26TH USENIX SECURITY SYMPOSIUM

DeustoTech

EURECOM
Sophia Antipolis

Extension Breakdown:

Security Analysis of Browsers Extension Resources Control Policies

Iskander Sanchez-Rola, Igor Santos, Davide Balzarotti

Extensions

Browser extensions are the most popular technique currently available to **extend the functionalities** of modern web browsers.

Extensions exist for **most browser families**, including major web browsers such as Firefox, Chrome, Safari, Opera and Edge.

They can be easily installed by users from a **central repository**.

Extensions

An extension is a **bundle of resources**, including code (such as HTML or JS), images, style sheets...

Third-party websites should **never have access** to them, as some contain private information, vulnerabilities, keys…

Browsers need to somehow control the access to extensions files. This task is **tricky and error prone**.

Resources Control Policies

Access Control Settings (Chromium+Firefox)
Extensions themselves specify which resources they need to be kept private and which can be made publicly available. By default all resources are considered private.

URI Randomization (Safari)
There is no distinction between private or public resources, but instead the base URI of the extension is randomly re-generated in each session.

Access Control Settings

Browsers currently implement ACS by performing two consecutive checks to verify:

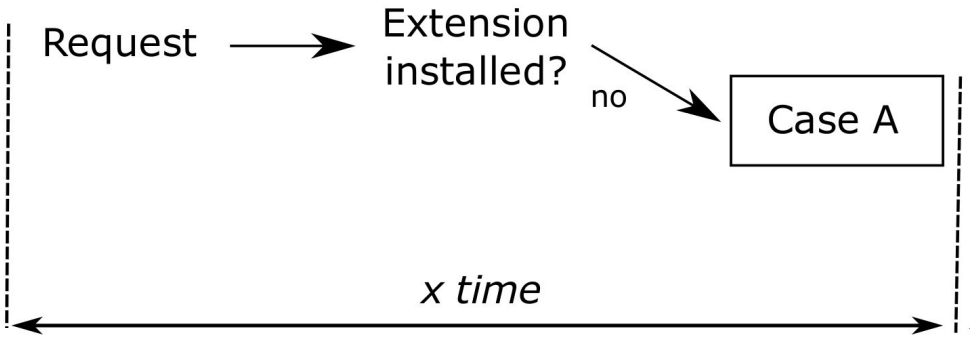
(i) if a certain extension **is installed**
(ii) if the requested resource **is publicly available**

This is prone to a **timing side-channel attack** that an adversary can use to identify the actual reason behind a request denial:

- The extension is not present
- Its resources are kept private

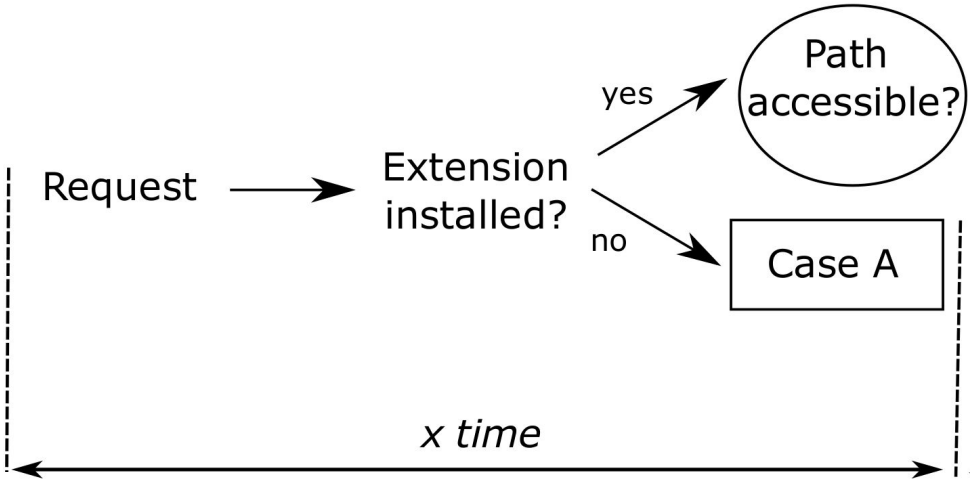
Access Control Settings

X-extension://[**fakeExtID**]/[fakePath]



