针对指令乱序变形技术的归一化研究

金 然 魏 强 王清贤
(信息工程大学信息工程学院 郑州 450002)

摘 要 新出现的恶意代码大部分是在原有恶意代码基础上修改转换而来。许多变形恶意代码更能自动完成该过程,由于其特征码不固定,给传统的基于特征码检测手段带来了极大挑战。采用归一化方法,并结合使用传统检测技术是一种应对思路。本文针对指令乱序这种常用变形技术提出了相应的归一化方案。该方案先通过控制依赖分析将待测代码划分为若干基本控制块,然后依据数据依赖图调整各基本控制块中的指令顺序,使得不同变种经处理后趋向于一致的规范形式。该方案对指令乱序的两种实现手段,即跳转法和非跳转法,同时有效。最后通过模拟测试对该方案的有效性进行了验证。

关键词 变形恶意代码,归一化,恶意代码检测

Research on Normalization towards Instructions Reordering Metamorphism Technique

JIN Ran WEI Qiang WAN Qing-Xian
(Information Engineering Institute, Information Engineering University, Zhengzhou 450002)

Abstract Much of apparently new malware comes from transformed known malware. Metamorphic malware could even complete this process automatically. The mutable signature makes the traditional detection method based on it difficult to detect metamorphic malware. Combining normalization idea with the traditional detection technology is a promising approach to resolve the problem. This paper proposes a normalization scheme towards instructions reordering metamorphism technique. In the scheme, the inspected code is firstly partitioned into some basic control blocks based on control-dependency analysis, then the instructions order in each block is adjusted according to the data-dependency graph. After the variants of malware are normalized according to the scheme, they tend to have the same form. The scheme is applicable to both jump method and non-jump method which are two implementations of instructions reordering. Testing has been conducted to validate the feasibility of the scheme.

Keywords Metamorphic malware, Normalization, Malware detection

1 引言

据统计,新出现的恶意代码中大部分是在原有恶意代码基础上修改转换而来[1]。变形[2](Metamorphism)技术是其中一种高级转换技术,它采用程序变换方法,在保持语义等价的前提下,改变代码形态。近年来出现的多种变形病毒(如Win32. Evol, Win32. Zmist, Win32. Metaphor 等)[2]更能自动完成这一过程。由于变换前后的代码不论是在静态文件中,还是动态执行在内存里,都不呈现相同的特征码,因此给基于特征码的检测方法带来了极大挑战[3]。

采用归一化方法,结合传统的基于特征码检测技术,是一种应对变形恶意代码的思路[4]。其主要思想是先将待测代码进行归一化处理,将其变换为规范形式,然后再使用匹配特征码的方法对规范代码进行检测。图1显示了该方法的整个过程:已知M为恶意代码,将其归一化后提取代码特征;对于待检测代码C1,…,Cn,同样先进行归一化,然后提取输出结果特征码与M的进行匹配,以此来判断待检测代码是否为M的变种。

恶意代码常用变形技术有:等价指令替换,插入垃圾代码和指令乱序,其中指令乱序有两种不同实现方式:跳转法和非跳转法。许多变形恶意代码都采用了其中一种或多种技术[2],因此研究针对这些变形技术的归一化策略有着重要意义。Christodorescu 和 Bruschi 分别在文[5,6]中针对这些变形技术讨论了相应的归一化方案,不过对于指令乱序,他们的方案仅对跳转法有效。本文专门针对指令乱序进行探讨,提出了一种适用于跳转法和非跳转法的归一化方案。通过模拟

金 然 博士研究生,主研方向:网络安全;魏 强 博士研究生,主研方向:网络安全;王清贤 教授,博导,主研方向:算法分析,网络安全。

结束语 测试表明,利用信息熵和决策分类技术对邮件进行分类识别,通过优选构建训练样本和测试样本能够通过学习得到能识别垃圾邮件的过滤规则,具有一定的智能性。该模型的识别准确率与训练样本的选择和构成、邮件特征选择以及评估阈值相关。阈值越小,准确率越高,但是将正常邮件误判为垃圾邮件的情况也较多;反之则相反。

