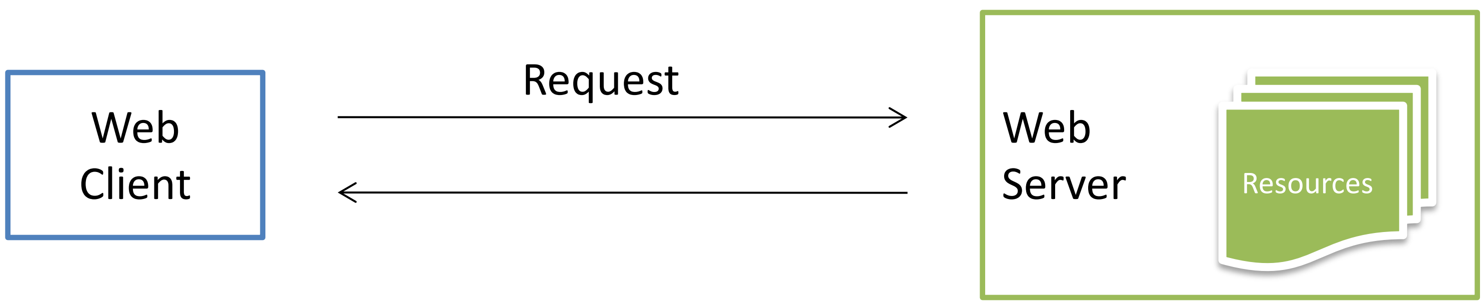
- HTML
  - forms
  - stylesheets
  - Javascript
  - ~~Flash, Java applets~~
- HTTP
  - cookies
- TLS (optional)
- TCP > IP > link layer

Web servers

- HTML
  - static files
  - CGI programs (PHP, Java, .Net, Ruby, Python, Javascript, Perl, …)
    - use XML web services
    - use SQL databases
- HTTP
  - cookies
- TLS (optional)
- TCP > IP > link layer

Hypertext Transport Protocol (HTTP)

- HTTP is a request/response protocol for communicating between web clients and web servers.
- A web client sends a request to a particular web server for a particular resource (identified by a URL) and the web server responds with some kind of data (often HTML data).



HTTP Request Message

- Given http://www.example.com/index.html
- Send TCP/IP message to www.example.com on port 80 containing the following:

GET /index.html HTTP/1.1	Request method and resource
Date: Mon 12 Jul 2021 21:12:55 GMT	General headers
Connection: close	
Host: www.example.com	Request headers
Accept: text/html, text/plain	
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36	

HTML forms: the GET method

- The GET method sends encoded data appended to the URL string.
- The data is separated from the URL by a '?'
- The encoded data and any path information are placed in the CGI environment variables QUERY_STRING and PATH_INFO.

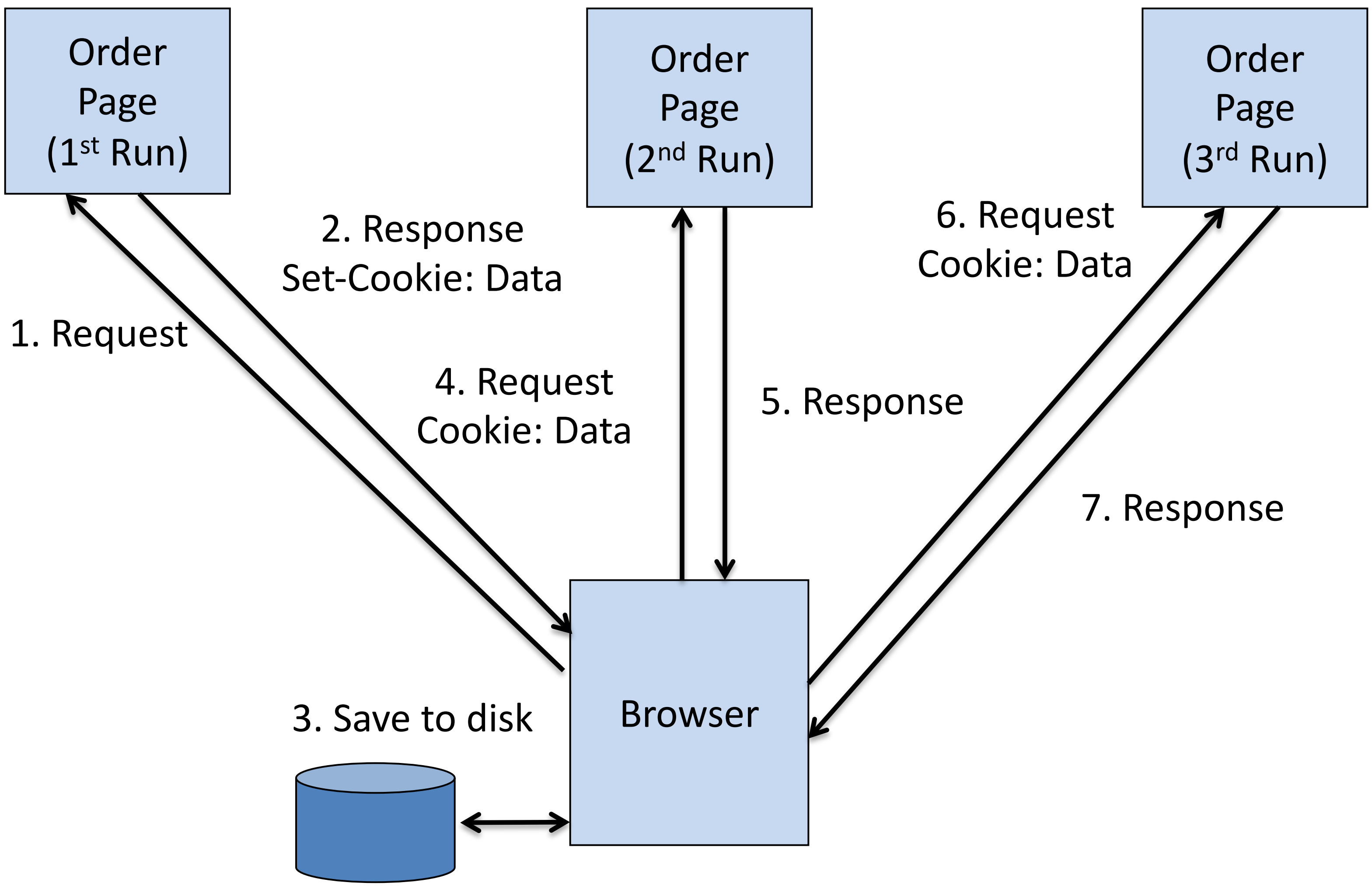
GET /cgi-bin/pizzaweb.cgi?order=152&delivery=Delivery&size=large&toppings=beef&toppings=pepperoni&Submit=Order+pizza HTTP/1.0
HOST: www.example.com
…

HTML forms: the POST method

- The POST method sends encoded data in the body section of the request.
- Data in the body is encoded in the same way as in the GET method.

POST /cgi-bin/pizzaweb.cgi HTTP/1.0
Host: www.example.com
Content-Length: 96
Content-Type: application/x-www-form-urlencoded
order=152&delivery=Delivery&size=large&toppings=beef&toppings=pepperoni&Submit=Order+pizza

Cookies



Order Page (1st Run)
Order Page (2nd Run)
Order Page (3rd Run)
1. Request
2. Response
Set-Cookie: Data
3. Save to disk
4. Request
Cookie: Data
5. Response
6. Request
Cookie: Data
7. Response
Browser

Sessions

- A sequence of requests and responses from one browser to one (or more) sites.
  - Session can be long or short:
    - Google advertising tracking: 1+ years
    - Google Mail login: 2 weeks

- Without sessions:
  - Users would have to constantly re-authenticate

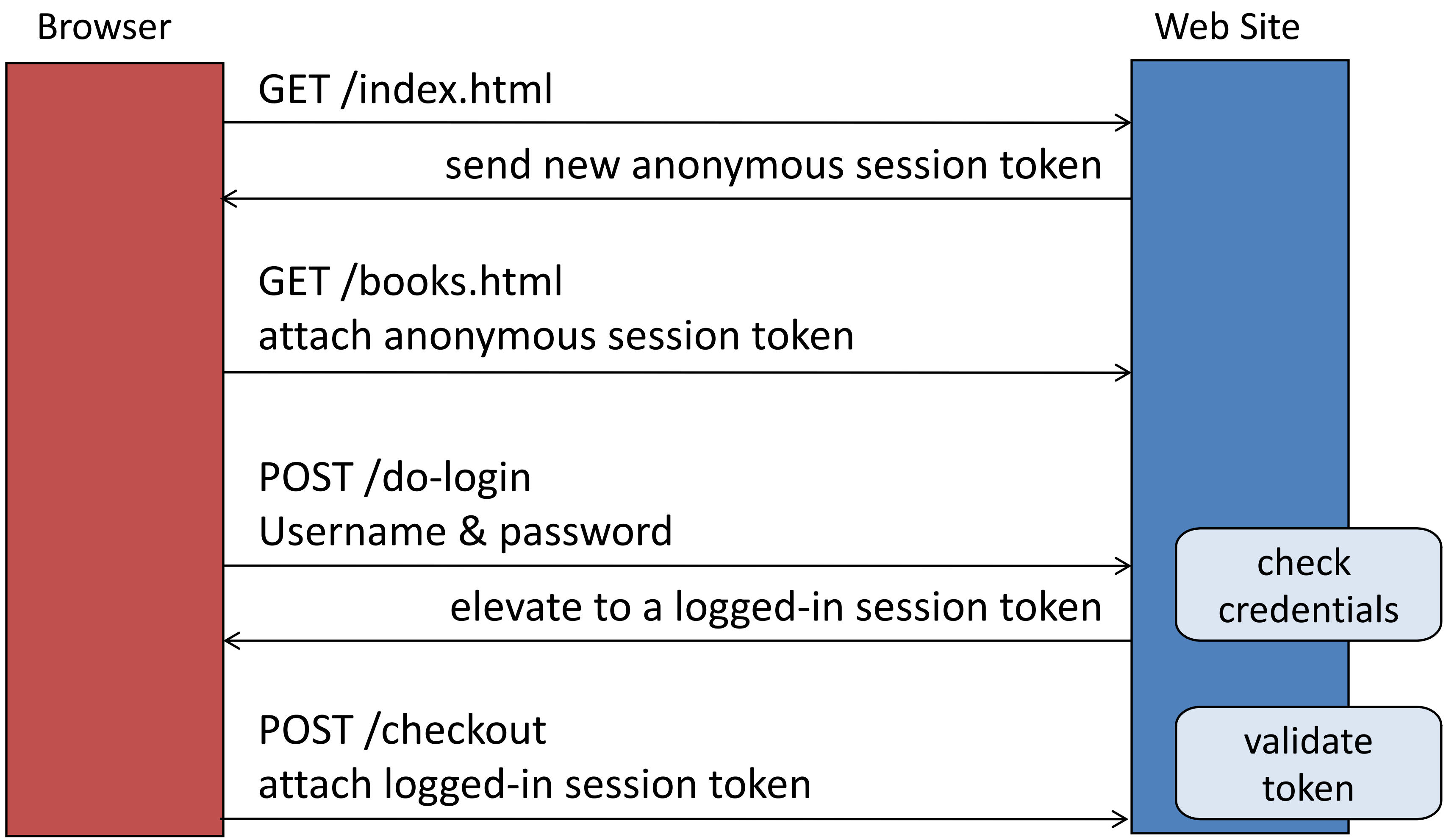
- With sessions:
  - Authenticate user once
  - All subsequent requests are tied to user

