前面三篇文章我们主要围绕同源策略介绍了Web页面安全的相关内容，那今天我们把视野向外延伸，来聊聊页面安全和操作系统安全之间的关系。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中，我们分析了浏览器架构的发展史，在最开始的阶段，浏览器是单进程的，这意味着渲染过程、JavaScript执行过程、网络加载过程、UI绘制过程和页面显示过程等都是在同一个进程中执行的，这种结构虽然简单，但是也带来了很多问题。

从稳定性视角来看，单进程架构的浏览器是不稳定的，因为只要浏览器进程中的任意一个功能出现异常都有可能影响到整个浏览器，如页面卡死、浏览器崩溃等。不过浏览器的稳定性并不是本文讨论的重点，我们今天主要聊的是**浏览器架构是如何影响到操作系统安全的**。

浏览器本身的漏洞是单进程浏览器的一个主要问题，如果浏览器被曝出存在漏洞，那么在这些漏洞没有被及时修复的情况下，黑客就有可能通过恶意的页面向浏览器中注入恶意程序，其中最常见的攻击方式是利用**缓冲区溢出**，不过需要**注意这种类型的攻击和XSS注入的脚本是不一样的**。

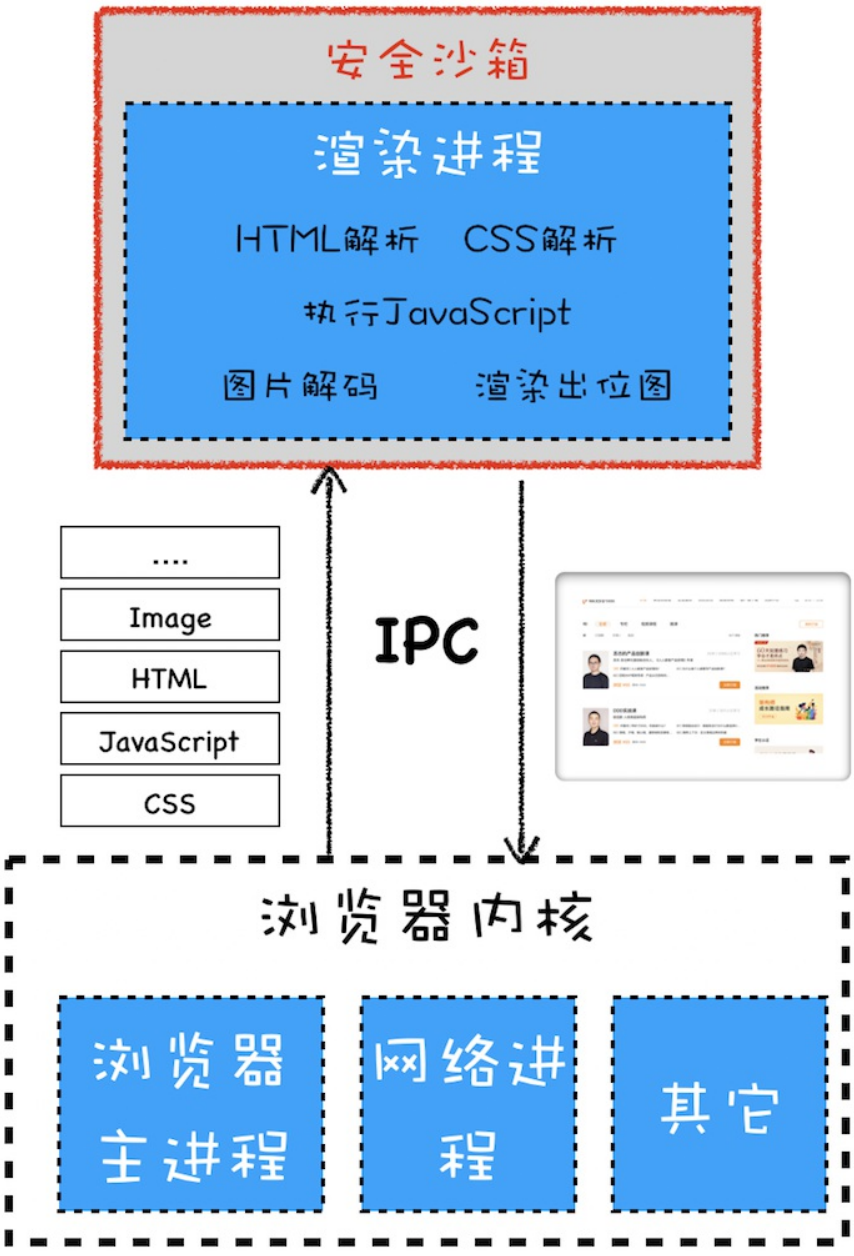
- XSS攻击只是将恶意的JavaScript脚本注入到页面中，虽然能窃取一些Cookie相关的数据，但是XSS无法对操作系统进行攻击。
- 而通过浏览器漏洞进行的攻击是可以入侵到浏览器进程内部的，可以读取和修改浏览器进程内部的任意内容，还可以穿透浏览器，在用户的操作系统上悄悄地安装恶意软件、监听用户键盘输入信息以及读取用户硬盘上的文件内容。

和XSS攻击页面相比，这类攻击无疑是枚“核弹”，它会将整个操作系统的内容都暴露给黑客，这样我们操作系统上所有的资料都是不安全的了。

安全视角下的多进程架构

现代浏览器的设计目标是**安全、快速**和**稳定**，而这种核弹级杀伤力的安全问题就是一个很大的潜在威胁，因此在设计现代浏览器的体系架构时，需要解决这个问题。

我们知道现代浏览器采用了多进程架构，将渲染进程和浏览器主进程做了分离，完整的进程架构我们已经在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中介绍过了，这里我就不重复介绍了。下面我们重点从操作系统安全的视角来看看浏览器的多进程架构，如下图：



浏览器内核和渲染进程

观察上图，我们知道浏览器被划分为**浏览器内核**和**渲染内核**两个核心模块，其中浏览器内核是由网络进程、浏览器主进程和GPU进程组成的，渲染内核就是渲染进程。那如果我们在浏览器中打开一个页面，这两个模块是怎么配合的呢？

所有的网络资源都是通过浏览器内核来下载的，下载后的资源会通过IPC将其提交给渲染进程（浏览器内核和渲染进程之间都是通过IPC来通信的）。然后渲染进程会对这些资源进行解析、绘制等操作，最终生成一幅图片。但是渲染进程并不负责将图片显示到界面上，而是将最终生成的图片提交给浏览器内核模块，由浏览器内核模块负责显示这张图片。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》中我们分析过，设计现代浏览器体系架构时，将浏览器划分为不同的进程是为了增加其稳定性。虽然设计成了多进程架构，不过这些模块之间的沟通方式却有些复杂，也许你还有以下问题：

- 为什么一定要通过浏览器内核去请求资源，再将数据转发给渲染进程，而不直接从进程内部去请求网络资源？
- 为什么渲染进程只负责生成页面图片，生成图片还要经过IPC通知浏览器内核模块，然后让浏览器内核去负责展示图片？

通过以上方式不是增加了工程的复杂度吗？

要解释现代浏览器为什么要把这个流程弄得这么复杂，我们就得从系统安全的角度来分析。

安全沙箱

不过在解释这些问题之前，我们得先看看什么是安全沙箱。

上面我们分析过了，由于渲染进程需要执行DOM解析、CSS解析、网络图片解码等操作，如果渲染进程中存在系统级别的漏洞，那么以上操作就有可能让恶意的站点获取到渲染进程的控制权限，进而又获取操作系统的控制权限，这对于用户来说是非常危险的。

因为网络资源的内容存在着各种可能性，所以浏览器会默认所有的网络资源都是不可信的，都是不安全的。但谁也不能保证浏览器不存在漏洞，只要出现漏洞，黑客就可以通过网络内容对用户发起攻击。

我们知道，如果你下载了一个恶意程序，但是没有执行它，那么恶意程序是不会生效的。同理，浏览器之于网络内容也是如此，浏览器可以安全地下载各种网络资源，但是如果要执行这些网络资源，比如解析HTML、解析CSS、执行JavaScript、图片编解码等操作，就需要非常谨慎了，因为一不小心，黑客就会利用这些操作对含有漏洞的浏览器发起攻击。

基于以上原因，我们需要在渲染进程和操作系统之间建一道墙，即便渲染进程由于存在漏洞被黑客攻击，但由于这道墙，黑客就获取不到渲染进程之外的任何操作权限。**将渲染进程和操作系统隔离的这道墙就是我们要聊的安全沙箱**。

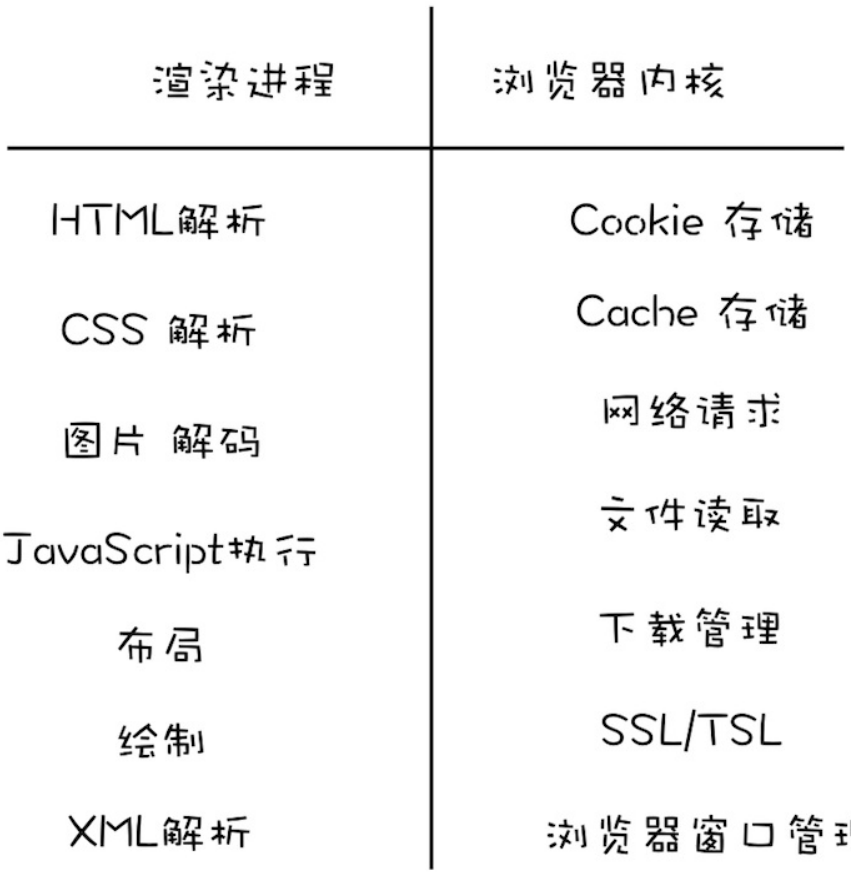
浏览器中的安全沙箱是利用操作系统提供的安全技术，让渲染进程在执行过程中无法访问或者修改操作系统中的数据，在渲染进程需要访问系统资源的时候，需要通过浏览器内核来实现，然后将访问的结果通过IPC转发给渲染进程。

安全沙箱最小的保护单位是进程。因为单进程浏览器需要频繁访问或者修改操作系统的数据，所以单进程浏览器是无法被安全沙箱保护的，而现代浏览器采用的多进程架构使得安全沙箱可以发挥作用。

安全沙箱如何影响各个模块功能

我们知道安全沙箱最小的保护单位是进程，并且能限制进程对操作系统资源的访问和修改，这就意味着如果要让安全沙箱应用在某个进程上，那么这个进程必须没有读写操作系统的功能，比如读写本地文件、发起网络请求、调用GPU接口等。

了解了被安全沙箱保护的进程会有一系列的受限操作之后，接下来我们就可以分析渲染进程和浏览器内核各自都有哪些职责，如下图：



浏览器内核和渲染进程各自职责

通过该图，我们可以看到由于渲染进程需要安全沙箱的保护，因此需要把在渲染进程内部涉及到和系统交互的功能都转移到浏览器内核中去实现。

那安全沙箱是如何影响到各个模块功能的呢？

1. 持久存储

我们先来看看安全沙箱是如何影响到浏览器持久存储的。由于安全沙箱需要负责确保渲染进程无法直接访问用户的文件系统，但是在渲染进程内部有访问Cookie的需求、有上传文件的需求，为了解决这些文件的访问需求，所以现代浏览器将读写文件的操作全部放在了浏览器内核中实现，然后通过IPC将操作结果转发给渲染进程。

具体地讲，如下文件内容的读写都是在浏览器内核中完成的：

- 存储Cookie数据的读写。通常浏览器内核会维护一个存放所有Cookie的Cookie数据库，然后当渲染进程通过JavaScript来读取Cookie时，渲染进程会通过IPC将读取Cookie的信息发送给浏览器内核，浏览器内核读取Cookie之后再将内容返回给渲染进程。
- 一些缓存文件的读写也是由浏览器内核实现的，比如网络文件缓存的读取。

2. 网络访问

同样有了安全沙箱的保护，在渲染进程内部也是不能直接访问网络的，如果要访问网络，则需要通过浏览器内核。不过浏览器内核在处理URL请求之前，会检查渲染进程是否有权限请求该URL，比如检查XMLHttpRequest或者Fetch是否是跨站点请求，或者检测HTTPS的站点中是否包含了HTTP的请求。

3. 用户交互

渲染进程实现了安全沙箱，还影响到了一个非常重要的用户交互功能。

通常情况下，如果你要实现一个UI程序，操作系统会提供一个界面给你，该界面允许应用程序与用户交互，允许应用程序在该界面上进行绘制，比如Windows提供的是HWND，Linux提供的X Window，我们就把HWND和X Window统称为窗口句柄。应用程序可以在窗口句柄上进行绘制和接收键盘鼠标消息。

不过在现代浏览器中，由于每个渲染进程都有安全沙箱的保护，所以在渲染进程内部是无法直接操作窗口句柄的，这也是为了限制渲染进程监控到用户的输入事件。

由于渲染进程不能直接访问**窗口句柄**，所以渲染进程需要完成以下两点大的改变。

第一点，渲染进程需要渲染出位图。为了向用户显示渲染进程渲染出来的位图，渲染进程需要将生成好的位图发送到浏览器内核，然后浏览器内核将位图复制到屏幕上。

第二点，操作系统没有将用户输入事件直接传递给渲染进程，而是将这些事件传递给浏览器内核。然后浏览器内核再根据当前浏览器界面的状态来判断如何调度这些事件，如果当前焦点位于浏览器地址栏中，则输入事件会在浏览器内核内部处理；如果当前焦点在页面的区域内，则浏览器内核会将输入事件转发给渲染进程。

之所以这样设计，就是为了限制渲染进程有监控到用户输入事件的能力，所以所有的键盘鼠标事件都是由浏览器内核来接收的，然后浏览器内核再通过IPC将这些事件发送给渲染进程。

上面我们分析了由于渲染进程引入了安全沙箱，所以浏览器的持久存储、网络访问和用户交互等功能都不能在渲染进程内直接使用了，因此我们需要把这些功能迁移到浏览器内核中去实现，这让原本比较简单的流程变得复杂了。

理解这些限制，我们就能解释开始提出的两个问题了。

站点隔离（Site Isolation）

所谓站点隔离是指Chrome将同一站点（包含了相同根域名和相同协议的地址）中相互关联的页面放到同一个渲染进程中执行。

最开始Chrome划分渲染进程是以标签页为单位，也就是说整个标签页会被划分给某个渲染进程。但是，按照标签页划分渲染进程存在一些问题，原因就是一个标签页中可能包含了多个iframe，而这些iframe又有可能来自于不同的站点，这就导致了多个不同站点中的内容通过iframe同时运行在同一个渲染进程中。

目前所有操作系统都面临着两个A级漏洞——幽灵（Spectre）和熔毁（Meltdown），这两个漏洞是由处理器架构导致的，很难修补，黑客通过这两个漏洞可以直接入侵到进程的内部，如果入侵的进程没有安全沙箱的保护，那么黑客还可以发起对操作系统的攻击。

所以如果一个银行站点包含了一个恶意iframe，然后这个恶意的iframe利用这两个A级漏洞去入侵渲染进程，那么恶意程序就能读取银行站点渲染进程内的所有内容了，这对于用户来说就存在很大的风险了。

因此Chrome几年前就开始重构代码，将标签级的渲染进程重构为iframe级的渲染进程，然后严格按照同一站点的策略来分配渲染进程，这就是Chrome中的站点隔离。

实现了站点隔离，就可以将恶意的iframe隔离在恶意进程内部，使得它无法继续访问其他iframe进程的内容，因此也就无法攻击其他站点了。

值得注意是，2019年10月20日Chrome团队宣布安卓版的Chrome已经全面支持站点隔离，你可以参考文中链接。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容。

首先我们分析了单进程浏览器在系统安全方面的不足，如果浏览器存在漏洞，那么黑客就有机会通过页面对系统发起攻击。

因此在设计现代浏览器的体系架构时，就考虑到这个问题了。于是，在多进程的基础之上引入了安全沙箱，有了安全沙箱，就可以将操作系统和渲染进程进行隔离，这样即便渲染进程由于漏洞被攻击，也不会影响到操作系统的。

由于渲染进程采用了安全沙箱，所以在渲染进程内部不能与操作系统直接交互，于是就在浏览器内核中实现了持久存储、网络访问和用户交互等一系列与操作系统交互的功能，然后通过IPC和渲染进程进行交互。

最后我们还分析了Chrome中最新的站点隔离功能。由于最初都是按照标签页来划分渲染进程的，所以如果一个标签页里面有多个不同源的iframe，那么这些iframe也会被分配到同一个渲染进程中，这样就很容易让黑客通过iframe来攻击当前渲染进程。而站点隔离会将不同源的iframe分配到不同的渲染进程中，这样即使黑客攻击恶意iframe的渲染进程，也不会影响到其他渲染进程的。

今天介绍的内容和概念都比较多，看上去离前端比较远，不过这些内容会影响你对浏览器整体架构的理解，而深入理解了浏览器架构能帮助你更加深刻地理解前端内容。为了方便你的理解，我把一些参考资料放到了文章的最后，有需要的话你可以拿来参考。

思考时间

今天留给你的思考题：你认为安全沙箱能防止XSS或者CSRF一类的攻击的吗？为什么？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

参考资料

1. 安全沙箱的设计参考了最小权限原则
2. The Security Architecture of the Chromium Browser
3. The Security Architecture of the Chromium Browser-ppt
4. chromium site-isolation
5. Site Isolation
6. Site Isolation: Process Separation for Web Sites within the Browser

