doi:10.3969/j.issn.1000-436x.2013.03.018

蠕虫正则表达式特征自动提取技术研究

唐勇[1]，诸葛建伟[2]，陈曙晖[1]，卢锡城[1]
(1. 国防科技大学 计算机学院，湖南 长沙 410073；2. 清华大学 信息网络工程研究中心，北京 100084)

摘 要：提出一种实用的蠕虫正则表达式特征自动提取方法，该方法由蠕虫传播网络流样本获取、特征树生成、高假阳性特征剔除、特征融合这 4 步组成。该方法的优点是可输出具有强描述能力的包含“.*”、“.{k}”、“|”、“(c){k}”等元字符的正则表达式特征。基于蜜罐系统(Honeybow)实现了该方法，并针对互联网上数种真实蠕虫进行了实验。实验结果表明，该方法可以准确地提取真实蠕虫的正则表达式特征，可以在蜜罐、蠕虫及恶意代码分析等系统中应用。

关键词：蠕虫；恶意代码；特征提取；特征树；正则表达式；蜜罐；入侵检测
中图分类号：TP311.134.3 **文献标识码：**B **文章编号：**1000-436X(2013)03-0141-07

Automatic generating regular expression signatures for real network worms

TANG Yong[1], ZHUGE Jian-wei[2], CHEN Shu-hui[1], LU Xi-cheng[1]
(1. College of Computer, National University of Defense Technology, Changsha 410073,China;
2. Network Research Center of Tsinghua University, Beijing 100084, China)

Abstract: A practical worm regular expression of automatic extraction method was presented, which involves four steps: sample collection from worm propagation, signature tre generation, removal of high false-positive signatures, and signature merging. The advantage of this method is that it can directly output expression signature with the rich syntax of ".*", ".{*k*}", "|", "(*c*){*k*}". Based on the honeypot system Honeybow deployed in Internet, the method was implemented and evaluated by several in-wild worms under real Internet environment. The experiment results show that the method can generate accurate regular expression signatures for real worms and is promising for the applications of automatic analysis of worms and malwares.

Key words: worm; malware; signature generation; signature tree; regular expression; honeybow; intrusion detection

1 引言

准确有效的特征是检测和防范网络蠕虫传播的关键，当前，蠕虫传播特征主要是依靠安全专家进行人工提取，过程复杂、速度慢，且由于变形技术[1, 2]在蠕虫中的普遍应用，准确性也无法得到保证。近年来，一些蠕虫传播特征自动提取方法被相继提出，这些方法通过分析蠕虫传播数据（可能包括噪声）自动生成蠕虫传播特征，它们可被划分为下列类别[3]：基于 LCS（longest common substring)[4~6]、基于固定长度负载出现频率[7, 8]、基于可变长度负载出现频率[9, 10]、基于有限状态自动机[11]、基于序列比对[12~14]等。基于序列比对的方法[12~14]在准确性上要优于其他方法，但是也只能输出仅包含“.*”和“.{*k*}”这两类元字符的简单正则表达式特征，例如，对 CodeRed 蠕虫产生的“‘GET /’.*‘.ida?’ *‘XX’ *‘%u’.{4}‘%u780’*‘=’.{7}‘HTTP /1.0\r\n’”特征。如何自动提取具有更丰富语义（即包含更多类型元字符）的完整正则表达式特征是一个挑战性问题。

本文提出一种蠕虫传播复杂正则表达式特征

收稿日期：2011-08-24；**修回日期：**2012-03-25
基金项目：国家高技术研究发展计划（“863”计划）基金资助项目（2011AA01A103）；国家自然科学基金资助项目（61003303，61003217)
Foundation Items: The National High Technology Research and Development Program of China (863 Program) (2011AA01A103); The National Natural Science Foundation of China (61003303, 61003217)

自动提取方法。相比现有方法，该方法输出的正则表达式特征包含“.*”、“.{k}”、“|”、“(c){k}”等元字符，分别表示“匹配任意长度字符串”、“k 个任意字符”、“或关系”、“k 个字符 c”，具有更准确的特征描述能力，从而能够更加精确地刻画网络蠕虫特征，降低蠕虫检测的误报率和漏报率。

2 变形蠕虫正则表达式特征自动提取方法

2.1 方法概述

本文提出的变形蠕虫正则表达式特征自动提取方法的步骤如图 1 所示，共分 4 步。第 1 步为蠕虫传播网络流样本获取，利用 HoneyBow（蜜罐系统）[15]中已捕获的蠕虫二进制样本，在蜜网环境中生成并获取蠕虫传播的网络流样本；第 2 步为特征树生成，通过多序列比对算法生成蠕虫网络流样本的一组简单正则表达式特征，并动态组织成特征树；第 3 步为高假阳性特征剔除，将特征树中可能引起大量误报的特征剔除，输出最小覆盖特征集；第 4 步为特征融合，将最小覆盖特征集融合为一个或多个复杂正则表达式特征，该特征可用于精确的蠕虫检测。以上 4 步将在 2.2 节~2.5 节分别进行介绍。

