A First Look at **Firefox OS Security**

Daniel DeFreez*, Bhargava Shastry†
Hao Chen*, Jean-Pierre Seifert†

* University of California, Davis
† Security in Telecommunications, TU Berlin
Why look at Firefox OS?

- Privileged web apps
  - Content Security Policy
  - Application review
- Desktop → mobile
  - Security UI
Background

- No native app code!
- Merges web and mobile security models

Gaia → Web Applications (HTML+JavaScript+CSS)
Gecko → Javascript engine / Permission Manager
Gonk → Underlying OS / Kernel (Android based)
Firefox OS Apps

- Three app types
  - Unprivileged
  - Certified
  - Privileged
Analyzing Apps

- Practical hurdles for dynamic analysis
- Precise JS analysis is difficult
  - Especially at market scale
  - Performance already a pain point
- Can lightweight analysis find bugs?
  - Yes
  - Fast (15min for entire market)
Reviewer Guidelines

1) AJAX over TLS
2) Validate origin of message events
3) HTML element sinks (.innerHTML)
Problem: Privacy concerns

Approach: Constant propagation, Deobfuscation

Finding: 50% of apps that use systemXHR do not use TLS

Example: Messaging apps send phone numbers in the clear
window.addEventListener('message', function(evt) {
    if (evt.origin !== 'app://example') {
        return;
    }

    // All is right with the world
});
Origin Validation

**Problem:** Developers fail to validate origin of messages

**Approach:** Scope-aware AST pattern matching

- Precise analysis unnecessary

**Finding:** 83 apps have message handlers

56 fail to validate origin (67%)
Problem: `.innerHTML` interprets string as HTML
- Can't flag all uses (false positives)

Approach: Taint propagation, type inference
- Look for inconsistencies
HTML Injection

[1] HTML injection via shared note

[2] `<iframe>` opens attack site

[3] `postMessage` OAuth token

Injection, validation failures overlap → exploitability

High profile app (bug bounty)
Lessons

• Content Security Policy is effective
  – Could be tweaked to be more so

• Lightweight tools helpful
  – Requires stepping beyond AST patterns

• HTML Injection remains a major problem
TLS Certificate Overrides

[1] Client Hello

[2] Server Hello

Verify server certificate

Fail

TLS Handshake with overridden certificate
Certificate Override Caching

- Although override is `temporary`
  - Persists until **device reboot** on Firefox OS

- Differences between desktop browser and Firefox OS
  - UI and UX inconsistency
  - Architectural difference: Process model
  - Security principal
Inconsistent UI: Site ID Button

Broken padlock expected
Inadequate UI: Revoking Overrides

Firefox Desktop Browser

Firefox OS Browser

No recourse except device reboot
UX Inconsistency: Overrides Across Apps

Users expect that **side-effects don't percolate to other apps**
Process Model

Firefox Browser

Single Process

Web content

Overides cache

Gecko

Firefox OS

Content Processes

App 1

App 2

App N

Privileged Rendering Process

Gecko
nsCertOverrideEntry *entry = mSettingsTable.GetEntry(hostPort.get());
Threat Scenario

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cert</th>
</tr>
</thead>
<tbody>
<tr>
<td>facebook</td>
<td>0A:2F...</td>
</tr>
</tbody>
</table>

On device

Off device

MITM

Gecko

Facebook App

Browser App
Summary

Retrofitting web apps on legacy software can lead to subtle security issues
Thank You!

Questions?
Extras
UX Inconsistency: Overrides Across Apps
Inadequate UI: Site Security Info

Mobile browsers compensate by **warning** users on **each visit**
Vulnerable Traffic: Code

- Unauthenticated installation of unprivileged apps
  - 92% of hosted apps are served over http
  - Mozilla *recommends* https

- Proof-of-Concept
  - Inserted unintended functionality
  - Limitations
    - Restricted permission set for unprivileged apps
    - User prompt for sensitive permissions
Security Principal

What is the unit of isolation?

Same origin policy → facebook.com

What users expect...

<Facebook App, facebook.com>