

典型 Shellcode 引擎特征检测方法研究

张登银, 洪福鑫

(南京邮电大学 计算机学院, 江苏 南京 210003)

摘 要: 通用的 shellcode 引擎大都采用特定运算对 shellcode 进行编码, 使得 shellcode 具有规避传统的代码特征检测系统的能力。为了检测具有规避传统检测能力的 shellcode, 深入分析目前典型 shellcode 引擎的工作原理, 在此基础上研究引擎产生 shellcode 的代码特征和行为特征, 进而提出了基于这两类特征的针对性综合检测方法。实验结果表明, 这种综合检测方法可以针对性地、有效地检测并阻止这类 shellcode 的执行, 同时对其它 shellcode 也能实现一定程度上的检测, 而且虚警和漏警率为 0。该检测系统对恶意代码的检测具有一定的应用价值。

关键词: Shellcode 引擎; 指令; 特征检测

中图分类号: TP393.08

文献标识码: A

文章编号: 1673-629X(2010)01-0018-04

Research on Character Detection for Typical Shellcode Engine

ZHANG Deng-yin, HONG Fu-xin

(College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: Shellcode engine usually adopts certain operations which enables shellcode to evade the detection based on data types. In order to detect these shellcode, in this paper, the principle of typical shellcode engine at present is in-depth analyzed. Then the data types and characters of shellcode action are researched. Furthermore, the detecting method aimed at these two kinds of shellcode characters is proposed. The experiment result shows that this approaches is effective and pointed to detect shellcode and prevent it from being executed, at the meantime, it can also detect other shellcode to a certain extent. In addition, its alarm probability and false alarm rate is zero. So, it play an active role in shellcode detection.

Key words: shellcode engine; opcode; character detection

0 引言

利用计算机漏洞实施攻击, 一般需要几个必备的条件: 一是具有可攻击漏洞的主机; 二是在攻击漏洞主机中实施恶意行为的 shellcode; 三是利用漏洞执行 shellcode 的 exploit。狭义上, shellcode 只是获得 shell 的代码; 广义上, shellcode 是一组可注入进程的指令(opcode), 可以在被攻击的程序内运行。由于 shellcode 要直接操作寄存器和函数, 并且一些细微的差别都可能导致 shellcode 罢工, 因此 shellcode 一般不用高级语言编写, 而且不同的漏洞对 shellcode 的内容和长度通常还有很多苛刻的限制, 其中最普遍的限制就是 shellcode 中的坏字符。如果让人工去考虑这些限制并编写适用的 shellcode 是非常困难的, 因此能自动生成适用

shellcode 的 shellcode 引擎也就应运而生。显然, 研究 shellcode 引擎原理以及由 shellcode 引擎生成的 shellcode 特征, 将有助于 shellcode 的检测。传统的 shellcode 检测方法多采用基于代码特征的检测方法, 由于典型 shellcode 引擎的产生, 使得其失去了检测的效果。

文中在深入分析典型 shellcode 引擎原理的基础上, 提出了针对 shellcode 代码特征和行为特征的检测方法。实验结果表明, 该检测方法对这类引擎生成的 shellcode 的检测效果十分明显, 同时对其它类型 shellcode 的检测也有较好效果。

1 Shellcode 引擎原理

在 shellcode 自动生成过程中, 需要集中解决坏字符的问题, 如: 在生成的可执行 shellcode 中不能出现全零字节数据等。目前解决该问题的主要方法是采用针对 shellcode 代码本身的编码/解码技术。而编码/解码技术目前又有许多的原理可以采用, 通常使用的有: xor 原理、character 或 word-based 之类的原理^[1]。文中的 shellcode 检测技术主要是针对目前应用最多也最

收稿日期: 2009-04-08; 修回日期: 2009-07-15

基金项目: 国家 863 计划项目(2007AA701302, 2008AA701202)

