Are your Sites Truly Isolated?

Automatically Detecting Logic Bugs in Site Isolation Implementations

Jan Drescher David Klein Martin Johns

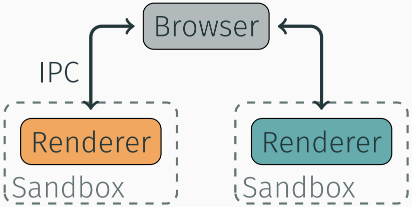
February 25, 2026

INSTITUTE FOR APPLICATION SECURITY

Technische Universität Braunschweig

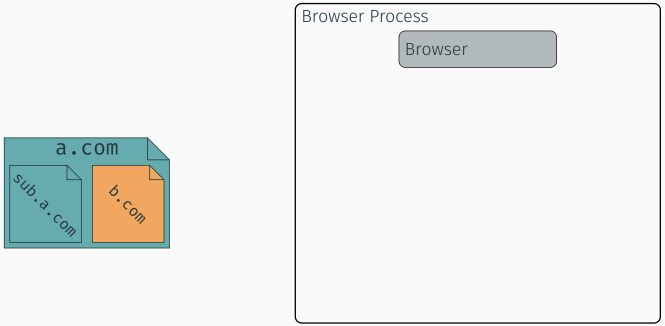
Site Isolation

- 1 renderer process per site[1]
- site = scheme + eTLD+1
- inter-process communication (IPC)

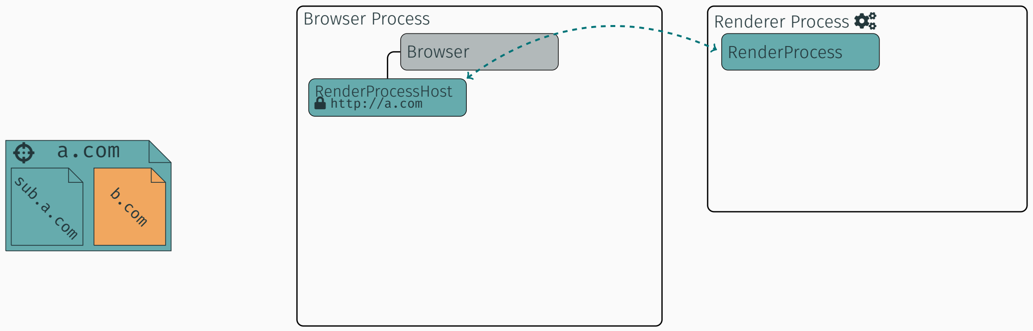


[1] Reis, Moshchuk, and Oskov, “Site Isolation: Process Separation for Web Sites within the Browser.”

