# EXTENSION-LAND
# EXPLOITS AND ROOTKITS IN YOUR BROWSER EXTENSIONS

BARAK STERNBERG // DEFCON 2021

# ABOUT ME

- Barak Sternberg (@livingbeef)
- Senior Security Researcher, Previously Author @ SentinelOne Labs.
- “Hacking smart-devices for fun and profit” // DC28 IoT Village.
- BSc & MSC in CS on algorithms (bioinfo) from TAU.
- Focus from vulnerability-research (IoT, embedded devices, Linux and web apps) to analyze malwares in the wild.
- DJ & Party Lover (mixcloud.com/barak-sternberg)

# MOTIVATION

- More than 2 million extensions in webstores - attackers develop malicious ones & exploit.
- Why Extensions?
  - More permissions (easy “uXSS“ to any origin)
  - Controlling you entire browser & more
  - Cross-platform – works on any desktop/OS
  - Easier to develop – “JS-malware”

# SYLLABUS

1. Intro to chrome-extensions
2. Extensions communication
3. Exploiting Zotero - “Jumping” from one chrome-app to chrome-extension.
4. Exploiting Vimium – from PRNG’s to uXSS.
5. Developing & Implanting an "Extension-Rootkit“
6. Implanting a rootkit inside "good" extensions

# EXTENSIONS ANATOMY – THE BASICS

- Content-Scripts – Extensions' "frontend":
  - Loaded inside "matching" sites ("sites extension works for").
  - Runs in a special VM context (its own vars and private-world).
  - Accessible to site DOM.
- Background-Scripts – Extensions' "Backend":
  - Run once in a special dedicated process.
  - Access to more API's.
  - Persistent – non-site dependent.
- Extension-Dir - ***%LocalAppData%\Google\Chrome\User Data\Default\Extensions\EXTENSION_ID***
- Extension-Manifest - Manifest.json (the manifest.xml of extensions)
- Extension-Signature - Gets verified & checked at run-time.

# MANIFEST ANATOMY

manifest.json - Notepad

File Edit Format View Help

```
{
  "background": {
     "persistent": true,
     "scripts": [ "background_script_filename.js" ]
  },
  "content_scripts": [ {
     "all_frames": true,
     "js": [ "content_script_filename.js" ],
     "matches": [ "*://*/*" ],
     "run_at": "document_start"
  }, ... ],
  "content_security_policy": "script-src 'self'; object-src 'self'",
  "externally_connectable": {
     "matches": [ "*://*.freedownloadmanager.org/*" ]
  },
  "web_accessible_resources": [ "html/*.html", .. ],
  "permissions": [ "cookies", "storage", "history", "tabs", "notifications" ],
  "key": "MIIB…",
  "manifest_version": 2,
  "update_url": "https://clients2.google.com/service/update2/crx",
}
```

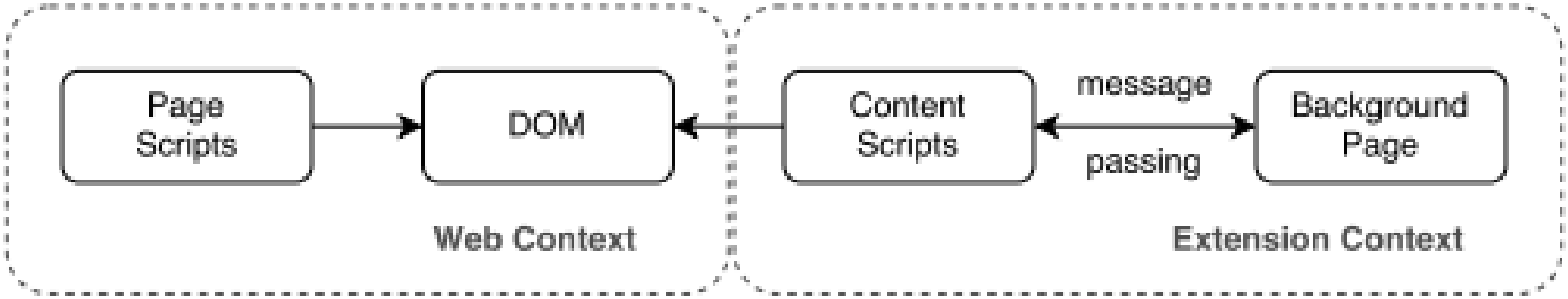
1

2

3

# COMMUNICATING IN EXTENSION-LAND

- For Example, let's say we go to https://google.com
- For every extension the following interactions are created.




*Picture Credit for "Attacking Browser Extensions" // Nicolas Golubovic

# CONTENT-SCRIPTS?

- Example: Ad-Blocker wants to remove ad-iframes from your page.
- How? It inspects the dom and remove them.
- Example code:

```
Let el = document.querySelector('div.slick-slide');
document.body.removeChild(el);
```

# BACKGROUND-SCRIPTS?

- Example: Ad-Blocker wants to block/redirect specific URL's.
- How? It adds new "WebRequest-Hook" and filters requests.
- Example code:

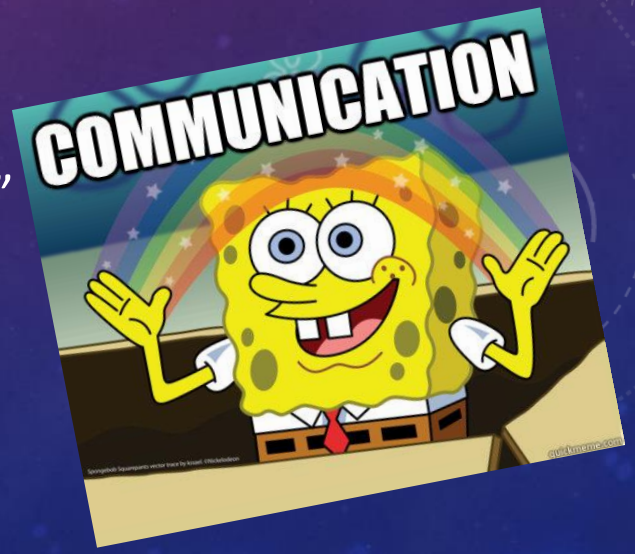
```
function ad_listener() {
	if (e.url === "https://BAD_SITE") {
		return {redirectUrl: "about:blank"};
	}
}
browser.webRequest.onBeforeRequest.addListener(ad_listener);
```