作者简介: 张登银(1964-), 男, 江苏靖江人, 研究员, 博士, CCF 会员, 研究方向为信号与信息处理、IP 网络技术、服务质量与信息安全。

典型的 xor 编码/解码原理所产生的 shellcode。

1.1 引擎结构

基于 xor 编码/解码原理的 shellcode 引擎,是对程序机器码进行直接操作。原理上,对于一个字节的数据,与不同于它的另一个字节数据异或运算,可以得到一个非零的新数据;对新数据再次异或运算后,结果将还原到数据本身。所以对于一段机器码 A,可以选择一个适当的字节数据 a,通过对 A 中每个字节数据进行异或运算,就能得到正常实用的 shellcode(不含全零字节)。当需要执行该 shellcode 时,可以将这段机器码再次异或运算即得到原代码。shellcode 引擎大都基于这一原理^[2],典型的 shellcode 引擎模块结构如图 1 所示。

头文件、宏定义模块	
固定“解码”头模块	
主模块	模块 1: 寻找编码/解码关键字,确定模块 4 的 Opcode 长度
	模块 2: 编码模块 4 的 Opcode
	模块 3: 添加解码头模块组合成可用的 shellcode
	模块 4: 需要 shellcode 实现的功能模块(使用内联汇编编写)

图 1 shellcode 引擎模块布局

1.2 编码模块

首先,寻找编码/解码关键字模块是通过从 0X00 至 0Xff 这 256 个数据,循环选取其中的一个字节数据(编码/解码关键字),然后用这个数据与模块 4 的 Opcode 中的每个字节数据进行异或运算,如果运算的结果出现以下任何一种情况:“&”、“=”、“?”、“@”、“\”以及回车、换行符和全零字节数据(NULL),则淘汰该数据,直到 256 个数据都跟 Opcode 中的每个数据异或,如果都被淘汰,则说明这个 Opcode 无法通过 xor 运算来编码/解码生成适用的 shellcode,如果其中有一个字节数据能满足不被淘汰,则不再寻找其它数据,直接将这个数据作为恶意代码的编码/解码关键字。找到该关键字后,对整个模块 4 的 Opcode 进行编码(对每个字节数据进行异或运算)。生成的 shellcode,则是由解码模块加上编码后的 Opcode 组成。

1.3 解码模块

当机器跳转到 shellcode 开始执行时,首先执行解码模块。解码模块的主要功能,是将编码后的 Opcode 解码成可正常执行的机器码,并将机器指引到这段机器码去执行。

一个适用的解码模块实例如下^[3]:

```
unsigned char decode[] =
"\xEB\x10\x5B\x4B\x33\xC9\x66\xB9"
"\x66\x01" //shellcode size
```

```
"\x80\x34\x0B"
```

```
"\x99" //xor byte
```

```
"\xE2\xFA\xEB\x05\xE8\xEB\xFF\xFF\xFF";
```

这段 shellcode 对应的汇编代码如下:

```
00400030 |EB 10 JMP SHORT 00400042
00400032 |5B POP EBX
00400033 |4B DEC EBX
00400034 |33C9 XOR ECX,ECX
00400036 |66:B9 6601 MOV CX,166
0040003A |80340B 99 XOR BYTE PTR DS:[EBX+ECX],99
0040003E |E2 FA LOOPD SHORT 0040003A
00400040 |EB 05 JMP SHORT 00400047
00400042 |E8 EBFFFFFF CALL 00400032
```

上述通用实例代码一开始使用了自动定位技术,因为紧跟在这段解码代码后面的是具有破坏行为的恶意代码,通过一个 jmp 和 call 可以动态获得恶意代码的起始地址。这段代码中 166 为恶意代码的长度,99 为搜索到的编码/解码关键字。当然,对于不同的恶意代码会有不同的恶意代码长度和编码/解码关键字。这两个数据是由 shellcode 引擎写入的。最后涉及一个关键的技巧:凡是 call 指令调用的是一个标号时,则 call 指令的地址跟 call 指令的下一条指令地址正好相差 5 个字节,这也是为什么在 00400040 这个地址的 jmp 指令必须跳到 00400047 的原因。很明显这个有自动解码功能的头部模块,就是一段可执行的 shellcode,它具有如下功能:

