Site Isolation
Confining Untrustworthy Code in the Web Browser

Nasko Oskov, Charlie Reis
Defense

Browser evolution

Site Isolation architecture

Making it shippable

Offense

How to look for bypasses

Example vulnerabilities
Late 1990s

Monolithic
Late 2000s

Multi-process

Browser Process

Renderer Process

evil.com
Late 2000s

Multi-process
Late 2010s

Site Isolation

Browser Process

Renderer Process

mail.com

Renderer Process

evil.com
2018

Spectre
about:site-isolation
Why electric scooters are illegal in New York and London

The laws, regulations, and politics working against them

By Jon Porter | @JonPorty | Nov 25, 2019, 9:00am EST

Spend any time in New York or London, and you'll inevitably come across dozens of people whizzing along each city's streets on electric scooters. Yet, despite their popularity, e-scooters are technically illegal in both places, and the politicians with the power to change things are in no rush to do so.
Why electric scooters are illegal in New York and London

The laws, regulations, and politics working against them

By Jon Porter | @JonPorty | Nov 25, 2019, 9:00am EST

Spend any time in New York or London, and you'll inevitably come across dozens of people whizzing along each city's streets on electric scooters. Yet, despite their popularity, e-scooters are technically illegal in both places, and the politicians with the power to change things are in no rush to do so.
Without Site Isolation
Example: Input events

Operating System → Browser Process

Rendere: Process

A

B
Input events with out-of-process iframes

A

Browser Process

B

Renderer Process

Renderer Process

Operating System
Updated browser features

- Accessibility
- Developer tools
- Drag and drop
- Extensions
- Find-in-page
- Focus
- Form autofill
- Fullscreen
- IME
- Input gestures
- JavaScript dialogs
- Mixed content handling
- Multiple monitor and device scale factor
- Password manager
- Pointer Lock API
- Printing
- Task manager
- Resource optimizations
- Malware and phishing detection
- Save page to disk
- Screen Orientation API
- Scroll bubbling
- Session restore
- Spellcheck
- Tooltips
- Unresponsive renderer detector and dialog
- User gesture tracking
- View source
- Visibility APIs
- Webdriver automation
- Zoom
Process Isolation FTW

Not yet...
Cross-Origin Read Blocking

Must allow images, scripts, stylesheets

Want to protect sensitive data (HTML, XML, JSON)

Mislabeled Content-Types

- Custom sniffing
- Must allow responses like:

```html
Content-Type: text/html

<!-- This is JS. -->
function a() {...}
```

```html
<img src= "bar.com/image.jpg">
<img src= "bar.com/secret.html">
```
Security Benefits
Defending against Spectre

JavaScript can leak any memory within address space. No bugs in browser required.

Must keep data worth stealing out of attacker's process.
Compromised Renderer Processes

Harder than Spectre: Renderer process can lie to you!

UXSS, logic/memory bugs, RCE.

Must ensure browser process always checks for access to site data.
Addressing Limitations

● Sites vs Origins
  ○ [https://google.com](https://google.com) vs [https://mail.google.com:443](https://mail.google.com:443) (due to document.domain)
  ○ Opt-in origin isolation

● Many data types are not yet protected
  ○ Headers (CORP, Sec-Fetch-Site), more CORB-protected types, SameSite cookie defaults

● Cross-process transient execution attacks (e.g., Fallout, RIDL)
  ○ Combine with OS/HW mitigations
Compatibility & Performance

Don't break the web!

Performance implications?

- More processes. Memory overhead?
- Parallelism. Smaller processes.
- Latency: navigation, input events
Desktop: Isolate all sites

Shipped in May 2018 (Chrome 67): Windows, Mac, Linux, ChromeOS.

Many optimizations: spare process, same-site process sharing, etc

Workload helps: often many tabs open

  Subframes can often share existing same-site process
Practical to Deploy

Renderer Process Count

Memory Overhead
Android: Isolate subset of sites

Harder workload: single active tab

Isolate only high value sites: password-based

Shipped in September 2019 (Chrome 77)

(Still working on compromised renderer defenses here)
Fun stats for desktop launch

5 years of development

~450k lines of code, ~9k files touched

~4000 commits

Top 20 contributors landed 72% of the commits
Result

Practical to deploy
Chrome Desktop: All sites
Chrome Android: Password sites
Best path to protection against Spectre
Can limit damage from fully compromised renderers
about:offense
Chrome VRP covers Site Isolation bypasses

Breaking the Site Isolation process model:

- Causing two sites to use the same process

Stealing cross-site data:

- Cookies
- HTML5 storage (localStorage, IndexedDB, etc)
- CORB bypass to fetch cross-site network data

Some areas are out of scope for now
Bounty treasure map!
Chrome treasure map

Operating System

Browser Process

Utility process

Web Renderer

Extensions Renderer

GPU Process

Network Process
Older Exploits: Attack OS kernel

- Operating System
- Browser Process
- Web Renderer
UXSS
Logic Bugs
RCE
How to look for bypasses?