前面三篇文章我们主要围绕同源策略介绍了Web页面安全的相关内容，那今天我们把视野向外延伸，来聊聊页面安全和操作系统安全之间的关系。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中，我们分析了浏览器架构的发展史，在最开始的阶段，浏览器是单进程的，这意味着渲染过程、JavaScript执行过程、网络加载过程、UI绘制过程和页面显示过程等都是在同一个进程中执行的，这种结构虽然简单，但是也带来了很多问题。

从稳定性视角来看，单进程架构的浏览器是不稳定的，因为只要浏览器进程中的任意一个功能出现异常都有可能影响到整个浏览器，如页面卡死、浏览器崩溃等。不过浏览器的稳定性并不是本文讨论的重点，我们今天主要聊的是**浏览器架构是如何影响到操作系统安全的**。

浏览器本身的漏洞是单进程浏览器的一个主要问题，如果浏览器被曝出存在漏洞，那么在这些漏洞没有被及时修复的情况下，黑客就有可能通过恶意的页面向浏览器中注入恶意程序，其中最常见的攻击方式是利用**缓冲区溢出**，不过需要**注意这种类型的攻击和XSS注入的脚本是不一样的**。

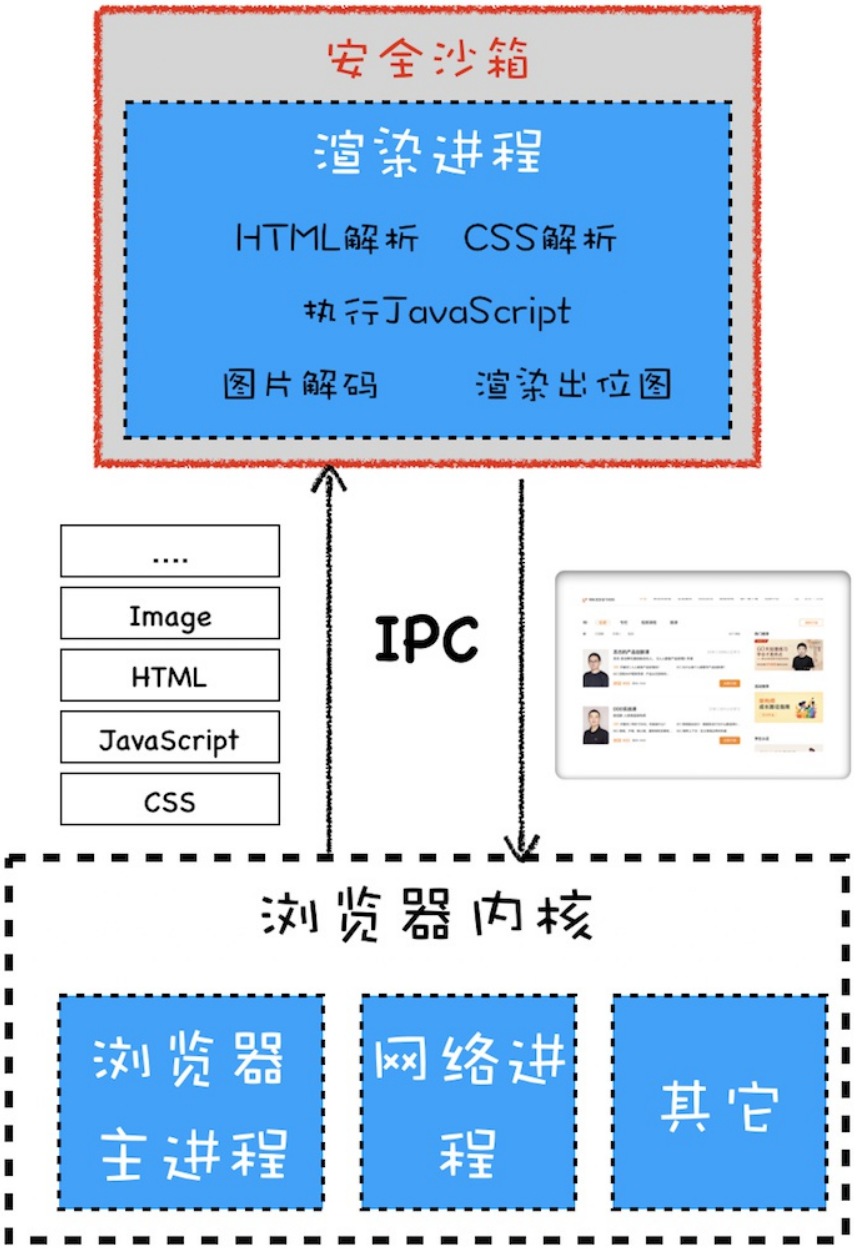
- XSS攻击只是将恶意的JavaScript脚本注入到页面中，虽然能窃取一些Cookie相关的数据，但是XSS无法对操作系统进行攻击。
- 而通过浏览器漏洞进行的攻击是可以入侵到浏览器进程内部的，可以读取和修改浏览器进程内部的任意内容，还可以穿透浏览器，在用户的操作系统上悄悄地安装恶意软件、监听用户键盘输入信息以及读取用户硬盘上的文件内容。

和XSS攻击页面相比，这类攻击无疑是枚“核弹”，它会将整个操作系统的内容都暴露给黑客，这样我们操作系统上所有的资料都是不安全的了。

安全视角下的多进程架构

现代浏览器的设计目标是**安全、快速**和**稳定**，而这种核弹级杀伤力的安全问题就是一个很大的潜在威胁，因此在设计现代浏览器的体系架构时，需要解决这个问题。

我们知道现代浏览器采用了多进程架构，将渲染进程和浏览器主进程做了分离，完整的进程架构我们已经在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中介绍过了，这里我就不重复介绍了。下面我们重点从操作系统安全的视角来看看浏览器的多进程架构，如下图：



浏览器内核和渲染进程

观察上图，我们知道浏览器被划分为**浏览器内核**和**渲染内核**两个核心模块，其中浏览器内核是由网络进程、浏览器主进程和GPU进程组成的，渲染内核就是渲染进程。那如果我们在浏览器中打开一个页面，这两个模块是怎么配合的呢？

所有的网络资源都是通过浏览器内核来下载的，下载后的资源会通过IPC将其提交给渲染进程（浏览器内核和渲染进程之间都是通过IPC来通信的）。然后渲染进程会对这些资源进行解析、绘制等操作，最终生成一幅图片。但是渲染进程并不负责将图片显示到界面上，而是将最终生成的图片提交给浏览器内核模块，由浏览器内核模块负责显示这张图片。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》中我们分析过，设计现代浏览器体系架构时，将浏览器划分为不同的进程是为了增加其稳定性。虽然设计成了多进程架构，不过这些模块之间的沟通方式却有些复杂，也许你还有以下问题：

- 为什么一定要通过浏览器内核去请求资源，再将数据转发给渲染进程，而不直接从进程内部去请求网络资源？
- 为什么渲染进程只负责生成页面图片，生成图片还要经过IPC通知浏览器内核模块，然后让浏览器内核去负责展示图片？

通过以上方式不是增加了工程的复杂度吗？

要解释现代浏览器为什么要把这个流程弄得这么复杂，我们就得从系统安全的角度来分析。

安全沙箱

不过在解释这些问题之前，我们得先看看什么是安全沙箱。

上面我们分析过了，由于渲染进程需要执行DOM解析、CSS解析、网络图片解码等操作，如果渲染进程中存在系统级别的漏洞，那么以上操作就有可能让恶意的站点获取到渲染进程的控制权限，进而又获取操作系统的控制权限，这对于用户来说是非常危险的。

因为网络资源的内容存在着各种可能性，所以浏览器会默认所有的网络资源都是不可信的，都是不安全的。但谁也不能保证浏览器不存在漏洞，只要出现漏洞，黑客就可以通过网络内容对用户发起攻击。

我们知道，如果你下载了一个恶意程序，但是没有执行它，那么恶意程序是不会生效的。同理，浏览器之于网络内容也是如此，浏览器可以安全地下载各种网络资源，但是如果要执行这些网络资源，比如解析HTML、解析CSS、执行JavaScript、图片编解码等操作，就需要非常谨慎了，因为一不小心，黑客就会利用这些操作对含有漏洞的浏览器发起攻击。

基于以上原因，我们需要在渲染进程和操作系统之间建一道墙，即便渲染进程由于存在漏洞被黑客攻击，但由于这道墙，黑客就获取不到渲染进程之外的任何操作权限。**将渲染进程和操作系统隔离的这道墙就是我们要聊的安全沙箱**。

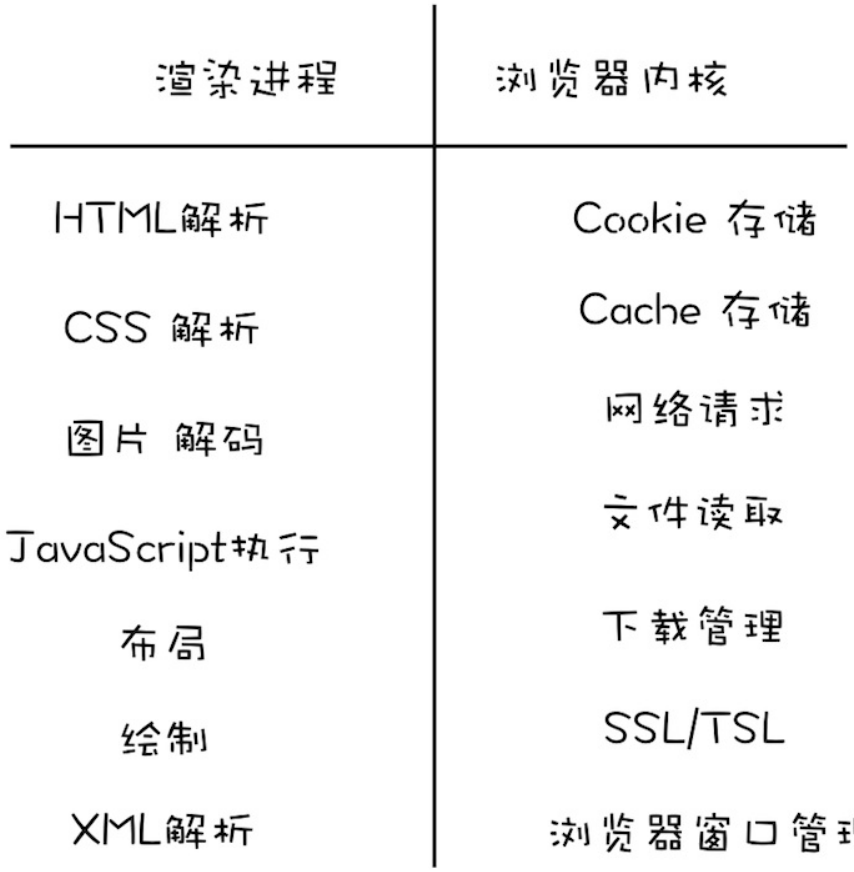
浏览器中的安全沙箱是利用操作系统提供的安全技术，让渲染进程在执行过程中无法访问或者修改操作系统中的数据，在渲染进程需要访问系统资源的时候，需要通过浏览器内核来实现，然后将访问的结果通过IPC转发给渲染进程。

安全沙箱最小的保护单位是进程。因为单进程浏览器需要频繁访问或者修改操作系统的数据，所以单进程浏览器是无法被安全沙箱保护的，而现代浏览器采用的多进程架构使得安全沙箱可以发挥作用。

安全沙箱如何影响各个模块功能

我们知道安全沙箱最小的保护单位是进程，并且能限制进程对操作系统资源的访问和修改，这就意味着如果要让安全沙箱应用在某个进程上，那么这个进程必须没有读写操作系统的功能，比如读写本地文件、发起网络请求、调用GPU接口等。

了解了被安全沙箱保护的进程会有一系列的受限操作之后，接下来我们就可以分析渲染进程和浏览器内核各自都有哪些职责，如下图：



浏览器内核和渲染进程各自职责

通过该图，我们可以看到由于渲染进程需要安全沙箱的保护，因此需要把在渲染进程内部涉及到和系统交互的功能都转移到浏览器内核中去实现。

那安全沙箱是如何影响到各个模块功能的呢？

1. 持久存储

我们先来看看安全沙箱是如何影响到浏览器持久存储的。由于安全沙箱需要负责确保渲染进程无法直接访问用户的文件系统，但是在渲染进程内部有访问Cookie的需求、有上传文件的需求，为了解决这些文件的访问需求，所以现代浏览器将读写文件的操作全部放在了浏览器内核中实现，然后通过IPC将操作结果转发给渲染进程。

具体地讲，如下文件内容的读写都是在浏览器内核中完成的：

- 存储Cookie数据的读写。通常浏览器内核会维护一个存放所有Cookie的Cookie数据库，然后当渲染进程通过JavaScript来读取Cookie时，渲染进程会通过IPC将读取Cookie的信息发送给浏览器内核，浏览器内核读取Cookie之后再将内容返回给渲染进程。
- 一些缓存文件的读写也是由浏览器内核实现的，比如网络文件缓存的读取。

2. 网络访问

同样有了安全沙箱的保护，在渲染进程内部也是不能直接访问网络的，如果要访问网络，则需要通过浏览器内核。不过浏览器内核在处理URL请求之前，会检查渲染进程是否有权限请求该URL，比如检查XMLHttpRequest或者Fetch是否是跨站点请求，或者检测HTTPS的站点中是否包含了HTTP的请求。

3. 用户交互

渲染进程实现了安全沙箱，还影响到了一个非常重要的用户交互功能。

通常情况下，如果你要实现一个UI程序，操作系统会提供一个界面给你，该界面允许应用程序与用户交互，允许应用程序在该界面上进行绘制，比如Windows提供的是HWND，Linux提供的X Window，我们就把HWND和X Window统称为窗口句柄。应用程序可以在窗口句柄上进行绘制和接收键盘鼠标消息。

不过在现代浏览器中，由于每个渲染进程都有安全沙箱的保护，所以在渲染进程内部是无法直接操作窗口句柄的，这也是为了限制渲染进程监控到用户的输入事件。

由于渲染进程不能直接访问**窗口句柄**，所以渲染进程需要完成以下两点大的改变。

第一点，渲染进程需要渲染出位图。为了向用户显示渲染进程渲染出来的位图，渲染进程需要将生成好的位图发送到浏览器内核，然后浏览器内核将位图复制到屏幕上。

第二点，操作系统没有将用户输入事件直接传递给渲染进程，而是将这些事件传递给浏览器内核。然后浏览器内核再根据当前浏览器界面的状态来判断如何调度这些事件，如果当前焦点位于浏览器地址栏中，则输入事件会在浏览器内核内部处理；如果当前焦点在页面的区域内，则浏览器内核会将输入事件转发给渲染进程。

之所以这样设计，就是为了限制渲染进程有监控到用户输入事件的能力，所以所有的键盘鼠标事件都是由浏览器内核来接收的，然后浏览器内核再通过IPC将这些事件发送给渲染进程。

上面我们分析了由于渲染进程引入了安全沙箱，所以浏览器的持久存储、网络访问和用户交互等功能都不能在渲染进程内直接使用了，因此我们需要把这些功能迁移到浏览器内核中去实现，这让原本比较简单的流程变得复杂了。

理解这些限制，我们就能解释开始提出的两个问题了。

站点隔离（Site Isolation）

所谓站点隔离是指Chrome将同一站点（包含了相同根域名和相同协议的地址）中相互关联的页面放到同一个渲染进程中执行。

最开始Chrome划分渲染进程是以标签页为单位，也就是说整个标签页会被划分给某个渲染进程。但是，按照标签页划分渲染进程存在一些问题，原因就是一个标签页中可能包含了多个iframe，而这些iframe又有可能来自于不同的站点，这就导致了多个不同站点中的内容通过iframe同时运行在同一个渲染进程中。

目前所有操作系统都面临着两个A级漏洞——幽灵（Spectre）和熔毁（Meltdown），这两个漏洞是由处理器架构导致的，很难修补，黑客通过这两个漏洞可以直接入侵到进程的内部，如果入侵的进程没有安全沙箱的保护，那么黑客还可以发起对操作系统的攻击。

所以如果一个银行站点包含了一个恶意iframe，然后这个恶意的iframe利用这两个A级漏洞去入侵渲染进程，那么恶意程序就能读取银行站点渲染进程内的所有内容了，这对于用户来说就存在很大的风险了。

因此Chrome几年前就开始重构代码，将标签级的渲染进程重构为iframe级的渲染进程，然后严格按照同一站点的策略来分配渲染进程，这就是Chrome中的站点隔离。

实现了站点隔离，就可以将恶意的iframe隔离在恶意进程内部，使得它无法继续访问其他iframe进程的内容，因此也就无法攻击其他站点了。

值得注意是，2019年10月20日Chrome团队宣布安卓版的Chrome已经全面支持站点隔离，你可以参考文中链接。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容。

首先我们分析了单进程浏览器在系统安全方面的不足，如果浏览器存在漏洞，那么黑客就有机会通过页面对系统发起攻击。

因此在设计现代浏览器的体系架构时，就考虑到这个问题了。于是，在多进程的基础之上引入了安全沙箱，有了安全沙箱，就可以将操作系统和渲染进程进行隔离，这样即便渲染进程由于漏洞被攻击，也不会影响到操作系统的。

由于渲染进程采用了安全沙箱，所以在渲染进程内部不能与操作系统直接交互，于是就在浏览器内核中实现了持久存储、网络访问和用户交互等一系列与操作系统交互的功能，然后通过IPC和渲染进程进行交互。

最后我们还分析了Chrome中最新的站点隔离功能。由于最初都是按照标签页来划分渲染进程的，所以如果一个标签页里面有多个不同源的iframe，那么这些iframe也会被分配到同一个渲染进程中，这样就很容易让黑客通过iframe来攻击当前渲染进程。而站点隔离会将不同源的iframe分配到不同的渲染进程中，这样即使黑客攻击恶意iframe的渲染进程，也不会影响到其他渲染进程的。

今天介绍的内容和概念都比较多，看上去离前端比较远，不过这些内容会影响你对浏览器整体架构的理解，而深入理解了浏览器架构能帮助你更加深刻地理解前端内容。为了方便你的理解，我把一些参考资料放到了文章的最后，有需要的话你可以拿来参考。

思考时间

今天留给你的思考题：你认为安全沙箱能防止XSS或者CSRF一类的攻击的吗？为什么？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

参考资料

1. 安全沙箱的设计参考了最小权限原则
2. The Security Architecture of the Chromium Browser
3. The Security Architecture of the Chromium Browser-ppt
4. chromium site-isolation
5. Site Isolation
6. Site Isolation: Process Separation for Web Sites within the Browser