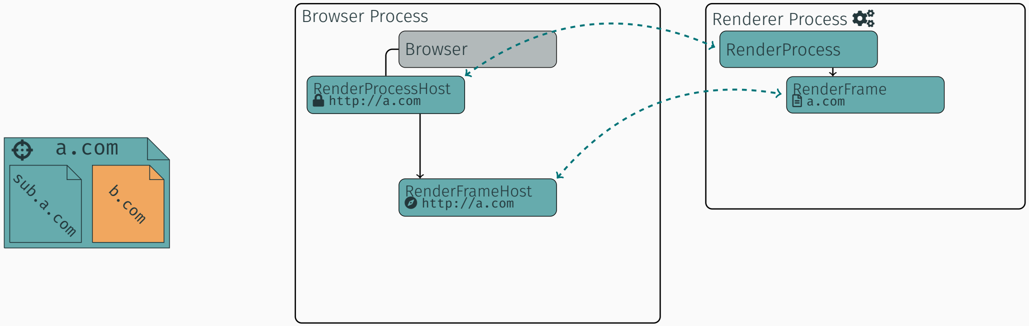
Site Isolation Architecture



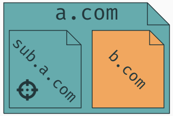
Site Isolation Architecture

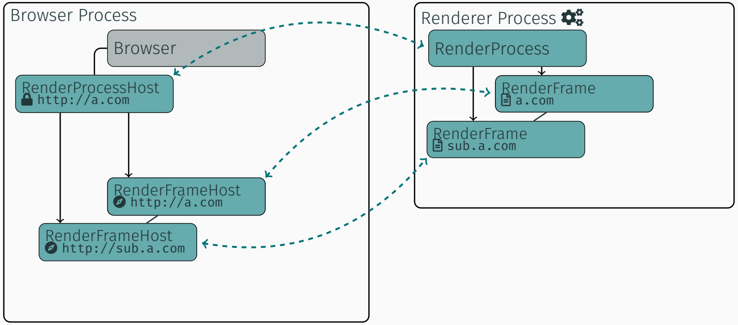


Site Isolation Architecture

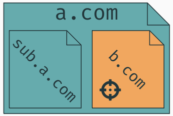


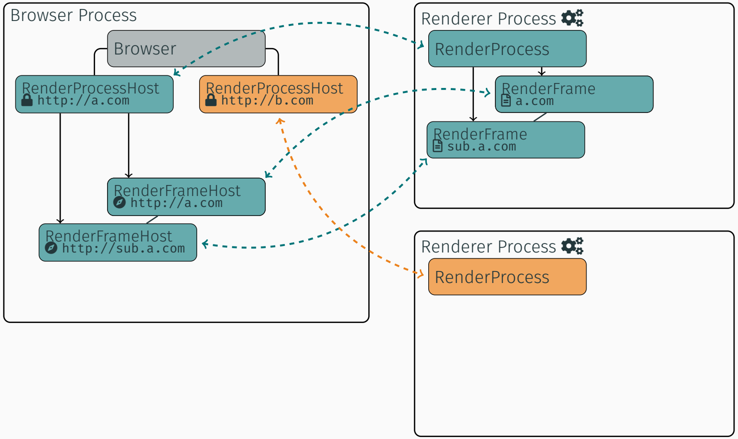
Site Isolation Architecture



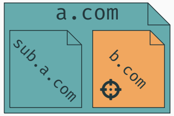


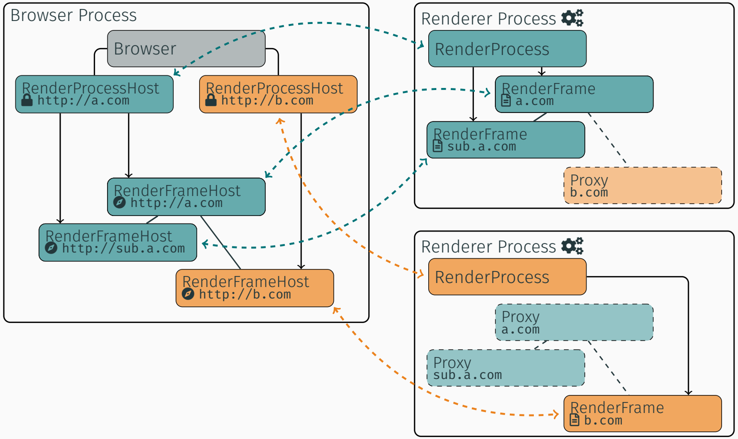
Site Isolation Architecture



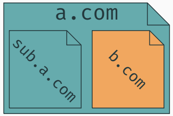

Browser Process
Browser
RenderProcessHost http://a.com
RenderProcessHost http://b.com
RenderFrameHost http://a.com
RenderFrameHost http://sub.a.com
Renderer Process
RenderProcess
RenderFrame a.com
RenderFrame sub.a.com
Renderer Process
RenderProcess

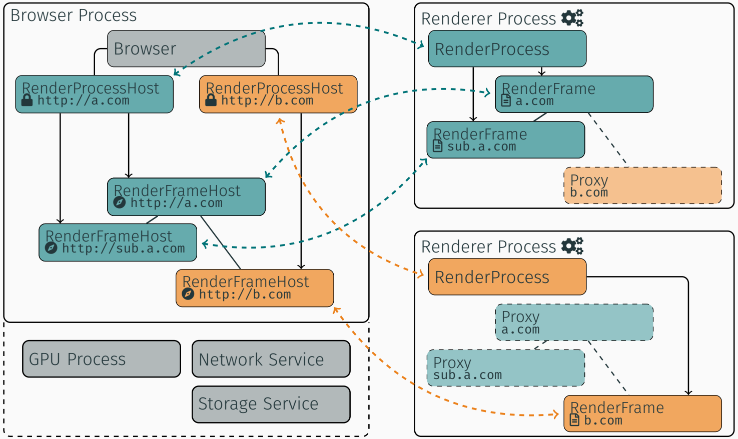
Site Isolation Architecture





Site Isolation Architecture



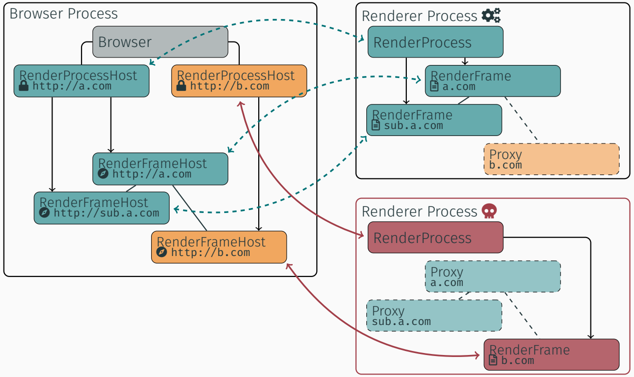


Site Isolation Bypass

Causes:

1. Missing Security Checks
2. Bypassed Security Checks
3. Origin Confusion

→ Semantic bug

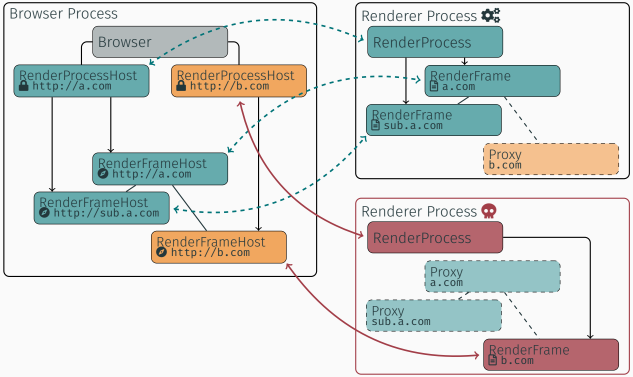


Site Isolation Bypass

Causes:

1. Missing Security Checks
2. Bypassed Security Checks
3. Origin Confusion

→ Semantic bug

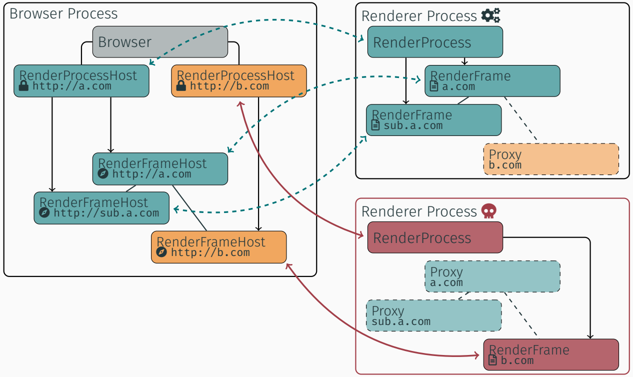


Site Isolation Bypass

Causes:

