

Browser Extension (In)Security

Aurore Fass

Tenure-Track Faculty at CISPA – Helmholtz Center for Information Security

UNIMORE, Modena, December 18th, 2024

Dr.-Ing. Aurore (/ʊRʊR/) FASS

🇫🇷 Graduated from TELECOM Nancy (FR, 2017)



🇩🇪 PhD Student + Postdoc at CISPA (DE, 2017–21)

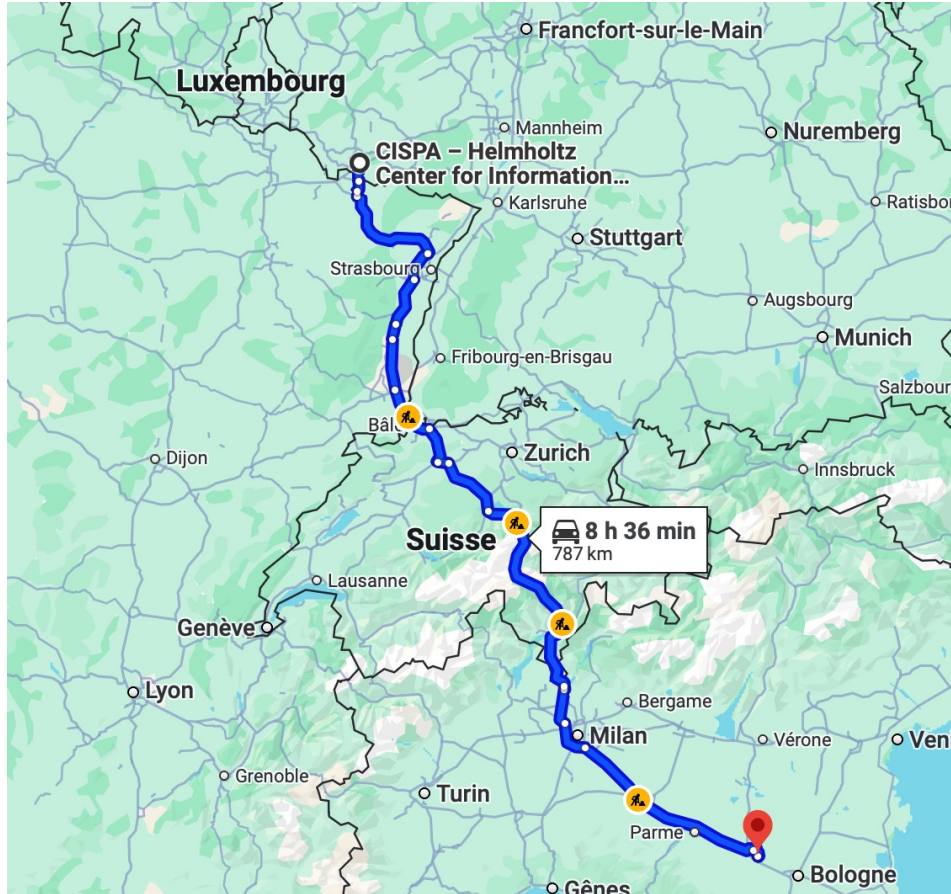
🇺🇸 Visiting Assistant Professor at Stanford (US, 2021–23)



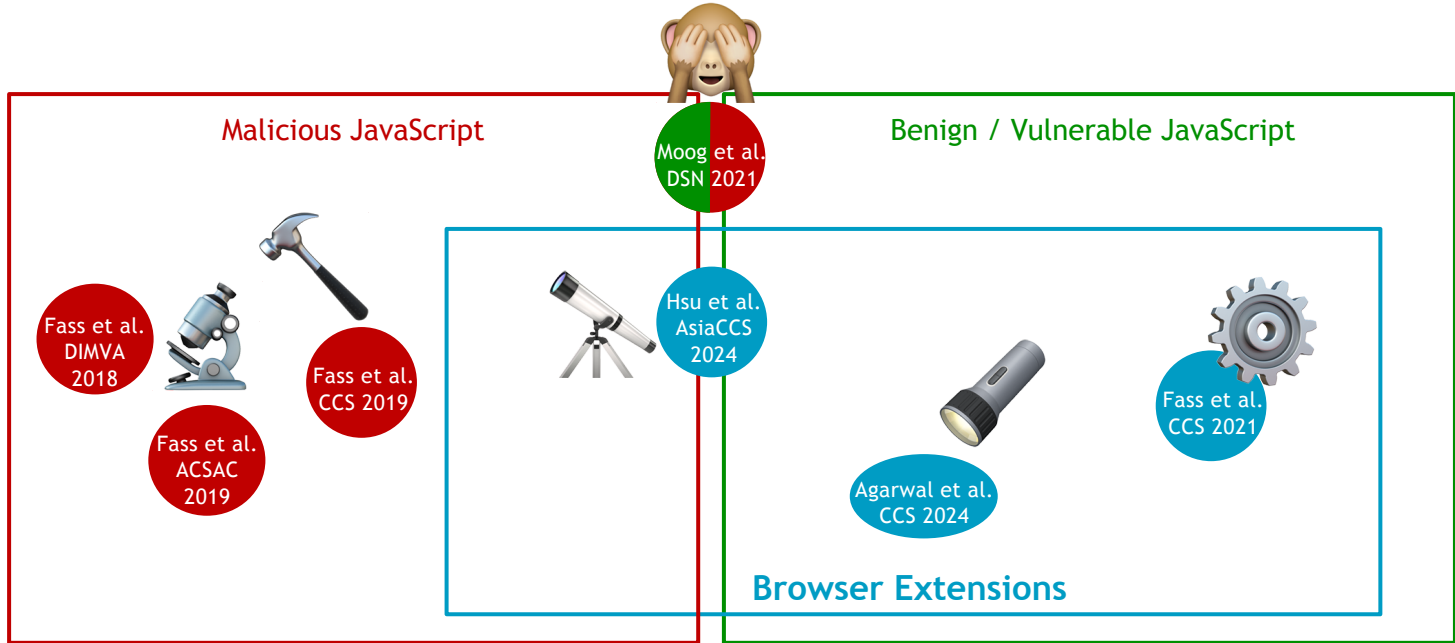
🇩🇪 Tenure-Track Faculty at CISPA (DE, 2023–)



CISPA - Helmholtz Center for Information Security



Research Work: Web Security & Privacy



Ruth et al.
IMC 2022

Internet Measurements

Izhikevich et al.
IMC 2023



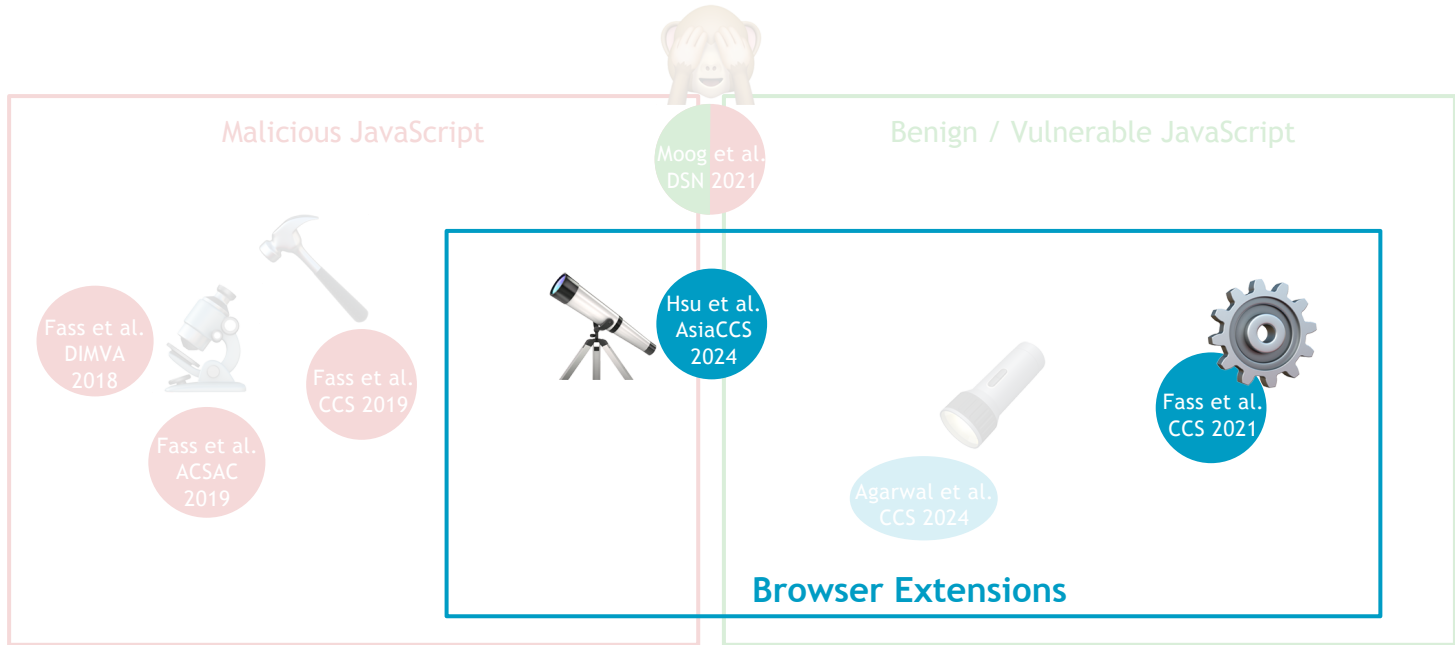
Apruzzese et al.
AISeC 2024

Machine Learning

Software Engineering

Troppmann et al.
ASE 2024

Research Work: Web Security & Privacy



Apruzzese et al.
AISeC 2024

Machine Learning



Ruth et al.
IMC 2022

Internet Measurements

Izhikevich et al.
IMC 2023



Software Engineering

Troppmann et al.
ASE 2024

Outline

- Background
 - Malicious/vulnerable JavaScript
 - Browser extensions
- Investigating Security-Noteworthy Extensions (SNE)
 - SNE definition
 - Extension analysis & SNE detection
- Detecting Vulnerable Extensions
 - Threat models & examples
 - Static analysis of extensions (JavaScript) & examples
 - Case studies, results, and potential defense strategies

Outline

- **Background**

- Malicious/vulnerable JavaScript
- Browser extensions

- **Investigating Security-Noteworthy Extensions (SNE)**

- SNE definition
- Extension analysis & SNE detection

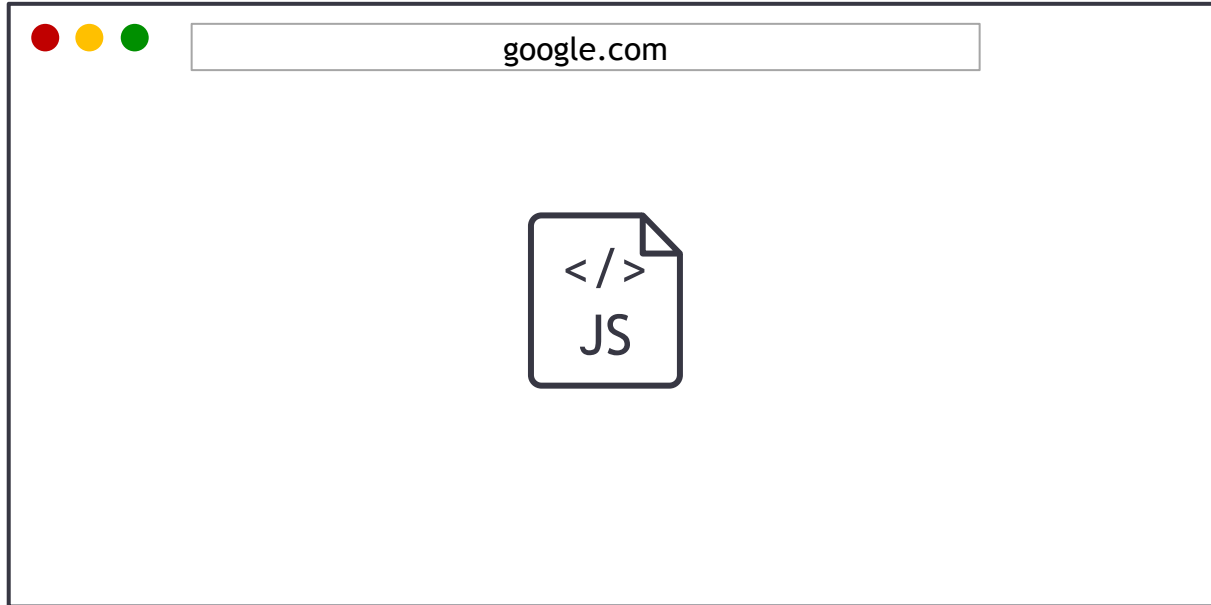
- **Detecting Vulnerable Extensions**

- Threat models & examples
- Static analysis of extensions (JavaScript) & examples
- Case studies, results, and potential defense strategies

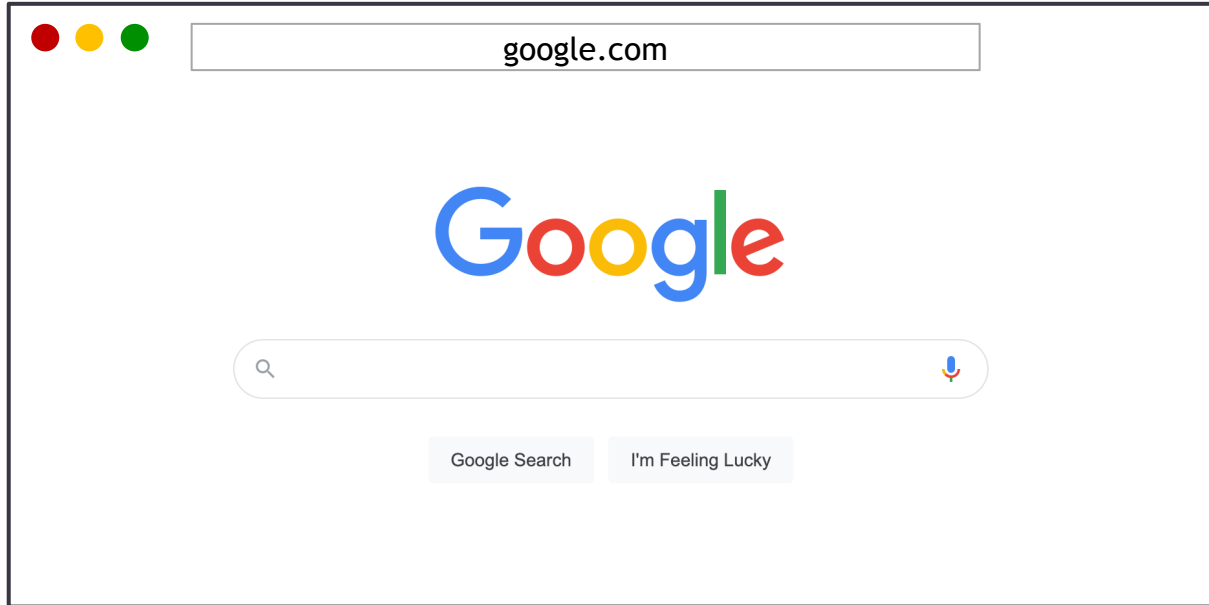
Background – JavaScript

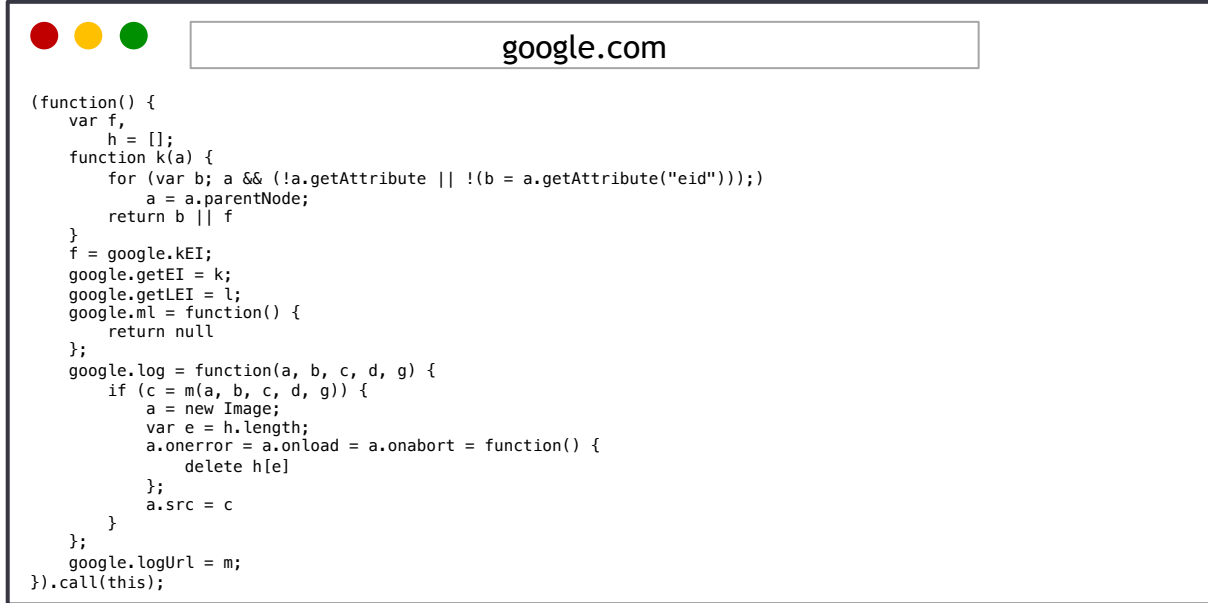


Background – JavaScript



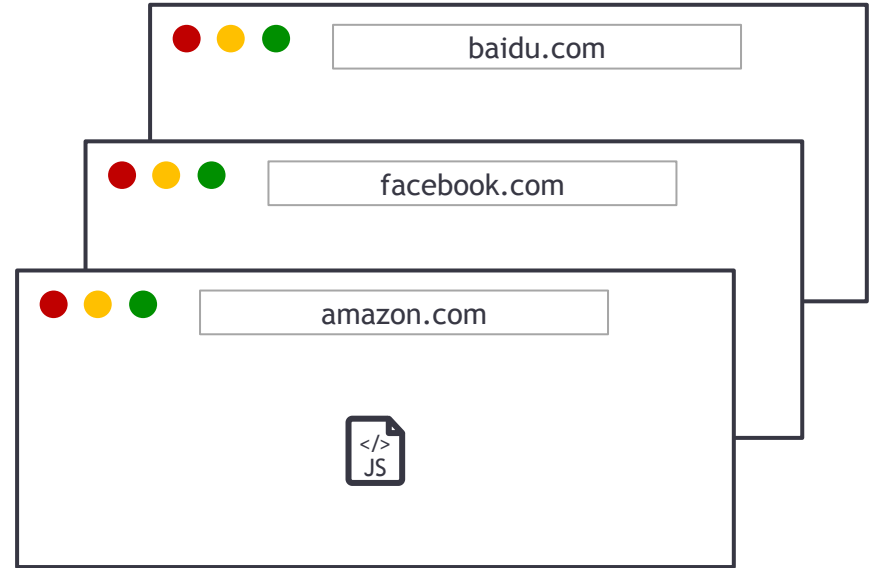
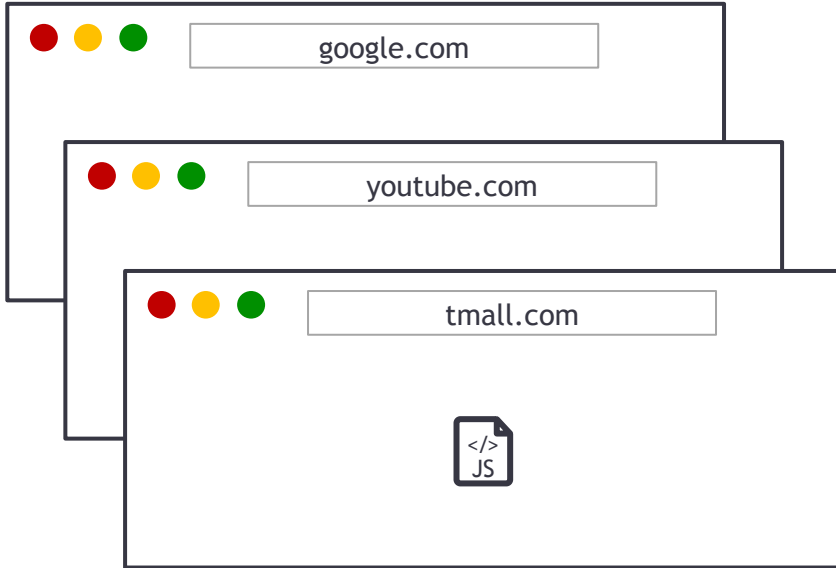
Background – JavaScript