# WEBSITES <-> EXTENSION' CONTENT-SCRIPTS:

1. Cross-Origin Messages:
   - Content-Script: Defines "message" listeners
   - Website: window.postMessage("DATA", "chrome-extension://…");
2. DOM Changes & Events:
   - DOM Events - onclick/onfocus/onload
   - DOM Queries – search div with class=X
3. Extension Accessible URL's:
   - Manifest: "web_accessible_urls" (URL's that can be iframed/opened by other sites)
   - Website: <iframe src="chrome-extension://EXTENSION_ID/iframe.html"/>

# WEBSITES <-> EXTENSION' BACKGROUND-SCRIPTS:

1. WebRequest Proxy:
   - Background script: onBeforeRequest/onBeforeResponse…
2. Tabs/Cookies/Storage Inspections:
   - Background-Script: chrome.tabs Hooks / cookies.get(…) / chrome.downloads / chrome.storage.
3. Externally connected pages:
   - Manifest: A URL "http://X.com" is defined as "externally_connectable"
   - WebSite: sendMessage API available on http://X.com: chrome.runtime.sendMessage(EXTENSION_ID, "DATA")



# EXTENSION <-> EXTENSION PART 1

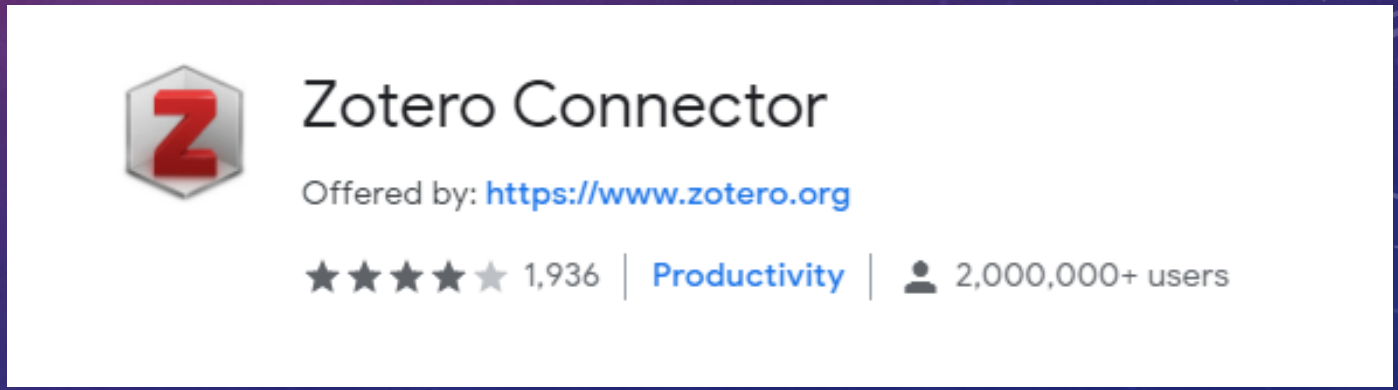
- All Website<->Extension comm is available.
- “Externally_Connectable” sites/extensions are allowed – sendMessage to background available.
- TCP/UDP connections.
  - Dependent on permissions.

<u>Cross-extensions Injection – of background-messages:</u>

- Extension 1 – injects code in <u>HTTPS://SITE.EXTENSION2.COM</u> :
  - *<u>chrome.runtime.sendMessage(EXTENSION_2_ID, DATA, ..);</u>*
- Extension 2 – receives message, “thinks” its from its site!

# ZOTERO EXTENSION

- Popular Academic extension used to organize citations/share research.
- Works with the “Zotero-Desktop” (saves data locally).
- Extension communicate with Zotero Desktop through TCP.



# ZOTERO TRANSLATORS OR "JUMPING" BETWEEN CHROME APPS/EXTENSIONS

- Zotero Translators –
  - 500+ JS-Translators can get executed at every site.
  - XSS/"Supply-chain" attacks - https://github.com/zotero/translators/
- Zotero's Translators' have auto-update system -
  - Check http://127.0.0.1:23119/GetTranslators for updates ("Zotero-Desktop" first).
  - Translators need to update? Get new JS code at http://127.0.0.1:23119/getTranslatorsCode
- Localhost Listener?
  - Download & Install my "Mappy" chrome-app! ☺
  - "Mappy" – a Chrome-app with one permission - "chrome.tcpServer".

# INJECTING JS IN ZOTERO CONTENT SCRIPTS: SANDBOX EXECUTION?

```
var parse = function(code) {
        Zotero.debug("Translate: Parsing code for " + translator.label + " "
                + "(" + translator.translatorID + ", " + translator.lastUpdated + ")", 4);
        try {
                this._sandboxManager.eval(
                        "var exports = {}, ZOTERO_TRANSLATOR_INFO = " + code,
                        [
                                "detect" + this._entryFunctionSuffix,
                                "do" + this._entryFunctionSuffix,
                                "exports",
                                "ZOTERO_TRANSLATOR_INFO"
                        ],
                        (translator.file ? translator.file.path : translator.label)
                );
        }
        catch (e) {
```

Our Translator JS Executed

# INJECTING JS IN ZOTERO CONTENT SCRIPTS: SANDBOX EXECUTION?

```
var parse = function(code) {
        Zotero.debug("Translate: Parsing code for " + translator.label + " "
                + "(                                                    + ")", 4);
        try {
                this
                                                                        el)
                );
        }
        catch (e) {
```

Our Translator JS Executed

# WITH CONTENT:
# EXPLORING ATTACK SURFACE OF CONTENT-SCRIPTS

- Inside content-scripts:
  - sendMessage/Connect
  - Access to shared extension URL's
  - Storage/Configuration
- Inside Zotero Background Scripts:
  - An interesting "eval" inside google docs integration
  - Why and how it is done?

