Web 浏览器缓冲区溢出攻击检测技术①

Web Browser-Based Buffer Overflow Attack Detection

李晓桢　戈　戟　徐良华　(江南计算技术研究所 江苏 无锡 214083)

摘　要： 通过 Web 浏览器访问网页已经成为人们最重要的访问互联网的方式，但是浏览器中存在的缓冲区安全漏洞给用户的系统安全带来很大的威胁。基于浏览器的缓冲区溢出攻击有自身的特点，传统的入侵检测系统无法检测到它的攻击数据。分析了基于 Web 浏览器的缓冲区溢出攻击的特点，提出了对其进行检测的算法，并实现了一个原型系统。

关键词： Web 浏览器 缓冲区溢出 攻击代码检测 Javascript 脚本 信息安全

1　引言

通过 Web 浏览器访问网页已经成为人们最重要的访问互联网的方式，根据统计，全球一半以上的互联网流量属于 Web 访问流量。但是 Web 浏览器(如 Microsoft Internet Explorer)以及第三方 ActiveX 控件中存在着危害很大的缓冲区溢出安全漏洞[1,2]，给用户的系统安全带来极大的隐患。

利用 WEB 浏览器或 ActiveX 控件中存在的缓冲区溢出漏洞，攻击者构造一个恶意的网页(可能以 HTML、ASP、JSP、ASPX、PHP 等为扩展名，以下简称 HTML 文档)诱骗用户访问，当用户访问这个网页后就会触发缓冲区溢出漏洞，执行攻击者嵌入在网页中的恶意代码，造成计算机系统被攻击者控制。

针对缓冲区溢出攻击，已经出现了多种防护技术，如返回地址保护[3]、数据执行保护[4]、地址空间随机化[5]等，这些技术在一定程度上降低了缓冲区溢出攻击带来的威胁，但是由于部署难度和开销等原因，这些技术并没有得到广泛应用。

由于 Web 浏览器缓冲区溢出攻击的攻击代码包含在浏览器访问的网页中，因此可以对网页的内容进行分析，检测其中是否包含恶意代码，阻止恶意代码的执行，实现对这类攻击的防护。

本文研究分析了嵌入在 HTML 文档中攻击代码的特点，结合通用的攻击代码的检测技术，提出了对嵌入在 HTML 文档中的攻击代码的检测方法，在此基础上实现了一个原型系统。

2　相关研究

入侵检测系统 Snort[6]已经加入了对缓冲区溢出攻击数据的检测。由于缓冲区溢出攻击数据中常包含一个“填充字段”，由 NOP 指令(在 IA32 指令集中指令码为字节 0x90)组成，Snort 依据这个特征制定了一个规则：如果一段数据中包含的 0x90 字节个数超过一定的阈值，则判定这段数据中包含攻击代码。由于可以采用其他指令来代替 NOP 指令构造填充字段，因此 Snort 的这条规则很容易被绕过。

STRIDE[7]同样检测一段数据中是否包含填充字段，依据的特征是，从填充字段中任意的位置开始都可以反汇编出有意义的指令流到实际的攻击代码。如果反汇编经过的字节数超过一定的阈值，则认为这段数据中包含攻击代码。STRIDE 对填充字段的误报较少，可以检测出单字节、多字节甚至带跳转的填充字段，是一种较好的检测带填充字段的攻击代码的方法。但是对于不需要填充字段的攻击代码，STRIDE 无法检测。HTML 文档中的攻击代码虽然常使用填充字段来触发漏洞，但是这样的填充字段常用 javascript 构造，在文档的传输过程中并没有表现出填充字段的特征，只在 javascript 被解释执行时填充字段才在进程空间

① 基金项目:国家高技术研究发展计划(863)(2006AA01Z431)
收稿时间:2008-12-06

© 中国科学院软件研究所 http://www.c-s-a.org.cn

中出现，所以依据填充字段来对 HTML 文档中的攻击代码进行检测是无法进行的。

APE[8]的检测依据是正常的网络访问请求中不会包含长的有意义的指令流，而普通的功能字段都是有意义的指令串。因此对网络访问请求中的数据按控制流进行反汇编，如果反汇编得到的有意义的指令条数大于特定阈值，则认为该网络访问请求中包含攻击代码。这种检测算法存在多个问题，其中之一是 HTML 文档中的攻击代码经过了特殊的编码，在传输过程中并不是有意义的指令流，在 javascript 被解释执行时才被解码。同时，由于 IA32 指令集的密集性，这种检测算法在检测普通的攻击代码时的误报也比较严重。