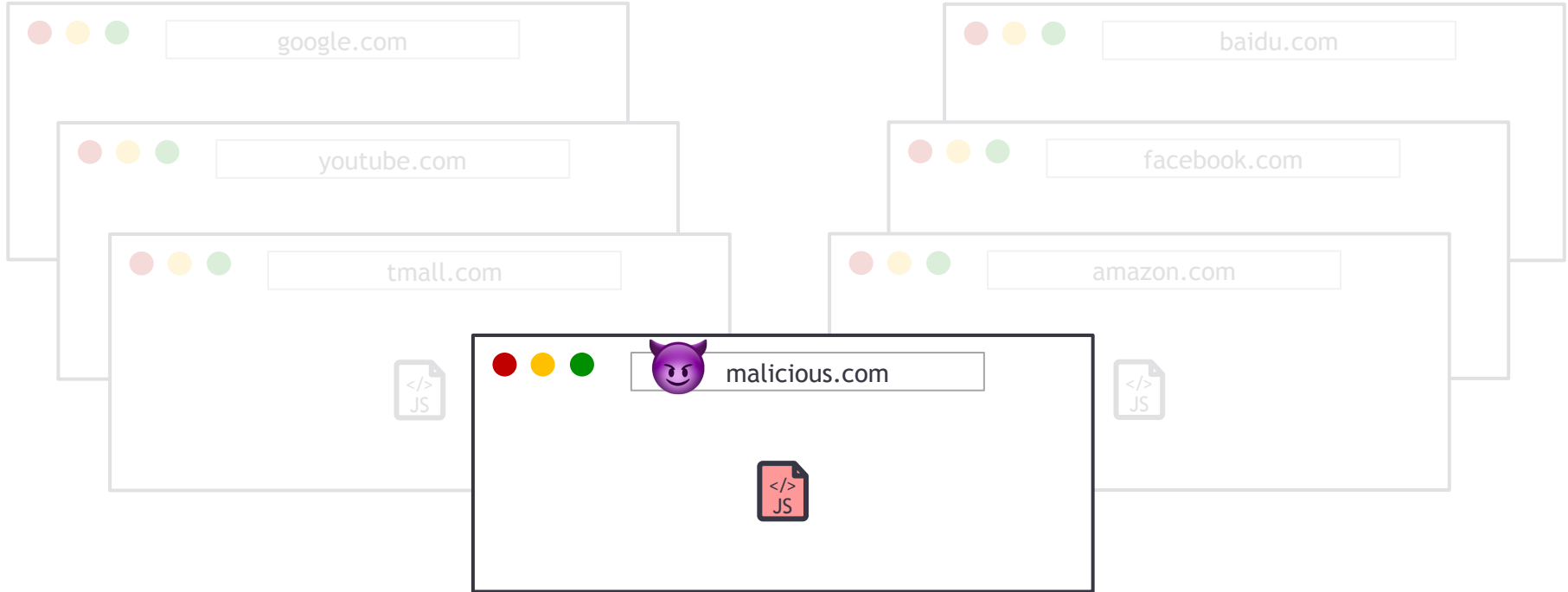
```
(function() {
  var f,
      h = [];
  function k(a) {
    for (var b; a && (!a.getAttribute || !(b = a.getAttribute("eid")));)
      a = a.parentNode;
    return b || f
  }
  f = google.kEI;
  google.getEI = k;
  google.getLEI = l;
  google.ml = function() {
    return null
  };
  google.log = function(a, b, c, d, g) {
    if (c = m(a, b, c, d, g)) {
      a = new Image;
      var e = h.length;
      a.onerror = a.onload = a.onabort = function() {
        delete h[e]
      };
      a.src = c
    }
  };
  google.logUrl = m;
}).call(this);
```

Background – JavaScript



JavaScript usage: > 99% websites

Background – JavaScript



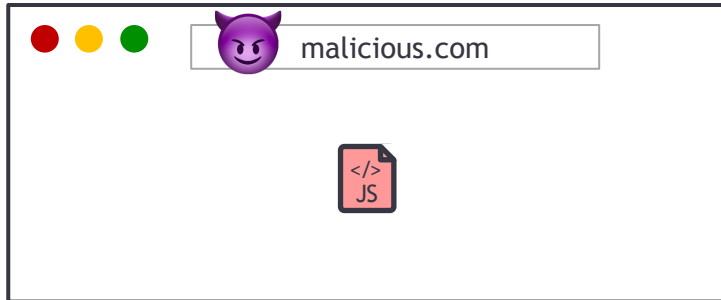
JavaScript usage: > 99% websites

Malicious JavaScript

- Designed by malicious actors
- Aim: harming victims
 - e.g., exploiting vulnerabilities, stealing sensitive user data

Malicious JavaScript

- Designed by malicious actors
- Aim: harming victims
 - e.g., exploiting vulnerabilities, stealing sensitive user data



Malicious JavaScript



Malicious actor
Tax reimbursement
To: me

Link



malicious.com



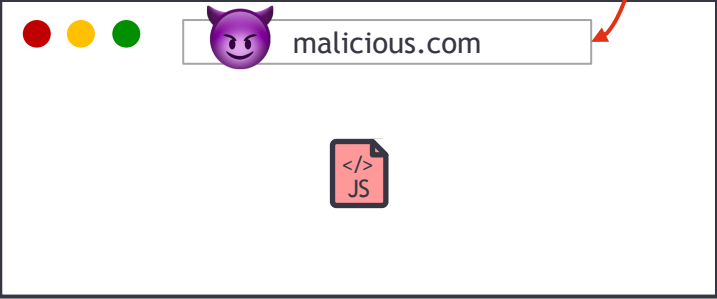
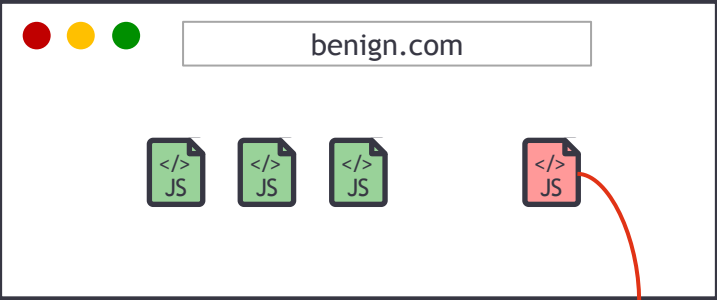
Malicious JavaScript



Malicious actor
Tax reimbursement
To: me



Malicious JavaScript

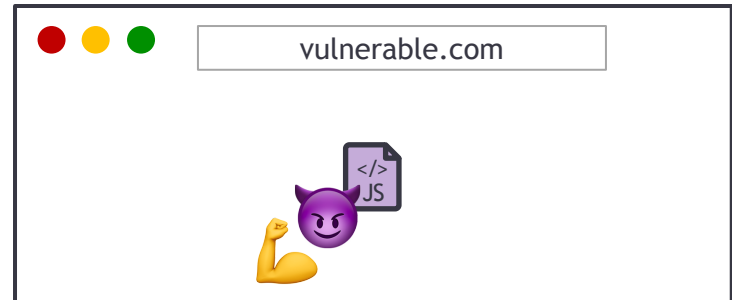


Malicious JavaScript

- Designed by malicious actors
- Aim: harming victims, e.g.,
 - exploiting vulnerabilities to download & execute malware,
 - stealing sensitive user data

-but-buggy Benign JavaScript

- Designed by well-intentioned developers
- ... but contains some vulnerabilities
 - can be exploited by malicious actors



Malicious JavaScript

- Designed by malicious actors
- Aim: harming victims, e.g.,
 - exploiting vulnerabilities for drive-by-download & execute malware,
 - to hijack sensitive user data

^{-but-buggy} Benign JavaScript

- Designed by well-intentioned developers
- ... but contains some vulnerabilities
 - can be exploited by malicious actors

**What about if you install a
malicious or vulnerable browser
extension?**

Background – What are Browser Extensions?

- Third-party programs to **improve user browsing experience**



Adblock — best ad blocker

Offered by: getadblock.com



Adblock Plus - free ad blocker

Offered by: adblockplus.org



Adobe Acrobat

Offered by: Adobe Inc.



Avast Online Security

Offered by: <https://www.avast.com>



Cisco Webex Extension

Offered by: webex.com



Google Translate

Offered by: translate.google.com



Grammarly for Chrome

Offered by: grammarly.com



Honey

Offered by: <https://www.joinhoney.com>



Pinterest Save Button

Offered by: pinterest.com



Skype

Offered by: www.skype.com



uBlock Origin

Offered by: Raymond Hill (gorhill)



LastPass: Free Password Manager

Offered by: LastPass

- Bundles* of JavaScript, HTML, or CSS files, defined in a `manifest.json`
- ~145k Chrome extensions totaling over 1.6B active users

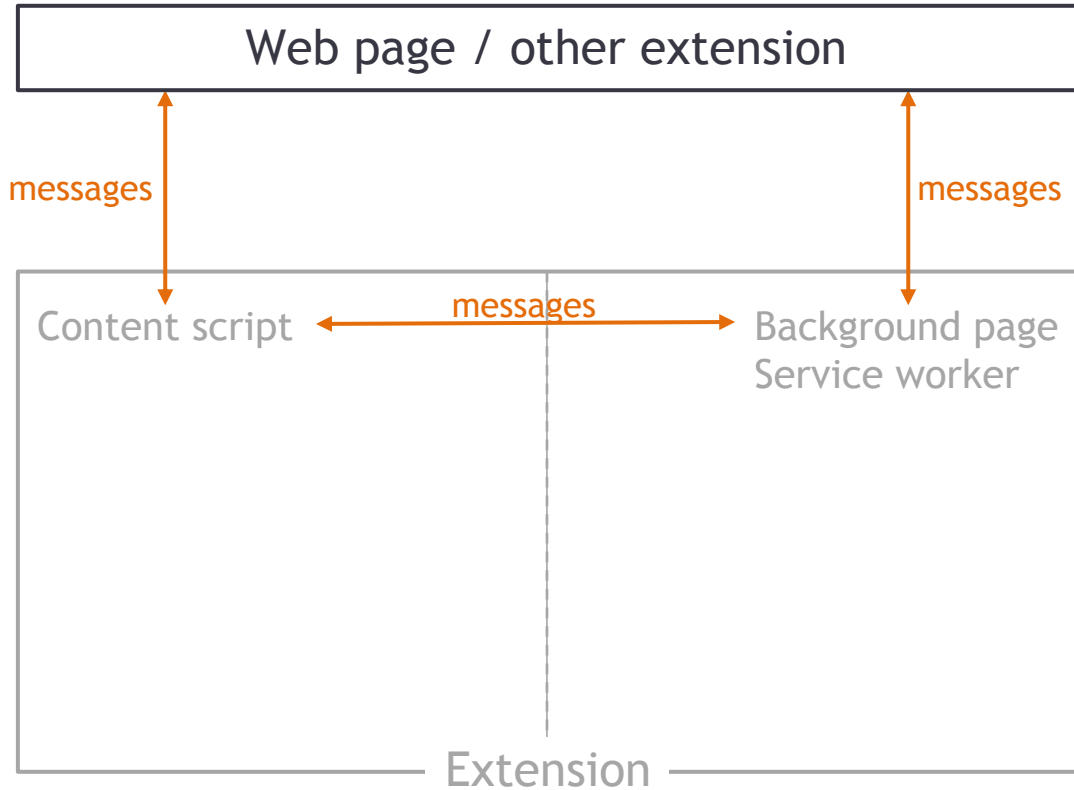
- Extensions only have access to:
 - APIs explicitly declared in the `manifest.json`, e.g.,
 - storage - store/access data from the *extension storage*
 - downloads - download files
 - history - access to a user's browsing history
 - bookmarks, cookies, topSites, ...
 - host declared in the `manifest.json` = web pages an extension can access (read/write), e.g., to do some *cross-origin* requests

- https://developer.chrome.com/docs/extensions/mv3/declare_permissions/

- <https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/permissions>

- Background page (BP) / Service worker (SW):
 - Core logic of an extension
 - Executed independently of the lifetime of a tab / window
 - Privileged part of an extension
- Content scripts (CS):
 - Injected by an extension into (a) web page(s)
 - Can use standard DOM APIs to read / modify a web page
 - Similar to scripts directly loaded by a web page + some more privileges
 - Restricted access to extension APIs

Background – Extension Architecture & Messages



- Every extension needs a manifest written in JSON, called `manifest.json`, which gives essential information, e.g.,
 - Extension's name, version, and manifest's version
 - Main components of an extension (CS, BP/SW, ...)
 - Permissions of an extension (downloads, history, ...)
 - ...


Background – manifest.json -- example

```
{
  "name": "My Extension",
  "version": "versionString",
  "description": "A plain text description",
  "manifest_version": 3
  "permissions": ["downloads", "history"],
  "host_permissions": ["https://example.com/*"],
  "background": {
    "service_worker": ["service_worker.js"],
  },
  "content_scripts": [{
    "matches": ["<all_urls>"],
    "js": ["content_script.js"]
  }],
}
```

Outline

- Background
 - Malicious/vulnerable JavaScript
 - Browser extensions
- Investigating Security-Noteworthy Extensions (SNE)
 - SNE definition
 - Extension analysis & SNE detection
- Detecting Vulnerable Extensions
 - Threat models & examples
 - Static analysis of extensions (JavaScript) & examples
 - Case studies, results, and potential defense strategies

How Safe are Browser Extensions?

- Browser extensions provide **additional functionality**...
- ... so browser extensions need **additional & elevated privileges** compared to web pages
- **Browser extensions are an attractive target for attackers** 

→ Extensions can put their users' security & privacy at risk

- Contain **malware**

- Designed by malicious actors to harm victims
- E.g., propagate malware, steal users' credentials, track users

- **Violate the Chrome Web Store policies**

- E.g., deceive users, promote unlawful activities, lack a privacy policy

- Contain **vulnerabilities**

- Designed by well-intentioned developers... but contain some vulnerabilities
- E.g., can lead to user-sensitive data exfiltration

Did you know that...

- **350M users** installed **Security-Noteworthy Extensions** in the last 3 years?
- These **dangerous extensions** stay in the Chrome Web Store *for years*?
- **60%** of extensions have **never received a single update**?

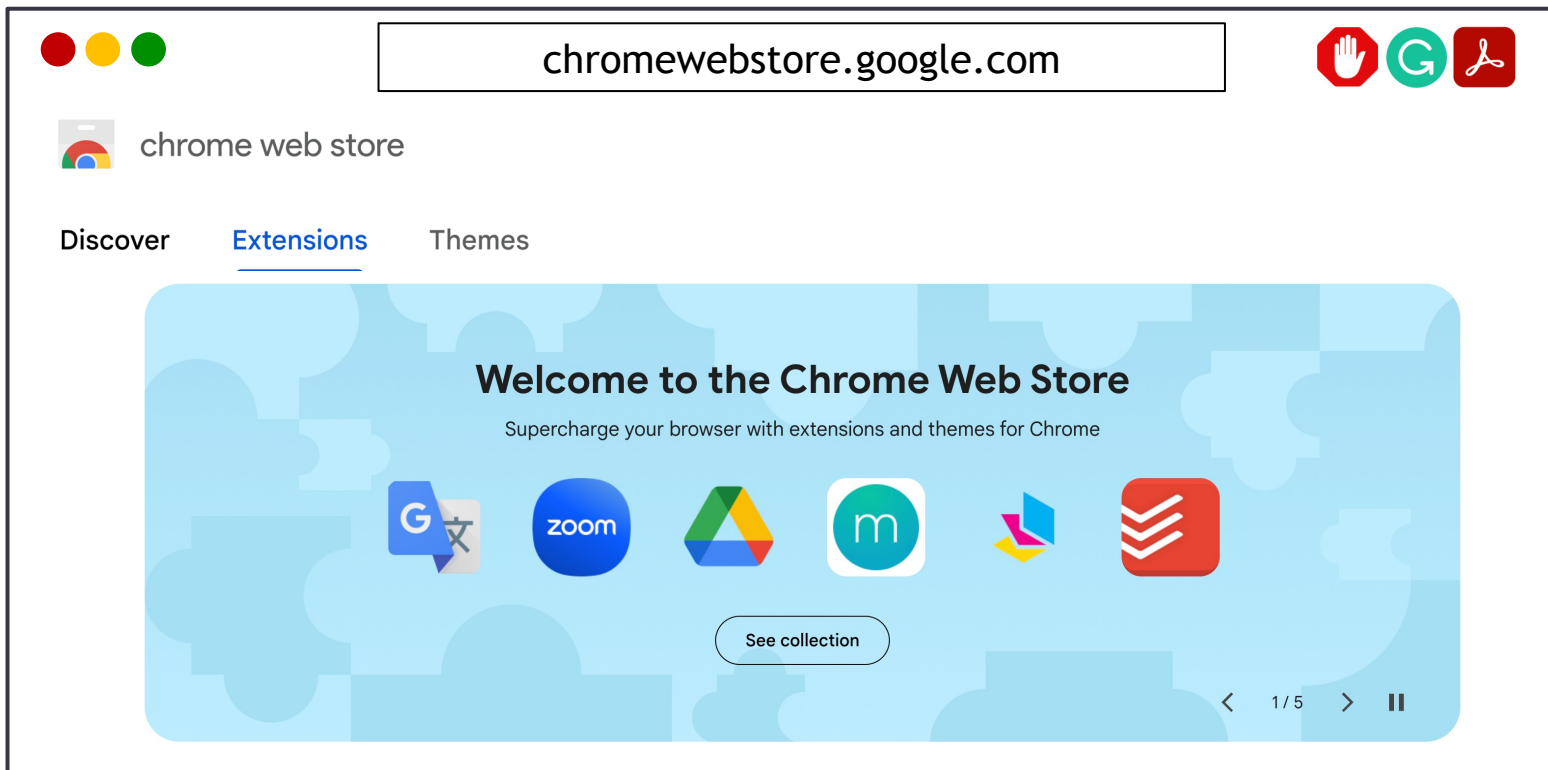


> What is in the Chrome Web Store?



In *ACM AsiaCCS 2024*. Sheryl Hsu, Manda Tran, and Aurore Fass

How to Install Extensions or SNE?



How to Install Extensions or SNE?

The image is a screenshot of a web browser window displaying the Chrome Web Store. The address bar shows 'chromewebstore.google.com'. The page title is 'chrome web store'. There are navigation tabs for 'Discover', 'Extensions', and 'Themes', with 'Extensions' being the active tab. A large blue banner is centered on the page, featuring a white text box that reads '>26k SNE (in the last 3 years)'. Below this text is a button labeled 'See collection'. The banner also includes a 'Welcome to the Chrome Web Store' message and a 'Subscribe to our newsletter' link. The background of the banner is light blue with abstract shapes and icons for Google, Zoom, and a list icon. The browser window has standard macOS window controls (red, yellow, green buttons) in the top left and navigation icons (stop, refresh, print) in the top right.

Browser Extension Collection: Chrome-Stats

chrome-stats.com

Compare and analyze Chrome extensions
All-in-one platform for competitor research, risk analysis, and growth tracking

127 862 Extensions 27 638 Themes

Chrome Web Store stats

Number of extension

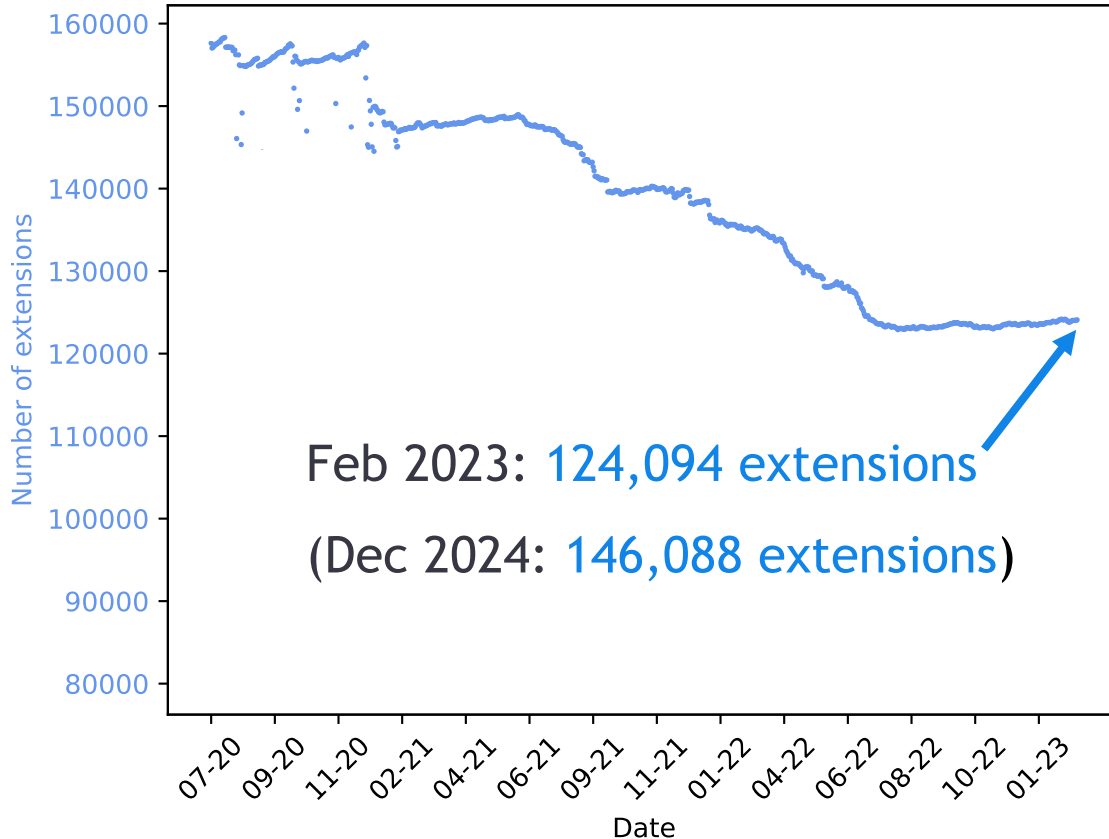
Extensions Themes

03-07 04-10 04-16 04-22 04-28 05-05 05-11 05-18 05-24 06-01 06-07 06-14 06-21 06-27 07-04 07-10 07-17 07-24 08-01 08-07 08-14 08-21 08-27 09-02

[Explore more Chrome extension statistics](#)

Chrome-Stats makes Chrome extension metrics more accessible to everyone, enable competitive analysis, identify bad actors, and help support the growth of good Chrome extensions.