Session tokens

1. Server gives each user a random token when they first visit that website
   – Server stores link between user and token in a database

2. User re-sends that token every time they visit that website
   – Server looks up which user that token corresponds to

Session tokens



OWASP Top Ten (2017 Edition)

http://www.owasp.org/

OWASP
The Open Web Application Security Project

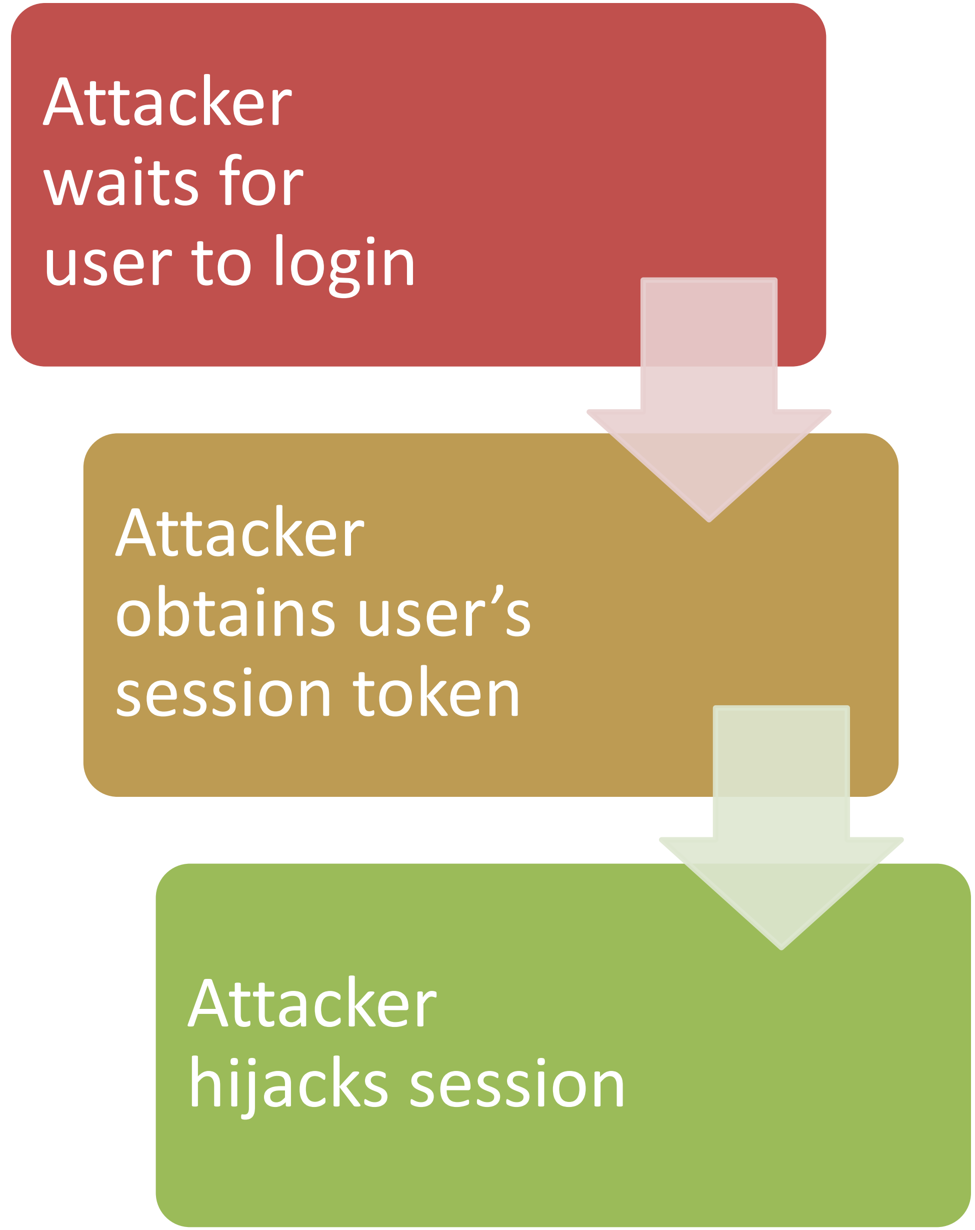
My classification:

- Web-specific issue
- General programming issue
- General security management issue

A1: Injection	A2: Broken Authentication and Session Management
A3: Sensitive Data Exposure	A4: XML External Entity (XXE)
A5: Broken Access Control	A6: Security Misconfiguration
A7: Cross-Site Script (XSS)	A8: Insecure Deserialization
A9: Using Components with Known Vulnerabilities	A10: Insufficient Logging & Monitoring

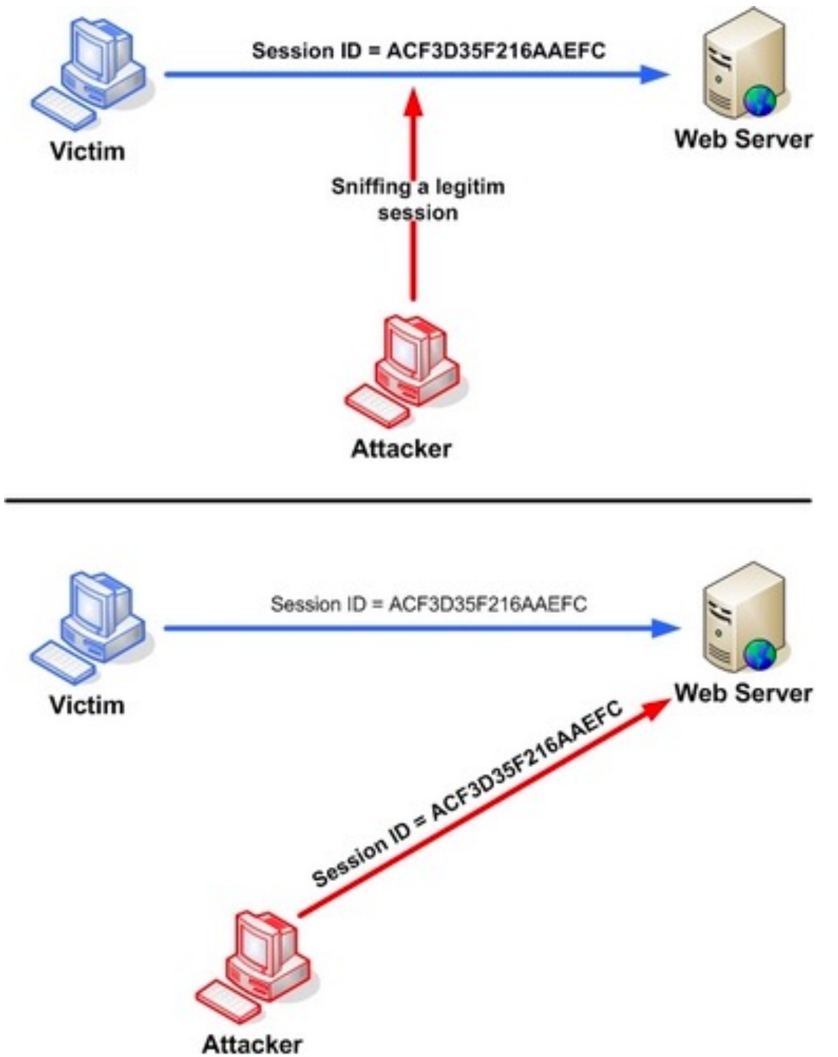
https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf.pdf

Session hijacking



Threats against session management

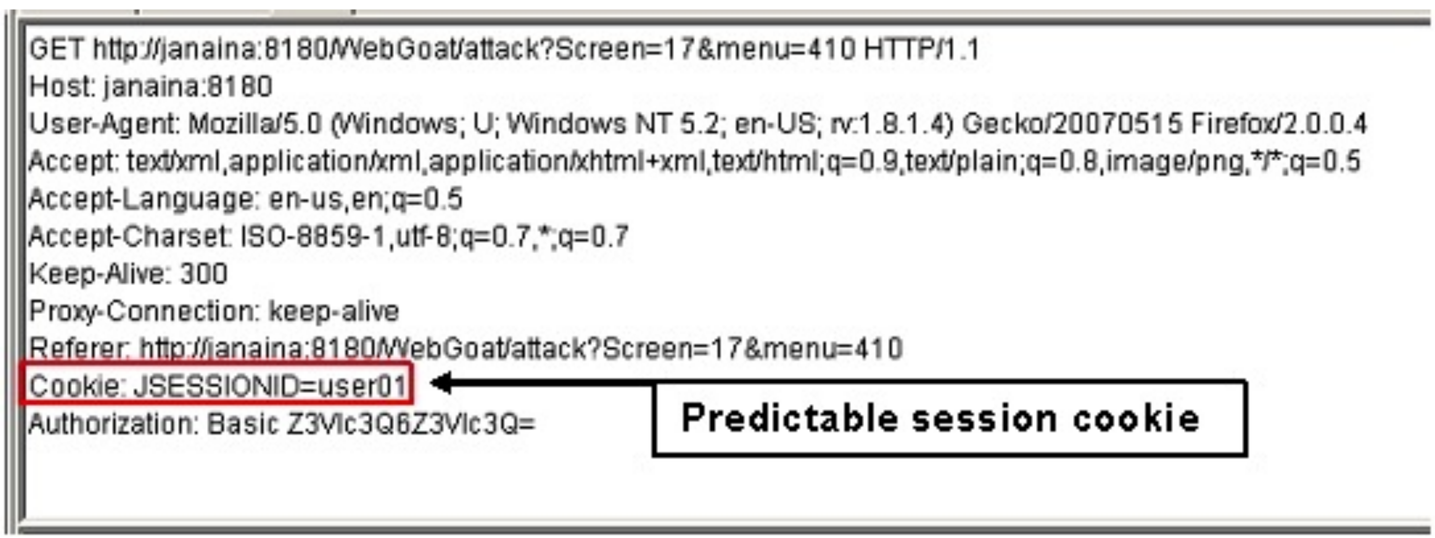
- **Session token stealing**:
  - Attacker can learn the session token somehow
    - If it's transmitted over an unencrypted connection
    - If it can be stolen through a cross-site scripting attack
    - If it can be stolen through malware
  - Attacker sends that value in its own headers to access the session



https://www.owasp.org/index.php/Session_hijacking_attack

Threats against session management

- **Session token prediction/guessing**:
  - Attacker can predict the value of the session token that a user will receive
  - Attacker sends that value in its own headers to access the session
  - Solution: use cryptographically-strong high-entropy session tokens



https://www.owasp.org/index.php/Session_Prediction

Sessions

- A sequence of requests and responses from one browser to one (or more) sites.

