Web Security

Encoding, validating & sanitizing

Sessions & Authentication

G&#252;ne&scedil; Acar & Erik Poll

Digital Security group
Radboud University Nijmegen

Web Security

Encoding, validating & sanitizing

Sessions & Authentication

Güneş Acar & Erik Poll

Digital Security group
Radboud University Nijmegen

Last week & today

Last week

- Web clients & servers interact using HTTP.
- The HTTP traffic contains URLs (for 'addressing') and HTML (for the 'content') which can contain JavaScript as code to be executed client-side
- HTTP requests are usually GET or POST requests
  - GET: parameters in URL
  - POST: parameters in HTTP body

Today

1) The languages & encodings of data in HTTP traffic
2) Two notions of sessions for security:
   - TLS / HTTPS at network level
   - cookies at application level

Exercises for this week

A. Check input sanitisation in Brightspace

How is input encoded & sanitised in Discussion Forums, at the client side and/or at the server side?

B. Check security settings for some sites where you have a login, incl. whether it support HTTP(S), HSTS and Certificate Transparency(CT) and which cookies flags it uses.

C. One more WebGoat lesson

Authentication Flaws - Authentication Bypasses

A & B to be handed in (in pairs) via Brightspace

Deadline Monday Sept 15, 23:59

Languages & encodings

(continued from last week)

Web pages contain HTML, CSS, JavaScript and URLs

```
 <html><title>The various languages and formats used inside web pages</title>
   <body>
     <h1 style="color:blue;">Sample exam question<h1>   is 3 &lt; 4?
     <a href="https://duckduckgo.com/?q=how+to+encode+<+in+HTML%3F">A link with special characters</a>
     <a href="https://duckduckgo.com/?q=how+to+encode+%2F+in+a+URL%3F">And another one</a>
     <script> var x = 'a string with a single quote \' and double quote ".';
             alert(x);
     </script>
  </body>
</html>
```

- Special characters may need to be *encoded* aka *escaped* to prevent unintended effects or preserve intended effect
- Which characters have to encoded, and how, depends on the context.
  - Eg < is a special character in HTML, but not in a URL
- Within a single language there can be several contexts. Eg
  - / is a special character in URLs, but not in the query string (i.e. after the ?)
  - For a JavaScript strings inside JavaScript the outer quotes (' or ") determine which quotes inside the string need to be escaped.

URL encoding aka %-encoding

Replaces reserved characters that have a special meaning in URLs

`/?!*';:@&=+$,#()[]`

with their ASCII value in hex preceded with escape character `%`

/	#	space	=	?	%	...
%27	%23	%20 or +	%3D	%3F	%25	...

Try this out with eg `https://duckduckgo.com/?q=%3F`

Encoding space as + comes from older x-www-form-urlencoded format

Possible sources of confusion (and bugs or security issues?)

- The reserved characters are different for different parts of the URL. Eg / in the path of a URL must be encoded, in the query it need not be
- What happens if you URL-encode unreserved characters? eg `A -> %41`
- What happens if you double URL-encode? eg `% -> %25 -> %2525`

HTML encoding

Replaces HTML special characters with similar looking ones

<	>	&	“
`&lt;`	`&gt;`	`&amp;`	`&quot;`

- HTML encoding and URL encoding are needed in different contexts
  - *Things can get confusing: what about URLs inside HTML or vv?*
- HTML also has the notion of character encoding: which character set is used, eg ASCI or UTF-8 (default)
- Browser engines can be sloppy or forgiving, and let you get away with *not* encoding e.g. `&` as `&amp;` in webpages
  - http://validator.w3.org checks if a page is correct HTML
- On top of HTML-encoding, websites may apply additional input sanitisation to remove or replace tags it wants to disallow in user input;
  - eg `<script>` tags are commonly stripped from user input

base64 encoding

HTTP is text-based, so all data transmitted has to be text

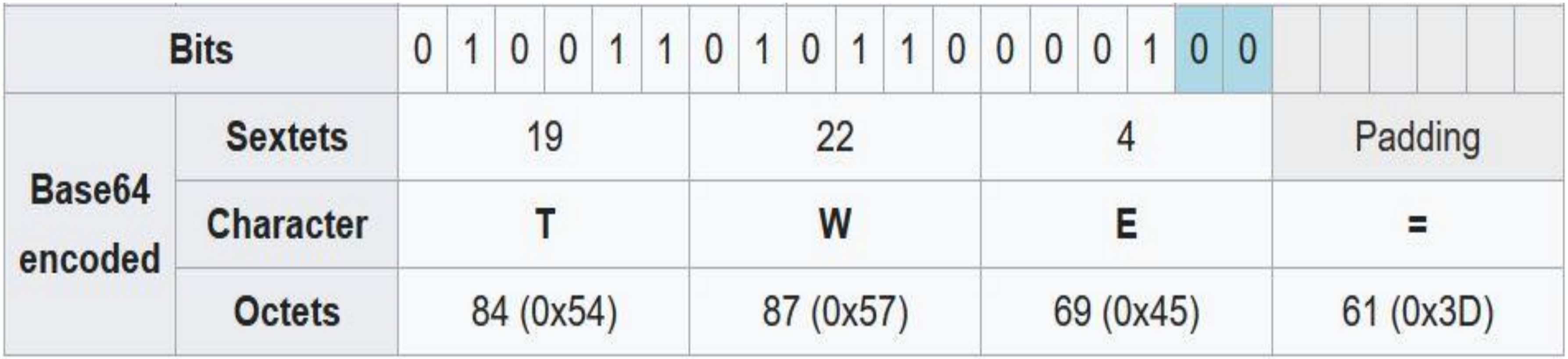
– ie. printable, displayable characters

Base64 encoding turns ‘raw’ binary data - bytes - into text so that it can be transferred via HTTP