1. Missing Security Checks
2. Bypassed Security Checks
3. Origin Confusion

→ Semantic bug

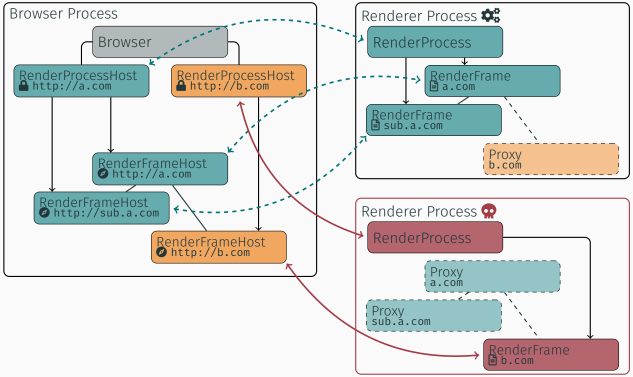


Site Isolation Bypass

Causes:

1. Missing Security Checks
2. Bypassed Security Checks
3. Origin Confusion

→ Semantic bug

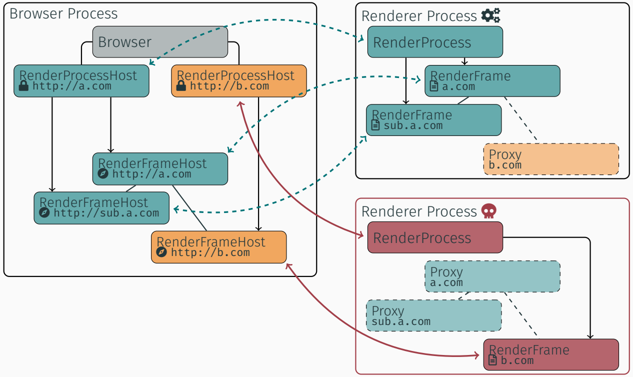


Site Isolation Bypass

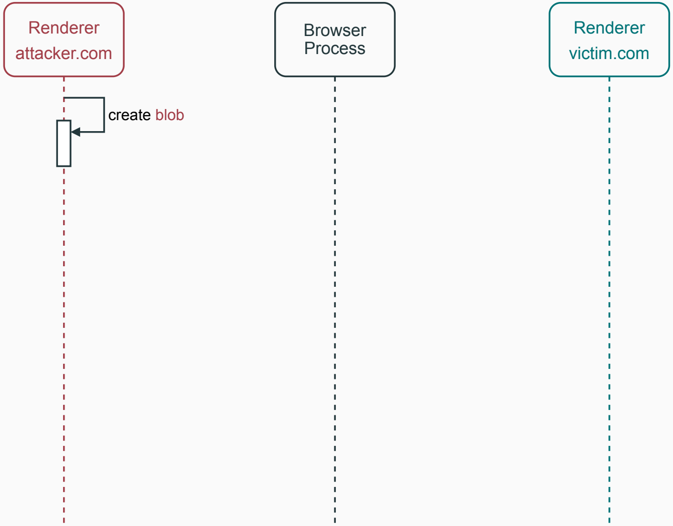
Causes:

1. Missing Security Checks
2. Bypassed Security Checks
3. Origin Confusion

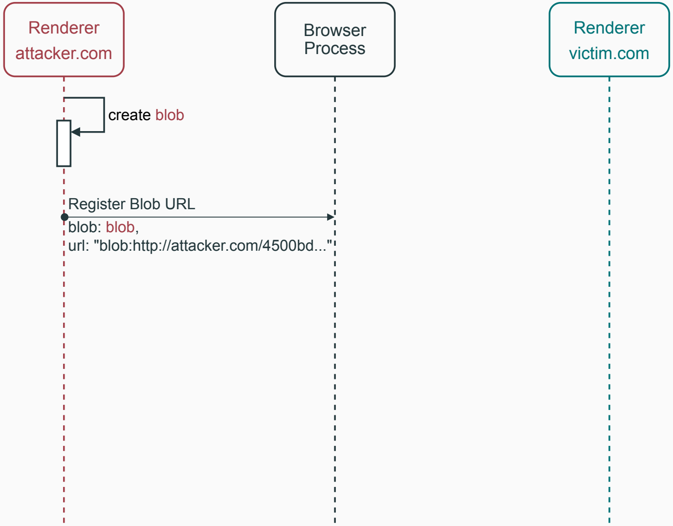
→ Semantic bug



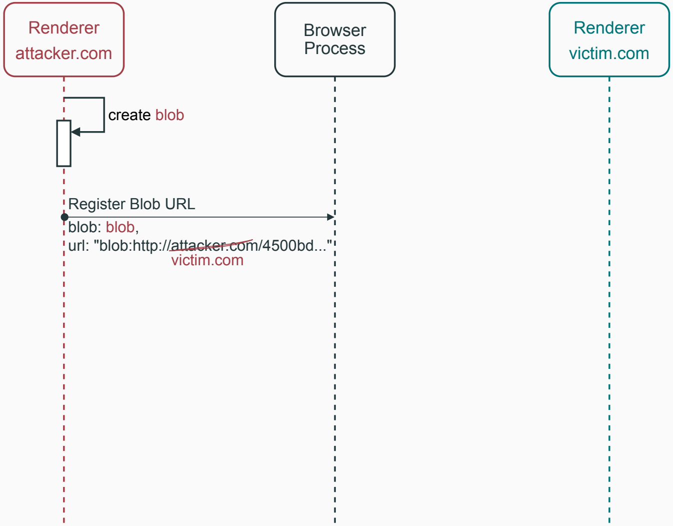
SI Bypass Example: CVE-2018-18345



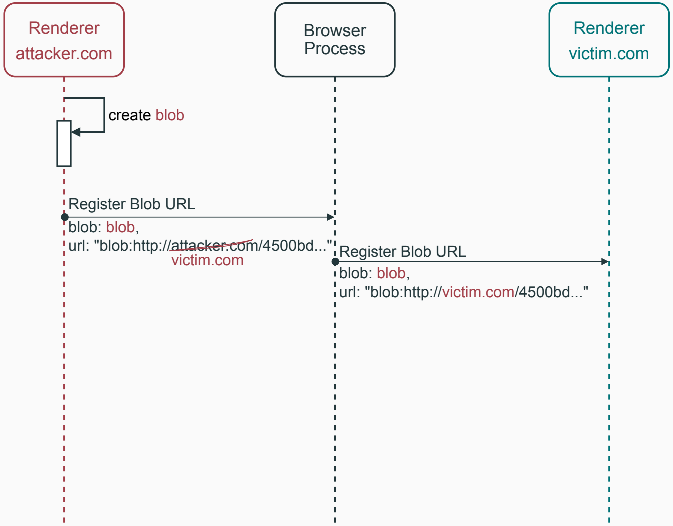
SI Bypass Example: CVE-2018-18345



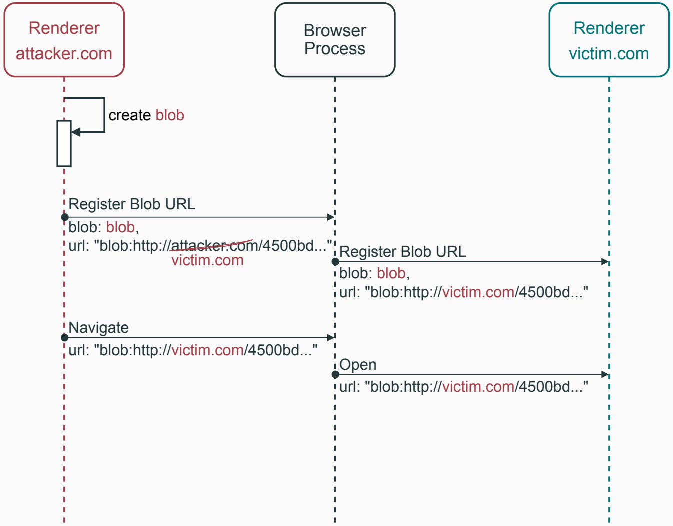
SI Bypass Example: CVE-2018-18345



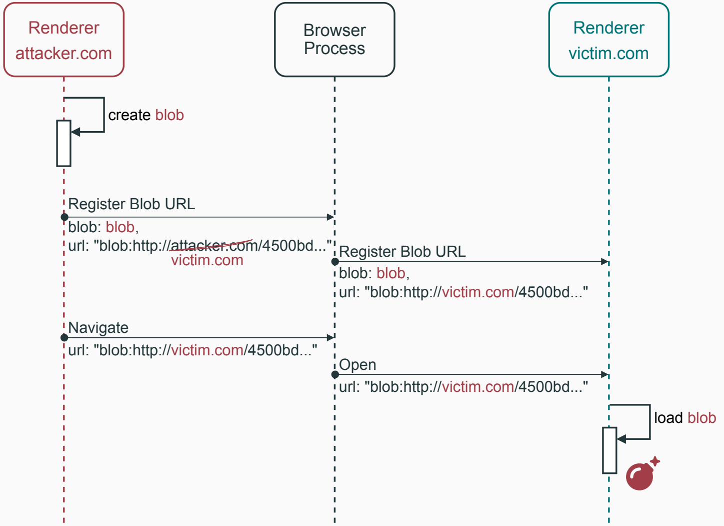
SI Bypass Example: CVE-2018-18345



SI Bypass Example: CVE-2018-18345


Renderer
attacker.com
Browser Process
Renderer
victim.com
create blob
Register Blob URL
blob: blob,
url: "blob:http://~~attacker.com~~/4500bd..."
victim.com
Register Blob URL
blob: blob,
url: "blob:http://victim.com/4500bd..."
Navigate
url: "blob:http://victim.com/4500bd..."
Open
url: "blob:http://victim.com/4500bd..."

SI Bypass Example: CVE-2018-18345


Renderer
attacker.com
Browser Process
Renderer
victim.com
create blob
Register Blob URL
blob: blob,
url: "blob:http://~~attacker.com~~/4500bd..."
victim.com
Register Blob URL
blob: blob,
url: "blob:http://victim.com/4500bd..."
Navigate
url: "blob:http://victim.com/4500bd..."
Open
url: "blob:http://victim.com/4500bd..."
load blob

Design

1. Detection of SI bypass bugs
   ➞ detection at runtime
   ➞ Process Sanitizer & Leak Sanitizer
2. Cover all APIs / IPC interactions
   ➞ WebIDL-based Grammar
3. Complex navigations to trigger Origin Confusion
   ➞ favor navigation API
4. Simulate compromised renderer process
   ➞ mutate IPC messages

Design

1. Detection of SI bypass bugs
   ➞ detection at runtime
   ➞ Process Sanitizer & Leak Sanitizer
2. Cover all APIs / IPC interactions
   ➞ WebIDL-based Grammar
3. Complex navigations to trigger Origin Confusion
   ➞ favor navigation API
4. Simulate compromised renderer process
   ➞ mutate IPC messages