2.2 蠕虫传播网络流获取

本文使用基于高交互式蜜罐技术的恶意代码自动捕获器 HoneyBow[15]来进行蠕虫样本的捕获。HoneyBow 利用真实的存有安全漏洞的服务来诱骗蠕虫感染，待蠕虫感染虚拟机后将之捕获并记录下该蠕虫样本的外连数据流。对于选定的一个蠕虫样本，HoneyBow 可利用杀毒软件对该样本进行分析，如果杀毒软件未能识别则就可能为未知蠕虫样本。此时，将该蠕虫样本的外连数据流预处理为网络流样本，交由特征生成算法的后续步骤进行处理。

2.3 特征树生成

2.3.1 SRE 特征及特征提取算法

给定某个蠕虫的一组网络流样本，本文采用文献[14]所提出的基于多序列比对的特征提取方法，生成“简化正则表达式特征”——SRE（simplified regular expression）特征[13]。SRE 特征只包含 2 种正则表达式元字符，分别为“.*”和“.{k}”，“.*”表示任意（包括零）长度的字符串；“.{k}”表示由 k 个字符组成的字符串。例如，“one.{2}two.*”是一个 SRE 特征。

本文将文献[12, 13]提出的基于多序列比对的特征提取方法抽象为函数 *MAlign*()，其输入为字符串或 SRE 特征的集合，输出为一个 SRE 特征。例如，*MAlign*("oxnxexzxtwoxxw","ytwoyownyeyz", "cvcvcvtwovcwc") =“.*‘two’.{2}‘w’.*”; *MAlign*("'*abc*'.{2}'*d*'","'*ab*'.*'*d*'") =“‘ab’.*‘d’”。

2.3.2 特征树定义

文献[14]形式化定义了 SRE 特征的“包含关系”和“更精确关系 p”，使得 2 个 SRE 特征可以比较：$X < Y$ 称为“Y 包含 X”，意义是能够匹配特征 X 的字符串都能够匹配特征 Y；X p Y 称为“X 比 Y 更精确”，意义是能够匹配特征 X 的字符串都能够匹配特征 Y，并且特征 X 的“长度更长”[14]（可理解为包含更多信息）。例如，“‘abc’.*‘cd’” < “‘ab’.*‘cd’”, “‘ab’.{2} ‘cd’” < “‘ab’.* ‘cd’”; “‘aaa’. *‘b’” p “‘aa’.*‘b’”, “‘aa’.{2}‘b’” p “‘aa’.*‘b’”。

定义 1 特征树（signature tree）是一个有根树，其中每个节点都是一个网络数据流（样本）的集合，并且被赋予了一个 SRE 特征，并用 *x.sig* 表示节点 x 被赋予的特征。特征树中，根节点被赋予最泛化的特征“*”，每条边都表示“更精确关系 p”。特征树必须满足下面 2 个性质：

1) 任何一个样本 $f \in$ 节点 C，则 $f < C.sig$；

2) 节点 C_1 是节点 C_2 的子节点，则 $C_1.sig$ p $C_2.sig$。

特征树是本文方法的重要中间结果，表示了一组简单正则表达式（SRE[14]）的逻辑关系，是后续步骤生成复杂正则表达式的基础。图 2 右侧给出了特征树的一个例子。

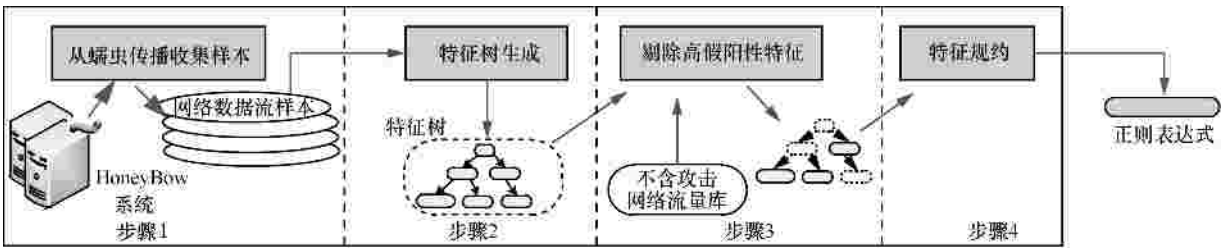

图 1 方法主要流程

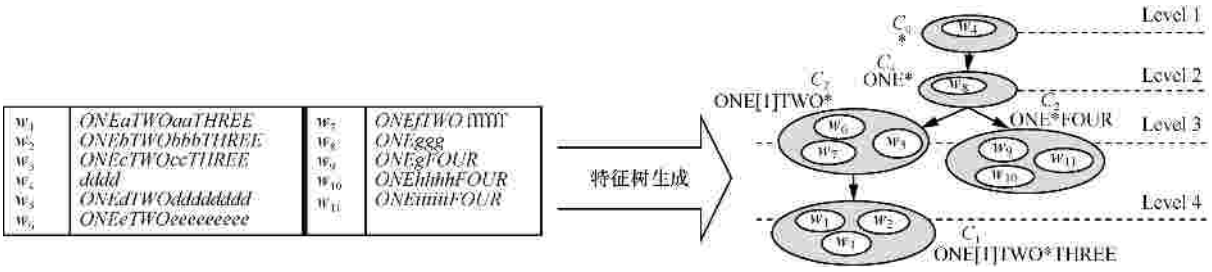

图 2　生成一组样本的特征树

2.3.3　特征树生成算法

特征树生成算法是本文方法的核心，如算法 1 所示。该算法是一个增量式算法，其基本思想是：初始时刻，特征树中只包含一个根节点，该根节点被赋予最泛化的特征“*”；然后逐个选择样本调用 *SigTreeUpdate* 过程对特征树进行更新；*SigTreeUpdate* 函数主要由样本聚类、新特征提取和特征树更新调整 3 个步骤组成。图 2 说明从左侧的 11 个样本，特征树生成算法如何产生右侧的特征树，其中，C_i 表示第 i 个生成的节点。

