基于动态污点分析的恶意代码通信协议逆向分析方法

刘 豫,王明华,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: 对恶意代码通信协议的逆向分析是多种网络安全应用的重要基础.针对现有方法在协议语法结构划分的完整性和准确性方面存在不足,对协议字段的语义理解尤为薄弱,提出了一种基于动态污点分析的协议逆向分析方法,通过构建恶意进程指令级和函数级行为的扩展污点传播流图(Extended Taint Propagation Graph, ETPG),完成对协议数据的语法划分和语义理解.通过实现原型系统并使用恶意代码样本进行测试,结果表明本方法可以实现有效的语法和语义分析,具有较高的准确性和可靠性.

关键词: 恶意代码; 协议逆向分析; 动态污点分析
中图分类号: TP302.7 **文献标识码:** A **文章编号:** 0372-2112 (2012) 04-0661-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.04.007

Communication Protocol Reverse Engineering of Malware Using Dynamic Taint Analysis

LIU Yu, WANG Ming-hua, SU Pu-rui, FENG Deng-Guo
(*State Key Laboratory of Information Security, Institution of Software, Chinese Academy of Sciences, Beijing* 100190, *China*)

Abstract: Communication protocol reverse engineering of malwares is significant base for various network security applications. However, recent works have limited accuracy and integrity in identifying protocol fields and are especially weak in understanding fields' semantics. This paper proposed a method for communication protocol reverse engineering based on dynamic taint analysis. By building an extended taint propagation graph (ETPG) recording both instruction and function level behaviors of a malicious process, dividing the protocol data into different syntax fields and inducing the semantic information of individual fields were achieved. A prototype system was implemented and evaluated with malware samples. The results show that this method can divide the syntax fields and extract semantic information accurately and effectively.

Key words: malware; protocol reverse engineering; dynamic taint analysis

1 引言

通信协议逆向分析在恶意代码机理分析、漏洞发掘[1]、入侵检测[2]、网络监控以及根据协议特征的流量过滤[3]等安全应用中发挥着重要的作用.目前恶意代码广泛采用流量加密、代码混淆、动态生成等技术来对抗分析,使已有通信协议逆向分析方法面临新的挑战.随着网络恶意代码更新不断加快,传统的手动分析方法已经难以跟上变种产生的速度,自动化的分析方法成为当前研究的主要方向.

在典型的研究成果中,Discoverer[4]借鉴了生物信息学的序列比对算法[5]提取消息样本中的协议格式信息,分析速度快,自动化程度高.但它单纯分析消息样本而忽略了恶意进程,因而存在不能处理加密通信、需要收集大量样本、无法提供语义信息等局限.Dawn Song 等人提出的 Polyglot[6]通过分析与协议数据相关的进程操作序列的特征来识别协议语法元素,具有较高的准确性.但它只采集了指令级的操作序列,难以从中获得字段的语义信息;此外,该方法不能对数据字段的内部结构作进一步解析.Lin 等人提出的 AutoFormat[7]根据消息数据被处理时所对应的函数调用栈和执行指令 EIP 等执行环境信息进行聚类实现协议字段的识别,可以得到协议字段间的关系,但也有相当的局限:不能处理使用了代码混淆技术的恶意代码,也不能提供字段的语义信息,甚至无法给出字段的语法属性.

针对当前分析方法的不足,本文提出了一种基于动

收稿日期:2011-02-28;修回日期:2011-04-08

态污点分析的协议逆向方法.通过构建恶意进程指令级和函数级行为的扩展污点传播流图(Extended Taint Propagation Graph,ETPG),分析进程对各种协议元素的不同处理过程,实现对协议数据的语法结构划分;并根据处理相关字段的特征 API 函数所蕴含的语义,推导得到字段的语义信息.我们实现了原型系统,并使用恶意代码样本进行实验,结果表明我们的方法实现快速有效的语法划分和语义理解.与[4,6,7]比较,我们的方法不仅能实现准确的语法划分,并且可以提供各个字段的语义,从而更好的为多种网络安全应用提供通信协议规范方面的支持.

2 相关工作

通信协议逆向分析可分为三个层次:协议语法结构划分,协议字段语义理解和协议状态机构建.其中,在没有源代码的逆向分析条件下构造完整的协议状态机非常困难[8],实现通信协议的语法划分和语义理解是目前主要的研究目标.

