北京大学
PEKING UNIVERSITY

ScriptChecker: To Tame Third-party Script Execution With Task Capabilities

2022.10 付烨齐

MORESHI POWERPOINT

目录

简介

MORESHI POWERPOINT

- 该文为发表于NDSS 2022的ScriptChecker: To Tame Third-party Script Execution With Task Capabilities。
- 由于第三方脚本在使用上的便利性，88.45%的网站至少包含一个第三方脚本。然而，由于这些脚本具有访问所有资源的完全权限，调用这样的第三方脚本函数可能会导致意外，甚至是恶意代码执行。

简介

MORESHI POWERPOINT

- 在本篇论文中，作者提出了ScriptChecker，这是一种基于浏览器的JS脚本检测框架，可以根据主机网页的指令有效地限制第三方脚本的执行。与现有基于JavaScript层运行的现有方案不同，ScriptChecker利用上下文分离和浏览器的安全监视器来对执行不受信任代码的任务实施按需地访问控制。主机页面可以在创建任务时灵活地为其分配资源访问能力。因此，ScriptChecker在安全性、可用性和性能方面优于现有技术。

介绍

MORESHI POWERPOINT

- 第三方脚本为专门的实用程序提供了现成的功能。例如 CryptoJS 以及jQuery等。大部分的网站都会至少包含一个这种第三方脚本。
- 然而，调用第三方脚本可能会造成非预期的资源访问甚至代码执行。

介绍

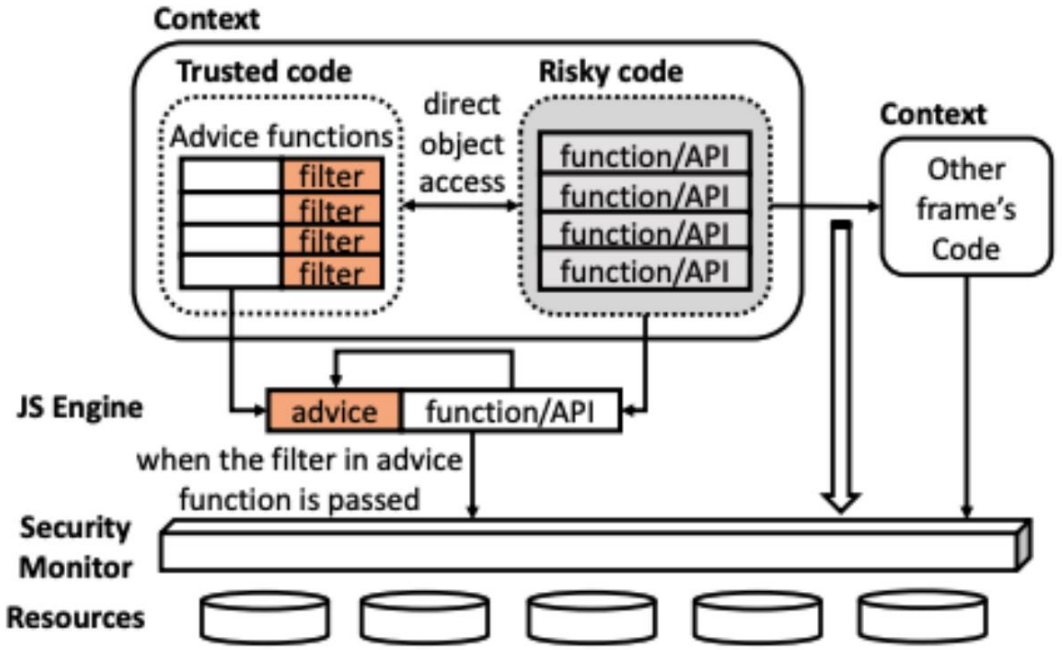
MORESHI POWERPOINT

- 第三方脚本的安全问题的挑战可以归结为两个方面。
  - 第一就是用户定义脚本（host script）以及第三方脚本（third script）同处于一个Frame以及共享origin，因此一些基于frame或者origin的隔离机制都不起作用。因此，制定一个Frame内部的第三方脚本的隔离机制是必要的。
  - 第二，不可能对所有网站的第三方脚本加载都使用同一套的脚本隔离策略。脚本隔离策略应该是特定于网站，而不是浏览器的。因此，一个理想的第三方脚本的限制策略应该允许host website动态的指定隔离策略。