分析算法 1 可知越精确的特征总是处在越深（层数越大）的层中。同时，加入到特征树中的样本总是会“沉”到所在层数最大的节点中。因此，对于变形蠕虫的 w，在由算法 1 生成的特征树中，同一层中只可能有一个节点（该节点的特征是 w 的特征之一）含有 w 的样本，即变形蠕虫 w 可被聚类，并可提取特征。若样本集包含来自 t 个攻击的 K 个样本，每个样本的平均长度为 l，则算法的时间开销为 $O(Kt^q l^2)$。

算法 1　*SigTreeGeneration*(S)

输入：样本集 S；

输出：特征树 *STree*；

生成一个只包含根节点（节点特征“*”）的新特征树 *STree*；

do 从 S 中随机取出一个样本 f，*SigTreeUpdate*(f, *STree*)

until S 为空

输出 *STree*

Procedure *SinkSamples*(C_T, *STree*)

在 *STree* 中，将所有满足下列 3 个条件的样本 f 移动到 C_T：

$f \in C$，$C.level = C_T.level - 1$，并且 $f < C_T.sig$

Procedure *MoveSubtree*(C_S, C_T, *STree*)

在 *STree* 中，将以 C_S 为根的子树移动到 C_T（C_S 成为 C_T 的子节点）；

对以 C_S 为根的子树中的每一个节点 C' 调用 *SinkSamples*(C', *STree*)

Procedure *SigTreeUpdate*(f, *STree*)

//步骤 1　样本聚类

按照下面的顺序查找满足 $f < C_{belong}.sig$ 的节点 C_{belong}：层次从高到低；如果在最高的某一层中有多个节点满足条件，则选择加入该层最早的那一个节点；

$C_{belong} \leftarrow C_{belong} \cup \{f\}$　//将 f 加入节点 C_{belong}

//步骤 2　新特征提取

if C_{belong} 包含一个子集 $C_{new} = \{f_1, f_2, \Lambda, f_{q-1}, f\}$ 满足 $MAlign(C_{new}) \pi C_{belong}.sig$，then

//产生新节点

$C_{belong} \leftarrow C_{belong} \setminus C_{new}$；

将 C_{new} 增加到攻击特征树中成为 C_{belong} 的子节点；//从 C_{belong} 分裂出一个新的节点 C_{new}

$C_{new}.sig \leftarrow MAlign(C_{new})$；

SinkSamples(C_{new})；

//步骤 3　特征树更新调整

对于所有满足 $C \neq C_{new}$，$C.level = C_{new}.level$ 并且 $C.sig \pi C_{new}.sig$ 的节点 C，do

MoveSubtree(C, C_{new});

// 移动 C 使之成为 C_{new} 的子节点

if C 是 C_{belong} 的子节点，并且 *Align*($C.sig$, $C_{new}.sig$) $\pi C_{belong}.sig$, then

产生一个新的节点 $C_{parent} = \{\}$ 作为 C_{belong} 的子节点；

$C_{parent}.sig \leftarrow Align(C_{new}.sig, C.sig)$;

MoveSubtree(C_{new}, C_{parent});

MoveSubtree(C, C_{parent});

// 让 C_{new} 和 C 成为 C_{parent} 的子节点

$SinkSamples(C_{\text{parent}})$;

2.4 高假阳性特征剔除

高假阳性特征剔除算法见算法2，其作用是通过不含攻击的数据流库N去除可能引起大量误报的特征，最后输出一个最小覆盖特征集合。算法从根节点开始遍历特征树，如果一个节点特征能够满足误报率要求，则用它代替其子节点。

算法2 $SigRemove(STree)$

输入：特征树$STree$；

输出：特征集合S；

对于每一个节点$C \in STree$，**通过正常数据流库**N**计算误报率**C_{FP}；

$S \leftarrow \{r\}$，**其中，**r**是**$STree$**的根节点；**

do

对S**中满足**$C_{\text{FP}} > r$**或包含的样本数为零的节点**C，do

$S \leftarrow S/\{C\}$；

if $C_1, C_2, \perp, C_k$ **是**C**的子节点，**

then $S \leftarrow S \cup \{C_1, C_2, \perp, C_k\}$；

until S **不再改变；**

输出S

由于算法2**输出的特征不含具有父子关系的冗余特征，所以算法**2**输出的特征数量少同时满足误报率要求。算法**2**的计算开销为**$O(|N||STree|)$，**其中，**$|N|$**是正常数据流库的大小，**$|STree|$**是攻击特征树**$STree$**中包含的特征数量。**