- Session can be long or short:
  - Google advertising tracking: 1+ years
  - Google Mail login: 2 weeks

- Without session management, users would have to constantly re-authenticate.
  - Authorize user once
  - All subsequent requests are tied to user
- Web application environments — ASP, PHP, etc. — or middleware provide session handling routines.

OWASP Top Ten (2017 Edition)

http://www.owasp.org/

OWASP
The Open Web Application Security Project

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Sensitive Data Exposure
- A4: XML External Entity (XXE)
- A5: Broken Access Control
- A6: Security Misconfiguration
- A7: Cross-Site Script (XSS)
- A8: Insecure Deserialization
- A9: Using Components with Known Vulnerabilities
- A10: Insufficient Logging & Monitoring

My classification:

- Web-specific issue
- General programming issue
- General security management issue

https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf.pdf

Injection attacks

- **Injection attack**: untrusted data is dynamically included in a command in an unsafe way
- Most common form:
  - SQL injection attacks on database commands
- Other examples:
  - shell command injection
  - Javascript eval() code injection

Shell command injection attacks

- Imagine a shell script allows a user to upload a file and rename the file

```
<?php

$tmpFilename =
    $_FILES["user_image"]["tmp_name"];

$userFilename =
    $_POST["user_filename"];

exec("mv $tmpFilename $userFilename");
```

- What happens if the user supplies a malicious filename?

  `my.jpg; rm -rf /`

- Command becomes:

  `mv tmp.jpg my.jpg; rm -rf /`

SQL: Structured Query Language

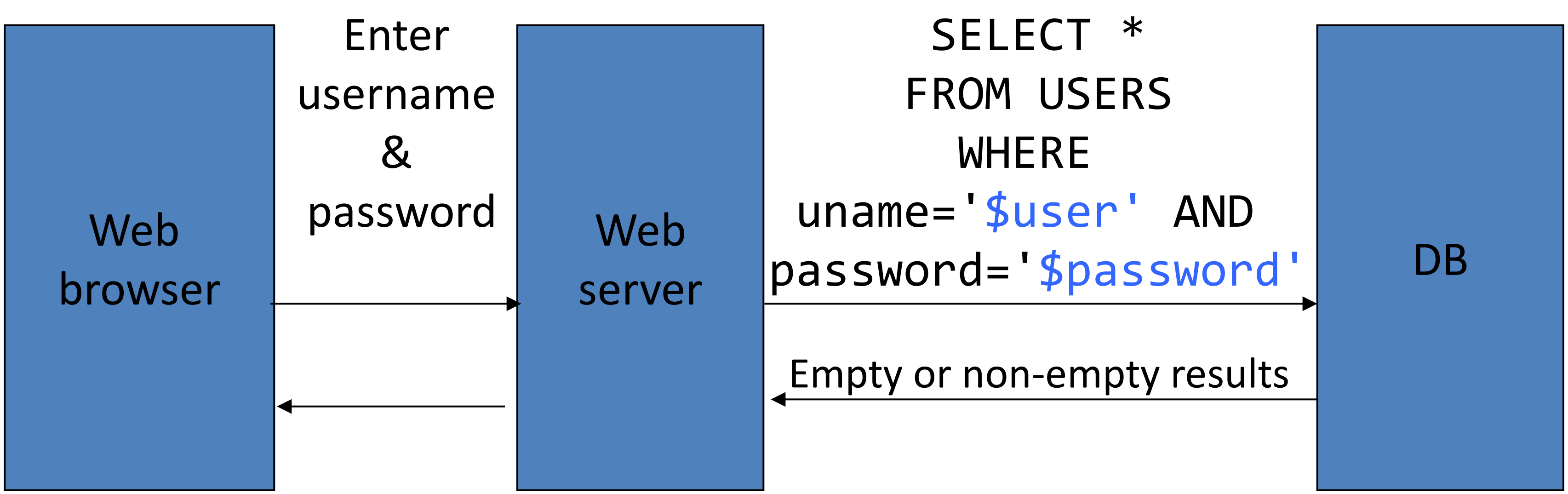
- Widely used database query language
- Fetch a set of records
  - `SELECT * FROM Person WHERE Username='Vitaly'`
- Add data to the table
  - `INSERT INTO Key (Username, Key) VALUES ('Vitaly', 3611BBFF)`
- Modify data
  - `UPDATE Keys SET Key=FA33452D WHERE PersonID=5`
- Query syntax (mostly) independent of vendor

SQL Injections

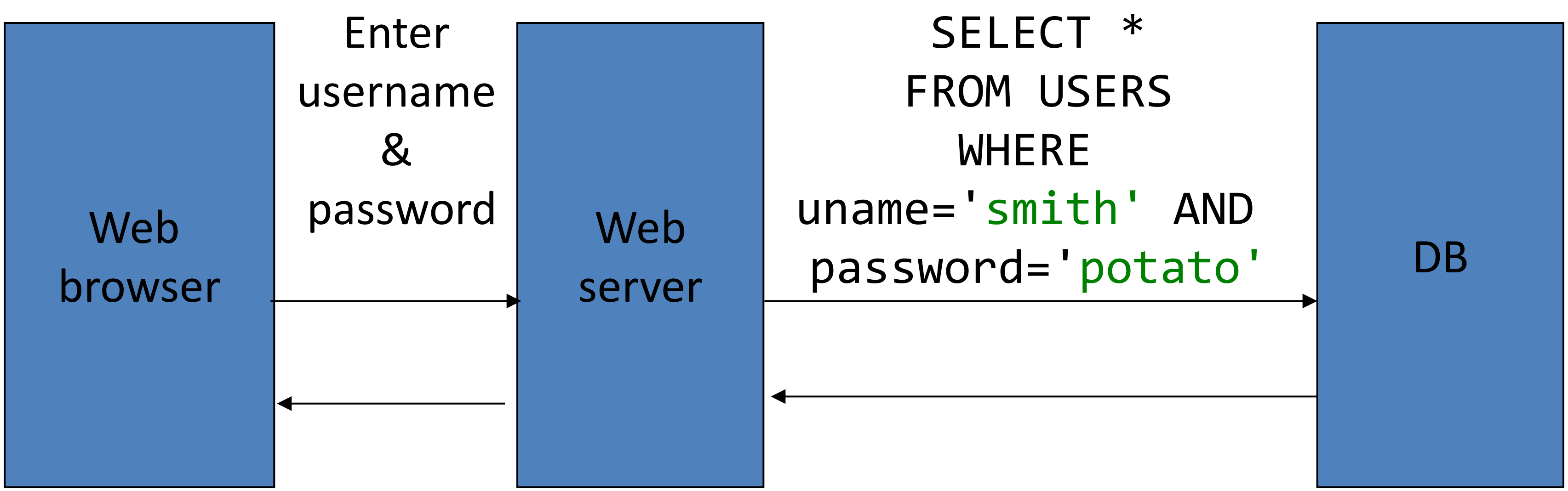
- Exploits vulnerabilities in how user input is inserted into database commands
- Attacker can execute arbitrary commands in the database

- Worse effects if the application uses an over-privileged account to connect to the database

