Chrome Extensions: Threat Analysis and Countermeasures

Lei Liu, Xinwen Zhang*, Guanhua Yan*, and Songqing Chen

George Mason University
Huawei R&D Center
Los Alamos National Laboratory

NDSS'12

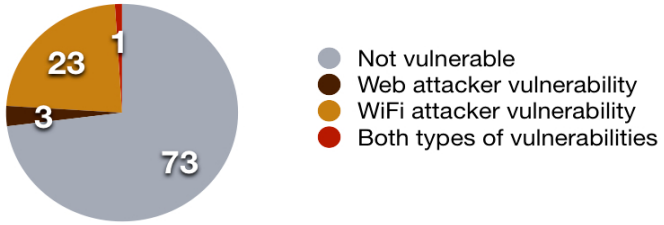
* Does not represent employer's opinion

Attacks via Extensions

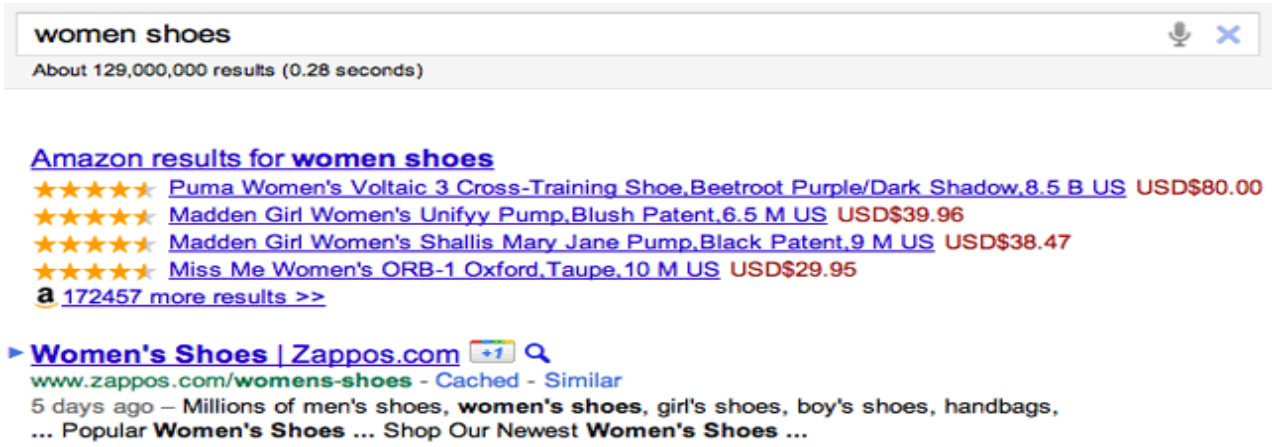
- Extension is the vehicle for increasing attacks
- BHO/add-on is the one of the techniques used by many spyware writers in IE.
  - Kida et al'05, CERT'05, Egele'07, Li'07, Guha'11
- Abusing of Firefox extensions has been widely recognized and studied in literature
  - Defcon'09, Ter-Louw'08, Dhawan'09, Bandhakav'10, Djeric'10, Guha'11

Attacks via Chrome Extensions

- Buggy Chrome extensions have been identified recently
  - 27 out of 100 leak data
    - http://www.adrienneporterfelt.com/blog/?p=226_