2.5 特征规约

特征规约的作用是等价合并特征，输出描述能力更强数量更少的特征。特征规约算法见算法3，定理1证明了该算法的正确性。算法通过5条规则进行规约，其中，第1~3条规则引入或关系元字符（“|”），第4条规则对冗余或关系进行规约，第5条规则引入元字符“$(c)\{k\}$”。

算法3 $SigMerge(S)$

输入：特征集合S

输出：规约后的特征集合

/*以下X、X_1、Y_2表示SRE特征，A、A_1、A_2、A_3表示字符串，$\prod_k A$表示字符串的k次连接*/

while S **中存在特征能够满足下列**5**条规则之一**, do

if $X = X_1A$，$Y = Y_1A$，then $X' = (X_1|Y_1)A$，$S \leftarrow S/\{X,Y\}, S \leftarrow S \cup \{X'\}$; continue; //**规则**1

if $X = AX_1$，$Y = AY_1$，then $X' = A(X_1|Y_1)$，$S \leftarrow S/\{X,Y\}, S \leftarrow S \cup \{X'\}$; continue; //**规则**2

if $X = A_1X_1A_2, Y = A_1Y_2A_2$, then $X' = A_1(X_1|Y_1)A_2$, $S \leftarrow S/\{X,Y\}, S \leftarrow S \cup \{X'\}$; continue; //**规则**3

if $X = (X|Y)A$ and $X < Y$，then $X' = YA$，$S \leftarrow S/\{X\}$，$S \leftarrow S \cup \{X'\}$; continue; //**规则**4

if $X = X_1A$ **或** $X = AX_1$ **或** $X = X_1AX_2$，**且** $A = \prod_k A_1 (k \geq 5)$, then

$X' = (A_1)\{k\}$，$S \leftarrow S/\{X\}, S \leftarrow S \cup \{X'\};;$

//**规则**5

end

输出S

定理1 **特征集合**S**经过特征规约算法后得到**S', S'**与**S**等价。**

证明 S'**与**S**等价意思是任何可匹配**S**中某一条规则的样本必然也可匹配**S'**中的某一条规则，反之亦然。显然，只需要证明通过规则**1~5**规约后，规约后的特征**X'**与原特征**X**或**$\{X,Y\}$**等价（**$L(X')=L(Y)$**或**$L(X')=L(X) \cup L(Y)$**）即可。**

1) **对于规则**1**，如果样本**$x \in L(X)$**，则**$x = x_1A$，$x_1 \in L(X_1)$**。由**$x_1 \in L(X_1)$**可知**$x_1 \in L(X_1) \cup L(Y_1) = L(X_1|Y)$**成立，，所以**$x_1A \in L((X_1|Y_1)A)$**，即**$x \in L(X')$**。同理，如果**$x \in L(Y)$**也可证**$x \in L(X')$**。反之，如果样本**$x \in L(X')$**，则必然存在**$x = x_1x_2$**使得**$x_1 \in L(X_1|Y_1)$**且**$x_2 \in L(A)$**。由**$x_1 \in L(X_1|Y_1)$**可知**$x_1 \in L(X_1)$**成立或**$x_1 \in L(Y_1)$**成立。若是**$x_1 \in L(X_1)$**成立且**$x_2 \in L(A)$**，可得到**$x \in L(X)$**；若是**$x_1 \in L(Y_1)$**成立且**$x_2 < A$**，得到**$x \in L(Y)$**。因此，如果样本**$x \in L(X')$**，则**$x \in L(X) \cup L(Y)$**。综上，**$L(X') = L(X) \cup L(Y)$**。**

2) **规则**2**、规则**3**的证明过程与**1**）类似。**

3) **对于规则**4**，如果样本**$x \in L(X)$**，则**$x = x_1A$**使得**$x_1 \in L(X|Y) = L(X) \cup L(Y)$**。如果**$x_1 \in L(X)$**，由**$X \in L(Y)$**可得到**$x_1 \in L(Y)$**，**$x = x_1A \in L(YA)$**；如果**$x_1 \in L(Y)$**显然**$x \in L(YA)$**。反之，如果**$x \in L(X')$**，则**$x = x_1A$**且**$x_1 \in L(Y)$**，此时显然**$x \in L((X|Y)A) = L(X')$**成立。综上，**$L(X') = L(X)$**成立。**

4) **规则**5**只是将**$\prod_k A_1$**替换为简化表示**$(A_1)\{k\}$**，因此**$L(A) = L((A_1)\{k\})$**，显然**$L(X') = L(X)$**也成立。**