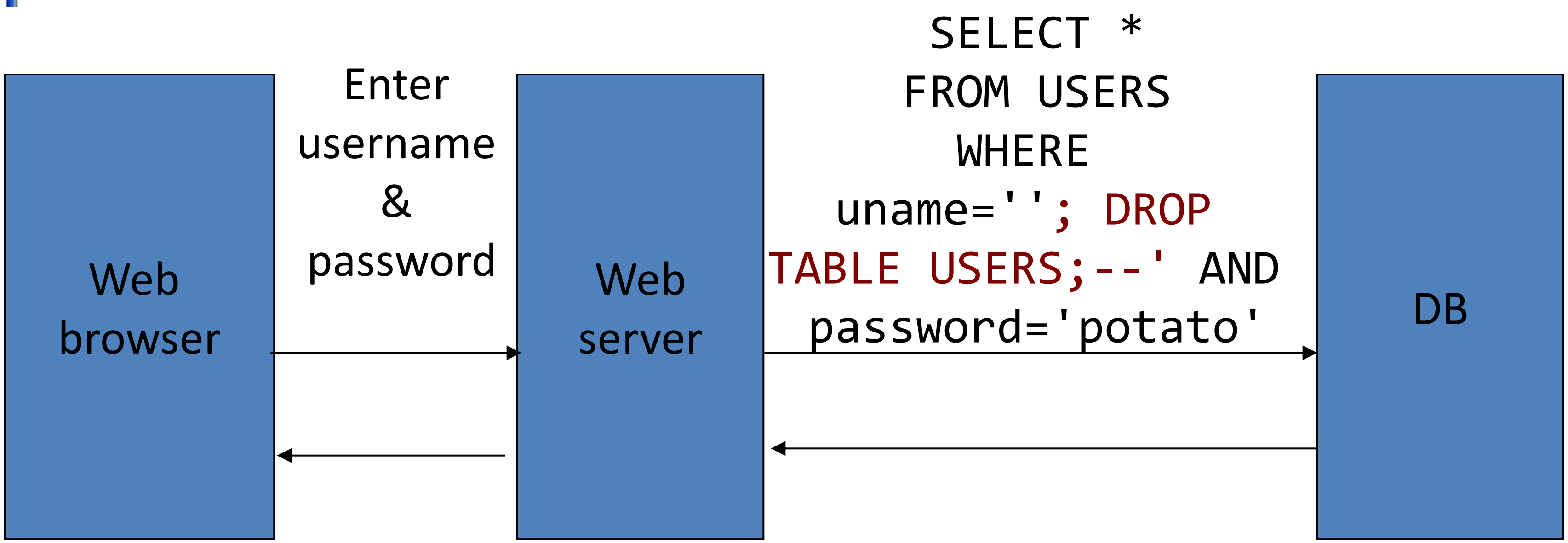
Web-based login sequence



Web-based login sequence

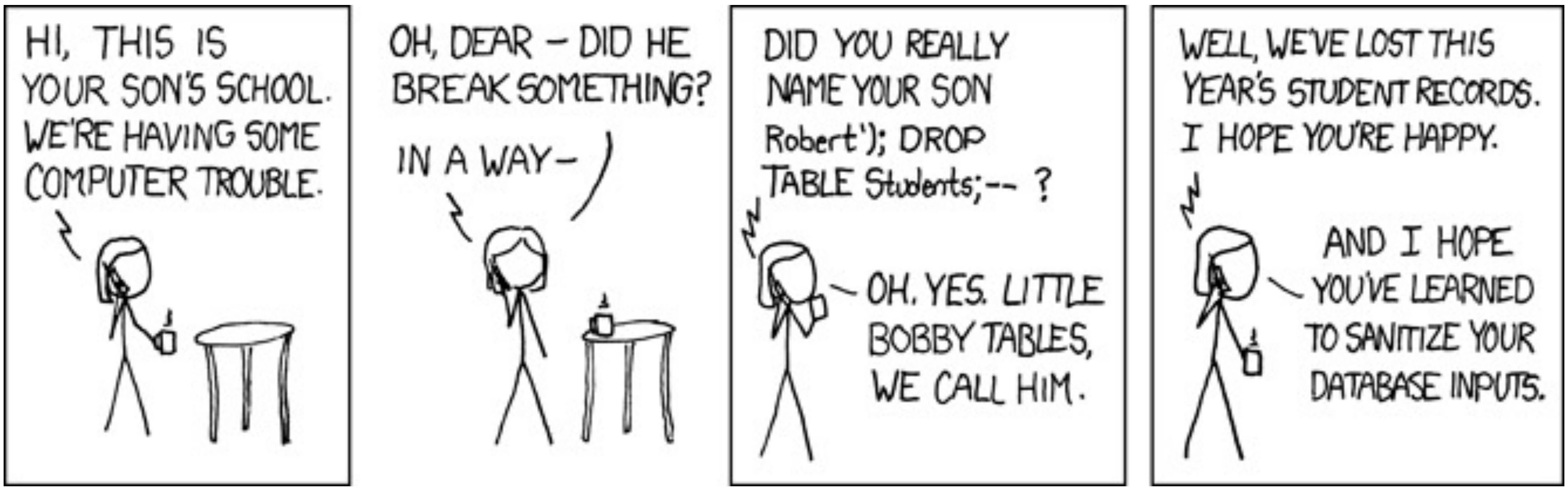


Web-based login sequence



Little Bobby Tables

http://xkcd.com/327/



Live example: IBM demo site

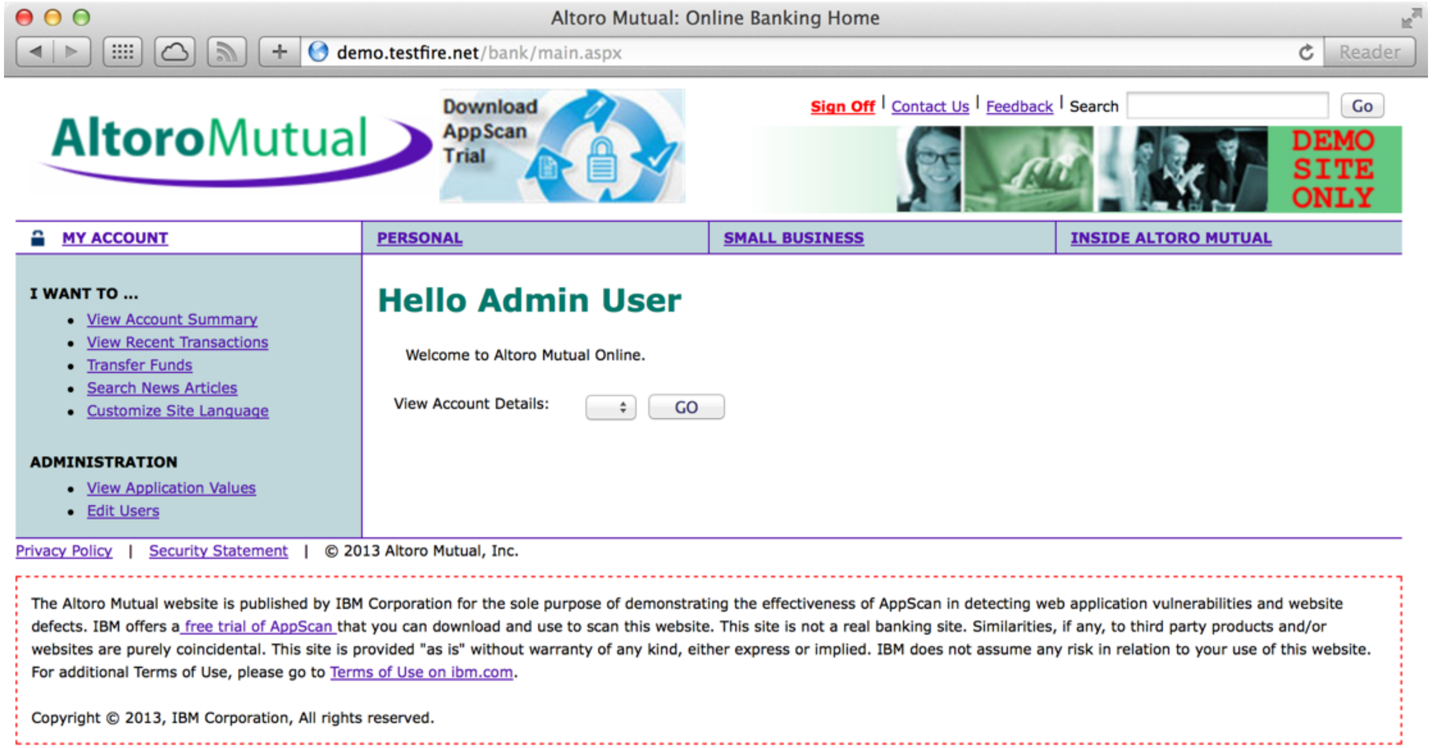
http://demo.testfire.net/

Reconnaissance: try to make an error

username: `alice` password: `123'456`

Break and enter:

username: admin password: ' OR 1=1;--



SQL injections can be used to...

Attack: using SQL injection to steal data from other databases

- User gives username

```
' AND 1=0 UNION SELECT cardholder,
number, exp_month, exp_year
FROM creditcards;--
```

- Results of two queries are combined.
- Empty table from the first query is displayed together with the entire contents of the credit card database.

Attack: using SQL injection to run shell commands

- User gives username

```
'; exec cmdshell  'net user badguy
badpwd';--
```

- Web server executes query

```
set UserFound=execute(
   "SELECT * FROM UserTable WHERE
    username= ''; exec cmdshell 'net
    user badguy badpwd';--...);
```

- Creates an account for badguy on DB server.

Attack: using SQL injections to create/modify accounts

- Create new users

```
'; INSERT INTO USERS
('uname','passwd')
VALUES ('hacker','38a74f');--
```

- Change email address (to run email-based password reset)

```
'; UPDATE USERS SET
email='hcker@root.org'
WHERE email='victim@yahoo.com';--
```

Example vulnerable PHP code

```
<?php
  $db = mysql_connect(“localhost”, “root”, “password”);
  mysql_select_db(“Shipping”, $db);
  $id = $HTTP_GET_VARS[“id”];
  $qry = “SELECT ccnum FROM cus WHERE id = $id”;
  $result = mysql_query($qry, $db);
  if ($result) {
    echo mysql_result($result, 0, “ccnum”);
  } else {
    echo “No result!” . Mysql_error();
  }
?>
```

Cause of SQL injection

- **Root cause: data is interpreted as command.**
- Control characters (such as ' in SQL) provide separation between data and commands.
- Any application that has the following pattern is at risk of SQL injection:
    1. Takes user input.
    2. Does not check user input for validity.
    3. Uses user input data to query a database.
    4. Use **string concatenation** or string replacement to build the SQL query or uses the SQL `exec` command.

Prevention techniques

- **Main idea:** need to stop control characters in data from being interpreted as delimiting commands.

- Approach 1: filter control characters
  - Hard to do reliably
  - Makes the O'Connor family sad

- Approach 2: escape control characters
  - E.g. replace O'Connor -> O\'Connor
  - Hard to do reliably

- Approach 3: use variable binding
  - Write "prepared statements" with placeholders
  - Use built-in subroutines that bind data to placeholders in a guaranteed safe way

Prepared statement: PHP

```
<?php
$db = ...;
$query = "SELECT email FROM users WHERE id=:id";
$stmt = $db->prepare($query);
$stmt->bind_param(":id", $_POST["id"]);
$stmt->execute();
…
```

This code **binds** values to the placeholders in the prepared statement

Fundamentals of Network Security

- Basics of Information Security
  - Security architecture and infrastructure; security goals (confidentiality, integrity, availability, and authenticity); threats/vulnerabilities/attacks; risk management
- Cryptographic Building Blocks
  - Symmetric crypto: ciphers (stream, block), hash functions, message authentication codes, pseudorandom functions
  - Public key crypto: public key encryption, digital signatures, key agreement
- Network Security Protocols & Standards
  - Overview of networking and PKI
  - Transport Layer Security (TLS) protocol
  - Overview: SSH, IPsec, Wireless (Tool: Wireshark)
- Offensive and defensive network security
  - Offensive: Pen-tester/attack sequence: reconnaissance; gaining access; maintaining access (Tool: nmap)
    - Supplemental material: denial of service attacks
  - Defensive: Firewalls and intrusion detection
- **Access Control & Authentication; Web Application Security**
  - **Access control: discretionary/mandatory/role-based; phases**
  - **Authentication: something you know/have/are/somewhere you are**
  - **Web security: cookies, SQL injection**
  - **Supplemental material: Passwords**

More aspects of network security

- Network equipment
- More web application vulnerabilities
- Honeypots
- Virtual Private Networks
- Content filters, anti-virus, proxies
- Denial of service resistance
- Security of outsourced/cloud resources

Fundamentals of Network Security

- Assessment:
  - 4 practical hands-on exercises with network and application security, with a few questions to submit from each
  - Due Thursday August 12

SUPPLEMENTAL MATERIAL: PASSWORDS

Knowledge-based authentication: **Something you know**

- Characterized by secrecy or obscurity
  - Should be something only that subject knows
- Commonly used:
  - Passwords:
    - Especially **user-selected reusable passwords**
  - Responses to questions:
    - your birth date, mother's maiden name, favourite food, pet's name, etc.
- Advantages include:
  - Readily accepted by users
  - Low cost implementation

Reusable passwords

- Most commonly used authentication mechanism
- User provides:
  - username or ID, and
  - password
- System has prior stored value to compare with
  - Successful provision of required value authenticates user to system
- Requirement: system must store the values used to verify the passwords for all system users

Passwords

- Passwords are human-memorizable strings that are used for authentication

Common attacks against passwords

- Attacker steals a password from a user (via malware, breaking kneecaps, …)
- Attacker guesses a user's password
  - Through online guessing
- Attacker steals a password database from a server
  - Then uses offline computation
- Hard-coded passwords

Security recommendations for passwords

- Use a ‘strong’ password
  - Aspects include minimum length, character set, prohibiting use of identifiers or known associated items as passwords, limitation on length of time before change required
- Store password securely
  - Not on a post-it note on your monitor (?)
- Don’t share password with other entities
  - Colleagues, friends, family, etc.
- Don’t use same password for multiple systems
  - Different unrelated passwords for work/study, online banking, social media, etc.

Strategies for selecting reusable passwords

User-selected

Computer-generated

User-selected reusable passwords

- Security policy should include:
  - User training
    - Explain importance of choosing ‘strong’ passwords.
  - Password selection guidelines
    - What are the characteristics of ‘good’ passwords?

- Unlikely to be effective in most organisations
  - Especially if large user population or high turnover of users.
  - Some users ignore guidelines, or can’t select ‘strong’ passwords.
  - Many choose passwords that are too short and very easy to guess.

RockYou.com password breach

- RockYou.com, a social media gaming site, had their password database compromised in 2009. Passwords were stored in plaintext.