Number of Extensions in the Chrome Web Store

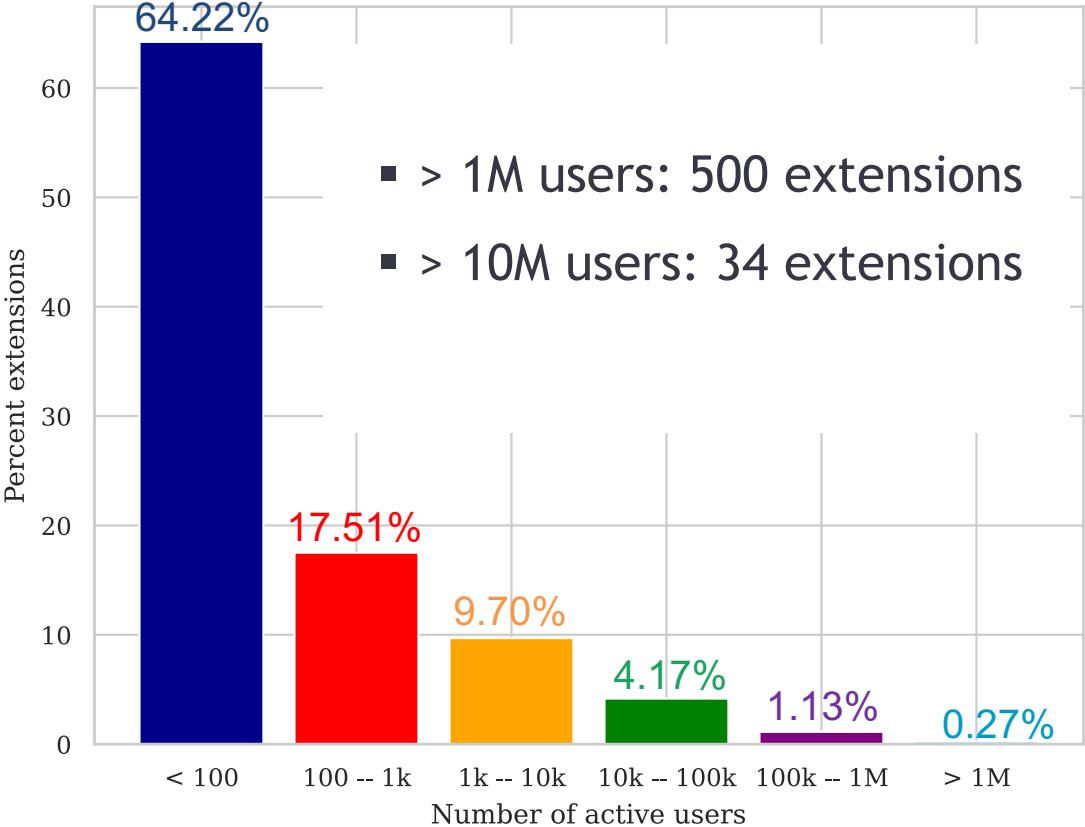


Every month:

- -3,775 extensions removed
- +2,687 extensions added

➤ **Analyses** on the CWS should be **run regularly**

Breakdown of Extension Users



The “**number of users**” on the CWS for a given extension corresponds to:

“the number of Chromes with the extension installed that are active and checking in to [their] update servers over the previous seven days only, not for all time. It is not equal to the sum of historic installs minus the sum of historic uninstalls”

~ Chrome Web Store Developer Support

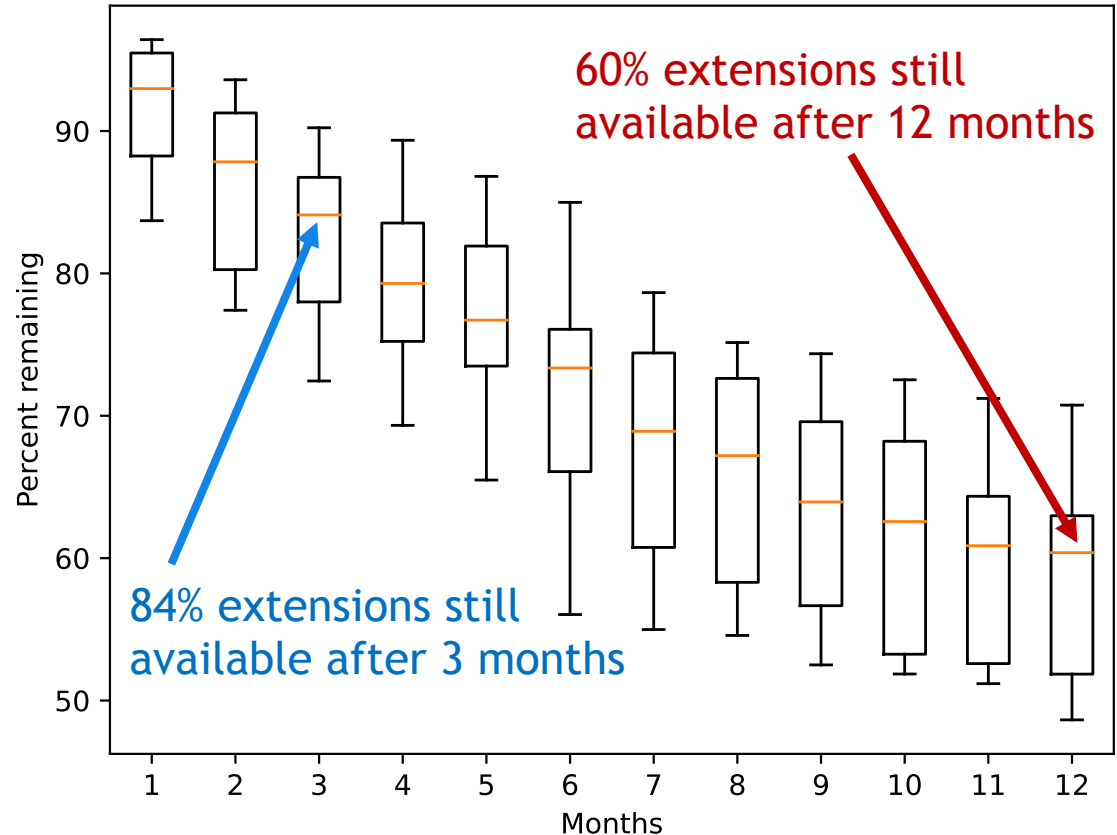
Life Cycle of Extensions

Methodology:

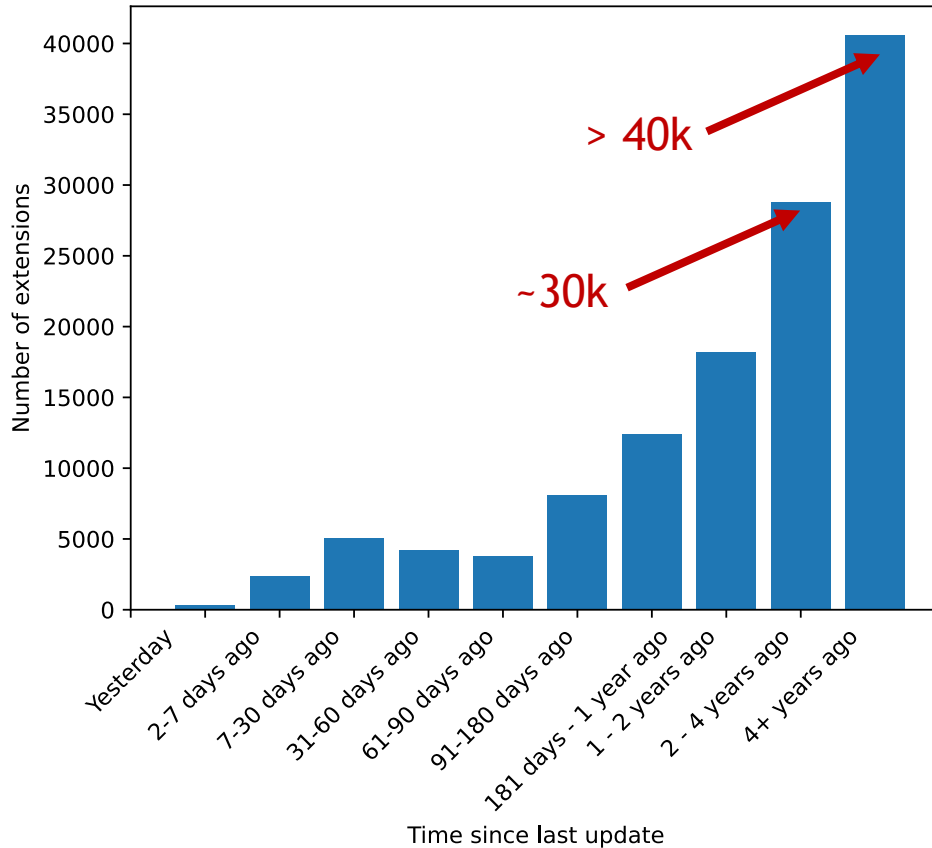
- Collected extensions added to the CWS in Jan–Dec 2021
- Computed the percentage of those extensions still in the CWS 1, 2, ..., 12 months later

➤ Extensions have a very short life cycle

➤ Analyses on the CWS should be run regularly



Extension Maintenance and Security



- Critical **lack of maintenance** in the CWS
- **60%** of the extensions have **never been updated**
- **Security & privacy implications**

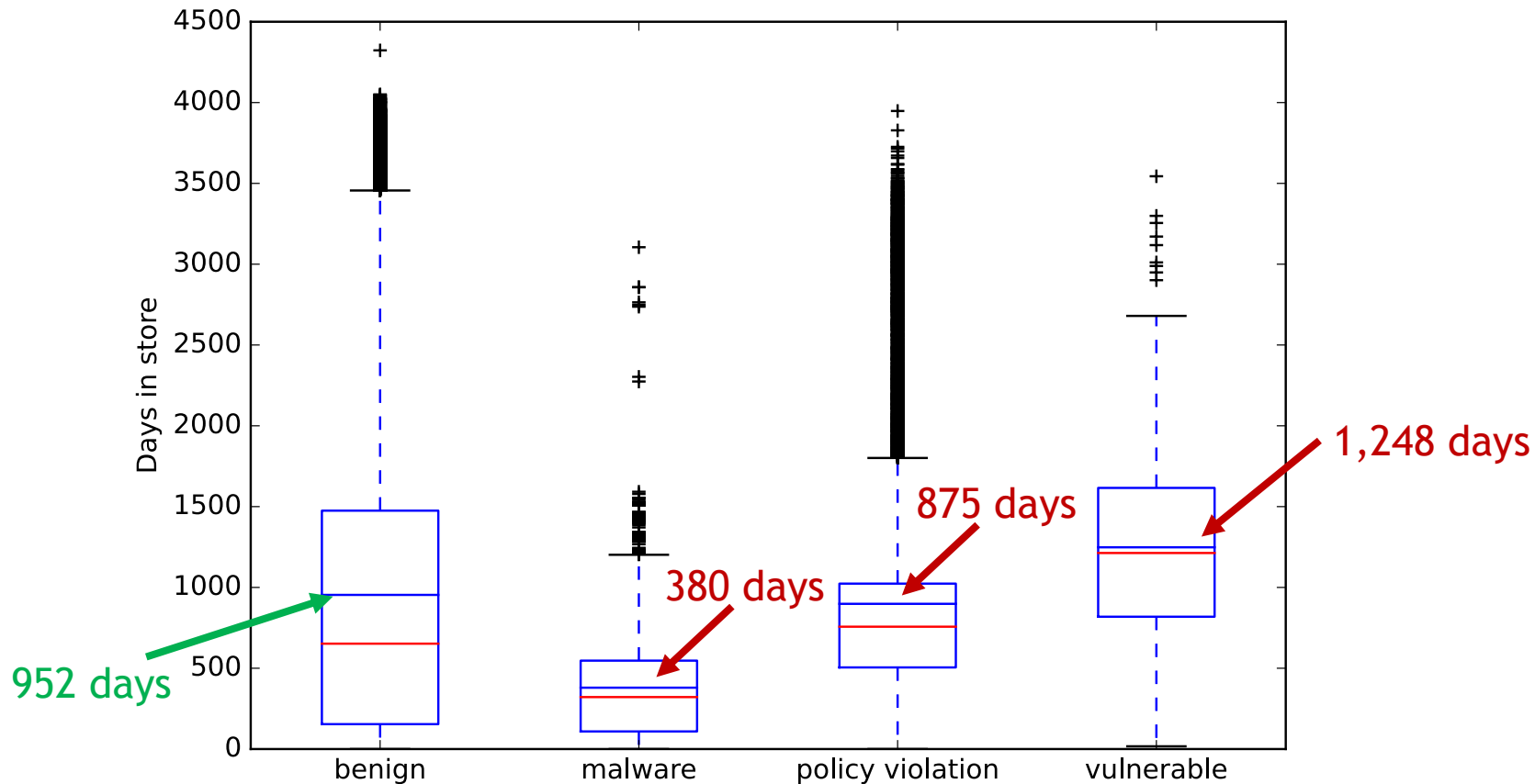
Malicious Extension Collection: Chrome-Stats

The screenshot shows the 'chrome-stats.com' website with an 'Advanced search' section. A red oval highlights the search filters: 'obsoleteReason' set to 'malware'. Below the search bar, there are buttons for 'Search', 'Export', 'Saved query', 'Visible columns', and '+ Add condition'. The search results show 10944 results on page 1 of 438. A table lists several extensions, all marked as 'malware'.

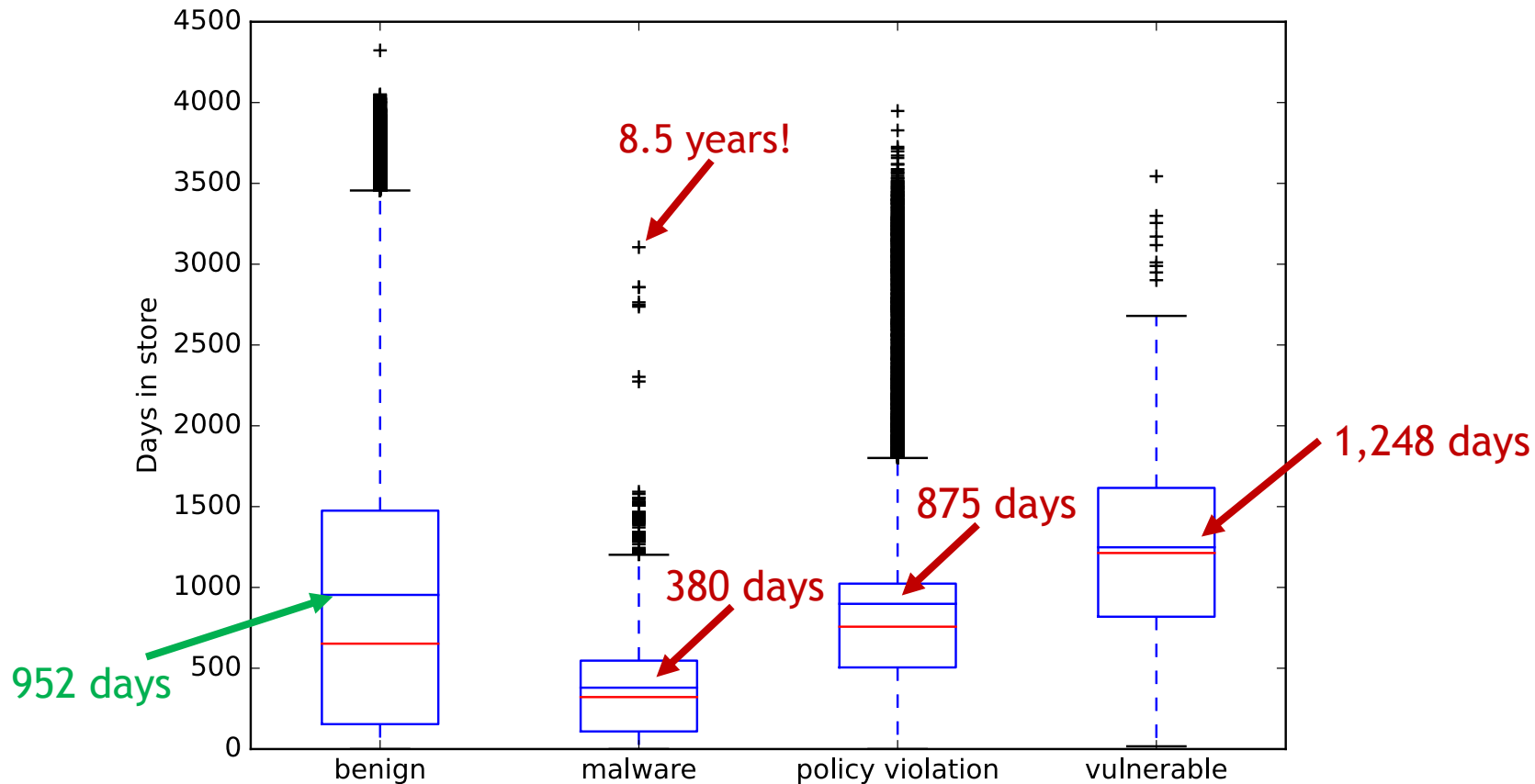
logo	name	userCount	author	ratingValue	ratingCount	obsoleteReason	lastUpdate	creationDate
	Video downloader for Instagram™	100000	https://instagram-downloader.instvid.site	4.27	30	malware	2024-03-07	2022-11-15
	Voice Aloud Reader for pc,windows and mac (Free Use)	11	https://voicealoudreaderforpc.blogspot.com	0.00	0	malware	2024-03-06	2024-03-06
	YTBlock - Adblock para Youtube	9000	YTAdblock	4.91	57	malware	2024-03-01	2024-02-09
	OVO Official	30	https://ovogame.pro	0.00	0	malware	2024-02-28	2024-02-28
	Snake	50000	https://snake.9834722.xyz	4.19	52	malware	2024-02-27	2021-10-04
	Settings for Chrome	600000	Chrome Settings	3.75	4	malware	2024-02-27	2022-06-24

Category	#Extensions Metadata collected	#Extensions Code collected	When collected
SNE	26,014	16,377	Before May 1, 2023
- Malware-containing	10,426	6,587	Before May 1, 2023
- Policy-violating	15,404	9,638	Before May 1, 2023
- Vulnerable [1]	184	152	March 16, 2021
Benign extensions	226,762	92,482	Before May 1, 2023