(1)自动重定位——动态获得恶意代码的起始地址。

(2)解码——二次异或运算。实现对编码后的恶意代码进行解码,使得恶意代码恢复源代码形式。

(3)将执行权交给恶意代码,通过 jmp-call 准确地跳转到恶意代码的起始位置执行恶意代码^[4]。

2 Shellcode 特征检测

2.1 代码特征检测

通过自动引擎生成的 shellcode 在被植入到进程或者被写入到文件、内存中后,其代码本身具有一定的特征,根据这些特征就可以在文件或内存中将具有恶意行为的 shellcode 检测出来。显然,shellcode 代码特征检测的关键,就是要识别出 shellcode 的代码特征。shellcode 引擎通过 xor 编码/解码原理自动生成的 shellcode,可以通过图 2 所示的检测系统来进行检测。

(1)全零字节和特殊字符数据检测。

在 shellcode 未执行前,shellcode 代码本身不会包含任何全零字节和一些特殊字符。因而通过搜索一定长度的二进制文件或者内存,查看是否出现全零字节

数据可以作为 shellcode 的判定条件。

另外,通过该引擎生成的 shellcode 不会包含以下特殊字符:“&”、“=”、“?”、“@”、“\”以及回车、换行符。因而这些特殊字符数据也可作为检测的条件。

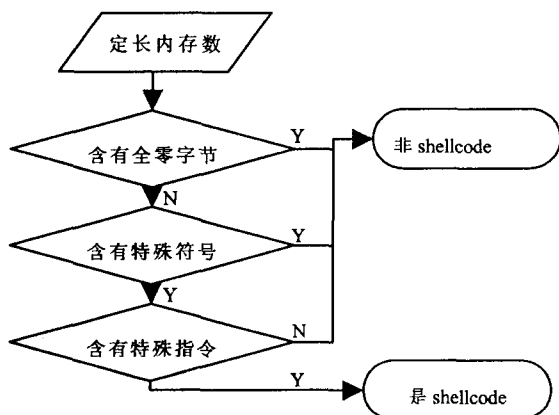


图2 代码特征检测

(2)特殊指令数据检测。

shellcode 的解码模块具有特定代码特征,即特殊指令。解码模块最核心的功能是解码,而解码需要动态定位 shellcode 中编码过的代码起始地址,文中引擎所使用的定位技术是 jmp 与 call 指令,在 shellcode 解码模块使用 jmp 跳转到 call 指令的地址(call 指令后紧跟编码过后的恶意代码),执行 call 指令后系统栈会自动保存返回地址(编码过后恶意代码的起始地址)。这里的 jmp 和 call 指令都是段内调用,而针对 jmp 和 call 指令的段内调用,这两条指令的 Opcode 是固定的,如: jmp 指令对应 Opcode 数据为 EB,call 指令对应的 Opcode 数据是 E8^[5]。这两条指令在 X86 系统中对应的 Opcode 如图 3 所示。

地址	HEX 数据	反汇编
00427A80	EB 10	JMP SHORT shellcod 00427A92
00427A82	5B	POP EBX
00427A83	4B	DEC EBX
00427A84	33C9	XOR ECX,ECX
00427A86	66:B9 2601	MOV CX,126
00427A8A	80340B F3	XOR BYTE PTR DS:[EBX+ECX],0F3
00427A8E	E2 FA	LOOPD SHORT shellcod 00427A8A
00427A90	EB 05	JMP SHORT shellcod 00427A97
00427A92	E8 EBFFFFFF	CALL shellcod 00427A82
00427A97	55	PUSH EBP

