基于可回溯动态污点分析的攻击特征生成方法

刘豫，聂眉宁，苏璞睿，冯登国
(中国科学院 软件研究所 信息安全国家重点实验室，北京 100190)

摘 要：现有黑盒或白盒的攻击特征生成方法面临样本采集困难、自动化程度较低、依赖源代码等问题。为此提出了一种基于可回溯动态污点分析的攻击特征生成方法，通过监控进程动态执行流程，提取与攻击输入相关的操作序列和约束条件，重建特征执行环境并添加判定语句，生成图灵机式的攻击特征。构造原型系统并进行测试的结果表明该方法能快速生成简洁高效的攻击特征。

关键词：恶意代码；攻击特征生成；动态污点分析；攻击路径
中图分类号： TP399 **文献标识码：**A **文章编号：**1000-436X(2012)05-0021-08

Attack signature generation by traceable dynamic taint analysis

LIU Yu, NIE Mei-ning, SU Pu-rui, FENG Deng-guo
(State Key Laboratory of Information Security, Institution of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Most of known attack signature generation systems took either black-box method or white-box method, both of which were limited in several aspects, such as costing a long time to capture sufficient samples, demanding arduous manual analysis and requiring source code of the vulnerable program. An attack signature generation method based on an innovative traceable dynamic taint analysis framework was proposed. By monitoring the vulnerable process execution, the executing trace and the constrain conditions exactly related to input data exploiting the vulnerability was extracted. Finally, by restoring the execution context and supplementing the determinant statements an executable Turing machine signature was attained. A prototype system was implemented and evaluated with different attack samples, which proved that the proposed method was able to generate accurate attack signature fast.

Key words: malware; attack signature generation; dynamic taint analysis; attack trace

1 引言

随着信息化的快速发展，入侵者的攻击目标和攻击方式都大大增加，攻击的更新速度更快，危害性更强。在各种防范系统中应用攻击特征过滤输入数据，是保护脆弱服务和主机免遭入侵的有效方法。然而传统的手动攻击特征生成依赖于人工经验，不仅耗时费力，生成特征的质量也难以保证。因此，快速高效的攻击特征生成系统成为当前的研究热点。

相关领域典型研究成果中，Nanda 等人提出的 Trag[1]系统利用动态污点分析捕获攻击过程中实际执行的指令集合，对条件表达式的变量进行后向切片[2]识别出所有攻击相关处理语句及约束条件初始化语句，自动生成图灵机式的攻击特征。但 Trag 依赖程序源码，并且生成特征规模较大，甚至和应用程序本身大小相当，导致执行效率较低。Brumley 等人[3]提出了一种使生成的特征可识别针对同一漏

收稿日期：2010-10-28；修回日期：2011-02-05
基金项目：国家自然科学基金资助项目（61073179，60703076）；中国科学院知识创新工程基金资助项目
Foundation Items: The National Natural Science Foundation of China (61073179, 60703076); Program of the Knowledge Innovation Project of Chinese Academy of Science

洞的更多可能攻击的方法。他们利用 Pin[4]记录攻击执行流程，使用程序砍片（chopping）技术[5]提取输入点到漏洞之间所有可能的执行路径，整合得到攻击特征。但文献[3]中的程序砍片需要反汇编可执行代码，容易受到软件保护手段的干扰，并且得到特征的大小按攻击路径上的分支语句数量指数增长，甚至超过程序本身大小若干个数量级。他们在后续工作中提出借助最弱前提条件解决这一问题[6]，但生成的特征大小仍然超过了程序本身。此外，Costa 等人在 Vigilante[7]中提出了一种自动过滤器生成算法，借助动态污点传播寻找含有输入数据的条件表达式，将这些表达式汇总形成过滤规则。由于没有准确识别真正引起漏洞攻击的数据，Vigilante 的过滤规则引入了与漏洞攻击无关的条件约束，导致规则过于严格，引发了漏报率较高的问题。

针对现有攻击特征生成方法的不足，本文提出了一种基于可回溯动态污点分析的攻击特征生成方法，利用动态污点分析技术对脆弱程序的漏洞利用过程进行完整的指令级记录，构建回溯分析算法提取出与攻击行为直接相关的代码执行序列，以此为基础构造特征执行上下文环境并对分支条件做出判断，从而生成图灵机式的攻击特征。对比同类研究成果，本方法无需获得程序源代码，可以准确识别出与漏洞利用直接相关的执行指令，既解决了文献[1]对条件语句中的所有变量进行后向切片所引起的特征过于庞大的问题，又排除了文献[7]中的攻击无关条件约束所造成的漏报率偏高，而与文献[3]相比，本方法更好地平衡了特征的覆盖范围和执行效率，生成特征简洁准确，生成过程快速高效。