前面三篇文章我们主要围绕同源策略介绍了Web页面安全的相关内容，那今天我们把视野向外延伸，来聊聊页面安全和操作系统安全之间的关系。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中，我们分析了浏览器架构的发展史，在最开始的阶段，浏览器是单进程的，这意味着渲染过程、JavaScript执行过程、网络加载过程、UI绘制过程和页面显示过程等都是在同一个进程中执行的，这种结构虽然简单，但是也带来了很多问题。

从稳定性视角来看，单进程架构的浏览器是不稳定的，因为只要浏览器进程中的任意一个功能出现异常都有可能影响到整个浏览器，如页面卡死、浏览器崩溃等。不过浏览器的稳定性并不是本文讨论的重点，我们今天主要聊的是**浏览器架构是如何影响到操作系统安全的**。

浏览器本身的漏洞是单进程浏览器的一个主要问题，如果浏览器被曝出存在漏洞，那么在这些漏洞没有被及时修复的情况下，黑客就有可能通过恶意的页面向浏览器中注入恶意程序，其中最常见的攻击方式是利用**缓冲区溢出**，不过需要**注意这种类型的攻击和XSS注入的脚本是不一样的**。

- XSS攻击只是将恶意的JavaScript脚本注入到页面中，虽然能窃取一些Cookie相关的数据，但是XSS无法对操作系统进行攻击。

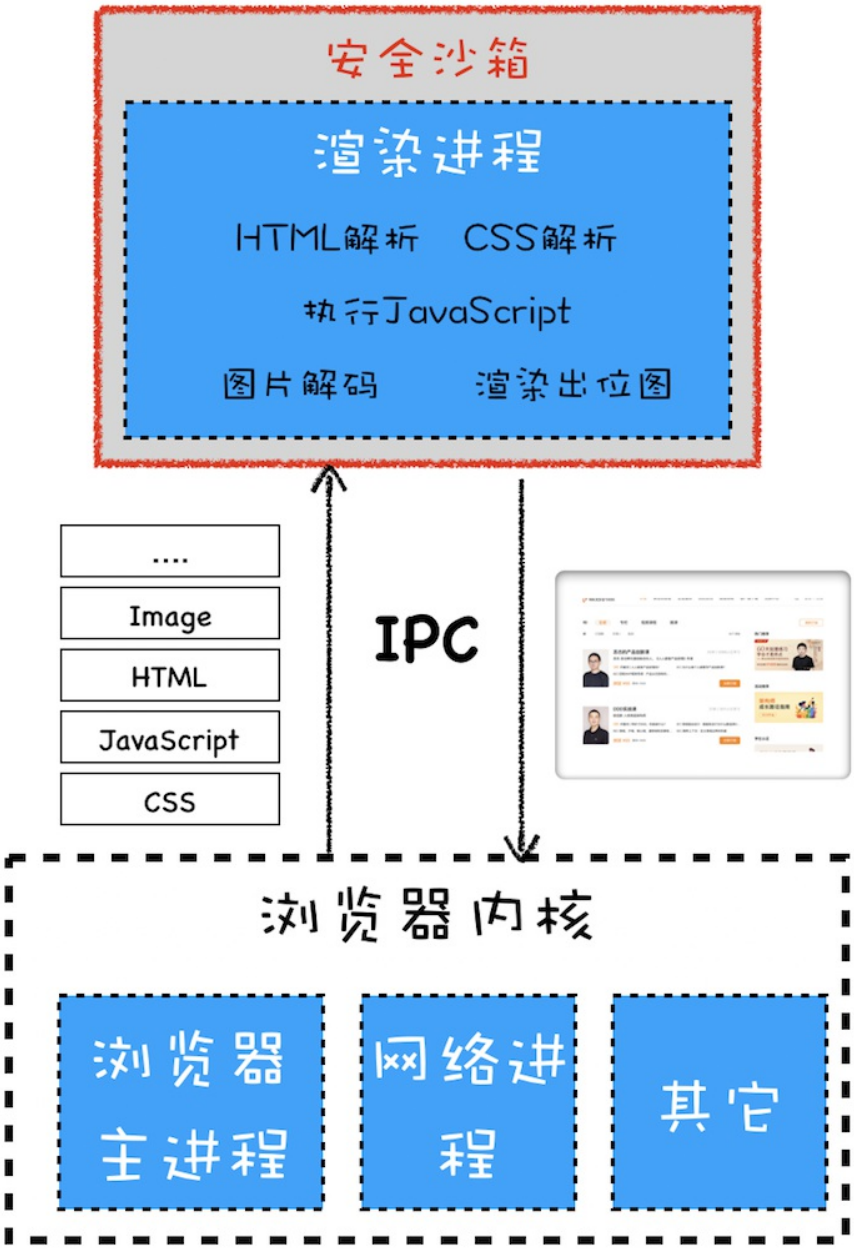
- 而通过浏览器漏洞进行的攻击是可以入侵到浏览器进程内部的，可以读取和修改浏览器进程内部的任意内容，还可以穿透浏览器，在用户的操作系统上悄悄地安装恶意软件、监听用户键盘输入信息以及读取用户硬盘上的文件内容。

和XSS攻击页面相比，这类攻击无疑是枚“核弹”，它会将整个操作系统的内容都暴露给黑客，这样我们操作系统上所有的资料都是不安全的了。

安全视角下的多进程架构

现代浏览器的设计目标是**安全**、**快速**和**稳定**，而这种核弹级杀伤力的安全问题就是一个很大的潜在威胁，因此在设计现代浏览器的体系架构时，需要解决这个问题。

我们知道现代浏览器采用了多进程架构，将渲染进程和浏览器主进程做了分离，完整的进程架构我们已经在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中介绍过了，这里我就不重复介绍了。下面我们重点从操作系统安全的视角来看看浏览器的多进程架构，如下图：



浏览器内核和渲染进程

观察上图，我们知道浏览器被划分为**浏览器内核**和**渲染内核**两个核心模块，其中浏览器内核是由网络进程、浏览器主进程和GPU进程组成的，渲染内核就是渲染进程。那如果我们在浏览器中打开一个页面，这两个模块是怎么配合的呢？

所有的网络资源都是通过浏览器内核来下载的，下载后的资源会通过IPC将其提交给渲染进程（浏览器内核和渲染进程之间都是通过IPC来通信的）。然后渲染进程会对这些资源进行解析、绘制等操作，最终生成一幅图片。但是渲染进程并不负责将图片显示到界面上，而是将最终生成的图片提交给浏览器内核模块，由浏览器内核模块负责显示这张图片。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》中我们分析过，设计现代浏览器体系架构时，将浏览器划分为不同的进程是为了增加其稳定性。虽然设计成了多进程架构，不过这些模块之间的沟通方式却有些复杂，也许你还有以下问题：

- 为什么一定要通过浏览器内核去请求资源，再将数据转发给渲染进程，而不直接从进程内部去请求网络资源？
- 为什么渲染进程只负责生成页面图片，生成图片还要经过IPC通知浏览器内核模块，然后让浏览器内核去负责展示图片？

通过以上方式不是增加了工程的复杂度吗？

要解释现代浏览器为什么要把这个流程弄得这么复杂，我们就得从系统安全的角度来分析。

安全沙箱

不过在解释这些问题之前，我们得先看看什么是安全沙箱。

上面我们分析过了，由于渲染进程需要执行DOM解析、CSS解析、网络图片解码等操作，如果渲染进程中存在系统级别的漏洞，那么以上操作就有可能让恶意的站点获取到渲染进程的控制权限，进而又获取操作系统的控制权限，这对于用户来说是非常危险的。

因为网络资源的内容存在着各种可能性，所以浏览器会默认所有的网络资源都是不可信的，都是不安全的。但谁也不能保证浏览器不存在漏洞，只要出现漏洞，黑客就可以通过网络内容对用户发起攻击。

我们知道，如果你下载了一个恶意程序，但是没有执行它，那么恶意程序是不会生效的。同理，浏览器之于网络内容也是如此，浏览器可以安全地下载各种网络资源，但是如果要执行这些网络资源，比如解析HTML、解析CSS、执行JavaScript、图片编解码等操作，就需要非常谨慎了，因为一不小心，黑客就会利用这些操作对含有漏洞的浏览器发起攻击。

基于以上原因，我们需要在渲染进程和操作系统之间建一道墙，即便渲染进程由于存在漏洞被黑客攻击，但由于这道墙，黑客就获取不到渲染进程之外的任何操作权限。**将渲染进程和操作系统隔离的这道墙就是我们要聊的安全沙箱**。

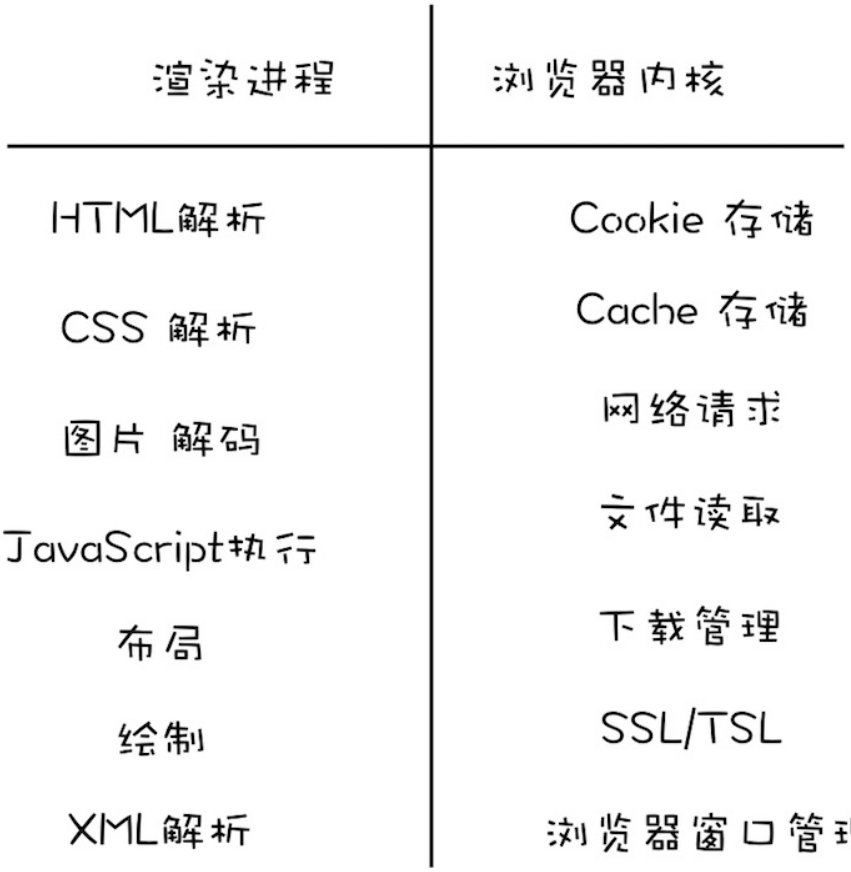
浏览器中的安全沙箱是利用操作系统提供的安全技术，让渲染进程在执行过程中无法访问或者修改操作系统中的数据，在渲染进程需要访问系统资源的时候，需要通过浏览器内核来实现，然后将访问的结果通过IPC转发给渲染进程。

安全沙箱最小的保护单位是进程。因为单进程浏览器需要频繁访问或者修改操作系统的数据，所以单进程浏览器是无法被安全沙箱保护的，而现代浏览器采用的多进程架构使得安全沙箱可以发挥作用。

安全沙箱如何影响各个模块功能

我们知道安全沙箱最小的保护单位是进程，并且能限制进程对操作系统资源的访问和修改，这就意味着如果要让安全沙箱应用在某个进程上，那么这个进程必须没有读写操作系统的功能，比如读写本地文件、发起网络请求、调用GPU接口等。

了解了被安全沙箱保护的进程会有一系列的受限操作之后，接下来我们就可以分析渲染进程和浏览器内核各自都有哪些职责，如下图：



浏览器内核和渲染进程各自职责

通过该图，我们可以看到由于渲染进程需要安全沙箱的保护，因此需要把在渲染进程内部涉及到和系统交互的功能都转移到浏览器内核中去实现。

那安全沙箱是如何影响到各个模块功能的呢？

1. 持久存储

我们先来看看安全沙箱是如何影响到浏览器持久存储的。由于安全沙箱需要负责确保渲染进程无法直接访问用户的文件系统，但是在渲染进程内部有访问Cookie的需求、有上传文件的需求，为了解决这些文件的访问需求，所以现代浏览器将读写文件的操作全部放在了浏览器内核中实现，然后通过IPC将操作结果转发给渲染进程。

具体地讲，如下文件内容的读写都是在浏览器内核中完成的：

- 存储Cookie数据的读写。通常浏览器内核会维护一个存放所有Cookie的Cookie数据库，然后当渲染进程通过JavaScript来读取Cookie时，渲染进程会通过IPC将读取Cookie的信息发送给浏览器内核，浏览器内核读取Cookie之后再将内容返回给渲染进程。
- 一些缓存文件的读写也是由浏览器内核实现的，比如网络文件缓存的读取。

2. 网络访问

同样有了安全沙箱的保护，在渲染进程内部也是不能直接访问网络的，如果要访问网络，则需要通过浏览器内核。不过浏览器内核在处理URL请求之前，会检查渲染进程是否有权限请求该URL，比如检查XMLHttpRequest或者Fetch是否是跨站点请求，或者检测HTTPS的站点中是否包含了HTTP的请求。

3. 用户交互

渲染进程实现了安全沙箱，还影响到了一个非常重要的用户交互功能。

通常情况下，如果你要实现一个UI程序，操作系统会提供一个界面给你，该界面允许应用程序与用户交互，允许应用程序在该界面上进行绘制，比如Windows提供的是HWND，Linux提供的X Window，我们就把HWND和X Window统称为窗口句柄。应用程序可以在窗口句柄上进行绘制和接收键盘鼠标消息。

不过在现代浏览器中，由于每个渲染进程都有安全沙箱的保护，所以在渲染进程内部是无法直接操作窗口句柄的，这也是为了限制渲染进程监控到用户的输入事件。

由于渲染进程不能直接访问**窗口句柄**，所以渲染进程需要完成以下两点大的改变。

第一点，渲染进程需要渲染出位图。为了向用户显示渲染进程渲染出来的位图，渲染进程需要将生成好的位图发送到浏览器内核，然后浏览器内核将位图复制到屏幕上。

第二点，操作系统没有将用户输入事件直接传递给渲染进程，而是将这些事件传递给浏览器内核。然后浏览器内核再根据当前浏览器界面的状态来判断如何调度这些事件，如果当前焦点位于浏览器地址栏中，则输入事件会在浏览器内核内部处理；如果当前焦点在页面的区域内，则浏览器内核会将输入事件转发给渲染进程。

之所以这样设计，就是为了限制渲染进程有监控到用户输入事件的能力，所以所有的键盘鼠标事件都是由浏览器内核来接收的，然后浏览器内核再通过IPC将这些事件发送给渲染进程。

上面我们分析了由于渲染进程引入了安全沙箱，所以浏览器的持久存储、网络访问和用户交互等功能都不能在渲染进程内直接使用了，因此我们需要把这些功能迁移到浏览器内核中去实现，这让原本比较简单的流程变得复杂了。

理解这些限制，我们就能解释开始提出的两个问题了。

站点隔离（Site Isolation）

所谓站点隔离是指Chrome将同一站点（包含了相同根域名和相同协议的地址）中相互关联的页面放到同一个渲染进程中执行。

最开始Chrome划分渲染进程是以标签页为单位，也就是说整个标签页会被划分给某个渲染进程。但是，按照标签页划分渲染进程存在一些问题，原因就是一个标签页中可能包含了多个iframe，而这些iframe又有可能来自于不同的站点，这就导致了多个不同站点中的内容通过iframe同时运行在同一个渲染进程中。

目前所有操作系统都面临着两个A级漏洞——幽灵（Spectre）和熔毁（Meltdown），这两个漏洞是由处理器架构导致的，很难修补，黑客通过这两个漏洞可以直接入侵到进程的内部，如果入侵的进程没有安全沙箱的保护，那么黑客还可以发起对操作系统的攻击。

所以如果一个银行站点包含了一个恶意iframe，然后这个恶意的iframe利用这两个A级漏洞去入侵渲染进程，那么恶意程序就能读取银行站点渲染进程内的所有内容了，这对于用户来说就存在很大的风险了。

因此Chrome几年前就开始重构代码，将标签级的渲染进程重构为iframe级的渲染进程，然后严格按照同一站点的策略来分配渲染进程，这就是Chrome中的站点隔离。

实现了站点隔离，就可以将恶意的iframe隔离在恶意进程内部，使得它无法继续访问其他iframe进程的内容，因此也就无法攻击其他站点了。

值得注意是，2019年10月20日Chrome团队宣布安卓版的Chrome已经全面支持站点隔离，你可以参考文中链接。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容。

首先我们分析了单进程浏览器在系统安全方面的不足，如果浏览器存在漏洞，那么黑客就有机会通过页面对系统发起攻击。

因此在设计现代浏览器的体系架构时，就考虑到这个问题了。于是，在多进程的基础之上引入了安全沙箱，有了安全沙箱，就可以将操作系统和渲染进程进行隔离，这样即便渲染进程由于漏洞被攻击，也不会影响到操作系统的。

由于渲染进程采用了安全沙箱，所以在渲染进程内部不能与操作系统直接交互，于是就在浏览器内核中实现了持久存储、网络访问和用户交互等一系列与操作系统交互的功能，然后通过IPC和渲染进程进行交互。

最后我们还分析了Chrome中最新的站点隔离功能。由于最初都是按照标签页来划分渲染进程的，所以如果一个标签页里面有多个不同源的iframe，那么这些iframe也会被分配到同一个渲染进程中，这样就很容易让黑客通过iframe来攻击当前渲染进程。而站点隔离会将不同源的iframe分配到不同的渲染进程中，这样即使黑客攻击恶意iframe的渲染进程，也不会影响到其他渲染进程的。

今天介绍的内容和概念都比较多，看上去离前端比较远，不过这些内容会影响你对浏览器整体架构的理解，而深入理解了浏览器架构能帮助你更加深刻地理解前端内容。为了方便你的理解，我把一些参考资料放到了文章的最后，有需要的话你可以拿来参考。

思考时间

今天留给你的思考题：你认为安全沙箱能防止XSS或者CSRF一类的攻击的吗？为什么？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

参考资料

1. 安全沙箱的设计参考了最小权限原则
2. The Security Architecture of the Chromium Browser
3. The Security Architecture of the Chromium Browser-ppt
4. chromium site-isolation
5. Site Isolation
6. Site Isolation: Process Separation for Web Sites within the Browser