Number of Days in the CWS



Number of Days in the CWS



Number of Days in the CWS

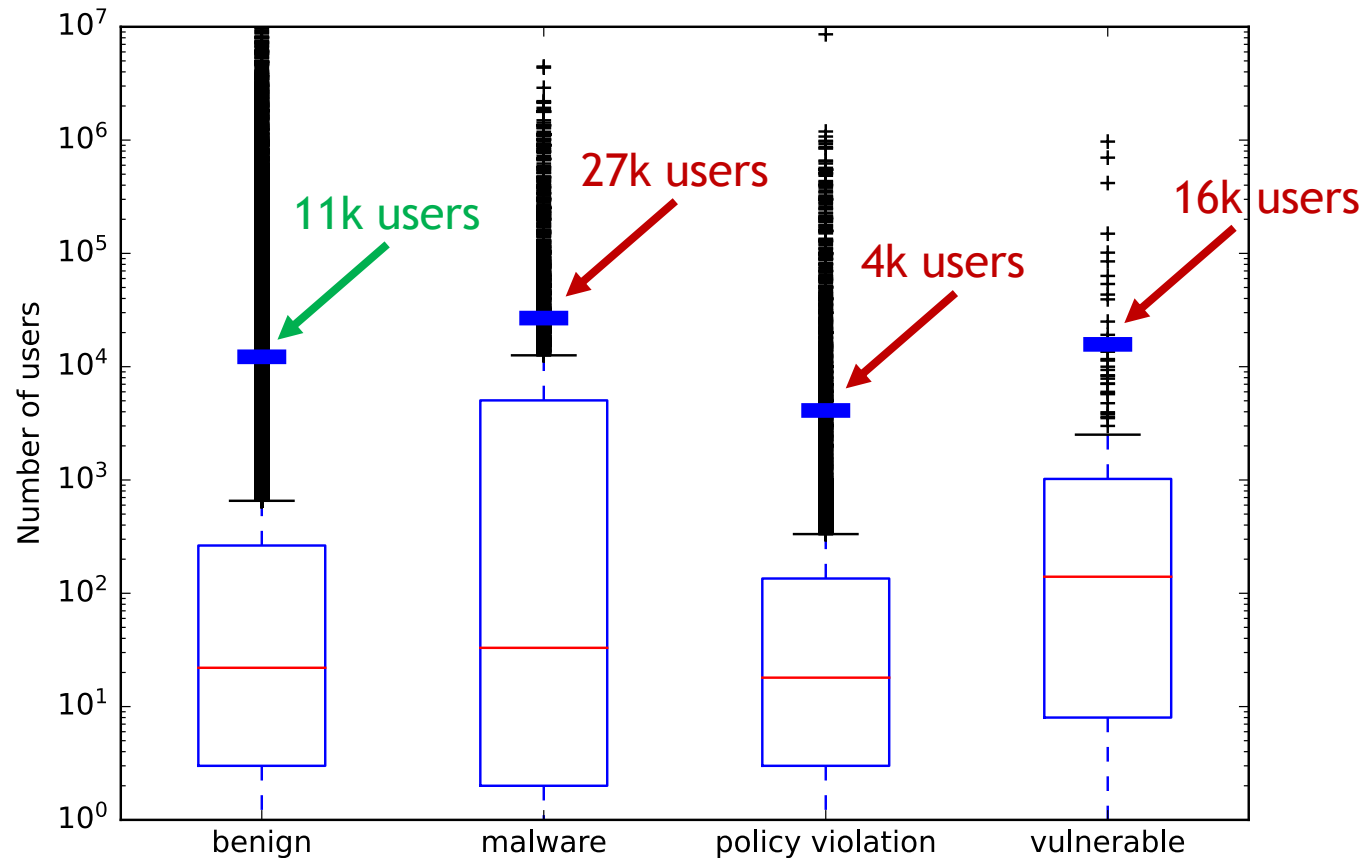


➤ SNE put the security & privacy of Web users *at risk for years*

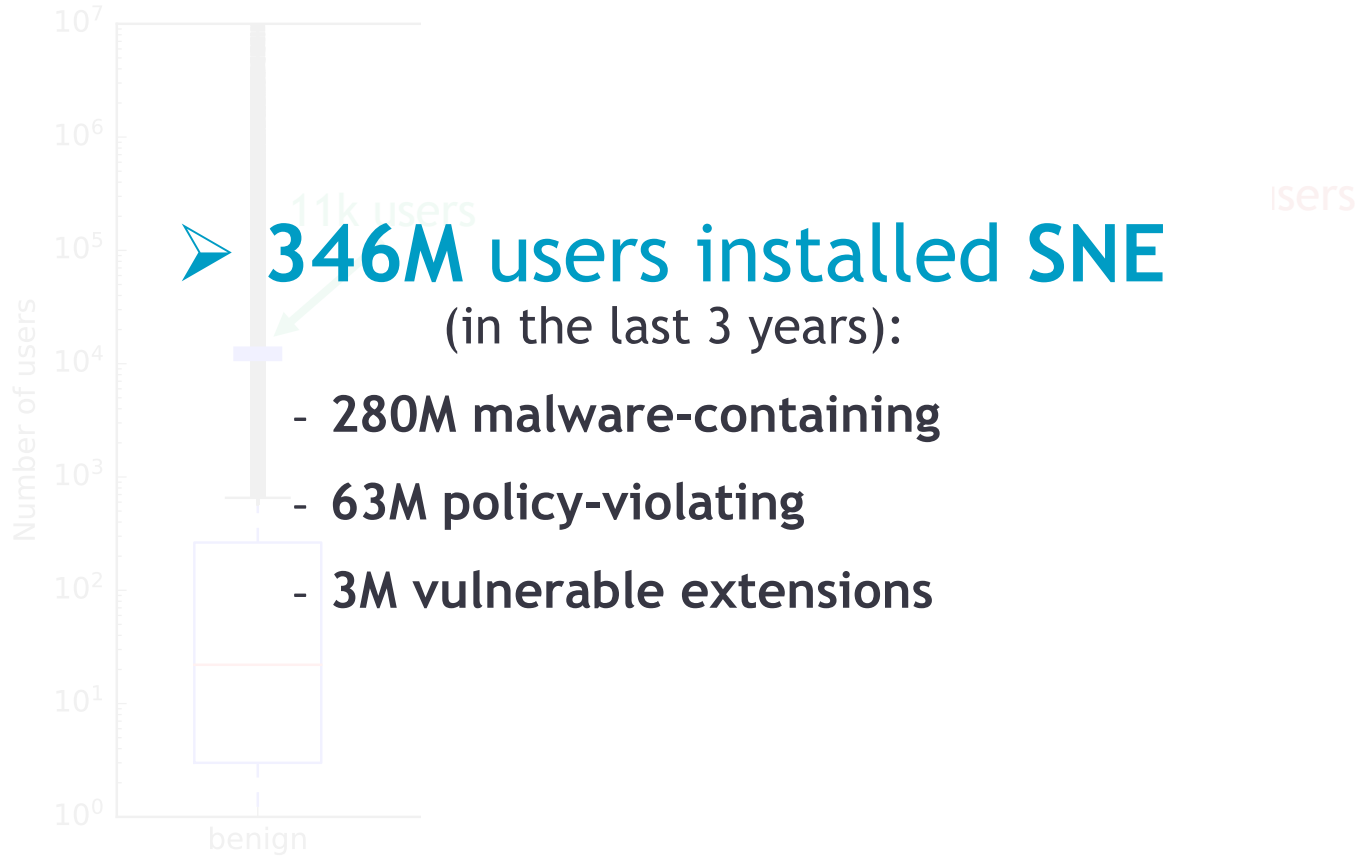
,248 days

952 days

Number of Users



Number of Users



- Source-code comparison across extensions (*ssdeep* fuzzy hash)
- Clustering similar extensions together (i.e., 100% *ssdeep* overlap)
- 3,270 clusters with [2; 1,397] extensions (20,822 extensions clustered)

- 3,270 clusters:
 - 2,296 clusters contain just benign extensions
 - 321 clusters only SNE
 - 14 clusters with > 100 SNE and 2 with > 863 SNE each
 - Analyzing extensions for similarities could enable to detect SNE
 - 653 clusters of benign (5,552 extensions) and SNE (5,126)
 - Extensions in a cluster with SNE should be flagged for more analyses

Media Coverage

Forbes

FORBES > INNOVATION > CYBERSECURITY

280 Million Google Chrome Users Installed Dangerous Extensions, Study Says

Davey Winder Senior Contributor @
Davey Winder is a veteran cybersecurity writer, hacker and analyst.

Follow

Jun 24, 2024, 06:57am EDT



How safe are Google Chrome extensions? SOPA IMAGES/LIGHTROCKET VIA GETTY IMAGES

The Register



Risk of installing dodgy extensions from Chrome store way worse than Google's letting on, study suggests

All depends on how you count it – Chocolate Factory claims 1% fail rate

[Thomas Claburn](#)

Sun 23 Jun 2024 // 10:36 UTC

ADGUARD

A⁵

Subscribe to news

Search blog

AdGuard > Blog > Google is failing miserably at weeding out bad extensions, new research indicates

Google is failing miserably at weeding out bad extensions, new research indicates

July 5, 2024 · 7 min read

TECHSPOT

TRENDING FEATURES REVIEWS THE BEST DOWNLOADS PRODUCT FINDER FORUMS

SECURITY THE WEB MALWARE CHROME

Researchers say 280 million people have installed malware-infected Chrome extensions in the last 3 years

Google claims less than 1% of all installs include malware

By Rob Thubron June 24, 2024 at 11:39 AM



Outline

- Background
 - Malicious/vulnerable JavaScript
 - Browser extensions
- Investigating Security-Noteworthy Extensions (SNE)
 - SNE definition
 - Extension analysis & SNE detection
- Detecting Vulnerable Extensions
 - Threat models & examples
 - Static analysis of extensions (JavaScript) & examples
 - Case studies, results, and potential defense strategies

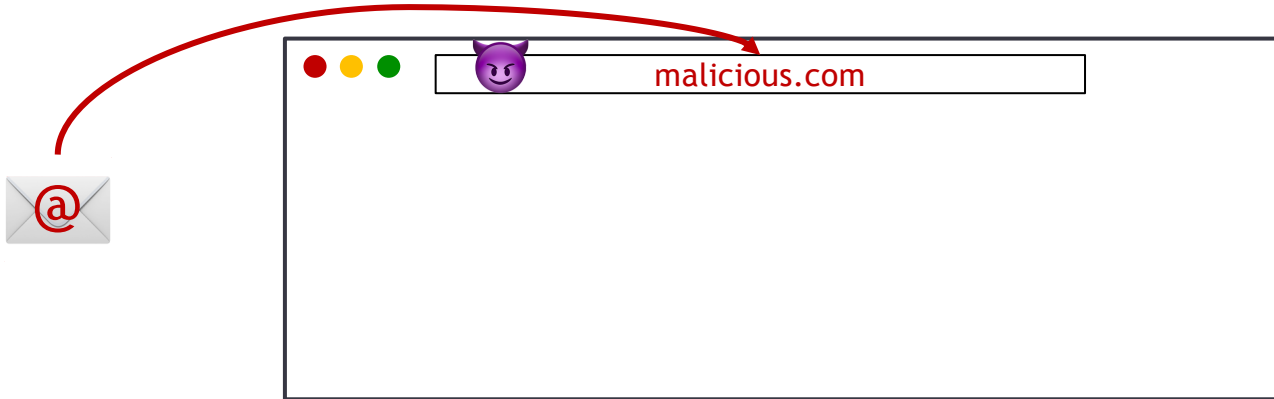
Analysis of Vulnerable Extensions: Web Attacker

Challenging to detect due to their inherently benign intent (*benign-but-buggy*)



Analysis of Vulnerable Extensions: Web Attacker

Challenging to detect due to their inherently benign intent (*benign-but-buggy*)



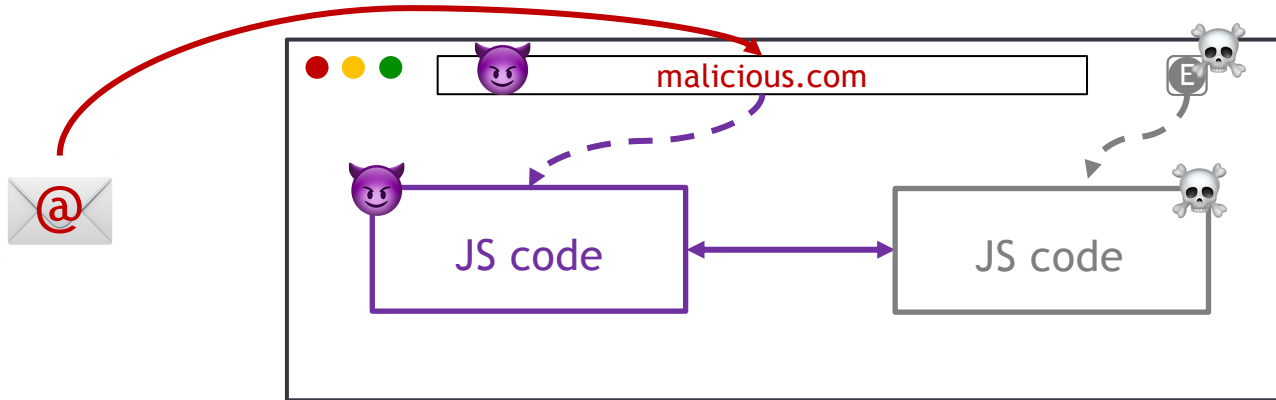
Analysis of Vulnerable Extensions: Web Attacker

Challenging to detect due to their inherently benign intent (*benign-but-buggy*)



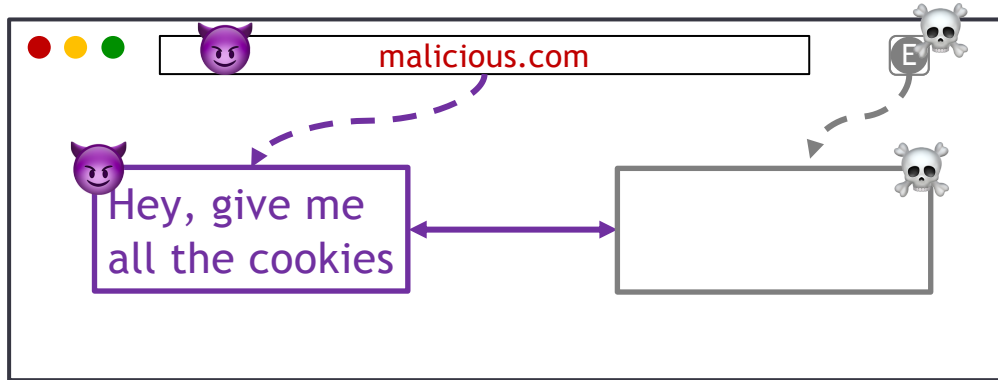
Analysis of Vulnerable Extensions: Web Attacker

Challenging to detect due to their inherently benign intent (*benign-but-buggy*)



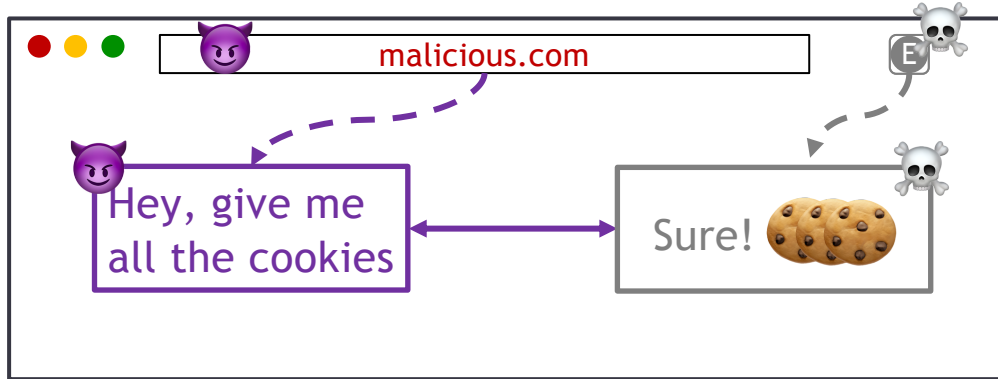
Analysis of Vulnerable Extensions: Web Attacker

Challenging to detect due to their inherently benign intent (*benign-but-buggy*)



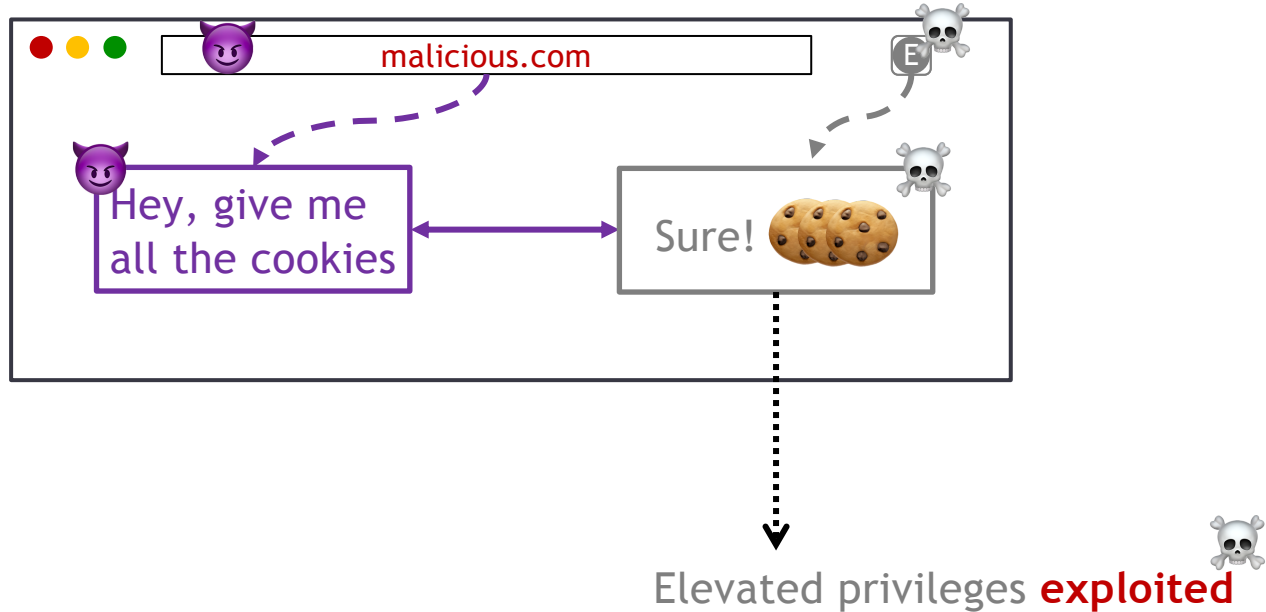
Analysis of Vulnerable Extensions: Web Attacker

Challenging to detect due to their inherently benign intent (*benign-but-buggy*)



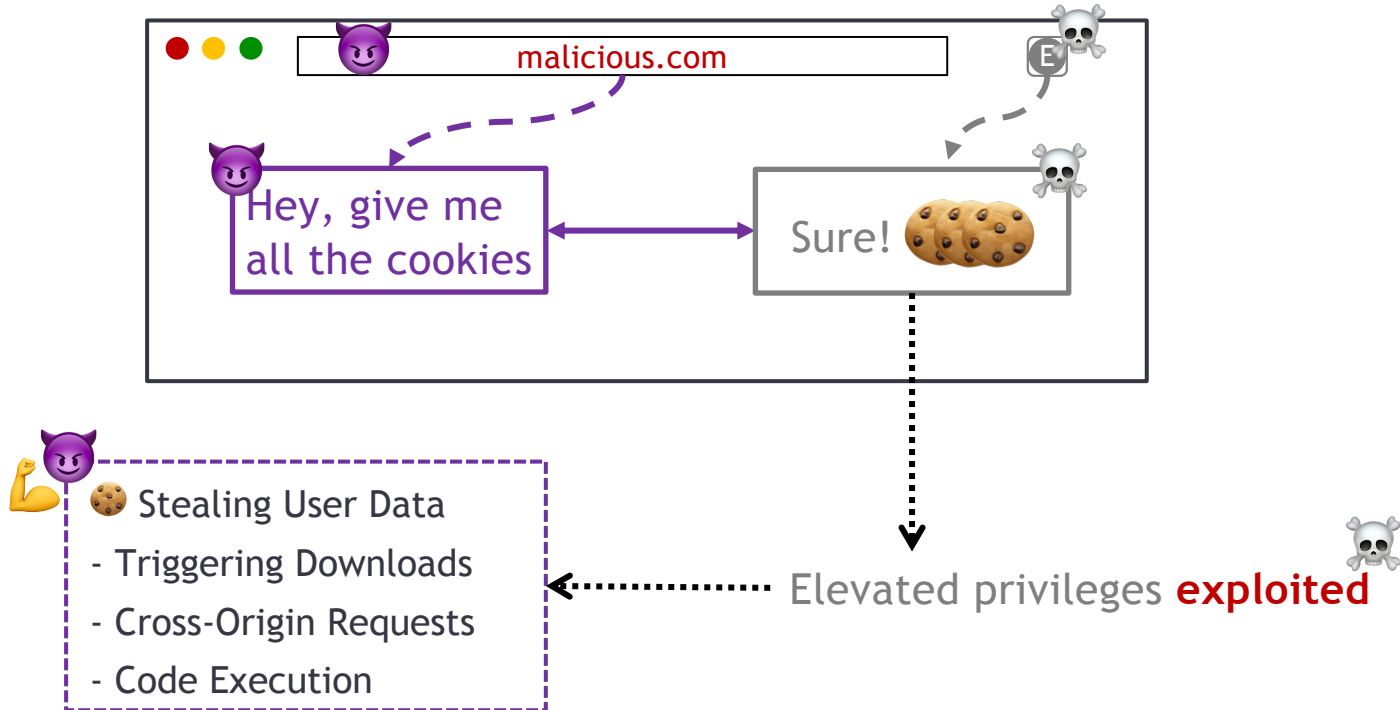
Analysis of Vulnerable Extensions: **Web Attacker**