本文的主要贡献如下。

1) 提出了一套可回溯的动态污点分析框架，可准确提取影响攻击路径的约束条件和操作攻击输入的指令序列，提升了对动态执行过程信息的管理能力，为自动化生成简洁高效的攻击特征提供了有力基础。

2) 提出了一种基于可回溯动态污点分析的攻击特征生成方法，本方法不依赖源代码，通过分析与漏洞利用相关的进程动态执行过程，可针对未知漏洞的攻击行为快速自动地生成图灵机式的攻击特征。

3) 建立了一个攻击特征生成的原型系统，通过对攻击样本的测试，验证了本文提出的方法可以不依赖源代码快速生成简洁高效的图灵机式攻击特征。

本文后续部分按如下方式进行组织：第 2 节介绍目前相关领域的研究工作；第 3 节提出了基于可回溯动态污点传播的攻击特征生成方法的基本思想和系统构成；第 4 节讨论了可回溯的动态污点分析框架的构造方法和回溯分析算法的设计细节；第 5 节说明图灵机式攻击特征生成的具体过程；第 6 节是原型系统的建立和实验评估结果；第 7 节是结束语。

2 相关工作

攻击特征的表现形式主要有正则表达式、条件约束和图灵机式 3 种[3]。正则表达式的形式简洁、匹配算法效率高，但对特征的描述能力最弱，准确性也最低。条件约束形式具有较强的描述能力，但不能等效描述循环结构，从而影响攻击特征的准确性。图灵机式攻击特征是判断输入是否为攻击的一个程序，可以对特征细节进行精确描述，因而具有较高的准确性。

攻击特征生成方法可分为黑盒法和白盒法 2 类。黑盒法[8~11]直接分析攻击样本的共性并对比与正常输入的区别，归纳聚类形成攻击特征。黑盒法面临的局限包括：攻击者容易伪造攻击输入进行混淆干扰，使得生成攻击特征的准确性降低；另外，需要耗费时间收集足量的攻击样本。白盒法[1,3,6,7]综合利用攻击输入、漏洞利用路径和漏洞自身特点等信息生成攻击特征，能更好地描述同类攻击的本质，准确性更高，因此成为当前攻击特征生成研究的主要方向。但现有白盒方法常存在依赖程序源代码、生成的特征规模庞大、执行效率较低等问题。

在基于漏洞利用的攻击特征生成的研究中，多种动态污点分析系统[12~14]常被用于记录和提取漏洞利用流程信息，并检测针对未知漏洞的攻击。文献[1]中使用 GIFT 架构[14]在编译环节向目标代码添加动态污点传播的功能代码，因此依赖于源代码。本方法采用的动态污点分析系统是基于硬件模拟器 Qemu[15]实现的，具有分析粒度精细、无需源代码等优点。与传统的关注数据依赖的污点传播不同，本文为了提取与污点相关的比较和跳转指令，污点操作指令对标志寄存器的影响也被纳入考虑。另外，还建成了一个可回溯的动态污点分析架构，提升了对污点传播过程信息的管理利用能力，可准确提取影响攻击路径的约束条件和操作攻击输入的指令序列，作为攻击特征生成的基础。

3　问题和解决思路

图灵机式的攻击特征(TMS, turing machine signature)**是对输入数据做出判断的可执行程序，它和脆弱程序、漏洞、攻击样本** 3 **方面密切相关。**

定义 1　**(图灵机式攻击特征)。对具有漏洞** V **的脆弱程序** P**，通过分析针对** V **的攻击样本** A**，可以得到一个图灵机式的攻击特征记作** TMS**。它是脆弱程序** P **的输入空间** I **到判断空间的一个映射，即** TMS:I->{ATTACK，BENIGN}**。对** P **的任意输入** $i \in$I,**，如果** i **符合攻击样本** A **的攻击特征，则有** TMS(i)=ATTACK**；否则，**TMS(i)=BENIGN**。**

本文的目标是在无法获取源代码的情况下，检测针对脆弱程序未知漏洞的成功攻击行为，生成图灵机式攻击特征。成功攻击由 2 **个方面因素决定：①脆弱程序存在可利用的漏洞；②攻击输入可以到达漏洞并触发漏洞。同类攻击输入将经由相同的路径到达程序漏洞。可见，攻击特征蕴含与漏洞攻击直接相关的输入数据的操作序列中，攻击路径、漏洞特征以及对攻击数据的条件约束是生成特征的关键信息。从中提取出攻击数据在攻击路径上的操作变化以及各步骤的约束条件，就可以验证输入数据是否为攻击。**