- First large-scale password breach with publicly analyzed datasets

- # of accounts: 32.6 million
- # of different passwords: 14.3 million

RockYou.com password statistics

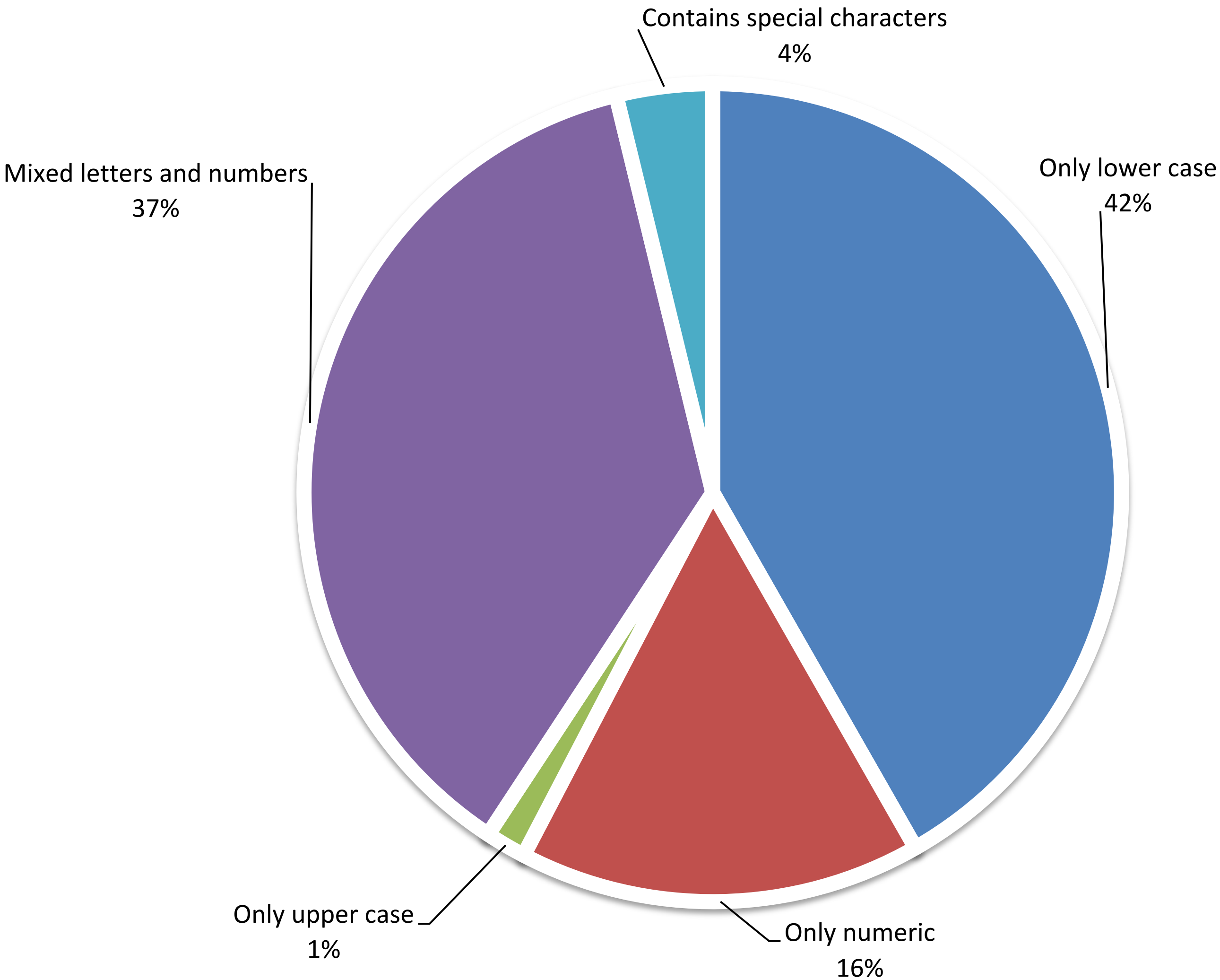
- About 30% of passwords length less than or equal to six characters.
- Nearly 50% of users used names, slang words, dictionary words or trivial passwords (consecutive digits, adjacent keyboard keys, and so on).
- Entropy of password set: 21.1 bits

Top 10 passwords:

1. 123456
2. 12345
3. 123456789
4. password
5. iloveyou
6. princess
7. 1234567
8. rockyou
9. 12345678
10. abc123

http://www.imperva.com/docs/WP_Consumer_Password_Worst_Practices.pdf

RockYou.com password statistics



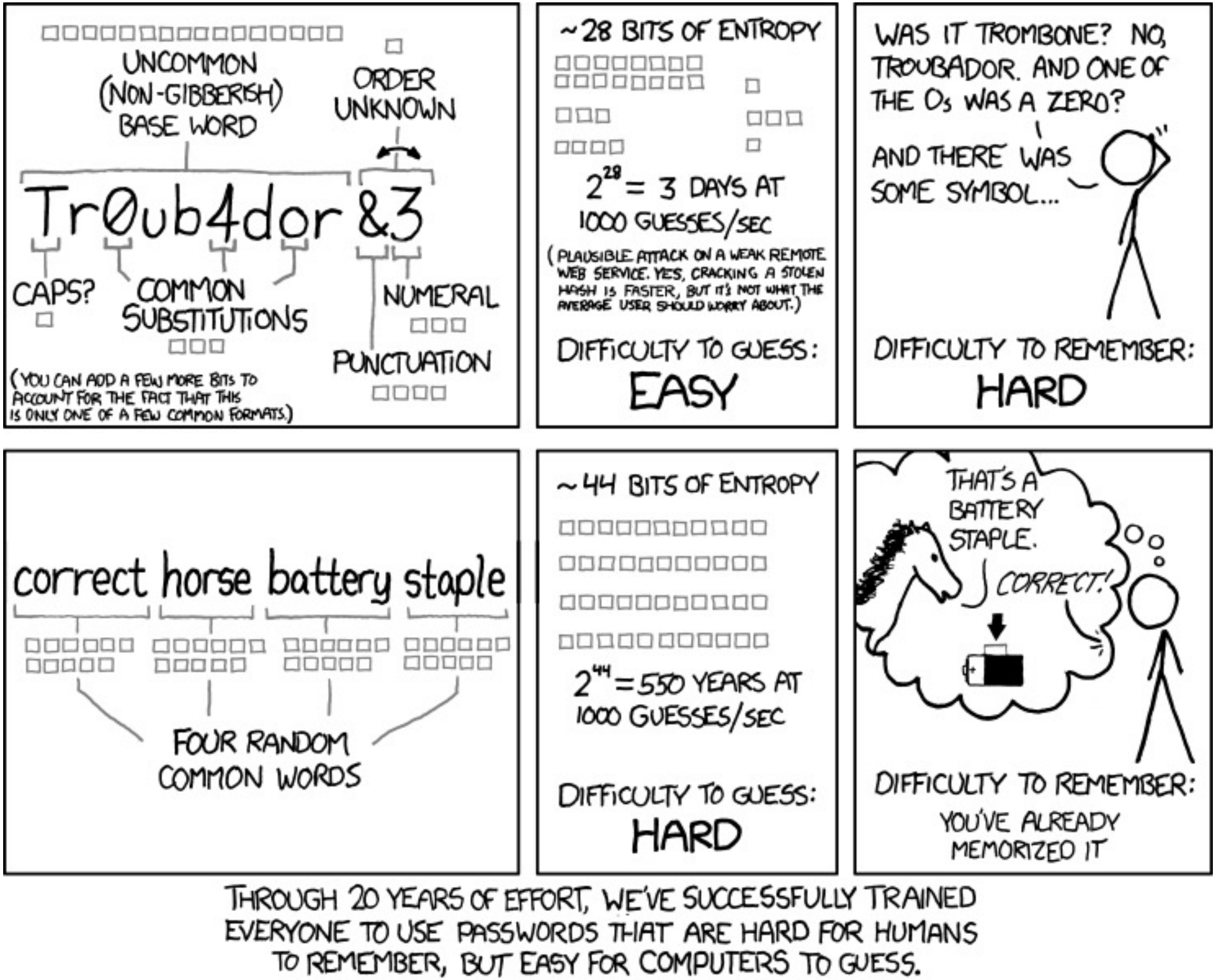
RockYou.com password statistics

- The top __ passwords covered __% of user accounts:
  - 1 0.9%
  - 5 1.7%
  - 10 2.1%
  - 100 4.6%
  - 1000 11.3%
  - 10000 22.3%

- An attacker could break into a random account in a single guess with probability around 2^{-13} (1 in 8000).

Computer-generated reusable passwords

- Computer generated passwords avoid the problem of users choosing weak passwords
- But have another security problem:
  - Passwords consisting of random characters difficult for users to remember, so they may write them down.
- Various mechanisms for generating human-friendly passwords:
  - Syllabic word-like: FIPS PUB 181 http://csrc.nist.gov/publications/fips/fips181/fips181.pdf
  - Sequences of words:
    - Diceware: http://world.std.com/~reinhold/diceware.html
    - xkcd: http://correcthorsebatterystaple.net, https://xkpasswd.net/s/



https://xkcd.com/936/

Entropy: a (somewhat okay) measure of password strength

- **Entropy** measures the uncertainty in values generated from a random process

- Think of passwords being generated from a random process with a certain distribution
- Predicts the number of guesses we have to make to learn the password

Not ideal measure of password guessing difficulty, but reasonable good (see http://jbonneau.com/doc/2012-jbonneau-phd_thesis.pdf for detailed analysis)

Entropy: a (somewhat okay) measure of password strength

- Suppose a process X generates n values x_1, …, x_n with probabilities p_1, …, p_n

- Formula for entropy of process X:

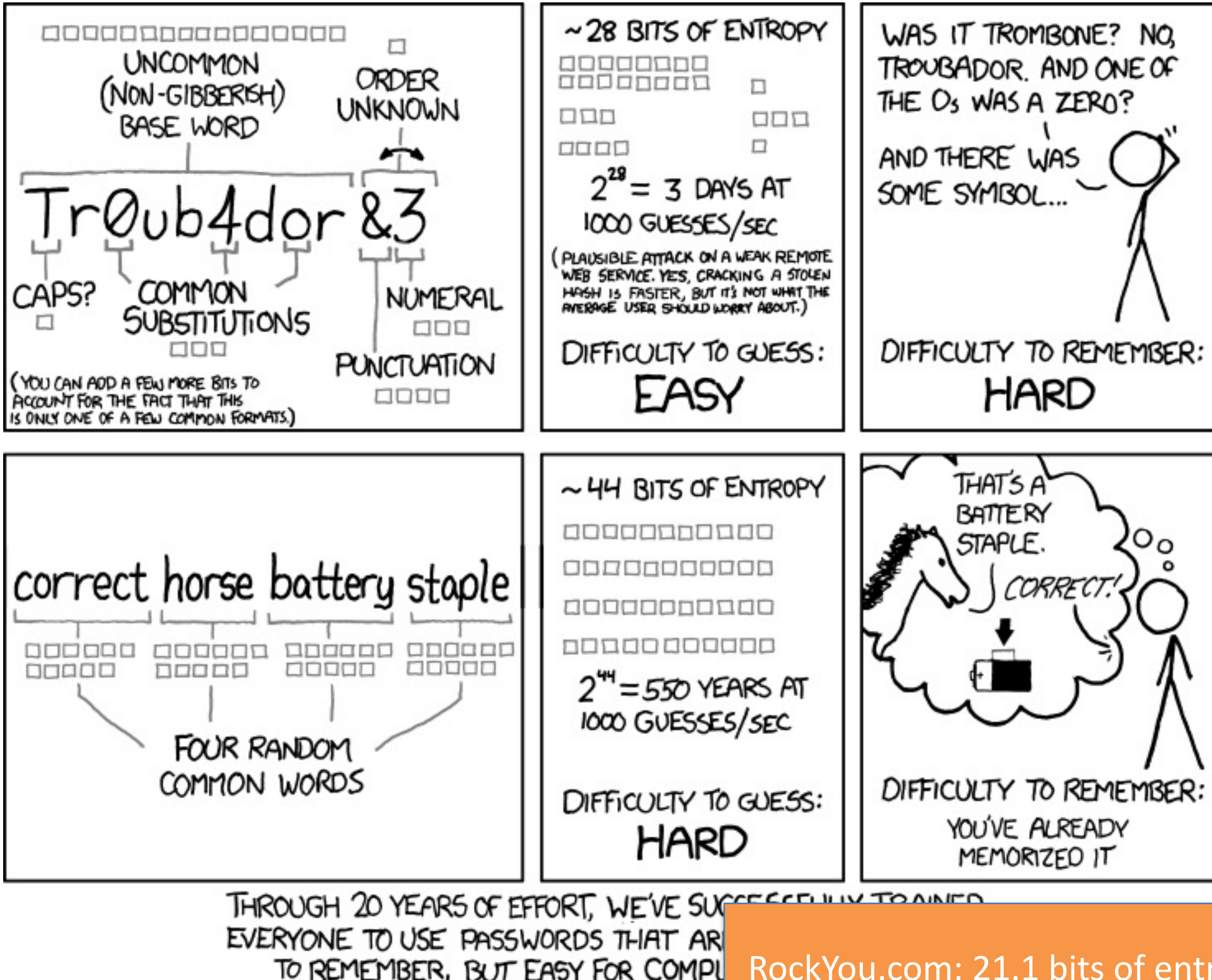
$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

  - Or alternatively:

$$H(X) = -p_1 \log_2(p_1) - p_2 \log_2(p_2) - \cdots - p_n \log_2(p_n)$$

Entropy: a (somewhat okay) measure of password strength

- Simple way of thinking about it:
  - If a password is chosen uniformly at random from a set of size 2^n,
  - then its entropy is n bits,
  - and we require around 2^{n-1} guesses on average to find it.



RockYou.com: 21.1 bits of entropy

https://xkcd.com/936/

Example: calculating entropy

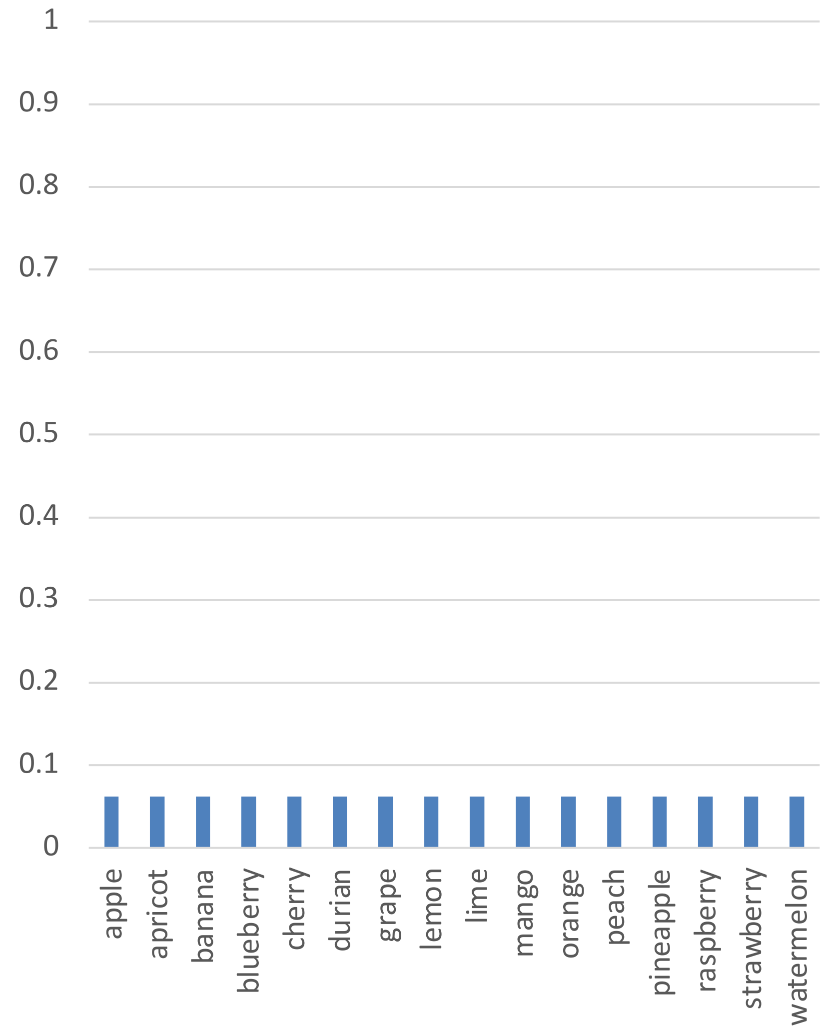
- Suppose we have a dictionary of 16 words.

- Scenario 1: Passwords generated uniformly at random from the dictionary
  - i.e., each password is equally likely

- Scenario 2: Passwords were NOT generated uniformly at random from the dictionary
  - i.e., some passwords more likely than others

Example: calculating entropy
Scenario 1: Equally likely passwords

Password (x_i)	Probability (p_i)
apple	1/16
apricot	1/16
banana	1/16
blueberry	1/16
cherry	1/16
durian	1/16
grape	1/16
lemon	1/16
lime	1/16
mango	1/16
orange	1/16
peach	1/16
pineapple	1/16
raspberry	1/16
strawberry	1/16
watermelon	1/16



Example: calculating entropy
Scenario 1: Equally likely passwords

Password (x_i)	Probability (p_i)
apple	1/16
apricot	1/16
banana	1/16
blueberry	1/16
cherry	1/16
durian	1/16
grape	1/16
lemon	1/16
lime	1/16
mango	1/16
orange	1/16
peach	1/16
pineapple	1/16
raspberry	1/16
strawberry	1/16
watermelon	1/16

$$
\begin{aligned}
H(X) &= -\sum_{i=1}^{16} p_i \log_2(p_i) \\
&= -\sum_{i=1}^{16} \frac{1}{16} \log_2\left(\frac{1}{16}\right) \\
&= -16 \cdot \frac{1}{16} \log_2\left(\frac{1}{16}\right) \\
&= -1 \cdot \log_2\left(\frac{1}{16}\right) \\
&= -1 \cdot \log_2\left(2^{-4}\right) \\
&= 4
\end{aligned}
$$

Example: calculating entropy
Scenario 1: Equally likely passwords

Password (x_i)	Probability (p_i)
apple	1/16
apricot	1/16
banana	1/16
blueberry	1/16
cherry	1/16
durian	1/16
grape	1/16
lemon	1/16
lime	1/16
mango	1/16
orange	1/16
peach	1/16
pineapple	1/16
raspberry	1/16
strawberry	1/16
watermelon	1/16

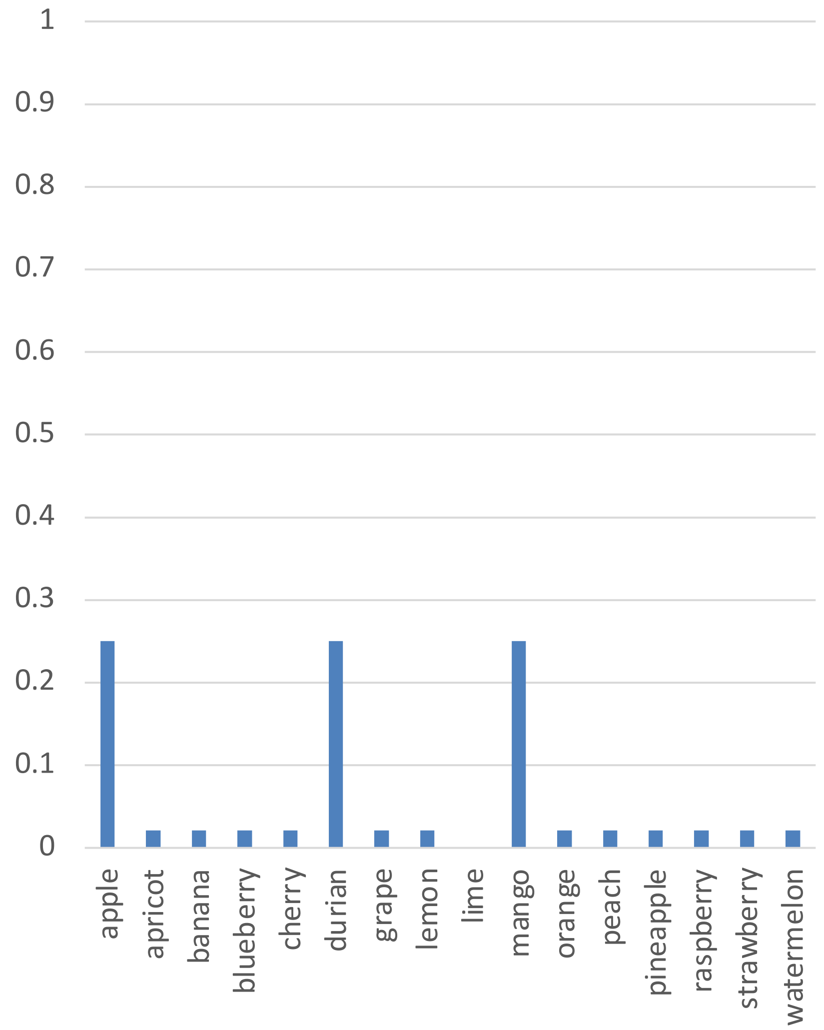
If you are trying to guess the password, you need to make about

2^{4-1} = **8 guesses**

on average

Example: calculating entropy
Scenario 2: Non-uniform passwords

Password (x_i)	Probability (p_i)
apple	1/4
apricot	1/48
banana	1/48
blueberry	1/48
cherry	1/48
durian	1/4
grape	1/48
lemon	1/48
lime	0
mango	1/4
orange	1/48
peach	1/48
pineapple	1/48
raspberry	1/48
strawberry	1/48
watermelon	1/48



Example: calculating entropy Scenario 2: Non-uniform passwords

Password (x_i)	Probability (p_i)
apple	1/4
apricot	1/48
banana	1/48
blueberry	1/48
cherry	1/48
durian	1/4
grape	1/48
lemon	1/48
lime	0
mango	1/4
orange	1/48
peach	1/48
pineapple	1/48
raspberry	1/48
strawberry	1/48
watermelon	1/48

$$
\begin{aligned}
H(X) &= -\sum_{i=1}^{16} p_i \log_2(p_i) \\
&= -3 \cdot \frac{1}{4} \log_2\left(\frac{1}{4}\right) \\
&\quad - 12 \cdot \frac{1}{48} \log_2\left(\frac{1}{48}\right) \\
&\quad - 0 \\
&\approx -\frac{3}{4} \log_2\left(2^{-2}\right) \\
&\quad - \frac{12}{48} \log_2\left(2^{-5.59}\right) \\
&= \frac{3}{4} \cdot 2 + \frac{12}{48} \cdot 5.59 \\
&= 2.8975
\end{aligned}
$$