由上述证明过程，可知通过规则1**~规则**5**进行**

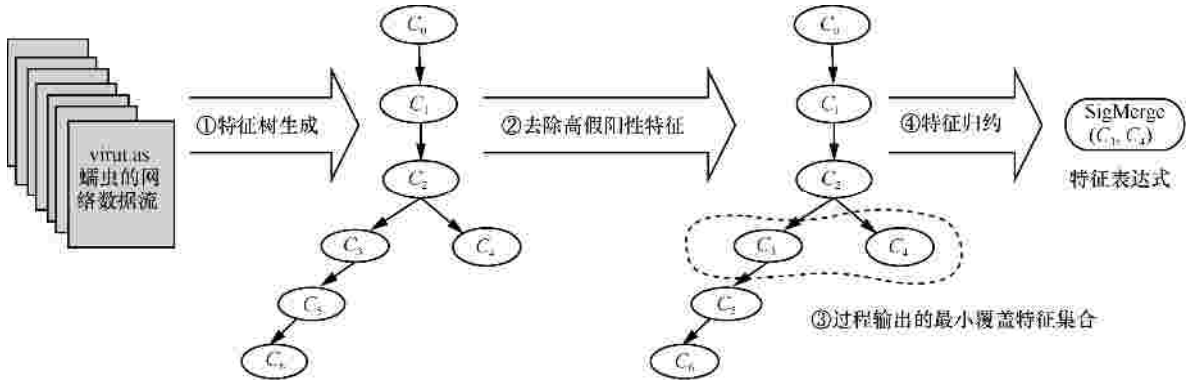

图 3　本方法提取 Virut.n 蠕虫传播特征

规约后，S 保持等价。

3　实验

3.1　实验方法

现有方法研究主要通过人工生成的模拟蠕虫样本进行实验。本文首先利用模拟的变形蠕虫样本评估特征提取的准确性。为 CodeRed 蠕虫生成 3 个变种 CodeRed.A、CodeRed.B、CodeRed.C，每个变种 20 个样本。

此外，本文对互联网上的真实蠕虫进行了实验。首先，利用了互联网上在线部署的 HoneyBow 系统进行蠕虫样本的捕获。对于选定蠕虫样本将其传播网络流记录为样本文件，这些样本一半用于特征提取，一半用于所提取特征的漏报率测试。实验中，算法 2 高假阳性特征剔除算法中的 r 设为 0.001%，用于误报率测试的正常数据流库来自 2007 年 9 月从国防科技大学网关上捕获的 24GB 的网络数据以及 DARPA 99 数据集[16]中的正常数据。

3.2　Virut.n 蠕虫传播特征提取示例

给定 Virut.n 蠕虫的 31 个传播数据流样本，图 3 显示了本方法提取 Virut.n 蠕虫特征的过程。可以看到，首先，2.3 节介绍的特征树生成算法生成包含 7 个特征节点的特征树，然后，2.4 节介绍的特征剔除算法输出特征 C_3 和 C_4，最后，2.5 节介绍的特征规约算法将特征 C_3 和 C_4 等价合并成最后的正则表达式特征（如表 1 第 2 行）。

3.3　模拟蠕虫数据实验结果

表 1 对比了已有主要特征提取方法与本文方法输出的特征以及性能比较。从表中可以看出本文方法输出的特征既包含全部特征子串，描述了它们的位置关系，并且包含由于 3 种蠕虫变种引起的特征“(aaa|bbb|ccc)”，因此本文方法提取的特征更加准确。从表 1 的性能对比可以看出，本文方法相比通过序列比对提取简单正则表达式的方法，在准确性更高的同时还大幅地减少了时间开销。这得益于特征树生产算法在特征提取过程中的“分治”作用。

3.4　真实蠕虫数据实验结果

本文对 Honeybow 系统捕获的被卡巴斯基识别为 virut.n、virut.a、rbot.bsz、sasser.d 的 4 个蠕虫样本进行了特征提取，产生的结果如表 2 所示。其中，第 1 列是蠕虫样本的名称，第 2 列是分析的蠕虫传播数据流样本数量以及特征提取所用时间，第 3 列是产生特征树中特征节点的数量，第 4 列是最后输出的正则表达式特征，第 5 列和第 6 列分别是输出特征的误报率和漏报率。可以看出，对 4 种蠕虫本文方法提取的正则表达式元字符均超过 20 个，且误报和漏报均为 0，因此特征具有较好的准确性。

表 1　　针对 60 个模拟蠕虫样本的特征提取准确性比较

方法	生成的特征	性能/s
早期单个子串方法[4, 5]	‘.ida?’ or ‘%u7801’	18.5
子串序列方法[9]	GET /.*.ida?.*XX.*%u.*%u7801.*HTTP/1.0\r\n	11
子串集合方法[10]	{‘.ida? ’: 1, ‘%u780’:1,‘HTTP/1.0\r\n’:1, ‘GET/’: 1,‘%u’: 2}	12.5
简单正则表达式方法[13]	‘GET/’.*‘.ida?’.*‘XX’[15]%u.*%u780.*=.[7]HTTP/1.0\r\n	869
本文方法	GET/.*.ida?.*XX.[15]%u.*%u780.*=(aaa|bbb|ccc)HTTP /1.0\r\n	42