- 6 bits coded up as one of the 64 standard characters

  `a-z A-Z 0-9 + /`
- Groups of 3 bytes (ie 24 bits) represented as 4 characters
- Padding with = or == to make sure results is multiple of 4 characters long

base64 encoding



Bits		0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	0						
Base64 encoded	Sextets	19						22						4						Padding					
	Character	T						W						E						=					
	Octets	84 (0x54)						87 (0x57)						69 (0x45)						61 (0x3D)					

- groups of 6 bits coded up as one of the standard characters

  `a-z A-Z 0-9 + /`
- So 3 bytes represented as 4 characters
- Padding with zeroes to make the input a multiple of 6 bits
- Padding with = or == to make sure results is multiple of 4 characters long

Details not that important for this course, but you may come across base64-encoded data

Encoding user content for security

User-supplied content in webpages may need to be encoded or sanitised to prevent malicious content from triggering unwanted effects.

```
 <html><title>Mallory’s Radboud Student Homepage</title>
   <body>
     <h1 color ="color:red;">Welcome to Mallory’s homepage</h1>
      Some text that Mallory provided.
      <a href="https://ru.nl/Mallory">My contact information</a>
      <img=https://bla.com/common/view/my_profile_image.jpg</img>
     <a href="https://brightspace.ru.nl/d2l/home/427025">My favourite course</a>
     <a href="https://ru.osiris-student.nl/grades.html?uid=s123456">My grades</a>
     <script> someJavaScriptFunction(someOtherString+'Mallory’);  </script>
  </body>
</html>
```

Beware of confusion

encoding

- changing the representation of data
- no information is lost or changed
- eg HTML or URL encoding

sanitisation and validation

- removing or changing ‘problematic’ data
- some information is lost; possibly an entire request is rejected as invalid
- eg removing <script> tags or rejecting incorrect date 31/2/2024

- Encoding can also be called *escaping* or *quoting*, and validation is sometimes called *filtering*
- Common distinction: validation *rejects entire inputs,* whereas sanitisation *changes* them or *removes problematic parts of inputs*
- Beware: some people use alle these terms interchangeably

Exercise to hand in this week

- Figure out how Brightspace encodes and/or sanitises user input in Discussion Forums
  - client-side in the browser and/or server-side
  - for header and body of forum posts

NB try to describe this as concise as possible, eg in terms of URL or HTML encoding.

Web Security

Authentication & Session Management

Security shortcomings of internet

“No security built into the internet”
But what does that mean?

- *No way of knowing who you are communicating with, apart from an IP address*
  - *ie no authentication*
- *Any party along the way (wifi router, ISP, ...) can read or modify the communication*
  - *ie no integrity & confidentiality of communication*

Adding security: two security

Two security requirements we want to add

1. Authentication
    a) of the web site by user
    b) of the user by the web site

   How?

   For a) TLS certificates aka X509 certificates

   For b) username/password or more secure solutions, eg MFA (Multi-factor Authentication)

2. Integrity & confidentiality of communication

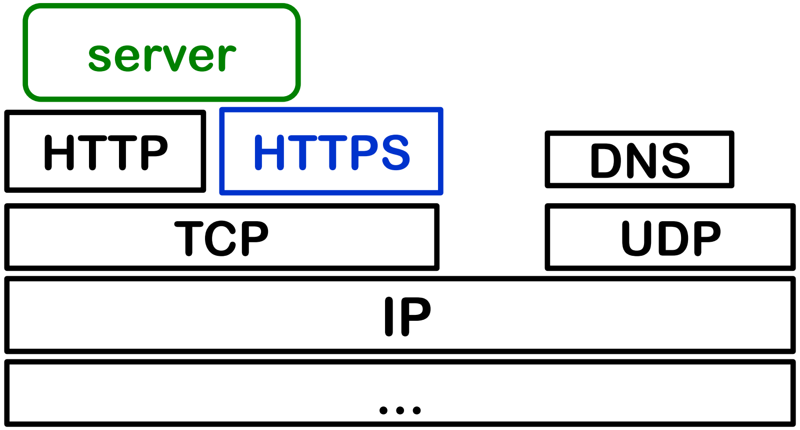
   How?