- Malicious extensions have appeared



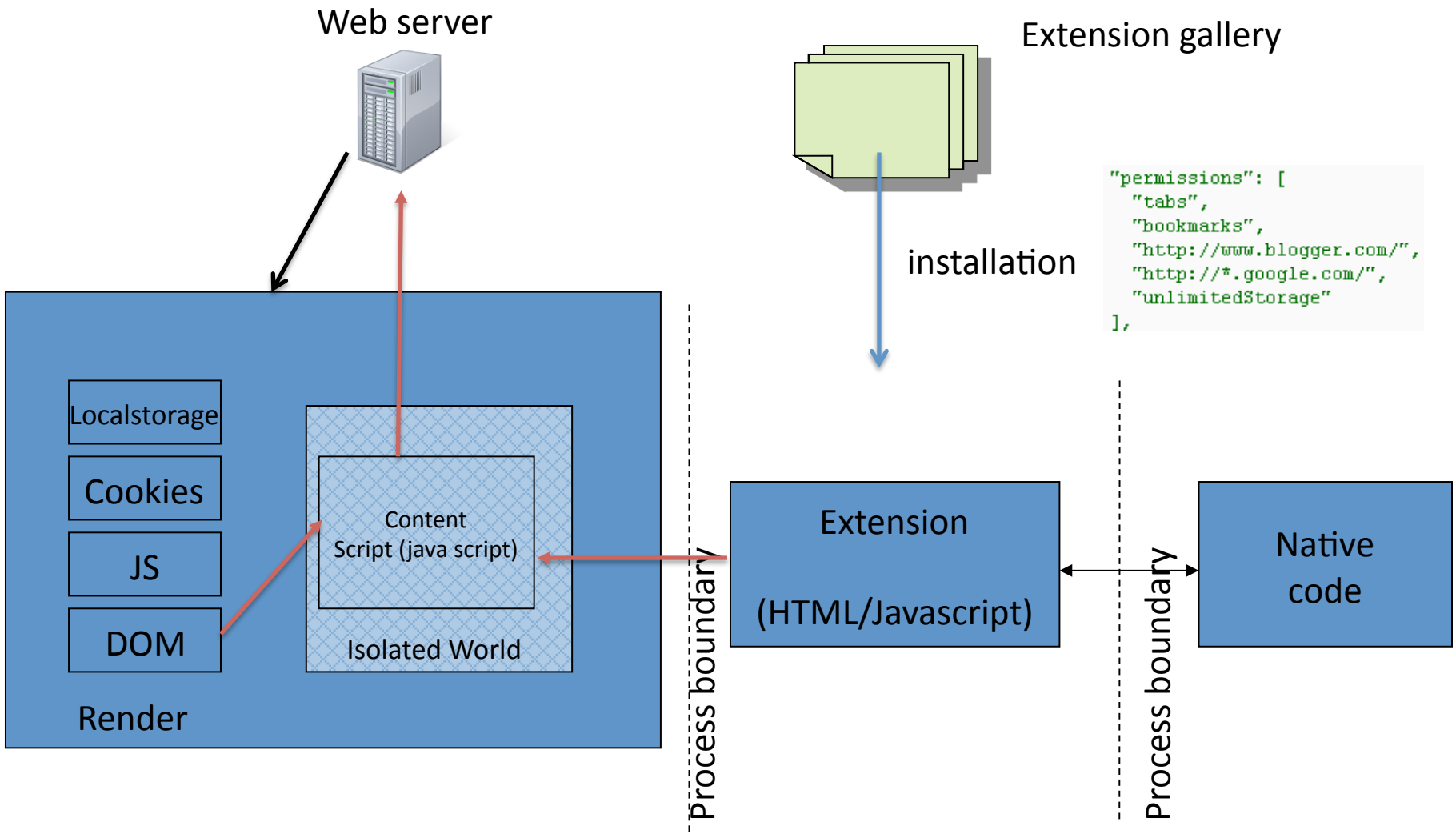
Problem Statement

- Why Chrome extension?
  - Chrome has built-in security model for browser architecture and extension
- Is current Chrome extension architecture good enough?
  - Particularly with the consideration of malicious extensions
- Easy to launch malicious extensions:
  - It is a difficult task to sanitize rapidly increasing extensions in Google Chrome Web Store with slow reviewing process
  - Users are free to download/install extensions from many (known/unknown) host servers
  - Strong incentive for attackers, e.g.,
    - harvest sensitive content in web pages
    - Modify web search content
- Google takes actions against malicious extension developers
  - singup fee for developers
  - Domain verification for developers
- Problem: Can we have a technical solution?
  - Or improvement of current permission model for better security with malware extension?

Contributions

- We demonstrate several attacks with malicious Chrome extensions through experimental implementation
- We do security analysis of the permission model of Chrome extension
  - With the assumption of malicious extensions
- We propose security enhanced extension permission model and enforcement mechanism
  - Following the principles of least privilege and separation of privilege in more strict way

Chrome Extension Architecture



Chrome Extension Security Model

- Least Privilege
  - Pre-defined permission set (e.g.,. To access web sites, browser tab, bookmarks, history, …)
  - Each extension declares permissions required
  - User authorizes permissions at installation time

- Privilege Separation
  - Different permissions for different components of extension
  - Content script can interact with web content, not browser modules
  - Extension core has more privileges, but insulated from web pages

- Strong isolation
  - Same origin policy
    - Each extension has unique origin
    - Accessing other origins requires cross-site permissions
    - Inject content script requires cross-site permissions
  - Process-level isolation: extension core runs in separated process from renderer and browser
  - Within a renderer process, content script runs in *isolated world* from Javascript of web page

Chrome Extension Trust Model

- The main trust model of Chrome extension assumes trusted but buggy extensions
- But malicious web pages
- Therefore the security objectives are mainly for restricting web pages to access browser resources via extensions
- And confine the damage propagation if possible

Experimental Attacks

- We develop a malicious extension as a bot
  - from Chrome 7 to the latest
  - does email spamming, DDoS, and phishing attacks easily
    - Through attacking web pages
  - Receive commands from bot master with built-in update mechanism of Chrome extension
    - No security check for update

Email Spamming

Upload update

update site spam

spam

Webmail server

POST spam

spam.txt - Notepad

File Edit Format View Help

```
URL = mail.***.**/comp_fs.html?rfpossible-true&lang-en
editor=TextArea
name=text
Text=Please visit this exciting site www.xxx.com
```