按此思路本文提出了一种攻击特征生成方法，通过将脆弱程序的所有输入数据标记为污点，利用可回溯的动态污点分析架构对污点数据相关操作行为构造指令级的污点传播流图（TPG**），并监控污点数据的异常使用检测未知的漏洞攻击行为。当攻击发生时，借助** TPG **的回溯分析算法，提取出与漏洞利用直接相关的污点数据操作序列和各个约束条件，恢复执行上下文并对条件分支语句添加判定，从而生成图灵机式的攻击特征。本方法的系统框架如图** 1 **所示，主要由可回溯动态污点分析、攻击检测和攻击特征生成** 3 **大模块构成，相关技术要点和实现方法将在下面** 2 **节中详细讨论。**

4　可回溯动态污点分析

准确提取脆弱进程与漏洞攻击相关的操作序列以及约束条件，是生成图灵机式攻击特征的关键。动态污点分析常用于进程行为分析，并可以检测攻击行为，因此在白盒法的特征生成中得到广泛应用。与文献[1,7]**不同，针对特征生成需求，提出了一套可回溯的动态污点分析框架，为自动化生成简洁高效的攻击特征提供了有力基础。**

4.1　基本概念和定义

定义 2　**(污点传播流图)。污点传播流图动态记录进程指令级的污点数据操作行为。由一系列的节点和边组成，可记为** *TPG<Node, Edge>***。其中，***Node*** 是** TPG **中所有节点的集合，包含了污点源和指令操作** 2 **类节点，***Node=SrcNode*∪*InsNode*；***Edge*** 是** TPG **的边的集合，均为有向边，记录了所连接节点的上下级关系。**TPG **中的节点和边的具体定义如表** 1 **所示。**

表 1　　**TPG 的节点和边的定义**

对象	定义	示例
SrcNode	污点源事件记录节点的集合	$Sn \in SrcNode$
InsNode	指令操作记录节点的集合	$In \in InsNode$
Edge	TPG 边的集合	$Ie<In_a, In_b> \in Edge$，其中，$In_a \in InsNode$, $In_b \in InsNode$，In_a 是 In_b 的上级节点

生成 TPG **的第一步是将污点数据引入系统，为此定义了污点源事件（**TSE, taint source event**）。**

定义 3　**(污点源事件)。污点源事件指将污点分**

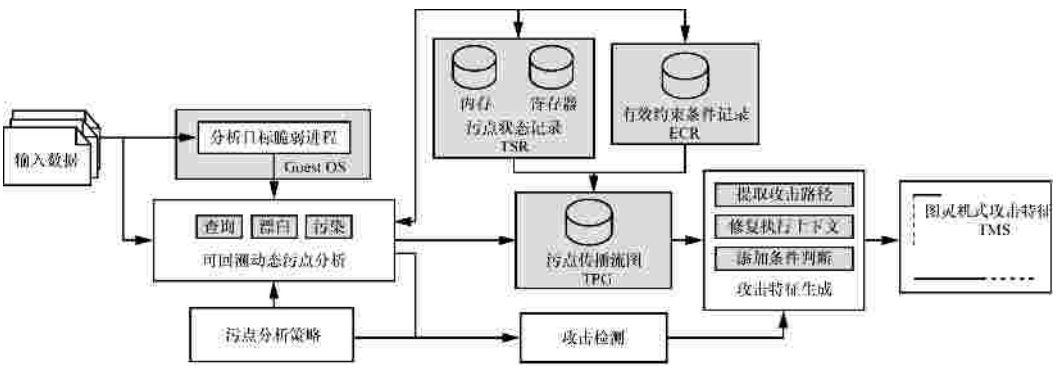

图 1　**系统框架**

析关注的外部数据引入进程空间的进程行为，记作 TSE<*Addr,Length*>，其中，*Addr* 指示外部数据的存储起始地址，*Length* 指示数据的字节数。

在特征生成系统中，污点源事件即脆弱进程接收输入数据，因此分析系统将相应的输入缓冲区标记为污点。为了实时反映污点数据在系统中的分布状况，特别定义了污点状态记录（TSR, taint state record）结构。

定义 4 (污点状态记录)。TSR 是进程执行过程中相关操作数据的污点状态实时记录，$TSR=TSR\_Reg \cup TSR\_Mem$，*TSR_Reg* 和 *TSR_Mem* 分别对应寄存器和内存数据的污点状态的集合，以字节为记录单位。*TSR* 中每条污点状态记录记作 *tsr*<*Addr, State, In*>，其中，*Addr* 指示了污点数据的地址，*State* 表明污点状态，特别地，*In* 记录该污点数据对应的 TPG 节点。