   TLS, which adds *encryption* and *integrity protection* with
    – MACs (Message Authentication Codes)
    – digital signatures

Today: two notions of sessions

1. HTTPS at the network layer
   - by TLS, on top of TCP or inside QUIC
   - includes authentication of the server

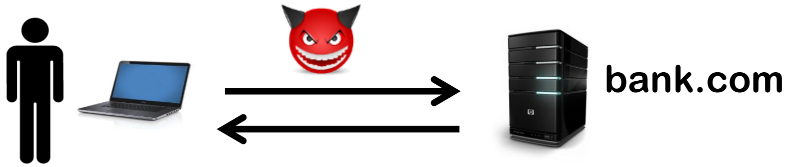
2. Session management at the application layer
   - by web application using sessions IDs and/or cookies
   - includes authentication of the user



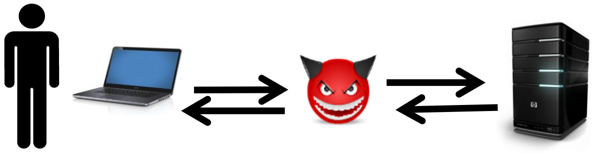
HTTPS

Attacker models for the internet & the web

a) Passive eavesdropper



b) Active Man-in-the-Middle (MitM) attacker



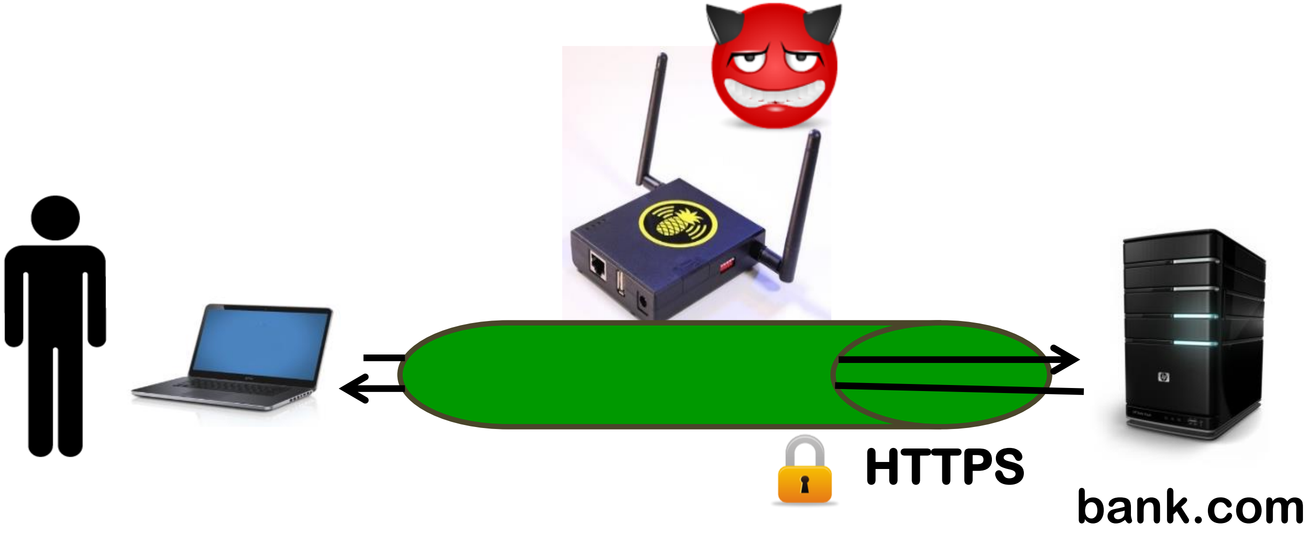
Eg a malicious or compromised ISP, router, or WiFi access point

WiFi security (eg WPA2) should prevent attackers eavesdropping on Wifi traffic

c) Malicious or vulnerable end points (browser or server)

A malicious server (eg fakebank.com) can act as MitM by relaying traffic to real website bank.com

(partial) security solution: TLS



TLS

1. Server sends X509 server certificate to client
   - Signed by a Certificate Authority (CA) or self-signed
   - Browsers come pre-configured with list of trusted CAs

2. Client checks that certificate has not been revoked
   - by requesting Certificate Revocation List (CRL) from CA

3. Client authenticates the server, with a challenge-response protocol

4. Client and server then agree a session key

5. Subsequent HTTP traffic in a secure tunnel

Does a self-signed certificate provide any security guarantee?

Yes, because at least clients knows they keep talking to the same server

TLS – crypto details; not important for this course

1. Server sends X509 server certificate to client

   includes server's public key `PK`

2. Client checks that certificate has not been revoked

3. Client authenticates the server, with a challenge-response protocol

   Client sends nonce `n` encrypted with public key `PK`, and checks if server response includes `n` which proves knowledge of corresponding private key

4. Client and server then agree a session key

   Typically an AES key

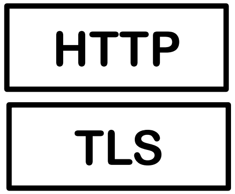
5. Subsequent HTTP traffic in a secure tunnel

   Traffic encrypted and MACed with session key

   - *encryption for confidentiality, MACing for integrity*

   Periodically the session key is refreshed

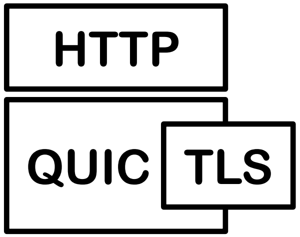
HTTPS: HTTP over TLS



Security guarantees :

- Confidentiality & integrity of the session
  - Attacker on the network can still see *that* two IP addresses communicate (an example of meta-data), but not *what*
    - All HTTP content, incl. headers & URL parameters are protected inside TLS tunnel
  - Attacker cannot change any traffic or replay it
- Server authentication, using certificates
- Possibly, but uncommon: Client authentication with a client certificate
  - Usually servers use another means to authenticate clients: often passwords ☹

The same holds if TLS is used as part of QUIC



Aside: name confusion TLS vs SSL

TLS (Transport Layer Security) used to be called **SSL** (Secure Sockets Layer)

- TLS version 1.0 is SSL version 3.1
- Latest TLS version is 1.3

This explains why X509 certificates are sometimes called SSL certificates and a well-known TLS implementation is called OpenSSL

We'll come back to TLS later in this lecture to discuss its limitations.