Browser

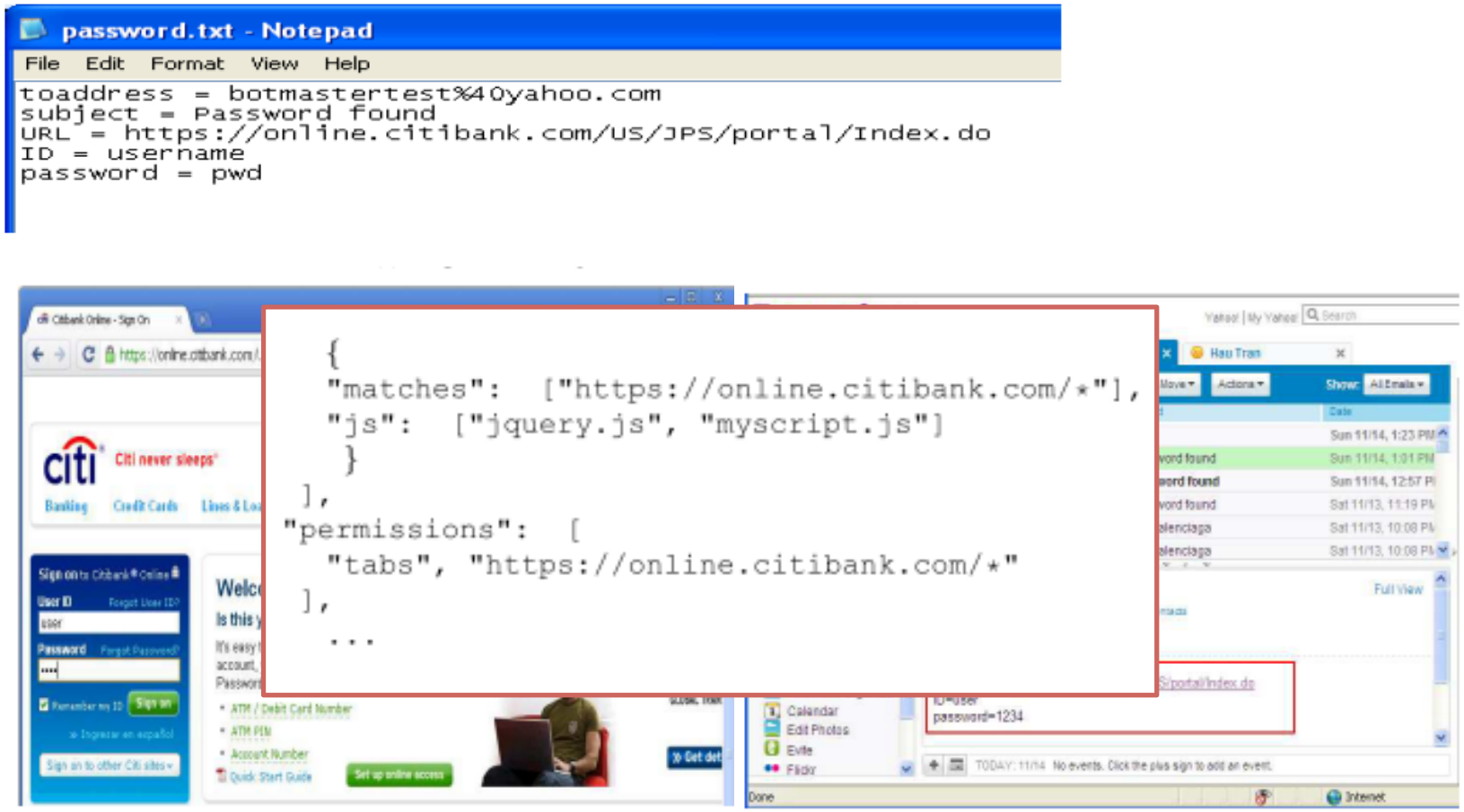
download update

Extension spam

manipulate

```
"permissions":  [
   "tabs", "http://*/*", "https://*/*"
]
```

Password Sniffing



DDoS Attack

DDoS.txt - Notepad

File Edit Format View Help

```
URL = http://www.google
Date = Apr 01, 2011
Time = 12:35
Interval = 1
duration = 1000
```

```
"permissions":  [
   "tabs", "http://*.yahoo.com/*"
]
```

<capture> - Ethereal

File Edit Capture Display Tools Help

No.	Time	u	Destination	Protocol	Info
22	1.183777	z	www.google.com	HTTP	GET /images/nav_logo65.png
23	1.231624	z	www.google.com	TCP	62458 > http [ACK] Seq=875274258 Ack=387691822
24	1.233798	z	www.google.com	TCP	62458 > http [ACK] Seq=875274258 Ack=387692075
25	1.234879	z	www.google.com	TCP	62458 > http [ACK] Seq=875274258 Ack=387692342
26	1.236211	z	www.google.com	TCP	62458 > http [ACK] Seq=875274258 Ack=387692628
27	1.238374	z	www.google.com	TCP	62458 > http [ACK] Seq=875274258 Ack=387692894
28	1.251596	z	www.google.com	TCP	62458 > http [ACK] Seq=875274258 Ack=387693180
29	1.257766	z	www.google.com	TCP	62458 > http [ACK] Seq=875274258 Ack=387693466
30	1.259492	z	www.google.com	TCP	62458 > http [ACK] Seq=875274258 Ack=387693752
31	1.264934	z	www.google.com	TCP	62458 > http [ACK] Seq=875274258 Ack=387694038
					75274258 Ack=387694324
					75274258 Ack=387694610
					75274258 Ack=387694896
					75274258 Ack=387695182
					75274258 Ack=387695468
					75274258 Ack=387695494
					Seq=875274258 Ack=3876
					620521597 Ack=0 Win=81
40	2.338279	z	www.google.com	TCP	62459 > http [ACK] Seq=3620521598 Ack=39074693
41	2.338925	z	www.google.com	HTTP	GET /images/nav_logo65.png

```
⊞ Frame 22 (84 on wire, 84 captured)
⊞ Ethernet II
⊞ Internet Protocol
⊞ Transmission Control Protocol, Src Port: 62458 (62458), Dst Port: http (80), Seq: 875274
```