Challenging to detect due to their inherently benign intent (*benign-but-buggy*)



Analysis of Vulnerable Extensions: **Web Attacker**

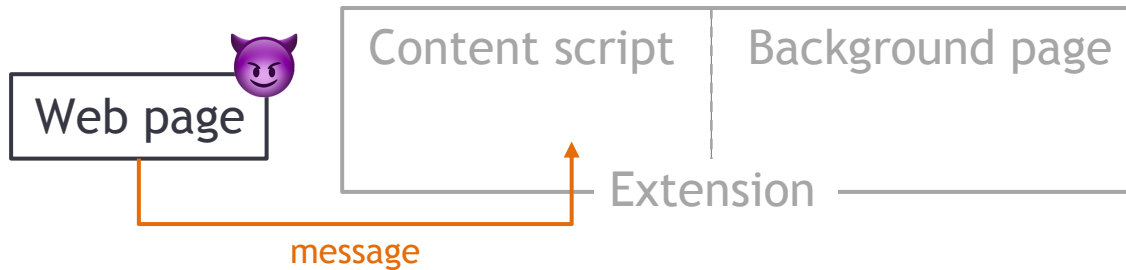
Challenging to detect due to their inherently benign intent (*benign-but-buggy*)



- To send messages:
 - `otherWindow.postMessage(message, targetOrigin)`
- To receive messages:
 - With an *event handler* (`addEventListener` or `onmessage`)
- `/!\` The 2 origins must trust each other → verify the origin before processing a message

Simplified Example of a Vulnerability

```
// Content script code  
window.addEventListener("message", function(event) {  
  
  
  
  
  
  
  
  
  
})
```

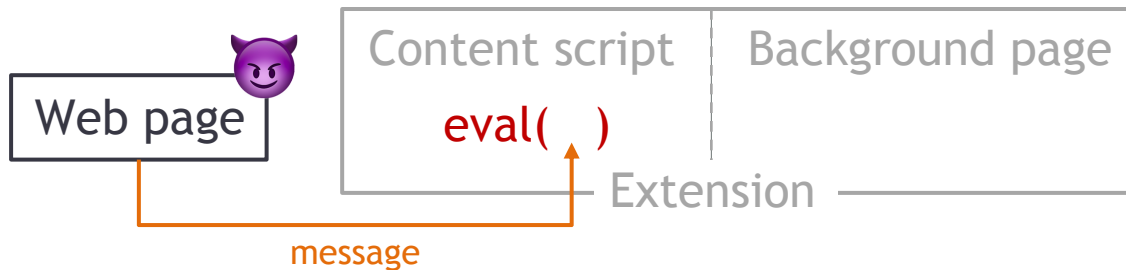


Simplified Example of a Vulnerability

```
// Content script code
window.addEventListener("message", function(event) {

    eval(event.data);

})
```



Simplified Example of a Vulnerability

```
// Content script code  
window.addEventListener("message", function(event) {  
    eval(event.data);  
})
```

```
// Attacker code = from the targeted web page  
postMessage("alert(1)", "*")
```

malicious payload

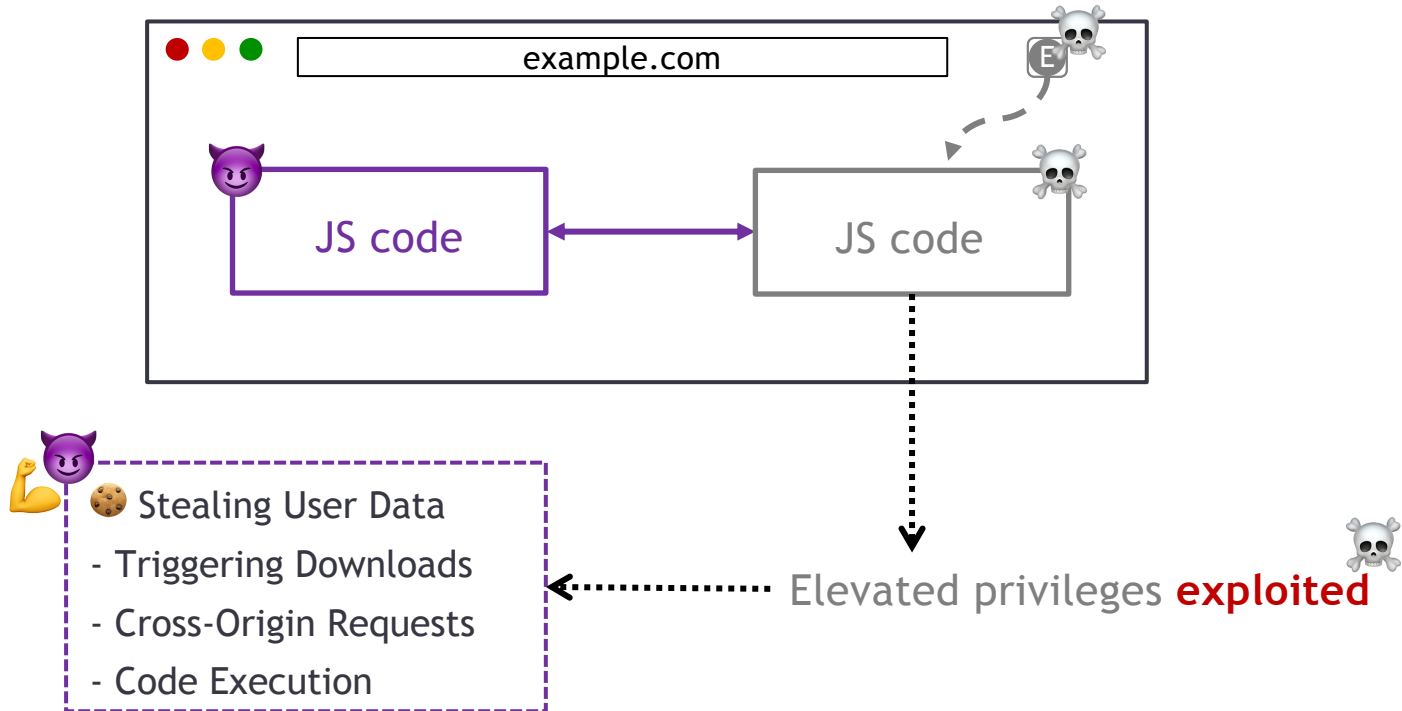
developer.chrome.com indique

1

OK

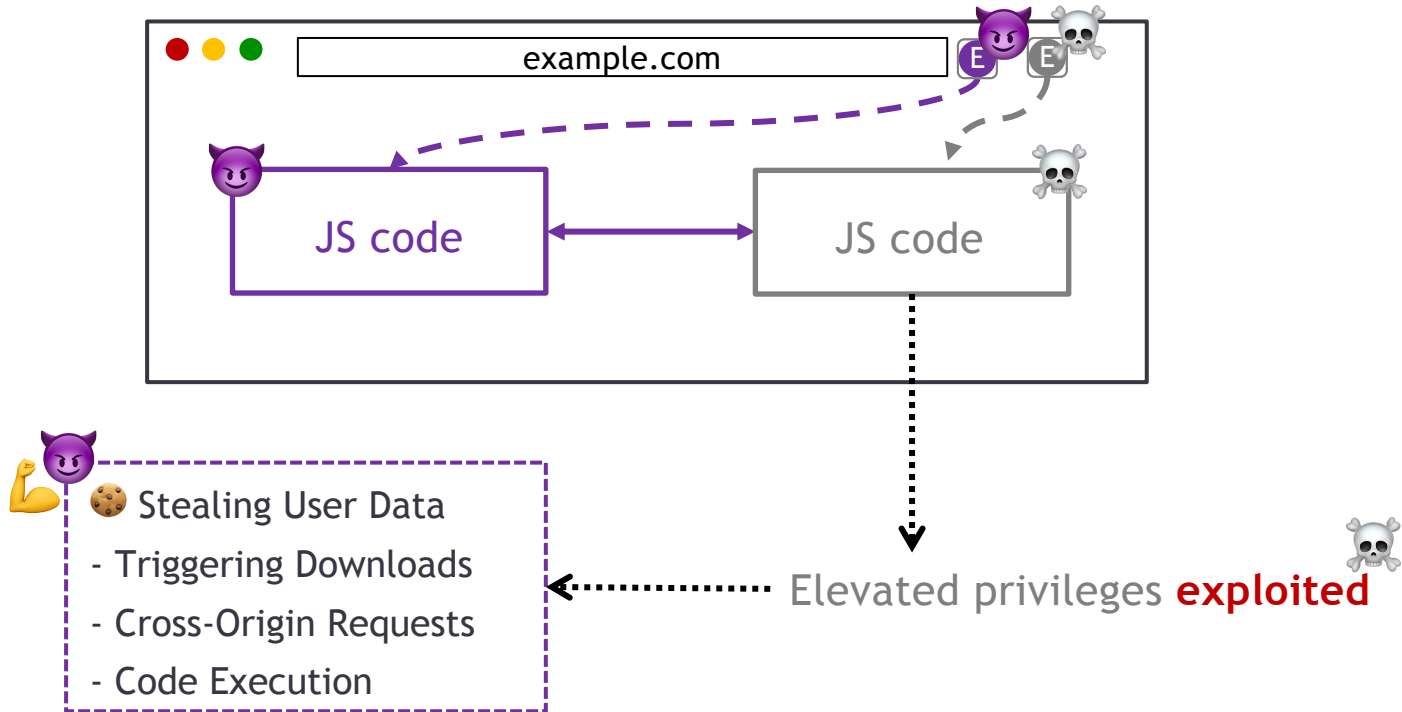
Analysis of Vulnerable Extensions: Confused Deputy

Challenging to detect due to their inherently benign intent (*benign-but-buggy*)



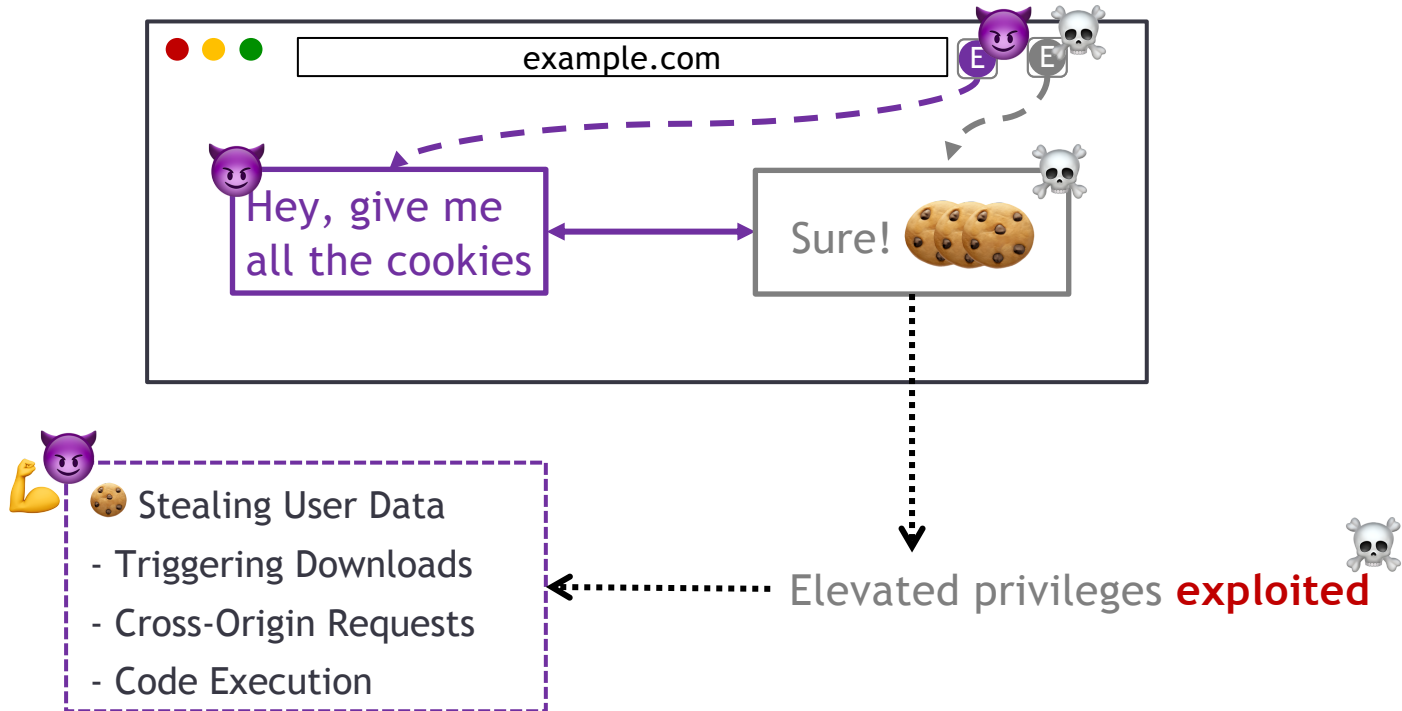
Analysis of Vulnerable Extensions: Confused Deputy

Challenging to detect due to their inherently benign intent (*benign-but-buggy*)



Analysis of Vulnerable Extensions: Confused Deputy

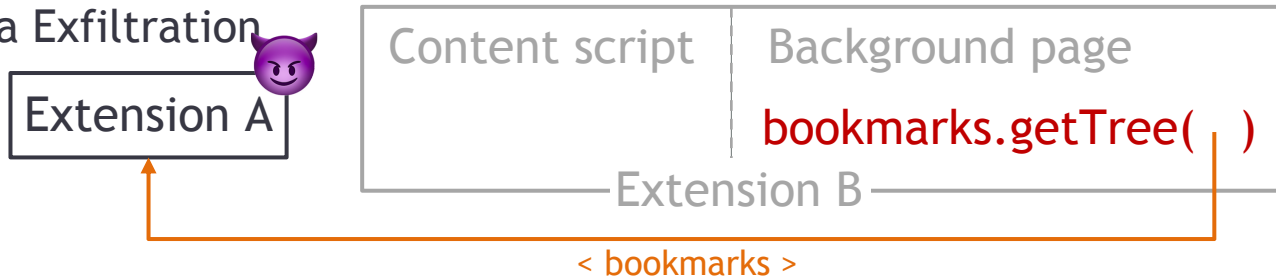
Challenging to detect due to their inherently benign intent (*benign-but-buggy*)



Exploiting Vulnerable Extensions: Confused Deputy

```
// Background page code of Extension B
chrome.runtime.onMessageExternal.addListener(
  function(request, sender, sendResponse) {
    chrome.bookmarks.getTree(function(data) {
      sendResponse(data);
    });
  });
```

 Data Exfiltration 



Detecting Vulnerable Extensions



Fass et al.
CCS 2021

> **DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions**
In ACM CCS 2021. Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock

DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale
Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock
CCS '21, October 2021, New York, NY, USA. ACM.

Abstract
Browser extensions are a popular way to customize and extend the functionality of web browsers. However, they are also a common source of security vulnerabilities. In this paper, we present DOUBLEX, a static analysis tool that detects vulnerable data flows in browser extensions. DOUBLEX is designed to be scalable and accurate, and is able to analyze millions of extensions. We evaluate DOUBLEX on a large dataset of browser extensions and show that it is able to detect a wide range of vulnerabilities, including information leaks, data tampering, and denial of service attacks. DOUBLEX is available as an open-source tool and can be used by researchers and practitioners alike.

1 Introduction
Browser extensions are a popular way to customize and extend the functionality of web browsers. However, they are also a common source of security vulnerabilities. In this paper, we present DOUBLEX, a static analysis tool that detects vulnerable data flows in browser extensions. DOUBLEX is designed to be scalable and accurate, and is able to analyze millions of extensions. We evaluate DOUBLEX on a large dataset of browser extensions and show that it is able to detect a wide range of vulnerabilities, including information leaks, data tampering, and denial of service attacks. DOUBLEX is available as an open-source tool and can be used by researchers and practitioners alike.

CCS Concepts
Security and privacy → Software security protection; Software security protection → Software security protection; Software security protection → Software security protection.

Keywords
Browser extensions; Static analysis; Security vulnerabilities; Information leaks; Data tampering; Denial of service attacks.

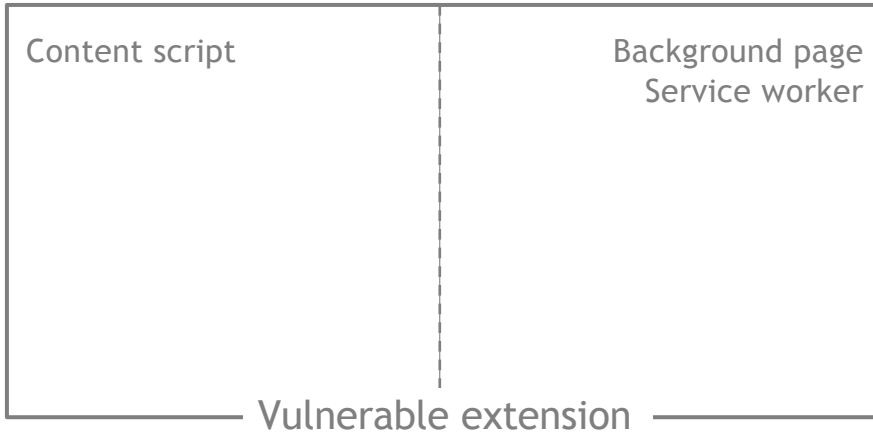
Detecting Vulnerable Extensions



Fass et al.
CCS 2021

DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale
Aurore Fass, Dolie Francis Somé, Michael Backes, and Ben Stock
1 Introduction

> **DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions**
In ACM CCS 2021. Aurore Fass, Dolie Francis Somé, Michael Backes, and Ben Stock



Detecting Vulnerable Extensions



Fass et al.
CCS 2021

DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale
Aurore Fass, Dolie Francis Somé, Michael Backes, and Ben Stock
1 Introduction

> **DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions**
In ACM CCS 2021. Aurore Fass, Dolie Francis Somé, Michael Backes, and Ben Stock



Malicious web page

Content script

Background page
Service worker

Vulnerable extension

Detecting Vulnerable Extensions



Fass et al.
CCS 2021

DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale
Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock
CCS 2021

Abstract
Browser extensions are a popular way to customize web browsing. However, they are often vulnerable to attacks. This paper presents DOUBLEX, a static analysis tool that detects vulnerable data flows in browser extensions. DOUBLEX is based on a novel abstraction of JavaScript code that allows for the detection of data flows that are vulnerable to attacks. DOUBLEX is able to detect vulnerable data flows in a large number of browser extensions, including those that are used by millions of users. DOUBLEX is able to detect vulnerable data flows in a large number of browser extensions, including those that are used by millions of users.

1 Introduction
Browser extensions are a popular way to customize web browsing. However, they are often vulnerable to attacks. This paper presents DOUBLEX, a static analysis tool that detects vulnerable data flows in browser extensions. DOUBLEX is based on a novel abstraction of JavaScript code that allows for the detection of data flows that are vulnerable to attacks. DOUBLEX is able to detect vulnerable data flows in a large number of browser extensions, including those that are used by millions of users.