现有工作

MORESHI POWERPOINT

- 更早的工作直接对JS的function call进行筛选，来决定需要禁止哪些functioncall。

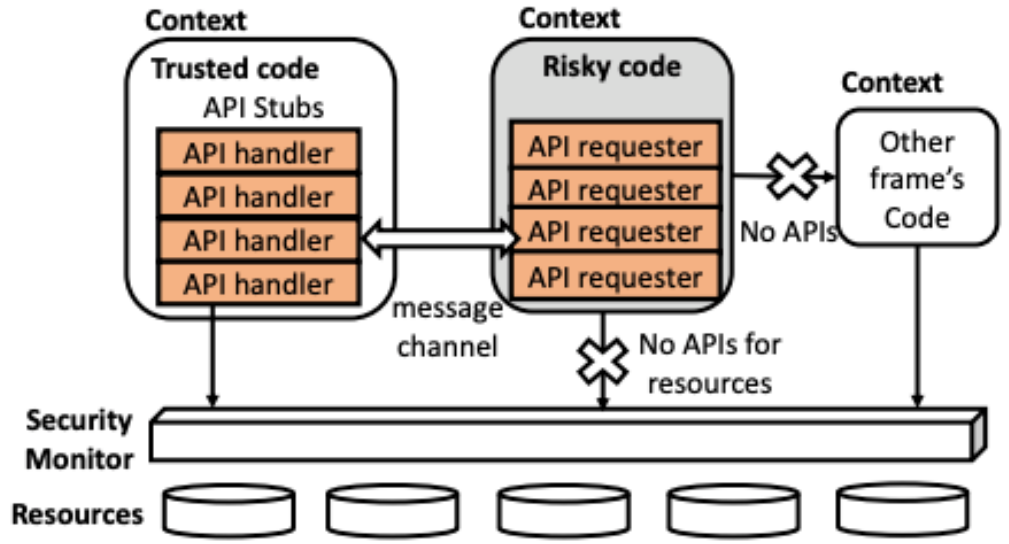


(a) Function Filter Schemes.

现有工作

MORESHI POWERPOINT

- 而最近的工作有一些是采用了单独的Javascript上下文来执行第三方脚本，并且从上下文的层面来捕获跨 isolation 的对象访问，并对这些访问进行筛选。



(b) Reference Restriction Schemes (using contexts).

缺点

MORESHI POWERPOINT

- 我们要拦截的是对于关键的DOM以及系统资源的访问，而javascript层总归来说是比较靠前的层级。从 Javascript 层很难做到百分百的拦截，因为开发人员要不断地更新敏感JS函数列表，并且防止绕过。
- 另外，即使可以做到对Thirdparty的js层面的隔离，但是Thirdparty依旧可以通过代理攻击来调用 host script 或者其他规则不同的Thirtyparty没有经过filter的函数。因此并不能从根本上解决问题。

缺点

MORESHI POWERPOINT

- 其次，当前的方法在适应性问题上也存在缺陷。对第三方脚本的限制对不同的网站或者对一个网站的不同生命周期来说是动态的。因此，静态的Filter很难满足这种变化的需求。

Chrome目前的安全策略

MORESHI POWERPOINT

- Chrome目前有两个级别的security monitor，render内部的local security monitor，以及broker进程内的kernel security monitor。
- Local security monitor 主要处理一些render本身可以处理的一些隔离，比如可以限制对DOM或者JS对象的访问以及网络响应，基于此能力可以发展origin- 或者frame- based的策略。
- Kernel security monitor 则是限制一些更为敏感的浏览器数据，比如cookies和书签等。

ScriptChecker设计

MORESHI POWERPOINT

- 本篇论文提出了ScriptCheker，一个创新的基于浏览器的第三方脚本权限控制框架。ScriptChecker的核心是，当host script需要调用third party script的时候，从chrome的 tasks 层面将调用thirdparty script的任务权限进行限制。
- 限制依赖于chrome的Security monitor。Security monitor 可以对单个task的资源访问做出限制，ScriptChecker要做的就是在这个层级加上用户可以自定义的限制条件。

ScriptChecker 优点

MORESHI POWERPOINT

- 由于ScriptChecker的设计完全基于Browser的任务层级，总体来说是比javascript engine更为底层的层级，因此ScriptChecker不会受到Javascript 层面的Filter所遇到的问题。
- 另外，ScriptChecker提供动态的权限设置。

ScriptChecker 设计

MORESHI POWERPOINT

- 基于Chrome 针对 tasks 的权限管理，ScriptChecker旨在将第三方Script在作为另外的task执行，并且从Security Monitor的角度针对该task进行限制。