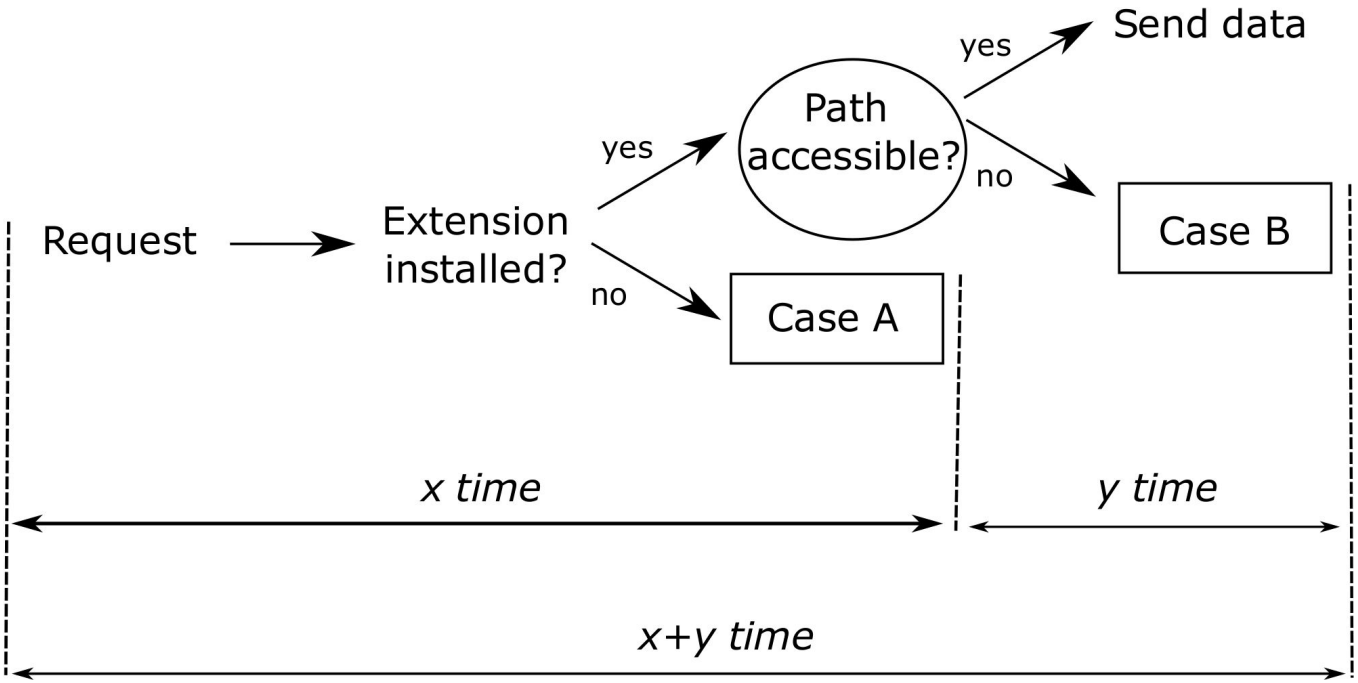
Access Control Settings

X-extension://[**realExtID**]/[fakePath]



Access Control Settings

X-extension://[**realExtID**]/[fakePath]



Access Control Settings

We compared our approach to **previous techniques** capable of enumerating extensions by subverting access control settings. These methods are based on checking the existence of **externally accessible resources** in extensions.

	Chrome	Firefox	Total
# Extensions Tested	10,620	10,620	21,240
% Previous Approaches	12.73%	8,17%	10,45%
% Our Approach	*100.00%*	*100.00%*	*100.00%*

URI Randomization

Extensions are often used to inject additional content, controls, or simply alert panels into a website.

This newly generated content can unintentionally **leak the random extension URI**, thus bypassing the security control measures and opening access to all the extension resources to any other code running in the same page.