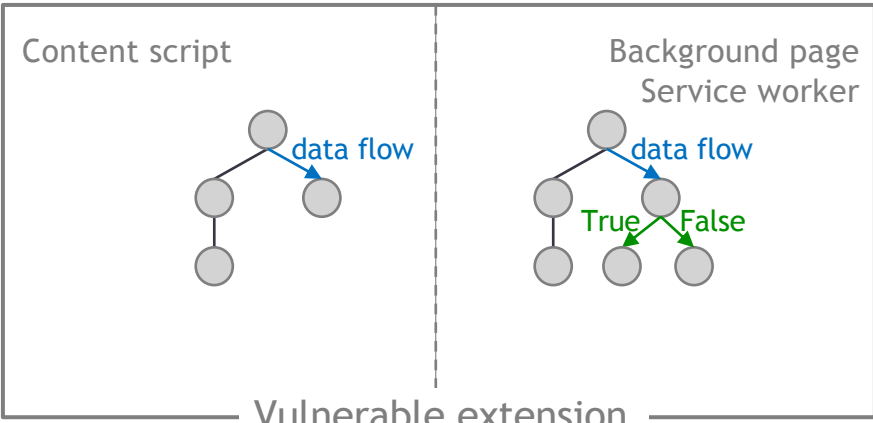
CCS keywords
Security and privacy, Web applications, Browser extensions

Keywords
Static analysis, JavaScript, Browser extensions, Vulnerability, Data flow analysis

> DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions

In ACM CCS 2021. Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock

 Malicious web page



Per-component JS code abstraction

- AST (Abstract Syntax Tree)
- Control flow
- Data flow
- Pointer analysis

Detecting Vulnerable Extensions



Fass et al.
CCS 2021

> DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions

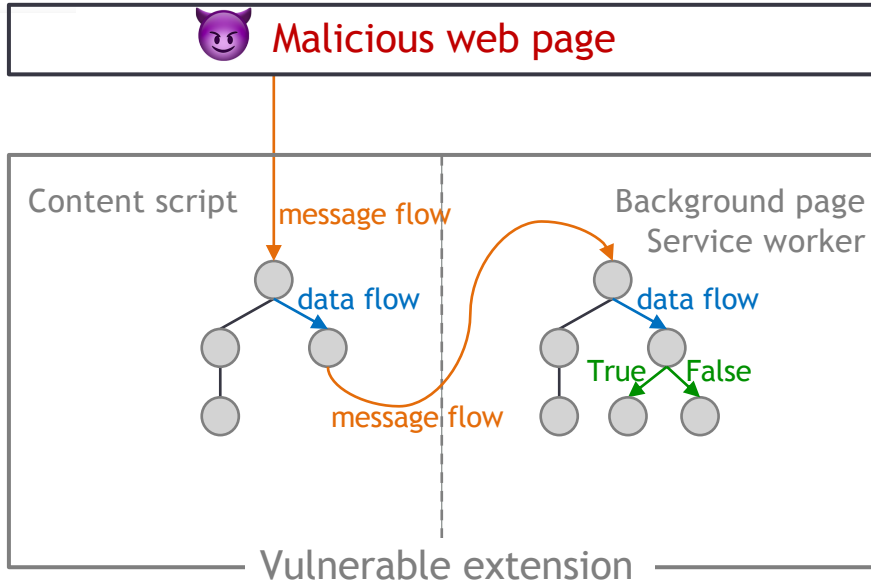
In ACM CCS 2021. Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock

DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale
Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock
CCS 2021

Abstract
Browser extensions are a popular way to customize the user experience of web browsers. However, they are also a common source of security vulnerabilities. In this paper, we present DOUBLEX, a static analysis tool for detecting vulnerable data flows in browser extensions. DOUBLEX is designed to be scalable and accurate, and is able to detect a wide range of vulnerabilities, including data leaks, data corruption, and data loss. We evaluate DOUBLEX on a large dataset of browser extensions and show that it is able to detect a high number of vulnerabilities. We also show that DOUBLEX is able to detect vulnerabilities that other tools are unable to detect.

1 Introduction
Browser extensions are a popular way to customize the user experience of web browsers. However, they are also a common source of security vulnerabilities. In this paper, we present DOUBLEX, a static analysis tool for detecting vulnerable data flows in browser extensions. DOUBLEX is designed to be scalable and accurate, and is able to detect a wide range of vulnerabilities, including data leaks, data corruption, and data loss. We evaluate DOUBLEX on a large dataset of browser extensions and show that it is able to detect a high number of vulnerabilities. We also show that DOUBLEX is able to detect vulnerabilities that other tools are unable to detect.

CCS keywords
Security and privacy, Web applications, Browser extensions, Static analysis, Vulnerability detection, Data flows, Message flows, Pointer analysis, Control flow, Data flow, Message interactions, Extension dependence graph (EDG).



Per-component JS code abstraction

- AST (Abstract Syntax Tree)
- Control flow
- Data flow
- Pointer analysis

Extension Dependence Graph (EDG)

- Message interactions

Detecting Vulnerable Extensions



Fass et al.
CCS 2021

> DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions

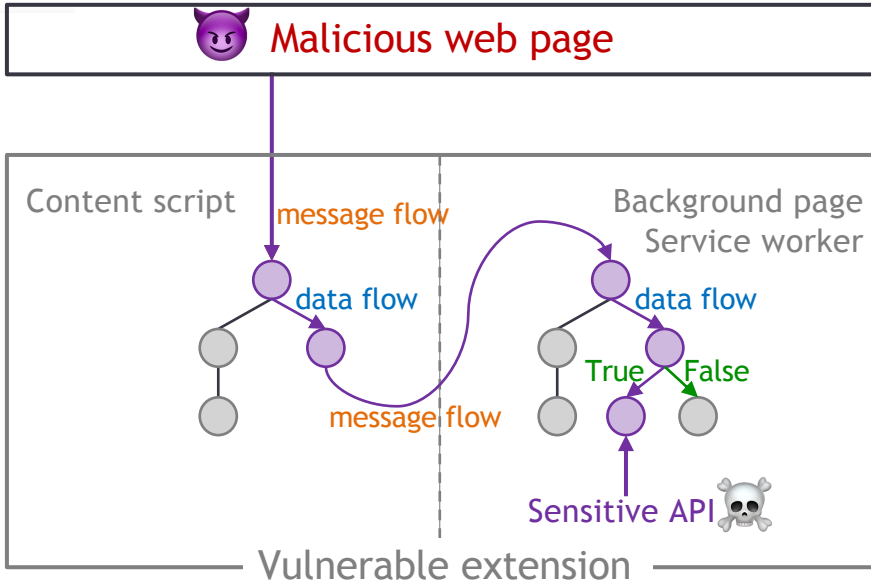
In ACM CCS 2021. Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock

DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale
Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock
CCS 2021

Abstract
Browser extensions are a popular way to extend the functionality of web browsers. However, they are often written by third parties and can be vulnerable to attacks. This paper presents DOUBLEX, a static analysis tool that detects vulnerable data flows in browser extensions. DOUBLEX is based on a novel abstraction of JavaScript code that captures the flow of data between different components of the browser. This abstraction is used to detect data flows that are vulnerable to attacks, such as those that leak sensitive information or perform unauthorized actions. DOUBLEX is implemented as a plugin for the JavaScript engine V8 and is able to analyze millions of browser extensions. The results of our analysis show that DOUBLEX is able to detect a large number of vulnerable data flows in browser extensions, including those that are not detected by existing tools. This work is the first to provide a static analysis of browser extensions that is able to detect vulnerable data flows at scale.

1 Introduction
Browser extensions are a popular way to extend the functionality of web browsers. However, they are often written by third parties and can be vulnerable to attacks. This paper presents DOUBLEX, a static analysis tool that detects vulnerable data flows in browser extensions. DOUBLEX is based on a novel abstraction of JavaScript code that captures the flow of data between different components of the browser. This abstraction is used to detect data flows that are vulnerable to attacks, such as those that leak sensitive information or perform unauthorized actions. DOUBLEX is implemented as a plugin for the JavaScript engine V8 and is able to analyze millions of browser extensions. The results of our analysis show that DOUBLEX is able to detect a large number of vulnerable data flows in browser extensions, including those that are not detected by existing tools. This work is the first to provide a static analysis of browser extensions that is able to detect vulnerable data flows at scale.

CCS keywords
Web browser extensions; Static analysis; Vulnerable data flows; Information security.



Per-component JS code abstraction

- AST (Abstract Syntax Tree)
- Control flow
- Data flow
- Pointer analysis

Extension Dependence Graph (EDG)

- Message interactions

Suspicious data flow tracking

- Detects any path between an attacker & sensitive APIs

Detecting Vulnerable Extensions



Fass et al.
CCS 2021

DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale
Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock
CCS 2021

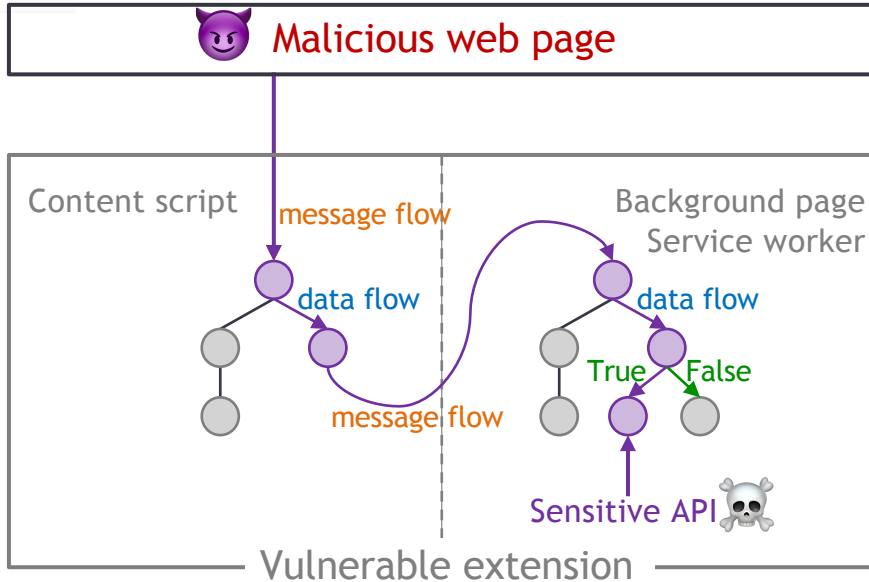
Abstract
Browser extensions are a popular way to enhance browser functionality. However, they are often vulnerable to attacks. This paper presents DOUBLEX, a static analysis tool for detecting vulnerable data flows in browser extensions. DOUBLEX is designed to be scalable and accurate. It is able to detect a wide range of vulnerabilities, including information leaks, data tampering, and unauthorized access to sensitive data. DOUBLEX is implemented as a plugin for the JavaScript engine. It is able to analyze extensions at scale, and it is able to detect vulnerabilities in extensions that are not detected by existing tools. DOUBLEX is able to detect vulnerabilities in extensions that are not detected by existing tools. DOUBLEX is able to detect vulnerabilities in extensions that are not detected by existing tools.

Introduction
Browser extensions are a popular way to enhance browser functionality. However, they are often vulnerable to attacks. This paper presents DOUBLEX, a static analysis tool for detecting vulnerable data flows in browser extensions. DOUBLEX is designed to be scalable and accurate. It is able to detect a wide range of vulnerabilities, including information leaks, data tampering, and unauthorized access to sensitive data. DOUBLEX is implemented as a plugin for the JavaScript engine. It is able to analyze extensions at scale, and it is able to detect vulnerabilities in extensions that are not detected by existing tools. DOUBLEX is able to detect vulnerabilities in extensions that are not detected by existing tools.

Conclusion
DOUBLEX is a static analysis tool for detecting vulnerable data flows in browser extensions. DOUBLEX is designed to be scalable and accurate. It is able to detect a wide range of vulnerabilities, including information leaks, data tampering, and unauthorized access to sensitive data. DOUBLEX is implemented as a plugin for the JavaScript engine. It is able to analyze extensions at scale, and it is able to detect vulnerabilities in extensions that are not detected by existing tools. DOUBLEX is able to detect vulnerabilities in extensions that are not detected by existing tools.

> DOUBLEX: Statically Detecting Vulnerable Data Flows in Browser Extensions

In ACM CCS 2021. Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock



Per-component JS code abstraction

- AST (Abstract Syntax Tree)
- Control flow
- Data flow
- Pointer analysis

Extension Dependence Graph (EDG)

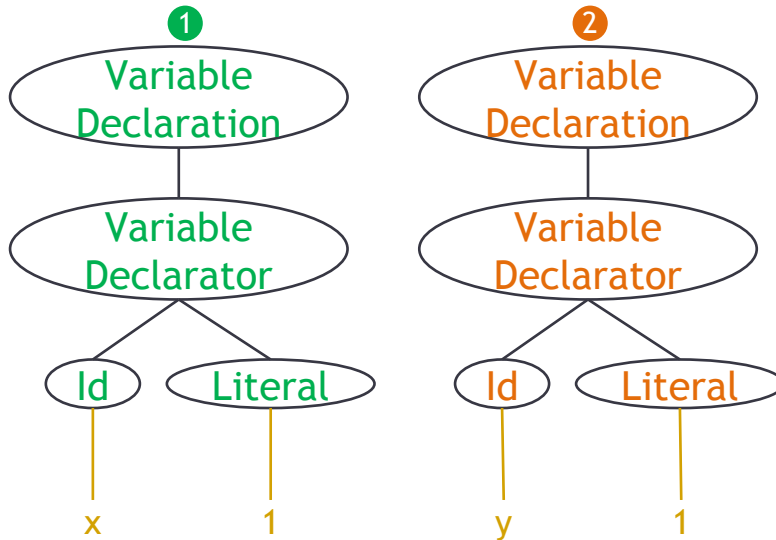
- > Message interactions

Suspicious data flow tracking

- > Detects any path between an attacker & sensitive APIs

Abstract Syntax Tree

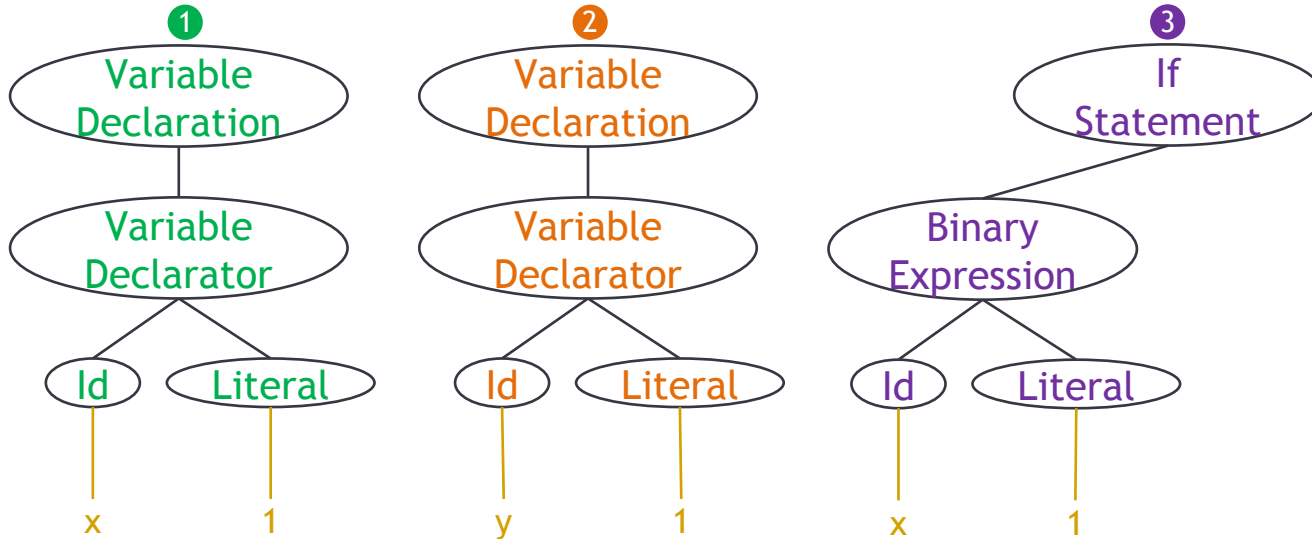
- AST: nesting of programming constructs



① `var x = 1;`
② `var y = 1;`

Abstract Syntax Tree

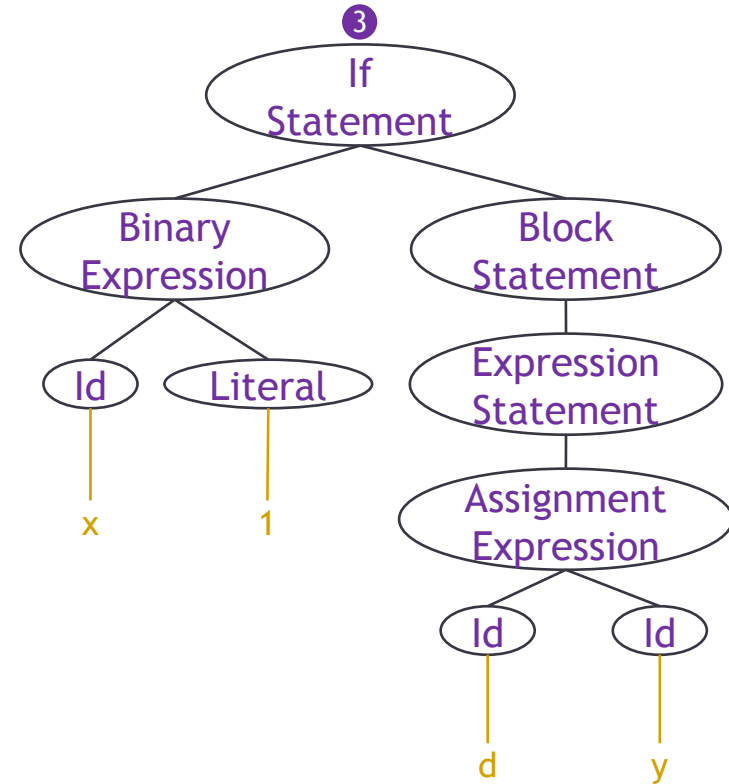
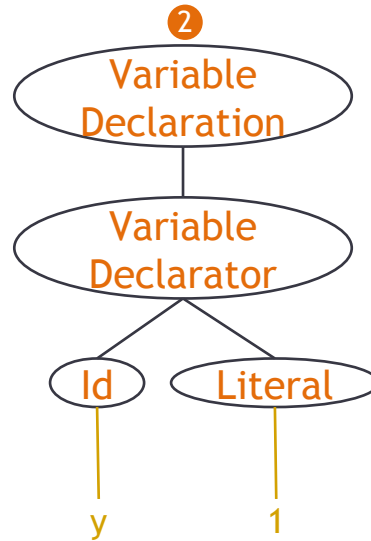
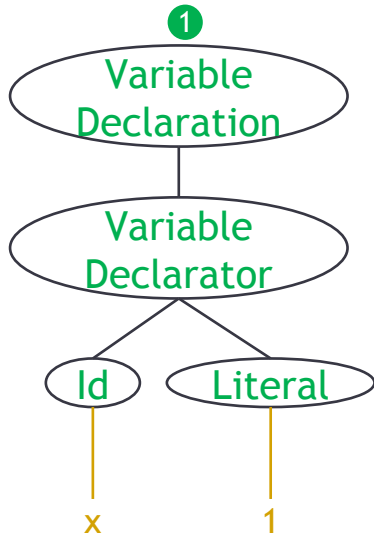
- AST: nesting of programming constructs



- 1 `var x = 1;`
- 2 `var y = 1;`
- 3 `if (x == 1)`

Abstract Syntax Tree

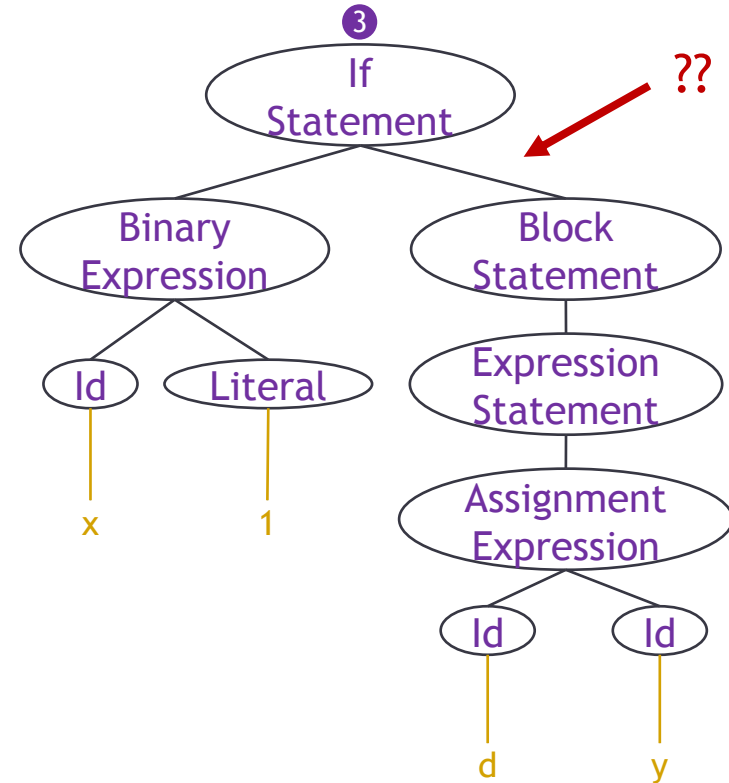
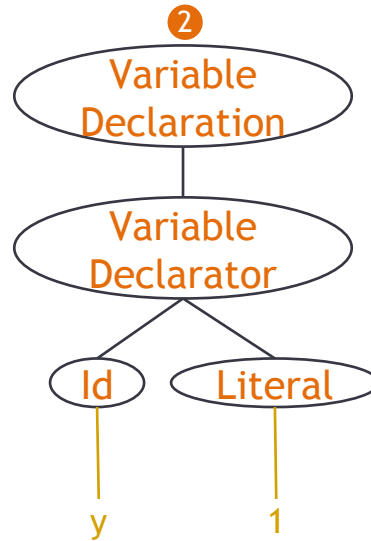
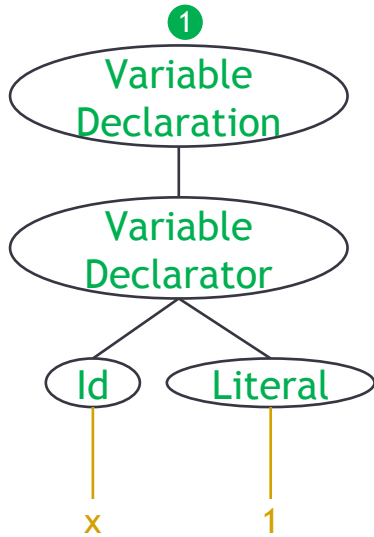
- AST: nesting of programming constructs