# WITH CONTENT:
# INJECTING JS INTO ZOTERO BACKGROUND CONTEXT

```
// Then fetch code from server
let serverURL = Zotero.Prefs.get('integration.googleDocs.codeRepositoryURL');
if (!serverURL) return;

try {
        Zotero.debug("Checking for updated remote Google Docs scripts");

        let xhr = await Zotero.HTTP.request('GET', serverURL + "package.json");
        let serverVersion = JSON.parse(xhr.responseText).version;
        let serverHasNewerVersion = Zotero.Utilities.Internal.semverCompare(this.version, serverVersion) < 0;
        if (!serverHasNewerVersion) {
                Zotero.debug("Google Docs scripts are up to date");
                return;
        }
        this.version = serverVersion;

        Zotero.debug(`Fetching Google Docs scripts from ${serverURL}: ${JSON.stringify(paths)}`);
        this.scriptContents = await this._fetchScripts(serverURL, paths);
        Zotero.debug('Remote Google Docs scripts fetched, reloading');
        this.loadBackgroundScripts();
}
```

Prefs from chrome.storage

Injecting Scripts

# WITH CONTENT:
# INJECTING JS INTO ZOTERO BACKGROUND CONTEXT

```
135        loadBackgroundScripts: async function() {
136                if (Zotero.version === '4.999.0') return;
137                Zotero.debug(`Loading Google Docs background scripts: ${JSON.stringify(this.backgroundScriptPaths)}`);
138                for (let path of this.backgroundScriptPaths) {
139                        try {
140                                eval(this.scriptContents[path]);
141                        }
142                        catch (e) {
143                                Zotero.debug(`Failed to load Google Docs background script "${path}"`, 1);
144                                Zotero.logError(e);
145                        }
146                }
147        },
148
```

Eval execution
In Background

# WITH CONTENT:
# INJECTING JS INTO ZOTERO BACKGROUND CONTEXT

- Config Injection?
  - Chrome.storage.local is shared across content & background scripts!
    - Inject new config from content-scripts.
    - Trigger XSS inside background-scripts ☺
- Loaded every time background context re-starts.

# “FULL-CHAINING ZOTERO” DEMO VIDEO

About Version | Extensions | Google

Chrome | chrome://extensions

# Extensions

Search extensions

Developer mode

Load unpacked | Pack extension | Update

Details | Remove | Details | Remove | Details | Remove

**Zoom Scheduler** 1.6.3

Schedule Zoom meetings directly from Google Calendar

ID: kgjfgplpablkjnlkjmjdecgdpfankdle

Details | Remove

**Zotero Connector** 5.0.84

Save references to Zotero from your web browser

ID: ekhagklcjbdpajgpjgmbionohlpdbjgc

Inspect views background page

Details | Remove

## Chrome Apps

**Mappy2** 2.0

Finds addresses in the web page you're on and pops up a map window.

ID: mpnnapbhhjpibfddpofdkgmlebbeikcn

Inspect views background page (Inactive)

Details | Remove

**Docs** 0.10

Create and edit documents

ID: aohghmighlieiainnegkcijnfilokake

Details | Remove

**Sheets** 1.2

Create and edit spreadsheets

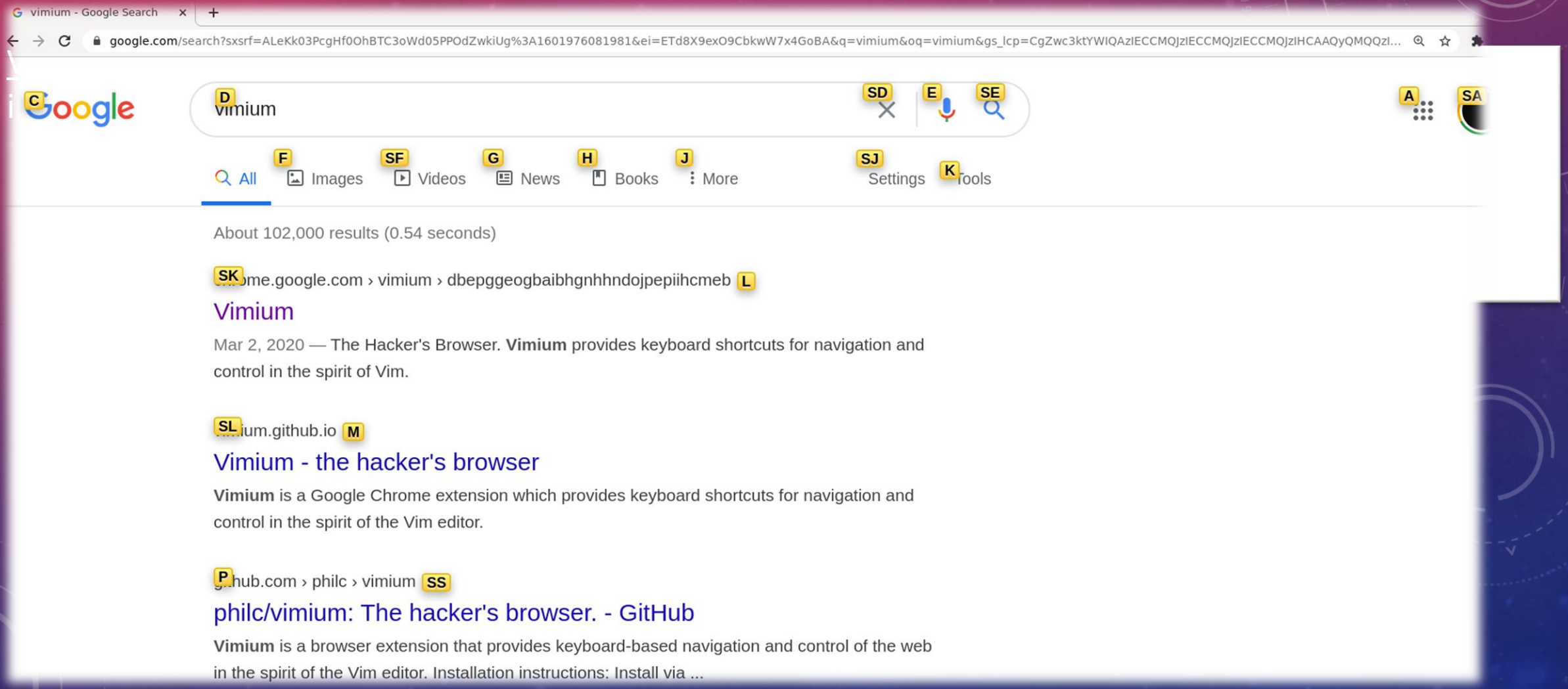
ID: felcaaldnbdncclmgdcncolpebgiejap

Details | Remove

Type here to search

ENG 1:41 AM 4/4/2021

# VIMIUM'ING FOR FUN AND PROFIT



# VIMIUM'ING FOR FUN AND PROFIT

Attack Scenario: You can make a user execute JS in your site (e.g: Ad, site, permission-less third-party extension, etc).

