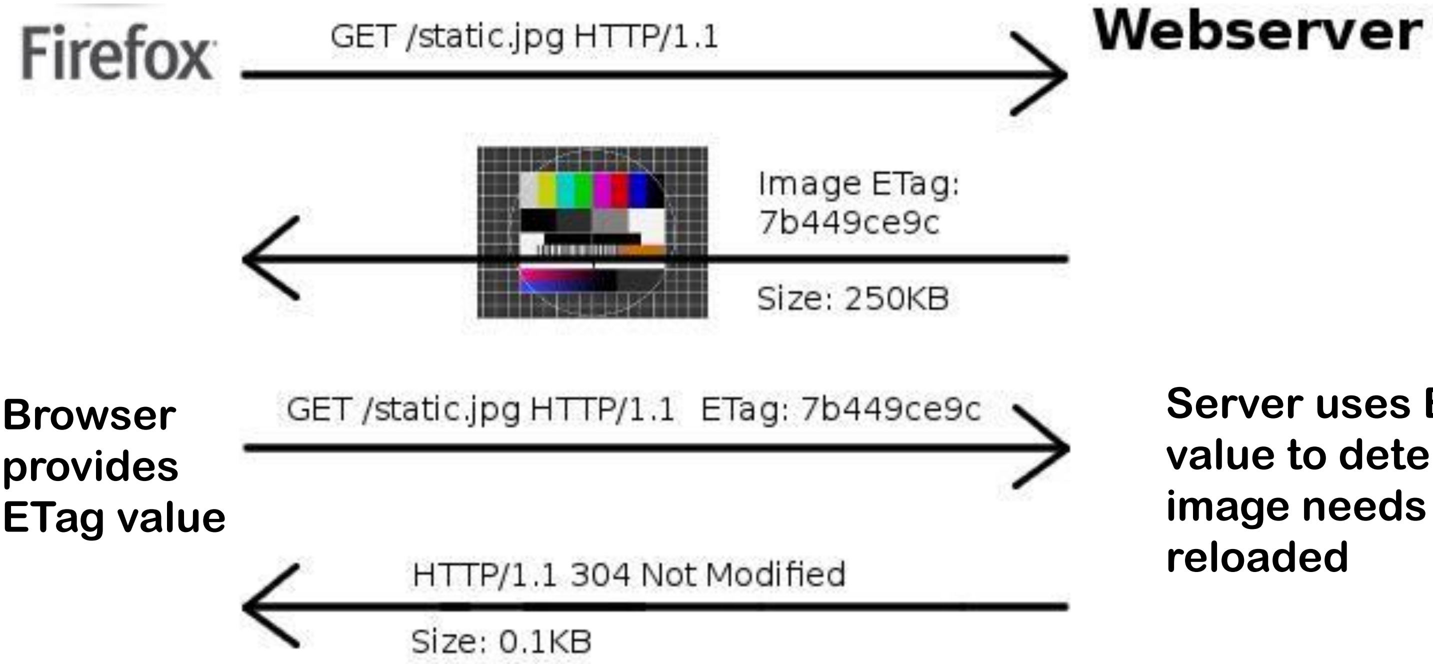
Privacy risk: ETags for cached images

ETags (entity tags) are identifiers added to control caching



Browser provides ETag value

Server uses ETag value to determine if image needs to be reloaded

Browser tells server which version of image it has cached;
This allows server to identify a user by adding unique Etag

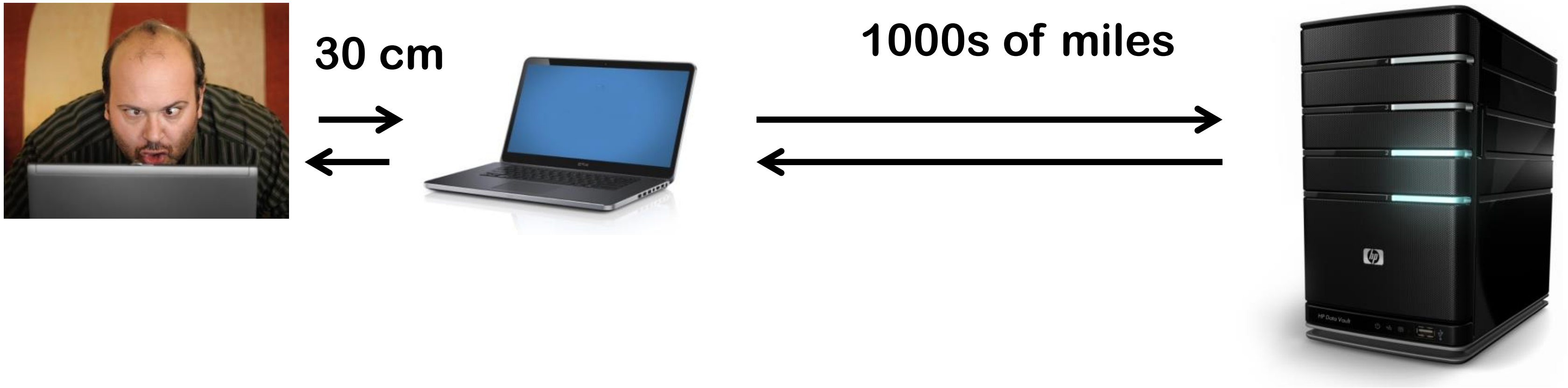
Demo-ed at https://cable.ayra.ch/toys/track.php

Today:

More attacks on clients, esp. the <u>user</u>

URL obfuscation,
Click-jacking/UI redressing,
CSRF revisited

Securing the last 30 centimeter...



30 cm

1000s of miles

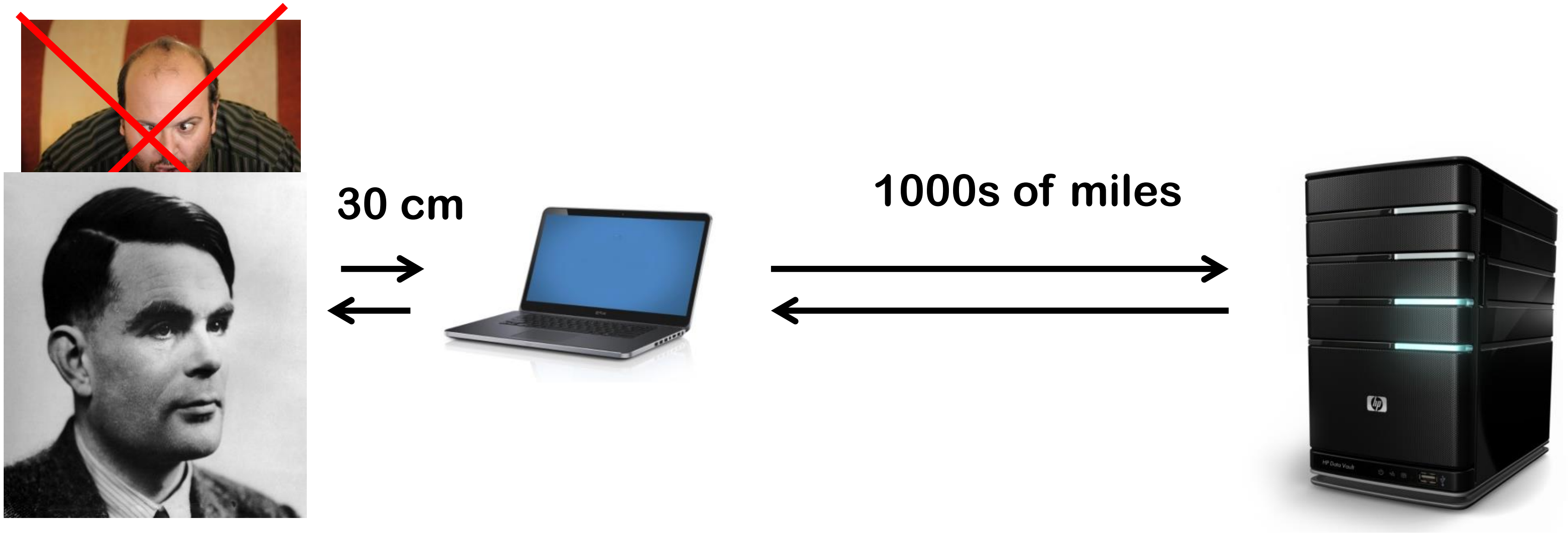
We can secure connections between computers 1000s of miles apart, eg using TLS,

but the remaining 30 cm between user and laptop remain a problem

Beware: blaming the ‘dumb user’ is usually unfair victim blaming.

We should blame computer scientists & engineers for making poor solutions

Securing the last 30 centimeter...



30 cm

1000s of miles

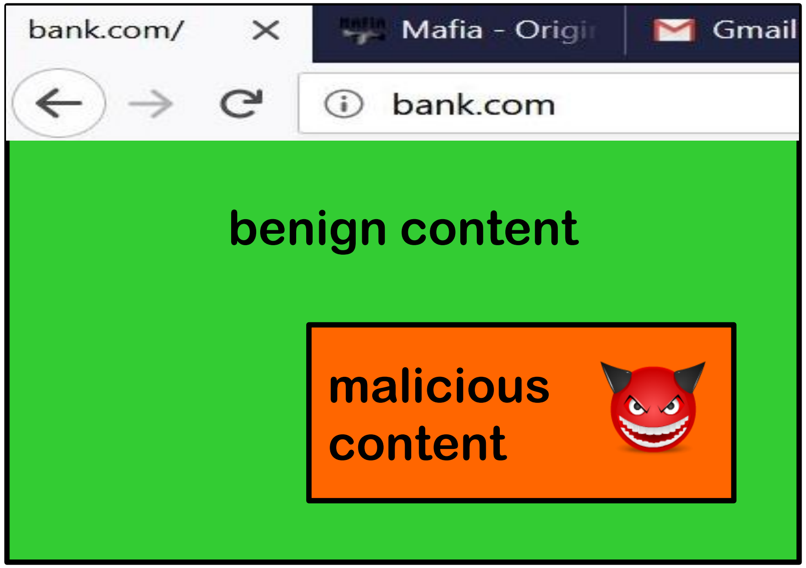
We can secure connections between computers 1000s of miles apart, eg using TLS,

but the remaining 30 cm between user and laptop remain a problem

Beware: blaming the ‘dumb user’ is usually unfair victim blaming.

We should blame computer scientists & engineers for making poor solutions

Attacker model (1) : malicious content on benign site

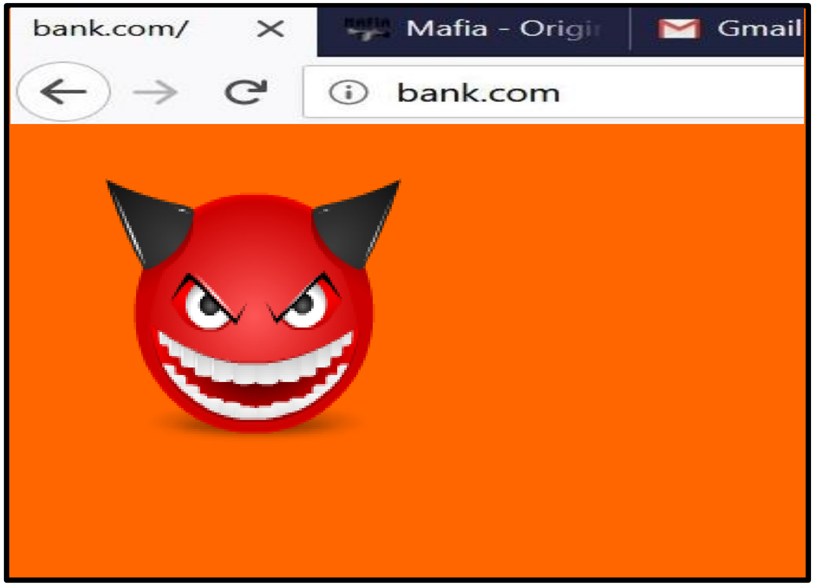


Such malicious content can be

1. 3rd party iframe (intentionally included, separated with SOP)
2. user-provided content

   (e.g. Facebook post; same-origin, so SOP imposes no restrictions)
3. injected with HTML injection or XSS

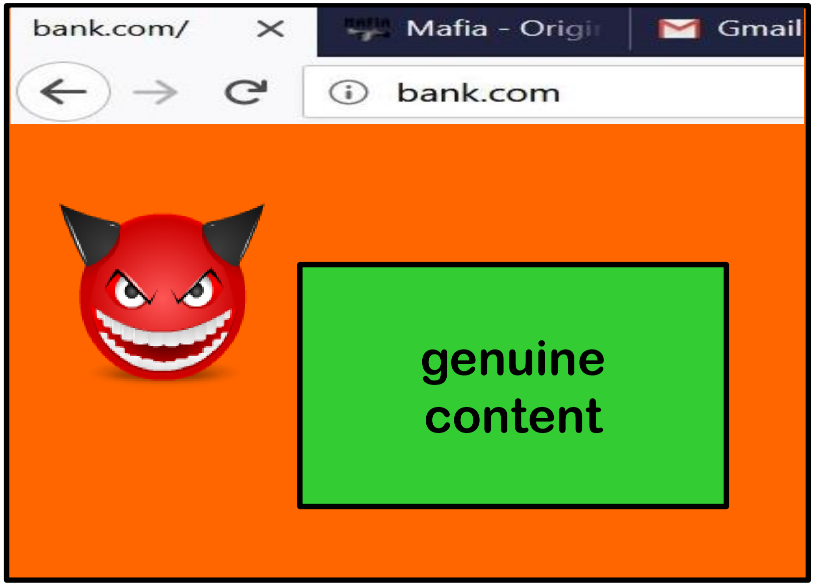
Attacker model (2) : a malicious website



Malicious site could for instance phish for logins & passwords.