前面三篇文章我们主要围绕同源策略介绍了Web页面安全的相关内容，那今天我们把视野向外延伸，来聊聊页面安全和操作系统安全之间的关系。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中，我们分析了浏览器架构的发展史，在最开始的阶段，浏览器是单进程的，这意味着渲染过程、JavaScript执行过程、网络加载过程、UI绘制过程和页面显示过程等都是在同一个进程中执行的，这种结构虽然简单，但是也带来了很多问题。

从稳定性视角来看，单进程架构的浏览器是不稳定的，因为只要浏览器进程中的任意一个功能出现异常都有可能影响到整个浏览器，如页面卡死、浏览器崩溃等。不过浏览器的稳定性并不是本文讨论的重点，我们今天主要聊的是**浏览器架构是如何影响到操作系统安全的**。

浏览器本身的漏洞是单进程浏览器的一个主要问题，如果浏览器被曝出存在漏洞，那么在这些漏洞没有被及时修复的情况下，黑客就有可能通过恶意的页面向浏览器中注入恶意程序，其中最常见的攻击方式是利用**缓冲区溢出**，不过需要**注意这种类型的攻击和XSS注入的脚本是不一样的**。

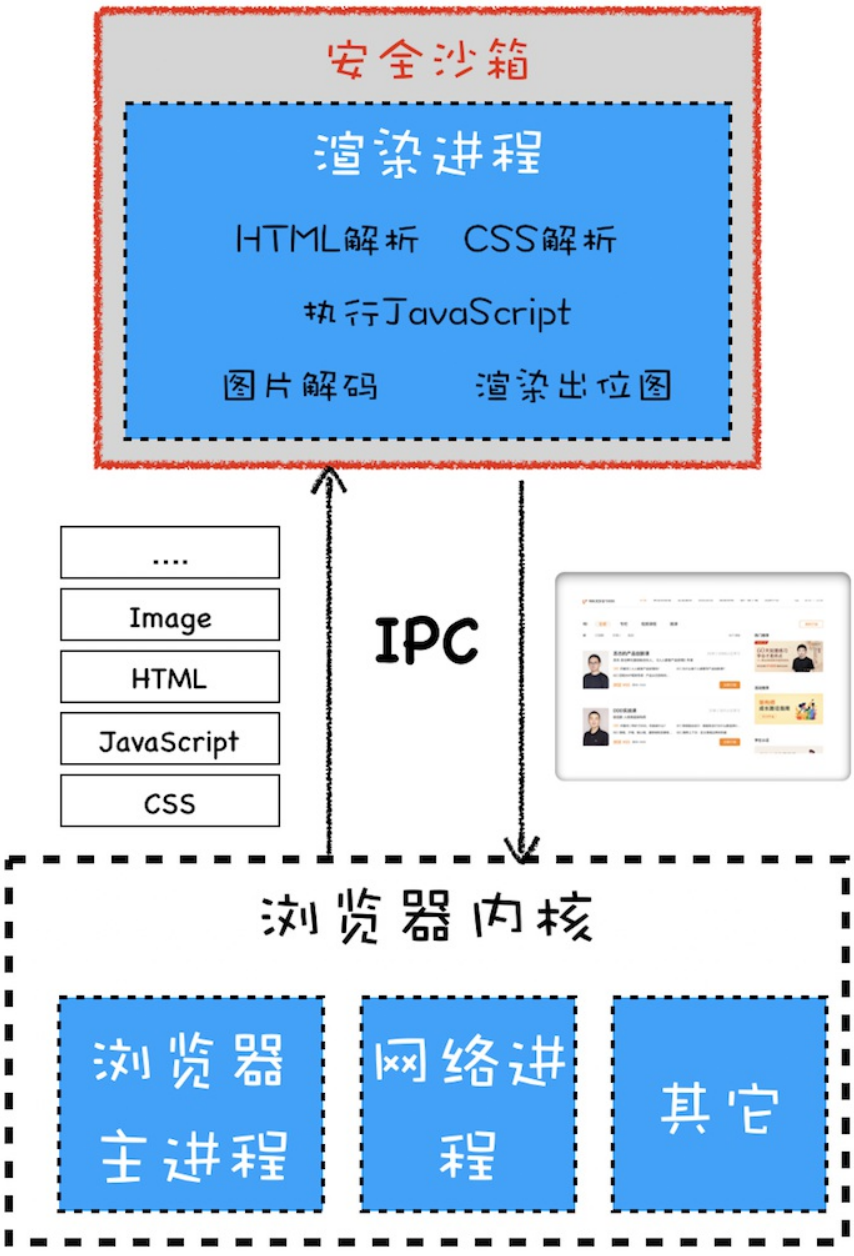
- XSS攻击只是将恶意的JavaScript脚本注入到页面中，虽然能窃取一些Cookie相关的数据，但是XSS无法对操作系统进行攻击。
- 而通过浏览器漏洞进行的攻击是可以入侵到浏览器进程内部的，可以读取和修改浏览器进程内部的任意内容，还可以穿透浏览器，在用户的操作系统上悄悄地安装恶意软件、监听用户键盘输入信息以及读取用户硬盘上的文件内容。

和XSS攻击页面相比，这类攻击无疑是枚“核弹”，它会将整个操作系统的内容都暴露给黑客，这样我们操作系统上所有的资料都是不安全的了。

安全视角下的多进程架构

现代浏览器的设计目标是**安全、快速**和**稳定**，而这种核弹级杀伤力的安全问题就是一个很大的潜在威胁，因此在设计现代浏览器的体系架构时，需要解决这个问题。

我们知道现代浏览器采用了多进程架构，将渲染进程和浏览器主进程做了分离，完整的进程架构我们已经在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中介绍过了，这里我就不重复介绍了。下面我们重点从操作系统安全的视角来看看浏览器的多进程架构，如下图：



浏览器内核和渲染进程

观察上图，我们知道浏览器被划分为**浏览器内核**和**渲染内核**两个核心模块，其中浏览器内核是由网络进程、浏览器主进程和GPU进程组成的，渲染内核就是渲染进程。那如果我们在浏览器中打开一个页面，这两个模块是怎么配合的呢？

所有的网络资源都是通过浏览器内核来下载的，下载后的资源会通过IPC将其提交给渲染进程（浏览器内核和渲染进程之间都是通过IPC来通信的）。然后渲染进程会对这些资源进行解析、绘制等操作，最终生成一幅图片。但是渲染进程并不负责将图片显示到界面上，而是将最终生成的图片提交给浏览器内核模块，由浏览器内核模块负责显示这张图片。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》中我们分析过，设计现代浏览器体系架构时，将浏览器划分为不同的进程是为了增加其稳定性。虽然设计成了多进程架构，不过这些模块之间的沟通方式却有些复杂，也许你还有以下问题：

- 为什么一定要通过浏览器内核去请求资源，再将数据转发给渲染进程，而不直接从进程内部去请求网络资源？
- 为什么渲染进程只负责生成页面图片，生成图片还要经过IPC通知浏览器内核模块，然后让浏览器内核去负责展示图片？

通过以上方式不是增加了工程的复杂度吗？

要解释现代浏览器为什么要把这个流程弄得这么复杂，我们就得从系统安全的角度来分析。

安全沙箱

不过在解释这些问题之前，我们得先看看什么是安全沙箱。

上面我们分析过了，由于渲染进程需要执行DOM解析、CSS解析、网络图片解码等操作，如果渲染进程中存在系统级别的漏洞，那么以上操作就有可能让恶意的站点获取到渲染进程的控制权限，进而又获取操作系统的控制权限，这对于用户来说是非常危险的。

因为网络资源的内容存在着各种可能性，所以浏览器会默认所有的网络资源都是不可信的，都是不安全的。但谁也不能保证浏览器不存在漏洞，只要出现漏洞，黑客就可以通过网络内容对用户发起攻击。

我们知道，如果你下载了一个恶意程序，但是没有执行它，那么恶意程序是不会生效的。同理，浏览器之于网络内容也是如此，浏览器可以安全地下载各种网络资源，但是如果要执行这些网络资源，比如解析HTML、解析CSS、执行JavaScript、图片编解码等操作，就需要非常谨慎了，因为一不小心，黑客就会利用这些操作对含有漏洞的浏览器发起攻击。

基于以上原因，我们需要在渲染进程和操作系统之间建一道墙，即便渲染进程由于存在漏洞被黑客攻击，但由于这道墙，黑客就获取不到渲染进程之外的任何操作权限。**将渲染进程和操作系统隔离的这道墙就是我们要聊的安全沙箱**。

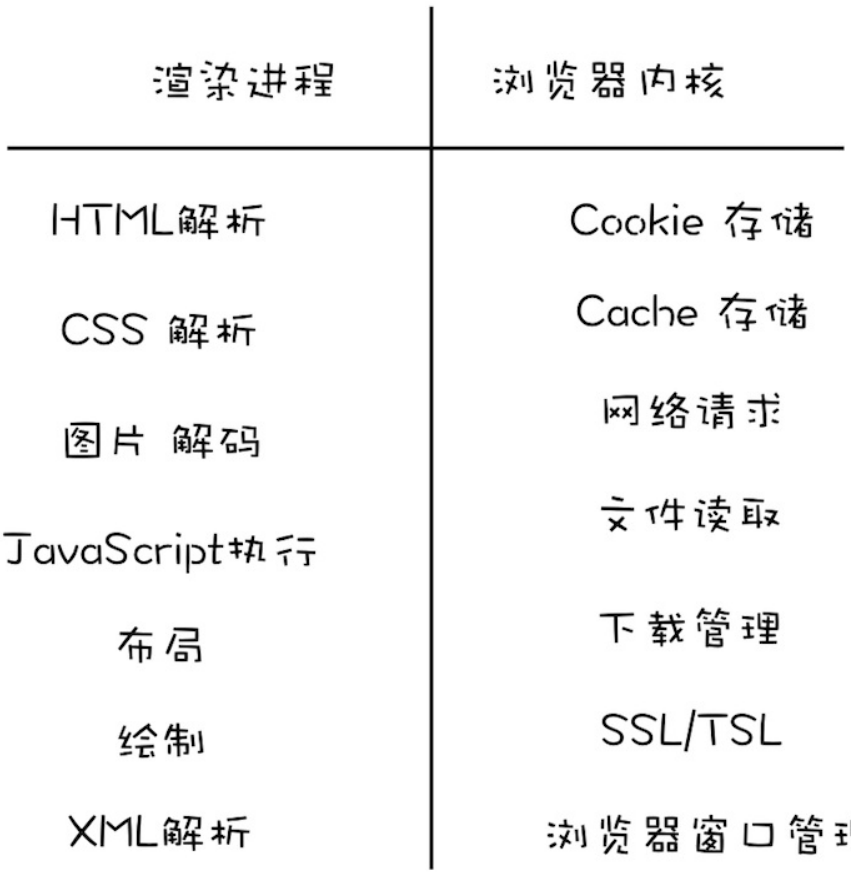
浏览器中的安全沙箱是利用操作系统提供的安全技术，让渲染进程在执行过程中无法访问或者修改操作系统中的数据，在渲染进程需要访问系统资源的时候，需要通过浏览器内核来实现，然后将访问的结果通过IPC转发给渲染进程。

安全沙箱最小的保护单位是进程。因为单进程浏览器需要频繁访问或者修改操作系统的数据，所以单进程浏览器是无法被安全沙箱保护的，而现代浏览器采用的多进程架构使得安全沙箱可以发挥作用。

安全沙箱如何影响各个模块功能

我们知道安全沙箱最小的保护单位是进程，并且能限制进程对操作系统资源的访问和修改，这就意味着如果要让安全沙箱应用在某个进程上，那么这个进程必须没有读写操作系统的功能，比如读写本地文件、发起网络请求、调用GPU接口等。

了解了被安全沙箱保护的进程会有一系列的受限操作之后，接下来我们就可以分析渲染进程和浏览器内核各自都有哪些职责，如下图：



浏览器内核和渲染进程各自职责

通过该图，我们可以看到由于渲染进程需要安全沙箱的保护，因此需要把在渲染进程内部涉及到和系统交互的功能都转移到浏览器内核中去实现。

那安全沙箱是如何影响到各个模块功能的呢？

1. 持久存储

我们先来看看安全沙箱是如何影响到浏览器持久存储的。由于安全沙箱需要负责确保渲染进程无法直接访问用户的文件系统，但是在渲染进程内部有访问Cookie的需求、有上传文件的需求，为了解决这些文件的访问需求，所以现代浏览器将读写文件的操作全部放在了浏览器内核中实现，然后通过IPC将操作结果转发给渲染进程。

具体地讲，如下文件内容的读写都是在浏览器内核中完成的：

- 存储Cookie数据的读写。通常浏览器内核会维护一个存放所有Cookie的Cookie数据库，然后当渲染进程通过JavaScript来读取Cookie时，渲染进程会通过IPC将读取Cookie的信息发送给浏览器内核，浏览器内核读取Cookie之后再将内容返回给渲染进程。
- 一些缓存文件的读写也是由浏览器内核实现的，比如网络文件缓存的读取。

2. 网络访问

同样有了安全沙箱的保护，在渲染进程内部也是不能直接访问网络的，如果要访问网络，则需要通过浏览器内核。不过浏览器内核在处理URL请求之前，会检查渲染进程是否有权限请求该URL，比如检查XMLHttpRequest或者Fetch是否是跨站点请求，或者检测HTTPS的站点中是否包含了HTTP的请求。

3. 用户交互

渲染进程实现了安全沙箱，还影响到了一个非常重要的用户交互功能。

通常情况下，如果你要实现一个UI程序，操作系统会提供一个界面给你，该界面允许应用程序与用户交互，允许应用程序在该界面上进行绘制，比如Windows提供的是HWND，Linux提供的X Window，我们就把HWND和X Window统称为窗口句柄。应用程序可以在窗口句柄上进行绘制和接收键盘鼠标消息。

不过在现代浏览器中，由于每个渲染进程都有安全沙箱的保护，所以在渲染进程内部是无法直接操作窗口句柄的，这也是为了限制渲染进程监控到用户的输入事件。

由于渲染进程不能直接访问**窗口句柄**，所以渲染进程需要完成以下两点大的改变。

第一点，渲染进程需要渲染出位图。为了向用户显示渲染进程渲染出来的位图，渲染进程需要将生成好的位图发送到浏览器内核，然后浏览器内核将位图复制到屏幕上。

第二点，操作系统没有将用户输入事件直接传递给渲染进程，而是将这些事件传递给浏览器内核。然后浏览器内核再根据当前浏览器界面的状态来判断如何调度这些事件，如果当前焦点位于浏览器地址栏中，则输入事件会在浏览器内核内部处理；如果当前焦点在页面的区域内，则浏览器内核会将输入事件转发给渲染进程。

之所以这样设计，就是为了限制渲染进程有监控到用户输入事件的能力，所以所有的键盘鼠标事件都是由浏览器内核来接收的，然后浏览器内核再通过IPC将这些事件发送给渲染进程。

上面我们分析了由于渲染进程引入了安全沙箱，所以浏览器的持久存储、网络访问和用户交互等功能都不能在渲染进程内直接使用了，因此我们需要把这些功能迁移到浏览器内核中去实现，这让原本比较简单的流程变得复杂了。

理解这些限制，我们就能解释开始提出的两个问题了。

站点隔离（Site Isolation）

所谓站点隔离是指Chrome将同一站点（包含了相同根域名和相同协议的地址）中相互关联的页面放到同一个渲染进程中执行。

最开始Chrome划分渲染进程是以标签页为单位，也就是说整个标签页会被划分给某个渲染进程。但是，按照标签页划分渲染进程存在一些问题，原因就是一个标签页中可能包含了多个iframe，而这些iframe又有可能来自于不同的站点，这就导致了多个不同站点中的内容通过iframe同时运行在同一个渲染进程中。

目前所有操作系统都面临着两个A级漏洞——幽灵（Spectre）和熔毁（Meltdown），这两个漏洞是由处理器架构导致的，很难修补，黑客通过这两个漏洞可以直接入侵到进程的内部，如果入侵的进程没有安全沙箱的保护，那么黑客还可以发起对操作系统的攻击。

所以如果一个银行站点包含了一个恶意iframe，然后这个恶意的iframe利用这两个A级漏洞去入侵渲染进程，那么恶意程序就能读取银行站点渲染进程内的所有内容了，这对于用户来说就存在很大的风险了。

因此Chrome几年前就开始重构代码，将标签级的渲染进程重构为iframe级的渲染进程，然后严格按照同一站点的策略来分配渲染进程，这就是Chrome中的站点隔离。

实现了站点隔离，就可以将恶意的iframe隔离在恶意进程内部，使得它无法继续访问其他iframe进程的内容，因此也就无法攻击其他站点了。

值得注意是，2019年10月20日Chrome团队宣布安卓版的Chrome已经全面支持站点隔离，你可以参考文中链接。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容。

首先我们分析了单进程浏览器在系统安全方面的不足，如果浏览器存在漏洞，那么黑客就有机会通过页面对系统发起攻击。

因此在设计现代浏览器的体系架构时，就考虑到这个问题了。于是，在多进程的基础之上引入了安全沙箱，有了安全沙箱，就可以将操作系统和渲染进程进行隔离，这样即便渲染进程由于漏洞被攻击，也不会影响到操作系统的。

由于渲染进程采用了安全沙箱，所以在渲染进程内部不能与操作系统直接交互，于是就在浏览器内核中实现了持久存储、网络访问和用户交互等一系列与操作系统交互的功能，然后通过IPC和渲染进程进行交互。

最后我们还分析了Chrome中最新的站点隔离功能。由于最初都是按照标签页来划分渲染进程的，所以如果一个标签页里面有多个不同源的iframe，那么这些iframe也会被分配到同一个渲染进程中，这样就很容易让黑客通过iframe来攻击当前渲染进程。而站点隔离会将不同源的iframe分配到不同的渲染进程中，这样即使黑客攻击恶意iframe的渲染进程，也不会影响到其他渲染进程的。

今天介绍的内容和概念都比较多，看上去离前端比较远，不过这些内容会影响你对浏览器整体架构的理解，而深入理解了浏览器架构能帮助你更加深刻地理解前端内容。为了方便你的理解，我把一些参考资料放到了文章的最后，有需要的话你可以拿来参考。

思考时间

今天留给你的思考题：你认为安全沙箱能防止XSS或者CSRF一类的攻击的吗？为什么？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

参考资料

1. 安全沙箱的设计参考了最小权限原则
2. The Security Architecture of the Chromium Browser
3. The Security Architecture of the Chromium Browser-ppt
4. chromium site-isolation
5. Site Isolation
6. Site Isolation: Process Separation for Web Sites within the Browser