Styx[9]采用反汇编的方法，构造出一段数据的控制流图，如果程序控制出现了可疑的系统调用、Call/Ret 指令对、循环写操作等就认为数据可疑。遇到条件跳转时，它反汇编所有可能的执行路径，构造控制流图，只要其中一条路径满足报警的条件就进行报警。Styx 是一种较好的采用静态反汇编的方法检测攻击代码的算法，具有较高的检测效率和较强的适用性，虽然无法直接对 HTML 文档的中攻击代码进行检测，但是具有很强的借鉴意义。

3　算法原理

3.1 浏览器攻击的特点

利用浏览器及浏览器插件的漏洞进行的缓冲区溢出攻击具有独特的特征，区别于普通的缓冲区溢出攻击，正是这些独特特征使得传统的检测攻击代码的算法失去作用。浏览器攻击表现出这些特征的根本原因是在 HTML 文档中可以使用 javascript 脚本，攻击者通过使用 javascript 脚本，可以在一定程度上操作浏览器运行时的进程空间，从而可以在浏览器进程内部构造攻击代码，而在传输过程中，攻击文档只表现为普通的文本。

3.1.1 漏洞触发的特点

以典型的的栈溢出为例，攻击者向目标主机发送形如图 1 所示的攻击数据，溢出发生后，攻击者伪造的返回地址字段覆盖了目标函数的返回地址，而伪造的返回地址指向图中的填充字段或功能字段，函数返回时，程序将跳转到返回地址指向的位置执行。

图 1　攻击数据组成

浏览器的缓冲区溢出攻击则不同，攻击者并不需要向目标主机发送类似于图 1 中的数据，而是在网页显示时用 javascript 脚本生成填充数据、返回地址，并将这些攻击数据传递给有漏洞的函数。在传输过程中，javascript 脚本只表现为普通的文本，如图 2 所示。

这段 javascipt 脚本为一个 ActiveX 控件对象的 Mask 属性赋了 512 字节的值，造成了缓冲区溢出，填充字段由 0x0c 组成，返回地址被覆盖成 0x0c0c0c0c。脚本的其他部分会事先将 0x0c0c0c0c 位置覆盖为攻击代码，从而完成一次缓冲区溢出攻击。但是这个 HTML 文档却没有暴露出填充字段和返回地址字段，使得传统的依靠检查填充字段来完成攻击检测的算法失去作用。

```
var body='<PARAM NAME="Mask"
VALUE="";
var buf='';
for (i=1;i<=512;i++)
{
    buf=buf+unescape("%0c");
}
var body1="">
document.write(body+buf+body1);
```

图 2　生成攻击数据的脚本

3.1.2 攻击代码特点

传统缓冲区溢出攻击的攻击代码，即图 1 中的功能字段，是一段二进制可执行代码，将被插入目标进程空间执行，在传输过程中表现出可执行代码的特征，因而容易被传统的安全系统(如入侵检测系统)拦截。

但是 HTML 文档中的攻击代码不同。虽然攻击代码也包含在文档中，但是这些攻击代码并不是以二进制可执行代码的形式存在，而是表现为普通的文本，如图 3 所示。

```
var shellcode = unescape((
"%u51eb%u758b%u8b3c%u3574%u0378%
u56f5%u768b%u0320" +
"%u33f5%u49c9%uad41%udb33%u0f36%u
14be%u3a28%u74d6" +
"%uc108%u0dcb%uda03%ueb40%u3bef%u
75df%u5ee7%u5e8b" +
```

图 3　HTML 文档中的攻击代码

© 中国科学院软件研究所 http://www.c-s-a.org.cn

以图中“%u51eb”为例，这个字符串在 HTML 文档中是以 ASCII 码存在的，用十六进制表示为“0x25 0x75 0x35 0x31 0x65 0x62”，没有表现出可执行代码的特征。但是在 unescape 函数解释执行后，“%u51eb”就被解码成“0xeb 0x51”，这是一条相对寻址的 JMP 指令。通过脚本的解释执行，攻击代码被还原到浏览器的进程空间中，如果程序的执行流程跳转到这些位置，攻击代码就会被执行。