① `var x = 1;`
② `var y = 1;`
③ `if (x == 1) {d = y;}`

Abstract Syntax Tree

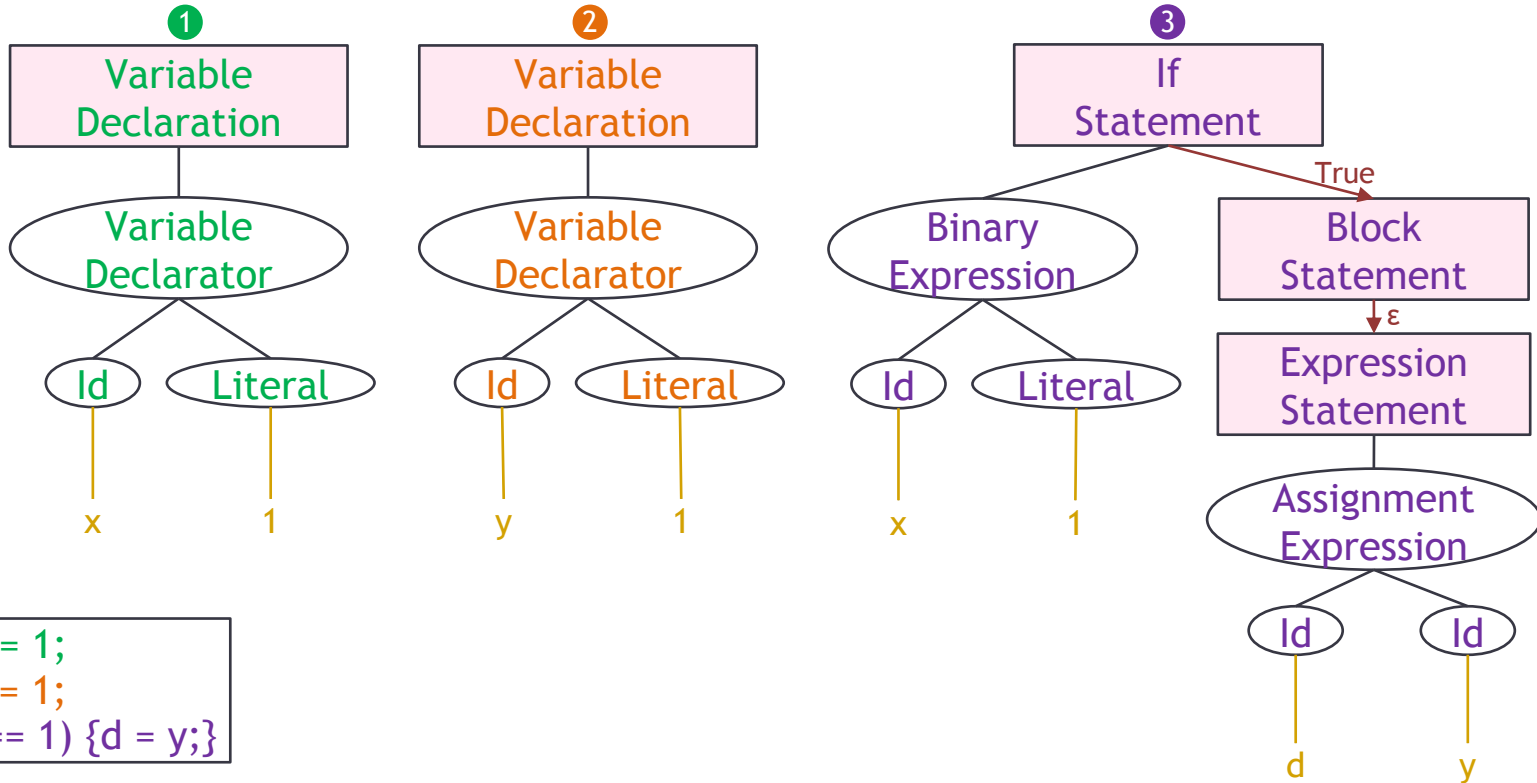
➤ AST: nesting of programming constructs



① `var x = 1;`
② `var y = 1;`
③ `if (x == 1) {d = y;}`

Control Flow Graph

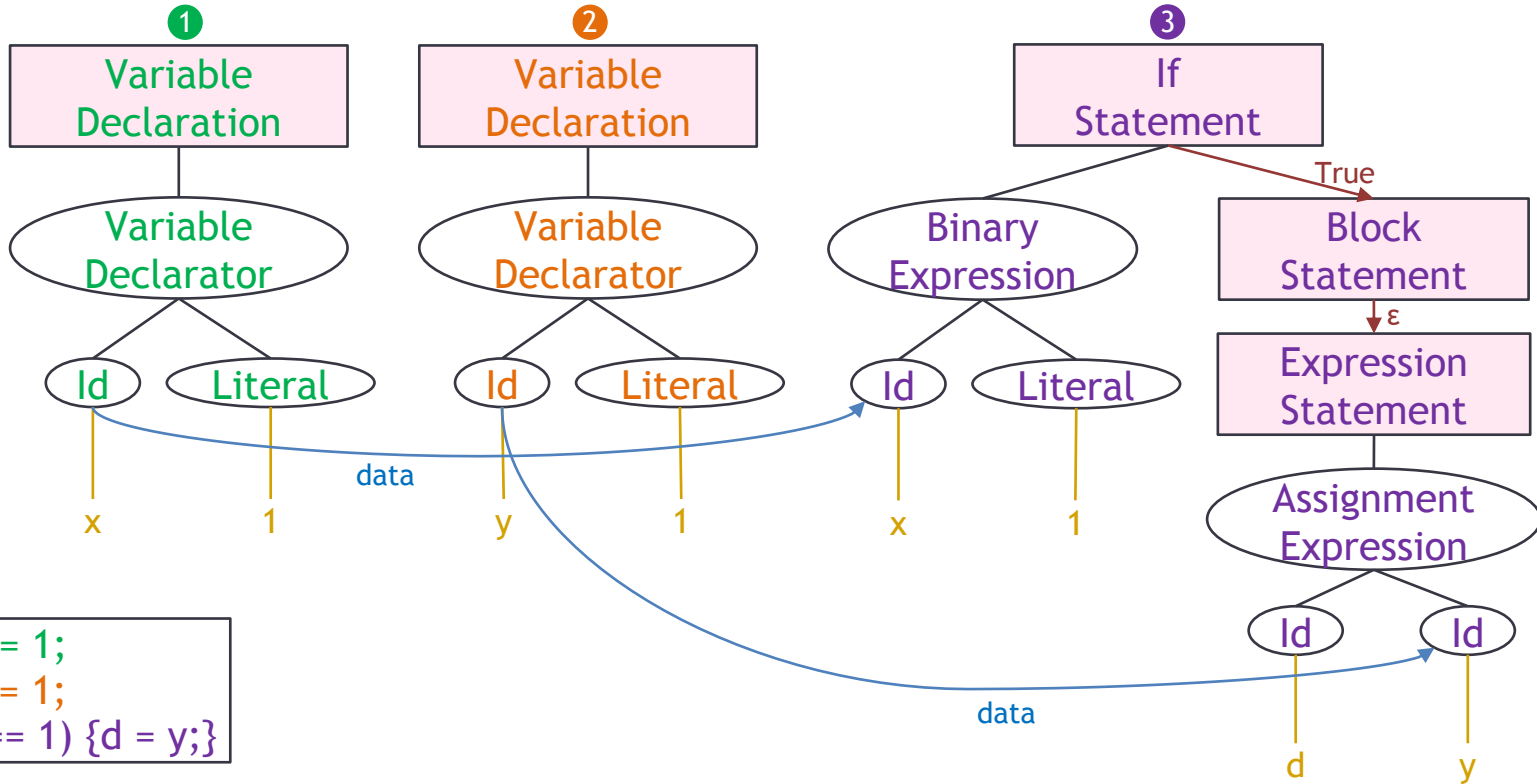
➤ CFG: execution path conditions



```
1 var x = 1;  
2 var y = 1;  
3 if (x == 1) {d = y;}
```

Program Dependence Graph

➤ PDG: variable dependency



```
1 var x = 1;  
2 var y = 1;  
3 if (x == 1) {d = y;}
```

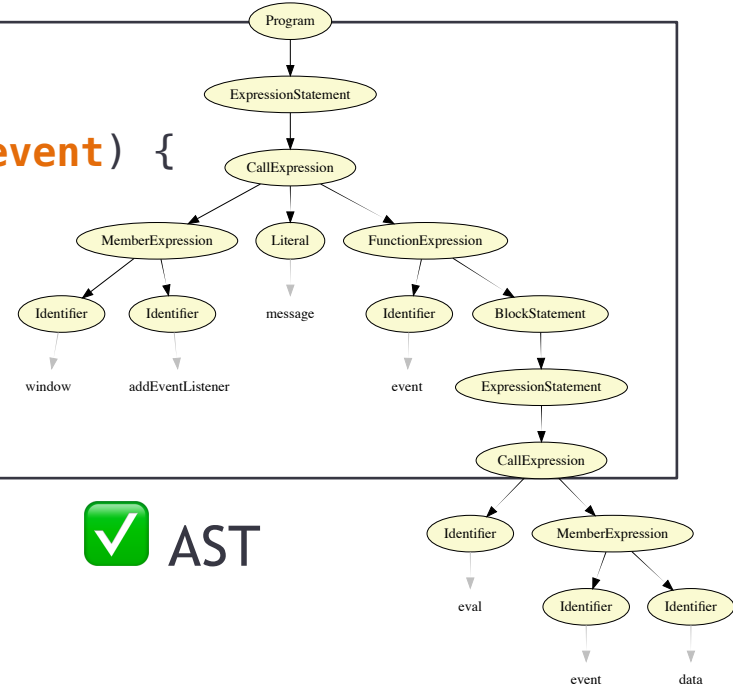
Per-Component JS Code Abstraction

```
// Content script code
```

```
window.addEventListener("message", function(event) {
```

```
    eval(event.data);
```

```
});
```




Abstract code representation



AST

Per-Component JS Code Abstraction

```
// Content script code
window.addEventListener("message", function(event) {
    eval(event.data);
})
```



Abstract code representation



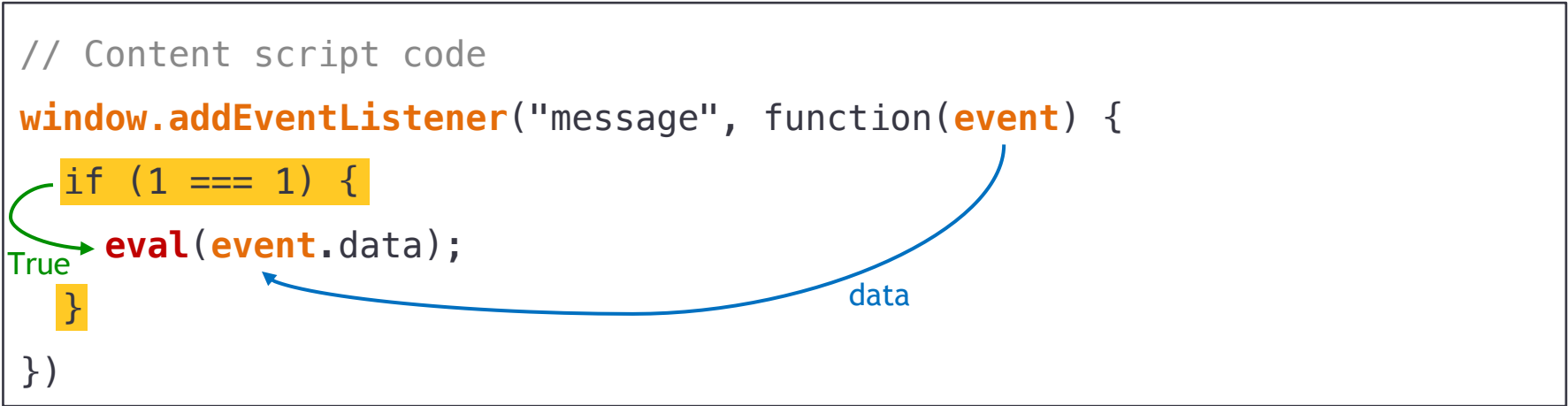
AST

– variable dependencies



data flow


```
// Content script code
window.addEventListener("message", function(event) {
  if (1 === 1) {
    eval(event.data);
  }
})
```



The diagram illustrates the flow of data and control in the provided JavaScript code. A blue arrow labeled 'data' originates from the 'event.data' property access in the function call and points to the argument of the 'eval' function. A green arrow labeled 'True' originates from the 'if (1 === 1)' condition and points to the 'eval' function call, indicating that the condition is satisfied and the code block is executed.

Abstract code representation



 AST

- conditions



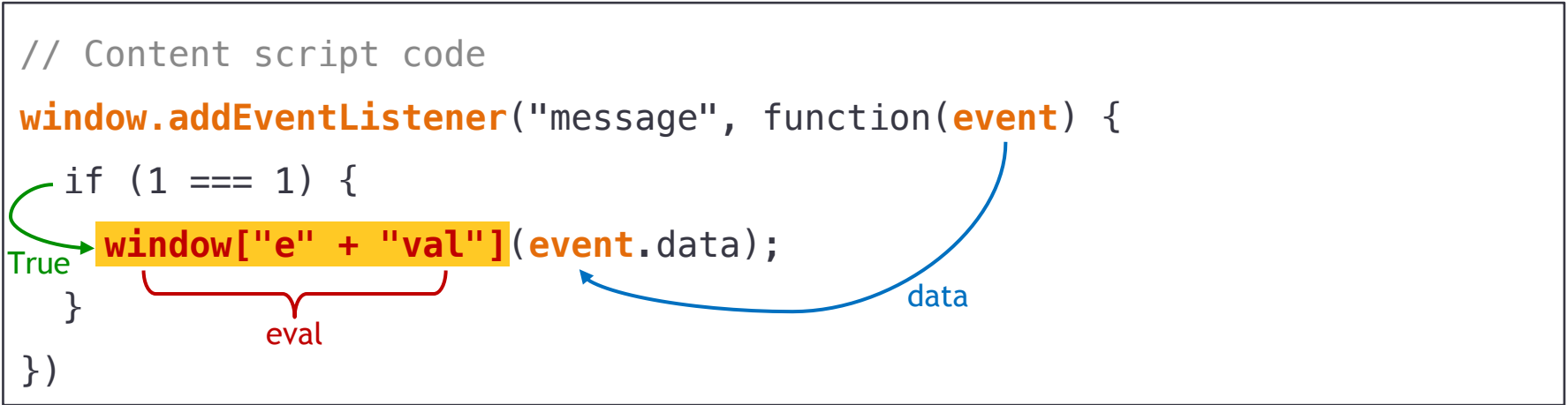
 control flow

- variable dependencies



 data flow

```
// Content script code
window.addEventListener("message", function(event) {
  if (1 === 1) {
    window["e" + "val"](event.data);
  }
})
```



Abstract code representation

– conditions

– variable dependencies

– variable values



✓ AST



✓ control flow

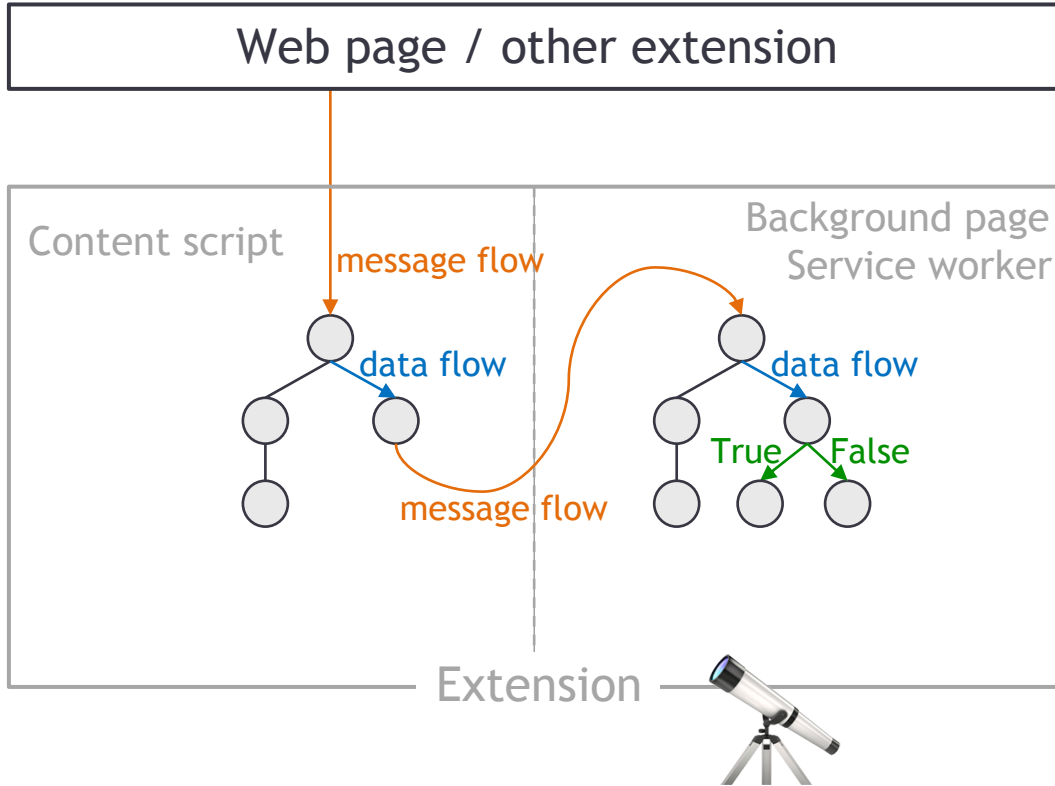


✓ data flow



✓ pointer analysis

Detecting Vulnerable Extensions



→ **DOUBLEX**: detects suspicious data flows from and toward an extension privileged context

Per-component JS code abstraction

- AST
- Control flow
- Data flow
- Pointer analysis

Extension Dependence Graph (EDG)

- Message interactions

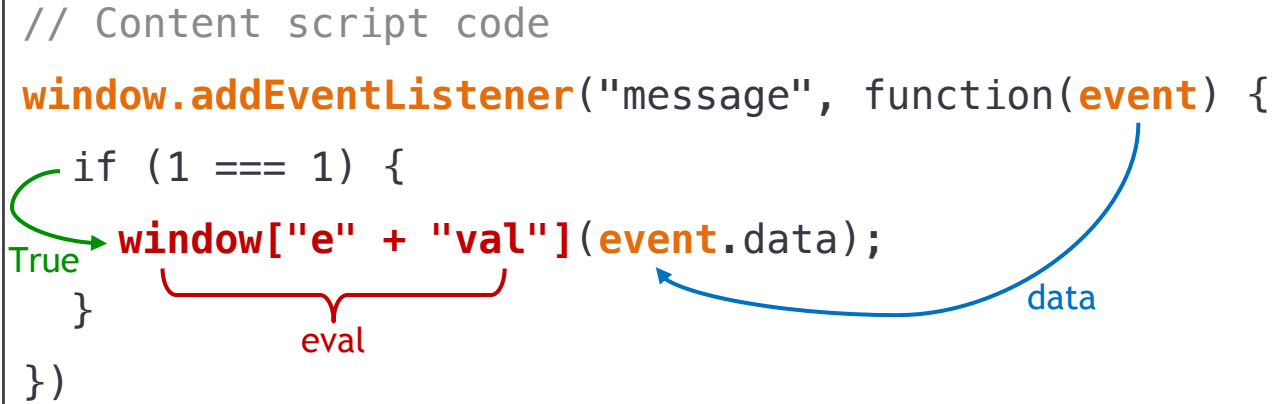
Suspicious data flow tracking

- Detects any path between an attacker & sensitive APIs

↓
Data flow report

Extension Dependence Graph

```
// Content script code
window.addEventListener("message", function(event) {
  if (1 === 1) {
    window["e" + "val"](event.data);
  }
})
```



- external messages
- internal messages

Extension Dependence Graph

```
// Content script code
window.addEventListener("message", function(event) {
  if (1 === 1) {
    window["e" + "val"](event.data);
  }
})
```



- external messages
- internal messages

Extension Dependence Graph

```
// Content script code
window.addEventListener("message", function(event) {
  if (1 === 1) {
    window["e" + "val"](event.data);
  }
})
```



– external messages



– internal messages

Extension Dependence Graph

```
// Content script code
window.addEventListener("message", function(event) {
  if (1 === 1) {
    window["e" + "val"](event.data);
  }
})
```

Diagram annotations:

- A purple devil emoji is placed above the `event` parameter in the function signature.
- A green arrow labeled "True" points from the `if (1 === 1)` condition to the function call.
- A red bracket labeled "eval" spans the string `"e" + "val"`.
- A blue arrow labeled "data" points from `event.data` to the function call.