前面三篇文章我们主要围绕同源策略介绍了Web页面安全的相关内容，那今天我们把视野向外延伸，来聊聊页面安全和操作系统安全之间的关系。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中，我们分析了浏览器架构的发展史，在最开始的阶段，浏览器是单进程的，这意味着渲染过程、JavaScript执行过程、网络加载过程、UI绘制过程和页面显示过程等都是在同一个进程中执行的，这种结构虽然简单，但是也带来了很多问题。

从稳定性视角来看，单进程架构的浏览器是不稳定的，因为只要浏览器进程中的任意一个功能出现异常都有可能影响到整个浏览器，如页面卡死、浏览器崩溃等。不过浏览器的稳定性并不是本文讨论的重点，我们今天主要聊的是**浏览器架构是如何影响到操作系统安全的**。

浏览器本身的漏洞是单进程浏览器的一个主要问题，如果浏览器被曝出存在漏洞，那么在这些漏洞没有被及时修复的情况下，黑客就有可能通过恶意的页面向浏览器中注入恶意程序，其中最常见的攻击方式是利用**缓冲区溢出**，不过需要**注意这种类型的攻击和XSS注入的脚本是不一样的**。

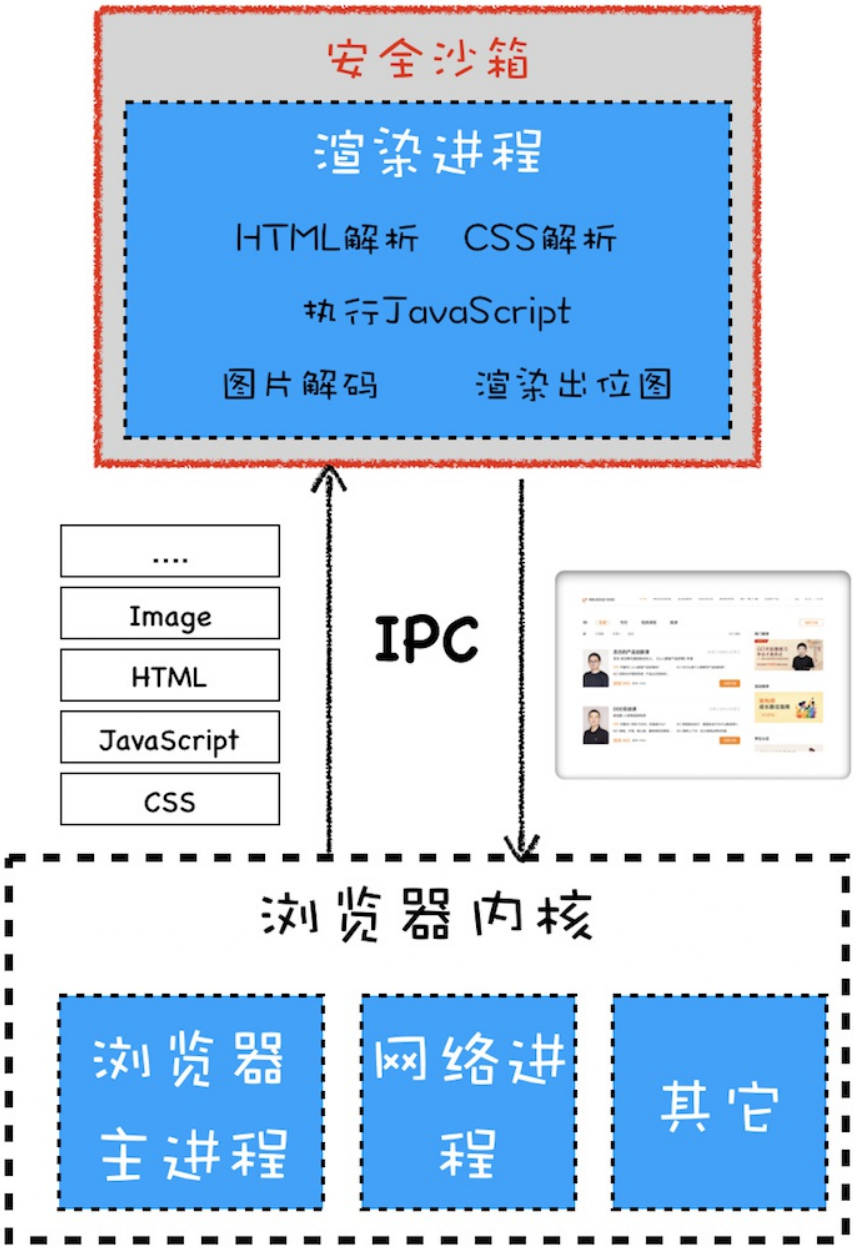
- XSS攻击只是将恶意的JavaScript脚本注入到页面中，虽然能窃取一些Cookie相关的数据，但是XSS无法对操作系统进行攻击。
- 而通过浏览器漏洞进行的攻击是可以入侵到浏览器进程内部的，可以读取和修改浏览器进程内部的任意内容，还可以穿透浏览器，在用户的操作系统上悄悄地安装恶意软件、监听用户键盘输入信息以及读取用户硬盘上的文件内容。

和XSS攻击页面相比，这类攻击无疑是枚“核弹”，它会将整个操作系统的内容都暴露给黑客，这样我们操作系统上所有的资料都是不安全的了。

安全视角下的多进程架构

现代浏览器的设计目标是**安全、快速**和**稳定**，而这种核弹级杀伤力的安全问题就是一个很大的潜在威胁，因此在设计现代浏览器的体系架构时，需要解决这个问题。

我们知道现代浏览器采用了多进程架构，将渲染进程和浏览器主进程做了分离，完整的进程架构我们已经在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中介绍过了，这里我就不重复介绍了。下面我们重点从操作系统安全的视角来看看浏览器的多进程架构，如下图：



浏览器内核和渲染进程

观察上图，我们知道浏览器被划分为**浏览器内核**和**渲染内核**两个核心模块，其中浏览器内核是由网络进程、浏览器主进程和GPU进程组成的，渲染内核就是渲染进程。那如果我们在浏览器中打开一个页面，这两个模块是怎么配合的呢？

所有的网络资源都是通过浏览器内核来下载的，下载后的资源会通过IPC将其提交给渲染进程（浏览器内核和渲染进程之间都是通过IPC来通信的）。然后渲染进程会对这些资源进行解析、绘制等操作，最终生成一幅图片。但是渲染进程并不负责将图片显示到界面上，而是将最终生成的图片提交给浏览器内核模块，由浏览器内核模块负责显示这张图片。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》中我们分析过，设计现代浏览器体系架构时，将浏览器划分为不同的进程是为了增加其稳定性。虽然设计成了多进程架构，不过这些模块之间的沟通方式却有些复杂，也许你还有以下问题：

- 为什么一定要通过浏览器内核去请求资源，再将数据转发给渲染进程，而不直接从进程内部去请求网络资源？
- 为什么渲染进程只负责生成页面图片，生成图片还要经过IPC通知浏览器内核模块，然后让浏览器内核去负责展示图片？

通过以上方式不是增加了工程的复杂度吗？

要解释现代浏览器为什么要把这个流程弄得这么复杂，我们就得从系统安全的角度来分析。

安全沙箱

不过在解释这些问题之前，我们得先看看什么是安全沙箱。

上面我们分析过了，由于渲染进程需要执行DOM解析、CSS解析、网络图片解码等操作，如果渲染进程中存在系统级别的漏洞，那么以上操作就有可能让恶意的站点获取到渲染进程的控制权限，进而又获取操作系统的控制权限，这对于用户来说是非常危险的。

因为网络资源的内容存在着各种可能性，所以浏览器会默认所有的网络资源都是不可信的，都是不安全的。但谁也不能保证浏览器不存在漏洞，只要出现漏洞，黑客就可以通过网络内容对用户发起攻击。

我们知道，如果你下载了一个恶意程序，但是没有执行它，那么恶意程序是不会生效的。同理，浏览器之于网络内容也是如此，浏览器可以安全地下载各种网络资源，但是如果要执行这些网络资源，比如解析HTML、解析CSS、执行JavaScript、图片编解码等操作，就需要非常谨慎了，因为一不小心，黑客就会利用这些操作对含有漏洞的浏览器发起攻击。

基于以上原因，我们需要在渲染进程和操作系统之间建一道墙，即便渲染进程由于存在漏洞被黑客攻击，但由于这道墙，黑客就获取不到渲染进程之外的任何操作权限。**将渲染进程和操作系统隔离的这道墙就是我们要聊的安全沙箱**。

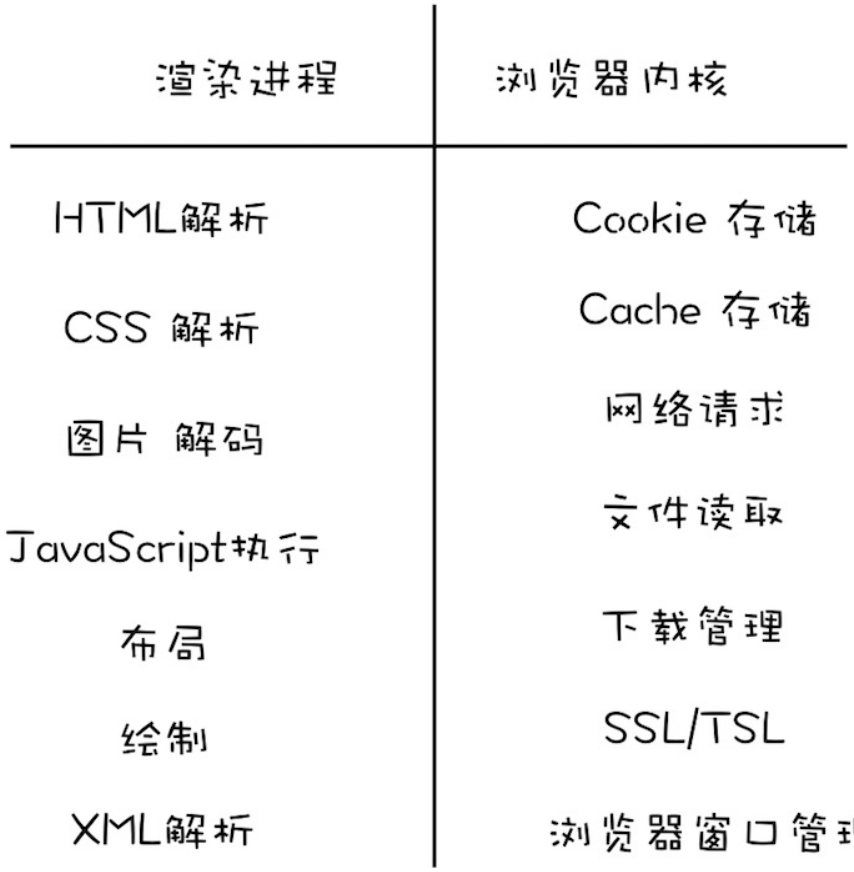
浏览器中的安全沙箱是利用操作系统提供的安全技术，让渲染进程在执行过程中无法访问或者修改操作系统中的数据，在渲染进程需要访问系统资源的时候，需要通过浏览器内核来实现，然后将访问的结果通过IPC转发给渲染进程。

安全沙箱最小的保护单位是进程。因为单进程浏览器需要频繁访问或者修改操作系统的数据，所以单进程浏览器是无法被安全沙箱保护的，而现代浏览器采用的多进程架构使得安全沙箱可以发挥作用。

安全沙箱如何影响各个模块功能

我们知道安全沙箱最小的保护单位是进程，并且能限制进程对操作系统资源的访问和修改，这就意味着如果要让安全沙箱应用在某个进程上，那么这个进程必须没有读写操作系统的功能，比如读写本地文件、发起网络请求、调用GPU接口等。

了解了被安全沙箱保护的进程会有一系列的受限操作之后，接下来我们就可以分析渲染进程和浏览器内核各自都有哪些职责，如下图：



浏览器内核和渲染进程各自职责

通过该图，我们可以看到由于渲染进程需要安全沙箱的保护，因此需要把在渲染进程内部涉及到和系统交互的功能都转移到浏览器内核中去实现。

那安全沙箱是如何影响到各个模块功能的呢？

1. 持久存储

我们先来看看安全沙箱是如何影响到浏览器持久存储的。由于安全沙箱需要负责确保渲染进程无法直接访问用户的文件系统，但是在渲染进程内部有访问Cookie的需求、有上传文件的需求，为了解决这些文件的访问需求，所以现代浏览器将读写文件的操作全部放在了浏览器内核中实现，然后通过IPC将操作结果转发给渲染进程。

具体地讲，如下文件内容的读写都是在浏览器内核中完成的：

- 存储Cookie数据的读写。通常浏览器内核会维护一个存放所有Cookie的Cookie数据库，然后当渲染进程通过JavaScript来读取Cookie时，渲染进程会通过IPC将读取Cookie的信息发送给浏览器内核，浏览器内核读取Cookie之后再将内容返回给渲染进程。
- 一些缓存文件的读写也是由浏览器内核实现的，比如网络文件缓存的读取。

2. 网络访问

同样有了安全沙箱的保护，在渲染进程内部也是不能直接访问网络的，如果要访问网络，则需要通过浏览器内核。不过浏览器内核在处理URL请求之前，会检查渲染进程是否有权限请求该URL，比如检查XMLHttpRequest或者Fetch是否是跨站点请求，或者检测HTTPS的站点中是否包含了HTTP的请求。

3. 用户交互

渲染进程实现了安全沙箱，还影响到了一个非常重要的用户交互功能。

通常情况下，如果你要实现一个UI程序，操作系统会提供一个界面给你，该界面允许应用程序与用户交互，允许应用程序在该界面上进行绘制，比如Windows提供的是HWND，Linux提供的X Window，我们就把HWND和X Window统称为窗口句柄。应用程序可以在窗口句柄上进行绘制和接收键盘鼠标消息。

不过在现代浏览器中，由于每个渲染进程都有安全沙箱的保护，所以在渲染进程内部是无法直接操作窗口句柄的，这也是为了限制渲染进程监控到用户的输入事件。

由于渲染进程不能直接访问**窗口句柄**，所以渲染进程需要完成以下两点大的改变。

第一点，渲染进程需要渲染出位图。为了向用户显示渲染进程渲染出来的位图，渲染进程需要将生成好的位图发送到浏览器内核，然后浏览器内核将位图复制到屏幕上。

第二点，操作系统没有将用户输入事件直接传递给渲染进程，而是将这些事件传递给浏览器内核。然后浏览器内核再根据当前浏览器界面的状态来判断如何调度这些事件，如果当前焦点位于浏览器地址栏中，则输入事件会在浏览器内核内部处理；如果当前焦点在页面的区域内，则浏览器内核会将输入事件转发给渲染进程。

之所以这样设计，就是为了限制渲染进程有监控到用户输入事件的能力，所以所有的键盘鼠标事件都是由浏览器内核来接收的，然后浏览器内核再通过IPC将这些事件发送给渲染进程。

上面我们分析了由于渲染进程引入了安全沙箱，所以浏览器的持久存储、网络访问和用户交互等功能都不能在渲染进程内直接使用了，因此我们需要把这些功能迁移到浏览器内核中去实现，这让原本比较简单的流程变得复杂了。

理解这些限制，我们就能解释开始提出的两个问题了。

站点隔离（Site Isolation）

所谓站点隔离是指Chrome将同一站点（包含了相同根域名和相同协议的地址）中相互关联的页面放到同一个渲染进程中执行。

最开始Chrome划分渲染进程是以标签页为单位，也就是说整个标签页会被划分给某个渲染进程。但是，按照标签页划分渲染进程存在一些问题，原因就是一个标签页中可能包含了多个iframe，而这些iframe又有可能来自于不同的站点，这就导致了多个不同站点中的内容通过iframe同时运行在同一个渲染进程中。

目前所有操作系统都面临着两个A级漏洞——幽灵（Spectre）和熔毁（Meltdown），这两个漏洞是由处理器架构导致的，很难修补，黑客通过这两个漏洞可以直接入侵到进程的内部，如果入侵的进程没有安全沙箱的保护，那么黑客还可以发起对操作系统的攻击。

所以如果一个银行站点包含了一个恶意iframe，然后这个恶意的iframe利用这两个A级漏洞去入侵渲染进程，那么恶意程序就能读取银行站点渲染进程内的所有内容了，这对于用户来说就存在很大的风险了。

因此Chrome几年前就开始重构代码，将标签级的渲染进程重构为iframe级的渲染进程，然后严格按照同一站点的策略来分配渲染进程，这就是Chrome中的站点隔离。

实现了站点隔离，就可以将恶意的iframe隔离在恶意进程内部，使得它无法继续访问其他iframe进程的内容，因此也就无法攻击其他站点了。

值得注意是，2019年10月20日Chrome团队宣布安卓版的Chrome已经全面支持站点隔离，你可以参考文中链接。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容。

首先我们分析了单进程浏览器在系统安全方面的不足，如果浏览器存在漏洞，那么黑客就有机会通过页面对系统发起攻击。

因此在设计现代浏览器的体系架构时，就考虑到这个问题了。于是，在多进程的基础之上引入了安全沙箱，有了安全沙箱，就可以将操作系统和渲染进程进行隔离，这样即便渲染进程由于漏洞被攻击，也不会影响到操作系统的。

由于渲染进程采用了安全沙箱，所以在渲染进程内部不能与操作系统直接交互，于是就在浏览器内核中实现了持久存储、网络访问和用户交互等一系列与操作系统交互的功能，然后通过IPC和渲染进程进行交互。

最后我们还分析了Chrome中最新的站点隔离功能。由于最初都是按照标签页来划分渲染进程的，所以如果一个标签页里面有多个不同源的iframe，那么这些iframe也会被分配到同一个渲染进程中，这样就很容易让黑客通过iframe来攻击当前渲染进程。而站点隔离会将不同源的iframe分配到不同的渲染进程中，这样即使黑客攻击恶意iframe的渲染进程，也不会影响到其他渲染进程的。

今天介绍的内容和概念都比较多，看上去离前端比较远，不过这些内容会影响你对浏览器整体架构的理解，而深入理解了浏览器架构能帮助你更加深刻地理解前端内容。为了方便你的理解，我把一些参考资料放到了文章的最后，有需要的话你可以拿来参考。

思考时间

今天留给你的思考题：你认为安全沙箱能防止XSS或者CSRF一类的攻击的吗？为什么？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

参考资料

1. 安全沙箱的设计参考了最小权限原则
2. The Security Architecture of the Chromium Browser
3. The Security Architecture of the Chromium Browser-ppt
4. chromium site-isolation
5. Site Isolation
6. Site Isolation: Process Separation for Web Sites within the Browser