参 考 文 献

1 赵晓明,郑少仁. 电子邮件过滤器的分析与设计. 东南大学学报,2001(9)
2 史忠植. 知识发现. 清华大学出版社,2002(1)
3 丁岳伟. 基于 SMTP 协议电子邮件的还原. 小型微型计算机系统,2002(3)
4 曹麒麟,张千里. 垃圾邮件与反垃圾邮件技术. 人民邮电出版社,2003
5 陈文伟、邓苏、张维明. 数据挖掘与知识发现综述. 计算机世界报,1997, 24期专题版
6 Postel J B. RFC821 simple mail transfer protocol. USA: IETF, 1982. 4~18
7 Freed N, Borenstein N. RFC2045 Multipurpose Internet Mail Extensions(MIME) part one: format of Internet message bodies. USA: IETF, 1996. 6~27

测试，验证了该方案的有效性。以下首先对恶意代码常用变形技术进行简单的介绍。

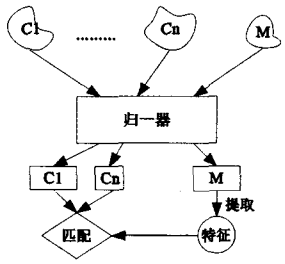

图 1　采用归一化方法检测恶意代码

2　恶意代码常用变形技术

1)等价指令替换。将指令替换为在语义上等价的其它指令(序列)。以图 2(a)为例，若将指令 1 替换为(add esp,4; mov [esp],eax)，得到图 2(b)。容易看出变换前后的代码在语义上是等价的。

2)插入垃圾代码。增加冗余操作指令，但不改变原有语义。在图 2(a)中指令 2 前增加一条给寄存器 eax 的赋值指令 mov eax, 0xff，得到图 2(c)。由于 mov eax, [esi]会给 eax 重新赋值，因此增加的指令并不会改变原有程序的语义。

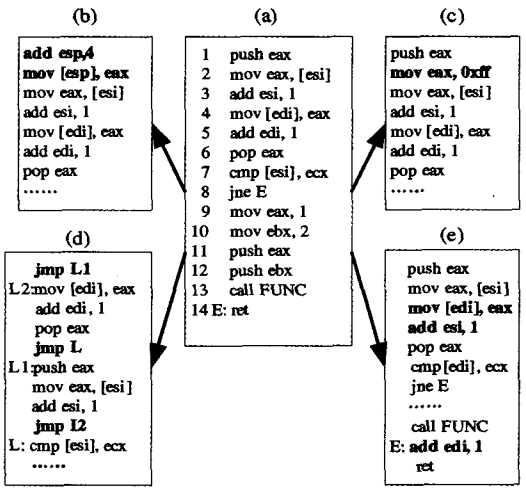

图 2　恶意代码常用变形技术

3)指令乱序。在保持代码语义前提下，混乱指令顺序。

• 跳转法。在改变指令位置的同时插入 jmp 指令以保持相对执行顺序。例如将图 2(a)中指令 1 到指令 6 之间的各条指令位置打乱，并插入相应的 jmp 指令后得到图 2(d)，从指令的执行顺序看，变换前后是一样的。

• 非跳转法。改变指令位置，但无须插入 jmp 指令。例如将图 2(a)中指令 3 调整到指令 4 后，并将指令 5 调整到指令 13 后得到图 2(e)，显然，这样的调整不会改变代码语义。

跳转法与非跳转法的不同在于：跳转法虽然改变了指令相对位置，但没有改变它们之间的相对执行顺序，而非跳转法则既改变了相对位置，又改变了相对执行顺序。

3　规范指令顺序

代码经过随机指令乱序后将得到各种不同形式的变种，规范指令顺序是指令乱序的逆过程，它使得各种不同的变种(包括初始代码)经过处理后趋向于一致的形式，即规范形式。

指令乱序虽然改变了指令的相对顺序，但它并没有改变代码各指令间原有的数据依赖和控制依赖关系。因为如果这些依赖关系被破坏，代码语义也将随之发生变化[7]。例如图 2(a)所示指令序列中，若把指令 2 调整到指令 3 后，将导致原有语义改变，因为指令 3 反向数据依赖于指令 2，这种调整破坏了该依赖关系。又如，若将指令 6 调整到指令 8 后，尽管没有改变各指令间的数据依赖关系，但代码语义也改变了，因为调整后指令 6 将控制依赖于指令 8，而调整前是不存在这种依赖关系的。

由此可得出一个结论：经指令乱序后的各变种(包括初始代码)具有一致的控制依赖和数据依赖关系。于是规范指令顺序可以这样的思路进行：首先分析待检测代码各指令间的控制依赖和数据依赖关系，然后在保持原有依赖关系的前提下按照一定规则调整指令顺序。