为了构造 TPG，借助 TSR 建立了 2 个映射关系 isTaint 和 whichInsNode，如表 2 所示。借助它们，定义了 TPG 中需要记录的污点相关指令操作 TI。

定义 5 (污点相关指令操作)。对目标进程执行的指令 Ins<Type,*Operand*>，其中，Type 表示 Ins 的指令类型，*Operand* 是 Ins 的操作数的集合。如果 $\exists op \in Operand$ 且 isTaint(op)=1，则 Ins 是污点相关指令操作。特别地，对于条件跳转指令 JC，判断标志寄存器 Flag_Reg，如果 isTaint(Flag_Reg)=1，则 JC 是 TI。

最后，定义有效约束条件记录（ECR, effective condition record），用于提取影响攻击路径的约束条件。

定义 6 (有效约束条件记录)。ECR 是影响当前运行指令的条件约束的集合，ECR 的记录表示为二元组 *ecr*<*In, Effect_Range*>，其中，*In* 指示该约束条件的跳转指令 JC 对应的 TPG 节点，*Effect_Range* 表示 *ecr* 的作用范围，*Effect_Range*= [min(Addr1, Addr2),max(Addr1, Addr2)]，其中，Addr1 是 JC 的 EIP，Addr2 是 JC 的跳转地址。

本文对每个属于 TI 的条件跳转指令生成一条 ecr 记录。利用 ECR 本文定义映射关系 whichJCNode，如表 2 所示。为确保 ECR 实时更新，如果执行到 eip 不属于 ecr 的 Effect_Range 指令 Ins 时，则从 ECR 中删除 ecr 记录，除非 Ins 属于 Effect_Range 内的指令调用的子程序。

上述方法确保特征生成不受与攻击无关的条件约束的干扰，但存在一种局限。如图 2 所示，本文可以正确识别图 2(a)和图 2(b)类条件约束的影响范围；但对图 2(c)类的条件约束，则不能识别出(Addr2,Addr3)同样受 JC 的影响。这由动态分析本身的制约引起：如果 JMP Addr3 指令不被执行，本文就无法得到代码结构的全貌。这一局限可以结合静态分析进行弥补。

4.2 构造 TPG

构造 TPG 的基本思路是，对每一个污点相关操作，生成一个节点对其进行记录，并将其连接到处理同源的污点数据的上一次污点相关操作所对应的节点，或者影响该操作的约束条件对应的节点，如此往复，从而生成 TPG。具体方法如下。

1) 对污点源事件 TSE_a<Addr,Length>

① 生成一个污点源节点 Sn_a 记录相关信息，Sn_a 是 TPG 的初始节点。

② 然后，生成个数为 *Length* 的污点状态记录 *tsr*<*Addr*,*State*,*In*>，记录 TSE_a 引入的污点数据的状态信息，并且初始化每个 $tsr \rightarrow In=Sn_a$。

2) 对于 TI 操作 Ins_a<Type,Operand>

① 首先，生成 TPG 节点 In_a 记录指令操作的相关信息。

表 2 TSR 和 ECR 相关的映射关系

对象	定义	意义	示例
isTaint	isTaint: Reg ∪ Mem→{0,1}	查询地址的污点状态	∀ addr∈Reg ∪ Mem，如果 addr 存储的数据是污点，则 isTaint(addr)=1，否则 isTaint(addr)=0
whichInsNode	whichInsNode: Reg ∪ Mem→InsNode ∪ SrcNode	查询地址对应的 TPG 节点	∀ addr∈Reg ∪ Mem，如果 isTaint(addr)=1，则 whichInsNode(addr)=In，IN∈InsNode
whichJCNode	WhichJCNode:EIP → p(InsNode)，P(InsNode)为 InsNode 的幂集	查询影响当前执行指令的所有约束条件	指令 Ins 的地址为，如果 ecr∈ECR ∃ ecr∈ECR，且 eip∈ecr→Effect_Range，则 ecr→In∈whichJCNode(eip)