Mixing http & https

A web page can mix http & https content, but this is a bad idea!

- ***Why would you never want to have an frame loaded via http inside a webpage loaded via https?***

Web browsers nowadays warn about or block mixed http/https content.

Demo: check out how this works in your browser, by visiting

http://www.cs.ru.nl/~erikpoll/websec/demo/mixed_content.html

https://www.cs.ru.nl/~erikpoll/websec/demo/mixed_content.html

This demo ~~no longer~~ still works in Firefox, but it does in Chrome

Sessions (at application level)

Functional shortcoming of HTTP

HTTP is stateless and has no notion of session, ie

- **No state is recorded about history of previous requests**
- **(Hence) no notion of a sequence of requests belonging together in one conversation between client and server**

This is very clumsy for interaction between a client and server

- ***Has this user logged in?***
- ***Did the user select English or Dutch as language for the site?***
- ***Has the user put items in their online shopping basket?***
- ***Did the user already agree to our privacy policy?***

Why can't we use IP address for this?

- Different clients may share the same IP address

  Eg different browsers & apps on the same device,
  different users on lilo.science.ru.nl,
  or different users on a local wifi network (esp for IPv4)

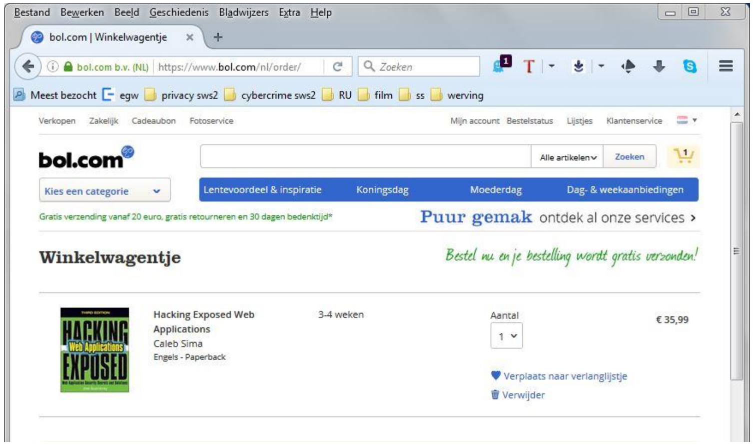
- Multiple web applications can share the same IP address
  - especially web applications hosted in the cloud

- Clients and servers can change IP address
  - eg. clients on mobile devices, when switching from mobile network to WiFi or v.v.
  - also: web applications hosted in the cloud, if they are migrated to other server

Session & session data

There is usually session data associated with a session that needs to be remembered.

- Eg: content of online shopping basket



https://www.bol.com/nl/order/

Gratis verzending vanaf 20 euro, gratis retourneren en 30 dagen bedenktijd*

Puur gemak ontdek al onze services ›

Winkelwagentje

Bestel nu en je bestelling wordt gratis verzonden!