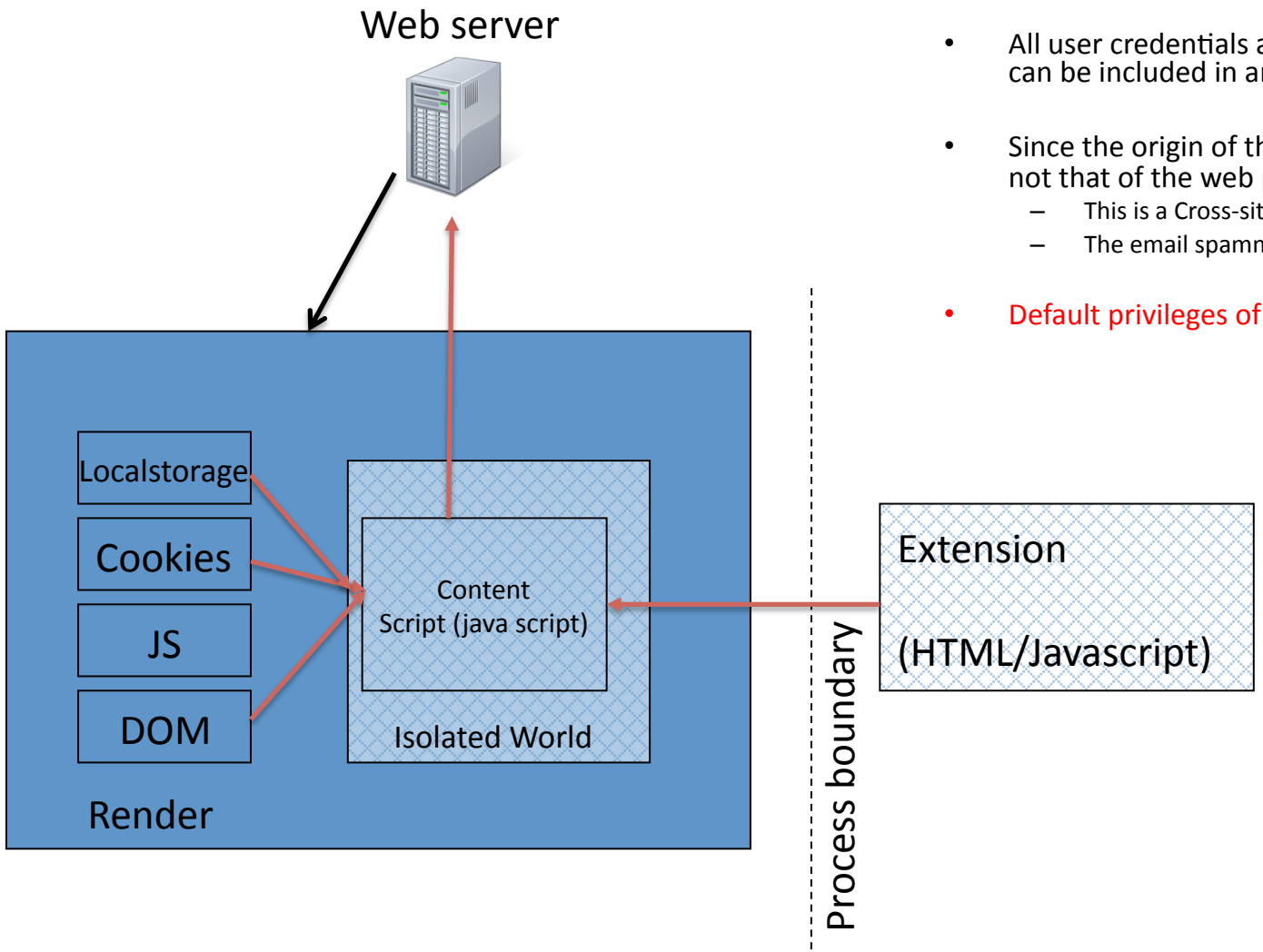
```
0000  00 23 54 f6 7a ee 00 24  d6 07 cf fe 08 00 45 00   .#T.z..$ ......E.
0010  00 46 77 8c 40 00 80 06  96 a7 c0 a8 01 08 4a 7d   .Fw.@... ......J}
0020  e0 50 f3 fa 00 50 34 2b  9f f4 e7 15 08 a3 50 18   .P...P4+ ......P.
```