Example: calculating entropy Scenario 2: Non-uniform passwords

Password (x_i)	Probability (p_i)
apple	1/4
apricot	1/48
banana	1/48
blueberry	1/48
cherry	1/48
durian	1/4
grape	1/48
lemon	1/48
lime	0
mango	1/4
orange	1/48
peach	1/48
pineapple	1/48
raspberry	1/48
strawberry	1/48
watermelon	1/48

If you are trying to guess the password, you need to make about

$2^{2.8975-1}$ = **3 guesses**

on average

Entropy

- If some words are more likely than others, there's less uncertainty
  => less entropy
  => easier to guess

- Entropy of passwords is a combination of length of password and randomness of each part of the password

Computer-generated randomness

- **Pseudorandom number generator**: expands a short truly random seed into a long pseudorandom string

- For security, seeds should be sufficiently unpredictable
- In a good PRNG, should be hard to predict output without knowing the seed

Computer-generated randomness

- Most programming languages have two types of PRNGS:
  - **non-cryptographically strong PRNG**
  - **cryptographically strong PRNG**

- Java: Random versus SecureRandom
- Python: random versus SystemRandom
- C: rand() versus (need to use a library)

- **Always** use a cryptographically strong PRNG for password generation

SUPPLEMENTAL MATERIAL: STORING PASSWORDS ON SERVERS

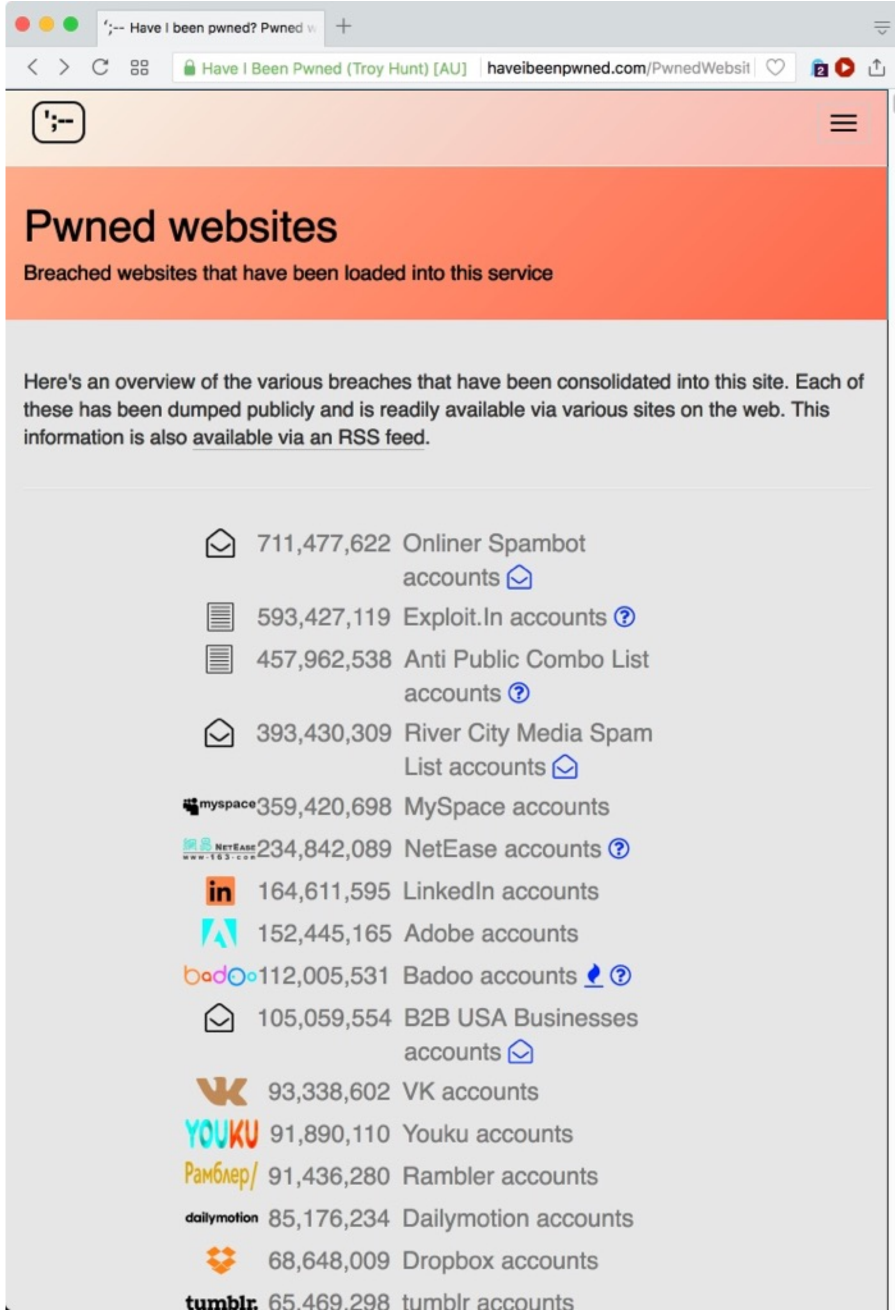
Login and registration, take 1

Registration

1. Store username and password in database

Login

1. User supplies username and purported password
2. Look up username and real password in database
3. Check if purported password = real password



https://haveibeenpwned.com/PwnedWebsites

Storing passwords securely

- Security requirements for system files storing passwords:
  - C: Can non-administrators read the password database? What useful information is in there?
  - I: Can the password file be modified? Can unauthorised modification be detected?
  - A: Need to be available when required for verification

- Note: no non-repudiation if password is known to system (or to others outside the system)

Confidentiality of passwords

- **Storage** (on authentication server)
- **Transmission** (between client and server over network)
- **Use** (display on screen when being entered?)

Login and registration, take 2

Registration

1. Store username and an **encrypted version** of the password in database

Problem: if someone learns the key, they can decrypt the database and recover all the passwords.

Login

1. User supplies username and purported password
2. Look up username and **encrypted password** in database
3. **Decrypt the stored password to recover the real password**
4. Check if purported password = real password

Login and registration, take 3

Registration

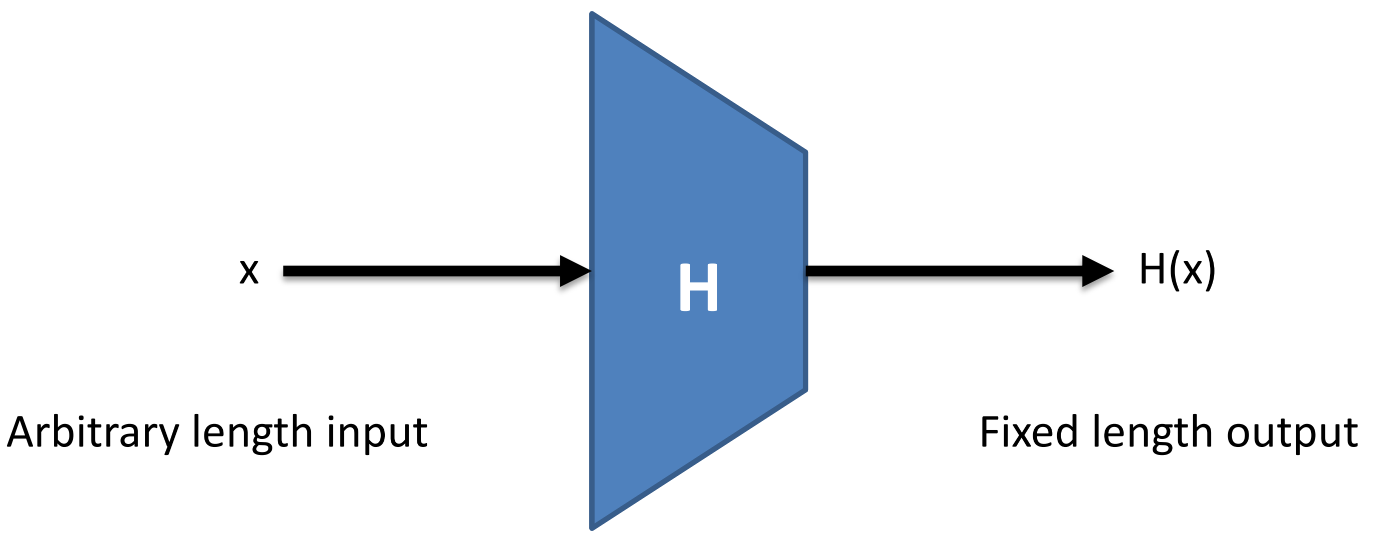
1. Store username and an **irreversible transformation ("hash")** of the password in database

Login

1. User supplies username and purported password
2. Look up username and **hash** in database
3. **Apply same irreversible transformation to the purported password**
4. Check if **hash** of purported password = **hash** of real password

Hash functions

- A hash function is a function H that maps arbitrary-length binary strings to fixed-length binary strings.



Cryptographic hash function

- A cryptographic hash function should be
  - **hard to invert**: given an output y, it should be hard to find x such that H(x)=y
    - a.k.a. "one-way", "pre-image resistant"
  - **collision-resistant**: it should be hard to find two distinct x and x' such that H(x) = H(x')
  - **pseudorandom**: H(x) should “look random”
    - Implies that if you make a small change in the input, it should make a large change in the output

Standardized cryptographic hash functions

General purpose

- MD5 (1992)
  - Collision resistance fully broken
- SHA-1 (1995)
  - Collision resistance broken
- SHA-2 family: SHA-256/SHA-512 (2001)
  - Unbroken so far
- SHA-3 family (2015)
  - Unbroken so far

Password-specific hash functions

- PBKDF2 (2000)
  - Widely used; fairly secure
- bcrypt
- scrypt
- Argon2 (2015)
  - Best available approach

Hash functions

- SHA-1: maps arbitrary length binary string inputs to 160-bit string outputs

```
SHA-1("potato") = 3e2e95f5ad970eadfa7e17eaf73da97024aa5359
SHA-1("potatO") = 5e0d1a9c2170e188c667276e1d9ed2567c754ba9
```

Using password hashes for login

Instead of storing the user’s password “potato”, store the hash of the password:

- `SHA-1("potato") = 3e2e95f5ad970eadfa7e17eaf73da97024aa5359`

At login time:

1. take the the password the user typed,
2. hash it,
3. see if it matches the hash stored in the database.

