Module 5 – Executing JavaScript Bridges

Let’s execute some Java from a WebView

Executing JavaScript Bridges

- Android WebViews have a feature called “JavaScript Bridges”
- It allows JavaScript to execute a whitelisted set of Java methods in the Android application
- Sometimes the whitelisted Java methods do privileged actions on the Android device itself, such as installing arbitrary applications
- This module will focus on executing JavaScript Bridges from a WebView, as well as trying to abuse misconfigured JavaScript Bridges

A JavaScript and a Bridge

MILICIOUS ERECTION LLC

Executing JavaScript Bridges

- Android provides some good documentation and tutorials about how to implement a JavaScript Bridge (aka “JavaScript Interface”) in an Android app
  - https://developer.android.com/guide/webapps/webview#BindingJavaScript
- At a high level, implementing a JavaScript Bridge involves:
  - Add the “JavaScript Interface” class to the WebView
  - Add the methods which can be executed by the WebView / JavaScript
  - Add custom JavaScript to the WebView HTML which executes the methods

Android Developer documentation on JavaScript Bridges

Bind JavaScript code to Android code

When developing a web application that's designed specifically for the `WebView` in your Android app, you can create interfaces between your JavaScript code and client-side Android code. For example, your JavaScript code can call a method in your Android code to display a `Dialog`, instead of using JavaScript's `alert()` function.

To bind a new interface between your JavaScript and Android code, call `addJavascriptInterface()`, passing it a class instance to bind to your JavaScript and an interface name that your JavaScript can call to access the class.

For example, you can include the following class in your Android app:

Kotlin | Java

```
public class WebAppInterface {
    Context mContext;

    /** Instantiate the interface and set the context */
    WebAppInterface(Context c) {
        mContext = c;
    }

    /** Show a toast from the web page */
    @JavascriptInterface
    public void showToast(String toast) {
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}
```

This creates an interface called `Android` for JavaScript running in the `WebView`. At this point, your web application has access to the `WebAppInterface` class. For example, here's some HTML and JavaScript that creates a toast message using the new interface when the user clicks a button:

```
<input type="button" value="Say hello" onClick="showAndroidToast('Hello Android!')" />

<script type="text/javascript">
    function showAndroidToast(toast) {
        Android.showToast(toast);
    }
</script>
```

Executing JavaScript Bridges

- Executing the JavaScript Bridge would require the WebView to simply browse to the website with the example JavaScript
- In the example, the result would be a Toast message saying the words “Hello Android!”
- This Toast message function seems safe, but what would happen if the executed Java function did something more abusive?

Android Developer documentation on JavaScript Bridges

Bind JavaScript code to Android code

When developing a web application that's designed specifically for the `WebView` in your Android app, you can create interfaces between your JavaScript code and client-side Android code. For example, your JavaScript code can call a method in your Android code to display a `Dialog`, instead of using JavaScript's `alert()` function.

To bind a new interface between your JavaScript and Android code, call `addJavascriptInterface()`, passing it a class instance to bind to your JavaScript and an interface name that your JavaScript can call to access the class.

For example, you can include the following class in your Android app:

Kotlin Java

```
public class WebAppInterface {
    Context mContext;

    /** Instantiate the interface and set the context */
    WebAppInterface(Context c) {
        mContext = c;
    }

    /** Show a toast from the web page */
    @JavascriptInterface
    public void showToast(String toast) {
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}
```