Security Analysis

- Trust Model:
  - We assume browser kernel and pulgins are trustworthy
  - Sandbox mechanism provided by OS works well
  - Native code for extensions is sandboxed
  - Web apps are trusted

- Threat model: malicious extensions
  - Extension core
  - Content scripts

Cross-site Forgery with Content Script

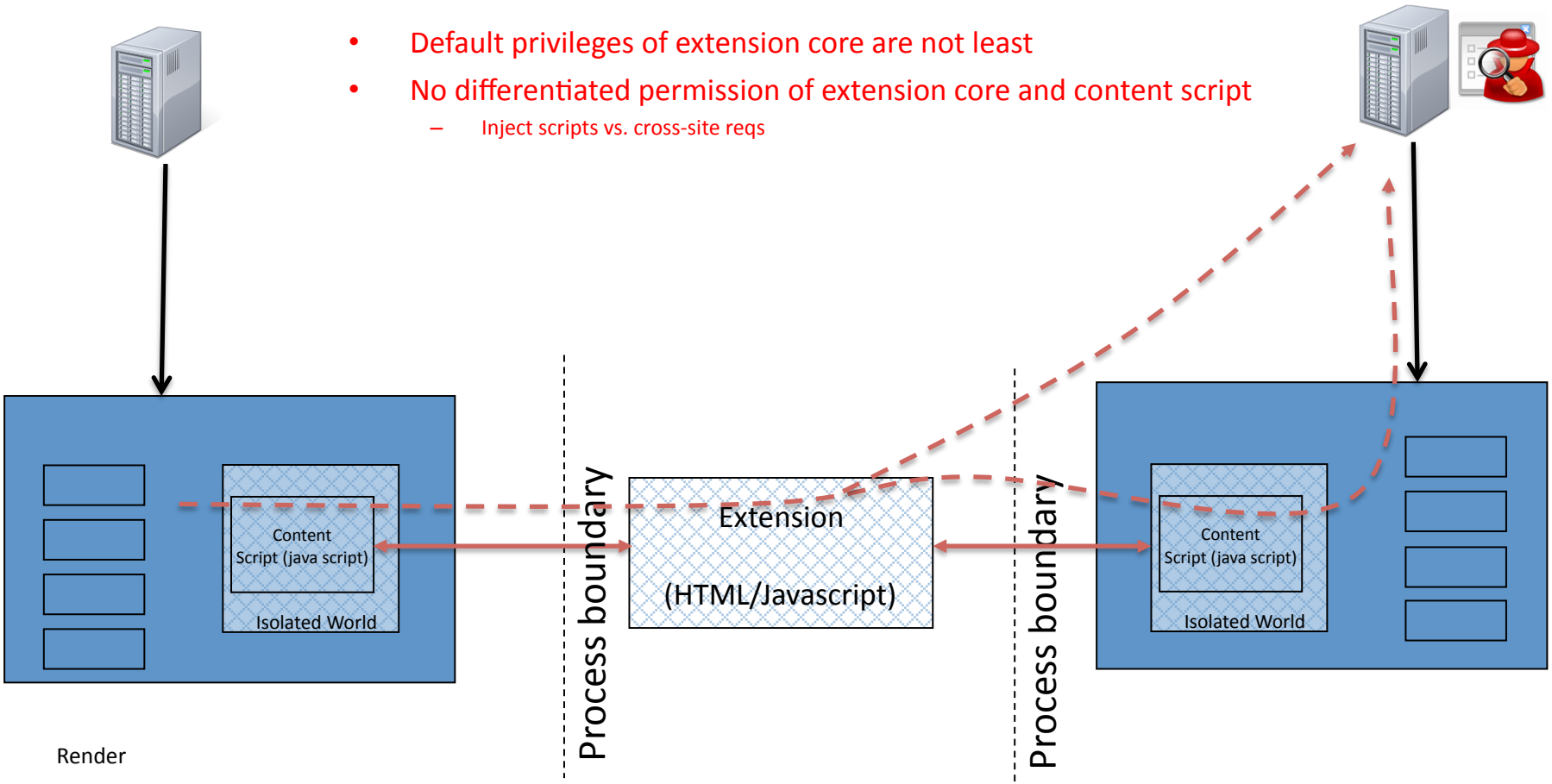


- A content script injected into web page can arbitrary access the origin of the page
- All user credentials associated with the origin can be included in an HTTP req
- Since the origin of the content script is usually not that of the web page
  - This is a Cross-site Forgery Req
  - The email spamming attack leverages this
- Default privileges of content script are not least