Goal: Attack Vimium Extension.

Vimium Widgets:

- Vomnibar widget
- Helper widget
- Visual-mode widget

# VIMIUM'ING FOR FUN AND PROFIT

## Website Context

## Content-Script Context

example.com

Example Domain

This
may use this domain in literature witho
asking for permission.

More information...

Elements Console Sources Network Performance Memo

(index) ui_component.js ×

```
48
49        this.iframeElement.src = chrome.runtime.getURL(iframeUrl)
          document.documentElement.appendChild(shadowWrapper);
51
52        this.iframeElement.addEventListener("load", () => {
53          // Get vimiumSecret so the iframe can determine that ou
54          chrome.storage.local.get("vimiumSecret", ({ vimiumSecre
            const { port1, port2 } = new MessageChannel;
56          this.iframeElement.contentWindow.postMessage(vimiumSe
57          port1.onmessage = event => {
58            let eventName = null;
59            if (event)
60              eventName = (event.data ? event.data.name : undef
61
            ch (eventName) {
            "uiComponentIsReady":
64            // If any other frame receives the focus, then hi
65            chrome.runtime.onMessage.addListener(({name, focu
```

**0. User Clicks "O" & Enter**

**1. Vimium Content-Script catch it & adds iframe**

**2. Vomnibar iframe added**

**3. Authorize to iframe & postMessage with "vimiumSecret" token.**

**4. Vimium Content-Script & Vomnibar iframe communicate freely.**

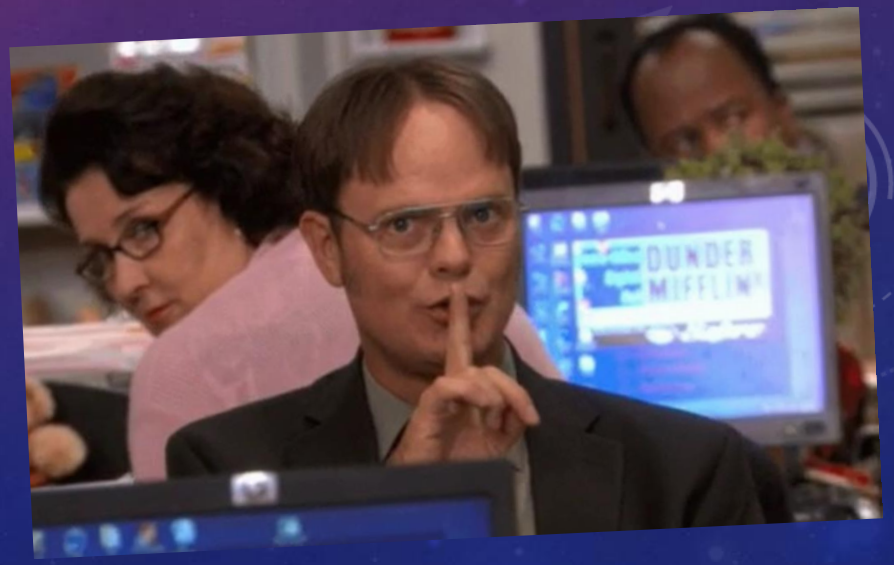
# VIMIUM'ING FOR FUN AND PROFIT: BREAKING THE VIMIUM SECRET #1

"VimiumSecret" Generation:

- Very "State-of-The-Art" Random Number Generator:

***chrome.storage.local.set({vimiumSecret: Math.floor(Math.random() * 2000000000)});***

- Math.random prediction works in same-process, the token is generated inside background process ☹
- Bruteforce?
  - Inject vomnibar iframe
  - Try to connect?



# VIMIUM'ING FOR FUN AND PROFIT: BREAKING THE VIMIUM SECRET #2

- Bruteforce PostMessage's 101:

```
let secret_to_bruteforce = 0xdeadbeef;
d = document.createElement('iframe');
d.src = 'chrome-extension://dbepggeogbaibhgnhhndojpepiihcmeb/pages/vomnibar.html';
document.body.appendChild(d);
d.contentWindow.postMessage(secret_to_bruteforce, '*', [channel.port2]);
```

- If success – Getting success response through "channel.port1"
- If fail – No response

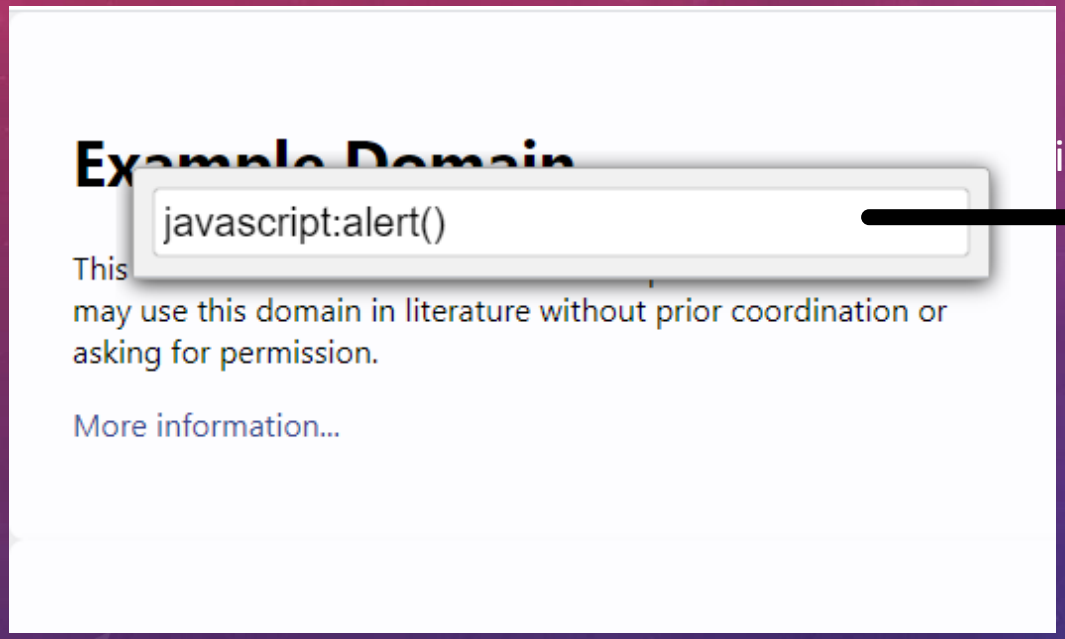
WebWorkers stays-up: as long as chrome & website not closed actively -

- Works when the screen is closed.
- Works when tab/window is hidden.