除了传统的手动分析,自动化的通信协议逆向分析方法可分为基于网络数据包和基于进程行为两类.基于网络数据包的方法如[4,9],通过比对同类消息样本间的字符序列异同推断协议格式信息.这类方法分析速度快,自动化程度也较高,但只从消息样本出发而忽略了恶意进程行为蕴含的信息,因此无法处理加密通信和提供字段语义信息,同时必须收集到大量样本.基于进程行为的方法[6,7,10]通过识别进程处理网络消息的行为特征分析协议格式.由于综合利用了样本和进程执行两方面的信息,因此结果更加准确可靠,并可以分析加密通信.但现有成果在语法要素识别的准确性和完整性方面还存在不足,特别是对协议字段的语义理解尤为薄弱.究其原因,进程处理语法元素的流程相对固定,行为特征较为明显,因此较易识别;而字段的语义则因实现功能的不同而大相径庭,从而给理解带来障碍.

本文提出的方法与[6,10]有类似之处,都采用了动态污点分析[11]获取进程行为,但他们的工作中动态污点分析仅提供了指令级的污点操作序列,难以从中获得相关字段的准确语义.我们通过污点分析构建包括指令级和函数级行为的 ETPG,获取了进程动态执行过程的更多细节,不仅使协议语法分析的结果更加精确,还能提供字段的数据类型和语义信息,弥补了现有工作的不足.

3 问题和解决思路

首先,我们定义协议逆向分析所解决的具体问题.

定义 1 (协议逆向分析).对于一条长度为 L 的消息 M,对它进行协议逆向分析是指将 M 还原为对应协议规范的 n 个字段,即 $M = \bigcup_{i=1}^{n} F_i$,$F_i$ 是标识字段的五元组,$F_i = < \text{range}_i, \text{syntax}_i, \text{type}_i, \text{value}_i, \text{semantic}_i >$.其中:

1. range_i 是构成字段 F_i 的各个字节在消息 M 的偏移值的集合,指示 F_i 在 M 中的位置,F_i 的长度记为 $|\text{range}_i|$;

2. syntax_i 指示 F_i 的语法类型,$\text{syntax}_i \in \{\text{Keyword}, \text{Separator}, \text{Direction}, \text{Data}\}$;

3. type_i 是 F_i 的数据类型,取值可以是 int,char 或 string 等;

4. value_i 是 F_i 的值;

5. semantic_i 指示了 F_i 的语义,是一个字符串类型的可理解的语句.

另外,分析结果需要满足如下两条基本属性.

1. 确定性. $M = \bigcup_{i=1}^{n} F_i$,则 $\forall\, i \neq j, 1 \leqslant i, j \leqslant n$, $\text{range}_i \cap \text{range}_j = \varnothing$.

2. 完整性. 消息 M 的长度为 L, $M = \bigcup_{i=1}^{n} F_i$,则 $\sum_{i=1}^{n} |\text{range}_j| = L$.