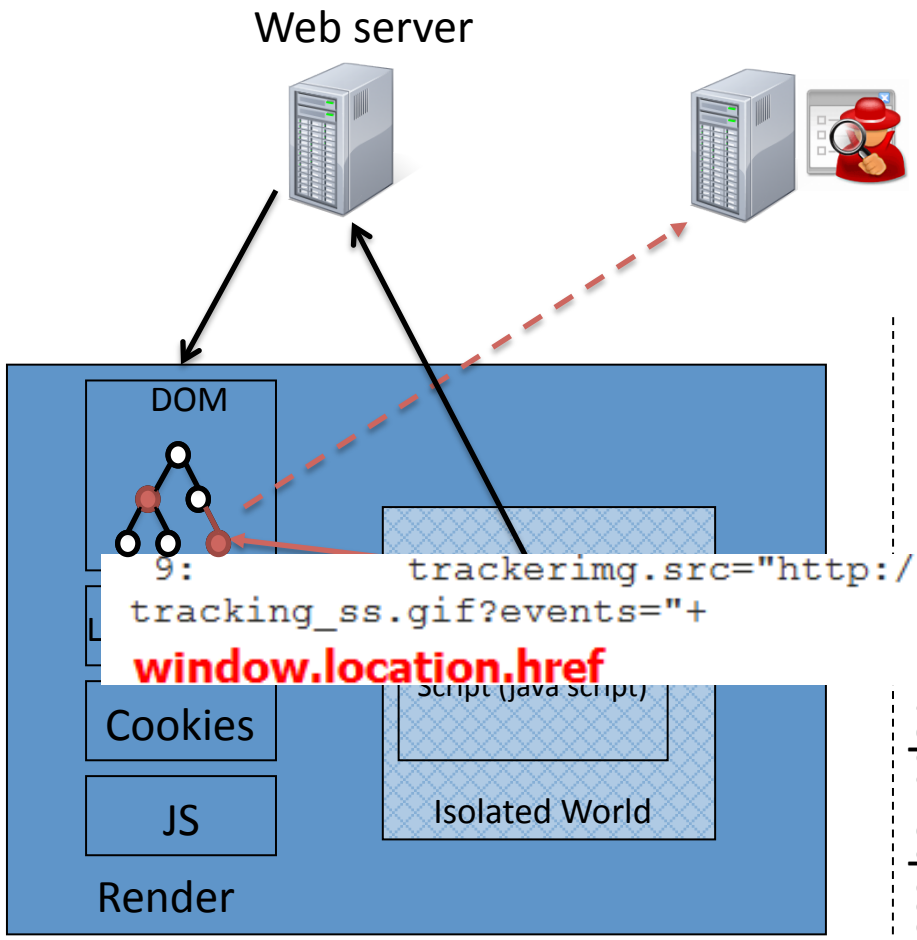
Cross-site Requests with Extension Core

- Cross-site reqs via content scripts through extension core
- The extension core can file cross-site HTTP reqs to multiple origins
  - Cross-site permissions are authorized in order to inject content scripts.

- Default privileges of extension core are not least
- No differentiated permission of extension core and content script
  - Inject scripts vs. cross-site reqs



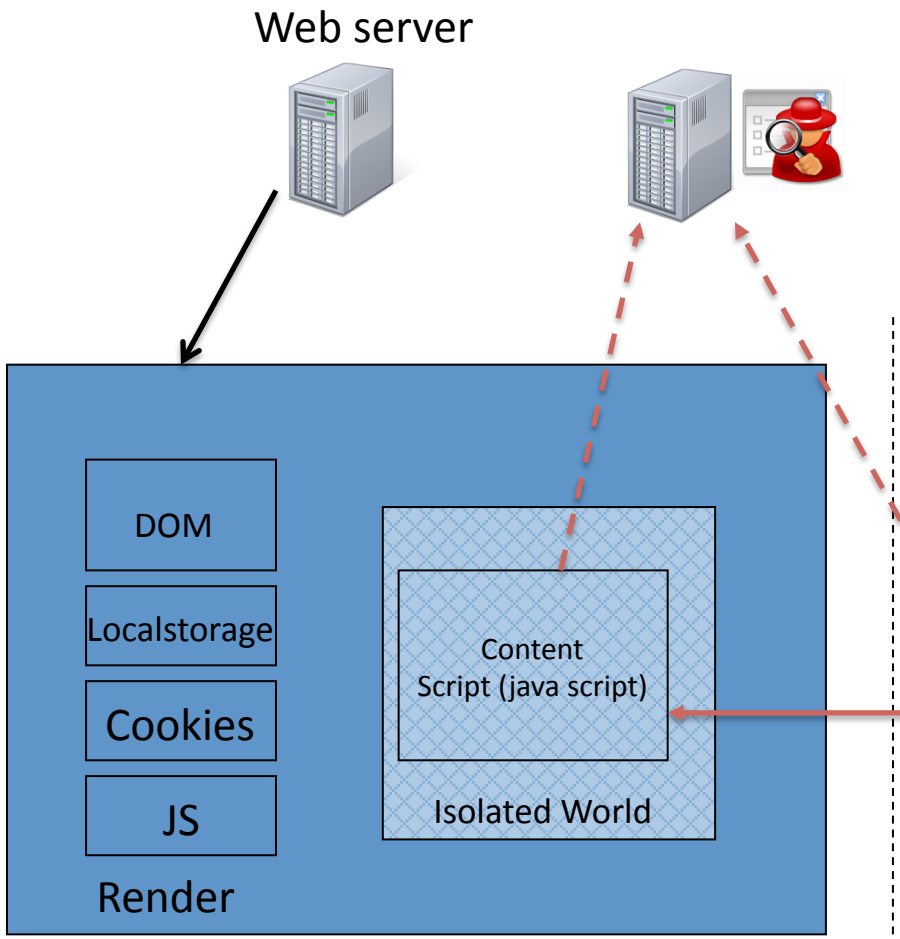
Cross-site Requests with Content Scripts




```
9:          trackerimg.src="http://www.smoothgesturesapp.com/tracking/
tracking_ss.gif?events="+
window.location.href
```