3.1　控制依赖分析

代码各指令间的控制依赖关系可以表示成一个控制依赖图。首先给出控制依赖图定义。

定义 1(控制依赖图)　控制依赖图是一个有向无环图 $CDG=(V,E)$。V 分为两类：指令结点 V_I 和区域结点 V_R，$V_I \cap V_R=\Phi$ 且 $V_I \cup V_R=V$。任取 $v_i \in V_I$，$v_j \in V_R$，若 $v_i \rightarrow v_j \in E$，则表示 v_i 代表的指令有一个控制区域，该区域以 v_j 表示，这时 v_i 也被称为控制结点；任取 $v_i \in V_R$，$v_j \in V_I$，若 $v_i \rightarrow v_j \in E$，则表示 v_j 代表的指令控制依赖于和 v_i 相连的控制结点所代表的指令，并且处于该控制结点的 v_i 控制区域中。另外对于 E 中所有边 $v_i \rightarrow v_j$，必有 $v_i \in V_I$，$v_j \in V_R$ 或者 $v_i \in V_R$，$v_j \in V_I$。

图 3(c)描绘了图 2(a)所示代码的控制依赖图。图中以圆型结点表示指令结点，以方型结点表示区域结点，start 结点是一个虚拟指令结点，它所代表的虚拟指令表示对该段代码进行过程调用。可以看出，该图直观的反映了图 2(a)所示代码各指令间的控制依赖关系。

由此，控制依赖分析实际上可等效为构造控制依赖图。以下详细描述该过程。

(1)生成指令流程图

指令流程图描述了代码指令的执行流，它是分析代码各指令间控制依赖关系的基础，因此在构造控制依赖图前需要先生成此图。

指令流程图与控制流图类似，只是它的各结点代表指令，而非基本块，因此可用生成控制流图的方法来生成指令流程图，本文不再赘述此算法。

需要说明的是，在生成的指令流程图中，若某结点对应 jmp 指令，且该结点是其后继结点的唯一前驱，则应将该结点删除，这样可以提前消除由跳转法引入的多余 jmp 指令，利于后续规范处理。另外在指令流程图中还需增加两个虚拟指令结点："start"和"exit"。"start"结点是分支结点，其"Y"边指向代码入口结点，"N"边指向"exit"结点，同时，所有代码出口均指向"exit"结点。图 3(a)描绘了图 2(a)所示代码的指令流程图。

(2)构造控制依赖图

为构造控制依赖图，先构造一种中间形式——基本控制依赖图。基本控制依赖图仅有指令结点而不含区域结点，如果在其基础上加入适当的区域结点就可得到所需的控制依赖图。

由于代码各指令只会控制依赖于代码中的分支转移指

令，如 jne、loop 等，因此构造过程只需以这些分支转移指令为线索进行。下面给出构造基本控制依赖图的算法。

输入：过程或函数 P 以及相应的指令流程图 IFG，P 中各指令按位置顺序从 1 到 M 编号，设 n_i 表示 IFG 中对应于指令 i 的结点。
输出：与 P 对应的基本控制依赖图 BCDG=(V,E)。
初始：$V=\{v_1,\cdots,v_M\}$，$E=\Phi$，$i=1$。
1 如果 $i\leqslant M$，转 2，否则结束；
2 如果指令 i 是分支转移指令，转 3，否则 $i=i+1$，转 1；
3 设 j 从 1 到 $M(j\neq i)$，在 IFG 中，如果 n_j 不是 n_i 的后必经结点，并且存在一条从 n_i 到 n_j 的控制路径，n_j 是该路径上除 n_i 之外的每个结点的后必经结点，则生成一有向边 $v_i\rightarrow v_j$，$E=E\cup\{v_i\rightarrow v_j\}$，当 $j>M$ 时，$i=i+1$ 并转 1。

图 3(b)描绘了图 2(a)所示代码的基本控制依赖图。由于不含区域结点，基本控制依赖图不能全面反映代码各指令间的控制依赖关系，例如对于图 4(a)所示指令流程图，图 4(b)是相应的基本控制依赖图，尽管该图能反映指令 4、5、6、7 控制依赖于指令 3，但实际上指令 4、5 和指令 6、7 分别处于指令 3 的两个不同控制区域，因此需要在基本控制依赖图上增加区域结点。