It could also include malicious links to the attacked website, eg for CSRF attacks

Attacker model (3): malicious website with genuine iframe



Does SOP help here?

Yes, SOP protects against malicious site from observing or messing with trusted content – and vice versa

- but, as we will see, user can still be misled

Would you trust these URLs?

- `https://www.paypal.com:get_request%2Eupdate&id=234782&`

  Recall that a URL can have the form

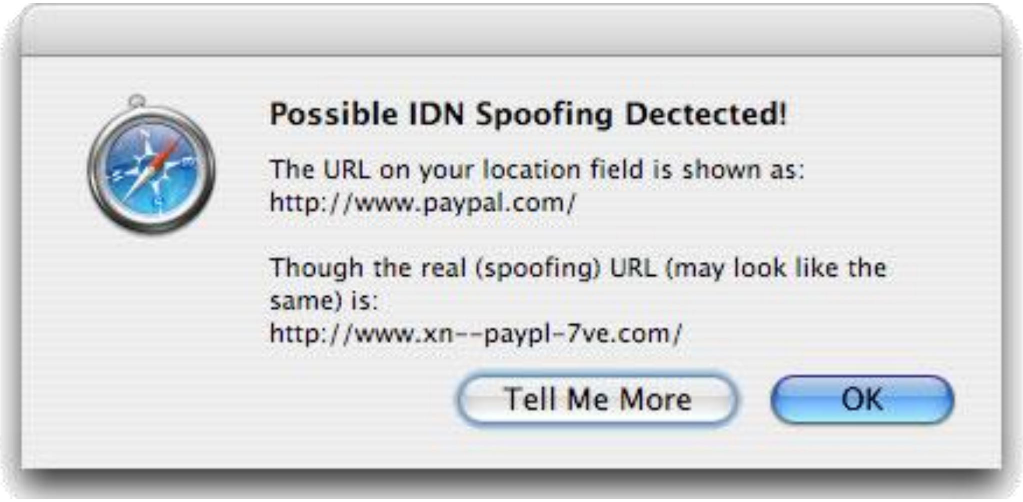
  `https://username:password@host/....`

  So what is the domain we are accessing?

- `https://www.paypal.com`

  How do you know that the first `p` *is not a Cyrillic character?*

Browser warnings – about strange character sets



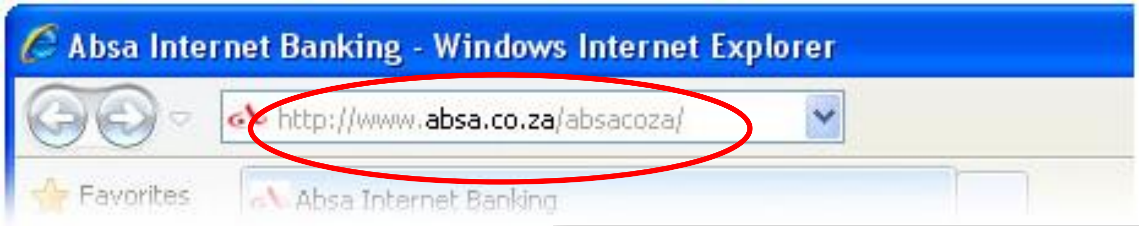
Possible IDN Spoofing Dectected!

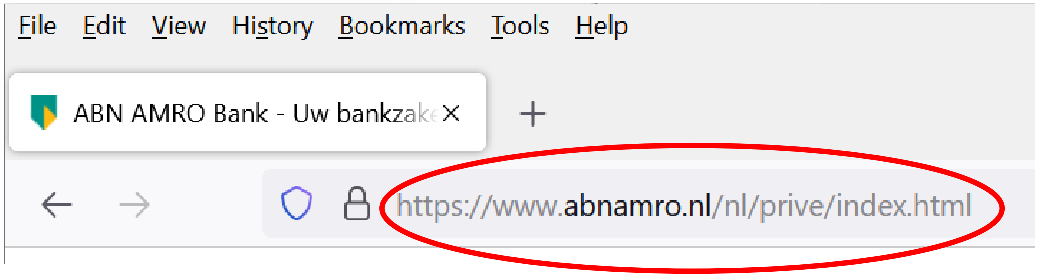
The URL on your location field is shown as:
http://www.paypal.com/

Though the real (spoofing) URL (may look like the same) is:
http://www.xn--paypl-7ve.com/

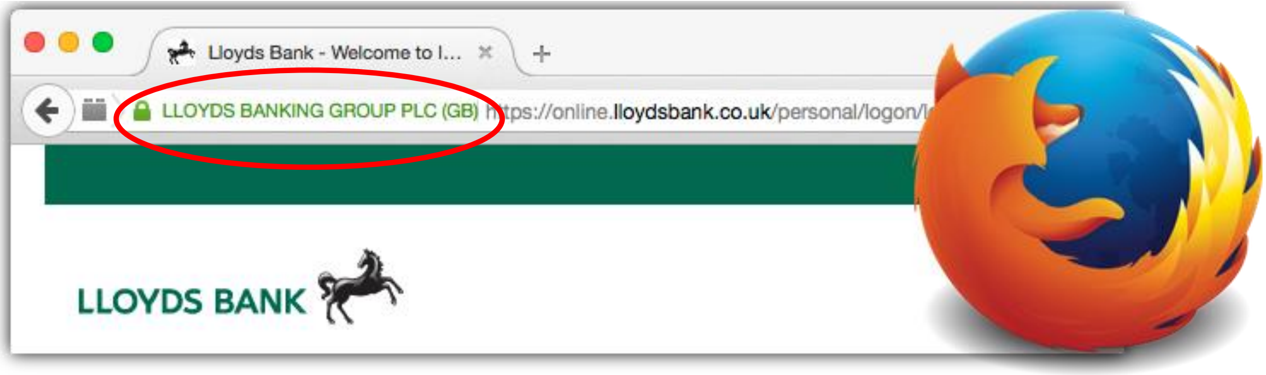
Punycode encoding of unusual characters

Highlighting **domain name** in the address bar


Absa Internet Banking - Windows Internet Explorer
http://www.absa.co.za/absacoza/
Favorites
Absa Internet Banking


File Edit View History Bookmarks Tools Help
ABN AMRO Bank - Uw bankzak
https://www.abnamro.nl/nl/prive/index.html

Alternative: show the organisation name from the certificate


Lloyds Bank - Welcome to l...
LLOYDS BANKING GROUP PLC (GB) https://online.lloydsbank.co.uk/personal/logon/l
LLOYDS BANK

URL obfuscation attacks

Attacker tries to confuse the user (in e.g. phishing attack) by

- including a username before the domain name

  `https://www.visa:com@%32%32%30%2E%36%38%2E%32%31%34%2E...`

  which translates to the IP address `220.68.214.213`

- using strange Unicode characters in a homograph attacks

  `https://paypal.com` with a Cyrillic `p`

Countermeasures:

1. Punycode: encode Unicode as ASCII to reveal funny characters

   `www.xn-pypal-4ve.com`

2. Domain highlighting: show which part of URL is the domain name

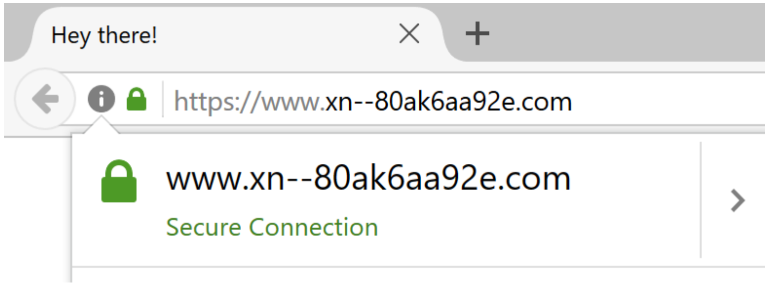
Browser bugs may offer more opportunities to confuse users.

- A bug in Internet Explorer displayed URLs with null character, eg. `http://paypal.com%00@mafia.com`, incorrectly

Newer homograph attack [2017, still works in some browsers]

Some browsers display `https://xn--80ak6aa92e.com`

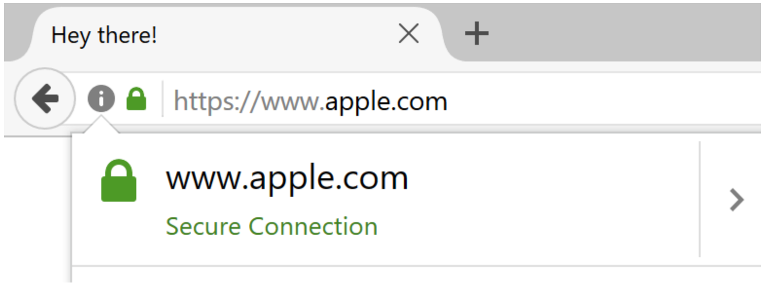
as `apple.com`



https://www.xn--80ak6aa92e.com

www.xn--80ak6aa92e.com

Secure Connection



https://www.apple.com

www.apple.com

Secure Connection

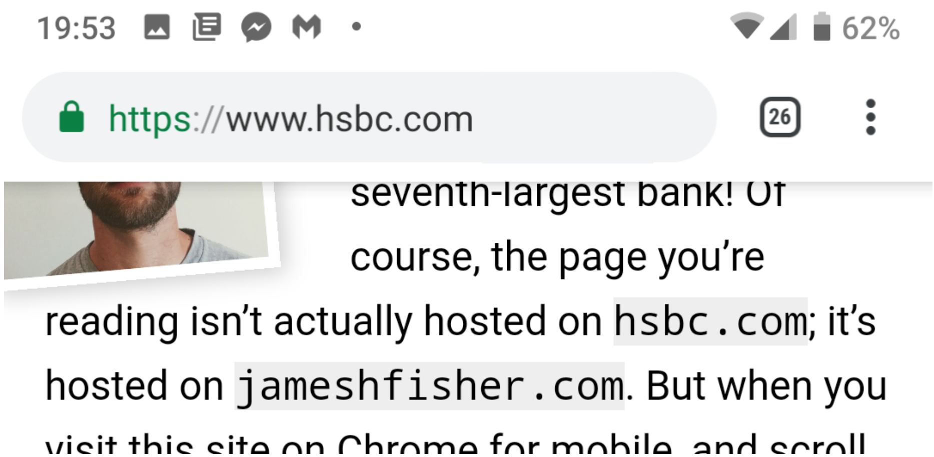
Problem:

browser uses puny encoding if URL mixes several characters sets, but not if *all* characters are from *one* - unusual - character set

See https://www.xudongz.com/blog/2017/idn-phishing/

For you to do: check if this attack works in the browser(s) you use.

UI confusion on mobile phones [2019]

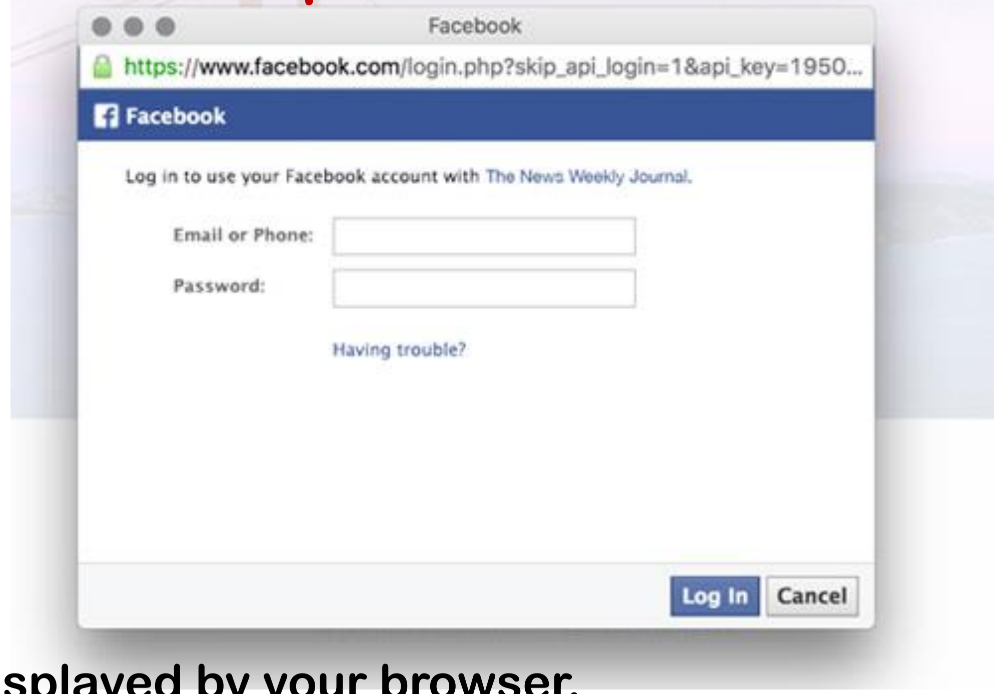


seventh-largest bank! Of course, the page you're reading isn't actually hosted on `hsbc.com`; it's hosted on `jameshfisher.com`. But when you visit this site on Chrome for mobile, and scroll

Chrome on mobile phone hides URL bar when you scroll down. Attacker can abuse this feature to display a fake URL bar.