It is left to the extension developers to make sure this does not happen.

URI Randomization

```
1  wot.rating = {
2  toggleframe: function(id, file, style){
3   try {
4     var frame = document.getElementById(
          id);
5     if (frame) {
6      frame.parentNode.removeChild(frame);
7      return true;
8     } else {
9      var body = document.
           getElementsByTagName("body");
10     if (body && body.length) {
11      frame = document.createElement("
           iframe");
12      if (frame) {
13        frame.src = safari.extension.
             baseURI+file;
14        frame.setAttribute("id", id);
15        frame.setAttribute("style", style)
             ;
16        if (body[0].appendChild(frame))
17         {return true;}
18      }
19     }
20    }
21   } catch (e) {
22     console.log("failed with"+e+"\n");}
23   return false;
24  }
```

URI Randomization

```
1  wot.rating = {
2  toggleframe: function(id, file, style){
3   try {
4     var frame = document.getElementById(
          id);
5     if (frame) {
6      frame.parentNode.removeChild(frame);
7      return true;
8     } else {
9      var body = document.
           getElementsByTagName("body");
10     if (body && body.length) {
11      frame = document.createElement("
           iframe");
12      if (frame) {
13        frame.src = safari.extension.
              baseURI+file;
14        frame.setAttribute("id", id);
15        frame.setAttribute("style", style)
              ;
16        if (body[0].appendChild(frame))
17         {return true;}
18      }
19     }
20    }
21   } catch (e) {
22     console.log("failed with"+e+"\n");}
23  return false;
24 }
```

URI Randomization

```
1  wot.rating = {
2  toggleframe: function(id, file, style){
3   try {
4     var frame = document.getElementById(
          id);
5     if (frame) {
6      frame.parentNode.removeChild(frame);
7      return true;
8     } else {
9      var body = document.
           getElementsByTagName("body");
10     if (body && body.length) {
11      frame = document.createElement("
           iframe");
12      if (frame) {
13       frame.src = safari.extension.
            baseURI+file;
14       frame.setAttribute("id", id);
15       frame.setAttribute("style", style)
            ;
16       if (body[0].appendChild(frame))
17        {return true;}
18     }
19    }
20   }
21  } catch (e) {
22    console.log("failed with"+e+"\n");}
23  return false;
24 }
```

URI Randomization

We propose a static analysis of all the JavaScript components of an extension.

URI Randomization

We propose a static analysis of all the JavaScript components of an extension.

(i) Identify the **source locations** where the code accesses the random extension URI (looking for calls to baseURI)

URI Randomization

We propose a static analysis of all the JavaScript components of an extension.

(i) Identify the **source locations** where the code accesses the random extension URI (looking for calls to baseURI)

(ii) Analyze all the components that **can use the retrieved value** following the information flow

URI Randomization

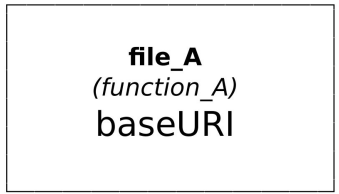
We propose a static analysis of all the JavaScript components of an extension.

(i) Identify the **source locations** where the code accesses the random extension URI (looking for calls to baseURI)