表 2 提取的真实蠕虫的正则表达式特征

蠕虫名称	样本数量/运行时间	特征树节点数量	归并后的输出的正则表达式特征	假阳性	假阴性
virut.n	31/719	6	.*0x000x000x00.{1}0xffSMBr0x000x000x000x00(0x18S0xc80|0x080x01@)0x00{14}.{2}0x00.{3}0x00.{1}0x000x02 PC NETWORK PROGRAM 1.*.00x000x02LANMAN1.0.*0x000x02LM1.2X002.*0x000x02NT LM 0.120x000x000x00 .{ 2}0xffSMBs0x000x000x000x00.{3}0x000x000x00.*0x00{9}.*0x000x000x00.*0x000x000x000x00.{1}0xff.*0x000x000x00 .{3}0x00{7}.*0x000x000x000x00.*0x000x00.*0x010x00.*	0	0
virut.as	40/902	9	1. 0x810x000x00H CKFDENECFDEFFCFGEFFC(CA){6}0x00 EMEPEDEBEMEIEPFDFE(CA){6}AA 0x000x000x{5}.{1}0xffSMBr0x000x000x000x00.{3}0x00{14}.{2}0x00.{1}0x00.{1}0x00.{1}0x000x02PC NETWORK PROGRAM 1.*.00x000x02LANMAN1.0.*0x000x02LM1.2X002.*0x000x02NT LM .120x000x000x00.{2} 0xffSMBs0x000x000x000x00.{3}0x000x000x00.*0x00{9}.*0x000x000x00.*0x000x000x000x00.{ 1}0xff.* 0x000x000x00.{3}0x00{6}.*0x000x000x000x00.*0x000x00.*0x010x00.*" 2. .*0x000x000x000xa80xffSMBr0x000x000x000x000x080x01@0x00{14} 0x020x000x08.{2}0x000x850x000x02PC NETWORK PROGRAM 1.00x000x02MICROSOFT NETWORKS 1.030x00 SOFT NETWORKS 00x000x02 LANMAN1.00x000x02LM1.2X0020x000x02LANMAN2.10x000x02NT LANMAN 1.00x000x02NT LM 0.120x00 0x000x000x00b0xffSMBs0x000x000x000x000x080x01@0x00{14}0x020x000x08.{2}0x0d0xff0x000x000x000x00D 0x020x000xfc0x090x00{12}0xf00x000x000x00%0x000x000x00Windows 2000 21950x00Windows 2000 5.00x000x000x00 0x00D0xffSMBu0x000x000x000x000x180x01 0x00{14}0x020x000x08.{2}0x040xff0x00{5}x010x000x190x000x00\\ .*SMBSERVER\IPC$0x00?????0x000x000x000x00Y0xffSMB0xa20x000x000x000x000x180x01 0x00{12}0x08 0x02 0x000x08.{2}0x180xff0x000x000x000x000x050x000x160x00{7}x9f0x010x020x00{13}0x070x000x000x000x010x000x00 0x00@0x000x000x000x020x000x000x000x000x060x00\samr0x000x000x000x000x920xffSMB%0x000x000x000x000x18 0x01 0x000{13}x08 0x020x000x08.{2}0x100x000x00H0x000x000x040xe00xff0x00{12}J0x00H0x00J0x000x020x00&0x00 0x00@O0x00\PIPE\0x000x050x000x0b0x030x100x000x000x00H0x00{6}xd00x160xd00x160x000x000x000x000x01 0x00{5}x010x00j(0x1990x0c0xb10xd00x110x9b0xa80x000xc0O0xd9.0xf50x000x000x000x000x04]0x880x8a0xeb0x1c0xc9 0x110x9f0xe80x080x00+0x10H`0x020x00{5}x0c0xdf0xffSMB/0x000x000x000x000x180x01 0x00{13}0x08 0x020x000x08 .{2}0x0e0xff0x000x000x000x00@0x000x000x000x000xff0xff0xff0xff0x080x000xa00x0c0x000x000xa00x0c?0x00{5} xa00x0c0x050x000x000x010x100x000x000x000xa00x0c0x00{6}x880x0c0x000x000x000x000x090x000xec0x030x00{6} xec0x030x000x000x90{160}0x810xc40xff0xef0xff0xffD0xeb0x020xebk0xe80xf90xff0xff fSUVW0x8bl$0x180x8bE<0x8bT (x0x030xd50x8bJ0x180x8bZ 0x030xdd0xe32I0x8b40x8b0x030xf530xff0xfc30xc00xac80xe0t0x070xc10xcf0x0d0x030xf80xeb 0xf2;|$0x14u0xe10x8bZ$0x030xddf0x8b0x0cK0x8bZ0x1c0x030xdd0x8b0x040x8b0x030xc50xeb0x0230xc0_.*	0	0