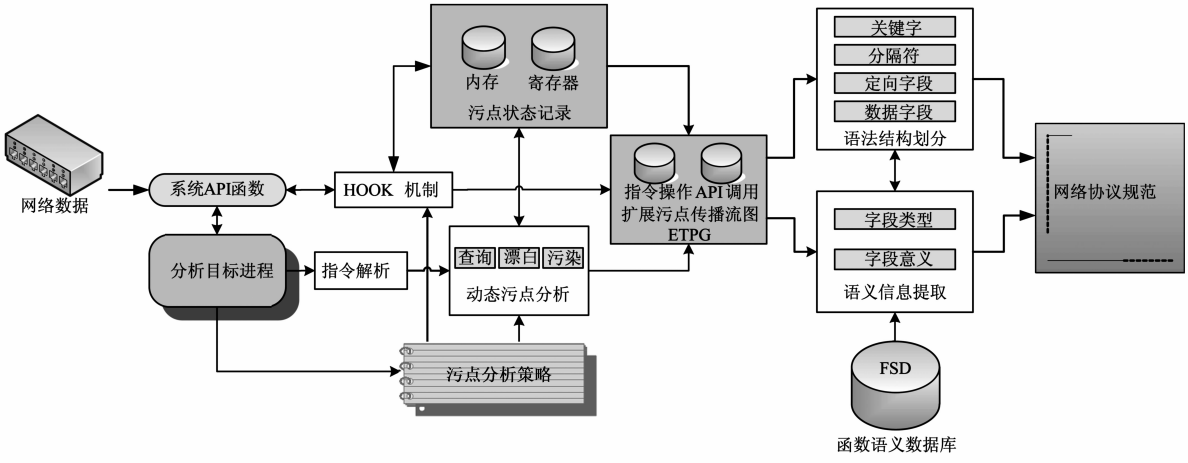


图1 系统架构图

我们的基本思路是恶意进程对消息的处理过程蕴含着通信协议的语法结构和语义信息.具体而言,消息中不同语法要素的进程处理流程表现出不同的特征,据此可以划分协议字段结构;借助各字段对应的处理函数的语义,可以推导字段语义.为了获得协议分析所需要的进程动态执行细节,我们采用了动态污点分析方法.将消息标记为污点,可以排除与消息处理无关的进程其他行为的干扰,建立消息数据和行为之间的直接关联,提高分析效率.本方法的系统架构如图 1 所示,由进程动态执行信息提取、语法结构划分和语义信息提取三大功能模块组成.

4　进程动态执行信息提取

对基于进程行为的协议逆向分析方法而言,获取真实可靠的进程动态执行信息是重要的基础环节.为此,我们特别提出了一套针对通信协议逆向的动态污点分析框架,构建进程指令级和函数级的 ETPG,为协议逆向分析提供有力的基础.

4.1　基本概念与定义

定义 2　(扩展污点传播流图,ETPG).污点传播流图 ETPG,记录进程操作污点数据的指令级和函数级行为,由一系列的节点和边组成,可记为 ETPG < Node, Edge > .其中,Node 是 ETPG 中所有节点的集合,包含了污点源、指令操作和函数调用三类节点,Node = SrcNode ∪ InsNode ∪ APINode;Edge 是 ETPG 的边的集合,均为有向边,记录了所连接节点的上下级关系.ETPG 中的节点和边的具体定义如表 1 所示.

表 1　ETPG 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∈SrcNode
InsNode	指令操作记录节点的集合	In∈InsNode
APINode	API 调用记录节点的集合	An∈APINode
Edge	ETPG 的边的集合	Ie〈In_a, In_b〉∈ Edge,其中 In_a ∈ InsNode, In_b ∈ InsNode, In_a 是 In_b 的上级节点.

生成 ETPG 的第一步是将污点数据引入系统,为此我们定义了污点源事件(Taint Source Event,TSE).

定义 3　(污点源事件,TSE).污点源事件指将污点分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.

污点源事件在进程空间中产生了污点数据,为了实时反映污点数据在系统中的分布状况,支持动态污点分析并生成 ETPG,我们特别定义了如下的污点状态记录结构(Taint State Record),记为 TSR.

定义 4　(污点状态记录,TSR).污点状态记录 TSR 是进程执行过程中操作数据的污点状态实时记录的集合,TSR = TSR_Reg ∪ TSR_Mem, TSR_Reg 和 TSR_Meg 分别对应寄存器和内存数据的污点状态记录集合,以字节为最小记录单位.TSR 中的每一条污点状态记录记作 tsr < Addr, State, In, An, Offset > ,其中 Addr 指示了污点数据的地址,State 表明污点状态,特别地,In 和 An 记录该污点数据对应的 ETPG 中的节点,Offset 记录污点数据对应在污点源中的偏移值的集合.

为了构造 ETPG,我们借助 TSR 建立了四个映射关系 isTaint、whichInsNode、whichAPINode 和 OffsetInSource.它们的定义如表 2 所示.利用它们,我们定义了污点相关的指令操作 TI 和函数操作 TAPI.

定义 5　(污点相关指令操作,TI).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 ∃op ∈ 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.

定义 6　(污点相关函数操作,TAPI).对目标进程调用的系统 API 函数 API〈Name, Parameter〉,其中 Name 表示 API 的名称,Parameter 是 API 的参数的集合.如果 ∃para∈ Parameter 且 isTaint(para) = 1,则 API 是污点相关函数操作.