前面三篇文章我们主要围绕同源策略介绍了Web页面安全的相关内容，那今天我们把视野向外延伸，来聊聊页面安全和操作系统安全之间的关系。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中，我们分析了浏览器架构的发展史，在最开始的阶段，浏览器是单进程的，这意味着渲染过程、JavaScript执行过程、网络加载过程、UI绘制过程和页面显示过程等都是在同一个进程中执行的，这种结构虽然简单，但是也带来了很多问题。

从稳定性视角来看，单进程架构的浏览器是不稳定的，因为只要浏览器进程中的任意一个功能出现异常都有可能影响到整个浏览器，如页面卡死、浏览器崩溃等。不过浏览器的稳定性并不是本文讨论的重点，我们今天主要聊的是**浏览器架构是如何影响到操作系统安全的**。

浏览器本身的漏洞是单进程浏览器的一个主要问题，如果浏览器被曝出存在漏洞，那么在这些漏洞没有被及时修复的情况下，黑客就有可能通过恶意的页面向浏览器中注入恶意程序，其中最常见的攻击方式是利用**缓冲区溢出**，不过需要**注意这种类型的攻击和XSS注入的脚本是不一样的**。

- XSS攻击只是将恶意的JavaScript脚本注入到页面中，虽然能窃取一些Cookie相关的数据，但是XSS无法对操作系统进行攻击。

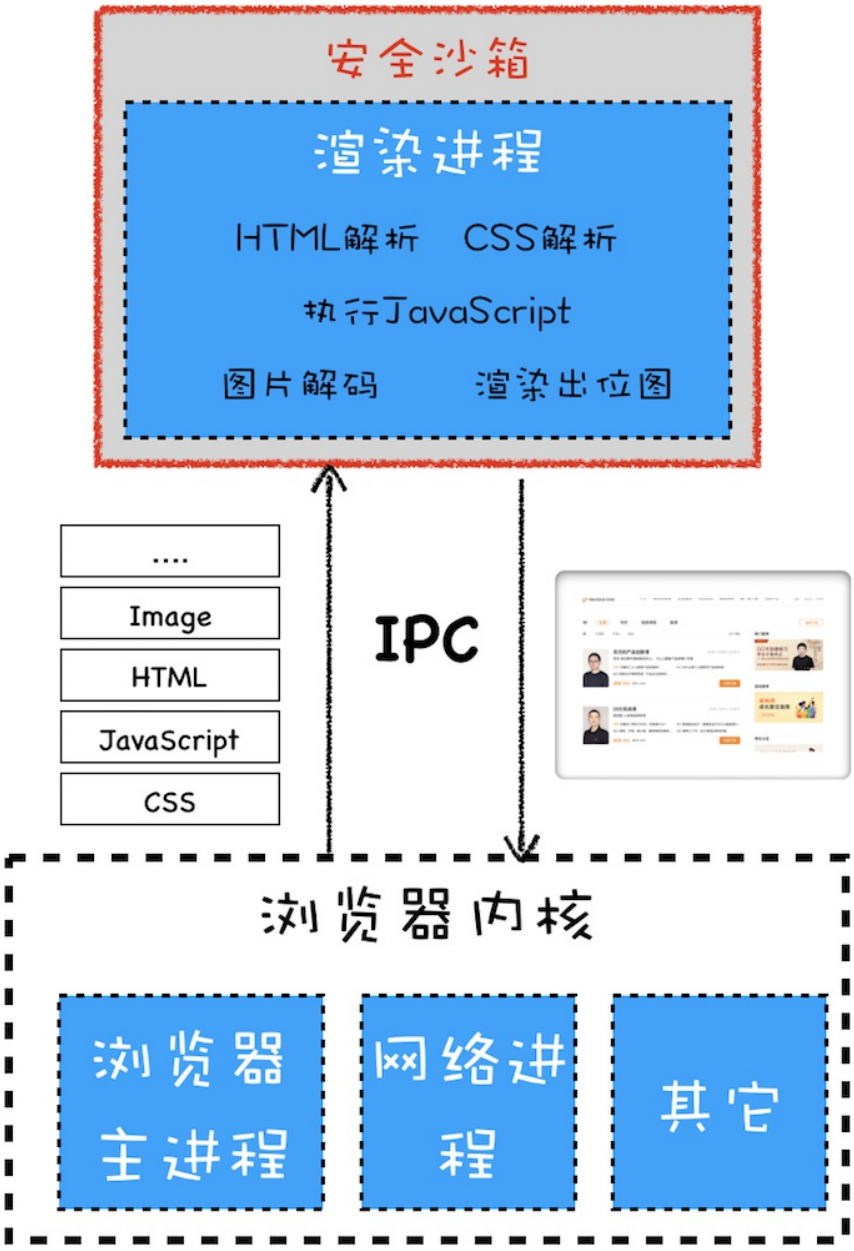
- 而通过浏览器漏洞进行的攻击是可以入侵到浏览器进程内部的，可以读取和修改浏览器进程内部的任意内容，还可以穿透浏览器，在用户的操作系统上悄悄地安装恶意软件、监听用户键盘输入信息以及读取用户硬盘上的文件内容。

和XSS攻击页面相比，这类攻击无疑是枚“核弹”，它会将整个操作系统的内容都暴露给黑客，这样我们操作系统上所有的资料都是不安全的了。

安全视角下的多进程架构

现代浏览器的设计目标是**安全**、**快速**和**稳定**，而这种核弹级杀伤力的安全问题就是一个很大的潜在威胁，因此在设计现代浏览器的体系架构时，需要解决这个问题。

我们知道现代浏览器采用了多进程架构，将渲染进程和浏览器主进程做了分离，完整的进程架构我们已经在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中介绍过了，这里我就不重复介绍了。下面我们重点从操作系统安全的视角来看看浏览器的多进程架构，如下图：



浏览器内核和渲染进程

观察上图，我们知道浏览器被划分为**浏览器内核**和**渲染内核**两个核心模块，其中浏览器内核是由网络进程、浏览器主进程和GPU进程组成的，渲染内核就是渲染进程。那如果我们在浏览器中打开一个页面，这两个模块是怎么配合的呢？

所有的网络资源都是通过浏览器内核来下载的，下载后的资源会通过IPC将其提交给渲染进程（浏览器内核和渲染进程之间都是通过IPC来通信的）。然后渲染进程会对这些资源进行解析、绘制等操作，最终生成一幅图片。但是渲染进程并不负责将图片显示到界面上，而是将最终生成的图片提交给浏览器内核模块，由浏览器内核模块负责显示这张图片。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》中我们分析过，设计现代浏览器体系架构时，将浏览器划分为不同的进程是为了增加其稳定性。虽然设计成了多进程架构，不过这些模块之间的沟通方式却有些复杂，也许你还有以下问题：

- 为什么一定要通过浏览器内核去请求资源，再将数据转发给渲染进程，而不直接从进程内部去请求网络资源？
- 为什么渲染进程只负责生成页面图片，生成图片还要经过IPC通知浏览器内核模块，然后让浏览器内核去负责展示图片？

通过以上方式不是增加了工程的复杂度吗？

要解释现代浏览器为什么要把这个流程弄得这么复杂，我们就得从系统安全的角度来分析。

安全沙箱

不过在解释这些问题之前，我们得先看看什么是安全沙箱。

上面我们分析过了，由于渲染进程需要执行DOM解析、CSS解析、网络图片解码等操作，如果渲染进程中存在系统级别的漏洞，那么以上操作就有可能让恶意的站点获取到渲染进程的控制权限，进而又获取操作系统的控制权限，这对于用户来说是非常危险的。

因为网络资源的内容存在着各种可能性，所以浏览器会默认所有的网络资源都是不可信的，都是不安全的。但谁也不能保证浏览器不存在漏洞，只要出现漏洞，黑客就可以通过网络内容对用户发起攻击。

我们知道，如果你下载了一个恶意程序，但是没有执行它，那么恶意程序是不会生效的。同理，浏览器之于网络内容也是如此，浏览器可以安全地下载各种网络资源，但是如果要执行这些网络资源，比如解析HTML、解析CSS、执行JavaScript、图片编解码等操作，就需要非常谨慎了，因为一不小心，黑客就会利用这些操作对含有漏洞的浏览器发起攻击。

基于以上原因，我们需要在渲染进程和操作系统之间建一道墙，即便渲染进程由于存在漏洞被黑客攻击，但由于这道墙，黑客就获取不到渲染进程之外的任何操作权限。**将渲染进程和操作系统隔离的这道墙就是我们要聊的安全沙箱**。

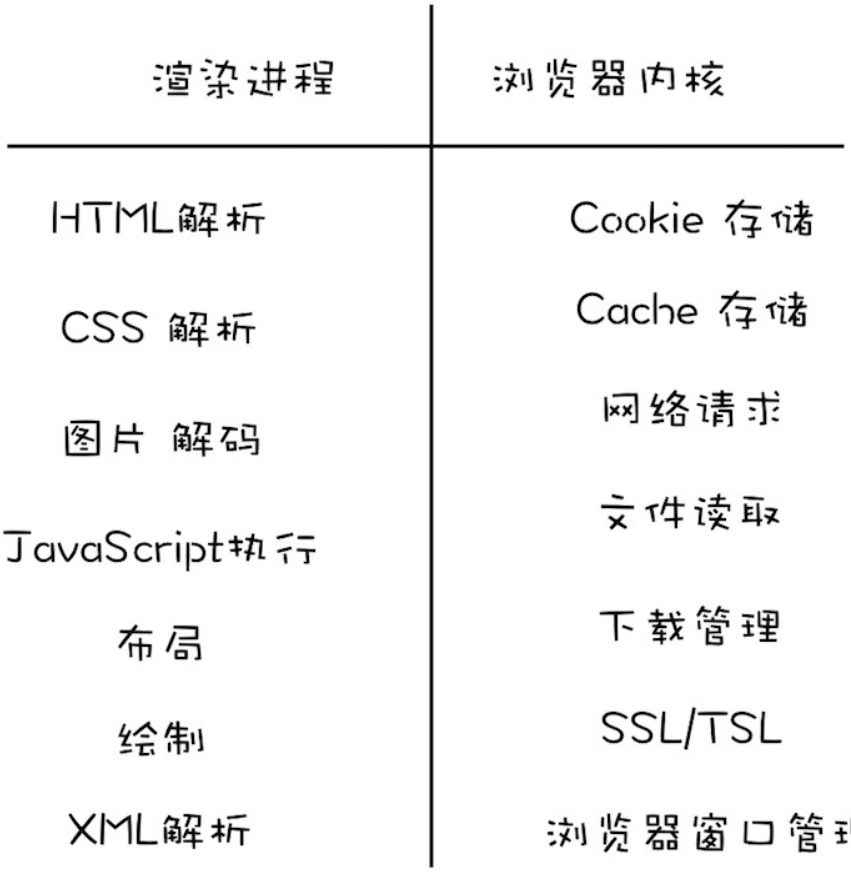
浏览器中的安全沙箱是利用操作系统提供的安全技术，让渲染进程在执行过程中无法访问或者修改操作系统中的数据，在渲染进程需要访问系统资源的时候，需要通过浏览器内核来实现，然后将访问的结果通过IPC转发给渲染进程。

安全沙箱最小的保护单位是进程。因为单进程浏览器需要频繁访问或者修改操作系统的数据，所以单进程浏览器是无法被安全沙箱保护的，而现代浏览器采用的多进程架构使得安全沙箱可以发挥作用。

安全沙箱如何影响各个模块功能

我们知道安全沙箱最小的保护单位是进程，并且能限制进程对操作系统资源的访问和修改，这就意味着如果要让安全沙箱应用在某个进程上，那么这个进程必须没有读写操作系统的功能，比如读写本地文件、发起网络请求、调用GPU接口等。

了解了被安全沙箱保护的进程会有一系列的受限操作之后，接下来我们就可以分析渲染进程和浏览器内核各自都有哪些职责，如下图：



浏览器内核和渲染进程各自职责

通过该图，我们可以看到由于渲染进程需要安全沙箱的保护，因此需要把在渲染进程内部涉及到和系统交互的功能都转移到浏览器内核中去实现。

那安全沙箱是如何影响到各个模块功能的呢？

1. 持久存储

我们先来看看安全沙箱是如何影响到浏览器持久存储的。由于安全沙箱需要负责确保渲染进程无法直接访问用户的文件系统，但是在渲染进程内部有访问Cookie的需求、有上传文件的需求，为了解决这些文件的访问需求，所以现代浏览器将读写文件的操作全部放在了浏览器内核中实现，然后通过IPC将操作结果转发给渲染进程。

具体地讲，如下文件内容的读写都是在浏览器内核中完成的：

- 存储Cookie数据的读写。通常浏览器内核会维护一个存放所有Cookie的Cookie数据库，然后当渲染进程通过JavaScript来读取Cookie时，渲染进程会通过IPC将读取Cookie的信息发送给浏览器内核，浏览器内核读取Cookie之后再将内容返回给渲染进程。
- 一些缓存文件的读写也是由浏览器内核实现的，比如网络文件缓存的读取。

2. 网络访问

同样有了安全沙箱的保护，在渲染进程内部也是不能直接访问网络的，如果要访问网络，则需要通过浏览器内核。不过浏览器内核在处理URL请求之前，会检查渲染进程是否有权限请求该URL，比如检查XMLHttpRequest或者Fetch是否是跨站点请求，或者检测HTTPS的站点中是否包含了HTTP的请求。

3. 用户交互

渲染进程实现了安全沙箱，还影响到了一个非常重要的用户交互功能。

通常情况下，如果你要实现一个UI程序，操作系统会提供一个界面给你，该界面允许应用程序与用户交互，允许应用程序在该界面上进行绘制，比如Windows提供的是HWND，Linux提供的X Window，我们就把HWND和X Window统称为窗口句柄。应用程序可以在窗口句柄上进行绘制和接收键盘鼠标消息。

不过在现代浏览器中，由于每个渲染进程都有安全沙箱的保护，所以在渲染进程内部是无法直接操作窗口句柄的，这也是为了限制渲染进程监控到用户的输入事件。

由于渲染进程不能直接访问**窗口句柄**，所以渲染进程需要完成以下两点大的改变。

第一点，渲染进程需要渲染出位图。为了向用户显示渲染进程渲染出来的位图，渲染进程需要将生成好的位图发送到浏览器内核，然后浏览器内核将位图复制到屏幕上。

第二点，操作系统没有将用户输入事件直接传递给渲染进程，而是将这些事件传递给浏览器内核。然后浏览器内核再根据当前浏览器界面的状态来判断如何调度这些事件，如果当前焦点位于浏览器地址栏中，则输入事件会在浏览器内核内部处理；如果当前焦点在页面的区域内，则浏览器内核会将输入事件转发给渲染进程。

之所以这样设计，就是为了限制渲染进程有监控到用户输入事件的能力，所以所有的键盘鼠标事件都是由浏览器内核来接收的，然后浏览器内核再通过IPC将这些事件发送给渲染进程。

上面我们分析了由于渲染进程引入了安全沙箱，所以浏览器的持久存储、网络访问和用户交互等功能都不能在渲染进程内直接使用了，因此我们需要把这些功能迁移到浏览器内核中去实现，这让原本比较简单的流程变得复杂了。

理解这些限制，我们就能解释开始提出的两个问题了。

站点隔离（Site Isolation）

所谓站点隔离是指Chrome将同一站点（包含了相同根域名和相同协议的地址）中相互关联的页面放到同一个渲染进程中执行。

最开始Chrome划分渲染进程是以标签页为单位，也就是说整个标签页会被划分给某个渲染进程。但是，按照标签页划分渲染进程存在一些问题，原因就是一个标签页中可能包含了多个iframe，而这些iframe又有可能来自于不同的站点，这就导致了多个不同站点中的内容通过iframe同时运行在同一个渲染进程中。

目前所有操作系统都面临着两个A级漏洞——幽灵（Spectre）和熔毁（Meltdown），这两个漏洞是由处理器架构导致的，很难修补，黑客通过这两个漏洞可以直接入侵到进程的内部，如果入侵的进程没有安全沙箱的保护，那么黑客还可以发起对操作系统的攻击。

所以如果一个银行站点包含了一个恶意iframe，然后这个恶意的iframe利用这两个A级漏洞去入侵渲染进程，那么恶意程序就能读取银行站点渲染进程内的所有内容了，这对于用户来说就存在很大的风险了。

因此Chrome几年前就开始重构代码，将标签级的渲染进程重构为iframe级的渲染进程，然后严格按照同一站点的策略来分配渲染进程，这就是Chrome中的站点隔离。

实现了站点隔离，就可以将恶意的iframe隔离在恶意进程内部，使得它无法继续访问其他iframe进程的内容，因此也就无法攻击其他站点了。

值得注意是，2019年10月20日Chrome团队宣布安卓版的Chrome已经全面支持站点隔离，你可以参考文中链接。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容。

首先我们分析了单进程浏览器在系统安全方面的不足，如果浏览器存在漏洞，那么黑客就有机会通过页面对系统发起攻击。

因此在设计现代浏览器的体系架构时，就考虑到这个问题了。于是，在多进程的基础之上引入了安全沙箱，有了安全沙箱，就可以将操作系统和渲染进程进行隔离，这样即便渲染进程由于漏洞被攻击，也不会影响到操作系统的。

由于渲染进程采用了安全沙箱，所以在渲染进程内部不能与操作系统直接交互，于是就在浏览器内核中实现了持久存储、网络访问和用户交互等一系列与操作系统交互的功能，然后通过IPC和渲染进程进行交互。

最后我们还分析了Chrome中最新的站点隔离功能。由于最初都是按照标签页来划分渲染进程的，所以如果一个标签页里面有多个不同源的iframe，那么这些iframe也会被分配到同一个渲染进程中，这样就很容易让黑客通过iframe来攻击当前渲染进程。而站点隔离会将不同源的iframe分配到不同的渲染进程中，这样即使黑客攻击恶意iframe的渲染进程，也不会影响到其他渲染进程的。

今天介绍的内容和概念都比较多，看上去离前端比较远，不过这些内容会影响你对浏览器整体架构的理解，而深入理解了浏览器架构能帮助你更加深刻地理解前端内容。为了方便你的理解，我把一些参考资料放到了文章的最后，有需要的话你可以拿来参考。

思考时间

今天留给你的思考题：你认为安全沙箱能防止XSS或者CSRF一类的攻击的吗？为什么？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

参考资料

1. 安全沙箱的设计参考了最小权限原则
2. The Security Architecture of the Chromium Browser
3. The Security Architecture of the Chromium Browser-ppt
4. chromium site-isolation
5. Site Isolation
6. Site Isolation: Process Separation for Web Sites within the Browser