No need for actual renderer exploit. Just use a debugger!

Explore the IPC surface

- *_messages.h
- *.mojom

Get creative and poke around different areas

- Escalate to higher privileged processes (e.g. Network, GPU)
- Look for corner cases - about:blank, session restore, blob:
about:bugs
917668: Cross Domain Bug in IndexedDB

By lying about origin, any renderer can

-Enumerate
-Read
-Delete

IndexedDB for other origins.
IndexedDB Interface

`blink/public/mojom/indexeddb/indexeddb.mojom`

```cpp
interface IDBFactory {
    GetDatabaseInfo(associated IDBCallbacks callbacks, url.mojom.Origin origin);
    Open(associated IDBCallbacks callbacks, 
        associated IDBDatabaseCallbacks database_callbacks, 
        url.mojom.Origin origin, 
        mojo_base.mojom.String16 name, 
        int64 version, 
        int64 transaction_id);
    DeleteDatabase(associated IDBCallbacks callbacks, 
        url.mojom.Origin origin, 
        mojo_base.mojom.String16 name, 
        bool force_close);
    ...
}
```
IndexedDB Bug

interface IDBFactory {
  GetDatabaseInfo(associated IDBCallbacks callbacks, url.mojom.Origin origin);
  Open(associated IDBCallbacks callbacks,
       associated IDBDatabaseCallbacks database_callbacks,
       url.mojom.Origin origin,
       mojo_base.mojom.String16 name,
       int64 version,
       int64 transaction_id);
  DeleteDatabase(associated IDBCallbacks callbacks,
                 url.mojom.Origin origin,
                 mojo_base.mojom.String16 name,
                 bool force_close);
...}
The Fix

blink/public/mojom/indexeddb/indexeddb.mojom

interface IDBFactory {
  GetDatabaseInfo(associated IDBCallbacks callbacks, url.mojom.Origin origin);
  Open(associated IDBCallbacks callbacks,
       associated IDBDatabaseCallbacks database_callbacks,
       url.mojom.Origin origin,
       mojo_base.mojom.String16 name,
       int64 version,
       int64 transaction_id);
  DeleteDatabase(associated IDBCallbacks callbacks,
                 url.mojom.Origin origin,
                 mojo_base.mojom.String16 name,
                 bool force_close);
...
886976: Site Isolation bypass using Blob URL

By lying about the origin of a blob: URL, attacker can:

- Cause the process model to put attacker blob: URL in victim process
- Use the blob: URL to execute arbitrary JavaScript in the victim origin

Awarded at $8000.
var text = '<script>console.log("attacker code")</script>'';

var blob = new Blob([text], {type : 'text/html'});
var url = URL.createObjectURL(blob);

// Lie to the browser about the origin url
frames[0].location.href = url;
void BlobDispatcherHost::OnRegisterPublicBlobURL(const GURL& public_url,
     const std::string& uuid) {

    // Blob urls have embedded origins. A frame should only be creating blob URLs
    // in the origin of its current document. Make sure that the origin advertised
    // on the URL is allowed to be rendered in this process.
    if (!public_url.SchemeIsBlob() ||
        !security_policy->CanCommitURL(process_id_, public_url)) {

        bad_message::ReceivedBadMessage(this, bad_message::BDH_DISALLOWED_ORIGIN);
        return;
    }

    ...
}
blob: URLs Bug

void BlobDispatcherHost::OnRegisterPublicBlobURL(const GURL& public_url, const std::string& uuid) {
    ...
    // Blob urls have embedded origins. A frame should only be creating blob URLs
    // in the origin of its current document. Make sure that the origin advertised
    // on the URL is allowed to be rendered in this process.
    if (!public_url.SchemeIsBlob() ||
        !security_policy->CanCommitURL(process_id_, public_url)) {
        bad_message::ReceivedBadMessage(this, bad_message::BDH_DISALLOWED_ORIGIN);
        return;
    }
    ...
}
bool ChildProcessSecurityPolicyImpl::CanCommitURL(int child_id, 
       const GURL& url) {

    ... 

    if (!CanAccessDataForOrigin(child_id, url)) 
        return false;

    ... 

}
Finding bypasses is a thing now!
Conclusion

Site Isolation reduces value of many attacks: Spectre, UXSS, even RCE

We are still addressing limitations: coverage, granularity. Web also needs to evolve to better protect data.

Explore this new security frontier and find new attacks!