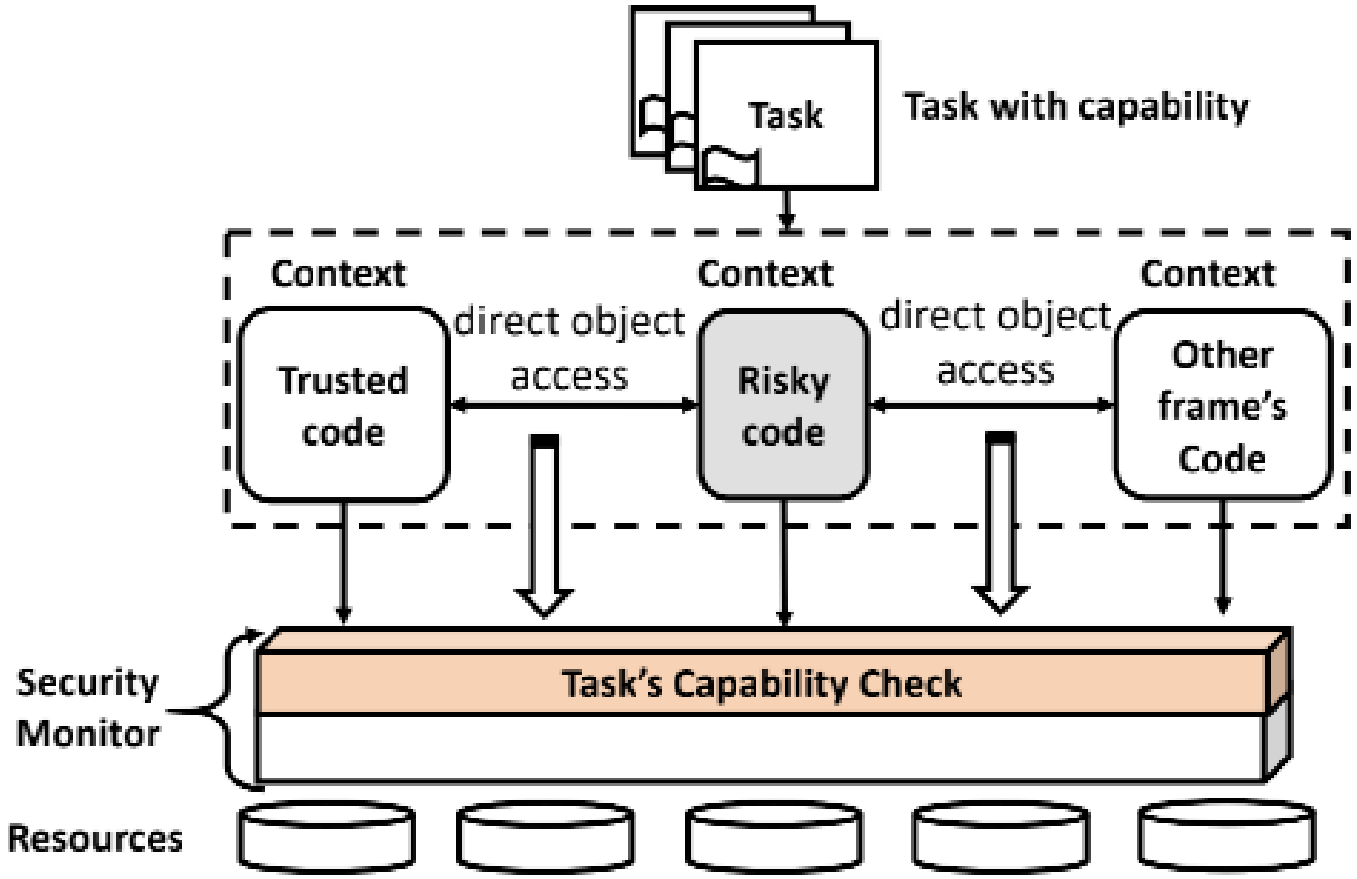
ScriptChecker 设计

MORESHI POWERPOINT

- 这里存在两个基本问题：
  - 1. 如何将第三方script的执行作为和 host script 执行不同的task？
  - 2. 如何限制Task的权限？

Sandbox Context

MORESHI POWERPOINT



(c) ScriptChecker.