表 2　TSR 相关的映射关系的定义

对象	意义	定义	示例
isTaint	查询地址的污点状态	isTaint Reg ∪ Mem → {0,1}	∀addr∈ Reg ∪ Mem,如果 addr 存储的数据是污点,则 isTaint(addr) = 1,否则 isTaint(addr) = 0.
whichInsNode	查询地址对应 ETPG 中的指令级节点	whichInsNode: Reg ∪ Mem → InsNode ∪ SrcNode	∀addr∈ Reg ∪ Mem, 如果 isTaint(addr) = 1, 则 whichInsNode(addr) = In, In∈ InsNode.
whichAPINode	查询地址对应 ETPG 中的函数级节点	whichAPINode: Reg ∪ Mem → APINode ∪ SrcNode	∀addr∈ Reg ∪ Mem, 如果 isTaint(addr) = 1, 则 whichAPINode(addr) = An, An∈ APINode.
OffsetInSource	查询地址对应污点源数据的偏移值	OffsetInSource: Reg ∪ Mem → P(N), 是自然数集 N 的幂集	∀addr∈ Reg ∪ Mem,如果 isTaint(addr) = 1,则 OffSetInSource(addr) = Off_Set, Off_Set∈ P(N).

4.2　生成 ETPG

生成 ETPG 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,如此往复,从而生成 ETPG.具体方法如下:

1.对污点源事件 TSE_a < Addr, Lenght > :

(a)生成一个污点源节点 Sn_a 记录它的相关信息,Sn_a 是 ETPG 的初始节点;

(b)然后,生成个数为 Length 的污点状态记录 tsr < Addr, State, In, An, Offset > ,记录 TSE_a 引入的污点数据的状态信息,并且初始化每个,tsr→In = Sn_a,tsr→An = Sn_a.

2.对于 TI 操作 Ins_a < Type, Operand > :

(a)生成一个 InsNode 节点 In_a 记录它的相关信息;

(b)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 ETPG 中的节点 In_i,并生成边 Ie < In_i, In_a > ;

(c)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→In = whichinsNode(op_i);

(d)若 Ins_a 为条件跳转指令,则查询 TSR 得到 whichInsNode(Flag_Reg) = In_k,并生成边 Ie < In_k, In_a > ;

(e)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 →In = In_a.

3.对于 TAPI 操作 API_a < Name, Parameter > :

(a)生成一个 APINode 节点 An_a 记录它的相关信息;

(b)根据 Name 分析 Parameter 中影响 API_a 执行结果的参数的集合 $Parameter_{src}$,遍历 $para_i \in Parameter_{src}$,如果 isTaint($para_i$) = 1,在 TSR 中查询 whichInsNode($para_i$) = An_i,得到 $para_i$ 对应的 ETPG 中的节点 An_i,并生成边 Ae < An_i, An_a > ;

(c)根据 Name 分析 Parameter 中受 API_a 执行影响的参数的集合 $Parameter_{dst}$. $\forall para_j \in Parameter_{dst}$,如果 API_a 执行后 Taint($para_j$) = 1,我们更新 $para_j$ 对应的污点状态记录 tsr_j→An = An_g.

随着目标进程对消息的处理,按上述的方法即可生成指令级和函数级结合的 ETPG.ETPG 中的边指示了各节点的污点数据的来源关系,从各节点出发可以方便的遍历 ETPG,得到的子图即为与该节点的污点数据直接相关的进程行为.

为了满足协议逆向分析的需要,InsNode 和 APINode 节点中记录的具体信息如表 3 所示.其中,Offset 信息具有特殊的重要性.因为 ETPG 节点中的污点数据可能是由消息数据通过若干操作得到的,并不直接对应污点源中的原始消息.在 ETPG 的节点记录 Offset 信息,可以在识别出语法元素后,直接对应得到消息数据中的语法字段结构.为此,我们在污点状态记录结构 TSR 中加入了 Offset 属性,污点分析时可以查询 TSR 得到污点数据对应的 Offset,并对 TI 指令执行后各污点数据对应的 Offset 进行修改,以维护 TSR 中的完整和准确.