See https://jameshfisher.com/2019/04/27/the-inception-bar-a-new-phishing-method/

UI confusion on desktops [2019]



https://www.facebook.com/login.php?skip_api_login=1&api_key=1950...

Log in to use your Facebook account with The News Weekly Journal.

Is this pop-up window legit?

The URL is a https-link to facebook.com; clicking lock shows valid certificate

No, this is not a pop-up window displayed by your browser, but a fake pop-up rendered inside a malicious webpage

How can you tell?

You can move this ‘pop-up window’ inside the webpage window but you cannot drag it outside of the browser window

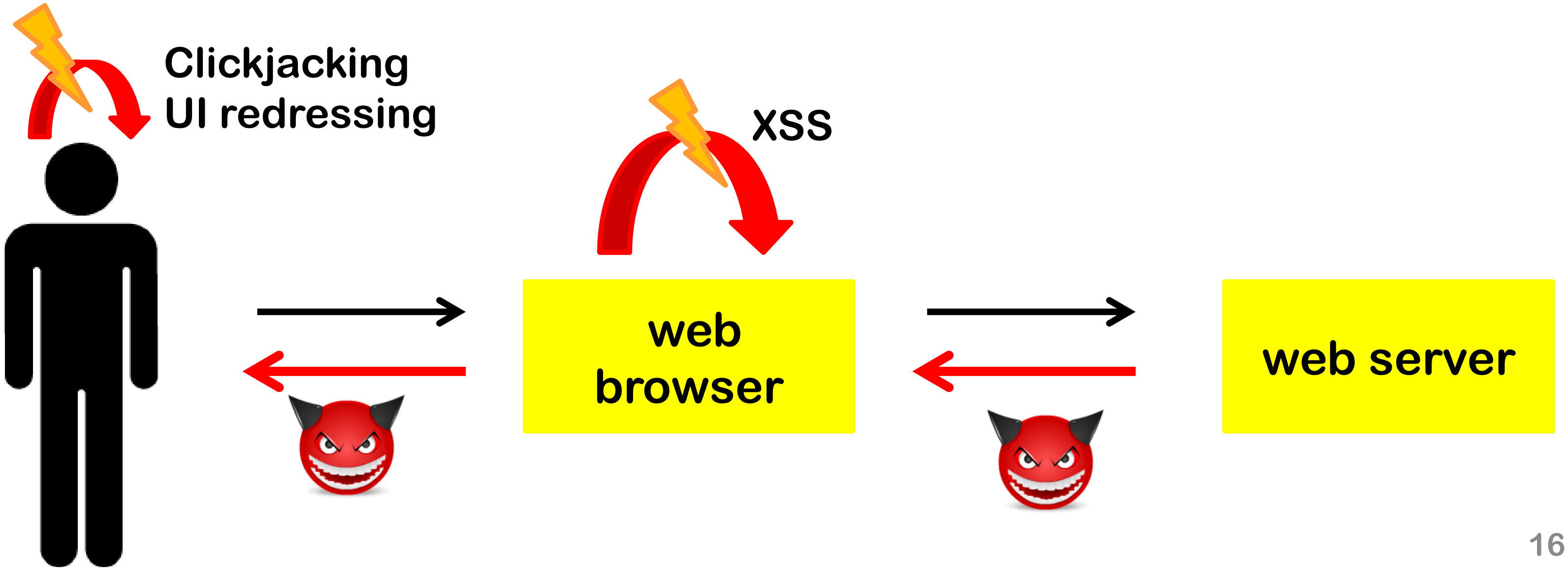
See https://myki.com/blog/facebook-login-phishing-campaign
and or https://youtu.be/nq1gnvYC144

Click-jacking & UI redressing

UI	=	User Interface
UX	=	User Experience
HMI	=	Human-Machine Interface

Click-jacking & UI redressing

- These attacks try to confuse the user into unintentionally doing something, such as
  - clicking some link
  - providing text input to some fields
- These attacks abuse *trust that users have in a webpage and their browser*
  - ie. the trust that users have in what they see
  - What you see may not be what it is!



Click-jacking & UI redressing

Terminology is very messy

- Click-jacking and UI redressing can be regarded as synonyms, but some people see UI redressing as a way to achieve clickjacking, while others see click-jacking as an ingredient in UI redressing

- To add to the confusion, these attacks often come in combination with CSRF or XSS

Basic click-jacking

Make the victim unintentionally click on some link

```
<a onMouseUp=window.open("http://mafia.org/")
 href="http://www.police.nl">Trust me, it is safe to
 click here, you will simply go to police.nl</a>
```

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

Why would attacker want to do this?

- Some unwanted side-effect of clicking the link
  Especially if user is automatically authenticated by the target website (thanks to cookie), ie. CSRF
- Click fraud

Business model for click jacking: click fraud

- Web sites that publish ads are paid for the number of click-throughs (ie, number of visitors that click on these ads)

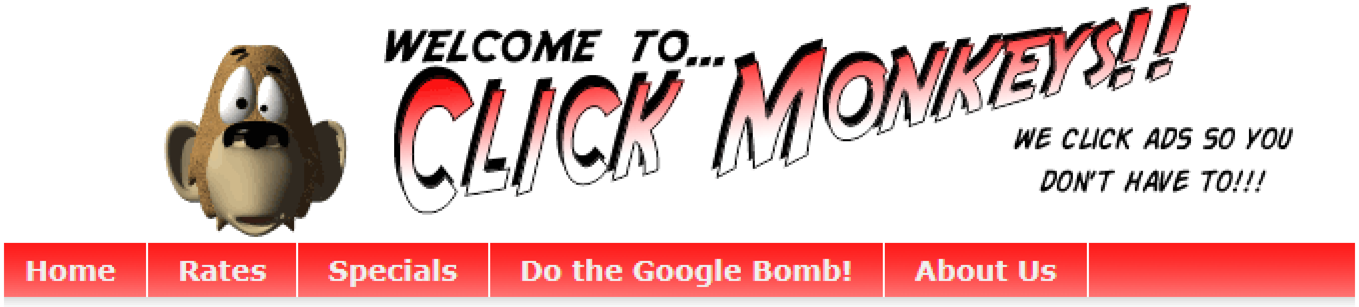
- Click fraud: attacker tries to generate lots of clicks on ads, that are not from genuinely interested visitors

- Motivations for attacker
  1. generate revenue for web site hosting the ad
  2. generate costs for a competitor who has to pay for clicks on their advertisements?

Click fraud

Other forms of click fraud (apart from click-jacking)

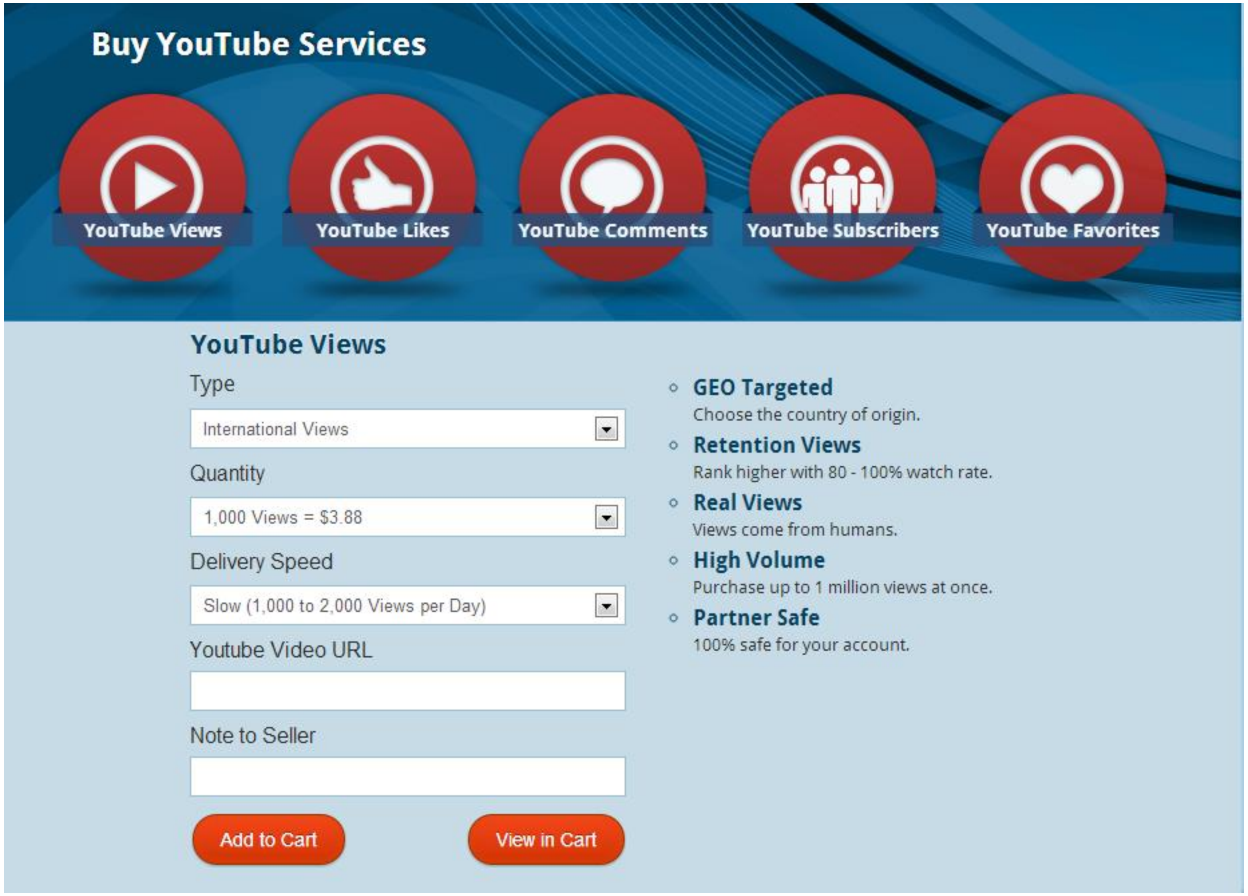
- Click farms (hiring individuals to manually click ads)
- Pay-to-click sites (pyramid schemes created by publishers)



- Click bots (hijacked computers in botnet, running software to automate clicking)

Criminal business models: YouTube views

Alternative business model to click fraud: generate & sell views, likes, … for websites that ranks results based on views, likes, …