Sandbox Context

MORESHI POWERPOINT

- 使用全新的JS 上下文Context（但是应该是在同一个Isolation内部）进行 Conext seperation 之后，有能力通过render的local security monitor抓捕 cross-context 的行为。

Task Capability System

MORESHI POWERPOINT

- ScriptChecker 提供一组权限字典，来供给 host script 动态的设置 cross-context 的 Object 访问权限。这种权限在 ScriptChecker 中叫做“能力”。
- 一旦对某个task提供了这种能力集合，那么该task对应的子task将会继承这种能力
- Task 能力的赋予依赖于security monitor

Asynchronous Execution

MORESHI POWERPOINT

- 为了可以在task级别上精准的限制第三方脚本，也就是解决Q1的问题，ScriptChecker定义了在一组api，调用该api可以将callee function的执行转移到一个新的task内。并且，Scriptchecker也修改了chrome，使其在script inclusion以及listener execution触发的时候使用这些产生新的task的api。

Example

MORESHI POWERPOINT

- 1. script inclusion
- 2. js function call
- both create new tasks with permission

```
1 <script src="https://.../utlity.js" risky
    task_capabitity="No_Access" />
2 <script>
3   function accessCookie() { return document.cookie;
        }
4   var secret = "user_info";
5   function shareData() { ...}
6   ...
7   // invoke untrusted code with restriction
8   var permissions = "JS_WL:shareData";
9   var data_to_hash = "data_to_hash";
10  var result = "";
11  asynCallCap(function() {
12    result = windowRisky[0].UtilityJS.Processing(
          data_to_hash);
13  }, permissions);
14
15  //followup computation
16  ...
17 </script>
```

Details

MORESHI POWERPOINT

- Context seperation
- Capability System
- Asynchronous Execution

Context seperation

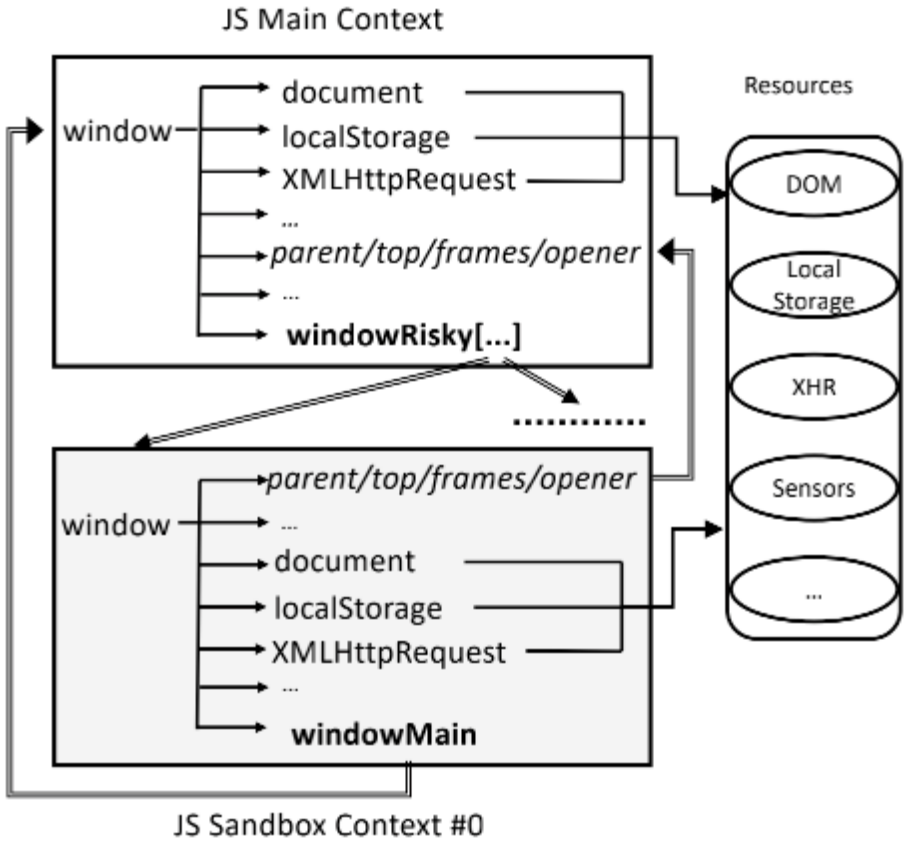
MORESHI POWERPOINT

- ScriptChecker 引入了 risky 标签，Blink在扫描到risky标签的时候，会为对应的script创建一个名为sandbox context的JS上下文。该上下文没有对JS的builtin做修改，旨在尽可能多的实现功能。
- 并且，sandbox context具有和main context相同的对DOM以及 system 资源的可见性（非可用性）
- 为了实现sandbox context和main context的同步，需要解决一些问题。