Hacking Exposed Web Applications
Caleb Sima
Engels - Paperback

3-4 weken

Aantal: 1

€ 35,99

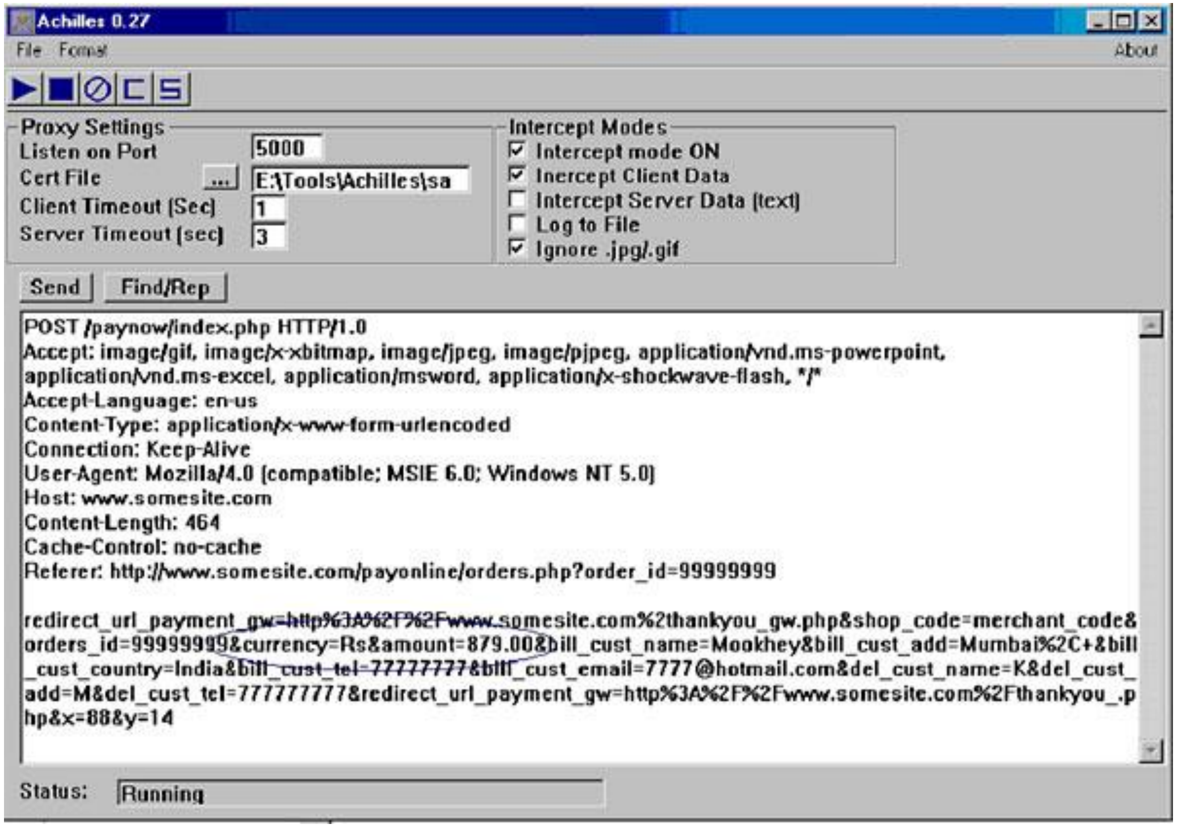
Ways to keep track of such data:

1. send it back & forth between server and browser
   with each request and response
   eg using hidden parameters
2. record it at the client side
   using HTML local storage
3. record it at the server side and just send back & forth a unique identifier

Pros & Cons ?

- Con 3: server has to record lots of info for many sessions
- Con of 1 & 2: client could mess with this data

Things that can go wrong with session data



Proxy Settings
Listen on Port: 5000
Cert File: E:\Tools\Achilles\sa
Client Timeout (Sec): 1
Server Timeout (sec): 3

Intercept Modes
☑ Intercept mode ON
☑ Inercept Client Data
□ Intercept Server Data (text)
□ Log to File
☑ Ignore .jpg/.gif

```
POST /paynow/index.php HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */*
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Host: www.somesite.com
Content-Length: 464
Cache-Control: no-cache
Referer: http://www.somesite.com/payonline/orders.php?order_id=99999999

redirect_url_payment_gw=http%3A%2F%2Fwww.somesite.com%2thankyou_gw.php&shop_code=merchant_code&
orders_id=99999999&currency=Rs&amount=879.00&bill_cust_name=Mookhey&bill_cust_add=Mumbai%2C+&bill
_cust_country=India&bill_cust_tel=77777777&bill_cust_email=7777@hotmail.com&del_cust_name=K&del_cust_
add=M&del_cust_tel=777777777&redirect_url_payment_gw=http%3A%2F%2Fwww.somesite.com%2Fthankyou_.p
hp&x=88&y=14
```

Status: Running

Classic security flaw: the price is recorded in a hidden form field, as shown in the proxy output above.

The client can change this...

Misplaced trust in the client

For data for which integrity is important (eg prices)

the server should never trust the client

to provide this data or to return this data unaltered

Instead, the server should

- store such data server-side

or

- add a cryptographic integrity check
  - eg using a MAC (Message Authentication Code) or Digital Signature
  - Such a check should also include a time stamp or some session id that frequently changes, to avoid replay or roll-back attacks.

Session data for authentication

Authentication often involves a notion of session, and then goes in two steps

1. Actual authentication, say with a username/password plus the response of an MFA token
2. Creating a session, with session identifier aka session token for fast & easy (re)authentication without repeating step 1

Most web applications use session cookies for this purpose

- Such cookies provide identity and proof that this identity has verified (aka authenticated)
- These cookies are just as valuable for attacker as original credentials used to authenticate, eg username/password ***plus*** MFA response

Sessions managed by the web application

Typical steps

1. Web application creates & manages sessions
   - Session data is stored at server and associated with a unique session ID
2. Client is informed of session ID
   - and client attaches session ID to subsequent requests

so server knows about previous requests

Web application frameworks usually provide built-in support for session management, but web application developers can implement their own

- NB it is better to use existing solutions than inventing your own
- Still, don't underestimate the complexity of using these correctly

Solution 1: session ID in URL

Web page returned by the server contains links with session ID as extra parameter

```
<html>
Example web page with session IDs in the URL.
The user can now click
<a href=”http://demo.net/nextpage.php?sid=1234”>here</a>
or
<a href=”http://demo.net/anotherpage.php?sid=1234”>here</a>
passing on its session id back to the server
wherever he goes next.
</html>
```

Hence: every user gets their own unique copy of a web page.

Solution 2: session ID in hidden parameter

```
<htm>
The form below uses a hidden field
<form method="POST" action= "http://ru.nl/register.php">
   Email: <input type="text" name=“Your email address">
   <input type="hidden" name="sid" value=“s1234">
   <input type="submit" value="Click here to submit">
</form>
```

Hidden means hidden from the user by browser,
   not hidden from a proxy like ZAP.