可见，HTML 文档中的攻击代码在脚本解释之前没有表现出传统攻击代码的特征，如果要实现对它们的检测，需要先对 HTML 文档进行预处理，将攻击代码还原成普通的二进制可执行代码，这是我们算法的出发点。

3.2 算法描述

算法首先对截获的 HTML 文档进行预处理，将可能包含攻击代码的内容解码还原成传统的二进制形式；然后调用攻击代码检测模块，采用传统的检测攻击代码方法对还原后的内容进行检测，如果检测到了攻击代码，则判定发生了浏览器缓冲区溢出攻击。

3.2.1 HTML 文档预处理

HTML 文档预处理的目的是提取文档中可疑的攻击代码，将其解码还原成普通形式的编码，为进一步检测做准备。

Javascript 的 unescape 函数是一个解码函数，将一个编码后的字符串解码成 Unicode 类型的串。如果源串是用“%uxxyy”格式表示的，则解码后将变成十六进制的“yyxx”。预处理算法将模拟这个功能，将形如图 3 所示的代码解码。

预处理算法搜索 HTML 文档，找到包含在“<script>”标签内的长度超过一定阈值的形如“%uxxyy”的串，除去其中的无效字符(如换行符)，将“%uxxyy”转换为“yyxx”，保存在缓冲区内。算法实现比较简单，在此不再详述。

3.2.2 攻击代码检测

一旦可疑的攻击代码被还原成传统形式，就可以利用传统的检测攻击代码的方法来完成检测，如 Styx、APE 等，Qinghua Zhang 等还提出了一种利用虚拟执行来检测攻击代码的算法[10]。出于效率的考虑，我们采用了类似于 Styx 的基于反汇编的检测算法，并对其进行了改进。

攻击代码在运行之前对进程空间完全陌生，为了进行数据读写操作，攻击代码需要得到自身在进程空间所处的环境，称为“自定位”。由于实现自定位的方法非常少，所以可以把自定位操作的控制流特征作为检测攻击代码的依据。

攻击代码作为可执行的程序段，会表现出一定的程序控制特征，如“Call/Ret”指令对，系统调用等，可以将这些程序控制特征作为检测依据。

功能字段经过编码的攻击代码(构成如图 4 所示)，其中的解码字段具有明显的循环写特征，在循环体内部有一个利用寄存器间接寻址的写操作，并且该寄存器在每次循环时发生改变，可以将这个特征作为检测依据。

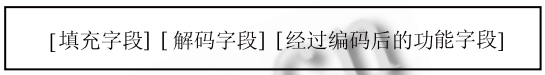

图 4　编码后的攻击代码

我们判定一段数据如果满足以下三个条件之一，则该段数据可能包含有攻击代码：①包含自定位操作控制流；②包含明显的程序控制特征；③包含循环写操作。

算法对预处理生成的数据段进行反汇编，遇到直接跳转则跳转到目标地址继续，遇到条件跳转则对两个分支分别进行反汇编，如果出现以上三个控制流特征则判定这段数据中包含攻击代码。

4　实现考虑

我们的算法可以作为一个模块集成在入侵检测系统中，对经过系统的采用 HTTP 协议传输的数据报文进行检测；也可以作为浏览器的插件，在浏览器解释 HTML 文档之前对文档内容进行检测，如果文档内容可疑，则阻止浏览器对可疑代码的解释执行。原型系统实现了第二种方案，由于 Internet Explorer(以下简称 IE)拥有最多的用户，所以原型系统设想的运行环境是在该类型的浏览器上，但是只要对程序稍加修改就可以适用于其他浏览器。

IE 浏览器在解释并显示 HTML 文档之前，会先将其下载到本地的“系统盘符:\Documents and settings\当前用户\Local Settings\Temporary Internet Files”目录下，然后调用 CreateFile 函数将文档装入存储器并解释显示。因此，原型系统设计为一个 IE 插件，在 IE 进程启动时进入进程空间，并挂钩 Kernel32.dll 中的 CreateFileW 函数，如果进程调用该函数时文件路径参数中的文件扩展名为可能的网页扩展名(HTML、HTM、ASP、ASPX、PHP、JSP 等)，则将其拦截。