Frame struct

MORESHI POWERPOINT

- Sandbox context 拷贝了main context中与frame相关的数据结构，使其中的 risky script 可以如同 main conext 中那样通过 parent/frames 等对象访问到其他的frames。
- Sandbox context 整体维护的windows 对象是拷贝于Main context的windows对象



Event Listeners

MORESHI POWERPOINT

- risky script 可以将 event 注册进 DOM 中， 当event触发之后，context内的listeners将会执行内部的script。
- 因为重新构建了一个sandbox context，因此event触发需要在所有的context，包括sandbox context内执行listeners

Code Extension

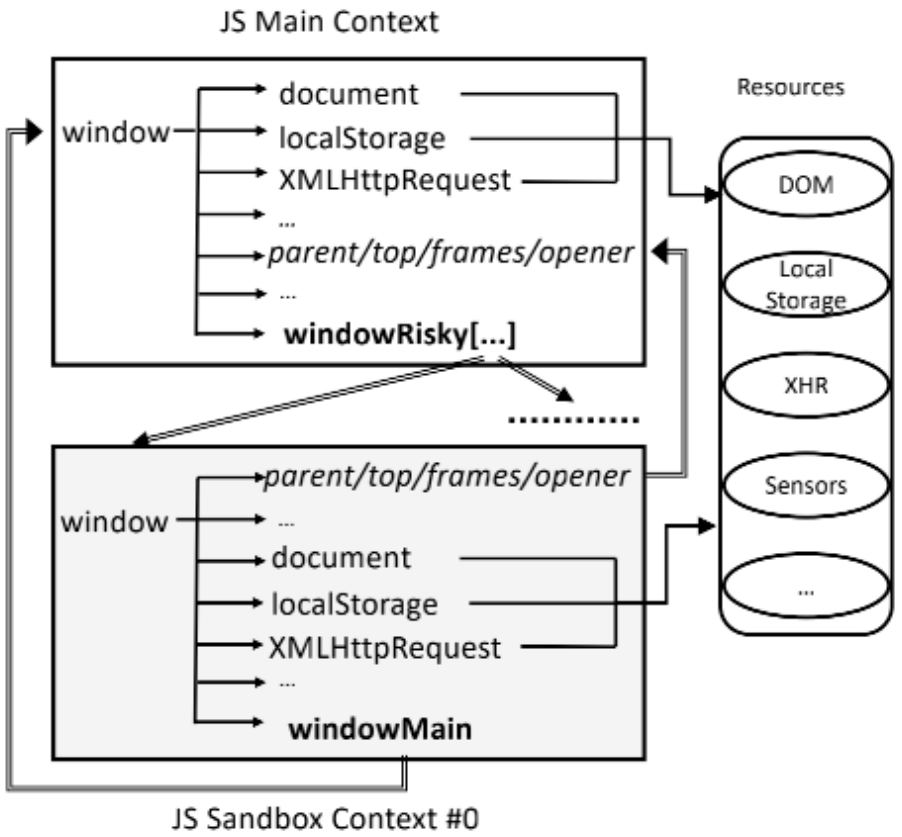
MORESHI POWERPOINT

- risky script 可能会调用code embedding来进行代码拓展。ScriptChecker为了保证拓展代码依旧可以被限制，ScriptChecker将会在所有的新创建的script标签处打上risky tag，并且，risky tag陪伴着标签的全部生命周期不允许修改。