表 3 Ins Node 和 API Node 节点记录的信息

InsNode 节点记录的信息		APINode 节点记录的信息	
EIP	指令 EIP	Name	函数名称
Type	指令类型	Parameter	函数参数的地址和具体的值
Operand	操作数的类型和地址	Return	函数的返回值
Value	操作数的值	Taint	参数或返回值的污点信息
Taint	操作数的污点状态信息	Offset	相关污点数据在污点源中的偏移值集合
Offset	污点操作数在污点源中的偏移值的集合	APIOrder	函数的调用序号
InsOrder	指令的操作序号		

5 通信协议逆向分析方法

5.1 语法结构划分

基于 ETPG,通信协议逆向分析分为语法结构划分和字段语义理解两部分实现.其中,语法结构划分方法实现对协议数据的语法要素的识别,包括分隔符、关键字、定向字段和数据字段.

5.1.1 分隔符

分隔符是用作分隔不同字段的常量.进程通常将分隔符常量与接收到的网络数据进行连续的比较,直至比较结果为真,从而在消息样本中定位分隔符.因此,可以根据分隔符的这个行为特征进行识别,具体方法如下:

1.从 SrcNode 开始遍历 ETPG,构造对污点数据和非污点数据进行比较的比较操作集合 CmpOpSet.CmpOpset 的每个元素表示为一个三元组 < Constant, Offset, TrueOffset > ,其中 Constant 记录参与比较的非污点数据的值,Offset 是参与比较的污点数据的污点源偏移值的集合,TrueOffset 是比较结果为真的污点数据的偏移值的集合.

2.将各 Constant 对应的 Offset 集合元素按从小到大的次序排列为 Offset$\langle n_1, n_2, \cdots, n_m \rangle$,如果其中存在长度 4 的连续序列 < n_i, n_{i+1}, $\cdots$, n_j > $n_j - i \geqslant 3$,并且 $n_j \in$ TrueOffset,则判断 Constant 为分隔符.

3.分析分隔符 Constant 对应的 TrueOffset,如果 $n_i \in$ TrueOffSet 且 $n_i - 1 \in$ Offset,则 n_i 记录了分隔符在消息数据中的偏移位置.分隔符在消息数据中的位置的集合,记为 SeparatorOffset.

4.对以上得到的单字节分隔符进行合并,得到 AllSeparatorOffset = $\bigcup_i$SeparatorOffset$_i$,并将 AllSeparatorOffset 中的连续成员合并,实现对多字节分隔符的识别.

在第 2 个步骤中,作为分隔符判断依据的连续序列

长度的选择非常重要:选值过小会引起不必要的划分,选值过大可能造成分隔符识别的遗漏,导致不能正确划分协议字段.选择连续序列长度不小于4作为分隔符的必要条件,是考虑到 int 等非字符的基本数据类型是以4字节存储的.

5.1.2 关键字

关键字也是协议数据中的常量,通常用于条件控制进程的执行流程.进程通过比较操作来使用协议数据中的关键字.因此,识别关键字也需要分析进程执行过程中的比较操作,与识别分隔符不同,关注的重点是结果为"真"的比较操作,而不是连续比较的次数.具体的识别方法如下:

1.同识别分隔符的第一步,构造比较操作集合 CmpOpSet.

2.合并各 Constant 对应的 TrueOffSet 得到 AllTrueOffSet = $\bigcup_i \text{TrueOffset}_i$,将 AllTrueOffset 的元素按从小到大的次序排列为 AllTrueOffset < $n_1, n_2, \cdots\cdots, n_m$ > .

3.排除 AllTrueOffset 中分隔符对应的偏移值,合并其中的连续序列即得到关键字.

由于 ETPG 的节点中记录了 Offset 信息,因此即便消息数据经过变换后才被进程使用,上述方法仍能实现正确识别.

5.1.3 定向字段

定向字段是指示消息数据中其他字段的位置或长度的字段,由定向字段说明的字段称为目标字段.

进程常通过循环操作访问目标字段,此时定向字段用于设置循环的结束条件.但[12]等研究成果表明在二进制执行码流的基础上直接检测循环结构非常困难,对此我们提出了一种变通的方法:

1.遍历 ETPG 的 InsNode,查找跳转目标地址小于当前 EIP 的条件跳转指令节点,构造集合 TaintConditionJMP.