– external messages



– internal messages

```
// Content script code  
chrome.runtime.sendMessage({toBP: mess});
```

– external messages



– internal messages

Extension Dependence Graph

```
// Content script code  
chrome.runtime.sendMessage({toBP: mess});
```

```
// Background page code  
chrome.runtime.onMessage.addListener(function(request) {  
  
})
```

– external messages



– internal messages

Extension Dependence Graph

```
// Content script code  
chrome.runtime.sendMessage({toBP: mess});
```

```
// Background page code  
chrome.runtime.onMessage.addListener(function(request) {  
  
})
```

– external messages



– internal messages

Extension Dependence Graph

```
// Content script code  
chrome.runtime.sendMessage({toBP: mess});
```

```
// Background page code  
chrome.runtime.onMessage.addListener(function(request) {  
  
})
```

- external messages
- internal messages



Extension Dependence Graph

```
// Content script code  
chrome.runtime.sendMessage({toBP: mess});
```

message

```
// Background page code  
chrome.runtime.onMessage.addListener(function(request) {  
})
```

– external messages



– internal messages



Extension Dependence Graph

```
// Content script code  
chrome.runtime.sendMessage({toBP: mess});
```

message

```
// Background page code  
chrome.runtime.onMessage.addListener(function(request) {  
  
})
```

– external messages

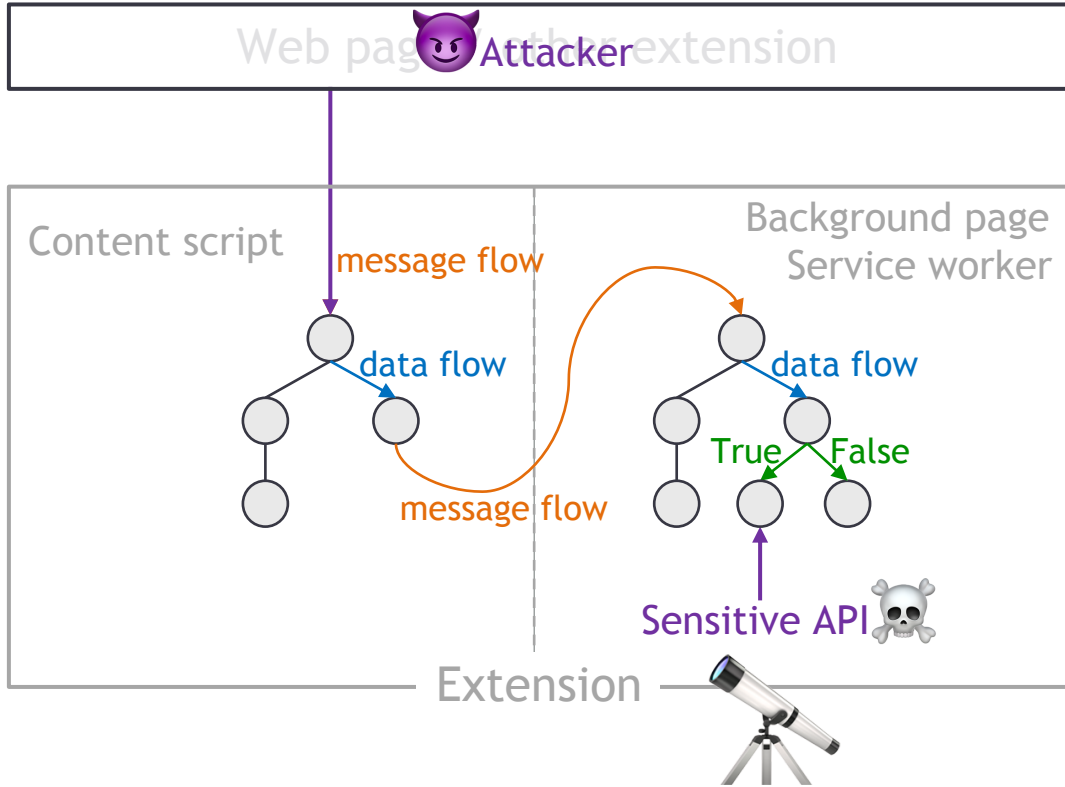


– internal messages



➤ Models message interaction within and outside of an extension

Detecting Vulnerable Extensions



→ **DOUBLEX**: detects suspicious data flows from and toward an extension privileged context

Per-component JS code abstraction

- AST
- Control flow
- Data flow
- Pointer analysis

Extension Dependence Graph (EDG)

- Message interactions


Suspicious data flow tracking

- Detects any path between an attacker & sensitive APIs

↓
Data flow report

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     if (1 === 1) {
4         window["e" + "val"](event.data);
5     }
6 })
```



The diagram highlights a suspicious data flow in the provided JavaScript code. A green arrow points from the `True` result of the `if (1 === 1)` condition to the `eval` expression. A red bracket underlines the `eval` expression. A blue arrow points from `event.data` to the `eval` expression.

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     if (1 === 1) {
4         window["e" + "val"](event.data);
5     }
6 })
```



The diagram highlights suspicious data flow tracking in the provided JavaScript code. A green arrow points from the condition `1 === 1` to the `eval` function call. A red bracket underlines the string `"e" + "val"` in the window property access, with a yellow box labeled `eval` below it. A blue arrow points from `event.data` to the function call, labeled `data`.

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     if (1 === 1) {
4         window["e" + "val"](event.data);
5     }
6 })
```

Diagram illustrating suspicious data flow tracking in the provided JavaScript code:

- A green arrow labeled "True" points from the condition `(1 === 1)` to the `if` block.
- A red bracket labeled "eval" spans the expression `"e" + "val"`, indicating a dynamic property access.
- A blue arrow labeled "data" points from the `event` parameter to the `event.data` property access.
- A purple devil emoji is placed above the `event` parameter, highlighting its role in the suspicious flow.

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3   if (1 === 1) {
4     window["e" + "val"](event.data);
5   }
6 })
```

The code is annotated with several elements: a purple devil emoji is placed above the `event` parameter in line 2. A yellow box highlights `event` in line 2. A blue arrow labeled `data` points from the `event` box in line 2 to the `event.data` property access in line 4. A red bracket labeled `eval` spans the string `"e" + "val"` in line 4. A green arrow labeled `True` points from the `if (1 === 1)` condition in line 3 to the `eval` bracket in line 4.



Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3   if (1 === 1) {
4     window["e" + "val"](event.data);
5   }
6 })
```

The diagram shows the code from the previous block with several annotations. A purple devil emoji is placed above the `event` parameter in line 2. A blue arrow labeled `data` points from the `event` parameter to the `event.data` property access in line 4. A red bracket labeled `eval` spans the expression `["e" + "val"]` in line 4. A green arrow labeled `True` points from the `if (1 === 1)` condition to the function call in line 4.



```
// Data flow report
{"direct-danger1": "eval",
 "value": "eval(event.data)",
 "line": "4 - 4",
 "dataflow": true,
 "param1": {
   "received": "event",
   "line": "2 - 2"}}}
```

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3   if (1 === 1) {
4     window["e" + "val"](event.data);
5   }
6 })
```



```
// Data flow report
{"direct-danger1": "eval",
"value": "eval(event.data)",
"line": "4 - 4",
"dataflow": true,
"param1": {
  "received": "event",
  "line": "2 - 2"}}}
```

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     if (1 === 1) {
4         window["e" + "val"](event.data);
5     }
6 }
7
8
9
10 }
```

True →

eval

data

```
// Data flow report
{"direct-danger1": "eval",
 "value": "eval(event.data)",
 "line": "4 - 4",
 "dataflow": true,
 "param1": {
   "received": "event",
   "line": "2 - 2"}},
```

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     if (1 === 1) {
4         window["e" + "val"](event.data);
5     }
6     event = {"data": 42};
7     eval(event.data);
8 }
9 }
10 }
```

True

eval

data

data

```
// Data flow report
{"direct-danger1": "eval",
 "value": "eval(event.data)",
 "line": "4 - 4",
 "dataflow": true,
 "param1": {
   "received": "event",
   "line": "2 - 2"}},
```

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     if (1 === 1) {
4         window["e" + "val"](event.data);
5     }
6     event = {"data": 42};
7     eval(event.data);
8 }
9 }
10 }
```

True

eval

data

data

```
// Data flow report
{"direct-danger1": "eval",
 "value": "eval(event.data)",
 "line": "4 - 4",
 "dataflow": true,
 "param1": {
   "received": "event",
   "line": "2 - 2"}},
{"direct-danger2": "eval",
 "value": "eval(42)",
 "line": "8 - 8",
 "dataflow": false}
```

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3   if (1 === 1) {
4     window["e" + "val"](event.data);
5   }
6   event = {"data": 42};
7   eval(event.data);
8 }
9 })
10
```

```
// Data flow report
{"direct-danger1": "eval",
 "value": "eval(event.data)",
 "line": "4 - 4",
 "dataflow": true,
 "param1": {
  "received": "event",
  "line": "2 - 2"}},

{"direct-danger2": "eval",
 "value": "eval(42)",
 "line": "8 - 8",
 "dataflow": false}
```


Suspicious Data Flow Tracking

1 // Content script code

2

3

4

5

6

7 // Background page code

8

9

10

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3
4
5
6 })
```

```
7 // Background page code
8
9
10
```

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     content = event.data.content;
4     tabId = event.data.tabId;
5
6 })
```

```
7 // Background page code
8
9
10
```

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     content = event.data.content;
4     tabId = event.data.tabId;
5     chrome.runtime.sendMessage({content: content, tabId: tabId});
6 })
```

```
7 // Background page code
8
9
10
```

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     content = event.data.content;
4     tabId = event.data.tabId;
5     chrome.runtime.sendMessage({content: content, tabId: tabId});
6 })
```

```
7 // Background page code
8 chrome.runtime.onMessage.addListener(function(request) {
9
10 }
```

message

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     content = event.data.content;
4     tabId = event.data.tabId;
5     chrome.runtime.sendMessage({content: content, tabId: tabId});
6 })
7 // Background page code
8 chrome.runtime.onMessage.addListener(function(request) {
9     chrome.tabs.executeScript(request.tabId, {code: request.content});
10 })
```

The diagram illustrates suspicious data flow tracking between two code snippets. The top snippet (lines 1-6) is content script code. It uses `addEventListener` to listen for a `message` event. The `event` parameter is marked with a purple devil emoji. The `content` and `tabId` properties of `event.data` are assigned to local variables, also marked with devil emojis. These variables are then passed to `chrome.runtime.sendMessage`, which is also marked with a devil emoji. An orange bracket groups the `content` and `tabId` arguments. An orange arrow labeled `message` points from this bracket to the `request` parameter of the `chrome.runtime.onMessage.addListener` function in the bottom snippet (lines 7-10). The `request` parameter is also marked with a devil emoji. Inside this listener, `chrome.tabs.executeScript` is called with `request.tabId` and `request.content`, both marked with devil emojis. A purple devil emoji is also placed at the start of line 9.

Suspicious Data Flow Tracking

```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     content = event.data.content;
4     tabId = event.data.tabId;
5     chrome.runtime.sendMessage({content: content, tabId: tabId});
6 })
```

```
7 // Background page code
8 chrome.runtime.onMessage.addListener(function(request) {
9     chrome.tabs.executeScript(request.tabId, {code: request.content});
10 })
```

```
// Data flow report
{"direct-danger1": "tabs.executeScript",
"line": "9 - 9",
"dataflow": true,
...
"param1": {
  "received": "event",
  "line": "2 - 2"}}}
```

Suspicious Data Flow Tracking

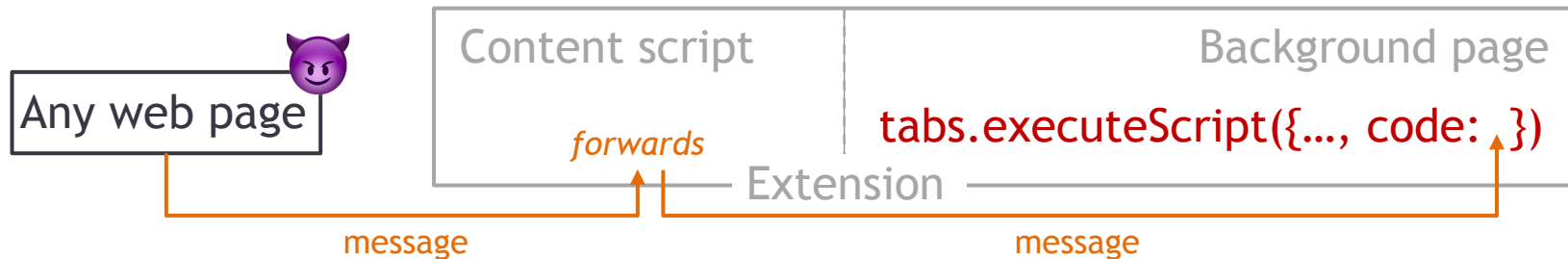
```
1 // Content script code
2 window.addEventListener("message", function(event) {
3     content = event.data.content;
4     tabId = event.data.tabId;
5     chrome.runtime.sendMessage({content: content, tabId: tabId});
6 })
```

```
7 // Background page code
8 chrome.runtime.onMessage.addListener(function(request) {
9     chrome.tabs.executeScript(request.tabId, {code: request.content});
10 })
```

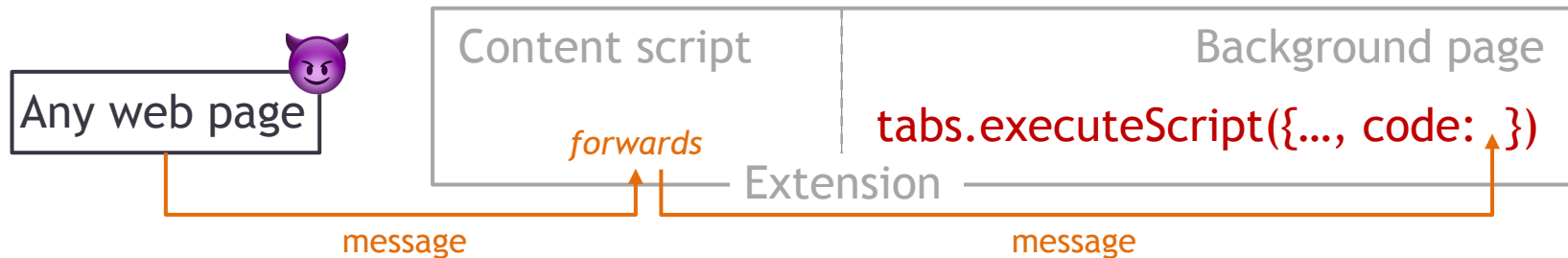
message

```
// Data flow report
{"direct-danger1": "tabs.executeScript",
"line": "9 - 9",
"dataflow": true,
...
"param1": {
  "received": "event",
  "line": "2 - 2"}}}
```

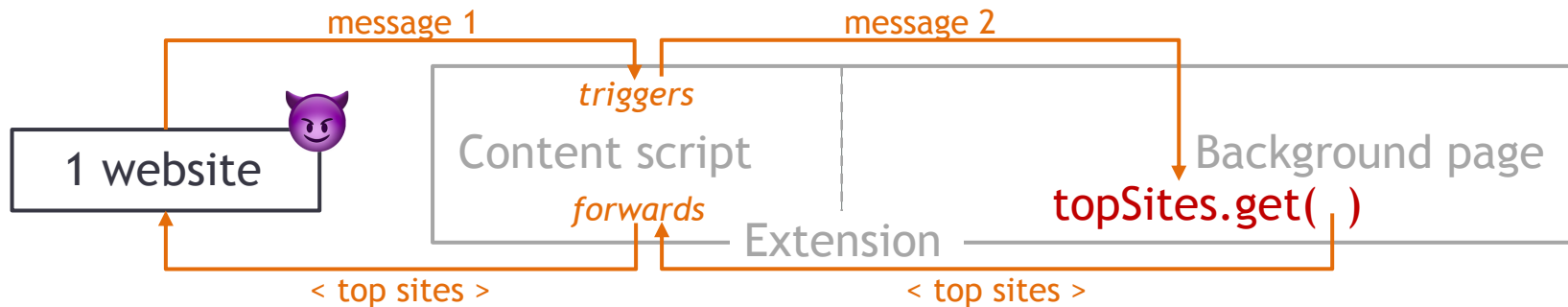

- Arbitrary code execution (*cdi...*, 4k+ users)



- Arbitrary code execution (*cdi...*, 4k+ users)



- Most visited website exfiltration (*lkl...*, 700k+ users)



Detecting Vulnerable Extensions with DOUBLEX

Analyzed 155k Chrome extensions from 2021 with DOUBLEX

- **184 vulnerable Chrome extensions**
- **Impacting 3M users**

- **Precision: 89%** of the flagged extensions are vulnerable
- **Recall: 93%** of known vulnerabilities [2] are detected

- **Integration** in the **vetting process** conducted by Google
- **Available online**, for developers
(even in other fields!)

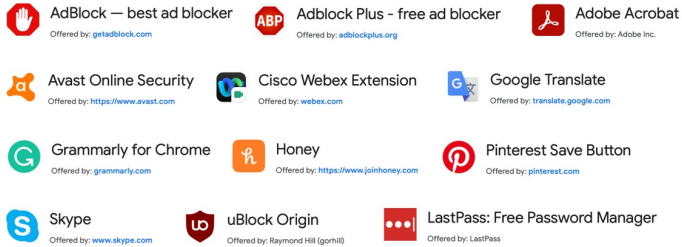


 Aurore54F/DoubleX

- Know that communication with external actors may be dangerous
- Only allow communication with specified extensions or web pages
- Limit:
 - code execution by sanitizing messages
 - SOP bypass by preferring CORS for cross-origin requests
- DOUBLEX could provide a feedback channel for developers
- Migrate an extension to Manifest v3

Takeaways — Browser Extension (In)Security

Browser Extensions are Popular



- **Bundles** of JS, HTML, or CSS files, defined in a `manifest.json`
- **145k** Chrome extensions totaling over **1.6B** active users

Security-Noteworthy Extensions (SNE)

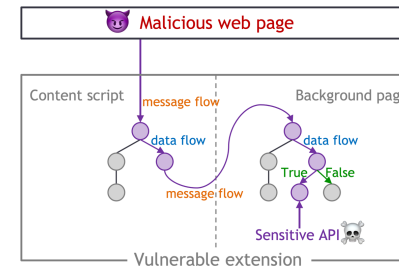
- **Contain malware**
 - Designed by malicious actors to harm victims
 - E.g., propagate malware, steal users' credentials, track users
- **Violate the Chrome Web Store policies**
 - E.g., deceive users, promote unlawful activities, lack a privacy policy
- **Contain vulnerabilities**
 - Designed by well-intentioned developers... but contain some vulnerabilities
 - E.g., can lead to user-sensitive data exfiltration

What is in the Chrome Web Store?

- **350M users** installed SNE in the last 3 years
- These SNE stay in the Chrome Web Store *for years*
- Extensions have a **short life cycle** in the CWS (60% stay 1 year)
- Critical **lack of maintenance** in the CWS (60% received no update)



Detecting Vulnerable Extensions with DOUBLEX



Auore54F/DoubleX

- DOUBLEX **detects suspicious data flows** in browser extensions
184 vulnerable extensions | **Precision: 89%** | **Recall: 93%**

- [What is in the Chrome Web Store?](#)

Sheryl Hsu, Manda Tran, and [Aurore Fass](#). In *ACM AsiaCCS 2024*

- [DoubleX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale](#)

[Aurore Fass](#), Dolière Francis Somé, Michael Backes, and Ben Stock. In *ACM CCS 2021*