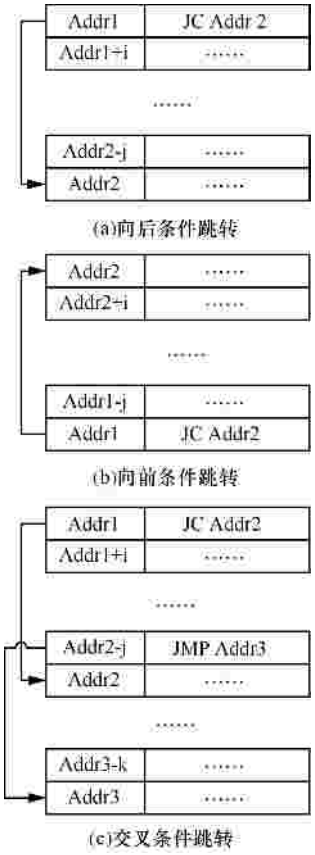

图 2　不同类型的条件跳转指令

② 若 Ins_a 为普通指令，根据 Type 分析 *Operand* 中影响 Ins_a 执行结果的操作数的集合 $Operand_{src}$，遍历 $op_i \in Operand_{src}$，如果 $isTaint(op_i)=1$，在 TSR 中查询 $whichInsNode(op_i)=In_i$，得到 op_i 对应的 TPG 节点 In_i，生成边 $Ie<In_i, In_a>$。

③ 如果标志寄存器 FlagReg 受污点数据 op_i 运算的影响而改变，将 tsr_FlagReg 记为污点，并修改 $tsr\_FlagReg \rightarrow In=whichInsNode(op_i)$。

④ 若 Ins_a 为条件跳转指令，则查询 TSR 得到 $whichInsNode(FlagReg)=In_k$，并生成边 $Ie<In_k, In_a>$。

⑤ 根据 Ins_a 的 EIP，对每个 $In_j \in whichJCNode(EIP)$，生成边 $Ie<In_j, In_a>$。

⑥ 根据 Type 分析 Operand 中受 Ins_a 执行影响的操作数的集合 $Operand_{dst}$。遍历 $op_j \in Operand_{dst}$，如果 Ins_a 执行后 $isTaint(op_j)=1$，更新 op_j 对应的污点状态记录 $tsr_j \rightarrow In= In_a$。

通过如上步骤，按指令的污点数据来源和所受条件约束影响完成了 TPG 的构建。

4.3　回溯算法

由于 TPG 中记录了节点间的上下级关系，因此从任意节点出发可以有向遍历 TPG，得到的 TPG 子图对应相关污点数据的操作序列以及影响操作的约束条件。要获取从节点 N_a 开始的回溯分析结果，则对 N_a 在 TPG 图中的每个上级节点进行递归的回溯分析。当节点的所有上级节点的回溯分析完成或者该节点已经没有上级节点时，将它加入回溯分析记录链表。最后，当节点 N_a 的回溯分析返回时，回溯分析记录链表中就得到了从污点源开始到 N_a 的所有相关的脆弱进程执行指令的完整记录。回溯分析的算法描述如图 3 所示。

```
Input：  回溯起始节点 Node，回溯记录链表 TraceBackList
Output： 回溯记录链表 TraceBackList

TraceBack(Node, TraceBackList)
{
    CurrentUpNodes = getUpNodes(Node)//获取 Node 的上级节点链表
while(CurrentUpNodes == false)//上级节点遍历完成，结束循环
      {
          if(CurrentUpNodes->Node ∈ SrcNode)//上级节点是污点源节点
          {
          addTraceBackList(TraceBackList,CurrentUpNodes-> Node);
           CurrentUpNodes=CurrentUpNodes->next;
             continue;
          }
          TraceBack(CurrentUpNodes->Node, TraceBackList);//递归回溯分析上级节点
CurrentUpNodes=CurrentUpNodes->next;
      }
      addTraceBackList (TraceBackList, Node);
     return;
}
```

图 3　TPG 回溯分析算法

4.4　攻击检测

根据污点分析策略，当脆弱程序执行可能引发漏洞利用的指令，如 JMP、CALL、RET 时，动态污点分析监控污点数据的异常使用，从而检测到针对未知漏洞的攻击。污点分析策略设置了 2 条常用的异常使用判断规则：污点数据用作跳转地址和污点数据作为执行代码。文献[16]的研究表明，利用这 2 条规则，可以覆盖大多数漏洞攻击行为，并且误报率很低。当检测到对漏洞的攻击时，则使用回溯分析算法，从 TPG 中回溯提取与漏洞攻击直接相关的污点数据操作序列以及影响攻击路径的约束条件。

5　图灵机式攻击特征生成

动态污点分析检测到对未知漏洞的攻击，并回溯 TPG 获得与攻击直接相关的污点数据操作指令

序列，记作 TOT，这是图灵机式的攻击特征 TMS 生成的基础。但 TOT 并不等同于 TMS：一方面 TOT 是脆弱进程动态执行的操作序列，指令操作数的寄存器和内存地址都处在脆弱进程的执行上下文中，不能直接移植到 TMS 的执行环境；另一方面，TOT 中缺少对条件分支语句的处理，没有判断输入数据分组是否为攻击。为此，需要在 TOT 的基础上对上述 2 方面进行修改和完善，才能生成图灵机式的攻击特征 TMS。