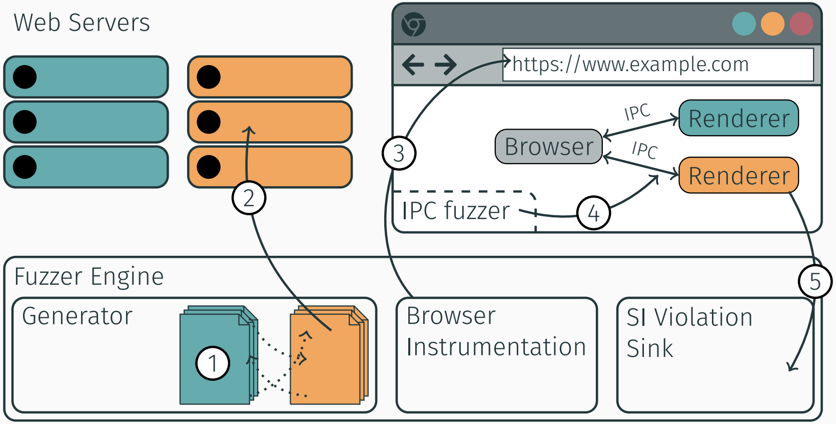
Design

1. Detection of SI bypass bugs
   ➞ detection at runtime
   ➞ Process Sanitizer & Leak Sanitizer
2. Cover all APIs / IPC interactions
   ➞ WebIDL-based Grammar
3. Complex navigations to trigger Origin Confusion
   ➞ favor navigation API
4. Simulate compromised renderer process
   ➞ mutate IPC messages

Design

1. Detection of SI bypass bugs
   ➞ detection at runtime
   ➞ Process Sanitizer & Leak Sanitizer
2. Cover all APIs / IPC interactions
   ➞ WebIDL-based Grammar
3. Complex navigations to trigger Origin Confusion
   ➞ favor navigation API
4. Simulate compromised renderer process
   ➞ mutate IPC messages

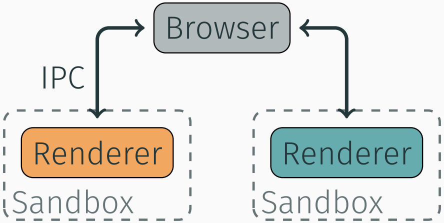
Site Isolation Bypass Fuzzing



Process Sanitizer

→ detect cross-site reuse of renderers

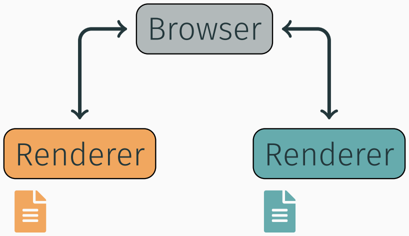
1. input documents contain correct site
2. tag renderer process with site
3. compare document and tag



Process Sanitizer

→ detect cross-site reuse of renderers

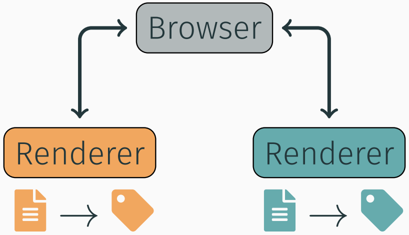
1. input documents contain correct site
2. tag renderer process with site
3. compare document and tag



Process Sanitizer

→ detect cross-site reuse of renderers

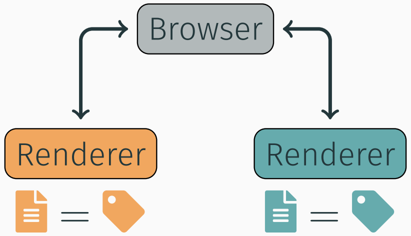
1. input documents contain correct site
2. tag renderer process with site
3. compare document and tag



Process Sanitizer

→ detect cross-site reuse of renderers

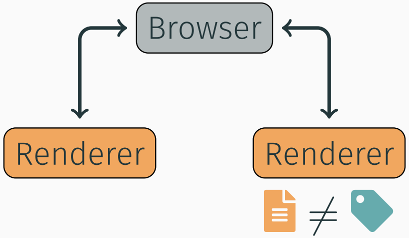
1. input documents contain correct site
2. tag renderer process with site
3. compare document and tag



Process Sanitizer

→ detect cross-site reuse of renderers

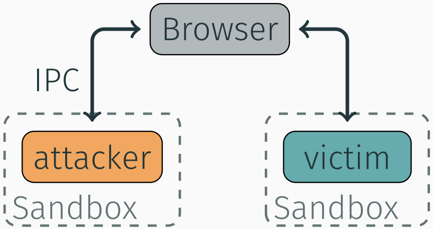
1. input documents contain correct site
2. tag renderer process with site
3. compare document and tag



Leak Sanitizer

→ detect data leaks across renderers

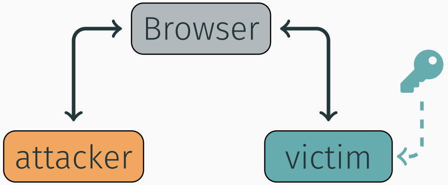
1. inject secret string in victim context
2. victim data leaked to attacker
3. detect secret string in ipc messages



Leak Sanitizer

→ detect data leaks across renderers

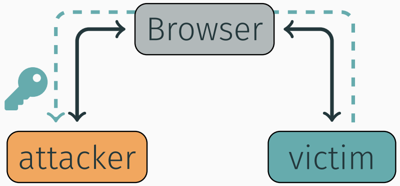
1. inject secret string in victim context
2. victim data leaked to attacker
3. detect secret string in ipc messages