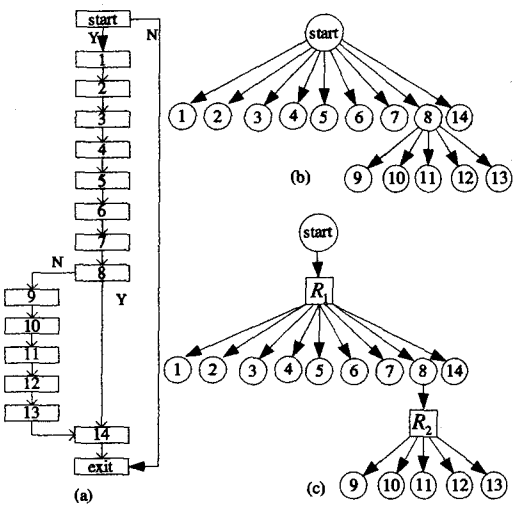

图 3　对应于图 2(a)的指令流程图、基本控制依赖图以及控制依赖图

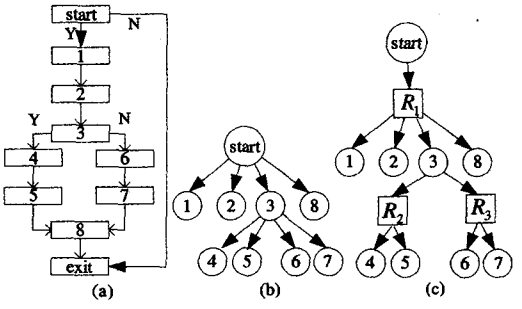

图 4　具有多控制区域的分支转移指令示例

下面给出从基本控制依赖图变换到控制依赖图的算法。

输入：指令流程图 IFG 和相应基本控制依赖图 BCDG=(V,E)，设 n_i 表示 IFG 中对应于指令 i 的结点，v_i 表示 BCDG 中对应于指令 i 的结点。
输出：控制依赖图 CDG。
初始：CDG=BCDG，$i=1$。
1 如果 $i\leqslant|V|$，转 2，否则结束；
2 如果 v_i 是控制节点，转 3，否则 $i=i+1$ 并转 1；
3 将指令节点集 $R=\{v\mid v_i\rightarrow v\in E\}$ 划分为若干子集 $R_1,\cdots,R_N$。使得 $R_1\cup\cdots\cup R_N=R$，且 $R_j\cap R_k=\Phi(1\leqslant j,k\leqslant N,\ j\neq k)$，同时 $\forall v_p,v_q\in R_j(1\leqslant j\leqslant N)$，在 IFG 中，$n_p$ 和 n_q 互为对方的必经结点和后必经结点。转 4。
4 生成与 $R_1,\cdots,R_N$ 对应的区域结点集 $\{r_1,\cdots,r_N\}$，$V=V\cup\{r_1,\cdots,r_N\}$，转 5。
5 $\forall v\in R_j(1\leqslant j\leqslant N)$，若 $v_c\rightarrow v\in E$ $(v_c\in V)$，则 $E=(E\backslash\{v_c\rightarrow v\})\cup\{v_c\rightarrow r_j\}\cup\{r_j\rightarrow v\}$。转 6。
6 $i=i+1$，转 1。

对图 4(b)运用以上算法可以得到图 4(c)。根据控制依赖图可将代码指令划分为若干基本控制块，每个基本控制块与一个区域结点对应，处于同一控制区域的指令组成一个基本控制块，这些控制块表征了代码的整个控制结构，从控制依赖图的构造过程可以看出，基本控制块具有以下性质：

- 一个基本控制块中的所有指令都依赖于相同的其它指令，要么都执行，要么都不执行，并且各条指令依次顺序执行。
- 在基本控制块内调整指令顺序不会影响控制依赖关系，但若在基本控制块间调整则会改变原有控制依赖关系。

因此，在规范指令顺序时，为保持控制依赖关系，只能在各基本控制块中调整指令顺序。

3.2　数据依赖分析

代码各指令间的数据依赖关系可以表示成一个数据依赖图。首先给出数据依赖图定义。

定义 2(数据依赖图)　数据依赖图是一个有向无环图 $DDG=(V,E)$。V 中各结点表示代码各指令，$\forall v_i,v_j\in V$，若 $v_i\rightarrow v_j\in E$，则表示 v_j 代表的指令数据依赖于 v_i 代表的指令。