YouTube Views

Type: International Views

Quantity: 1,000 Views = $3.88

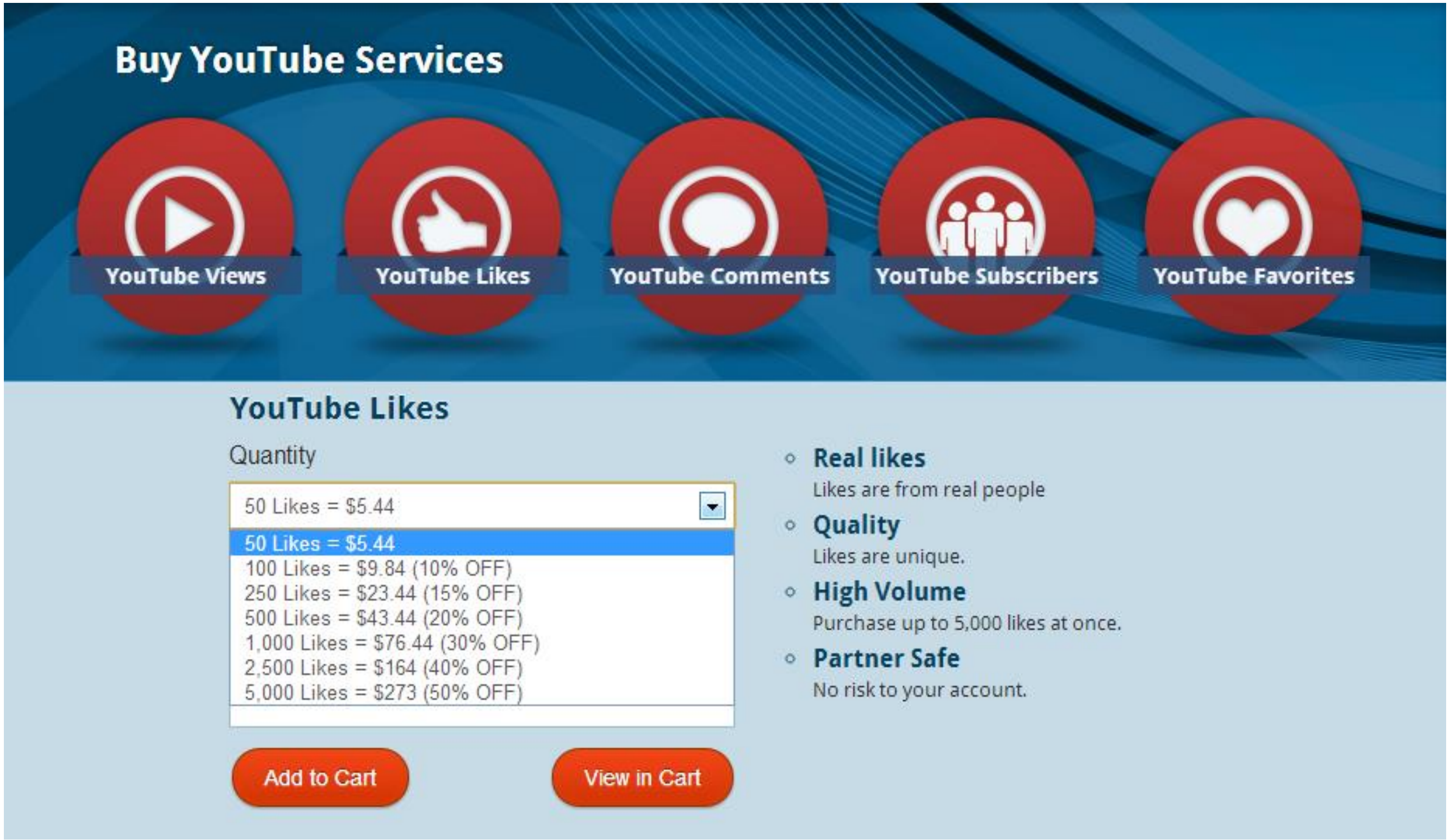
Delivery Speed: Slow (1,000 to 2,000 Views per Day)

Youtube Video URL:

Note to Seller:

- **GEO Targeted**
  Choose the country of origin.
- **Retention Views**
  Rank higher with 80 - 100% watch rate.
- **Real Views**
  Views come from humans.
- **High Volume**
  Purchase up to 1 million views at once.
- **Partner Safe**
  100% safe for your account.

Criminal business models: YouTube likes



YouTube Likes

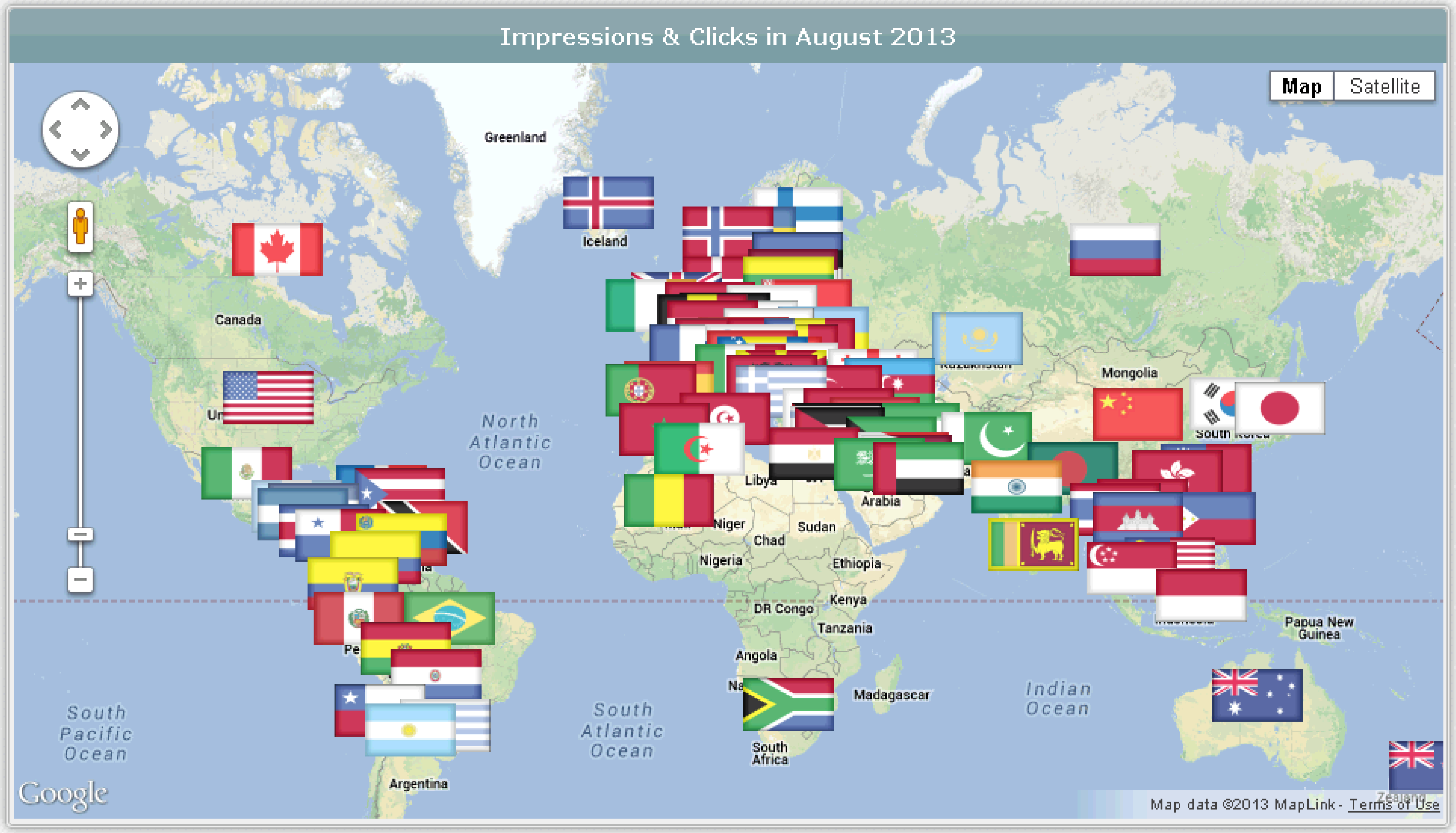
Quantity

50 Likes = $5.44

50 Likes = $5.44
100 Likes = $9.84 (10% OFF)
250 Likes = $23.44 (15% OFF)
500 Likes = $43.44 (20% OFF)
1,000 Likes = $76.44 (30% OFF)
2,500 Likes = $164 (40% OFF)
5,000 Likes = $273 (50% OFF)

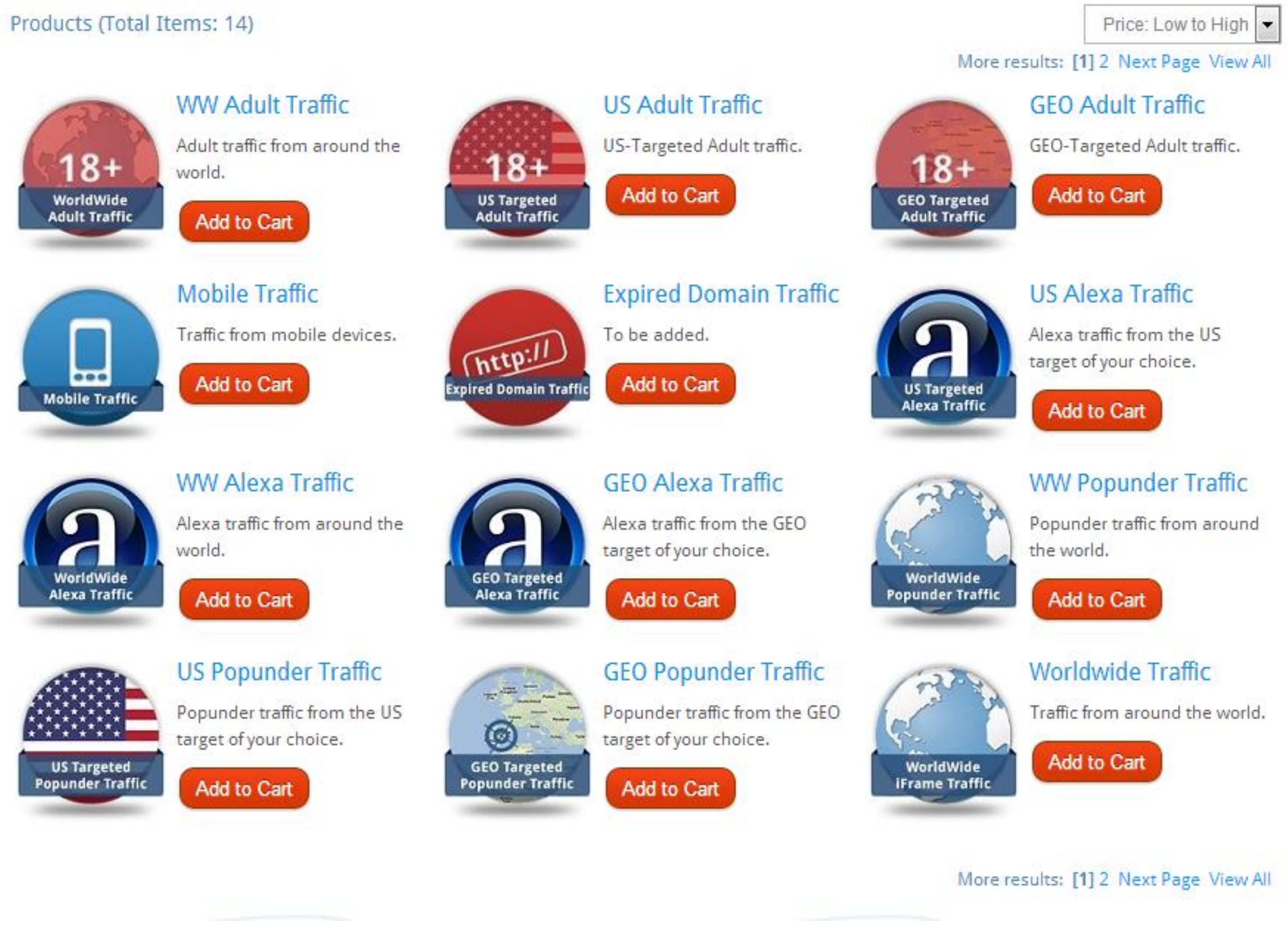
- **Real likes**
  Likes are from real people
- **Quality**
  Likes are unique.
- **High Volume**
  Purchase up to 5,000 likes at once.
- **Partner Safe**
  No risk to your account.

Criminal business models: selling traffic or clicks



Impressions & Clicks in August 2013

Criminal business models: selling traffic or clicks


18+
WorldWide Adult Traffic
18+
US Targeted Adult Traffic
18+
GEO Targeted Adult Traffic
Mobile Traffic
http://
Expired Domain Traffic
US Targeted Alexa Traffic
WorldWide Alexa Traffic
GEO Targeted Alexa Traffic
WorldWide Popunder Traffic
US Targeted Popunder Traffic
GEO Targeted Popunder Traffic
WorldWide iFrame Traffic
Add to Cart

Products (Total Items: 14)

Price: Low to High

More results: [1] 2 Next Page View All

WW Adult Traffic

Adult traffic from around the world.

US Adult Traffic

US-Targeted Adult traffic.

GEO Adult Traffic

GEO-Targeted Adult traffic.

Mobile Traffic

Traffic from mobile devices.

Expired Domain Traffic

To be added.

US Alexa Traffic

Alexa traffic from the US target of your choice.

WW Alexa Traffic

Alexa traffic from around the world.

GEO Alexa Traffic

Alexa traffic from the GEO target of your choice.

WW Popunder Traffic

Popunder traffic from around the world.

US Popunder Traffic

Popunder traffic from the US target of your choice.

GEO Popunder Traffic

Popunder traffic from the GEO target of your choice.