Leak Sanitizer

→ detect data leaks across renderers

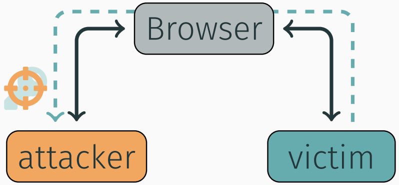
1. inject secret string in victim context
2. victim data leaked to attacker
3. detect secret string in ipc messages



Leak Sanitizer

→ detect data leaks across renderers

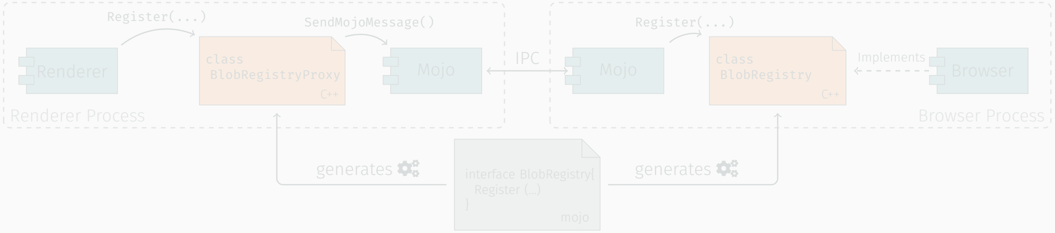
1. inject secret string in victim context
2. victim data leaked to attacker
3. detect secret string in ipc messages



IPC Hooks

- We want...
  - to mutate all IPC messages
  - little changes to the browser code

→ Patch the IPC interface generation

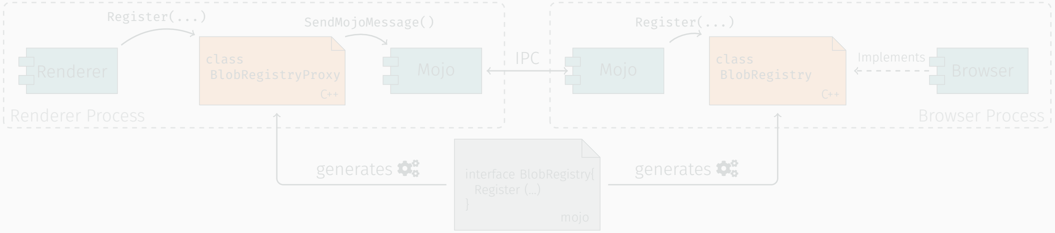


C++ IPC bindings generation in Chrome

IPC Hooks

- We want...
  - to mutate all IPC messages
  - little changes to the browser code

→ Patch the IPC interface generation

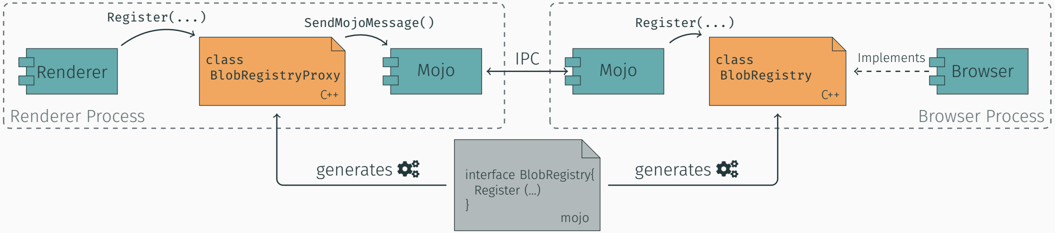


C++ IPC bindings generation in Chrome

IPC Hooks

- We want...
  - to mutate all IPC messages
  - little changes to the browser code

→ Patch the IPC interface generation



C++ IPC bindings generation in Chrome

IPC Fuzzer JavaScript API

- How to sync JS generation and IPC mutations?

→ JavaScript API to enqueue mutations

→ Reproducible crashes

IPC Fuzzer JavaScript API

- How to sync JS generation and IPC mutations?

→ JavaScript API to enqueue mutations

→ Reproducible crashes

IPC Fuzzer JavaScript API

- How to sync JS generation and IPC mutations?

→ JavaScript API to enqueue mutations

→ Reproducible crashes

IPC Fuzzer JavaScript API

- How to sync JS generation and IPC mutations?

→ JavaScript API to enqueue mutations

→ Reproducible crashes

```
let text = `<html><body><script>
            var src = "http://attacker.com";
            IPCFuzzer.check_isolation(src);
            <\/script></body></html>`;
```

IPC Fuzzer JavaScript API

- How to sync JS generation and IPC mutations?

→ JavaScript API to enqueue mutations

→ Reproducible crashes

```
let text = `<html><body><script>
            var src = "http://attacker.com";
            IPCFuzzer.check_isolation(src);
            <\/script></body></html>`;
var blob = new Blob([text], { type: "text/html" });
```

IPC Fuzzer JavaScript API

- How to sync JS generation and IPC mutations?

→ JavaScript API to enqueue mutations

→ Reproducible crashes

```
let text = `<html><body><script>
            var src = "http://attacker.com";
            IPCFuzzer.check_isolation(src);
            <\/script></body></html>`;
var blob = new Blob([text], { type: "text/html" });

IPCFuzzer.mutate_url_replace_host("http://victim.com");
var url = URL.createObjectURL(blob);
```

IPC Fuzzer JavaScript API

- How to sync JS generation and IPC mutations?