- Without cross-site permission, a running content script can only make HTTP reqs to the origin of the tab page
- However, since content script has full privileges of DOM, it can file unlimited cross-site HTTP reqs to arbitrary origin, e.g.,
  - Insert iframe
  - Load img
  - Modify src of DOM objects
  - With user credentials included in the req
- Loading of new DOM objects results in cross-site reqs
- Privilege to access DOM is not least for content script

Undifferentiated Permissions



- An extension may inject content script to many origins
  - It does not need to file HTTP reqs to all origins
  - But only to a dedicated one, e.g., a translation web service
- When an origin is assigned to extension, all components get the full privileges
  - Extension core can file cross-site reqs freely
  - Content script can arbitrarily modify the DOM, and file cross-sit reqs
- Privilege separation is not fine-grained enough

Process boundary

Extension
(HTML/Javascript)

permissions": ["http://*/*"]

19 out of 30 most popular extensions have this type of over privileges

Security Enhanced Chrome Extensions

- Micro-privilege management
- Differentiate DOM elements with sensitivity

Micro-privilege Management

- More fine-grained permission definition and enforcement
- Fine-grained permission differentiation for extension core and content script
  - Permission specs are separated from different components
- Least default privileges
  - Content script cannot introduce new origin to DOM
  - no HTTP req to tab origin

	Permissions	Example Permission Spec
Extension core	`inject_script`	`"http://*/*", "https://*/*"`
	`cross_site`	`"http://www.translate.com"`
Content script	`sensitivity_level`	`"medium"`
	`same_origin_request`	`"false"`
	`new_origin`	`"http://www.translate.com"`

Example Permission Spec

- Permissions for a translation extension:

```
"extension_core_permissions": [
   "inject_script":[
       "http://*/*", "https://*/*"
    ]
   "cross_site":[
       "tabs", "http://www.translate.com"
    ]
 ]
"content_script_permissions": [
   "sensitivity_level":[medium]
   "same_origin_request":[false]
   "new_origin":[
       "http://www.translate.com"
    ]
 ]
```

Differentiating DOM Elements

- To further reduce possible sensitive data leakage by content script, DOM elements can be differentiated with sensitivity levels

- A web app developer can identify sensitive information in a web page, e.g.,
  - High level data: only can flow to web origin
  - Medium level: may flow to authorized origins
  - Low level (default): can flow to any origin

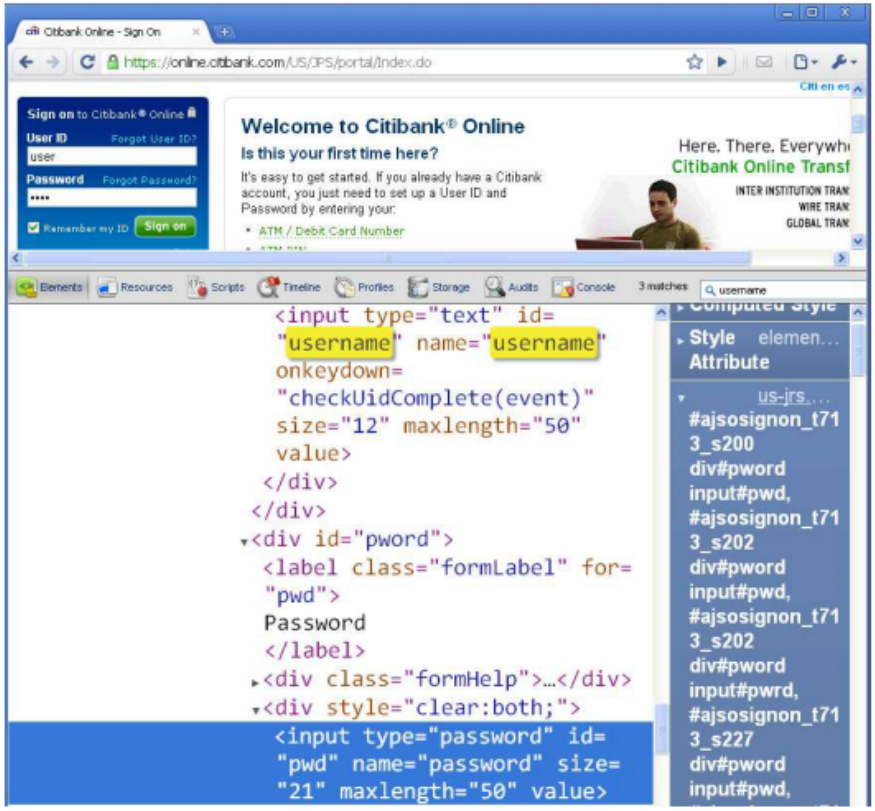
- An extension developer can specify permissions accordingly:
  - E.g., HIGH for username/pw, MEDIUM for other user info