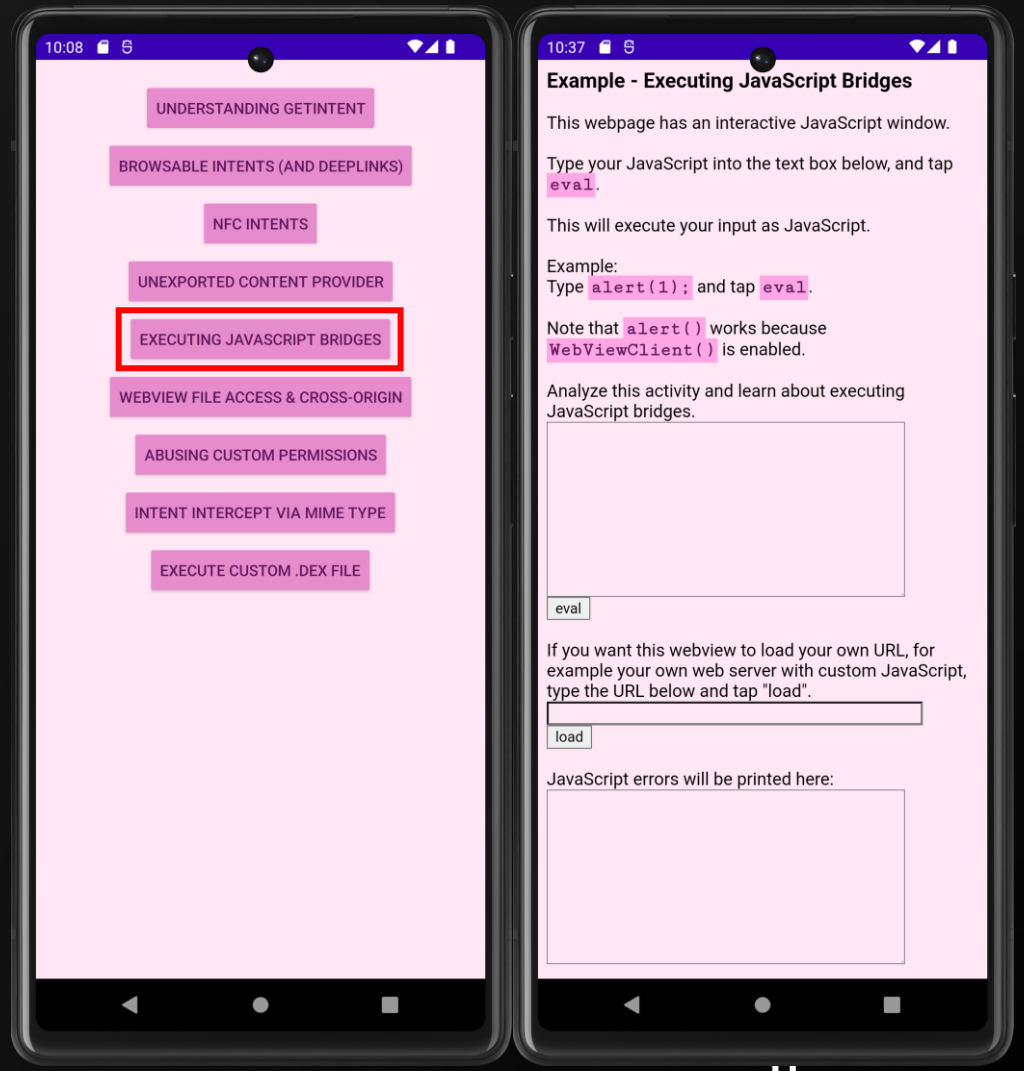
This creates an interface called `Android` for JavaScript running in the `WebView`. At this point, your web application has access to the `WebAppInterface` class. For example, here's some HTML and JavaScript that creates a toast message using the new interface when the user clicks a button:

```
<input type="button" value="Say hello" onClick="showAndroidToast('Hello Android!')" />

<script type="text/javascript">
    function showAndroidToast(toast) {
        Android.showToast(toast);
    }
</script>
```