→ JavaScript API to enqueue mutations

→ Reproducible crashes

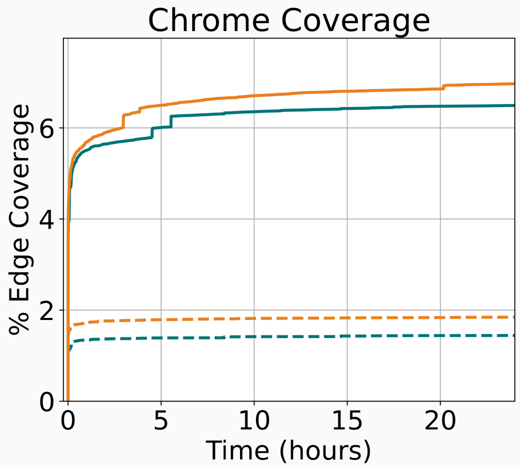
```
let text = `<html><body><script>
            var src = "http://attacker.com";
            IPCFuzzer.check_isolation(src);
            <\/script></body></html>`;
var blob = new Blob([text], { type: "text/html" });

IPCFuzzer.mutate_url_replace_host("http://victim.com");
var url = URL.createObjectURL(blob);

url = url.replace("attacker.com", "victim.com");
location.href = url;
```

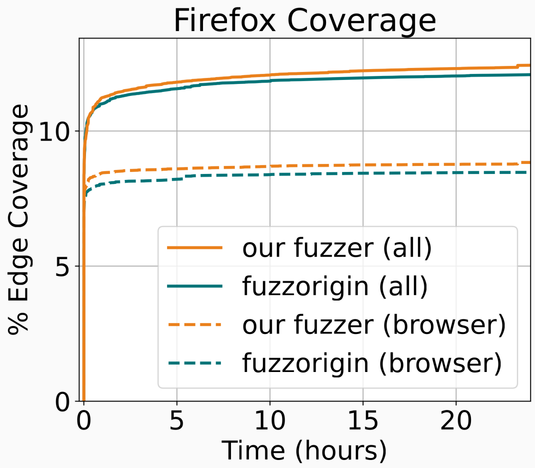
Findings

Browser	Description	Tracker
●	renderer can spoof URL in `history.replaceState`	CVE-2024-9392
◎	`Window.name` leaks on navigation	#384781865†
●	visited URLs are leaked for link colouring	#1938107
●	Cross-Origin-Read-Blocking (CORB) missing	#1532642‡

† Marked Duplicate
‡ Known issue

Coverage Evaluation



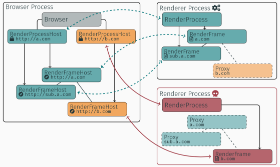


Site Isolation Bypass

Causes:

1. Missing Security Checks
2. Bypassed Security Checks
3. Origin Confusion

→ Semantic bug



Site Isolation Bypass Fuzzing

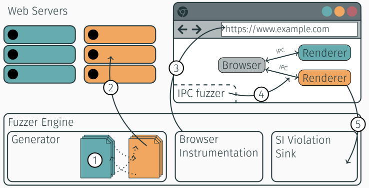
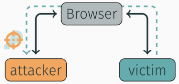


Figure 2: SI Bypass Fuzzer Architecture

Leak Sanitizer

→ detect data leaks across renderers

1. inject secret string in victim context
2. victim data leaked to attacker
3. detect secret string in ipc messages



Findings

Table 1: SI Bypass Fuzzer Findings

Browser	Description	Tracker
●	renderer can spoof URL in `history.replaceState`	CVE-2024-9392
●	`Window.name` leaks on navigation	#384781865†
●	visited URLs are leaked for link colouring	#1938107
●	Cross-Origin-Read-Blocking (CORB) missing	#1532642‡

† Marked Duplicate
‡ Known issue

Questions?

Full Paper:

jan.drescher@tu-braunschweig.de

jan-niklas-drescher-5968081

@jndre@infosec.exchange

References

Reis, Charles, Alexander Moshchuk, and Nasko Oskov. “Site Isolation: Process Separation for Web Sites within the Browser.”. In: 2019, pp. 1661–1678.

Findings: CVE-2024-9392

```
IPCFuzzer.activate_leak_sanitizer();
IPCFuzzer.mutate_url("http://127.0.0.2:8080/victim.html");
window.history.replaceState("foo","", null);
window.location.reload();
```

Proof-of-Concept for Firefox History Confusion

Fuzzer Evaluation

Reproduction on known vulnerabilities

Vulnerability	Chrome Version		Class	Reproduction Time
	Vulnerable	Evaluated		
CVE-2022-1637	< 101.0.4951.64	99.0.4844.84	3	14 minutes
CVE-2019-5856	< 76.0.3809.87	67.0.3396.99	1	1 minute
CVE-2018-18345	< 71.0.3578.80	67.0.3396.99	1	11.4 hours

Oracle Evaluation

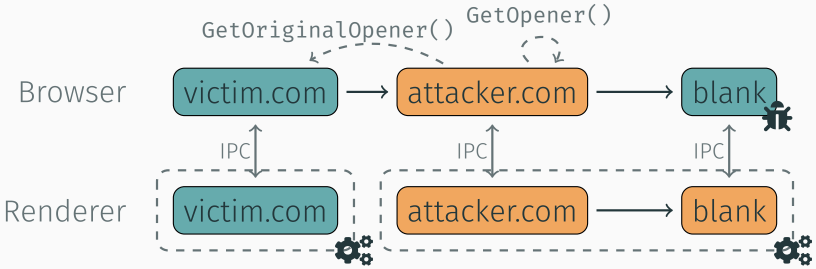
- Add sanitizer to vulnerable browser
- Test if sanitizer detects the PoC exploit