前面三篇文章我们主要围绕同源策略介绍了Web页面安全的相关内容，那今天我们把视野向外延伸，来聊聊页面安全和操作系统安全之间的关系。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中，我们分析了浏览器架构的发展史，在最开始的阶段，浏览器是单进程的，这意味着渲染过程、JavaScript执行过程、网络加载过程、UI绘制过程和页面显示过程等都是在同一个进程中执行的，这种结构虽然简单，但是也带来了很多问题。

从稳定性视角来看，单进程架构的浏览器是不稳定的，因为只要浏览器进程中的任意一个功能出现异常都有可能影响到整个浏览器，如页面卡死、浏览器崩溃等。不过浏览器的稳定性并不是本文讨论的重点，我们今天主要聊的是**浏览器架构是如何影响到操作系统安全的**。

浏览器本身的漏洞是单进程浏览器的一个主要问题，如果浏览器被曝出存在漏洞，那么在这些漏洞没有被及时修复的情况下，黑客就有可能通过恶意的页面向浏览器中注入恶意程序，其中最常见的攻击方式是利用**缓冲区溢出**，不过需要**注意这种类型的攻击和XSS注入的脚本是不一样的**。

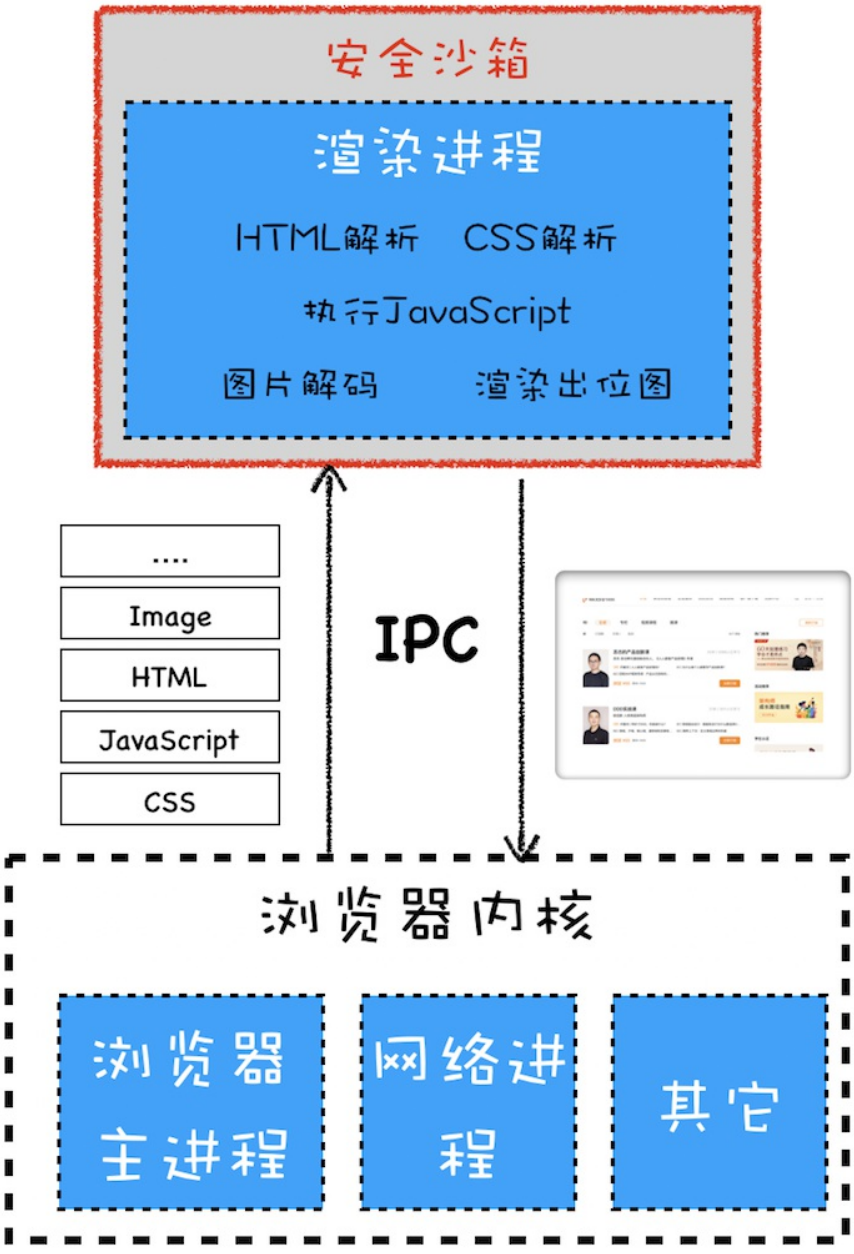
- XSS攻击只是将恶意的JavaScript脚本注入到页面中，虽然能窃取一些Cookie相关的数据，但是XSS无法对操作系统进行攻击。
- 而通过浏览器漏洞进行的攻击是可以入侵到浏览器进程内部的，可以读取和修改浏览器进程内部的任意内容，还可以穿透浏览器，在用户的操作系统上悄悄地安装恶意软件、监听用户键盘输入信息以及读取用户硬盘上的文件内容。

和XSS攻击页面相比，这类攻击无疑是枚“核弹”，它会将整个操作系统的内容都暴露给黑客，这样我们操作系统上所有的资料都是不安全的了。

安全视角下的多进程架构

现代浏览器的设计目标是**安全、快速**和**稳定**，而这种核弹级杀伤力的安全问题就是一个很大的潜在威胁，因此在设计现代浏览器的体系架构时，需要解决这个问题。

我们知道现代浏览器采用了多进程架构，将渲染进程和浏览器主进程做了分离，完整的进程架构我们已经在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中介绍过了，这里我就不重复介绍了。下面我们重点从操作系统安全的视角来看看浏览器的多进程架构，如下图：



浏览器内核和渲染进程

观察上图，我们知道浏览器被划分为**浏览器内核**和**渲染内核**两个核心模块，其中浏览器内核是由网络进程、浏览器主进程和GPU进程组成的，渲染内核就是渲染进程。那如果我们在浏览器中打开一个页面，这两个模块是怎么配合的呢？

所有的网络资源都是通过浏览器内核来下载的，下载后的资源会通过IPC将其提交给渲染进程（浏览器内核和渲染进程之间都是通过IPC来通信的）。然后渲染进程会对这些资源进行解析、绘制等操作，最终生成一幅图片。但是渲染进程并不负责将图片显示到界面上，而是将最终生成的图片提交给浏览器内核模块，由浏览器内核模块负责显示这张图片。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》中我们分析过，设计现代浏览器体系架构时，将浏览器划分为不同的进程是为了增加其稳定性。虽然设计成了多进程架构，不过这些模块之间的沟通方式却有些复杂，也许你还有以下问题：

- 为什么一定要通过浏览器内核去请求资源，再将数据转发给渲染进程，而不直接从进程内部去请求网络资源？
- 为什么渲染进程只负责生成页面图片，生成图片还要经过IPC通知浏览器内核模块，然后让浏览器内核去负责展示图片？

通过以上方式不是增加了工程的复杂度吗？

要解释现代浏览器为什么要把这个流程弄得这么复杂，我们就得从系统安全的角度来分析。

安全沙箱

不过在解释这些问题之前，我们得先看看什么是安全沙箱。

上面我们分析过了，由于渲染进程需要执行DOM解析、CSS解析、网络图片解码等操作，如果渲染进程中存在系统级别的漏洞，那么以上操作就有可能让恶意的站点获取到渲染进程的控制权限，进而又获取操作系统的控制权限，这对于用户来说是非常危险的。

因为网络资源的内容存在着各种可能性，所以浏览器会默认所有的网络资源都是不可信的，都是不安全的。但谁也不能保证浏览器不存在漏洞，只要出现漏洞，黑客就可以通过网络内容对用户发起攻击。

我们知道，如果你下载了一个恶意程序，但是没有执行它，那么恶意程序是不会生效的。同理，浏览器之于网络内容也是如此，浏览器可以安全地下载各种网络资源，但是如果要执行这些网络资源，比如解析HTML、解析CSS、执行JavaScript、图片编解码等操作，就需要非常谨慎了，因为一不小心，黑客就会利用这些操作对含有漏洞的浏览器发起攻击。

基于以上原因，我们需要在渲染进程和操作系统之间建一道墙，即便渲染进程由于存在漏洞被黑客攻击，但由于这道墙，黑客就获取不到渲染进程之外的任何操作权限。**将渲染进程和操作系统隔离的这道墙就是我们要聊的安全沙箱**。

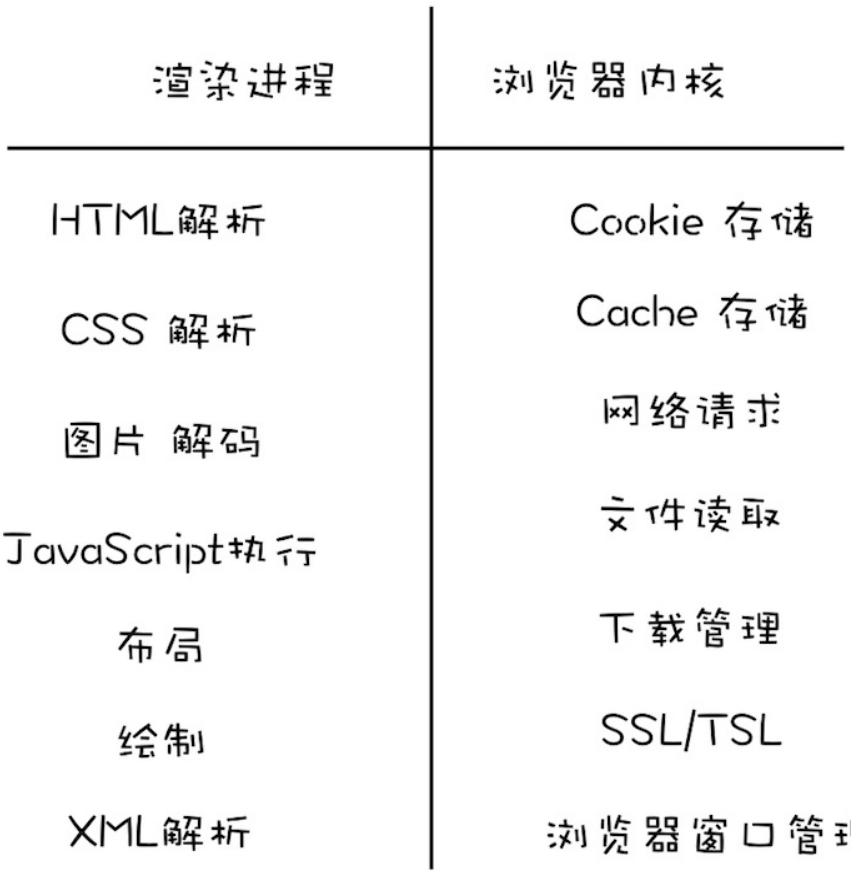
浏览器中的安全沙箱是利用操作系统提供的安全技术，让渲染进程在执行过程中无法访问或者修改操作系统中的数据，在渲染进程需要访问系统资源的时候，需要通过浏览器内核来实现，然后将访问的结果通过IPC转发给渲染进程。

安全沙箱最小的保护单位是进程。因为单进程浏览器需要频繁访问或者修改操作系统的数据，所以单进程浏览器是无法被安全沙箱保护的，而现代浏览器采用的多进程架构使得安全沙箱可以发挥作用。

安全沙箱如何影响各个模块功能

我们知道安全沙箱最小的保护单位是进程，并且能限制进程对操作系统资源的访问和修改，这就意味着如果要让安全沙箱应用在某个进程上，那么这个进程必须没有读写操作系统的功能，比如读写本地文件、发起网络请求、调用GPU接口等。

了解了被安全沙箱保护的进程会有一系列的受限操作之后，接下来我们就可以分析渲染进程和浏览器内核各自都有哪些职责，如下图：



浏览器内核和渲染进程各自职责

通过该图，我们可以看到由于渲染进程需要安全沙箱的保护，因此需要把在渲染进程内部涉及到和系统交互的功能都转移到浏览器内核中去实现。

那安全沙箱是如何影响到各个模块功能的呢？

1. 持久存储

我们先来看看安全沙箱是如何影响到浏览器持久存储的。由于安全沙箱需要负责确保渲染进程无法直接访问用户的文件系统，但是在渲染进程内部有访问Cookie的需求、有上传文件的需求，为了解决这些文件的访问需求，所以现代浏览器将读写文件的操作全部放在了浏览器内核中实现，然后通过IPC将操作结果转发给渲染进程。

具体地讲，如下文件内容的读写都是在浏览器内核中完成的：

- 存储Cookie数据的读写。通常浏览器内核会维护一个存放所有Cookie的Cookie数据库，然后当渲染进程通过JavaScript来读取Cookie时，渲染进程会通过IPC将读取Cookie的信息发送给浏览器内核，浏览器内核读取Cookie之后再将内容返回给渲染进程。
- 一些缓存文件的读写也是由浏览器内核实现的，比如网络文件缓存的读取。

2. 网络访问

同样有了安全沙箱的保护，在渲染进程内部也是不能直接访问网络的，如果要访问网络，则需要通过浏览器内核。不过浏览器内核在处理URL请求之前，会检查渲染进程是否有权限请求该URL，比如检查XMLHttpRequest或者Fetch是否是跨站点请求，或者检测HTTPS的站点中是否包含了HTTP的请求。

3. 用户交互

渲染进程实现了安全沙箱，还影响到了一个非常重要的用户交互功能。

通常情况下，如果你要实现一个UI程序，操作系统会提供一个界面给你，该界面允许应用程序与用户交互，允许应用程序在该界面上进行绘制，比如Windows提供的是HWND，Linux提供的X Window，我们就把HWND和X Window统称为窗口句柄。应用程序可以在窗口句柄上进行绘制和接收键盘鼠标消息。

不过在现代浏览器中，由于每个渲染进程都有安全沙箱的保护，所以在渲染进程内部是无法直接操作窗口句柄的，这也是为了限制渲染进程监控到用户的输入事件。

由于渲染进程不能直接访问**窗口句柄**，所以渲染进程需要完成以下两点大的改变。

第一点，渲染进程需要渲染出位图。为了向用户显示渲染进程渲染出来的位图，渲染进程需要将生成好的位图发送到浏览器内核，然后浏览器内核将位图复制到屏幕上。

第二点，操作系统没有将用户输入事件直接传递给渲染进程，而是将这些事件传递给浏览器内核。然后浏览器内核再根据当前浏览器界面的状态来判断如何调度这些事件，如果当前焦点位于浏览器地址栏中，则输入事件会在浏览器内核内部处理；如果当前焦点在页面的区域内，则浏览器内核会将输入事件转发给渲染进程。

之所以这样设计，就是为了限制渲染进程有监控到用户输入事件的能力，所以所有的键盘鼠标事件都是由浏览器内核来接收的，然后浏览器内核再通过IPC将这些事件发送给渲染进程。

上面我们分析了由于渲染进程引入了安全沙箱，所以浏览器的持久存储、网络访问和用户交互等功能都不能在渲染进程内直接使用了，因此我们需要把这些功能迁移到浏览器内核中去实现，这让原本比较简单的流程变得复杂了。

理解这些限制，我们就能解释开始提出的两个问题了。

站点隔离（Site Isolation）

所谓站点隔离是指Chrome将同一站点（包含了相同根域名和相同协议的地址）中相互关联的页面放到同一个渲染进程中执行。

最开始Chrome划分渲染进程是以标签页为单位，也就是说整个标签页会被划分给某个渲染进程。但是，按照标签页划分渲染进程存在一些问题，原因就是一个标签页中可能包含了多个iframe，而这些iframe又有可能来自于不同的站点，这就导致了多个不同站点中的内容通过iframe同时运行在同一个渲染进程中。

目前所有操作系统都面临着两个A级漏洞——幽灵（Spectre）和熔毁（Meltdown），这两个漏洞是由处理器架构导致的，很难修补，黑客通过这两个漏洞可以直接入侵到进程的内部，如果入侵的进程没有安全沙箱的保护，那么黑客还可以发起对操作系统的攻击。

所以如果一个银行站点包含了一个恶意iframe，然后这个恶意的iframe利用这两个A级漏洞去入侵渲染进程，那么恶意程序就能读取银行站点渲染进程内的所有内容了，这对于用户来说就存在很大的风险了。

因此Chrome几年前就开始重构代码，将标签级的渲染进程重构为iframe级的渲染进程，然后严格按照同一站点的策略来分配渲染进程，这就是Chrome中的站点隔离。

实现了站点隔离，就可以将恶意的iframe隔离在恶意进程内部，使得它无法继续访问其他iframe进程的内容，因此也就无法攻击其他站点了。

值得注意是，2019年10月20日Chrome团队宣布安卓版的Chrome已经全面支持站点隔离，你可以参考文中链接。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容。

首先我们分析了单进程浏览器在系统安全方面的不足，如果浏览器存在漏洞，那么黑客就有机会通过页面对系统发起攻击。

因此在设计现代浏览器的体系架构时，就考虑到这个问题了。于是，在多进程的基础之上引入了安全沙箱，有了安全沙箱，就可以将操作系统和渲染进程进行隔离，这样即便渲染进程由于漏洞被攻击，也不会影响到操作系统的。

由于渲染进程采用了安全沙箱，所以在渲染进程内部不能与操作系统直接交互，于是就在浏览器内核中实现了持久存储、网络访问和用户交互等一系列与操作系统交互的功能，然后通过IPC和渲染进程进行交互。

最后我们还分析了Chrome中最新的站点隔离功能。由于最初都是按照标签页来划分渲染进程的，所以如果一个标签页里面有多个不同源的iframe，那么这些iframe也会被分配到同一个渲染进程中，这样就很容易让黑客通过iframe来攻击当前渲染进程。而站点隔离会将不同源的iframe分配到不同的渲染进程中，这样即使黑客攻击恶意iframe的渲染进程，也不会影响到其他渲染进程的。

今天介绍的内容和概念都比较多，看上去离前端比较远，不过这些内容会影响你对浏览器整体架构的理解，而深入理解了浏览器架构能帮助你更加深刻地理解前端内容。为了方便你的理解，我把一些参考资料放到了文章的最后，有需要的话你可以拿来参考。

思考时间

今天留给你的思考题：你认为安全沙箱能防止XSS或者CSRF一类的攻击的吗？为什么？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

参考资料

1. 安全沙箱的设计参考了最小权限原则
2. The Security Architecture of the Chromium Browser
3. The Security Architecture of the Chromium Browser-ppt
4. chromium site-isolation
5. Site Isolation
6. Site Isolation: Process Separation for Web Sites within the Browser