rbot.bsz	40/2160	1	0x810x000x00D KFDENECFDEFFCFGEFFC(CA){23}AA0x000x000x000x000x850xffSMBr0x000x000x000x000x18S 0xc80x00{12}x0070x130x00{5}b0x000x02PC NETWORK PROGRAM 1.00x000x02LANMAN1.0 0x000x02Windows for Workgroups 3.1a0x000x02LM1.2X0020x000x02LANMAN2.10x000x02NT LM 0.120x000x00 0x000x100xbf0xffSMBs0x000x000x000x000x180x070xc80x00{12}x0070x130x000x000x000x000x0c0xff0x00 0x000x000x040x110x0a0x00{7}~0x100x000x000x000x000xd40x000x000x80~0x10`0x820x10z0x060x06 +0x060x010x050x050x020xa00x820x10n00x820x10j0xa10x820x10f#0x820x10b0x030x820x040x010x00A{70}	0	0
Sasser.d	48/552	13	0x000x000x000x850xffSMBr0x000x000x000x000x18S0xc80x00{14}xff0xfe0x00{5}b0x000x02PC NETWORK PROGRAM 1.00x000x02LANMAN1.00x000x02Windows for Workgroups 3.1a0x000x02LM1.2X0020x000x02 LANMAN2.10x000x02NT LM 0.120x000x000x000x000xa40xffSMBs0x000x000x000x000x180x070xc80x00{19} 0x100x000x0c0xff0x000xa40x000x040x110x0a0x00{12}0x000xd40x000x000x80i0x00NTLMSSP0x000x010x000x00 0x000x970x820x080xe00x00{17}W0x00i0x00n0x00d0x00o0x00w0x00s0x00 0x0020x0000x0000x0000x000x0020x001 0x0090x0050x000x000x00W0x00i0x00n0x00d0x00o0x00w0x00s0x00 0x0020x0000x0000x0000x00 0x0050x00.0x00{8} xda0xffSMBs0x000x000x000x000x180x070xc80x00{14}xff0xfe0x000x08 0x000x0c0xff0x000xda0x000x040x110x0a0x00{7} x00W0x00{5}xd40x000x000x800x9f0x00NTLMSSP0x000x030x000x000x000x01 0x010x00F0x00{14}@0x00{7} @0x000x000x000x060x000x060x00@0x000x000x000x100x000x100x00G0x000x000x000x150x8a0x880xe0H0x00O0x00D 0x000x000x810x19jz0xf20xe4I0x1c(0xaf0%t0x10gSW0x00i0x00n0x00d0x00o0x00w0x00s0x00 0x0020x0000x0000x0000x00 0x0020x0010x0090x0050x000x000x00W0x00i0x00n0x00d0x00o0x00w0x00s0x00 0x0020x0000x0000x0000x00 0x0050x00. 0x0000x00{8}[1]0xffSMBu0x000x000x000x000x180x070xc80x00{14}0xff0xfe0x000x0800x000x040xff0x00\0x000x08 0x000x010x00.{1}0x000x00\0x00\0x0010x0090x0020x00.0x0010x0060x0080x00.0x00.*0x00.*0x00.{1}0x00.{1}0x00\0x00 i0x00p0x00c0x00$0x000x000x00?????0x000x000x000x00d0xffSMB0xa20x000x000x000x000x180x070xc80x00{13} x080xdc0x040x000x08@0x000x180xff0x000xde0xde0x000x0e0x 160x00{7}x9f0x010x020x00{13}x030x000x00 0x000x010x000x000x00@0x000x000x000x020x000x000x000x030x110x000x00\0x00l0x00s0x00a0x00r0x00p0x00c0x00{6} x9c0xffSMB%0x000x000x000x000x180x070xc80x00.{13}x080xdc0x040x000x08P0x000x100x000x00H0x000x000x000x00 0x040x00{12}x00T0x00H0x00T0x000x020x00&0x000x00@Y0x000x10\0x00P0x00I0x00P0x00E0x00\0x00{5}x050x00 0x0b0x030x100x000x000x00H0x000x000x000x010x000x000x000xb80x100xb80x100x000x000x000x000x010x00{5}0x01 0x00j(0x1990x0c0xb10xd00x110x9b0xa80x000xc0O0xd9.0xf50x000x000x000x000x04]0x880x8a0xeb0x1c0xc90x110x9f 0xe80x080x00+0x10H`0x020x00{5}0x0c0xf40xffSMB%0x000x000x000x000x180x070xc80x00{13}0x080xdc0x040x00 0x08`0x000x100x000x000xa00x0c0x000x000x000x040x00{12}T0x000xa00x0cT0x000x020x00&0x000x00@0xb10x0c0x10 \0x00P0x00I0x00P0x00E0x00\0x00{5}0x050x000x000x030x100x000x000x000xa00x0c0x000x000x010 x000x000x88 0x0c0x000x000x000x000x090x000xec0x030x00{6}xec0x03*0x900x900x900x900x900x900x900x900x90"	0	0