Cross-Context Reference

MORESHI POWERPOINT

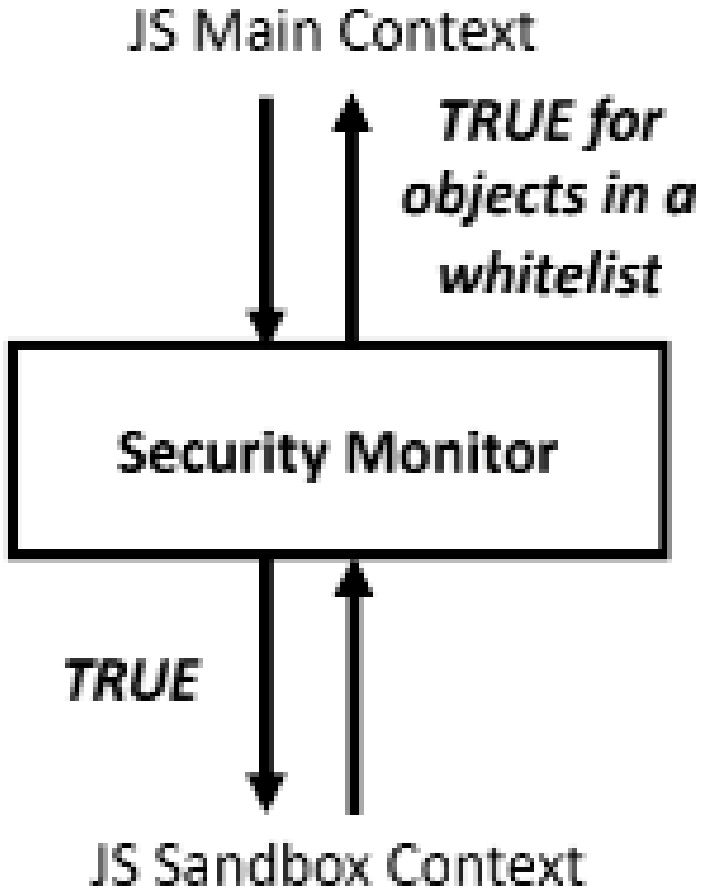
- 由于ScriptChecker将第三方的脚本执行分离到了Sandbox context，因此两个script之间的交互就变成了跨context的访问。为了支持这个特性，ScriptChecker对windows对象进行了修改



单向隔离

MORESHI POWERPOINT

- 分离的context天生具有预防context污染的能力，第三方脚本想要污染main context必须进行cross-context的访问。
- cross-context的访问是单向的，右图



Details

MORESHI POWERPOINT

- Context seperation
- Capability System
- Asynchronous Execution

CAPABILITY SYSTEM

MORESHI POWERPOINT

- Capabilities 只会赋予risky task，host script的运行不会受到限制。
- Risky tasks有三种情景：
  - 1. ScriptChecker 的 asynchronous API （newly created task）
  - 2. Javascript buildin asynchronous APIs （派生）
  - 3. Create by system tasks（系统创建 task ）
    - script inclusion：通过 tag 的值
    - listener execution：通过修改listerner注册实现，如果注册的task被标记为有风险的，那么Scriptchecker将会在注册的listener上标记该task。当该listener 被具体执行的时候，如果是标记的task的子task，那么将会被限制

CAPABILITY SYSTEM

MORESHI POWERPOINT

- 能力系统需要对local/kernel security monitor作出修改，基于security monitor本身的限制能力制定策略。
- kernel security monitor的限制能力依赖于ScriptChecker为IPC消息添加的字段，通过扫描该字段内的权限信息来判断当前IPC请求是否符合规范。
- local security monitor的限制能力来源于当前 task 附带的 capability 字段

Details

MORESHI POWERPOINT

- Context seperation
- Capability System
- Asynchronous Execution

ASYNCHRONOUS EXECUTION

MORESHI POWERPOINT

- function call JS 函数调用
- script inclusion 脚本引入
- listener execution 监听

Function Call

MORESHI POWERPOINT

- 在同一个task内执行js代码是同步的，但是将第三方脚本分离到了单独的task之后，会产生异步。
- 为了解决异步带来的逻辑问题，SC提供了两个JS api，分别为asynCallCap以及asynCallNoCap。
- asynCallCap的作用是为第三方脚本创建新的task1，并且将task移动到task queue的队首，确保立即执行。
- 而asynCallNoCap的作用是将剩余的host script也作为异步代码运行，emit一个task2，并且将该task2插入到task1的后面，以此来保证同步问题

Function Call

MORESHI POWERPOINT

```
1 function myfunc(params) {
2   // Call an untrusted function foo with parameters
3   var results = windowRisky.foo(params);
4   // Handle the return value for the untrusted
        function
5   rest_of_mycode(results);}
```

Listing 2. To synchronously call a third-party function `foo()`

```
1 function asyn_myfunc(params) {
2   var results;
3   ...
4   // create a child task to run untrusted function
        foo
5   tid= asynCallCap(function () {
6     results = windowRisky.foo(params);
7   }, "No_Cookie_Access");
8   // create another child task to run the rest
9   asynCallNoCap(function() {rest_of_mycode(results)
        ;}, tid); }
```