5.1 特征执行上下文恢复

由于提取 TOT 时的进程执行上下文与 TMS 的运行环境有较大差异，因此需要根据指令操作数的类型和污点状态进行处理，使其能在 TMS 的运行环境中顺利执行。为此，TMS 需要提供一个缓冲区 SRC_BUFFER 存放输入数据分组，并将 TOT 中读取污点源数据的指令寻址映射到 SRC_BUFFER 的内存区间。另外，TMS 还需要提供一个数据段 *TMS_DATA*，存储污点数据相关操作的变量，避免 TOT 中的寻址地址和 TMS 发生冲突。然后，按如下的规则处理 TOT 中的污点数据相关操作指令。

1) 对于操作数本身是污点数据，并且是寄存器类型的，无需进行特别处理，因为污点数据的值将随着 TMS 对输入数据的处理自动获得。

2) 对于本身不是污点数据而且不会被污点数据污染的操作数，则需要将它的实际值作为立即数替换它在 TOT 中的类型。无论该操作数是一个寄存器还是内存寻址地址，动态污点分析系统都能够获取攻击发生时它的实际值，并记录在 TPG 的对应节点中。

3) 对于本身不是污点数据，但通过指令执行被污点操作数污染的操作数，如果类型是寄存器，无需进行处理；如果是内存寻址地址，用 ADDR1 表示，则将 ADDR1 替换为 *TMS_DATA* 的一个空闲地址 ADDR2，并将后面指令中出现的 ADDR1 都使用 ADDR2 进行替换。如果该操作数原有的值会影响操作结果，还需要提取出它的实际值代入污染指令的计算。

使用处理规则 2）是因为 TOT 中只含有污点数据相关的操作，没有非污点数据的变量初始化过程。如果在 TMS 中直接使用非污点数据在攻击发生时的内存地址和寄存器，将会引起系统崩溃或者读入错误的值，本文解决方法是将攻击发生时的实际值代入它们在TMS的相关操作。使用处理规则 3）是因为 TOT 中不含经过污点传播成为污点数据的变量的初始化过程，直接使用攻击发生时的内存地址也可能和 TMS 的内存空间发生冲突。但与非污点数据不同，污点数据变量最终将获得来源于输入数据分组的赋值，因此只需要在 TMS 提供的 *TMS_DATA* 中为其分配一个合理的存储空间即可。

5.2 判定结果

TOT 的指令序列经过上述的修正，已经满足在 TMS 的环境中执行的需求，但还缺少对输入数据是否为攻击的判断输出。因此，需要处理 TOT 中的条件分支语句，添加判断输出语句。在动态污点分析系统处理属于 TI 的条件分支跳转指令 JC 时，记录 JC 运行时的条件满足情况，据此对 TOT 中对应的条件分支语句进行处理，使得与当时条件满足情况一致时，沿着攻击路径执行，否则攻击特征返回 BENIGN。在 TOT 的最后一个条件分支跳转指令，满足当时条件的分支，可以直接返回 ATTACK 判断，因为之后已经没有条件分支语句，输入数据必然能够到达漏洞并满足漏洞利用条件，形成有效攻击。通过对 TOT 进行上述的修改和完善，最终重新编译为可执行程序，生成了图灵机式攻击特征。

6 评估测试

6.1 原型系统构建

本文基于 WooKon 恶意代码分析平台[17]开发实现了生成图灵机式攻击特征的原型系统。将实验样本部署在 WooKon 平台上运行的 Windows XP 系统中，利用第三方的反汇编引擎 Udis86[18]对样本执行的每条指令进行解析，实现了对指令类型和操作数的识别，并可以获取操作数的寄存器或内存地址以及它们的值，从而按照预设的污点分析策略进行指令级的动态污点分析。原型系统运行在一台 DELL Optiplex 360 主机上，基本软硬件运行环境如下。内存：DDR2 667MHz 2GB；硬盘：SATA 250GB；主板：Intel G31；主机操作系统：Fedora Core 10 x86_64；虚拟操作系统：Windows XP SP2。

6.2 实验结果和讨论

本文使用 2 个脆弱程序 IrfanView_3.99 和 easyftp-server-1.7.0.11-cn 及其攻击样本进行了实验。IrfanView_3.99 是一个图像查看和格式转换程序，它的插件 Formats.dll 中存在一个本地溢出攻击漏洞，使得读入特别构造的.iff 文件时，引起进程崩溃。easyFtp-server 是一个支持 Web 访问的 FTP 服