前面三篇文章我们主要围绕同源策略介绍了Web页面安全的相关内容，那今天我们把视野向外延伸，来聊聊页面安全和操作系统安全之间的关系。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中，我们分析了浏览器架构的发展史，在最开始的阶段，浏览器是单进程的，这意味着渲染过程、JavaScript执行过程、网络加载过程、UI绘制过程和页面显示过程等都是在同一个进程中执行的，这种结构虽然简单，但是也带来了很多问题。

从稳定性视角来看，单进程架构的浏览器是不稳定的，因为只要浏览器进程中的任意一个功能出现异常都有可能影响到整个浏览器，如页面卡死、浏览器崩溃等。不过浏览器的稳定性并不是本文讨论的重点，我们今天主要聊的是**浏览器架构是如何影响到操作系统安全的**。

浏览器本身的漏洞是单进程浏览器的一个主要问题，如果浏览器被曝出存在漏洞，那么在这些漏洞没有被及时修复的情况下，黑客就有可能通过恶意的页面向浏览器中注入恶意程序，其中最常见的攻击方式是利用**缓冲区溢出**，不过需要**注意这种类型的攻击和XSS注入的脚本是不一样的**。

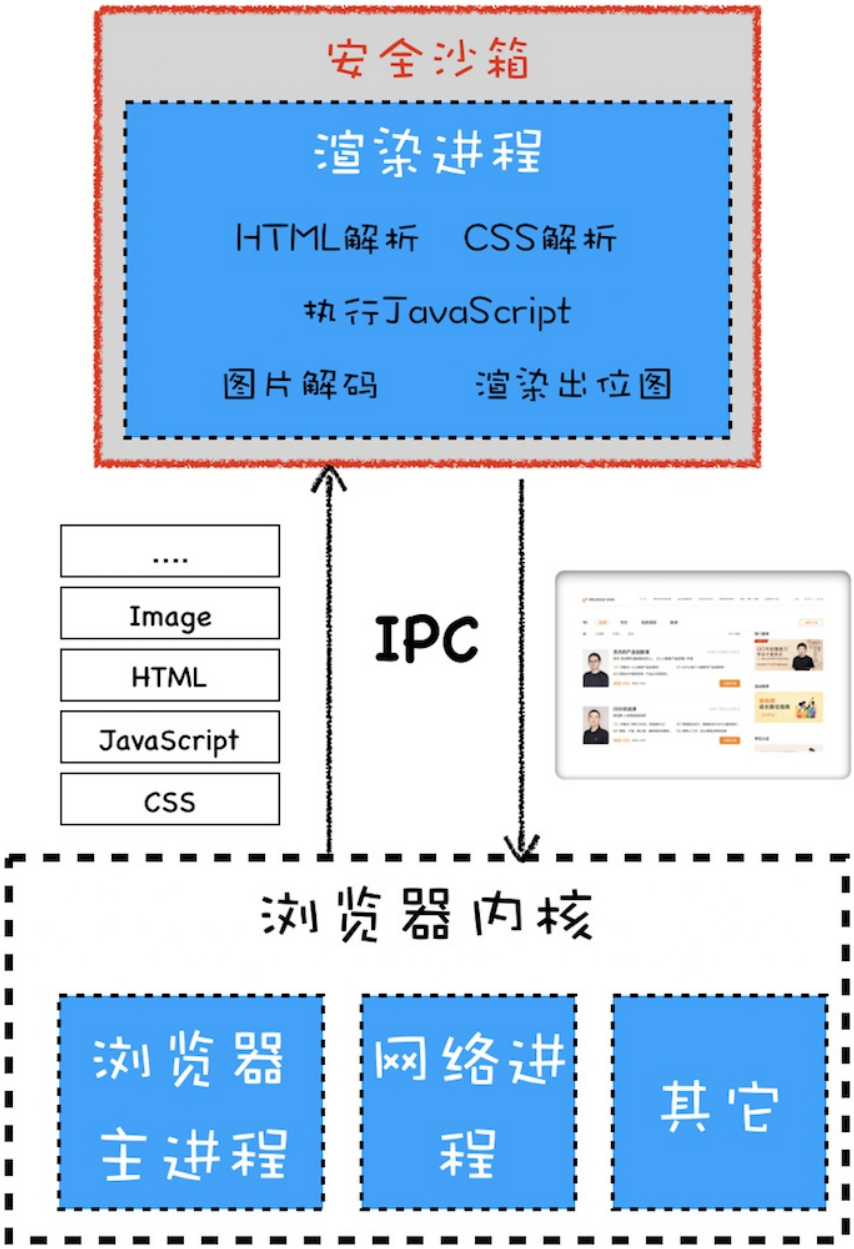
- XSS攻击只是将恶意的JavaScript脚本注入到页面中，虽然能窃取一些Cookie相关的数据，但是XSS无法对操作系统进行攻击。
- 而通过浏览器漏洞进行的攻击是可以入侵到浏览器进程内部的，可以读取和修改浏览器进程内部的任意内容，还可以穿透浏览器，在用户的操作系统上悄悄地安装恶意软件、监听用户键盘输入信息以及读取用户硬盘上的文件内容。

和XSS攻击页面相比，这类攻击无疑是枚“核弹”，它会将整个操作系统的内容都暴露给黑客，这样我们操作系统上所有的资料都是不安全的了。

安全视角下的多进程架构

现代浏览器的设计目标是**安全、快速**和**稳定**，而这种核弹级杀伤力的安全问题就是一个很大的潜在威胁，因此在设计现代浏览器的体系架构时，需要解决这个问题。

我们知道现代浏览器采用了多进程架构，将渲染进程和浏览器主进程做了分离，完整的进程架构我们已经在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》那篇文章中介绍过了，这里我就不重复介绍了。下面我们重点从操作系统安全的视角来看看浏览器的多进程架构，如下图：



浏览器内核和渲染进程

观察上图，我们知道浏览器被划分为**浏览器内核**和**渲染内核**两个核心模块，其中浏览器内核是由网络进程、浏览器主进程和GPU进程组成的，渲染内核就是渲染进程。那如果我们在浏览器中打开一个页面，这两个模块是怎么配合的呢？

所有的网络资源都是通过浏览器内核来下载的，下载后的资源会通过IPC将其提交给渲染进程（浏览器内核和渲染进程之间都是通过IPC来通信的）。然后渲染进程会对这些资源进行解析、绘制等操作，最终生成一幅图片。但是渲染进程并不负责将图片显示到界面上，而是将最终生成的图片提交给浏览器内核模块，由浏览器内核模块负责显示这张图片。

在《01 | Chrome架构：仅仅打开了1个页面，为什么有4个进程？》中我们分析过，设计现代浏览器体系架构时，将浏览器划分为不同的进程是为了增加其稳定性。虽然设计成了多进程架构，不过这些模块之间的沟通方式却有些复杂，也许你还有以下问题：

- 为什么一定要通过浏览器内核去请求资源，再将数据转发给渲染进程，而不直接从进程内部去请求网络资源？
- 为什么渲染进程只负责生成页面图片，生成图片还要经过IPC通知浏览器内核模块，然后让浏览器内核去负责展示图片？

通过以上方式不是增加了工程的复杂度吗？

要解释现代浏览器为什么要把这个流程弄得这么复杂，我们就得从系统安全的角度来分析。

安全沙箱

不过在解释这些问题之前，我们得先看看什么是安全沙箱。

上面我们分析过了，由于渲染进程需要执行DOM解析、CSS解析、网络图片解码等操作，如果渲染进程中存在系统级别的漏洞，那么以上操作就有可能让恶意的站点获取到渲染进程的控制权限，进而又获取操作系统的控制权限，这对于用户来说是非常危险的。

因为网络资源的内容存在着各种可能性，所以浏览器会默认所有的网络资源都是不可信的，都是不安全的。但谁也不能保证浏览器不存在漏洞，只要出现漏洞，黑客就可以通过网络内容对用户发起攻击。

我们知道，如果你下载了一个恶意程序，但是没有执行它，那么恶意程序是不会生效的。同理，浏览器之于网络内容也是如此，浏览器可以安全地下载各种网络资源，但是如果要执行这些网络资源，比如解析HTML、解析CSS、执行JavaScript、图片编解码等操作，就需要非常谨慎了，因为一不小心，黑客就会利用这些操作对含有漏洞的浏览器发起攻击。

基于以上原因，我们需要在渲染进程和操作系统之间建一道墙，即便渲染进程由于存在漏洞被黑客攻击，但由于这道墙，黑客就获取不到渲染进程之外的任何操作权限。**将渲染进程和操作系统隔离的这道墙就是我们要聊的安全沙箱**。

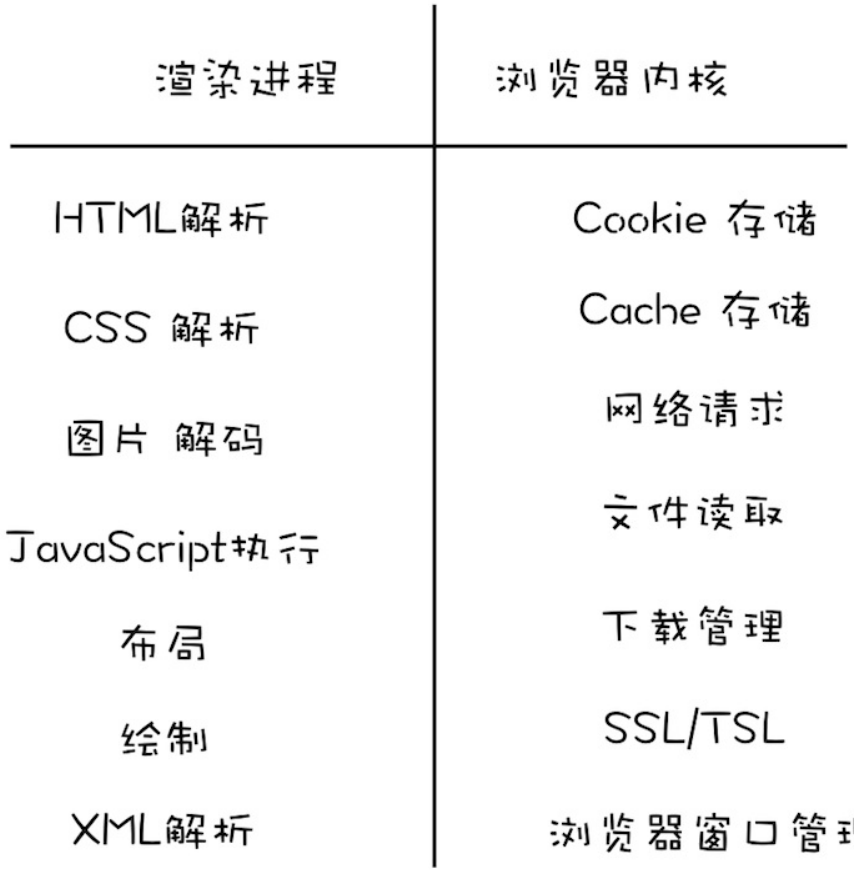
浏览器中的安全沙箱是利用操作系统提供的安全技术，让渲染进程在执行过程中无法访问或者修改操作系统中的数据，在渲染进程需要访问系统资源的时候，需要通过浏览器内核来实现，然后将访问的结果通过IPC转发给渲染进程。

安全沙箱最小的保护单位是进程。因为单进程浏览器需要频繁访问或者修改操作系统的数据，所以单进程浏览器是无法被安全沙箱保护的，而现代浏览器采用的多进程架构使得安全沙箱可以发挥作用。

安全沙箱如何影响各个模块功能

我们知道安全沙箱最小的保护单位是进程，并且能限制进程对操作系统资源的访问和修改，这就意味着如果要让安全沙箱应用在某个进程上，那么这个进程必须没有读写操作系统的功能，比如读写本地文件、发起网络请求、调用GPU接口等。

了解了被安全沙箱保护的进程会有一系列的受限操作之后，接下来我们就可以分析渲染进程和浏览器内核各自都有哪些职责，如下图：



浏览器内核和渲染进程各自职责

通过该图，我们可以看到由于渲染进程需要安全沙箱的保护，因此需要把在渲染进程内部涉及到和系统交互的功能都转移到浏览器内核中去实现。

那安全沙箱是如何影响到各个模块功能的呢？

1. 持久存储

我们先来看看安全沙箱是如何影响到浏览器持久存储的。由于安全沙箱需要负责确保渲染进程无法直接访问用户的文件系统，但是在渲染进程内部有访问Cookie的需求、有上传文件的需求，为了解决这些文件的访问需求，所以现代浏览器将读写文件的操作全部放在了浏览器内核中实现，然后通过IPC将操作结果转发给渲染进程。

具体地讲，如下文件内容的读写都是在浏览器内核中完成的：

- 存储Cookie数据的读写。通常浏览器内核会维护一个存放所有Cookie的Cookie数据库，然后当渲染进程通过JavaScript来读取Cookie时，渲染进程会通过IPC将读取Cookie的信息发送给浏览器内核，浏览器内核读取Cookie之后再将内容返回给渲染进程。
- 一些缓存文件的读写也是由浏览器内核实现的，比如网络文件缓存的读取。

2. 网络访问

同样有了安全沙箱的保护，在渲染进程内部也是不能直接访问网络的，如果要访问网络，则需要通过浏览器内核。不过浏览器内核在处理URL请求之前，会检查渲染进程是否有权限请求该URL，比如检查XMLHttpRequest或者Fetch是否是跨站点请求，或者检测HTTPS的站点中是否包含了HTTP的请求。

3. 用户交互

渲染进程实现了安全沙箱，还影响到了一个非常重要的用户交互功能。

通常情况下，如果你要实现一个UI程序，操作系统会提供一个界面给你，该界面允许应用程序与用户交互，允许应用程序在该界面上进行绘制，比如Windows提供的是HWND，Linux提供的X Window，我们就把HWND和X Window统称为窗口句柄。应用程序可以在窗口句柄上进行绘制和接收键盘鼠标消息。

不过在现代浏览器中，由于每个渲染进程都有安全沙箱的保护，所以在渲染进程内部是无法直接操作窗口句柄的，这也是为了限制渲染进程监控到用户的输入事件。

由于渲染进程不能直接访问**窗口句柄**，所以渲染进程需要完成以下两点大的改变。

第一点，渲染进程需要渲染出位图。为了向用户显示渲染进程渲染出来的位图，渲染进程需要将生成好的位图发送到浏览器内核，然后浏览器内核将位图复制到屏幕上。

第二点，操作系统没有将用户输入事件直接传递给渲染进程，而是将这些事件传递给浏览器内核。然后浏览器内核再根据当前浏览器界面的状态来判断如何调度这些事件，如果当前焦点位于浏览器地址栏中，则输入事件会在浏览器内核内部处理；如果当前焦点在页面的区域内，则浏览器内核会将输入事件转发给渲染进程。

之所以这样设计，就是为了限制渲染进程有监控到用户输入事件的能力，所以所有的键盘鼠标事件都是由浏览器内核来接收的，然后浏览器内核再通过IPC将这些事件发送给渲染进程。

上面我们分析了由于渲染进程引入了安全沙箱，所以浏览器的持久存储、网络访问和用户交互等功能都不能在渲染进程内直接使用了，因此我们需要把这些功能迁移到浏览器内核中去实现，这让原本比较简单的流程变得复杂了。

理解这些限制，我们就能解释开始提出的两个问题了。

站点隔离（Site Isolation）

所谓站点隔离是指Chrome将同一站点（包含了相同根域名和相同协议的地址）中相互关联的页面放到同一个渲染进程中执行。

最开始Chrome划分渲染进程是以标签页为单位，也就是说整个标签页会被划分给某个渲染进程。但是，按照标签页划分渲染进程存在一些问题，原因就是一个标签页中可能包含了多个iframe，而这些iframe又有可能来自于不同的站点，这就导致了多个不同站点中的内容通过iframe同时运行在同一个渲染进程中。

目前所有操作系统都面临着两个A级漏洞——幽灵（Spectre）和熔毁（Meltdown），这两个漏洞是由处理器架构导致的，很难修补，黑客通过这两个漏洞可以直接入侵到进程的内部，如果入侵的进程没有安全沙箱的保护，那么黑客还可以发起对操作系统的攻击。

所以如果一个银行站点包含了一个恶意iframe，然后这个恶意的iframe利用这两个A级漏洞去入侵渲染进程，那么恶意程序就能读取银行站点渲染进程内的所有内容了，这对于用户来说就存在很大的风险了。

因此Chrome几年前就开始重构代码，将标签级的渲染进程重构为iframe级的渲染进程，然后严格按照同一站点的策略来分配渲染进程，这就是Chrome中的站点隔离。

实现了站点隔离，就可以将恶意的iframe隔离在恶意进程内部，使得它无法继续访问其他iframe进程的内容，因此也就无法攻击其他站点了。

值得注意是，2019年10月20日Chrome团队宣布安卓版的Chrome已经全面支持站点隔离，你可以参考文中链接。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容。

首先我们分析了单进程浏览器在系统安全方面的不足，如果浏览器存在漏洞，那么黑客就有机会通过页面对系统发起攻击。

因此在设计现代浏览器的体系架构时，就考虑到这个问题了。于是，在多进程的基础之上引入了安全沙箱，有了安全沙箱，就可以将操作系统和渲染进程进行隔离，这样即便渲染进程由于漏洞被攻击，也不会影响到操作系统的。

由于渲染进程采用了安全沙箱，所以在渲染进程内部不能与操作系统直接交互，于是就在浏览器内核中实现了持久存储、网络访问和用户交互等一系列与操作系统交互的功能，然后通过IPC和渲染进程进行交互。

最后我们还分析了Chrome中最新的站点隔离功能。由于最初都是按照标签页来划分渲染进程的，所以如果一个标签页里面有多个不同源的iframe，那么这些iframe也会被分配到同一个渲染进程中，这样就很容易让黑客通过iframe来攻击当前渲染进程。而站点隔离会将不同源的iframe分配到不同的渲染进程中，这样即使黑客攻击恶意iframe的渲染进程，也不会影响到其他渲染进程的。

今天介绍的内容和概念都比较多，看上去离前端比较远，不过这些内容会影响你对浏览器整体架构的理解，而深入理解了浏览器架构能帮助你更加深刻地理解前端内容。为了方便你的理解，我把一些参考资料放到了文章的最后，有需要的话你可以拿来参考。

思考时间

今天留给你的思考题：你认为安全沙箱能防止XSS或者CSRF一类的攻击的吗？为什么？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

参考资料

1. 安全沙箱的设计参考了最小权限原则
2. The Security Architecture of the Chromium Browser
3. The Security Architecture of the Chromium Browser-ppt
4. chromium site-isolation
5. Site Isolation
6. Site Isolation: Process Separation for Web Sites within the Browser