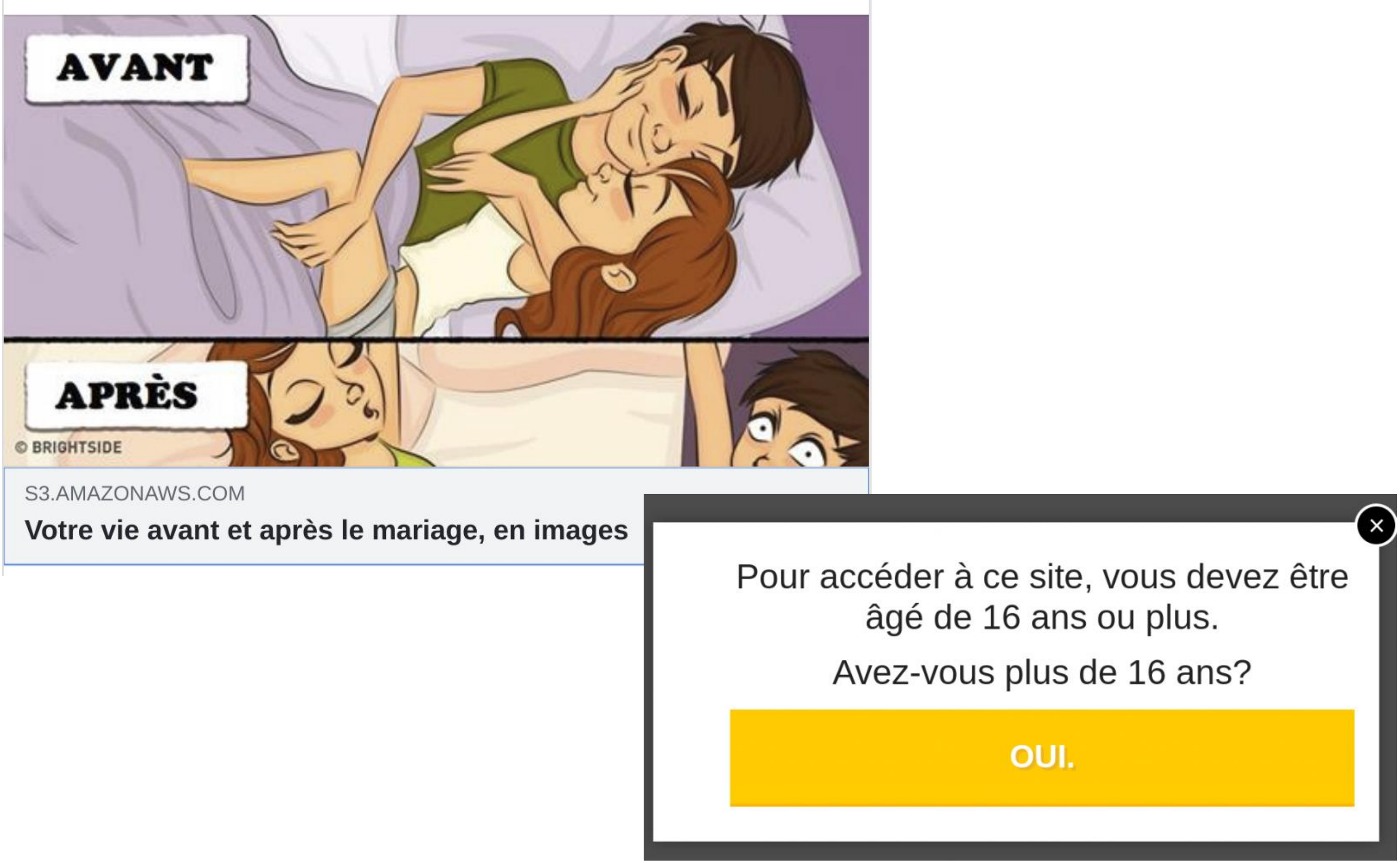
Worldwide Traffic

Traffic from around the world.

More results: [1] 2 Next Page View All

Example clickjacking attack:

with age confirmation check



S3.AMAZONAWS.COM

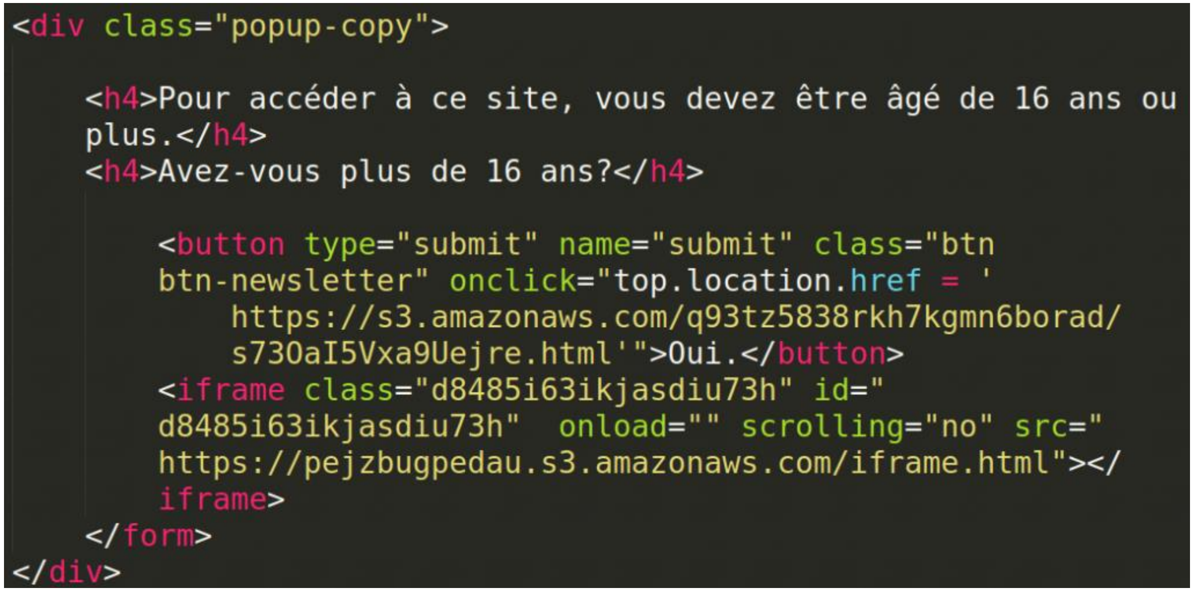
Votre vie avant et après le mariage, en images

Pour accéder à ce site, vous devez être âgé de 16 ans ou plus.

Avez-vous plus de 16 ans?

Example clickjacking attack

Inspecting HTML source to see what you are actually clicking



```
<div class="popup-copy">

    <h4>Pour accéder à ce site, vous devez être âgé de 16 ans ou
    plus.</h4>
    <h4>Avez-vous plus de 16 ans?</h4>

        <button type="submit" name="submit" class="btn
        btn-newsletter" onclick="top.location.href = '
            https://s3.amazonaws.com/q93tz5838rkh7kgmn6borad/
            s730aI5Vxa9Uejre.html'">Oui.</button>
        <iframe class="d8485i63ikjasdiu73h" id="
        d8485i63ikjasdiu73h"  onload="" scrolling="no" src="
        https://pejzbugpedau.s3.amazonaws.com/iframe.html"></
        iframe>
    </form>
</div>
```

Inspecting contents of these Amazon S3 buckets leads to

https://mobile.facebook.com/v2.6/dialog/share?app_id=283197842324324
&href=https://example.com&in_iframe=1&locale=en_US&mobile_iframe=1

Example clickjacking attack

Clicking age confirmation shares a post on Facebook.

Such clickjacking can get you many likes or shares!

Attack only worked in the Facebook mobile app,
not in a normal browser

- NB the Facebook app ‘is’ (or ‘includes’) a web-browser

Read the description at

https://malfind.com/index.php/2018/12/21/how-i-accidentaly-found-clickjacking-in-facebook/

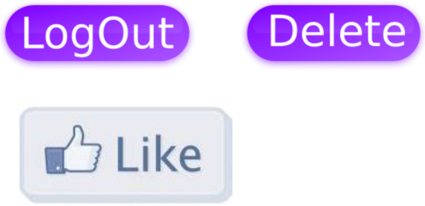
UI redressing

Attacker creates a malicious web page that includes elements of a target website, esp. links victims can click.

- With iframe (inline frame) with content from attacked website
  - iframes allow flexible nesting, cropping, and overlapping

Two approaches

1. “steal” a button with non-specific text



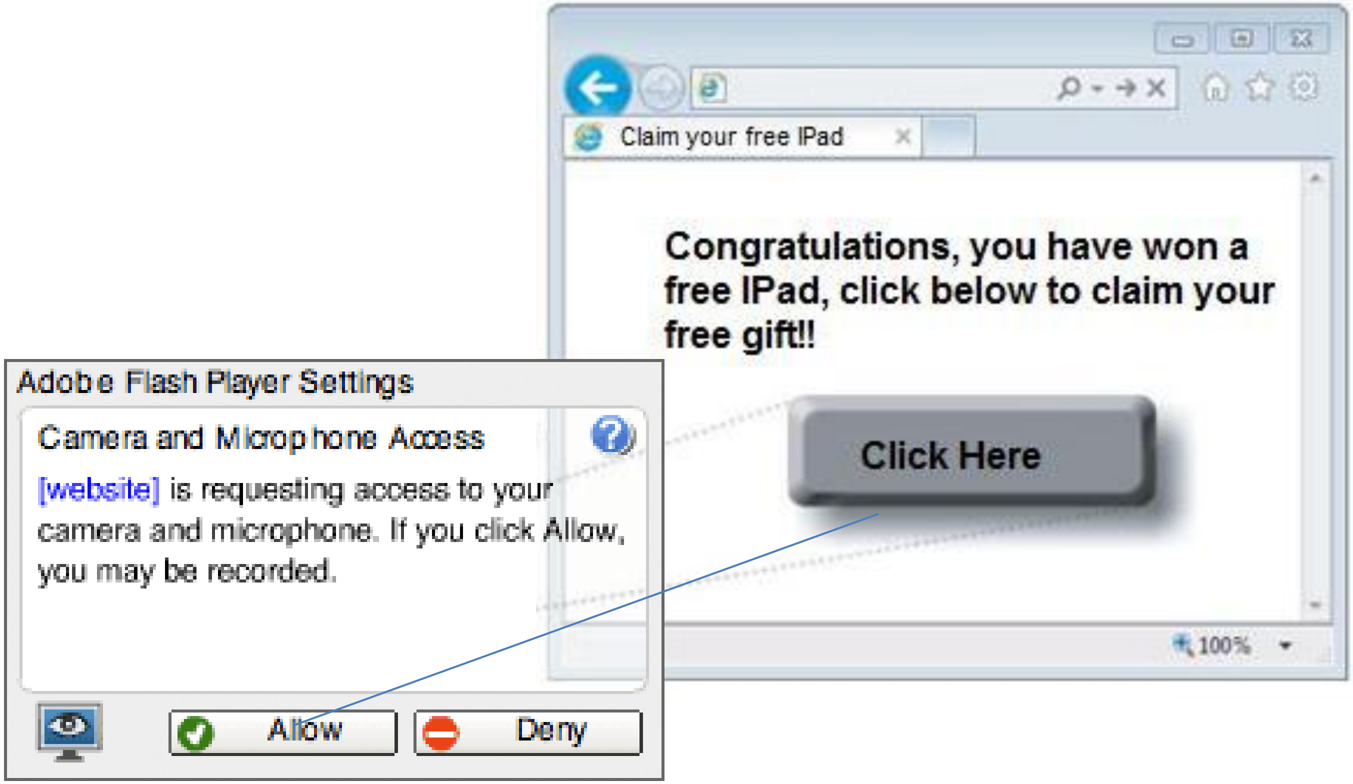
2. make a iframe transparent

NB esp. 1 looks a lot like CSRF, as we’ll discuss later

Old UI redressing example

Tricking users into altering security settings of Flash

- Load Adobe Flash player settings into an invisible iframe
- Click will give permission for any Flash animation to use the computer's microphone and camera