通过构造数据依赖图可以清楚地了解代码各指令间的数据依赖关系，规范指令顺序时可据此来保持这种关系，不过根据上一节的分析结果，使用全局数据依赖图作为规范依据并不合适，因为调整指令顺序只能在各基本控制块中进行，所以使用各基本控制块的数据依赖图更合适，不过这些依赖图必须要能反映全局数据依赖关系，由此，在构造各基本控制块的数据依赖图时，需要考虑以下 4 种数据依赖关系：

- 流依赖。指令 I_1 写一个寄存器或存储单元，而指令 I_2 使用它，这时，I_2 流依赖于 I_1。
- 反向依赖。指令 I_1 使用一个寄存器或存储单元，而指令 I_2 改变它，这时，I_2 反向依赖于 I_1。
- 输出依赖。指令 I_1 写一个寄存器或存储单元，而指令 I_2 同样改变它，这时，I_2 输出依赖于 I_1。
- 扩展依赖。设控制依赖图中指令结点 v 对应于指令 I_2，从 v 可达的其它指令结点为 $v_1,\cdots,v_n$，与其对应的指令为 $I_{21},\cdots,I_{2n}$，若这些指令中有一条和指令 I_1 有以上 3 种依赖关系，则指令 I_1 和指令 I_2 存在扩展依赖关系。例如图 2(a)中，指令 8 扩展依赖于指令 6。

此外，对于 ret 指令和跳转到其它基本控制块的 jmp 指令，设定它们数据依赖于其它所有指令，因为若它们出现在基本控制块中，则必是最后执行的出口语句，而指令的相对执行顺序主要由依赖关系决定，例如，指令 I_2 数据依赖于指令 I_1，则 I_1 必须在 I_2 前执行，否则这种关系将被破坏，所以这种设定是合理的。显然，ret 指令和跳转到其它控制块的 jmp 指令不会同时处于一个基本控制块中。以下给出构造基本控制块数据依赖图的算法：

输入：基本控制块 B，B 中指令应保持原有执行顺序，设 B 中共有 M 条指令，ins_i 表示第 i 个执行的指令。
输出：B 的数据依赖图 DDG=(V,E)。
初始：$V=\{v_1,\cdots,v_M\}$，$i=1$。
1 如果 $i\leqslant M$，则 $j=i$，转 2，否则，结束；
2 如果 ins_j 数据依赖于 ins_i，则 $E=E\cup\{v_i\rightarrow v_j\}$，转 3；
3 如果 $j\leqslant M$，则 $j=j+1$，转 2，否则 $i=i+1$，转 1。

对图 2(a)所示代码的两个基本控制块使用上述算法得到的数据依赖图如图 5。为简化图示，图 5(a)未画出对应于 ret 指令(指令 14)的结点及其数据依赖边。

3.3　调整指令顺序

根据前面的分析结果，调整指令顺序只能在各基本块中

进行，同时为保持数据依赖关系，需以数据依赖图作为调整依据。以下是对基本控制块进行整序的算法。

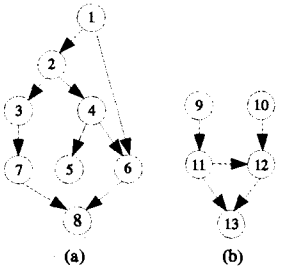

图 5　对应于图 2(a)中两个基本控制块的数据依赖图

输入：基本控制块 B_{in} 和相应的数据依赖图 DDG。
输出：调整指令顺序后的基本控制块 B_{out}。
1 分配指令缓冲池 P，转 2；
2 选取 DDG 中无入边的结点，将其对应指令放入 P，转 3；
3 按输出规则(稍后介绍)，从 P 中选择一条指令输出，将其对应的结点从 DDG 中删除，并删除该结点所有出边，转 4；
4 查看 DDG 和 P 是否空，如果空则结束，否则转 2。

上述算法执行时，P 中可能同时有多条指令，这些指令无依赖关系，都能被输出，因此需要根据一定规则来选择输出指令，以此保证规范的一致性，例如可定义以下两条规则：

- 规定操作符优先关系，例如 add 优先于 mov。如果 P 中操作符优先级最高的指令只有一条，则选择该指令输出。
- 如果 P 中操作符优先级最高的指令有多条，则分别计算这些指令与其所依赖指令之间的距离，选择距离最大的那条指令输出。距离以当前输出位置到所依赖的指令操作数之间包含的操作数目计算，例如对图 5 示例，当指令 2 可以被输出时，它到指令 1 间的距离为 1。如果指令有多个依赖，则距离取平均值。