Using password hashes for login

Benefits

- Compromise of the database doesn’t reveal the user’s password
- Almost no overhead for storage and login

Drawbacks

- Can’t recover passwords for users who forget
- Attackers could create a table of password hashes to compare against database
- Can learn if two users use the same password (even if you don't know what it is)

Password hash cracking

- Suppose you learn that the hash of Alice's password is `3e2e95f5ad970eadfa7e17eaf73da97024aa5359`
  - Maybe by a database breach

- Goal: find Alice's password

Brute force attack

- Search through all possible passwords
- Possibly ordered by frequency based on known human-picked password distributions

- How big is a password space?
  - A-Z=26, a-z=26, 0-9=10
  - 8 character password
  - $62^8 = 2^{47.6}$ possible passwords
- How much can one computer do?
  - On a single computer, this would take around 1 year
  - < $200 on Amazon
  - < $50 on a botnet

Attacking using hash tables

- **Hash table**: A table containing hashes of many/all possible passwords

- Would allow an attacker with the password database to quickly find the user’s password.
- More work to crack one password hash, but can reuse work ("precomputation") to crack many password

Attacking using hash tables

- Hash tables allow for instant cracking of a password hash
- But require a massive amount of storage
  - password set: 8 character passwords, 26+26+10=62 characters
  - $62^8 = 2^{47.6}$ passwords
  - SHA-1 hash table would take 160 bits = 20 bytes per password
  - approx. $2^{52.4}$ bytes = 6 petabytes

- Can we find a time-memory trade-off where we can store less, but not increase time too much?

Attacking using rainbow tables

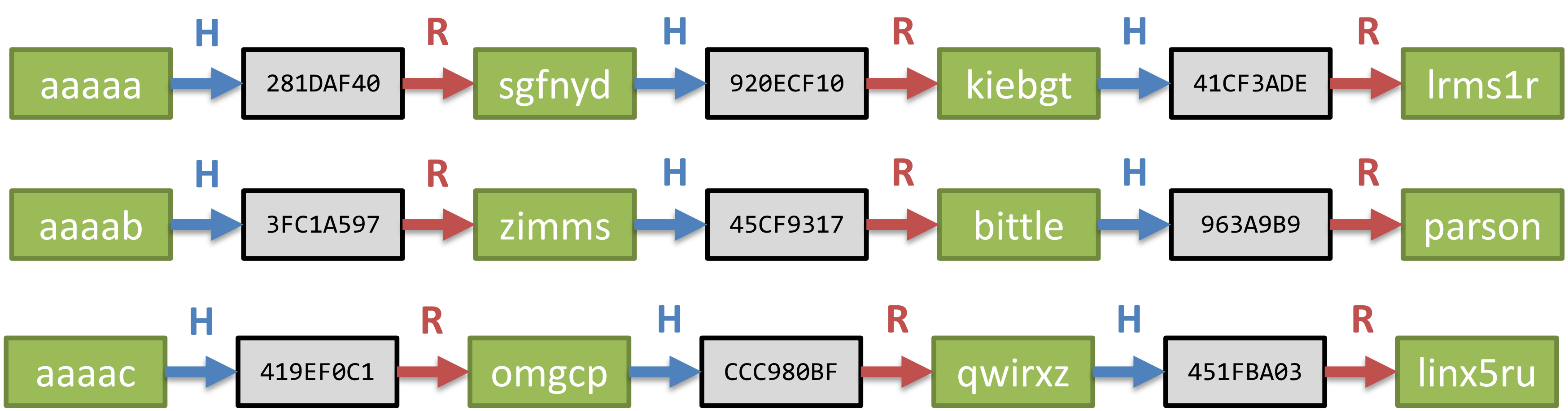
- **Rainbow tables** are an example of a time-space tradeoff using hash chains.

- Ophcrack and RainbowCrack are examples of software that can crack passwords using rainbow tables.

- RainbowCrack example:
  - 1-8 character mixed-case alphanumeric password
  - 160GB rainbow table
  - time to crack 1 password using CPU: approx. 26 minutes
  - time to crack 1 password using GPU: approx. 103 seconds
  - success rate: 99.9%

Constructing a rainbow table

1. Pick a random password
2. Construct a hash chain (hash with H, map hash back to the password space with R)
3. Store the start and end of the chain
4. Repeat many times

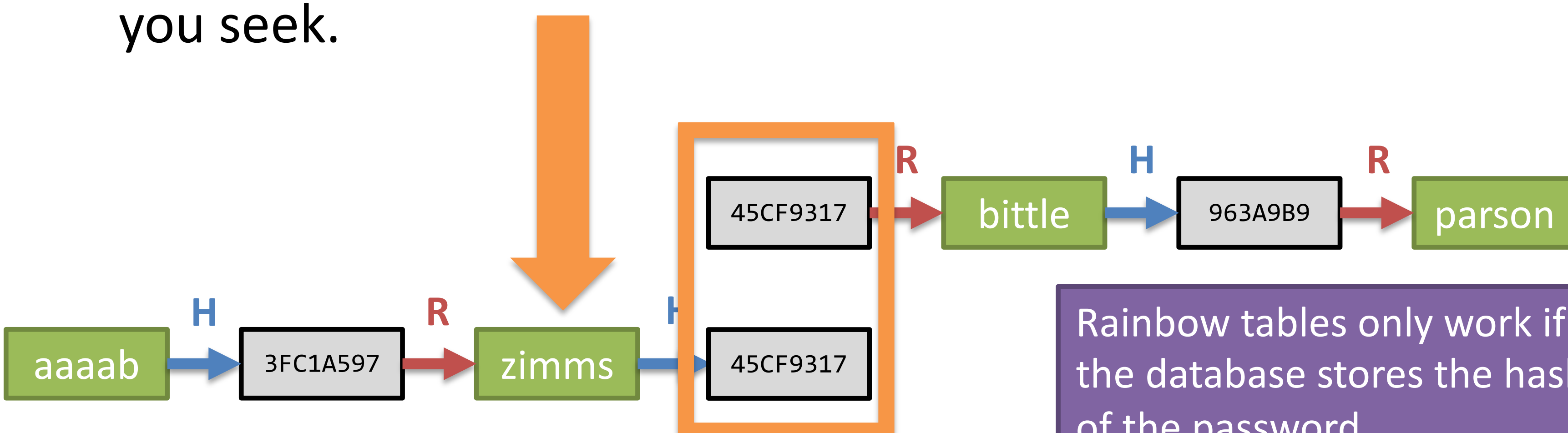


Using a rainbow table

1. Given a hash 45CF9317
2. Construct a hash chain from that hash (R, H, R, H, …), each time checking to see if the value matches any stored tail.
3. Once tail is found, take the corresponding head, and construct a hash chain (H, R, H, R, …) until you find your hash
4. The one immediately before is the password you seek.

Rainbow table

Head	Tail
aaaaa	lrms1r
aaaab	parson
aaaac	linx5ru



Rainbow tables only work if the database stores the hash of the password H(password).

Login and registration, take 4

Registration

1. Pick a random ≥80-bit salt
2. Store username, salt, and H(password, salt) in database where H is a cryptographic hash function

Login

1. User supplies username and purported password'
2. Look up username, salt, and hash in database
3. Check if H(password', salt) = stored hash

Benefits of salting

- Salting **protects against rainbow tables** since you would need a different table for each salt.

- Salting **makes brute-force attacks harder** because you can’t reuse the work from one attack on another attack.

Password hardening

- You can slow down brute-force attacks even more by **hashing the password multiple times**.
- Instead of storing
  H(salt, password)
  store
  H(H(H(…H(salt, password)))
  with 10000 hash function applications.

- My computer can apply SHA1 3190046 times per second
- So 10000 times only takes in 0.003 seconds

- Doesn't slow down login much.
- But it does slow down brute-force attacks by a factor of 10000.

Password hardening functions

- PBKDF2 (2000)
  - Widely used; fairly secure
- bcrypt
- scrypt
- Argon2 (2015)
  - Best available approach

Login and registration, take 5

Registration

1. Pick a random ≥80-bit salt
2. Store username, salt, and H(password, salt) in database **where H is a password hardening function**

Login

1. User supplies username and purported password'
2. Look up username, salt, and hash in database
3. Check if H(password', salt) = stored hash

Passwords on Unix

- `/etc/passwd` stores the list of accounts but typically not the hashed passwords; this is because `/etc/passwd` is world-readable
- `/etc/shadow` or `/etc/master.passwd` stores the hashed, salted passwords; this file is readable only by root

- Typically uses the crypt(3) algorithm with a particular hash function; e.g., default on Ubuntu 11.04 is SHA-512 with an 8-character salt

Passwords on Windows

- Up to and including Windows XP, Windows hashed passwords using the LM (LAN Manager) hash algorithm which did not use a salt.
  - Rainbow tables can be used to break LM hashes.

- **Remote authentication** up to and including Windows XP used a protocol called NTLM which required storing an additional unsalted NTLM hash.
  - Rainbow tables can be used to break NTLM hashes.

- LM disabled by default in Windows Vista and above.
- I think modern Windows still stores NTLM hashes, but it's hard to get an exact confirmation.

- Windows 8 stored encrypted (but not hashed) passwords in a file that all users had the key to decrypt

https://www.guidingtech.com/61991/cracking-windows-10-password-prevent/

https://hotforsecurity.bitdefender.com/blog/windows-8-stores-logon-passwords-in-plain-text-3914.html

Passwords on Mac OS X

- Up to Mac OS X 10.2, unsalted hashes were stored in the NetInfo database, which anyone could read.
- In Mac OS X 10.3, unsalted hashes and LM hashes were stored in a shadow file.
- In Mac OS X 10.4-10.6, salted hashes were stored in a shadow file. LM hashes are not stored by default, but are turned on when Windows File Sharing is enabled.
- In Mac OS X 10.8 and higher, salted password hashes (using PBKDF2 with SHA512) are stored in a shadow file.

http://www.dribin.org/dave/blog/archives/2006/04/28/os_x_passwords_2/

http://www.defenceindepth.net/2011/09/cracking-os-x-lion-passwords.html

Passwords in web applications

- Since there are no standard protocols for authentication in web applications, it’s up to the application itself to decide how to store passwords.

- SQL databases (e.g., MySQL) typically have MD5(…) and SHA1(…) functions built in, but developers still need to do salting/hardening in the application code.

How can a remote user prove that they know their password?

- Send the password over an unencrypted channel
  - Bad.
- Send the password over an encrypted channel.
  - Okay, but only if the user knows that the encrypted channel is with the right server.
- Send a hash of the password over an encrypted channel.
  - Good, but still vulnerable to rainbow tables.
- Send a salted hash of the password over an encrypted channel.
  - Better, but still vulnerable to brute force attacks (called offline dictionary attacks).
- Use a **password authenticated key exchange protocol**.
  - Very good, secure against dictionary attacks.
  - Not widely implemented (and many have patent restrictions).

Default and hard-coded passwords

- Many password-protected vendor-supplied software and hardware has default passwords.

- It is often that users are not prompted to change the passwords on setup.
- Or even that it is not possible to change the default passwords (they are hard-coded).

- "Well over 50 percent of the control system suppliers" hard-code passwords into their software or firmware.
  - Joe Weiss, Protecting Industrial Control Systems from Electronic Threats

- Databases of default passwords:
  - http://www.cirt.net/passwords

- Hard-coded Siemens WinCC SCADA passwords:
  - http://www.wired.com/threatlevel/2010/07/siemens-scada/

- Samsung printers:
  - http://www.kb.cert.org/vuls/id/281284