Adobe Flash Player Settings

Camera and Microphone Access

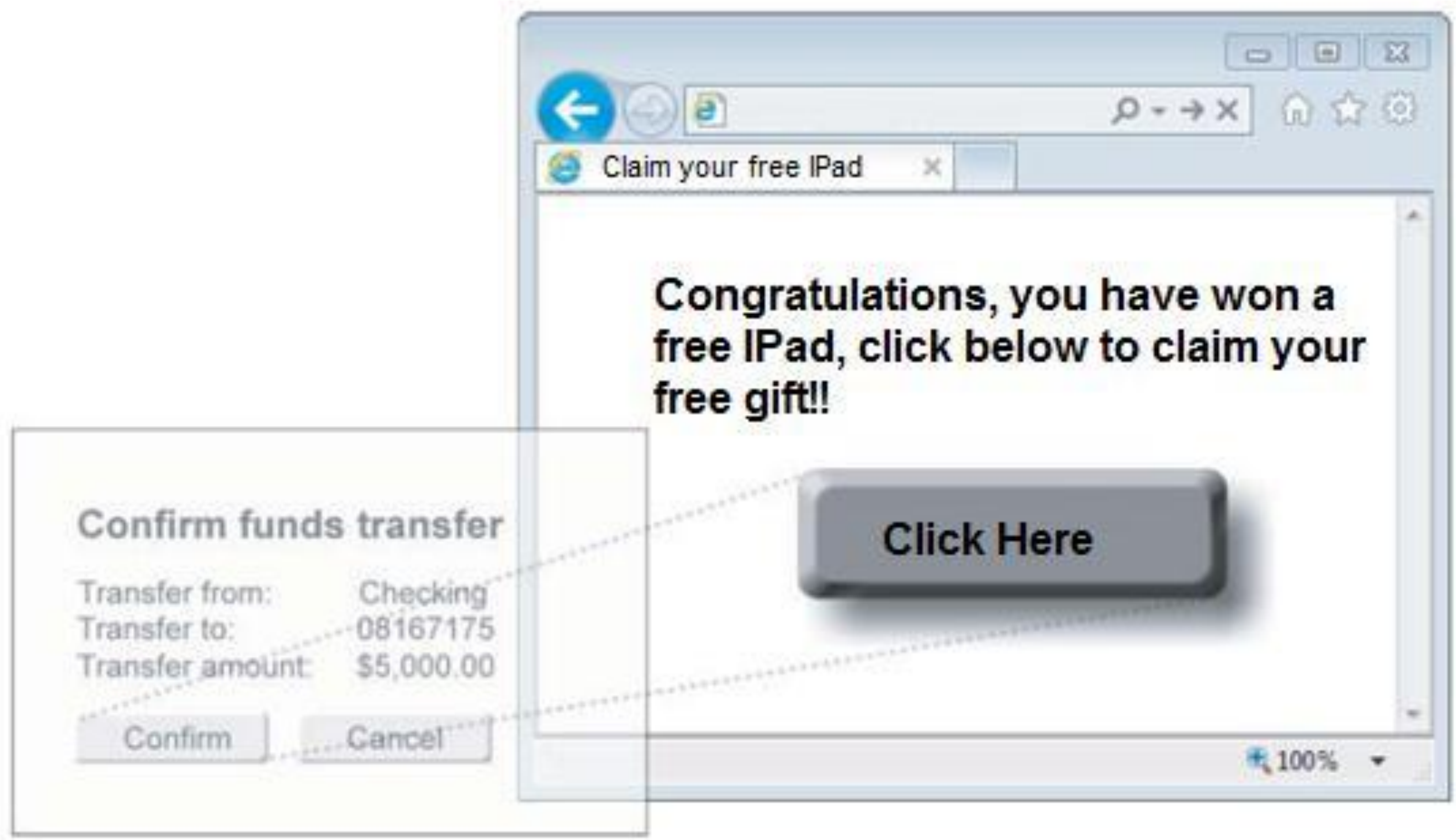
[website] is requesting access to your camera and microphone. If you click Allow, you may be recorded.

Claim your free IPad

Congratulations, you have won a free IPad, click below to claim your free gift!!

UI redressing example

Trick users into confirming a financial transaction



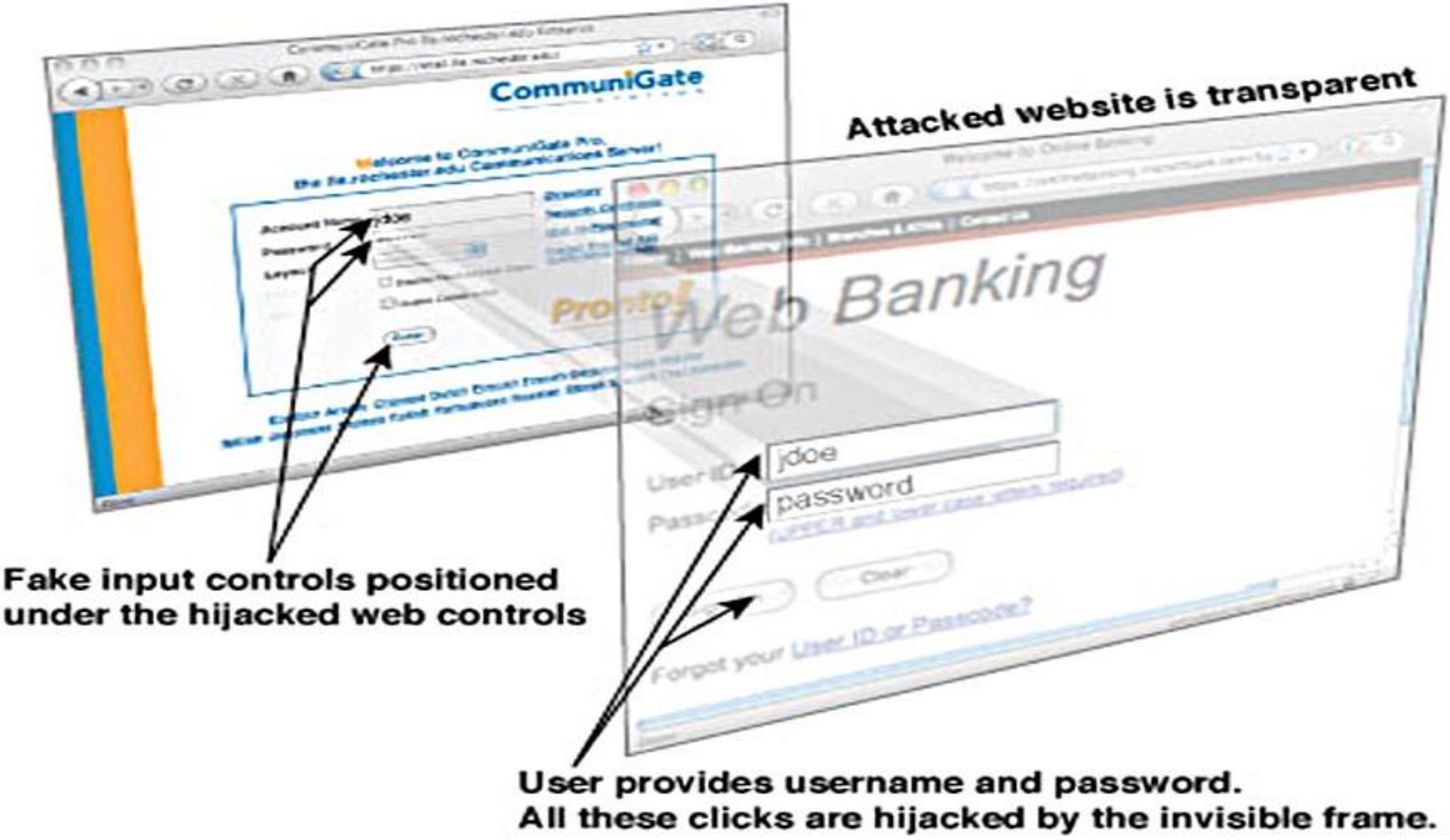
Congratulations, you have won a free IPad, click below to claim your free gift!!

Confirm funds transfer

Transfer from: Checking
Transfer to: 08167175
Transfer amount: $5,000.00

UI redressing example

Trick users to login to a banking website



Attacked website is transparent

Fake input controls positioned
under the hijacked web controls

User provides username and password.
All these clicks are hijacked by the invisible frame.

Click-jacking and UI redressing: abusing trust

- These attacks abuse trust users have in a webpage
  - in what they *see* in their browser

- These attacks also abuse trust the web server has in browsers
  - Web server trusts that all actions from the browser performed *willingly* & *intentionally by the user*

- Some browser will prevent users from interacting with transparent content

  Check if your browsers does at

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

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

Variations of click-jacking

- like-jacking and share-jacking 👍 Share
- cursor-jacking
  (See https://www.cs.ru.nl/~erikpoll/websec/demo/cursor-jacking.html)
- file-jacking (unintentional uploads in Google Chrome)
- event-jacking
- class-jacking
- double click-jacking
- content extraction
- pop-up blocker bypassing
- stroke-jacking
- event recycling
- SVG (Scalable Vector Graphics) masking
- tap-jacking on Android phones
- …

Countermeasures against click-jacking & UI redressing

Frame busting

Countermeasure to prevent being included as iframe: webpage tries to bust any frames it is included in

- Example JavaScript code for frame busting

```
if (top!=self){
      top.location.href = self.location.href
}
```

  - `top` is the top or outer window in the DOM;
    `self` is the current window
  - If an iframe executes this code, it will make itself the top window.
  - For a demo, see
    https://www.cs.ru.nl/~erikpoll/websec/demo/framebusting1.html
    which includes a frame-busting iframe
    https://www.cs.ru.nl/~erikpoll/websec/demo/framebuster.html

Lots of variations possible, some more robust than others

Busting frame busting

Recall sandboxing of iframes (discussed 2 weeks ago):
This allows attacker to restrict capabilities of a victim iframe

- eg. iframe be disallowed to change `top.location`

This can block the framebusting

- Example HTML code for sandboxing:

```
<iframe sandbox="allow-scripts allow-forms"
            src="facebook.html"> </iframe>
```

  - `allow-scripts`: allow scripts
  - `allow-forms`: allow forms
  - there is no `allow-top-navigation`, so the iframe is not allowed to change of `top.location`

For a demo, see
https://www.cs.ru.nl/~erikpoll/websec/demo/framebusting2.html

Better solution: X-Frame options

`X-Frame-Options` in HTTP response header introduced to indicate if webpage can be loaded as iframe

- Possible values

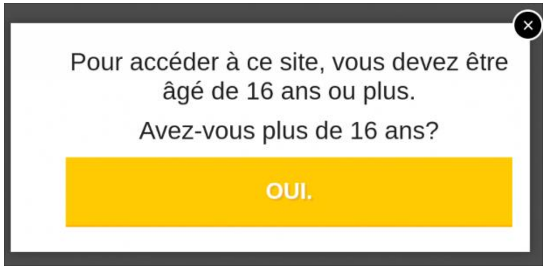
`DENY`	never allowed
`SAMEORIGIN`	only allowed if other page has same origin
`ALLOW-FROM <url>`	only allowed for specific URL (Only ?)

- Simpler than using JavaScript to do frame busting, and cannot be disabled with HTML sandboxing

- CSP (Content Server Policy) also provides ways to do this, but given the complexity of CSP, many sites continue to use `X-Frame-Options`

Example: website with age confirmation check

Why doesn't Facebook use `X-Frame-Options` to prevent malicious inclusion of share or like buttons?



Pour accéder à ce site, vous devez être âgé de 16 ans ou plus.

Avez-vous plus de 16 ans?

Facebook does set `X-Frame-Options` to `DENY`, but only for content served to a normal web browser, not for content sent to their mobile facebook app

See also

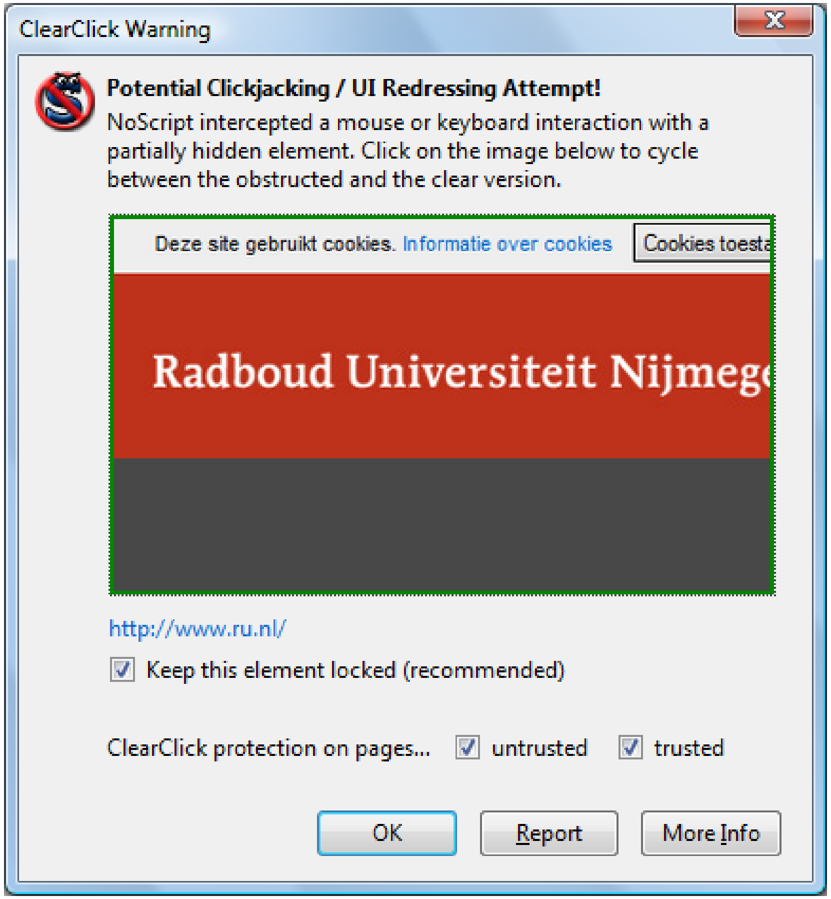
https://malfind.com/index.php/2018/12/21/how-i-accidentaly-found-clickjacking-in-facebook/

Browser protection against UI redressing

Firefox extension NoScript has a ClearClick option, that warns when clicking or typing on hidden elements

How this works:

- Activated when user clicks on object in an iframe
- Comparison made between screenshots of
  a) the web page
  b) the web page with any opaqueness/transparency in iframe turned off
- If screenshots differ, user is warned and screenshot is shown so user can evaluate it themselves



ClearClick Warning

Potential Clickjacking / UI Redressing Attempt!