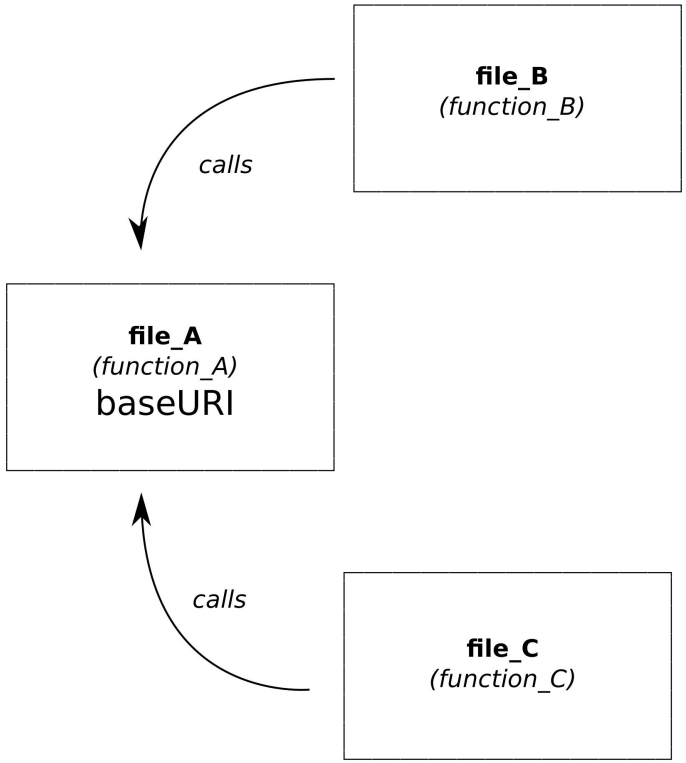
(ii) Analyze all the components that **can use the retrieved value** following the information flow

(iii) For every identified components, locate the sinks
(i.e., the location where new **content is injected in the page**)

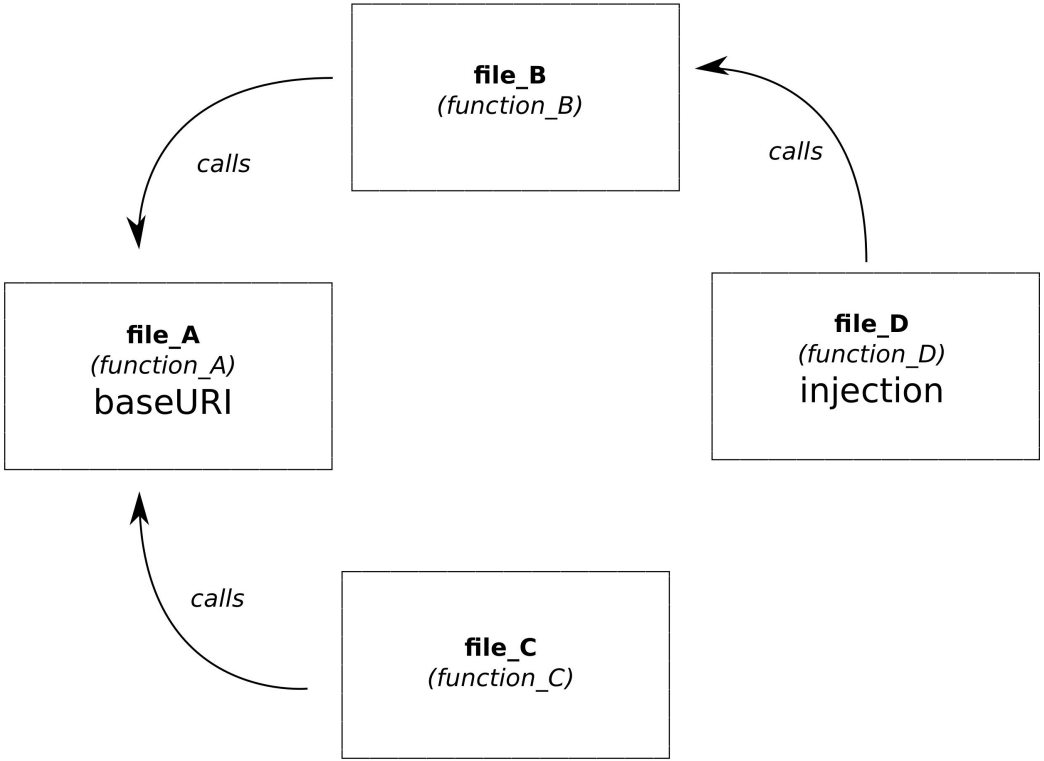
URI Randomization



URI Randomization



URI Randomization



URI Randomization

Category	# Ext.	% Leak
Shopping	95	57.89%
Email	13	53.85%
Security	84	52.38%
News	20	45.00%
Photos	25	44.00%
Bookmarking	61	42.62%
Productivity	147	40.82%
RSStools	5	40.00%
Entertainment	37	37.84%
Translation	8	37.50%
Social	80	30.00%
Developer	57	29.82%
Other	42	26.19%
Search	42	24.43%
urlshorteners	5	0.00%
Total	*721*	*40.50%*

URI Randomization

We performed an exhaustive manual code review of security extensions to confirm the leakage.