2.合并集合 TaintConditionJMP 中每个节点记录的 Offset 得到 DirectionOffSet = $\bigcup_i \text{OffSet}_i$.

3.排除 DirectionOffset 中的分隔符和关键字,即得到定向字段在污点源中的偏移值的集合.

该方法虽然不完备,但是执行效率很高,识别用作设置循环结束条件的定向字段具有可行性.定向字段还有另一种使用方式,即被用于间接寻址污点数据.例如,对指令"MOV EBX,[EDI + EAX]",如果 isTaint(EAX) = 1,且 isTaint([EDI + EAX]) = 1,那么 OffsetInSource(EAX)即为定向数据.分析 ETPG 的 InsNode 中存在的污点数据用于间接寻址污点数据的情况,即可识别出以这种方式作用的定向字段.

5.1.4 数据字段

数据字段是协议数据中除分隔符、关键字和定向字段之外的字段.数据字段分为由定向字段指示的目标字段和由分隔符、关键字等协议元素分隔的其他字段.完成对这些语法要素的识别后,数据字段自然显现.

5.2 字段语义理解

字段语义理解的目标是提供各个字段的数据类型和意义.对分隔符、关键字、定向字段等语法元素,由于形式的任意性和使用方式固定性,它们的语义由各自的角色决定,分析它们的数据类型意义不大.进程对数据字段的处理往往体现了进程的具体功能,因此,数据字段是协议语义理解的重点对象.

基本思路是借助函数级 ETPG 提取得到数据字段相关的特征 API 函数调用序列,利用对应函数的语义信息,推导出各数据字段的数据类型和语义.为此,我们选取了文件、进程、注册表、网络、系统服务和字符串处理六大类 API 函数中具有代表性的特征函数,结合函数功能对函数自身、函数参数的数据类型和语义进行了细致的解析,建立起函数语义数据库(Function Semantic Database,FSD).对于指针类型的函数参数,如果是结构体指针,我们跟踪到指针对应的结构体类型,并解析记录该结构体的各个成员的数据类型和语义信息.基于 FSD,我们实现数据字段语义理解的方法如下:

1.根据数据字段在污点源中的地址,从 SrcNode 开始查询函数级 ETPG,汇总查询得到的 APINode 生成该数据字段对应的特征 API 调用序列 APINodeSeq.

2.对每一个 $\text{An}_i \in$ APINodeSeq,根据节点中记录的信息 $\text{An}_i \rightarrow$ Taint,得到该节点对应的 API 函数调用的参数或返回值的污点状态,查询 FSD,得到相关污点数据对应的语义信息.

3.合并从调用序列 APINodeSeq 各个节点处得到的语义信息,从而得到对应字段的语义.

通过对数据字段的语义分析,可以区分出数据字段中的有效字段和没有被进程使用的空白字段,补充和完善语法划分的结果.虽然仅从函数语义出发推导字段语义并不完备,但是 API 函数的数据类型和参数语义具有很好的可靠性,难以篡改利用,由此得出的字段数据类型和语义理解的准确性较高.另外,如果函数语义不足以为数据字段提供足够的语义理解,我们将参考数据字段相关的可理解的关键字的语义信息作为补充.

6 实验评估

6.1 系统实现

我们基于 WooKon 恶意代码分析平台[13]开发实现了通信协议逆向分析的原型系统.将实验样本部署在 WooKon 平台的虚拟环境中运行的 Windows XP 系统中,

利用反汇编引擎 Udis86[14] 对单步执行的指令进行解析,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.函数级进程执行信息提取方面,我们通过 Hook 机制监控 FSD 中的特征 API 调用,获取函数的名称、参数值、返回值等信息,用以查询 FSD 获得函数语义.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表 4 所示.

表 4 原型系统的软硬件环境

处理器	Intel Core2 Duo CPU E7300@2.66G
内存	DDR2 667MHz 2G
硬盘	SATA 250G
主板	Intel G31
主机操作系统	Fedora Core 10 64bit
虚拟操作系统	Windows XP SP2

6.2 木马样本 Trojan

Trojan.exe 是一个典型木马,具有上传、下载、浏览文件以及执行代码等功能.实验分析了控制下载行为的命令消息.