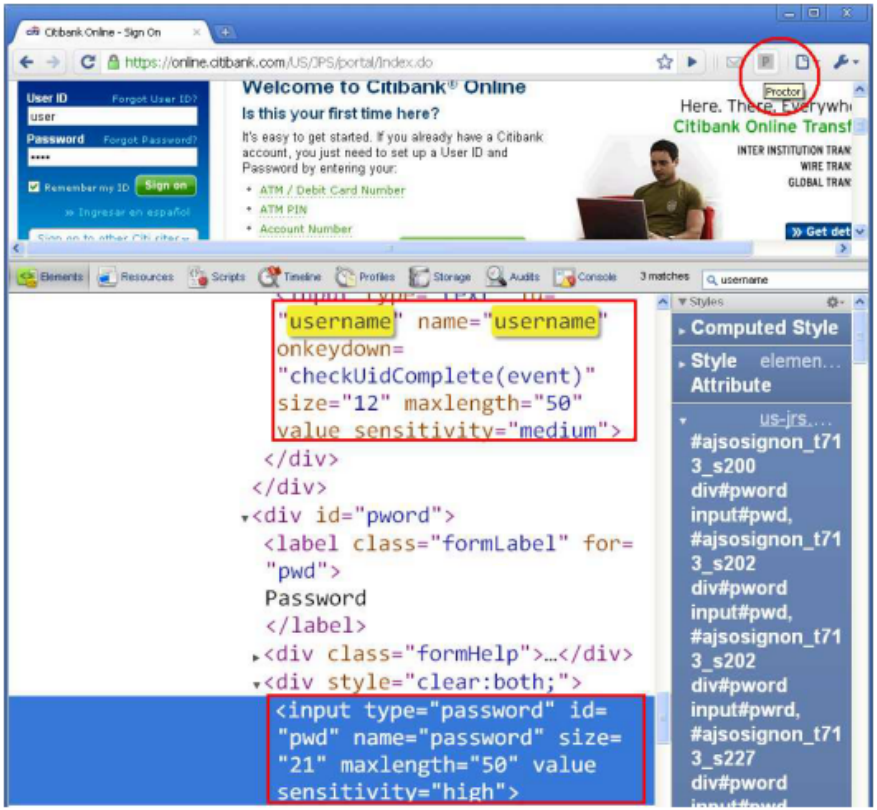
Implementation

- We have implemented the micro-privilege management and spec.

- For DOM sensitivity, we develop a helper extension (trusted):
  - To identify and label sensitive DOM elements
  - Re-write DOM element properties
    - According to configurable dictionary
  - Chrome enforces permission check based on extension manifest
  - Explicitly mark sensitive info by web app developer is not practical right now

Evaluations





Evaluation

- We selected 30 most popular extensions from Google extension gallery
  - 24 of them have granted network access
  - 19 of them request higher privileges than necessary (http://*/*)
- Our implementation easily changes their spec to reduce privileges

TABLE II
SUMMARY OF TOP 30 CHROME EXTENSION PRIVILEGE ANALYSIS

rank	name	over-privileged?	rank	name	over-privileged?
1	AdBlock	✓	16	RSS Subscription Extension	✓
2	Google Mail Checker	✗	17	Clip to Evernote	✓
3	FastestChrome	✓	18	Google Chrome to Phone Extension	✓
4	IE Tab	✗	19	Webpage Screenshot	✓
5	Browser Button for AdBlock	✓	20	Xmarks Bookmark Sync	✓
6	DocsPDF/PowerPoint Viewer	✗	21	SmileyCentral	✗
7	Downloads	✗	22	SocialPlus!	✗
8	Google Translate	✓	23	Facebook for Google Chrome	✗
9	Facebook Photo Zoom	✗	24	Speed Dial	✓
10	Google Dictionary	✓	25	Google Voice	✗
11	Turn Off the Lights	✓	26	Cooliris	✓
12	Firebug Lite	✓	27	FlashBlock	✓
13	Download Master	✓	28	Smooth Gestures	✗
14	Google Mail Checker Plus	✓	29	Awesome Screenshot	✓
15	Adblock Plus	✓	30	WOT	✗

Evaluation

- Our implementation blocks all experimental attacks on the bot extension.

TABLE III
RE-EVALUATION OF BOT ATTACKS

attack	result	reason
spamming	✗	unauthorized cross-site requests are completely blocked
DDoS	✗	unauthorized cross-site requests are completely blocked
password sniffing	✗	Proctor forbids sensitive information access
cross-site forgery	✗	content scripts are not allowed to make same origin request
unlimited cross-site requests by content scripts	✗	content scripts cannot change the `src` property of DOM elements to unauthorized origins

Conclusions

- Demonstrated spamming, phishing, and DDoS attacks with implemented Chrome extensions
- Analyzed the permissions model that causes these problems
- Proposed security enhanced permission model and enforcement for Chrome extension architecture
  - Micro-privileged permission management and spec
  - Differentiate content script's permission with DOM sensitivity levels

Thank You!

Q&A