- Popular **protection extensions** such as Adblock, Ghostery, Web Of Trust, and Adguard
- **Password managers**, such as LastPass, Dashline, Keeper, and TeedyID
- **Combinations** of the two, such as Blur from Abine

Impact

There are several possible consequences of abusing the information provided by our two techniques:

- **Fingerprinting and Analytics:**
  - ➔ Stateless tracking
  - ➔ Browser identification (checking built-in extensions)
  - ➔ Determine users' demographics

Impact

There are several possible consequences of abusing the information provided by our two techniques:

- **Fingerprinting and Analytics:**
  - ➔ Stateless tracking
  - ➔ Browser identification (checking built-in extensions)
  - ➔ Determine users' demographics

- **Malicious Applications**
  - ➔ Information gathering phase
  - ➔ Social-driven attacks
  - ➔ Exploitation of potential vulnerabilities

Impact

Device Fingerprinting **Viability Study**

Method	Entropy
Extensions	*0.869*
List of Plugins	0.718
List of Fonts	0.548
User Agent	0.550
Canvas	0.475
Content Language	0.344
Screen Resolution	0.263

Vulnerability Disclosure

Chromium Family

Developers were quite surprised, because they believed that the time difference in the checking phase were not significant enough to allow this type of attack.

Developers are still working to solve this problem.

In addition, as the new **Firefox WebExtensions and Microsoft Edge** (both currently in their early stages) use the same extension control mechanisms, we also notified their developers.

Vulnerability Disclosure

Firefox Family

Firefox **non-WebExtensions** problem was acknowledged and developers are currently discussing how to proceed.

Regarding **WebExtensions**, the Firefox developers recently changed the way extensions are accessed to solve this timing side-channel and other related attacks.
In particular, they changed the initial scheme from
moz-extension://[extID]/[path] to **moz-extension://[random-UUID]/[path]**

Vulnerability Disclosure

Firefox Family

Firefox **non-WebExtensions** problem was acknowledged and developers are currently discussing how to proceed.

Regarding **WebExtensions**, the Firefox developers recently changed the way extensions are accessed to solve this timing side-channel and other related attacks.
In particular, they changed the initial scheme from
moz-extension://[extID]/[path] to **moz-extension://[random-UUID]/[path]**

This change introduced a new dangerous problem: the random-UUID token can now be used to precisely fingerprint users as once it is generated it **never changes** (also reported).

Vulnerability Disclosure

Safari

The method that Safari's extension control employs to assure the proper accessibility of resources is, in principle, **correct**.

We started reporting the problem to the developers of **security extensions** we already manually confirmed vulnerable, to help them solve their URI leakage problem.

Security Proposal

1

All browsers should follow an extension scheme that includes a random value in the URI: **X-extension://[randomVal]/[path]**.

This random value should be modified across and during the same session and should be independent for each extension installed. In this way, the random value cannot be used to fingerprint users.

Security Proposal

2

Browsers should also implement an **access control** (such as web accessible resource) to avoid any undesirable access to all extensions resources even when the random value is unintentionally leaked by the extension.

Security Proposal

3

Extensions should be analyzed for possible leakages before making them public to the users. For example, adopting a **lightweight static analysis** solution (similar to the one we discuss) to analyze the extensions in their market and flag those that leak the random token.

Moreover, **developer manuals** should specifically discuss the problems that can cause the leakage of any random value generated.

We already knew about the communication breakdown…

We already knew about the communication breakdown…

We already knew about the communication breakdown…

We already knew about the communication breakdown…

but browsers didn’t told us about…

their new single…

Extension Breakdown

iskander.sanchez@deusto.es iskander-sanchez-rola.github.io