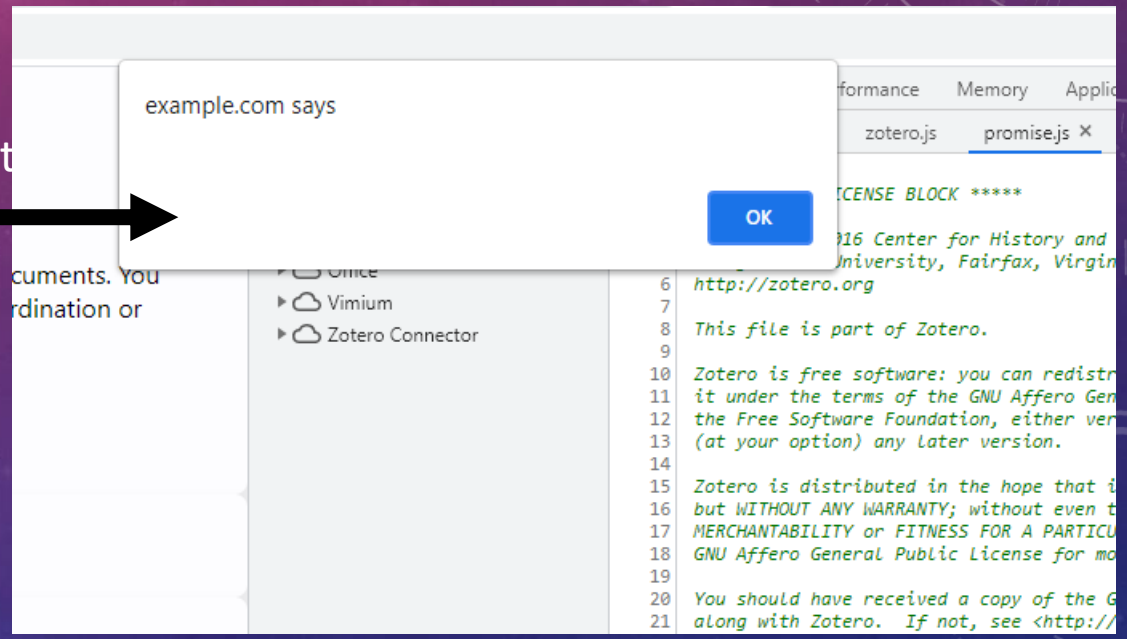
# VIMIUM'ING FOR FUN AND PROFIT: VOMNIBAR COMMUNICATION

- What is the communication between "Vimium" Content-Script & Vomnibar iframe:
  - Search for URL completions
  - Activate search / jump to new URLs.
  - Search for hints & Auto-completion.
  - Run JS code.