当浏览器打开一个网页时，原型系统会在文档从本地被装入存储器时将其拦截，检测文档内容中是否

© 中国科学院软件研究所 http://www.c-s-a.org.cn

包含攻击代码。如果文档的某一部分满足可疑条件，则报警并将这部分替换掉，从而消除文档的潜在威胁。

5 结果分析

原型系统的运行环境为：CPU 主频 3.2GHZ，存储器大小 1G，Windows XP sp2 操作系统，IE6.0 Web 浏览器。在这样的环境下，对原型系统的性能和开销进行了测试分析，结果如下。

编写脚本使用 IE 浏览器打开 2000 个正常的网页文档，系统误报了 3 个，误报率取得了较好的结果，部分原因是在对网页进行预处理时，只提取了形如“%uxxyy”的部分，对网页的其他部分不进行进一步处理，这会大大降低系统的误报率。

在对系统打好补丁的前提下，使用 IE 浏览器打开 1000 个确定包含有攻击代码的网页，原型系统漏报了 12 个。没有达到 100%的检测效果的原因是部分攻击代码并没有表现出 3.3 节中定义的特征，但是由于原型系统部署特别简单，并且开销不大，系统仍有一定的实用效果。

在开销方面，在安装了原型系统的主机上进行正常的网页浏览，没有明显感觉到打开网页速度的变化，系统带来的开销很小。

传统的入侵检测系统(如 Snort)只能检测出极少的包含有明显填充字段恶意网页，这是因为它只采用了非常简单的特征。

主机安全软件(如卡巴斯基互联网安全套装等)可以阻止部分恶意网页的下载，但是只能阻止包含有以知的明显病毒特征的网页，对利用方法稍加修改就可以规避。而原型系统不依赖病毒特征库，只检测通用的攻击代码，具有更好的检测效果。

6 结束语

基于 Web 浏览器的缓冲区溢出攻击给用户的系统安全带来很大威胁，本文分析了浏览器攻击的特点和传统的恶意代码检测技术不能用于这类攻击的原因，提出了新的针对浏览器攻击的检测算法，并实现了一个易于部署的原型系统，以较小的开销实现了对多数攻击的检测。原形系统仍然存在不足，对经过特殊构造的攻击代码的漏报较多，需要在今后进一步完善。

参考文献

1 MSIE <IFRAME> and <FRAME> tag NAME property buffer overflow POC exploit. http://seclists.org/fulldisclosure/2004/Nov/0053.html..

2 Vulnerability Summary for CVE-2008-3704. http://web.nvd.nist.gov /view/vuln/detail? execution=e2s1.

3 Cowan C, Pu C, Maier D, Walpole J, Bakke P, Beattie S, Grier A, Wagle P, Zhang Q, Hinton H. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. Proc. of the 7th USENIX Security Symposium, San Antonio, Texas, January 1998:63－78.

4 PaX Team. PaX. http://pax.grsecurity.net, 2001.

5 Bhatkar S,DuVarney DC,Sekar R. Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits. Proc. of the 12th USENIX Security Symposium, Washington D.C., August 2003:105－120.

6 Snort rule 651,648. www.snort.org /pub-bin/ sigs.cgi?sid=651.

7 Akritidis P, Markatos E, Polychronakis M, Anagnostakis K. STRIDE: Polymorphic Sled Detection through Instruction Sequence Analysis. Proc. of the 20th IFIP International Information Security Conference (SEC’05), June 2005:375－392.

8 Toth T, Kruegel C. Accurate Buffer Overflow Detection via Abstract Payload Execution. Proc. of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID’02), October 2002:274－291.

9 Chinchani R, Berg E. A Fast Static Analysis Approach To Detect Exploit Code Inside Network Flows. Proc. of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID’05), September 2005:284－308.

10 Zhang QH, Reeves DS, Ning P, Iyer SP. Analyzing Network Traffic To Detect Self-Decrypting Exploit Code. Proc. of the 2rd ACM Symposium on Information, Computer and Communications Security (ASIACCS), Singapore, March 2007:4－12.

© 中国科学院软件研究所 http://www.c-s-a.org.cn