Executing JavaScript Bridges - Example

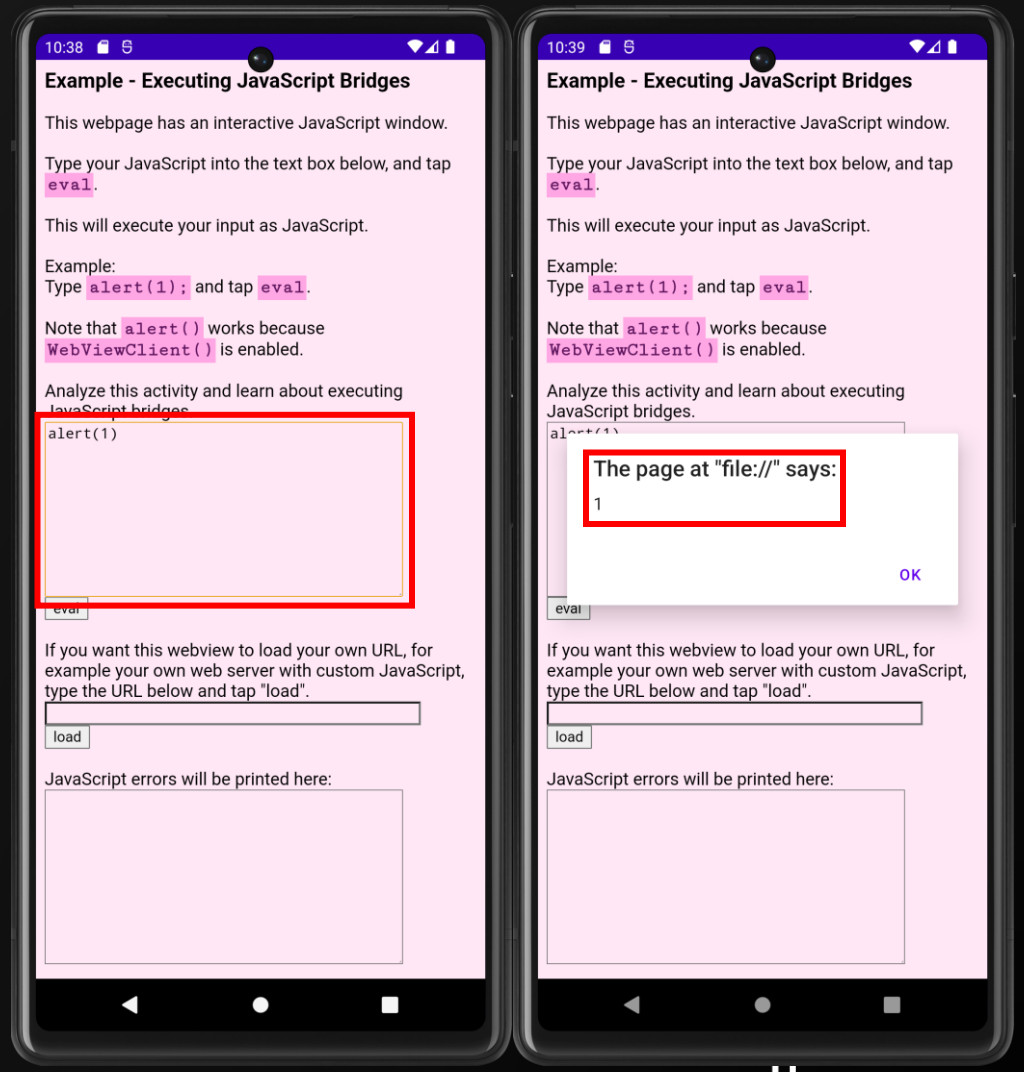
- We will now use Axolotl to better demonstrate how to use JavaScript Bridges
- On Axolotl's main menu, tap:
  - "Exercise Modules"
  - "Executing JavaScript Bridges"
- An Activity with a WebView will launch
  - The launched activity is programmed via the Java class `com.maliciouserection.axolotl.example.activity.webview.javascriptInterface`



MALICIOUS ERECTION LLC

Executing JavaScript Bridges - Example

- The `javascriptInterface` Activity contains a WebView that loads `_example_index_1.html` from the application's Assets folder
  - In a real world scenario, the WebView would most likely load an externally facing website
- The middle of the HTML page is an input window with an `eval` button; whatever you input in this area will be executed as JavaScript
  - This is to simulate a real world scenario where an attacker can execute JavaScript on a target webpage, such as through a XSS vulnerability
- We will be analyzing the Activity `javascriptInterface` to play with the available JavaScript Bridge



MALICIOUS ERECTION LLC

Executing JavaScript Bridges - Example

- After decompiling `javascriptInterface`, we see that a WebView is created in the method `theMainMethod()`
- At the end of `theMainMethod()`, there is code which adds a JavaScript Interface (Bridge) to the WebView
  - The class `yayjsinterfaceyay` is added to the WebView as a JavaScript object called `yayjsinterfaceyay`
  - `yayjsinterfaceyay` is also a nested class within the `javascriptInterface` class itself

Decompiled `javascriptInterface` class

```
@Override // android.app.Activity
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_webview);
    theMainMethod();
}

private void theMainMethod() {
    WebView webView = (WebView) findViewById(R.id.webview);
    this.theWebView = webView;
    WebSettings settings = webView.getSettings();
    this.theWebSettings = settings;
    settings.setJavaScriptEnabled(true);
    this.theWebView.setWebViewClient(new WebViewClient() { // from class: com.maliciousere
    });
    this.theWebView.setWebChromeClient(new WebChromeClient() { // from class: com.maliciou
    });
    this.theWebView.clearCache(true);
    this.theWebView.clearHistory();
    this.theWebView.addJavascriptInterface(new yayjsinterfaceyay(), "yayjsinterfaceyay");
    this.theWebView.loadUrl(this.yayurlyay);
}
```

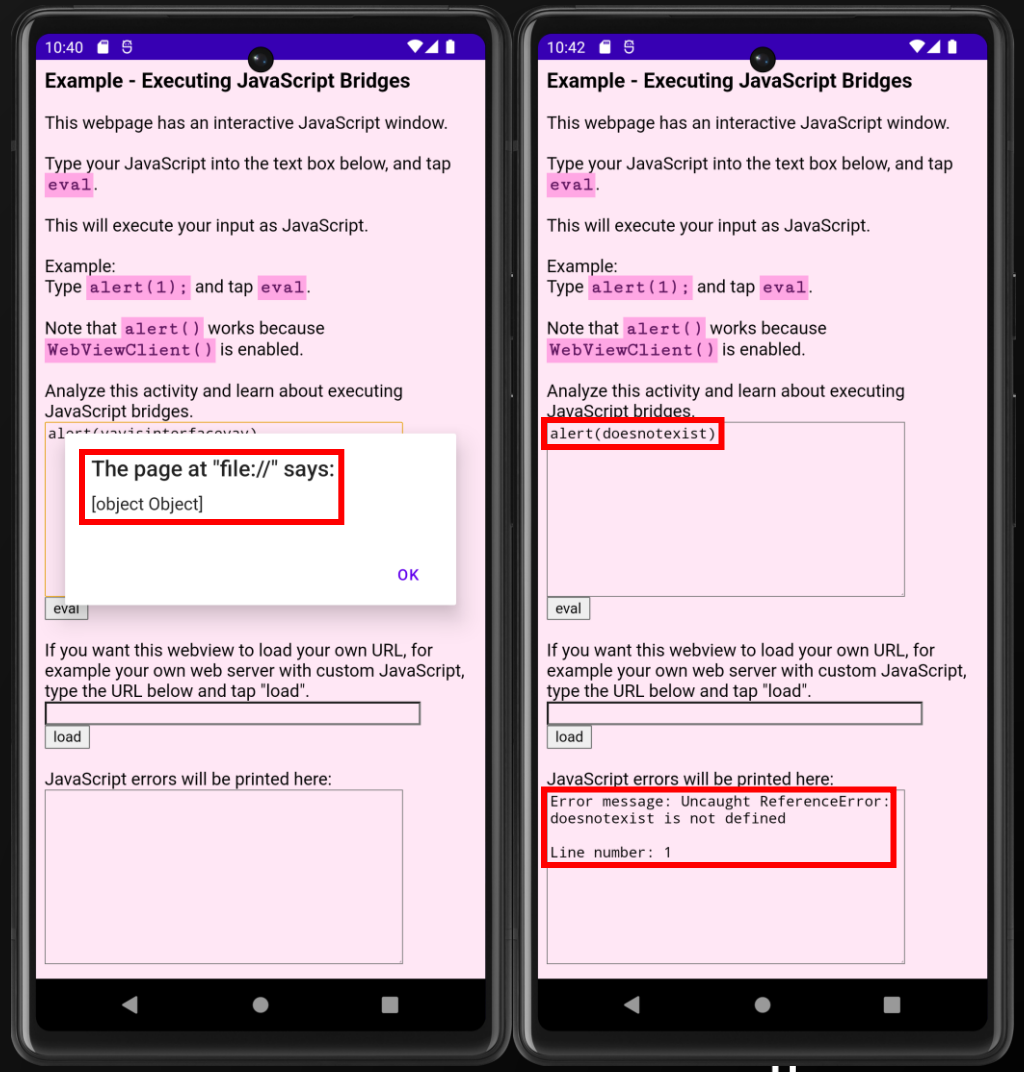
```
public final class yayjsinterfaceyay {
    public yayjsinterfaceyay() {
    }

    @JavascriptInterface
    public void showToastFunction(String yaytoastyay) {
        Toast.makeText(javascriptInterface.this.getApplicationContext(), yaytoastyay, 1).show();
    }
}
```

MALICIOUS ERECTION LLC

Executing JavaScript Bridges - Example

- The Java code in the Activity `javascriptBridge` revealed that a JavaScript Interface can be executed via the JavaScript Object `yayjsinterfaceyay`
- To test this, in the WebView, we can execute an alert box with the JavaScript Object as the content
  - `alert(yayjsinterfaceyay)`
  - Notice that there are no single or double quotes in the above code
- If the JavaScript Object exists, then the alert box will be populated with `[object Object]`
- Otherwise, the alert box won't appear
  - The JavaScript error also gets printed on the screen for your convenience



MALICIOUS ERECTION LLC

Executing JavaScript Bridges - Example

- Going back to `yayjsinterfaceyay`, there is a method `showToastFunction(String)`
- This method pops up an Android Toast message based on the String variable `yaytoastyay`
- Since this method has the `@JavascriptInterface` tag, and this method is within `yayjsinterfaceyay`, this method is callable from the WebView

The JavaScript Interface `yayjsinterfaceyay`

```
public final class yayjsinterfaceyay {
    public yayjsinterfaceyay() {
    }

    @JavascriptInterface
    public void showToastFunction(String yaytoastyay) {
        Toast.makeText(javascriptInterface.this.getApplicationContext(), yaytoastyay, 1).show();
    }
}
```

MALICIOUS ERECTION LLC

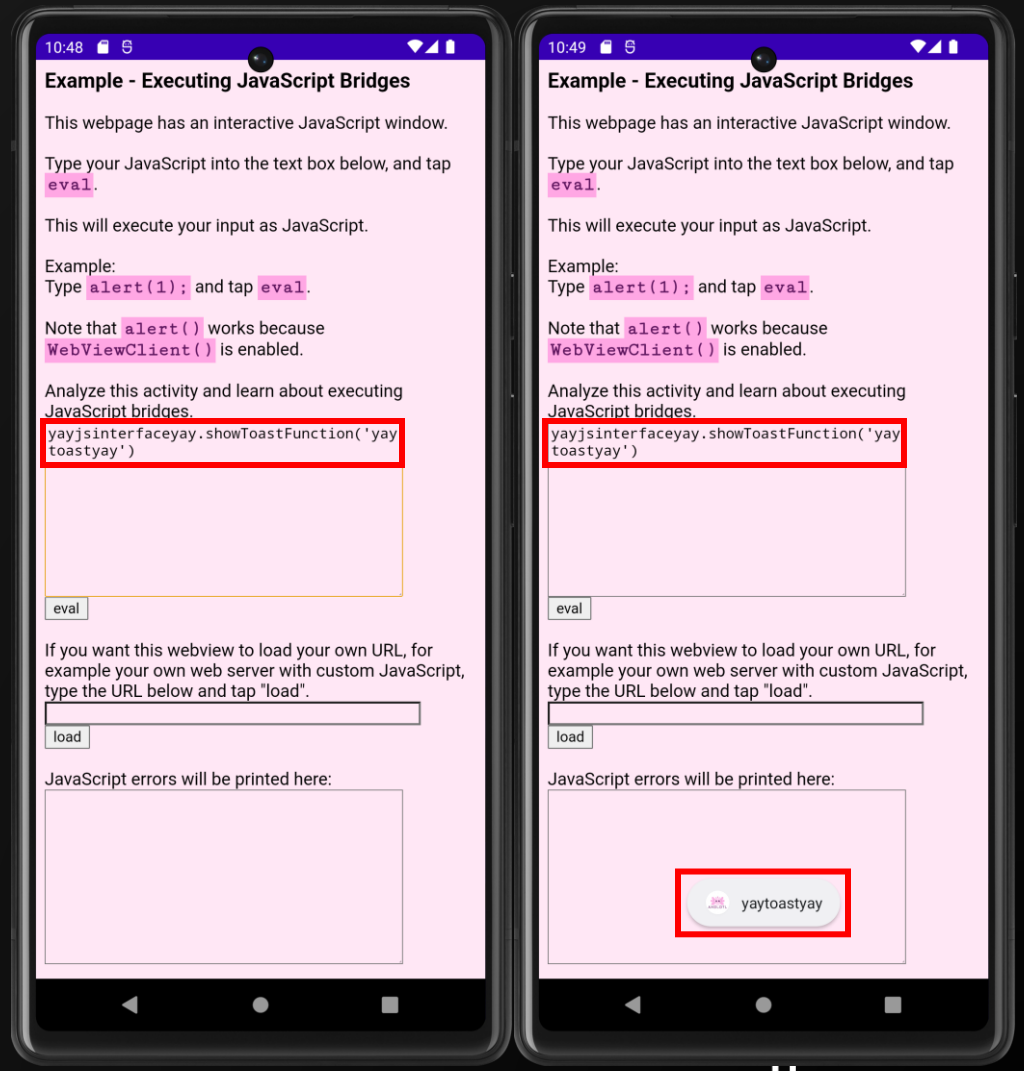
Executing JavaScript Bridges - Example

- We can confirm this by calling the JavaScript Object `yayjsinterfaceyay` and then calling the method `showToastFunction(String)`
  - Make sure to replace `String` with whatever you want to appear in the Toast message
  - Example: `yayjsinterfaceyay.showToastFunction('yaytoastyay')`
- Once the Toast message appears, you have successfully executed a JavaScript Bridge function

The JavaScript Interface `yayjsinterfaceyay`

```
public final class yayjsinterfaceyay {
    public yayjsinterfaceyay() {
    }

    @JavascriptInterface
    public void showToastFunction(String yaytoastyay) {
        Toast.makeText(javascriptInterface.this.getApplicationContext(), yaytoastyay, 1).show();
    }
}
```



MALICIOUS ERECTION LLC

Module 5 Exercise

- The WebView `com.maliciouserection.axolotl.activity.webview_activity` contains a Debug Mode. Use what you have learned so far to enable Debug Mode
  - This WebView can be accessed from the `MainActivity` by tapping “More Axolotls"
- Capture The Flag - With Debug Mode enabled, exfiltrate Flag 5
  - BONUS: Take this a step further by redirecting `webview_activity` to your own custom web server
  - Using your custom web server, execute JavaScript that will automatically exfiltrate Flag 5 for you without ever touching the device
    - So imagine a scenario where Example Exploit launches and then Flag 5 is exfiltrated without touching the device

MALICIOUS ERECTION LLC