Site Isolation:
Process Separation for Web Sites within the Browser

Charlie Reis, Alex Moshchuk, Nasko Oskov
Google
Protecting Web Sites against Strong Attackers

- Rendering engine vulnerabilities are common
- Spectre / transient execution attacks work in the browser

- **Shipped Site Isolation to all Chrome desktop users as mitigation**
  - Overcame challenges beyond prior research browsers
  - Practical to deploy: compatibility, performance
  - Some limitations, but offers the best path to protection
Multi-Process Web Browsers
1. Renderer Exploit Attacker

Browser Process

Renderer Process

Sandbox

evil.com

youtube.com
2. Memory Disclosure Attacker
Site Isolation
Site Isolation Architecture

Site-Dedicated Processes

Browser Process

Renderer Process: evil.com
- evil.com

Renderer Process: youtube.com
- youtube.com

Cross-Origin Read Blocking (CORB)

foo.com
- Cross-site images, scripts

- Cross-site data

foo.com
Out-of-process iframes

- Challenging to support web platform
  - Secure compositing
  - Frame proxies
  - State replication
  - Many affected features
    (e.g., find-in-page)
Cross-Origin Read Blocking

- Must allow subresources
- Want to protect sensitive data (HTML, XML, JSON)
- Mislabeled Content-Types
  - Custom sniffing
  - Must allow responses like:

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Example</title>
  </head>
  <body>
    <!-- This is JS. -->
    <script>
      function a() {...}
    </script>
    <img src="bar.com/image.jpg">
    <img src="bar.com/secret.html">
  </body>
</html>
```

Content-Type: text/html

foo.com
Enforcements

- Catch malicious IPC messages
  - Limit access to site data
  - Terminate misbehaving processes
- Matters for renderer exploits
Evaluation
Mitigating Renderer Exploits

- Renderer vulnerabilities matter in practice
  - 94 UXSS-like bugs in 2014-2018

- Web developer practices now robust to renderer exploits:
  - Authentication
  - Confidential data in HTML/XML/JSON
  - Cross-Origin Messaging
  - Anti-Clickjacking
  - Use of storage and permissions
Transient Execution Attacks: Mitigation Strategies

1. Remove precise timers (e.g., SharedArrayBuffers)
   - Not effective: Coarse timers can be amplified
   - Harmful to Web Platform

2. Compiler/Runtime mitigations
   - Not effective: Can’t handle all variants

3. Site Isolation
   - Put data worth stealing out of reach
   - Effective for **same-process** variants
   - Combine with OS/HW mitigations for cross-process
Addressing Limitations

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

- Many data types are not yet protected
  - Opt-in header, more CORB-protected types, SameSite cookie defaults

- Cross-process transient execution attacks (e.g., MDS)
  - Combine with OS/HW mitigations

- Not yet deployed on mobile devices
  - Preparing to isolate a subset of sites on Android
Practical to Deploy

- **Performance Optimizations**
  - Reduced potential process count and total memory overhead
  - Reduced latency for navigations and input

![Graph showing performance optimization results](image)
Conclusion

- Transient execution attacks change the web threat model

- Site Isolation offers best path to protection
  - Don’t leak data to renderer exploits or Spectre attacks
  - Practical to deploy to all Chrome desktop users
  - Need to push further to protect more types of data

- Other systems may want to revisit their architectures
  - Not safe to run untrustworthy code in same process as sensitive data