A hidden form field could also be used to track user preferences, eg

```
     <input type="hidden" name="Language" value="Dutch">
```

Session ID in URL vs hidden parameter

Can you think of a downside of a session ID in the URL?

> ***If you give a link with your session ID to someone else, then that person might continue with your session!***
>
> ***Also, bookmarking a URL incl. the session ID does not (or should not) make sense, as the next time you use the bookmark you should start a different session***

Solution 3: sessionID in a cookie

Standard solution built into HTTP and browser

- Cookie is piece of information that is set by the server and stored by the browser
  - namely when HTTP response includes `Set-Cookie` field in header
  - It belongs to some domain, eg `www.test.com`
  - It includes expiry date, domain name, optional path, optional flags
    - eg `secure`, `HTTPOnly`, and `SameSite` flags
- Cookie is *automatically* included in any HTTP request by the browser, for any request to that domain
  - in the `Cookie` field of HTTP request
- Cookie can include any type of information
  - sensitive information, such as session ID
  - less sensitive information, such as language preferences

Example cookie traffic

- Setting a cookie set with an HTTP response

```
HTTP/1.0 200 OK
Content-type  text/html
Set-Cookie: language=Dutch
Set-Cookie: sessionID=123; Expires=Tue, 26 Apr 2021 11:30:00 GMT
...
```

- Sending a cookie in an HTTP request

```
GET someurl.html HTTP/1.0 200 OK
Host: example.com
Cookie: language=Dutch, sessionID=123
```

Different types of cookies

- non-persistent cookies
  - only stored while current browser session lasts

- persistent cookies
  - preserved between browser sessions
  - useful for maintaining login and user preferences across sessions
  - bad for privacy

Domains, subdomain, and top level domains

The domain in a cookie can be a subdomain of a website (eg `cs.ru.nl` is a subdomain of `ru.nl`) which raises questions, such as

Are cookies for `cs.ru.nl` sent with requests to `ru.nl`? Or v.v.?

Can `ru.nl` set a cookie for `cs.ru.nl` ?

Complex rules restrict cookie access across (sub)domains [RFC 6265]

Overall rationale: subdomains need not trust their superdomain

- Subdomains can *access* cookie for domain, but not vice versa
- Subdomains can *set* cookie for direct superdomain, but not vv
- With the `HostOnly` flag, cookies can further restrict access

For details, check [RFC6265] and hope browsers do not still implement outdated parts of [RFC 2109] or [RFC 2965].

For top level domains, eg `.nl`, there are additional rules,

to prevent say `ru.nl` from setting a cookie for `.nl`

But does this work as intended for countries using 3 level domain names?
Eg for `somecompany.co.uk`, where `co.uk` is not a top level domain

Different ways to provide session ID

1. Encoding it in the URL

   Downsides: 1) stored in logs (eg browser history), 2) can be cached & bookmarked, 3) visible in the browser location bar.

2. Hidden form field

   Better: won’t appear in URLs, so cannot be bookmarked, and less likely to be logged

3. Cookies

   Best choice: automatically handled by browser; easier & more flexible.

   But such automation has downsides, as we’ll see: CSRF

Now: attacking this!

Session attacks

Aim of attacker: get the session ID

- This can be session cookie, or other form of session ID
- If the victim is logged in, this is just as good as stealing his username and password!

How would you do this?

Eavesdropping & MiTM attack

If traffic is not protected with TLS
(or Wifi protection on lower network layers)
then someone sniffing the network traffic can obtain session IDs.

Attacker can also set up a (fake) network access point to do this

- and then even do active Man-in-the-Middle attack.

There are some variants to by-pass TLS protection, as we will see later.

Session ID prediction attack

Suppose you can check your grades in blackboard on page

`brightspace.ru.nl/grades.php?s=s776823`

Is this a security problem?

If s776823 is your student number and also the session id (in the URL in this case) then it is!

Attacker could try other student IDs or – better still – the university employee number of a teacher.

Session ID prediction attack

Suppose you can check your grades in blackboard on page

`brightspace.ru.nl/grades.php?s=s776823`

Is this a security problem?

If s776823 is your student number and also the session id (in the URL in this case) then it is!

Attacker could try other student IDs or – better still – the university employee number of a teacher.

Session fixation attack *(aka Login CSRF)*

If the sessionID is in the URL, an attacker can

1. start a session with bank.com and obtain a session ID;
2. craft a link with that session ID and gets victims to click it, by
   a) emailing victims with that link in the email; *or*
   b) luring victims to a webpage with that link
3. The victim now goes to the website using a known session ID;
4. If victim logs in, and *session ID is not changed*,
   then attacker can join the session & abuse the user's rights!