图3 jmp 和 call 指令对应 Opcode

通过在可疑代码头部搜索这两个 Opcode,可以帮助判定典型引擎生成的 shellcode。

除此之外,shellcode 的解码过程必定是一个循环操作,循环操作多采用 loopd 指令,对于这个指令,shellcode 必须在指令执行之前做好对默认循环计数器 ecx 的初始化。为了避免在其 Opcode 中出现全零字节,会使用比较简洁的 xor 指令来初始化 ecx 寄存器,这就使得 shellcode 头部的解码代码中必定包含

“xor ecx,ecx”这一指令。因此,通过搜索这条指令对应的 Opcode,即 33c9,可以判定是否典型引擎生成的 shellcode。

上述这些特殊指令,可以构成一个代码特征集。图 2 所示的代码特征检测就是基于该特征集,可以准确判定典型引擎生成的这类 shellcode。

2.2 行为特征检测

传统的 shellcode 行为特征检测是从 shellcode 的破坏行为入手,找到一般 shellcode 实施破坏行为的共同点,从而提取出 shellcode 的行为特征^[6],检测系统如图 4 所示。

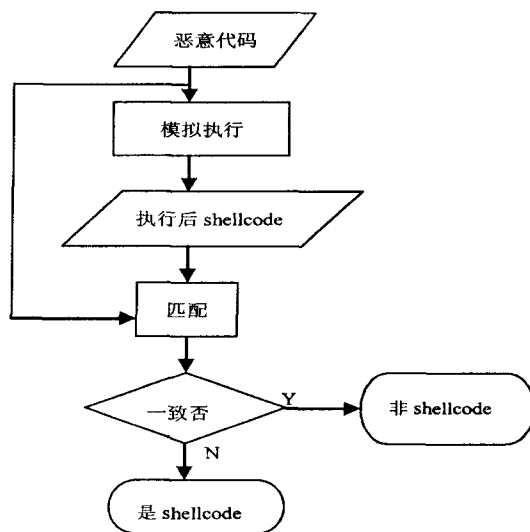


图4 行为特征检测系统

传统的 shellcode 行为特征包括:

(1)常调用的 API 函数如:LoadLibraryA,GetProcAddress, CreateFileA, WriteFileA, GetModuleHandleA,CreateProcessA,WinExec 等^[7]。

(2)分析和利用 PEB、SEH、TEB、导出函数表、IAT 等结构来获取模块和 API 地址^[8]。

(3)搜索导出表等^[9]。文中提出的 shellcode 行为检测法是针对典型引擎产生的 shellcode 所特有的循环解码行为的。解码引擎通过一个解密循环对负载进行解码。在此循环中,将对负载进行反复的读取和写入操作。解码操作结束后,程序指针跳入已解码的 shellcode 起始位置。依据 shellcode 的解码行为,可以通过模拟执行可疑代码,并将执行前代码与执行后代码本身进行匹配,就可以完全发现 shellcode 的解码行为特征,从而检测出这类 shellcode。

3 综合检测系统实现及实验结果

上述两类检测方法,是分别针对典型 shellcode 恶意代码所普遍具有的代码特征和行为特征。实际应用

中,可以将这两类检测方法综合起来,如图 5 所示。

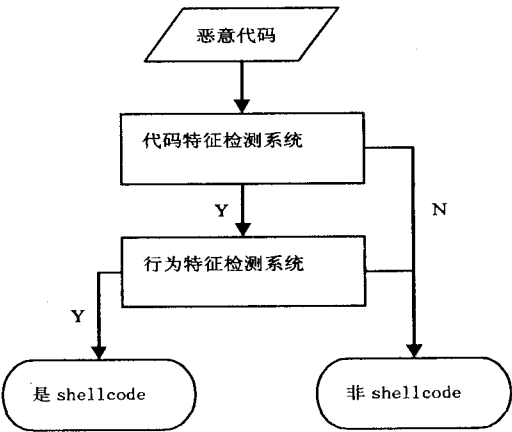


图 5 综合检测系统

3.1 系统实现

已经实现了基于上述体系结构的原型系统,检测系统原始的恶意代码数据通过文中讨论的经典引擎生成或者网络搜集获得,代码特征检测系统和行为特征检测的匹配过程,均通过数据匹配程序来完成。