Oracle Evaluation on known PoC's

ID	Class	LeakSan	ProcessSan
CVE-2018-16074	3	○	●
CVE-2019-5773	1	●	○
#40093844	2	●	○
CVE-2024-1671	3	●	○
CVE-2022-3044	1	○	○

SI Bypass Example: CVE-2022-1637

```
let win = window.open('data:,hello', '_blank');
// manipulate IPC message
console.log('Exfiltrated cookies: ' + win.document.cookie);
```



Chrome SI bypass caused by Origin Confusion

Known SI Bypass Vulnerabilities

ID	Description	Class	In Scope
CVE-2024-1671	Origin confusion in session history leaks URL of srcdoc iframe	3	●
CVE-2022-4913	Compromised renderer can access extension storage	1	◑
CVE-2022-3661	Compromised renderer can message any extension content script	1	◑
CVE-2022-3044	No access checks for clipboard interface	1	●
CVE-2022-1637	Cross-origin iframe can spoof the hostname of top-frame by opening new window with `javascript:` URI and target `_blank`	3	●
CVE-2022-0305	Hidden bug report for Service Worker	?	?
CVE-2022-0294	No checks in PushMessaging interface that verify if the referenced ServiceWorker belongs to the same origin as the renderer	1	●
CVE-2022-0292	Fenced frame can open textttfile: URLs	1	●
CVE-2022-0291	Hidden bug report for storage	?	?
#40060671	Compromised renderer can spoof PortContext and claim to be WorkerContext of arbitrary extension	1	◑
CVE-2021-38010	`URLLoader` leaked to ServiceWorker, compromised renderer can read the response of redirected cross-origin requests	1	○
CVE-2021-30507	Compromised renderer can spoof texttX-Chrome-offline header to read arbitrary file	1	○
CVE-2021-21222	TOCTOU bug in `GeneratedCodeCache`: compromised renderer can change value after the hash computation	2	○
CVE-2021-21175	`X-Frame-Options` error of cross-origin iframe is leaked to parent	1	●
#40054801	Compromised renderer that outlives state in the browser process can bypass security checks to spoof origin	2	●
CVE-2020-6435	Compromised renderer can spoof sender id to extension	1	◑
CVE-2020-6385	Origin checks in `BlobURLStoreImpl::Register` skipped if renderer process simulates detachment	2	●
CVE-2020-6380	Compromised renderer can spoof origin, message any extension	1	◑
CVE-2019-13763	Compromised renderer can spoof origin and leak data from textttPaymentManager	1	●
CVE-2019-13738	Sandboxed iframe shares execution context with initial non-sandboxed `about:blank` frame	3	●
CVE-2019-13727	Compromised renderer can create WebSocket to arbitrary URL and leak the response headers	1	●
CVE-2019-13682	Spoofing origin in protocol handler registration leads to SI bypass	1	●
CVE-2019-5865	CORS bypass: compromised renderer can set `Host` header during redirect	1	○
CVE-2019-5862	Compromised renderer can spoof `document_url_` and register arbitrary files from victims site in AppChache	1	○
CVE-2019-5856	Missing browser-side checks, compromised renderer can access filesystem of other origins	1	●
CVE-2019-5773	Compromised renderer can spoof origin when accessing IndexedDB	1	●
#40093845	Compromised renderer can spoof origin and access code cache of other site	1	●
#40093844	Invalid checks on texttttws: URLs, compromised renderer can leak cookies	2	●
CVE-2018-18345	`BlobURLRegistry::RegisterURL` access check based on renderer provided `host` and `public_url`	1	●
CVE-2018-16074	BlobURLs created from different opaque origins have opaque origin but are all handled in the same process	3	●
CVE-2018-16073	Data URL in iframe is loaded in same process if embedding page is loaded from cache	3	●
CVE-2018-6165	Refresh during navigation triggers origin confusion	3	●
CVE-2018-6121	Compromised renderer can commit url of extension	1	●
#40092826	Cookies leaked to cross-site renderer in presence of DevTools	1	●
#40092525	Compromised renderer can spoof origin during filesystem url creation	1	●

Mojo IDL Example

```
module blink.mojom;

import "mojo/public/mojom/base/unguessable_token.mojom";
import "services/network/public/mojom/url_loader_factory.mojom";
import "third_party/blink/public/mojom/blob/blob.mojom";
import "url/mojom/url.mojom";

interface BlobURLStore {
  // TODO(https://crbug.com/1376126): This should probably create and return a
  // new blob: URL rather than letting the caller in the renderer provide one.
  [Sync] Register(
      pending_remote<blink.mojom.Blob> blob,
      url.mojom.Url url) => ();

  Revoke(url.mojom.Url url);

  ResolveAsURLLoaderFactory(
    url.mojom.Url url,
    pending_receiver<network.mojom.URLLoaderFactory> factory);

  ResolveAsBlobURLToken(url.mojom.Url url,
                        pending_receiver<BlobURLToken> token,
                        bool is_top_level_navigation);
};
```

Firefox IPDL Example

```
struct BlobURLRegistrationData
{
  nsCString url;
  IPCBlob blob;
  nsIPrincipal principal;
  nsCString partitionKey;
  bool revoked;
};

sync protocol PContent
{
parent:
     async StoreAndBroadcastBlobURLRegistration(nsCString url, IPCBlob blob,
                                               nullable nsIPrincipal principal, nsCString aPartitionKey);
child:
    async BlobURLRegistration(nsCString aURI, IPCBlob aBlob,
                              nullable nsIPrincipal aPrincipal, nsCString aPartitionKey);

}
```

Excerpt of PContent.ipdl