Therefore: web server should change session ID on login actions

If the session-ID is a hidden form field, it does not end up in URL, attacker cannot email a link, but option 2b) is still possible, with a POST request

Variant: attacker has already logged in, so victim joins the attacker's session and may enter confidential data (eg credit card number) for attacker's account

Making these attacks on sessions harder

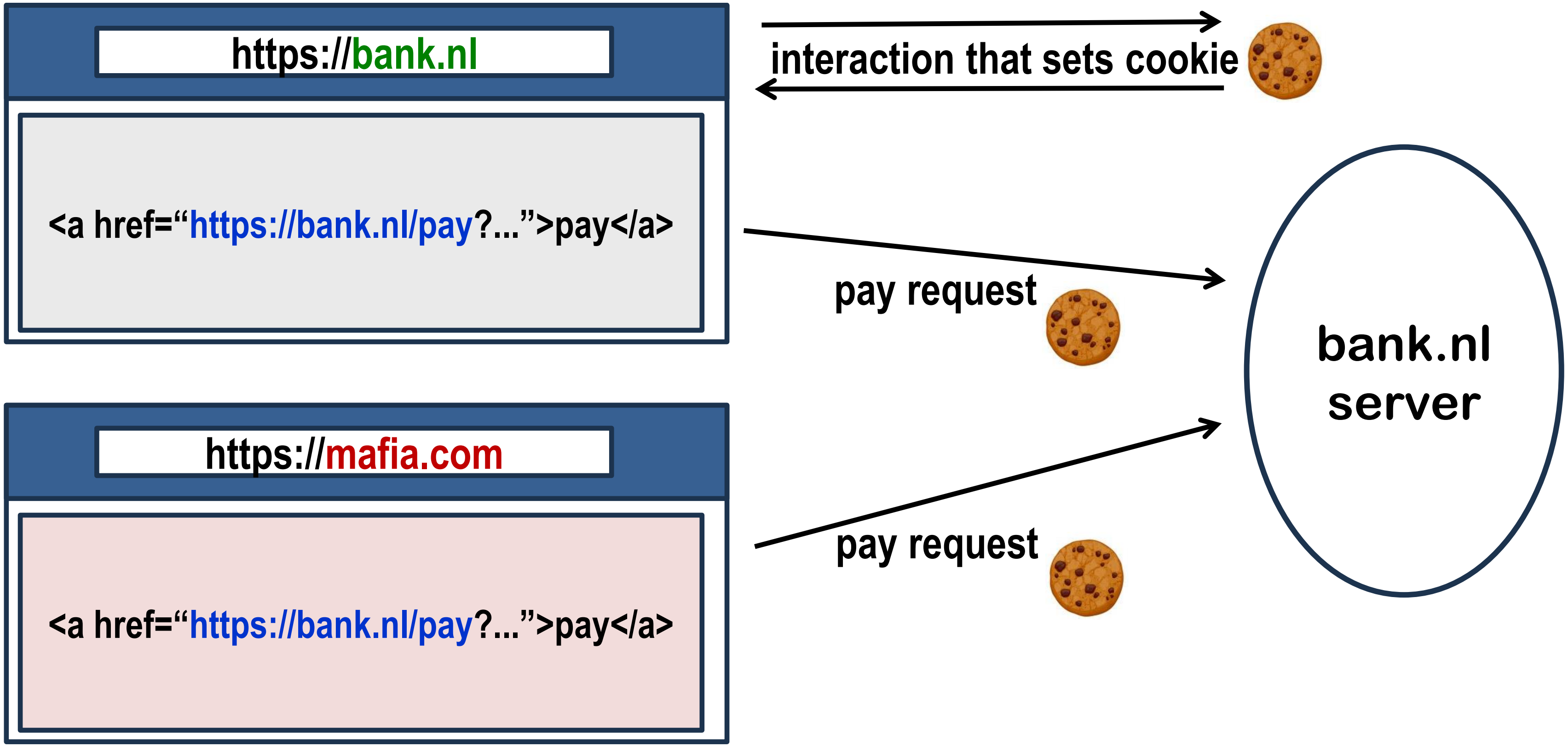
- Use long enough, random session IDs – ie with enough entropy
  - prevents session prediction and brute forcing
- Change session ID after any change in privilege level

  eg after logging in
  - prevents session fixations
- Expire sessions

  eg by setting expiration time on cookies
  - reduces the attack surface in time
- Use HTTPS

  for *all* requests & responses that include session ID, not just the login
  - prevents networking sniffing of session ID
- Let clients re-authenticate before important actions
  - reduces the value of any stolen session ID

CSRF: the downside of browser automatically adding cookies



<a href="https://bank.nl/pay?...">pay</a>

<a href="https://bank.nl/pay?...">pay</a>

Browser attaches cookie to cross-domain requests from any site

Abusing cookies without stealing them (CSRF)

Suppose for a bank transfer a website `bank.com` contains the URL

```
   <a href="transferMoney?amount=1000
                                &toAccount=52.12.57.762">
```

Suppose attacker sets up a malicious website `mafia.com` with

```
  <a href="https://bank.com/transferMoney?amount=1000
                                  &toAccount=52.12.57.762">
```

If attacker tricks users to click on second link *while they are logged on at bank.com,* the browser automatically attaches the bank's cookies to both requests! And money will be transferred...

This is called a Cross-Site Request Forgery (CSRF)

Root cause of the problem: browser automatically attaches cookies to all requests, regardless of which page the link is on.