NoScript intercepted a mouse or keyboard interaction with a partially hidden element. Click on the image below to cycle between the obstructed and the clear version.

Deze site gebruikt cookies. Informatie over cookies
Cookies toesta
Radboud Universiteit Nijmege

http://www.ru.nl/

☑ Keep this element locked (recommended)

ClearClick protection on pages... ☑ untrusted ☑ trusted

CSRF revisited

Recall : CSRF abuses cookies without stealing them

Attacker sets up malicious website mafia.com with link to bank.com

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

If victim visits mafia.com and click this link,
then if they are logged in to the back,
    this request will be sent with the victim's cookies for bank.com

CSRF

- Ingredients
  - malicious link or JavaScript on attacker’s website
  - automatic authentication by cookie at targeted website
- Requirements
  - the victim must have a valid cookie for the attacked website
  - that site must have actions which only require a single HTTP request

- It’s a bit like click-jacking, except
  - it does not involve UI redressing
  - if JavaScript is used, it is more than just clicking a link

CSRF on GET vs POST requests

Action on the targeted website might need a POST or GET request

- Recall: GET parameters in URL, POST parameters in body

- For action with a GET request:
  - Easy!
  - Attacker can even use an image tag `<img..>` to execute request

```
<img scr="http://bank.com/transfer?amount=1000
                                   &toAccount=52.12.57.762">
```

- For action with a POST request:
  - Trickier!
  - Attacker cannot append data in the URL
  - Instead, attackers can use JavaScript on own website to make a form which then results in a POST request to the target website

CSRF of a POST request using JavaScript

If bank.com uses

```
<form action="transfer.php" method="POST">
    To: <input type="text" name="to"/>
    Amount: <input type="text" name="amount"/>
    <input type="submit" value="Submit"/>
</form>
```

attacker could use

```
<form action="http://bank.com/transfer.php"  method="POST">
  <input type="hidden" name="to" value="52.12.57.762"/>
  <input type="hidden" name="amount" value="1000" />
  <input type="submit"/>
</form>
<script> document.forms[0].submit(); </script>
```

Note: no need for victims to click anything!

The JavaScript code clicks it for them

Countermeasures against CSRF
-
which might also help against clickjacking?

Recall: Countermeasures against CSRF [week 2 & 3]

1. Let client re-authenticate before important actions
2. Keep sessions short
3. Anti-CSRF token [aka Tokenization]
   - an unpredictable CSRF token as hidden parameter in requests that changes every time
4. Looking at the `Referer` or `Origin` headers
5. Setting `SameSite` flag for cookies
6. Let browser add `Sec-Fetch-Site` header to distinguish cross site requests and let your server check these

Which of these help against click-jacking/UI redressing?

- 1&2 obviously help.
- 3 does not help; if mafia.com's webpage loads 'fresh' iframes from bank.com, links inside these iframes probably have valid tokens.
- 4-6 help, but what counts as same site for `SameSite` or cross-origin for `Sec-Fetch-Site` gets confusing! See example on next slide.

CSRF vs UI redressing: defenses

CSRF attack: suppose a webpage from mafia.com (or an HTML email send by mafia) includes a link to bank.

```
<html> ...
    <img scr="http://bank.com/transfer?amount=1000 &toAccount=52.12.57.76"></img>
</html>
```

- If bank cookies are declared as `SameSite`, the browser will not attach these cookies if link is clicked.
- Also, the browser will mark this request as `cross-origin` with `Sec-Fetch-Site`.
- If bank includes anti-CSRF tokens in links, e.g. the link should be
  http://bank.com/transfer?amount=1000 &toAccount=52.12.57.76&token=097123571
  the mafia people have no way of predicting a valid value for that token

So all these defences help against this CSRF attack.

(Btw, it is unlikely that a bank transfer could be done with a simple GET request.)

CSRF vs UI redressing: defenses

UI redressing/clickjacking attack: suppose a webpage from mafia.com includes an iframe from bank.com

```
<html> . . .
   <iframe src=http://bank.com/somepage.html?param=...></iframe>
</html>
```

For the request to retrieve this iframe

- if bank cookies are declared as `SameSite`, the browser will not attach these cookies to that request.
- Also, the browser will mark this request `cross-origin` as `Sec-Fetch-Site`.

Suppose that there are links inside the iframe, i.e. inside somepage.html

- These links might have a valid value for the anti-CSRF token.
- If user click these links, the browser will not attach `SameSite` cookies and declared the request as `cross-origin` with `Sec-Fetch-Site`. This may seem counterintuitive, as the iframe comes from bank.com, but the domain of the webpage, here mafia.com, not the domain of the iframe, determines howthe browser deals with `SameSite` and `Sec-Fetch-Site`

Beware of confusion!

XSS

vs

CSRF

vs

Click-jacking & UI redressing

CSRF vs Click-jacking/UI-redressing

Easy to confuse! Some differences:

- Unlike Click-jacking, CSRF might not need a click
- Unlike UI redressing, CSRF does not involve recycling parts of the target website
  - So frame-busting or `XFRAME-Options` won’t help
  - UI redressing involves a more powerful attacker model

CSRF meets HTML injection & XSS

Instead of attacker using their own site or emails with malicious links for CSRF,

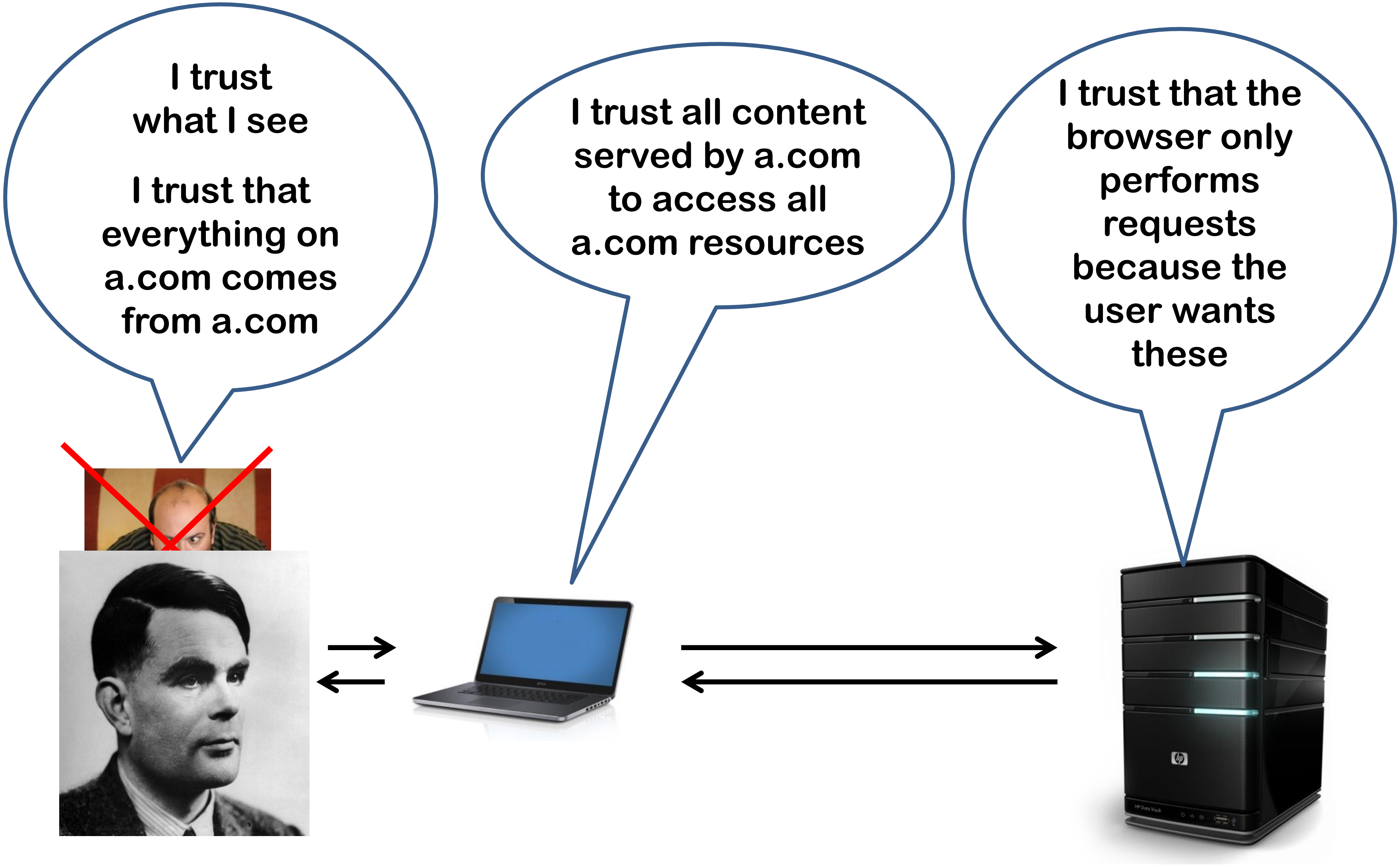
malicious links can also be inserted as content on the vulnerable target site

- Ideally this vulnerable site is target site itself, as user is then guaranteed to be logged in
  - Classic example: malicious link in an amazon.com book review to order books at amazon.com

- This is then *also* an HTML injection attack

- If the CSRF attack uses JavaScript (eg for a POST), then it is *also* a XSS attack

Trust



I trust what I see

I trust that everything on a.com comes from a.com

I trust all content served by a.com to access all a.com resources

I trust that the browser only performs requests because the user wants these

CSRF vs XSS

Easy to confuse! Some differences:

- CSRF does not require JavaScript (for GET actions), XSS always does

- For any JavaScript used:
  - XSS: script is in webpage of the attacked website
  - CSRF: script can be anywhere, also the attacker's website
    - You can use XSS to do CSRF, as shown on previous slide 44, where code will be in the attacked site

Trust: CSRF vs XSS

- CSRF abuses trust of the webserver in the client,
  where client = the web browser *or* its human user
  - The webserver trusts that all actions are actions that the user does willingly and knowingly
- XSS abuses trust of user & browser in the webserver
  - The user & browser trusts that all content of a webpage is really coming from that webserver
    - even though it may include HTML and scripts that are really coming from an attacker
- Clickjacking/UI redressing abuses both types of trust

Root causes

***Why** are web applications often so insecure?*

FUNCTIONALITY vs security

Security is only a secondary concern:

- The primary purpose of any IT system, application, or API is to provide functionality

  The more (general) functionality, the better!

- All this functionality comes with risks.

  Security is about managing these risks.

Companies, developers, and users, all like more of functionality,

even at the expense of less security.

Often security risks may only become clear later.

COMPLEXITY

Root cause of many security problems is complexity

- in technologies, languages, features
- in the interactions between them

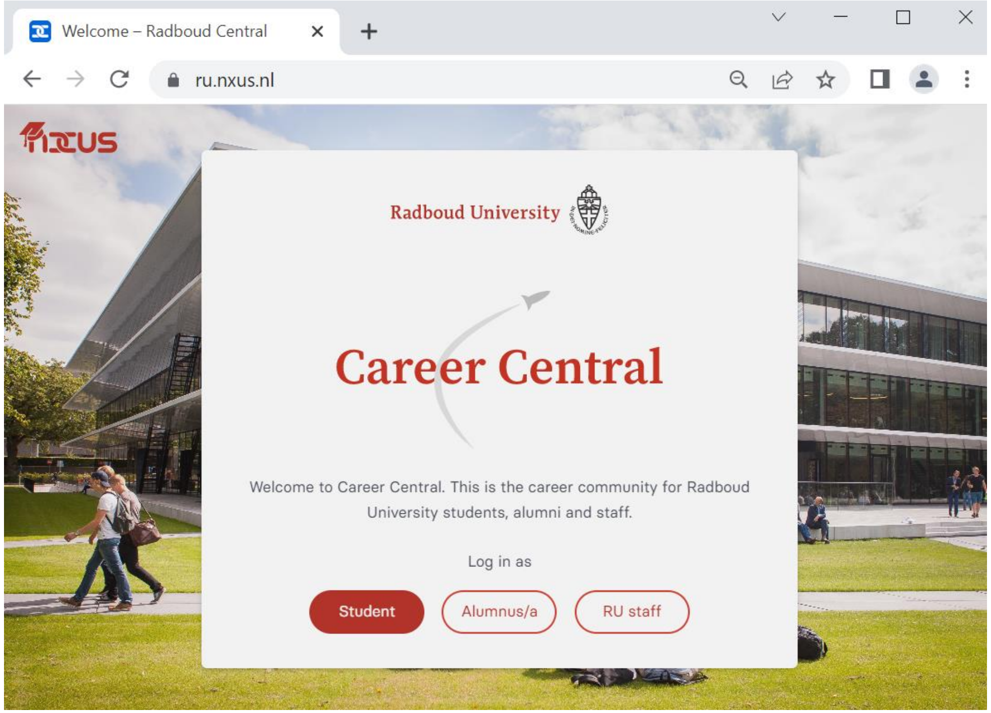
This complexity can be

- hard to use correctly (for users, sys-admins, and programmers)
- may come with unexpected corner cases
- hard to implement incorrectly

Lack of economic incentives

There is often no economic incentive to provide better security

- Making more secure applications takes more time & effort, but are people commissioning them willing to pay? And are companies willing to give programmers more time & training?