各基本控制块整序完毕后，对指令流程图进行相应调整，使其反映出整序后的指令流，然后根据指令流程图重新组织代码就可得到最终规范形式。例如，对图 2(a)(d)(e)所示代码运用以上各节介绍的方法进行规范后可得如图 6 所示的一致规范结果。

```
    push eax
    mov eax, [esi]
    add esi, 1
    mov [edi], eax
    add edi, 1
    pop eax
    cmp [esi], ecx
    jne E
    mov eax, 1
    push eax
    mov ebx, 2
    push ebx
    call FUNC
E: ret
```

图 6　对图 2(a)(d)(e)进行规范后得到的一致规范结果

4　测试及讨论

4.1　测试

实验环境：Pentium4 1.8GHz + 256M RAM + Windows XP + IDA Pro 5.0。

实验数据：提取 Win32.Evol(从 vx.netlux.org 得到样本)中 4 个不同函数，然后根据指令乱序变形技术手工混乱它们的指令顺序，每个函数都进行 5 次独立不同的变换，最终对应每个函数获得 5 个不同变种。

表 1　测试结果

函数	大小	归一化后不同点(累计)
sub_4019FE	30	0
sub_4011C1	72	4
sub_401B84	108	2
sub_401D5C	194	7

笔者以 IDA 插件的形式实现了前面介绍的归一化方法，并在 IDA 反汇编基础上对上述变种及初始代码进行了归一化测试。将初始代码的归一化结果作为基准，然后比较各变种归一化结果与基准间的差异，结果如表 1(大小以指令计算)。

从实验结果可看出，各变种和初始代码经过归一化后，在形式上趋向一致。本实验中出现的微弱差异是由于测试中使用的输出规则不够完备引起的，以下将对此进行讨论。

4.2　讨论

对于本文介绍的归一化方案，输出规则是决定规范一致性的重要因素。以 3.3 节中列举的两条输出规则为例，如果指令缓冲池中有两条指令都可被输出，但它们的操作符相同，而且与其所依赖指令间的距离也相同，这时将无法依据输出规则唯一选择输出指令，这也是上述实验中出现微弱差异的原因。针对这个问题，可以通过完善输出规则来解决，例如对于 3.3 节中的输出规则，可再增加一条根据寻址方式选择输出指令的规则。

现实情况中，恶意代码往往会采用多种变形技术，因此在进行归一化时，需要综合采用针对不同变形技术的归一化方法，而以何种顺序调用这些方法是一个需要深入研究的问题。除本文所提常用变形技术外，一些病毒还采用寄存器替换的变形技术(如 Win95.Regswap)，对此可直接采用带通配符的特征码来应对，而无须采取相应的归一化措施。

总结　本文针对恶意代码常用的一种变形技术——指令乱序，提出了相应的归一化方案。该方案先对待测代码进行控制依赖分析，并根据分析结果将代码分成若干基本控制块，而后通过数据依赖分析构造各基本控制块的数据依赖图，并以此调整指令顺序。该方案适用于指令乱序的两种实现方式，即跳转法和非跳转法。

参 考 文 献

1　Tumer D, Entwisle S, Fossi M, et al. Symantec Internet Security Thread Report Trends for January06-June06 [R]. Volume X. Symantec Inc. 2006.9
2　Szor P, Ferrie P. Hunting for Metamorphic [C]. In 11th International Virus Bulletin Conference, 2001
3　Christodorescu M, Jha S, Seshia A, et al. Semantics-aware malware detection [C]. In: 2005 IEEE Symposium on Security and Privacy, 2005.32～46
4　Walenstein A, Mathur R, Chouchane M R, et al. Normalizing Metamorphic Malware Using Term Rewriting [C]. The 6th IEEE Workshop on Source Code Analysis and Manipulation. 2006,9:27～29
5　Christodorescu M, Kinder J, Jha S, et al. Malware Normalization [R]. #1539. Madison, Wisconsin, USA. University of Wisconsin. 2005.12
6　Bruschi D, Martignoni L, Monga M. Using Code Normalization for Fighting Self-Mutating Malware [C]. In: Proceedings of International Symposium on Secure Software Engineering, Washington DC, USA. 2006
7　Muchnick S S. Advanced Compiler Design and Implementation [M]. Morgan Kaufmann Publishers, 1997