务器，在处理超长参数时存在溢出漏洞，攻击者可利用其漏洞远程执行任意指令。本文利用原型系统分析了上述 2 种脆弱程序的漏洞利用过程，成功生成了攻击样本所对应的图灵机式的攻击特征。表 3 记录了特征生成过程和最终生成特征的基本信息。

表 3　攻击特征生成实验结果

生成攻击特征	IrfanView_TMS	EasyFTP_TMS
TPG 节点总数	11 413	3 756
有效节点数量	2 468	253
分析耗时/s	48	37
生成特征大小/byte	6 656	7 680
脆弱程序大小/KB	434	188
特征占程序大小	1.50%	3.99%

从实验结果可以看出本文得到攻击特征都非常简洁，不到脆弱程序自身大小的 4%，相对于文献[1,3]的结果具有明显优势。原因在于恢复特征的执行上下文环节，文献[1,3]中使用了切片方法，而本文直接使用攻击发生时的实际值替代非污点数据，从而省略了它们的初始化过程，并避免了其扩散影响。事实上，非污点数据表示进程执行环境中的其他资源对输入的影响；使用攻击发生时的实际值，等效于模拟了特征生成时的脆弱进程以及部署环境。如果部署环境具有一般性，那么图灵机式攻击特征的判断就应该被接受，因为丢弃一个可能攻击典型应用部署的数据分组是符合逻辑的。

为了验证生成特征的有效性，本文利用 Fuzz 测试开发组件 SPIKE[19]，针对 2 种脆弱程序生成了多个灰盒测试样本，然后使用攻击特征对这些样本进行判断，并将所有被判断为攻击数据的测试样本输入脆弱程序进行实际验证。对于被攻击特征判断为正常的测试样本，由于数量太大，本文从中分别随机抽取了 1 000 个样本输入脆弱程序进行验证。得到的实验结果如表 4 所示。可见生成特征的判断结果和脆弱程序实际验证的结果非常接近，说明本文生成的特征具有较好的准确性。进一步分析 IrfanView 和 EasyFTP 的攻击特征准确性具有差异的原因。结合表 3，可知在 IrfanView 漏洞攻击过程中获得了大量的用于特征生成的有效节点，而 EasyFTP 的攻击样本提供的有效节点数量较少，导致生成特征的刻画能力较弱，准确性受到影响。如果攻击数据在漏洞发生前参与运算较少，本文方法会受到一些制约。但在攻击数据频繁参与运算并且相互关联制约形成重要的约束条件的情况下，本文方法可以做出非常准确的判断，而这正是手工分析最为困难的场景。

表 4　攻击特征有效性验证

攻击特征	IrfanView_TMS	EasyFTP_TMS
Fuzz 测试样本数量	4 421	17 452
特征判定攻击样本数	27	194
验证攻击样本数	27	191
误报率	0	1.55%
抽取特征判定正常样本数	1 000	1 000
验证正常样本数	1 000	996
漏报率	0	0.4%

最后，在对 Fuzz 生成样本的测试过程中，本文在表 5 中特别记录了图灵机式攻击特征的运行性能。可见，生成的攻击特征只占用很少的系统资源，并且做出判断的速度很快。相比于普通硬盘平均寻道时间 8ms 和宽带网络 100ms 左右的正常延迟，攻击特征检查输入数据分组对应用程序的性能影响非常微小。

表 5　攻击特征运行性能

攻击特征	占用内存大小/KB	判断单个样本平均耗时/ms
IrfanView_TMS	1 544	0.306
EasyFTP_TMS	1 316	0.261

以上的实验验证了本文想出的特征生成方法既具有较高的准确性，同时也有很好的执行效率。在一个应用程序有多个漏洞或者同一漏洞具有不同的利用方式时，本系统将针对不同的攻击利用生成多个图灵机式的特征，从而需要对一个数据分组进行多次的检验，可能引起额外的开销。如何合并不同的攻击特征以减少重复计算，将是笔者下一步工作需要解决的问题。

7　结束语

本文提出一种基于可回溯动态污点分析的攻击特征生成方法，借助动态污点分析构建脆弱程序漏洞利用过程的 TPG，提取出与攻击行为直接相关的代码执行序列，以此为基础恢复特征执行上下文环境并对条件分支做出判断，从而生成图灵机式的攻击特征。通过对原型系统的评估测试，验证了本方法具有不依赖源代码，生成特征简洁高效，生成过程自动化程度高的优点。下一步，将研究如何扩展现有方法以实现多进程应用系统的攻击特征生

成，并探索合并不同的攻击特征的方法，提高多特征情况下对输入数据做出判断的效率。

参考文献：

[1] NANDA S, CHIUEH T. Execution trace-driven automated attack signature generation[A]. 24th Annual Computer Security Applications Conference[C]. USA, 2008. Anaheim(CA), 2008.195-204.