该匹配程序采用 C 语言通过 vc 开发工具实现。行为特征检测系统的程序模拟执行以 CPU 模拟器为基础,它用高级语言实现,能够模拟 CPU 对二进制代码进行执行,根据指令动态改变模拟的 CPU 及内存状态。

3.2 实验结果

实验原始数据分为三组:

- (1)典型 shellcode,为文中研究针对的典型 shellcode 引擎所产生;
- (2)一般 shellcode,为搜集获得的其它 shellcode;
- (3)正常程序,为能正常运行的、不含恶意行为的计算机程序。

每组样本各 30 个,检测试验结果如表 1 所示。

表 1 检测试验结果

特征检测		典型 shellcode	一般 shellcode	正常程序
代码特征	含全零字节	0	0	28
	含特殊符号	0	3	27
	含特殊指令	30	24	25
行为特征	执行数	30	23	25
	匹配数	0	0	25
检测率		100%	76.6%	0

实验结果显示,该检测系统对典型 shellcode 具有 100% 的检测效果,同时对其它 shellcode 也能实现一定程度上的检测,而且虚警和漏警率为 0。

与传统的 shellcode 检测系统相比,文中所述检测系统具有以下特点:

- (1)传统检测系统能检测更多种类的 shellcode,但是在典型 shellcode 的检测上却达不到文中所述系统的检测精度。
- (2)本检测系统的检测实现方法简单,从代码特征检测到行为特征检测,最终都是数据匹配,这比传统的检测法需要监控 API 函数、跟踪程序执行等要占用更少的系统资源。

4 结束语

传统的 shellcode 检测技术针对的是所有 shellcode,由于关注的广泛性,在一些特定 shellcode 的检测上难免会顾此失彼,失去检测的精确性和有效性。文中在详细分析基于编码/解码原理的典型 shellcode 引擎基础之上,提出了一个综合代码特征和行为特征的 shellcode 检测系统。

实验结果表明,该检测系统可以精确检测出典型引擎产生的 shellcode,同时,这两类方法对其他 shellcode 的检测也具有明显效果,具有一定的应用参考价值。

参考文献:

[1] SANS Institute. The Twenty Most Critical Internet Security Vulnerabilities[EB/OL]. 2006-05-31. <http://www.sans.org/top20/>.

[2] ARCE I. The shellcode generation[J]. IEEE Security & Privacy, 2004, 2(5): 72-76.

[3] San. 渗透防火墙的 shellcode 技术[EB/OL]. 2004. http://www.forccus.net/projects/Xcon/2004/Xcon2004_san.pdf.

[4] Sk. History and advances in Windows shellcode[EB/OL]. 2004. <http://www.phrack.org/show.php?p=62&a=7>.

[5] R IX. Writing IA32 alphanumeric shellcodes[J]. Phrack Magazine, 2004, 12(57): 15-18.

[6] SKAPE. Understanding Windows shellcode[EB/OL]. 2003-10-23. <http://www.hick.org/code/skape/papers/win32-shellcod.pdf>.

[7] Destr Istan T, Ulensp Iegel T. Polymorphic shellcode engine using spectrum analysis[J]. Phrack Magazine, 2003, 11(61): 9-15.

[8] AKA TW. Writing UTF-8 compatible shellcodes[J]. Phrack Magazine, 2003, 11(61): 1-3.

[9] 何 乔, 吴廖丹, 张天刚. 基于 shellcode 检测的缓冲区溢出攻击防御技术研究[J]. 计算机应用, 2007, 27(5): 2-4.