Listing 3. To asynchronously call `foo()` in ScriptChecker

Restricted Risky Script Inclusion

MORESHI POWERPOINT

- Script 加载是需要借助系统能力的工作，当渲染器扫描到script标签时，会通过IPC向broker进程发送数据下载任务，然后下载过的脚本将会自动执行。
- ScriptChecker 的方法是在parser发送Network请求的时候对request task 赋予 capability。

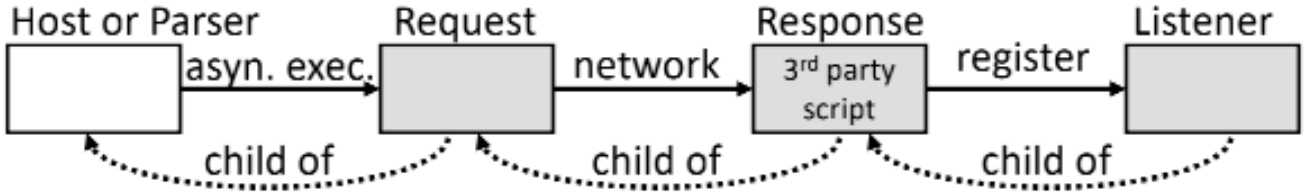


Fig. 5. Direct or dynamic inclusion of a script with `async` attribute. Grey boxes denote tasks with a capability.

Restricted Risky Listener Execution

MORESHI POWERPOINT

- ScriptChecker通过依赖task之间的父子关系来处理有风险的侦听器函数。执行侦听器的任务是其注册任务的子任务 并继承了它的功能。ScriptChecker 修改调用Listener的方式，以便在单独的任务中运行有风险的Listener。

Restricted Risky Listener Execution

MORESHI POWERPOINT

- Chrome Listener：当一个risky task向chrome event注册了一个listener时，ScriptChecker将会把task的标识附加在listener上。Chrome event触发了之后，render process将会对事件进行响应。响应办法是检查所有的Listener是否携带了task标志。如果携带，那么render将会创建新的task来运行listener，并且赋予task标识对应的权限。

Restricted Risky Listener Execution

MORESHI POWERPOINT

- jQuery Listener： Listener 同样也可以注册在DOM元素上。默认情况下，DOM元素对应的event触发时，所有的脚本都会在同一个task内运行。为了解决这个问题，ScriptChecker使用两套DOM，对应两套event以实现task的隔离。

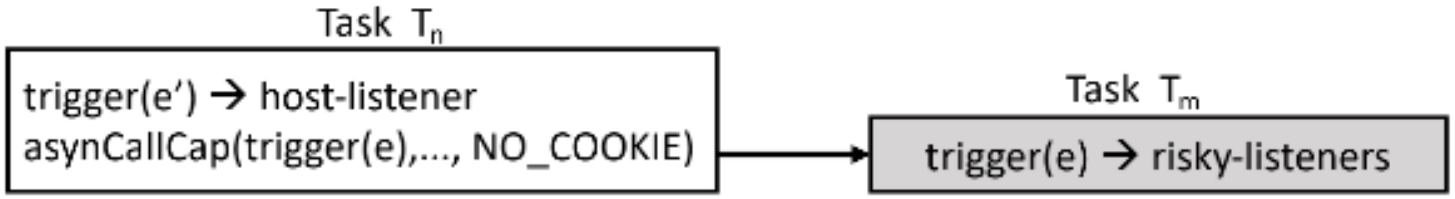


Fig. 6. Host listeners on e' and risky listeners on e

评估-能力

MORESHI POWERPOINT

- 文章选择了一个曾经被报告受到第三方库攻击的网站，National Baseball Hall web page。然后使用预定义的ScriptChecker规则。结果表明防御了所有攻击。

3rd-party script description	Attacks	Capability
gbh.js for analytics	billing form (direct access)	`DOM_Access_Protective, Network_Access, Cookie_Access`
modernizr.js for network feature checking	billing form (event tracking)	`DOM_Access(Deny, {Tag:form,textarea,input}), Network_Access`
fake-crypto.js for SHA256 computation	cookie (direct access, CDA)	`NO_Cookie_Access`