CSRF

CSRF is only possible if we use cookies for sessions, not if we have session ID in URLs or in hidden forms fields

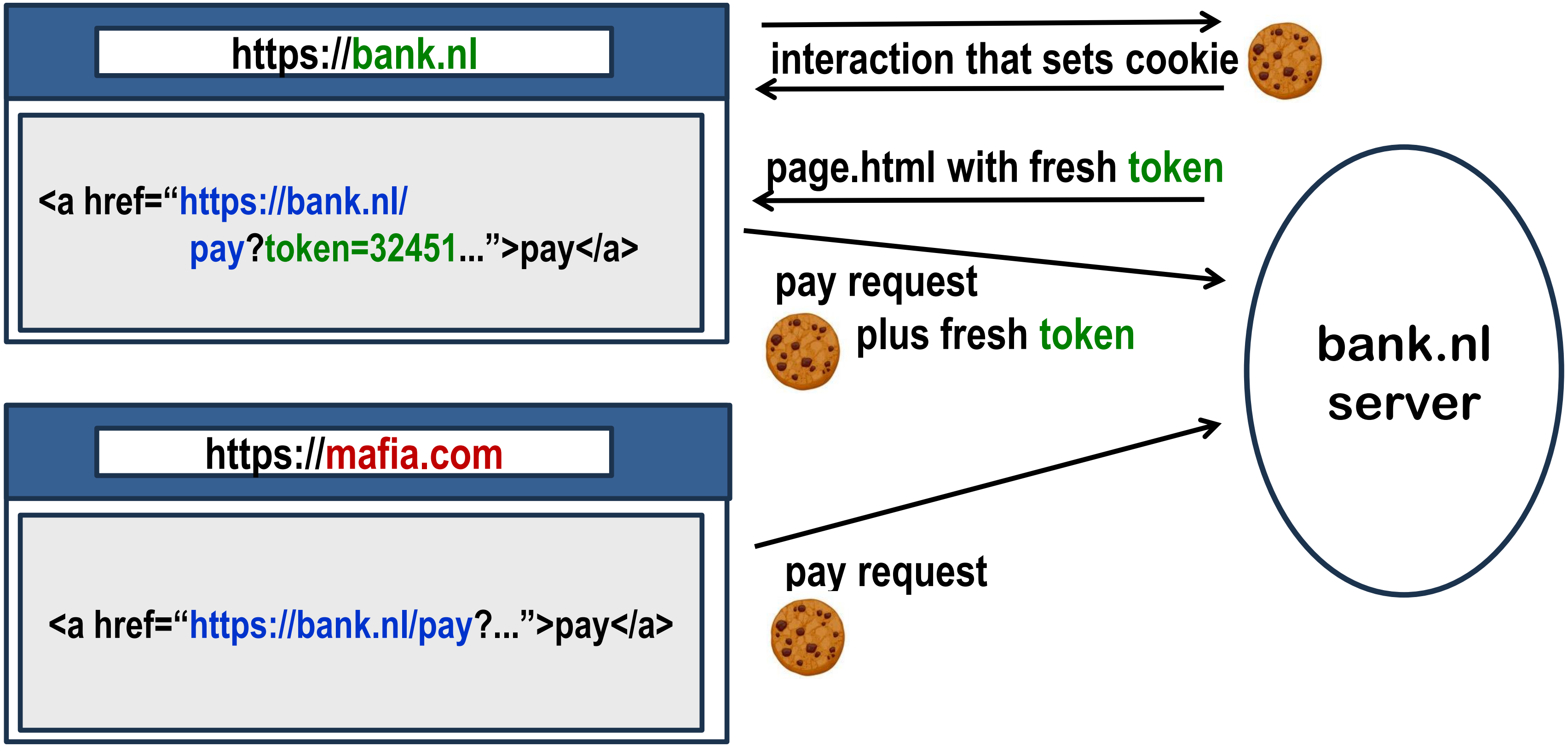
CSRF is an example of feature interaction, namely of the features that:

1) any web page can link to another other web page
2) browser automatically attaches the cookies of A.com to any requests to A.com

The combinations of these features can be abused

by attackers creating a webpage or HTML email with links to bank.com, where the browser will automatically attach the bank's cookies with the correct value to authenticate these requests (assuming victim is logged on)

Countermeasure: (anti)CSRF token



https://bank.nl

```
<a href="https://bank.nl/
          pay?token=32451...">pay</a>
```

interaction that sets cookie

page.html with fresh token

pay request

plus fresh token

bank.nl server

https://mafia.com

```
<a href="https://bank.nl/pay?...">pay</a>
```

pay request

Standard solution to prevent CSRF

Use two special numbers to identify a session

1. a fixed session ID stored in a cookie
2. a changing anti-CSRF token, as URL parameter or hidden form field, that changes to a new random value for each request

For any malicious cross-site requests, say from mafia.com to bank.com, the browser will attach the right session ID cookie, but these requests will not have the right CSRF token.

Confusingly, anti-CSRF tokens sometimes called CSRF tokens or anti-XSRF token

Other countermeasures against CSRF

1. Check `Refer(r)er` or `Origin` headers

Browser includes these in HTTP requests to indicate where a request is made from (eg mafia.com or bank.com), so bank.com can check which webpage made the request

`Origin` is just the domain, `Referer` the domain plus the path

But these headers may be absent ☹ because

- Browsers can be configured not to include these headers, for privacy reasons
- Websites can specify a Referrer Policy to tell browser not to include them under certain conditions (eg when making HTTP request from HTTPS context, only for requestions within the same site, ...)

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy