Advanced Chrome Extension Exploitation

Leveraging API Powers for the Better Evil

Krzysztof Kotowicz
Kyle ‘Kos’ Osborn

Note:

This slide deck version is fairly draft material. Please check out the following website for the version that is presented:

http://kyleosborn.com/bh2012

Further updates to the white paper will be available there also.

Introductions

Krzysztof Kotowicz

- IT security consultant at SecuRing

Kyle Osborn

- Information Security Specialist at AppSec Consulting

Chrome Extension Security

- Common web vulnerabilities that effect higher privileged applications.

- Cross Site Scripting and Cross Site Request Forgery are the most common vulnerabilities in extensions.

Chrome Extension Security

- Currently, Chrome extension security is very reliant on the developer.

- Writing bad code is easy, giving extensions more permissions than necessary is easier.

Chrome Extension Security

- Most commonly vulnerable:
  - RSS Readers
  - Note Extensions
  - Web Developer extensions

Finger Printing

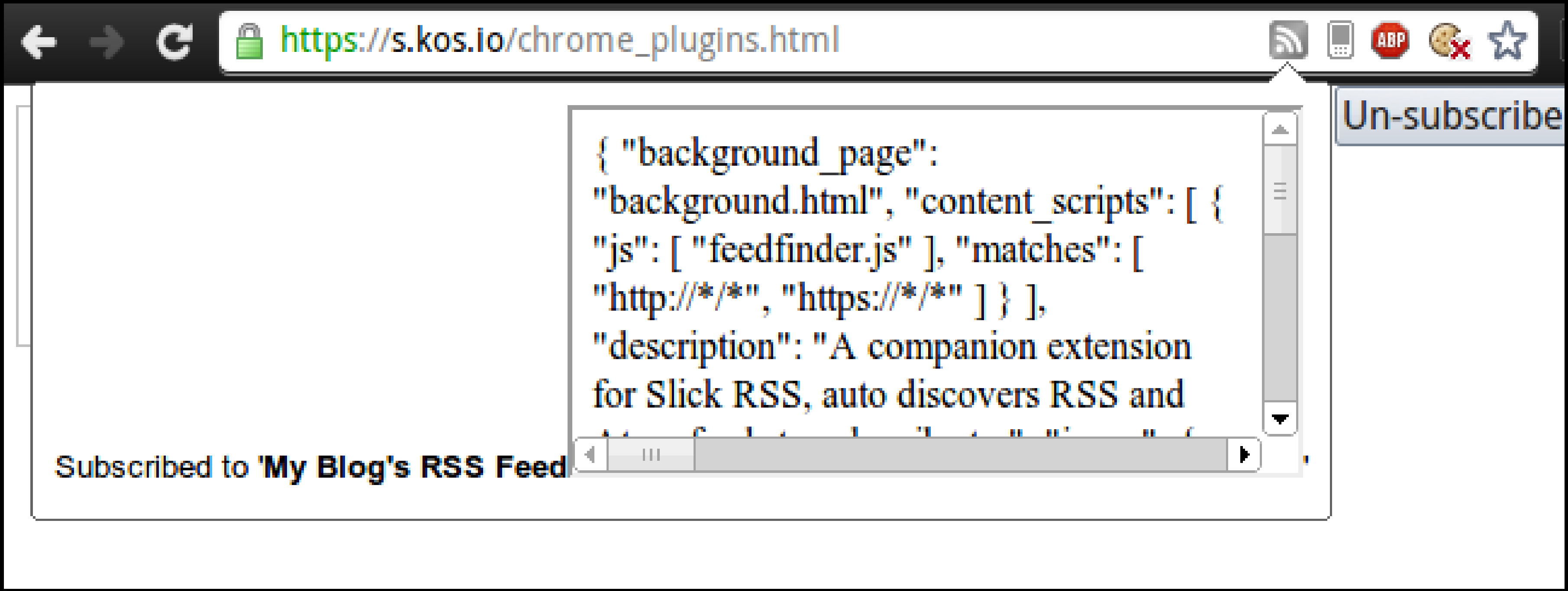
- The simplest method of fingerprinting was described by Krzysztof.
- http://blog.kotowicz.net/2012/02/intro-to-chrome-addons-hacking.html
- Chrome-extension: URIs aren't (currently) restricted from a website's DOM
- It is simple to generate a list of known extensionIDs, and bruteforce chrome-extension://ID/ resources to discovered extensions

Previous Research

- Kotowicz
  - http://blog.kotowicz.net/2012/02/intro-to-chrome-addons-hacking.html
- UC Berkeley – Extension security evaluation
  - http://www.eecs.berkeley.edu/~afelt/extensionvul
- Hacking Google ChromeOS (BH 2011)

Examples/Demos

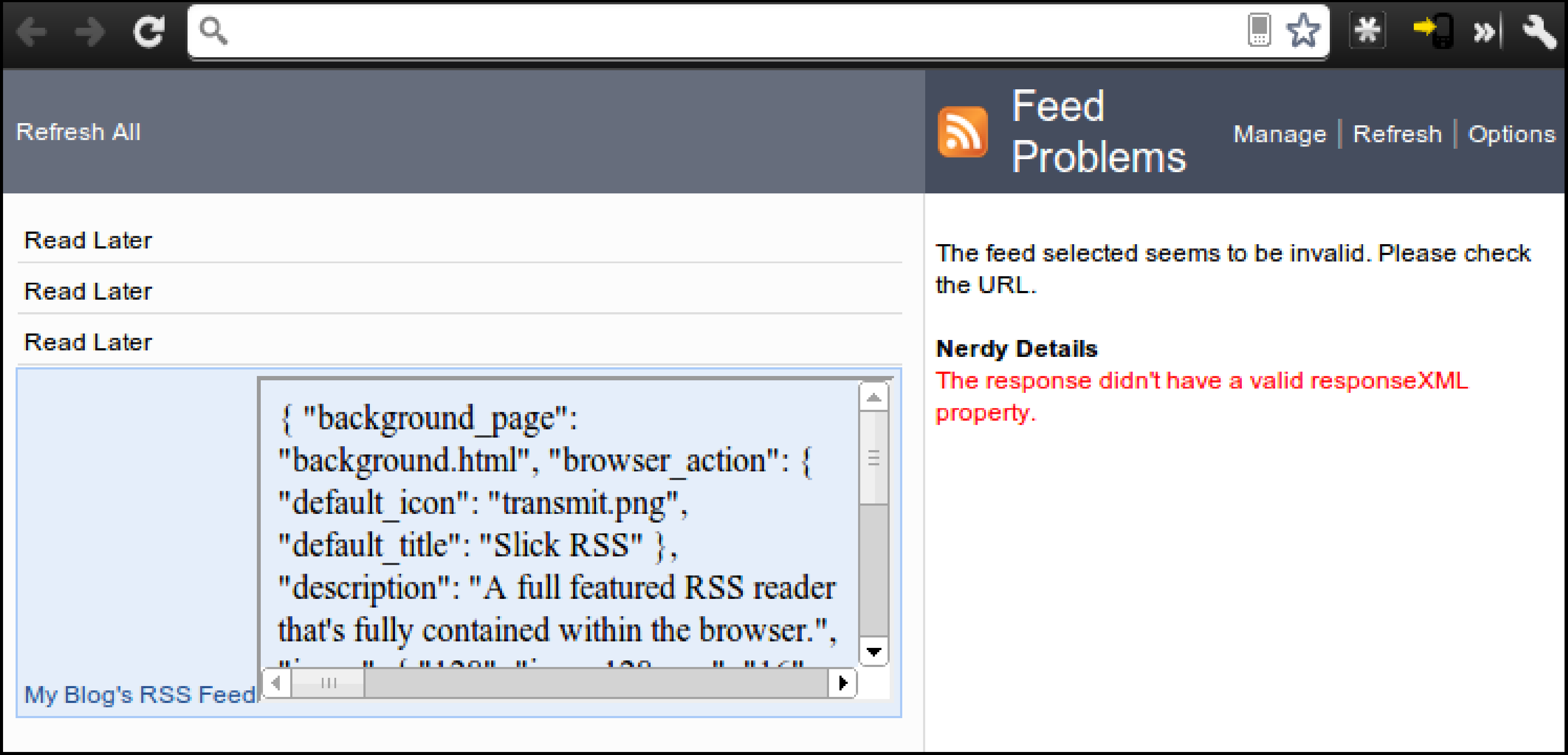
- Slick RSS & Slick RSS: Feed Finder
  - Simple injection location (<link> tag title)



```
{ "background_page":
"background.html", "content_scripts": [ {
"js": [ "feedfinder.js" ], "matches": [
"http://*/*", "https://*/*" ] } ],
"description": "A companion extension
for Slick RSS, auto discovers RSS and
```

Subscribed to 'My Blog's RSS Feed

Examples/Demos



Refresh All

Read Later

Read Later

Read Later

```
{ "background_page":
"background.html", "browser_action": {
"default_icon": "transmit.png",
"default_title": "Slick RSS" },
"description": "A full featured RSS reader
that's fully contained within the browser.",
```

My Blog's RSS Feed

Feed Problems

Manage | Refresh | Options

The feed selected seems to be invalid. Please check the URL.

Nerdy Details

The response didn't have a valid responseXML property.

Examples/Demos

More demos and examples to be released during presentation.

Automating Post-exploitation

- Found <script>alert(1)</script> - Now what?
- Use an automated tool to pillage and plunder
- The Browser Exploitation Framework (BeEF) does a great job hooking into DOMs
- But – Need a special tool designed to take advantage of Chrome Extension APIs.

Automating Post-exploitation

- Enter XSS ChEF (Chrome Extension Exploitation Framework)

  - Designed from the ground up as a chrome extension exploitation framework.
  - Fast (uses WebSockets)
  - Preloaded with automated attack scripts

Automating Post-exploitation

- Monitor open tabs of victims
- **Execute JS** on **every tab**
- Extract HTML
- **Read**/write **cookies**
- Access localStorage
- Manipulate browser history
- Take screenshots of tabs
- Inject **BeEF** hooks / **keyloggers**

End of Part One

Hopefully with the information provided, exploiting Chrome Extensions can prove to be a useful tactic in real life security assessments.

Part two: workshop!

The workshop portion will be released during the presentation.

CAESARS PALACE