TABLE II. TO RESTRICT THREE SCRIPTS IN THE TEST PAGE

评估-适应性

MORESHI POWERPOINT

- 文章使用JS API来评估动态改变权限的可能。

```
1<script src="https://.../jquery.min.js" risky
     task_capabitity="DOM_Access_Protective" />
2<script>
3   asynCallCap(function () {
4     windowRisky.$.ajax({url:"config.json",async:
          false}).responseJSON;
5   }, "Network_Access", 0);
6   // protect sensitive DOM after the user login
7   asynCallCap(function () {
8     windowRisky.$("#status").html();
9   }, "DOM_Access_Protective", 0);
10</script>
```

Listing 6. Dynamic permission setting for jQuery

评估-防御 Benchmark

MORESHI POWERPOINT

- 文章创建了一个简单的网页，其中包含一些DOM对象（包括一个密码提交表单）和一个cookie。然后在Github中对1373个恶意脚本进行脚本检查，使用最多限制性政策，即拒绝所有访问DOM、Cookie和网络。结果显示，1021个脚本被阻止，332个脚本最终导致崩溃。其余的20个脚本没有被捕获。其中，12个由于环境错误而抑制了他们的攻击，4个使用弹出窗口，这超出了ScriptChecker的 permission scope。

评估-效率

MORESHI POWERPOINT

Schemes	Setup Overhead
AdSentry [29]	**31 ms** (9×10^7 cycles), script loading to shadow JS engine
TreeHouse [34]	**350 ms** (9×10^8 cycles), script loading to Web Workers
Jate [51]	**58 ms**, script rewriting growing drastically with dynamic code (No CPU frequency reported)
ConScript [42]	**2 µs** (7.2×10^4 cycles) for environment creation plus time for policy loading
ScriptChecker	**2.6 ms** (7.5×10^6 cycles) for sandbox context creation

TABLE IV. COMPARISON OF THE SETUP COST. THE NUMBER OF CPU CYCLES IS CALCULATED BASED ON THE RESPECTIVE CPU FREQUENCY.

评估-效率

MORESHI POWERPOINT

Graphic I	**chartjs**	**highcharts**	**particlesjs**	**raphael**	**d3**
Baseline	1218.37	415.87	237.69	248.69	625.04
Script-Checker	1223.17 (4.8)	427.57 (11.7)	248.19 (10.5)	258.95 (10.26)	670.68 (45.64)
Graphic II	**threejs**	**mathjax**	**amcharts**	**supersized**	**googlecharts**
Baseline	3211.22	3081.13	5922.22	188.65	2118.10
Script-Checker	3227.09 (15.87)	3130.03 (48.9)	5996.25 (74.03)	200.30 (11.65)	2154.44 (36.34)
Utility	**jquery**	**lodash**	**modernizr**	**moment**	**underscore**
Baseline	102.88	85.28	200.40	68.47	55.86
Script-Checker	111.64 (8.76)	93.75 (8.47)	206.55 (6.15)	75.36 (6.89)	63.62 (7.76)

TABLE VI. LATENCY (IN MS) ON LOADING WEB PAGES AND INVOKING THE FUNCTIONALITIES OF JS LIBRARIES REPORTED IN [17].

评估-与已有方案的对比

MORESHI POWERPOINT

Main Approach	Schemes	Runtime Overhead		Usability			Security	
		DOM Access	Page Loading	Developer's work	Policy Enforcer	Adaptable Policy	JS Attack Surface Coverage	Confused Deputy Attack & Collusion
Resource restriction	AdSentry [29]	590x	4.08% on Ads restriction	write call-back functions	Host JS code	No support	i) Manually done by developers; ii) error prone; iii) Need to update with JS;	Vulnerable
	TreeHouse [34]	N/A	15x on DOMTRIS [3]					
	Jate [51]	11.97%	19.5% on Ads restriction; 219% on JQuery loading;					
Function filtering	ConScript [42]	15%	N/A	write advice functions				
	WebJail [52]	N/A	**42%** on self-made pages					
Task capability	**ScriptChecker**	**9%**	**9%** on utility libraries [17] **8%** on JQuery **7.67%** on DOMTRIS [3]	replace function call interface	Browser	Easy	Orthogonal	Secure against

TABLE VIII. COMPARISON WITH EXISTING SCHEMES RESTRICTING RISKY SCRIPS.

讨论

MORESHI POWERPOINT

- 本文实现的基于Task的方案理论上可以与现有的js filter共存，来进一步加强安全性能。
- 本文在chromium基础上实现的原型较为复杂，需要对chromium内核有比较深的理解，并且移植到其他引擎较为困难。但这可以归结为工程难题而非学术。
- More

北京大学
PEKING UNIVERSITY

Thank You !

2022.10 付烨齐