Career Central

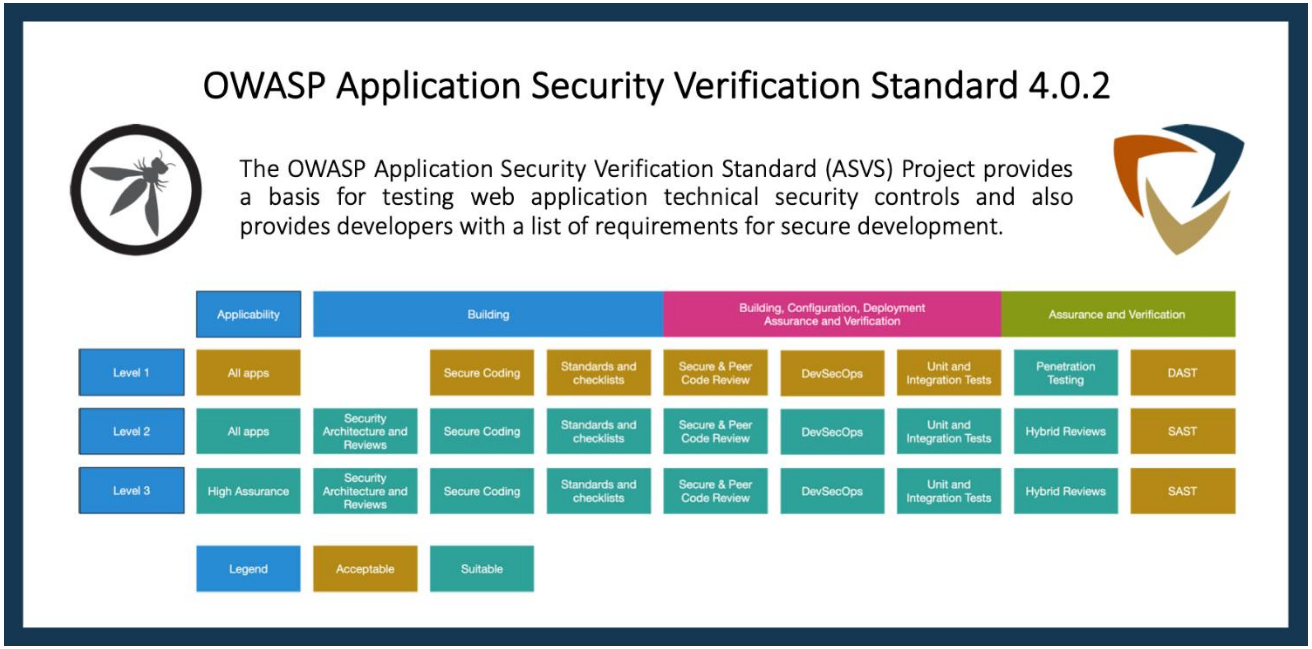
Welcome to Career Central. This is the career community for Radboud University students, alumni and staff.

Log in as

More attacks

OWASP
Open Web Application Security Project

OWASP
TOP 10



OWASP Application Security Verification Standard 4.0.2

The OWASP Application Security Verification Standard (ASVS) Project provides a basis for testing web application technical security controls and also provides developers with a list of requirements for secure development.

	Applicability	Building			Building, Configuration, Deployment Assurance and Verification			Assurance and Verification	
Level 1	All apps		Secure Coding	Standards and checklists	Secure & Peer Code Review	DevSecOps	Unit and Integration Tests	Penetration Testing	DAST
Level 2	All apps	Security Architecture and Reviews	Secure Coding	Standards and checklists	Secure & Peer Code Review	DevSecOps	Unit and Integration Tests	Hybrid Reviews	SAST
Level 3	High Assurance	Security Architecture and Reviews	Secure Coding	Standards and checklists	Secure & Peer Code Review	DevSecOps	Unit and Integration Tests	Hybrid Reviews	SAST

Legend	Acceptable	Suitable

OWASP Top10 & ASVS

There are more attacks than we discussed, but usually variations on the same theme (notably some form of injection)

OWASP produces a well-known OWASP Top 10 of web applications security vulnerabilities

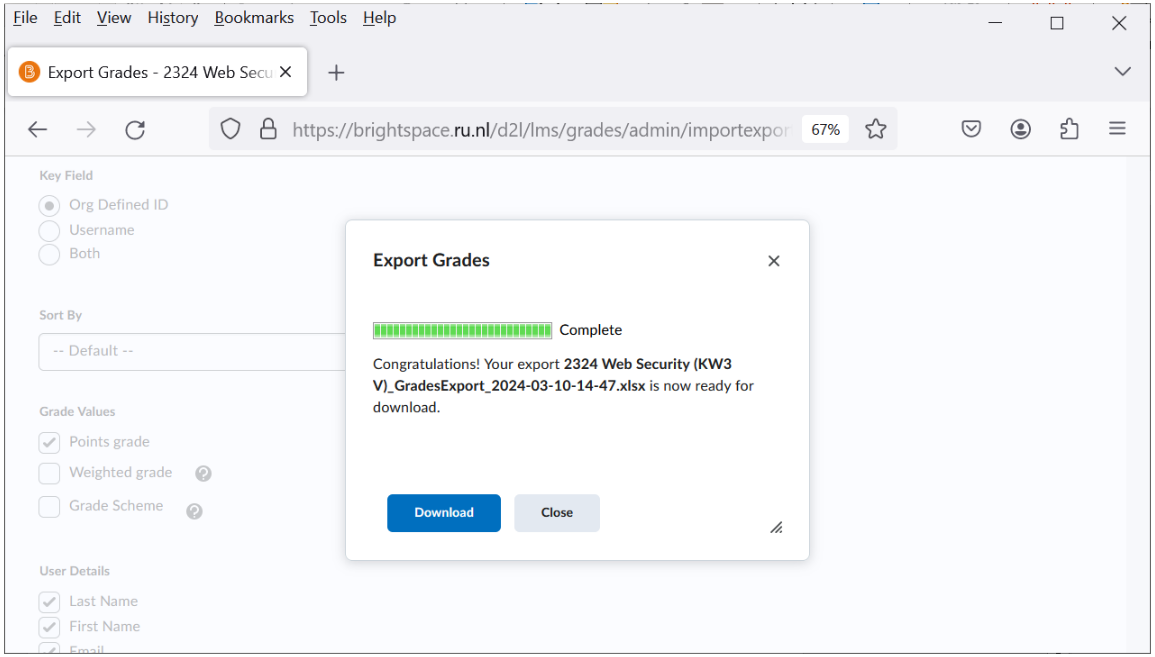
Knowing OWASP Top 10 helps to find flaws & develop more secure applications but better, more structural approach to produce secure web applications: OWASP ASVS (Application Security Verification Standard)

For Dutch speakers & Dutch government agencies, CIP-overheid.nl provides similar standards for 'Grip op SSD (Secure Software Development)'

IDOR

Most promising way to earn bounties with bugs in Brightspace.

Brightspace website provides lots of functionality to view or download information, e.g.


File
Edit
View
History
Bookmarks
Tools
Help
Export Grades - 2324 Web Secu
https://brightspace.ru.nl/d2l/lms/grades/admin/importexpor
67%
Download
Close

Key Field

● Org Defined ID
○ Username
○ Both

Sort By

-- Default --

Grade Values

☑ Points grade
□ Weighted grade
□ Grade Scheme

User Details

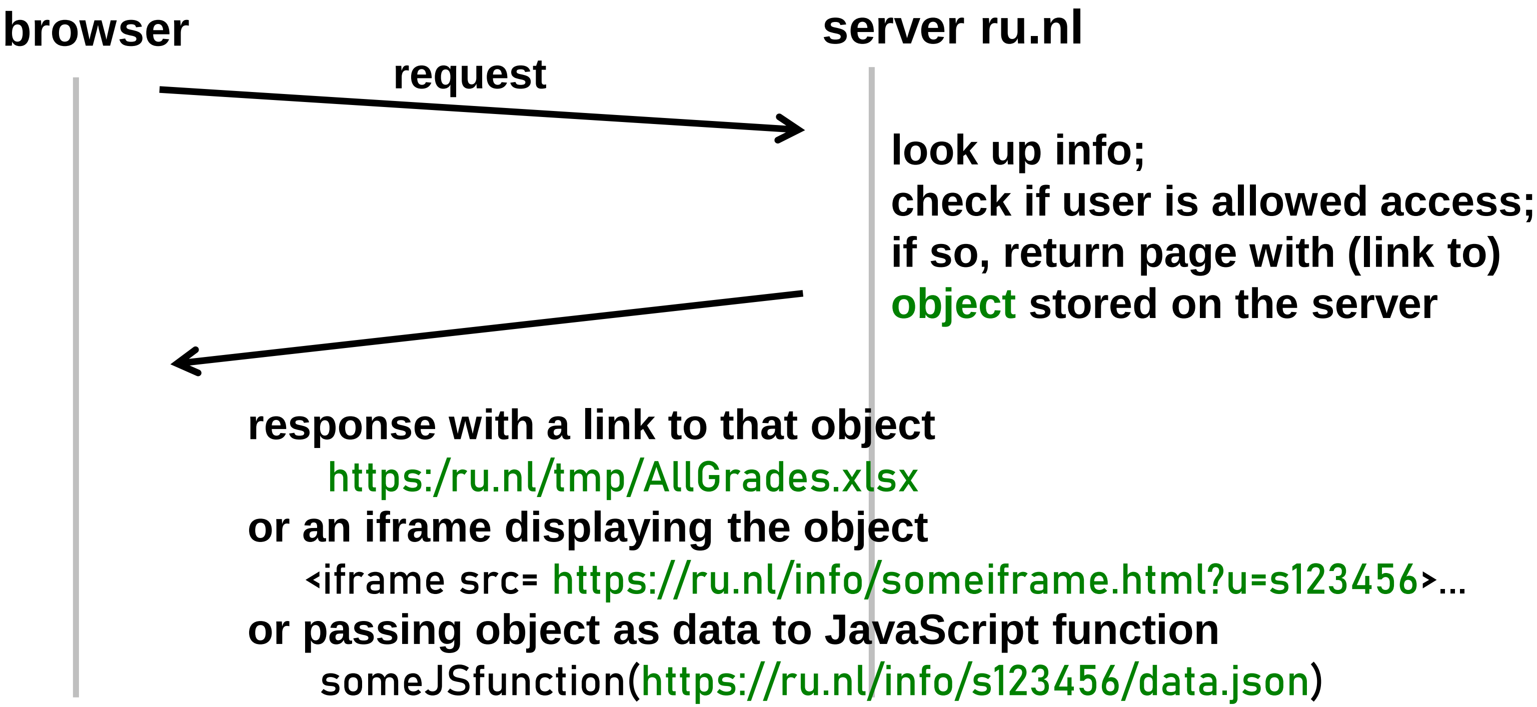
☑ Last Name
☑ First Name
☑ Email

Export Grades

Complete

Congratulations! Your export **2324 Web Security (KW3 V)_GradesExport_2024-03-10-14-47.xlsx** is now ready for download.

IDOR (Insecure Direct Object Reference)



browser

server ru.nl

request

look up info;
check if user is allowed access;
if so, return page with (link to)
object stored on the server

response with a link to that object

https:/ru.nl/tmp/AllGrades.xlsx

or an iframe displaying the object

<iframe src= https://ru.nl/info/someiframe.html?u=s123456>...

or passing object as data to JavaScript function

someJSfunction(https://ru.nl/info/s123456/data.json)

Attacker could modify these links to the object

and by-pass access control to access other objects

Countermeasure: re-do access control checks for every access!

Path traversal can be viewed as a special case of IDOR