4 方法比较

以表 1 为例，表 3 给出了本文方法与现有攻击特征提取方法在准确方面的比较。早期的方法[4, 5, 8]特征提取准确性差，大量信息不被发掘。基于 Token 频率的方法（如 Polygraph[9]和 Hamsa[10]）主要问题是不能提取长度极短的特征片段和特征片段之间的距离关系。较新提出的基于序列比对的方法[12~14]可提取简单正则表达式特征，缺点是不包含“|”等元字符因而特征描述能力有限。从表中可以看出，相比现有方法本文方法特征提取准确性更好。

表 3　本文方法与现有特征提取方法的比较

方法	特征准确性				
	能否提取多个特征片段	能否提取长度极短的特征片段	能否提取特征片段之间的顺序（.*）	能否提取特征片段之间的距离关系（.{k}）	能否提取特征片段的或关系（|）
早期方法[4,5,8]	√	×	×	×	√
Polygraph[9]	√	×	√	×	×
Hamsa[10]	√	×	×	×	√
文献[12]	√	√	√	×	×
SRE[13, 14]	√	√	√	√	×
本文方法	√	√	√	√	√

5　结束语

与现有攻击特征自动提取方法相比，本文方法输出的特征更接近完整形式的正则表达式，因此具有更强的攻击描述能力，准确性更好。使用 HoneyBow（蜜罐系统）捕获的蠕虫样本，在真实互联网环境中测试了该方法的有效性。实验表明，该方法可准确提取互联网上真实蠕虫样本的正则表达式特征，并可用于准确地检测网络蠕虫传播。该方法可集成于蜜罐与蜜网、蠕虫特征自动提取、恶意代码分析沙盒等系统中，用于快速提取包括未知样本在内的网络蠕虫的特征。本文方法提取的正则表达式特征中仍存在着一些冗余特征字符（例如部分冗余的网络协议字段），在保证检测精确度的前提下，如何自动提取更为简略、语义更为明确的正则表达式特征，以及如何进一步在实际环境中应用、通过量化指标评价特征提取的准确性均可成为下一步研究方向。

参考文献：

[1] FOGLA P, SHARIF M, PERDISCI R, *et al*. Polymorphic blending attacks[A]. Proceedings of the 15th Conference on USENIX Security Symposium[C]. Berkeley, CA, USA, 2006. 17.

[2] GUNDY M V, BALZAROTTI D, FIELDSCHEMA G V. Catch me, if you can: evading network signatures with Web-based polymorphic worms[A]. Proceedings of the First USENIX Workshop on Offensive Technologies (WOOT)[C]. Boston, MA, USA, 2007.

[3] **唐勇, 卢锡城, 王勇军**. **攻击特征自动提取技术综述**[J]. **通信学报**, 2009, 30(2): 96-105.
TANG Y, LU X C, WANG Y J. Survey of automatic attack s gnature generation[J]. Journal on Communications, 2009, 30(2): 96-105.

[4] KREIBICH C, CROWCROFT J. Honeycomb- creating intrusion detection signatures using honeypots[A]. Proceedings of the Second Workshop on Hot Topics in Networks (Hotnets II)[C]. Boston, 2003. 51-56.

[5] WANG K, CRETU G, STOLFO S J. Anomalous payload-based worm detection and signature generation[A].Proceedings of Recent Advances in Intrusion Detection (RAID)[C]. 2003. 227-246.

[6] SINGH S, ESTAN C, VARGHESE G, *et al*. Automated worm fingerprinting[A]. Proceedings of the 6th USENIX OSDI[C]. 2004. 45-60.

[7] KIM H A, KARP B. Autograph: toward automated, distribu worm signature detection[A]. Proceedings of USENIX Security Symposium[C]. 2004. 271-286.

[8] SINGH S, ESTAN C, VARGHESE G, *et al*. Automated worm fingerprinting[A]. Proceedings of the 6th USENIX OSDI[C]. San Francisco, CA, 2004. 45-60.

[9] NEWSOME J, KARP B, SONG D. Polygraph: automatically generating signatures for polymorphic worms[A]. Proceedings of IEEE Symposium on Security and Privacy[C]. Washington, DC, USA, 2005. 226-241.

[10] LI Z, SANGHI M, CHEN Y, *et al*. Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience[A]. Proceedings of IEEE Symposium on Security and Privacy[C]. Washington, DC, USA, 2006. 32-47.

[11] YEGNESWARAN V, GIFFIN J T, BARFORD P, *et al*. An architecture for generating semantics-aware signatures[A]. Proceedings of the 14th USENIX Security Symposium[C]. Baltimore, MD, USA, 2005. 97-112.

[12] **唐勇, 卢锡城, 胡华平等**. **基于多序列联配的攻击特征自动提取技术研究**[J]. **计算机学报**, 2006, 29(9): 1533-1541.
TANG Y, LU X C, HU H P, *et al*. Automatic generation of attack signatures based on multi-sequence alignment[J]. Chinese Journal of computers, 2006, 29(9):1533-1541.

[13] TANG Y, LU X, XIAO B. Generating simplified regular expression Signatures for polymorphic worms[A]. Proceedings of the 4th International Conference on Autonomic and Trusted Computing (ATC-07)[C]. 2007. 478-488.

[14] TANG Y, LU X, XIAO B. Using a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms[J]. Comput, Secur, 2009.

[15] **诸葛建伟, 韩心慧, 周勇林等**. HoneyBow:**一个基于高交互式蜜罐技术的恶意代码自动捕获器**[J]. **通信学报**, 2007, 28(12): 8-13.
ZHUGE J W, HAN X H, ZHOU Y L, *et al*. HoneyBow: an automated malware collection tool based on the high-interaction honeypot principle[J]. Journal on Communications, 2007, 28(12): 8-13.

[16] LIPPMANN R, HAINES J W, FRIED D J, *et al*. The 1999 DARPA off-line intrusion detection evaluation[J]. Comput, Networks, 2000, 34(4): 579-595.

作者简介：

唐勇（1979-），男，湖南衡阳人，博士，国防科技大学助理研究员，主要研究方向为网络与信息安全、数据挖掘。

诸葛建伟（1980-），男，浙江瑞安人，博士，清华大学副研究员，主要研究方向为网络与系统安全。

陈曙晖（1974-），男，湖南益阳人，博士，国防科技大学副教授，主要研究方向为网络协议、网络安全。

卢锡城（1946-），男，江苏靖江人，国防科技大学教授、博士生导师，中国工程院院士，并行与分布处理国防科技重点实验室主任，主要研究方向为计算机网络、并行与分布处理等。