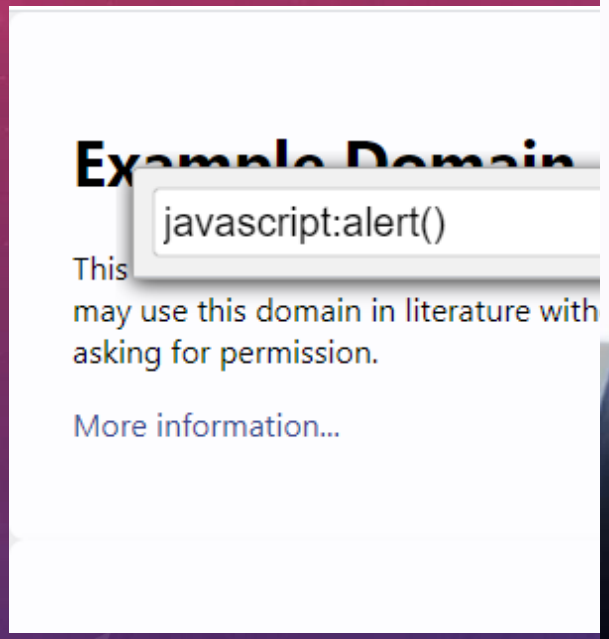
# VIMIUM'ING FOR FUN AND PROFIT: VOMNIBAR COMMUNICATION

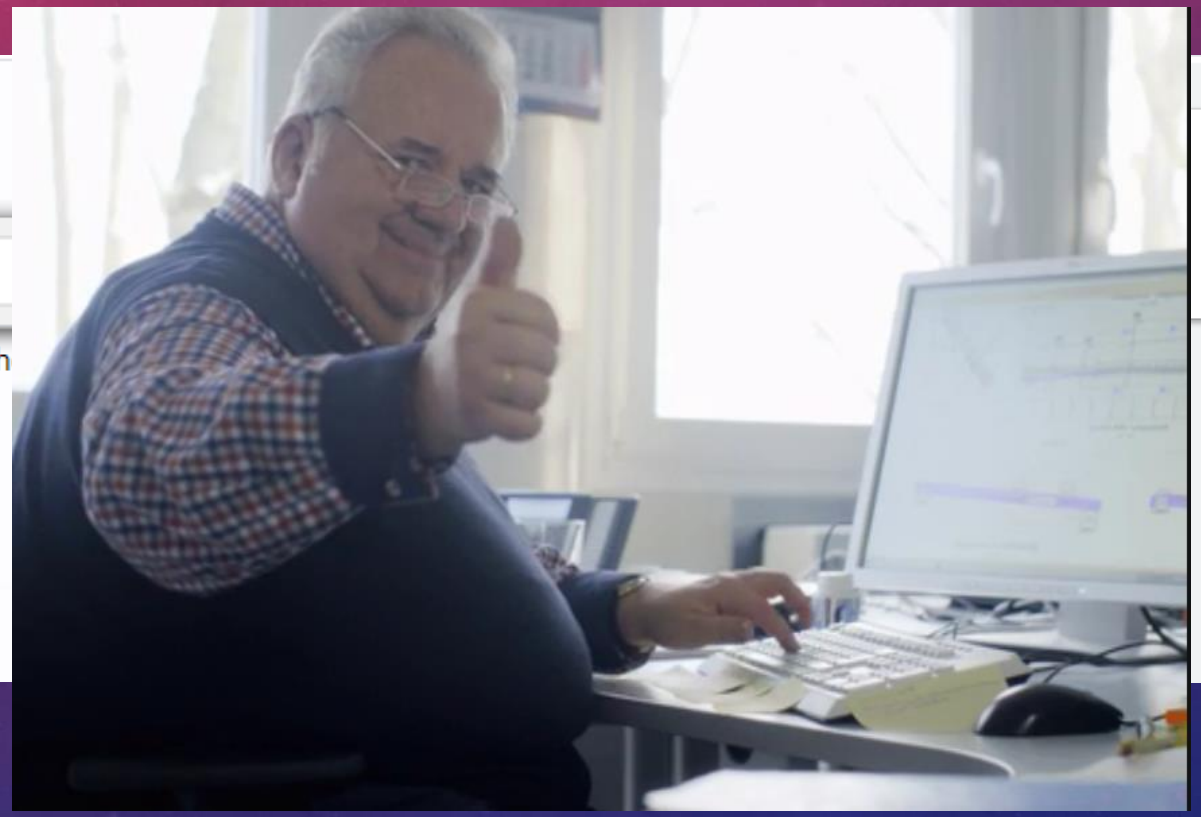


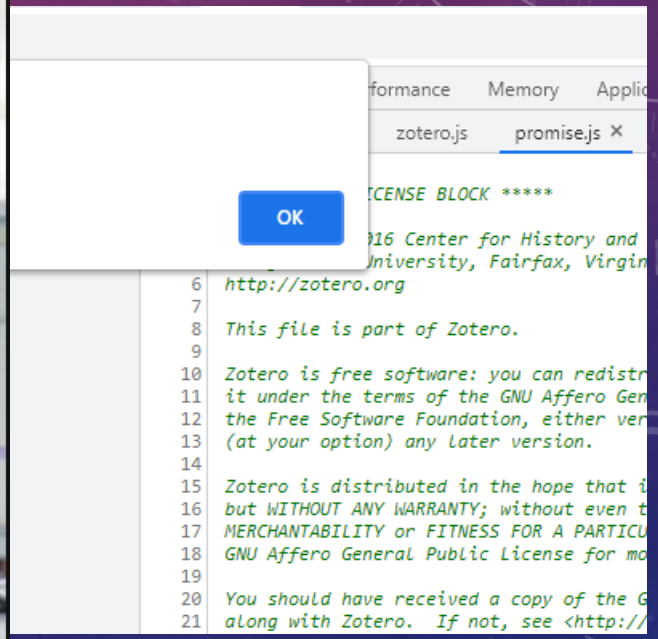

ium" Content



# VIMIUM'ING FOR FUN AND PROFIT: VOMNIBAR COMMUNICATION





# VIMIUM'ING FOR FUN AND PROFIT: VIMIUM COMMUNICATION #2

How Vomnibar handle javascript scheme?

- Tries to find auto-completion.
- Calls background-script method to find relevant auto-complete.
- Background-script "sendMessage" back to sender tab's content-scripts.

Problema?

- How about placing another iframe inside our tab?
- Vimium Content-Scripts are loaded at any iframe on tab.
- No validation for targeted url/frameId – JS executed in all iframes!

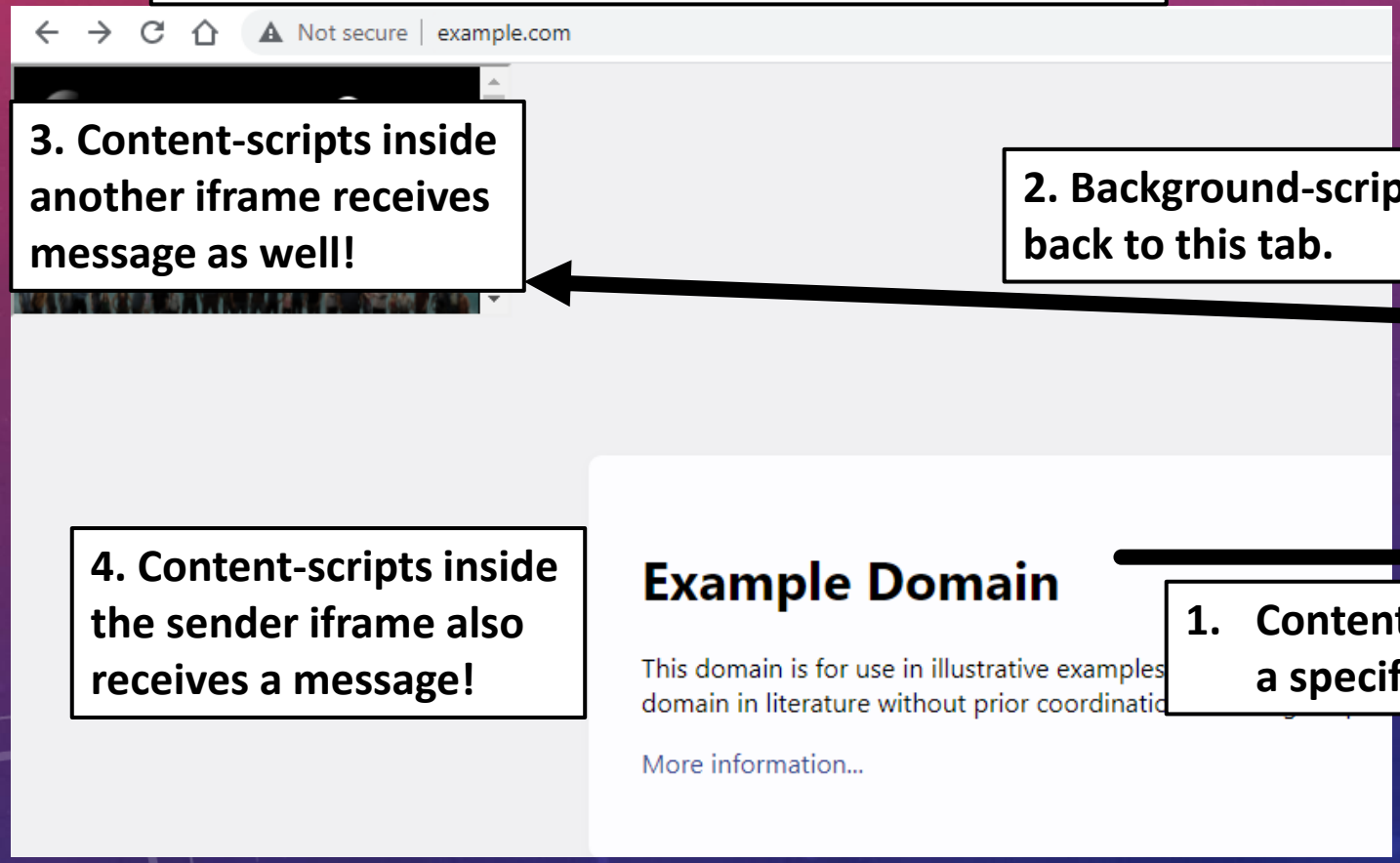
Why? Read sendMessage reference:

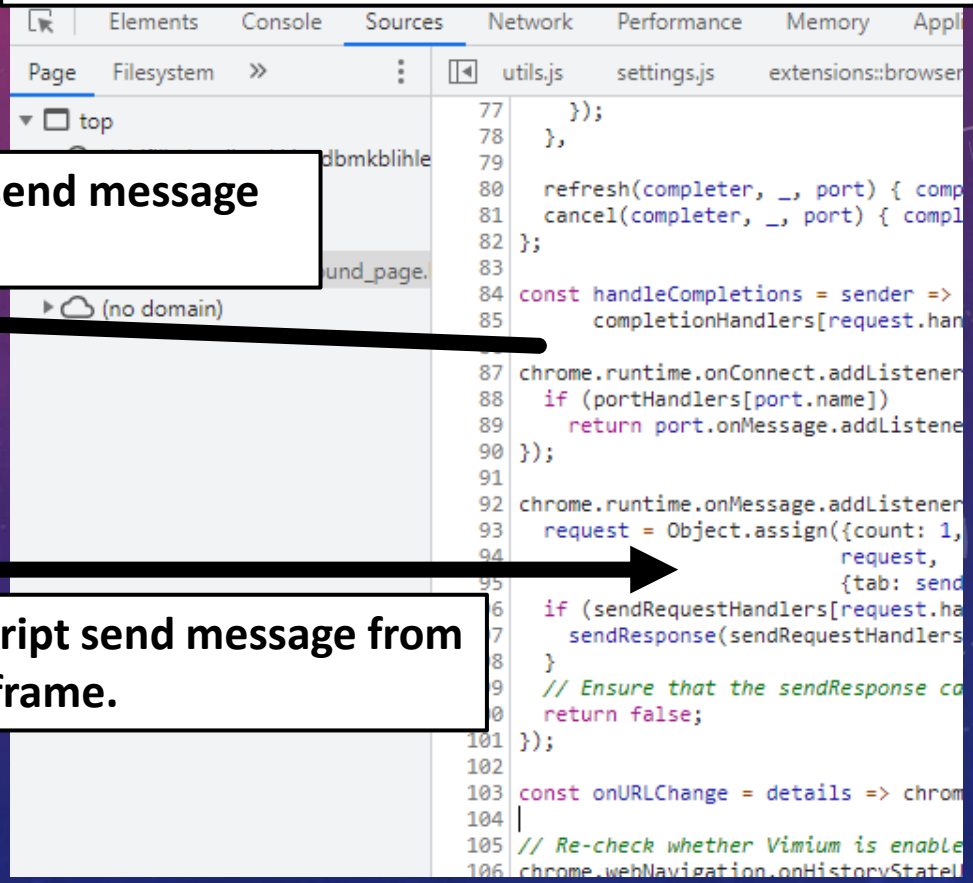
**"The runtime.onMessage event is fired in each content script running in the specified tab for the current extension."**

# VIMIUM'ING FOR FUN AND PROFIT: CONTENT-SCRIPTS MESSAGING INJECTION

## Content-Script Context

## Background Context




**3. Content-scripts inside another iframe receives message as well!**

**2. Background-scripts send message back to this tab.**

**4. Content-scripts inside the sender iframe also receives a message!**

1. **Content-Script send message from a specific iframe.**

# VIMIUM'ING FOR FUN AND PROFIT: UXSS DEMO

About Version | Extensions - Vimium | Vimium - Chrome Web Store

Chrome | chrome://version

**Google Chrome:** 89.0.4389.114 (Official Build) (64-bit) (cohort: Stable)
**Revision:** 1ea76e193b4fadb723bfea2a19a66c93a1bc0ca6-refs/branch-heads/4389@{#1616}
**OS:** Windows 10 OS Version 1909 (Build 18363.1440)
**JavaScript:** V8 8.9.255.24
**User Agent:** Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.114 Safari/537.36
**Command Line:** "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --load-extension="C:\Users\ASUS\AppData\Local\Google\Chrome\User Data\Default\Extensions\cfhdojbkjhnklbpkdaibdccddilifddb\3.10.2_0" --flag-switches-begin --disable-features=WebRtcRemoteEventLog --flag-switches-end --origin-trial-disabled-features=SecurePaymentConfirmation
**Executable Path:** C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
**Profile Path:** C:\Users\ASUS\AppData\Local\Google\Chrome\User Data\Default
**Variations:**
af81735d-ca7d8d80
84085631-ab02a1cf
90a7075b-12ede6a2
16b16054-ca7d8d80
b0f75187-377be55a
59b6f412-ca7d8d80
60d4b352-cdbea37c
4ca682fe-377be55a
b3249ec4-ca7d8d80
a9ef513c-ca7d8d80
8ae424bf-ca7d8d80
9d6a857b-4804dae4
aed6e5d4-ca7d8d80
8816d952-377be55a
3095aa95-3f4a17df
c27fec31-2c161c2f
4d936449-ca7d8d80
6d6d60a5-5acaf23b
38b9885d-ca7d8d80
edb58ea9-cdbea37c
9a13dddd-ca7d8d80
9e604a08-377be55a
deb1cb12-ca7d8d80
8e44abde-d03dcea1
47b5f350-377be55a