[2] KOREL B, LASKI J. Dynamic Program Slicing[M]. Inf, Process Lett, 1988, 29(3):155-163.

[3] BRUMLEY D, NEWSOME J, SONG. Towards automatic generation of vulnerability-based signatures[A]. Proceedings of the 2006 IEEE Symposium on Security and Privacy[C]. Oakland, Virginia, USA, 2006.15-16.

[4] LUK C, COHN R, MUTH R. Pin: building customized program analysis tools with dynamic instrumentation[A]. Proc of 2005 Programming Language Design and Implementation (PLDI) Conference[C]. Chicago, USA, 2005. 190-200.

[5] JACKSON D, ROLLINS E. Chopping: a generalization of slicing[A]. Proc of the Second ACM SIGSOFT Symposium on the Foundations of Software Engineering[C]. New Orleans, USA, 1994.

[6] BRUMLEY D, WANG H, JHA S. Creating vulnerability signatures using weakest preconditions[A]. IEEE Computer Security Foundations Symposium[C]. Venice, Italy, 2007. 311-325.

[7] COSTA M, CROWCROFT J, CASTRO M. Vigilante: end-to-end containment of internet worms[J]. SIGOPS Oper Syst Rev, 2005, 39(5):133-147.

[8] LIANG Z, SEKAR R. Automatic generation of buffer over?ow signatures: an approach based on program behavior models[A]. 21st Annual Computer Security Applications Conference[C]. Tucson, Arizona, USA, 2005.10-224.

[9] LIANG Z, SEKAR R. Fast and automated generation of attack signatures: a basis for building self-protecting servers[A]. Proceedings of the 12th ACM Conference on Computer and Communications Security, Alexandria(VA)[C]. USA, 2005.

[10] NEWSOME J, KARP B, SONG D. Polygraph: automatically generating signatures for polymorphic worms[A]. IEEE Symposium on Security and Privacy[C]. Oakland, Virginia, USA, 2005. 226-241.

[11] SINGH S, ESTAN C, VARGHESE G. Automated worm ?ngerprinting[A]. Symposium on Opearting Systems Design & Implementation[C]. San Francisco, USA, 2004. 45-60.

[12] VASUDEYAN A, YERRABALLI R. Cobra: fine-grained malware analysis using stealth localized-executions[A], Proceedings of the 2006 IEEE Symposium on Security and Privacy[C]. Oakland, Virginia, USA, 2006. 264-279.

[13] CLAUSE J, LI W, ORSO A. Dytan: a generic dynamic taint analysis framework[A]. Proceedings of the 2007 International Symposium on Software Testing and Analysis, London, UK, 2007.196-206.

[14] LAM L, CHIUEH T. A general dynamic information flow tracking framework for security applications[A]. 22nd Annual Computer Security Applications Conference[C]. Miami, USA, 2006.463-472.

[15] QEMU[EB/OL]. http://www.nongnu.org/qemu/.

[16] NEWSOME J, SONG D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[A]. 12th Network and Distributed System Security Symposium. San Diego, California, February 2005.

[17] WooKon **恶意代码分析平台**[EB/OL]. http://www.isec.ac.cn/group_detail.jsp? gid=2&id=62.
WooKon mulicicus code analysis platform[EB/OL].http://www.isec.ac. cn/ group _detail.jsp? gid=2&id=62.

[18] Udis86[EB/OL]. http://udis86.sourceforge.net/.

[19] SPIKE Fuzz **测试开发组件**[EB/OL]. http://www.immunitysec.com/downloads/SPIKE2.9.tgz.
SPIKE Fuzz testing kit[EB/OL]. http://www.immunitysec.com/downloads/SPIKE2.9.tgz.

作者简介：

刘豫（1983-），男，四川眉山人，中国科学院博士生，主要研究方向为网络信息安全与恶意代码分析。

聂眉宁（1987-），男，四川乐山人，中国科学院硕士生，主要研究方向为网络信息安全与恶意代码分析。

苏璞睿（1976-），男，湖北宜昌人，博士，中国科学院副研究员，主要研究方向为恶意代码分析与防范。

冯登国（1965-），男，陕西靖边人，博士，中国科学院研究员、博士生导师，主要研究方向为密码学与信息安全。