chrome

# PERSISTENT JS INJECTION INTO ANY EXTENSION OR GOTTA LOVE KUNPACKED

- Scenario: Post-Exploitation, Managed to run code over users' device.
- Goal: install a persistent "rootkit"
- Extension unpacked-mode?
  - Argument –load-extension=YOUR_EXTENSION_PATH
  - Replace original extension – keeps its ID but still can change files/perms.
- Modifying "good" extension:
  - Adding Any permissions as needed – cookies/tabs/sites and more.
  - Full File-System Access (Read-Access)
  - Hidden – All is done in chrome context
  - Access to user cookies, mail, data, tabs, and much more in user context.

# KUNPACKED DEMO TIME

About Version | Adblock Plus - free ad blocker - | Extensions

Chrome | chrome://version

**Google Chrome:** 89.0.4389.114 (Official Build) (64-bit) (cohort: Stable)
**Revision:** 1ea76e193b4fadb723bfea2a19a66c93a1bc0ca6-refs/branch-heads/4389@{#1616}
**OS:** Windows 10 OS Version 1909 (Build 18363.1440)
**JavaScript:** V8 8.9.255.24
**User Agent:** Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.114 Safari/537.36
**Command Line:** "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --load-extension="C:\Users\ASUS\AppData\Local\Google\Chrome\User Data\Default\Extensions\cfhdojbkjhnklbpkdaibdccddilifddb\3.10.2_0" --flag-switches-begin --disable-features=WebRtcRemoteEventLog --flag-switches-end --origin-trial-disabled-features=SecurePaymentConfirmation
**Executable Path:** C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
**Profile Path:** C:\Users\ASUS\AppData\Local\Google\Chrome\User Data\Default
**Variations:**
af81735d-ca7d8d80
84085631-ab02a1cf
90a7075b-12ede6a2
16b16054-ca7d8d80
b0f75187-377be55a
59b6f412-ca7d8d80
60d4b352-cdbea37c
8ae424bf-ca7d8d80
9d6a857b-4804dae4
aed6e5d4-ca7d8d80
8816d952-377be55a
3095aa95-3f4a17df
c27fec31-2c161c2f
4d936449-ca7d8d80
6d6d60a5-5acaf23b
38b9885d-ca7d8d80
edb58ea9-cdbea37c
9a13dddd-ca7d8d80
9e604a08-377be55a
deb1cb12-ca7d8d80
8e44abde-d03dcea1
47b5f350-377be55a
8f83697a-ca7d8d80
8f000ce6-607070a4
f2cb61f-ca7d8d80

chrome



Type here to search

ENG 1:35 AM 4/4/2021

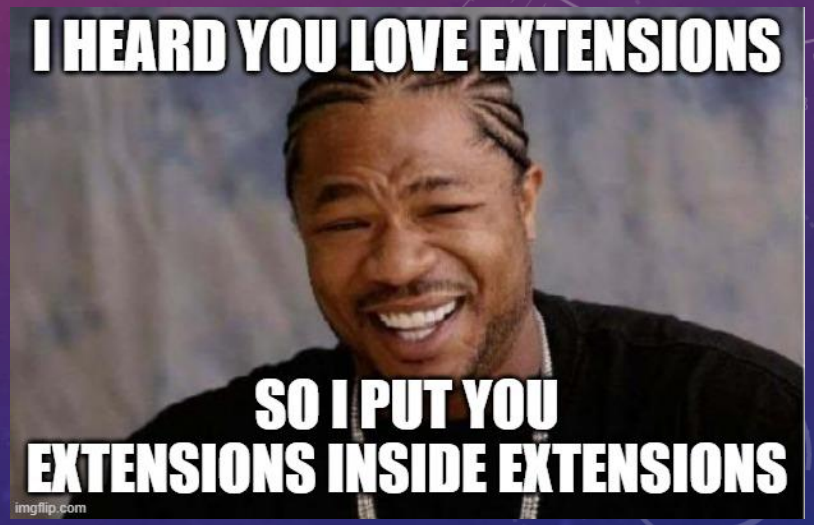
# INTRODUCING “MALTENSIONS”: GENERATOR FOR JS-MAWARE INSIDE EXTENSIONS

Utility to generate and test malware-techniques inside your browser extensions.

Code: https://github.com/barakolo/Maltensions

Featured-Techniques:

- Inject & run JS in hidden context inside tabs.
- File-System Access /Access to sites/tabs/user-storage data.
- C&C communication.
- Output Formats:
  - Unpacked extension mode
  - JS to inject inside your favorite extension.



# CONCLUSIONS

1. Extension can be abused for “PE” –
   - Extensions may abuse others to gain privs & stay hidden!
2. Detections will get harder –
   - Injection of malicious scripts inside “good” extensions!
   - Hidden techniques to exfiltrate data!
3. More Attack surfaces to explore:
   - inner communication (cs <-> bg, bg <-> website …)
   - Attack surface from one extension to another.
   - storage mis-configs & injections.
4. Malicious extensions are here to stay!

# THANK YOU!

# CREDITS & EXTRA-MATERIALS

- p.4,12,33,41 - URL & picture credits  - Generated & Downloaded from - https://imgflip.com
- p.3 – picture Credits go to Mozilla Foundation / Mozilla Firefox & Google, Google Chrome.
- p.19 – credit goes to https://memecreator.org.
- p.18,22,38 - https://www.pinterest.at/pin/410672059765026188/  / https://knowyourmeme.org / https://memesdroid.org / https://tenor.com/view/hacker-gif-18087134
- p.28 - https://gfycat.com/gifs/tag/schrute
- p.12 - https://memegenerator.net/instance/57339379/spongebob-rainbow-communication
- Any other picture/extra-materials being used, besides the lecture content, are fully credited to their respective owners, if an author/any owner wants to add copyrights/credits – please contact us and it can be added accordingly.