表 5 Trojan.exe 下载行为第一条消息的协议逆向分析结果

范围 Range	语法类型 Syntax	数据类型 Type	值 Value	语义 Semantic
[0,6]	Keyword	- -	COMMAND	COMMAND
7	Separator	- -	0x3a(:)	:
[8,15]	Keyword	- -	download	download
16	Separator	- -	0x3a(:)	:
[17,24]	Data	unsigned long	0xc0a80599	网络地址 192.168.5.153
25	Separator	- -	0x3a(:)	:
[26,29]	Data	unsigned long	0x15b3	端口号 5555
30	Separator	- -	0x3a(:)	:
[31,37]	Keyword	- -	SrcFile	SrcFile
38	Separator	- -	0x3a(:)	:
[39,42]	Direction	- -	0x00000009	远端源文件名长度
[43,51]	Data	string	c:\1.txt	远端源文件名
[52,58]	Keyword	- -	DstFile	DstFile
59	Separator	- -	0x3a(:)	:
[60,63]	Direction	- -	0x00000009	本地目标文件名长度
[64,72]	Data	string	c:\2.txt	本地目标文件名

实验中,Trojan.exe 木马进程在发生下载行为时,接收到两次网络消息数据,长度分别为 73 字节和 44 字节,进程动态执行信息提取模块将它们作为 Trojan.exe 进程的原始污点源,并跟踪记录进程对它们的操作流程,生成 ETPG.等待时间 60s 后,基于 ETPG 进行消息协议的逆向分析,得到了两条消息的协议规范五元组.逆向分析的具体结果如表 5 和表 6 所示.

表 6 Trojan.exe 下载行为第二条消息的协议逆向分析结果

范围 Range	语法类型 Syntax	数据类型 Type	值 Value	语义 Semantic
[0,3]	Keyword	- -	0x00000002	命令标号
[4,7]	Direction	- -	0x00000009	文件名长度
[8,16]	Data	string	c:\1.txt	文件名 c:\1.txt
[17,24]	Keyword	- -	FileSize	FileSize
25	Separator	- -	0x3a(:)	:
[26,29]	Data	unsigned int	0x00000264	文件大小
[30,38]	Keyword	- -	BlockSize	BlockSize
39	Separator	- -	0x3a(:)	:
[40,43]	Data	unsigned int	0x00000040	传输单位长度

从实验结果可知,我们准确地识别出了协议数据中的分隔符、关键字、定向字段和数据字段,并给出了每个元素在协议数据中的具体位置.利用 FSD 也得出了数据字段的语义信息.根据分析结果可以推断,木马第一次接收到的消息为下载命令,第二次接收到的消息是文件传输数据过程中的控制参数.

以本实验为例详细说明对数据字段语义理解的实现过程.分析 ETPG 中的 APINode,得到各数据字段的 API 调用序列如表 7 所示.例如,消息 1 中偏移值为[17,24]的数据字段,它的特征 API 调用依次为:sscanf 和 connect.具体分析 sscanf 的调用详情,此次调用 sscanf 的第二个参数为"%x",查询 FSD,可以推断[17,24]字段的类型为无符号整型.分析 connect 对应 APINode 节点,[17,24]对应的污点数据在第二个参数指向的结构体中使用,进一步分析其在结构体中的偏移,查询 FSD 可知其为无符号长整型数据,并被用作 IP 地址.其他数据字段的类型和语义可通过类似过程得到.其中,消息 2 的[26,29]和[40,43]两个字段按上述过程只能得知它们的数据类型,从 malloc 函数可以大致推断它们指示长度,但不能更深入了解它们的具体用途.为此,我们参考了它们对应的关键字,从而给出了更具体的语义信息.

表 7 Trojan.exe 下载行为消息中各数据字段对应的函数调用序列

范围	特征 API 序列	数据类型	值	语义
[17,24]	sscanf,connect	unsigned long	0xc0a80599	网络地址 192.168.5.153
[26,29]	sscanf,htons,connect	unsigned short	0x15b3	端口号 5555
[43,51]	sscanf,memcpy,send	string	c:\1.txt	远端源文件名
[64,72]	sscanf,CreateFileA	string	c:\2.txt	本地目标文件名
[8,16]	sscanf,memcpy,send	string	c:\1.txt	文件名 c:\1.txt
[26,29]	sscanf,malloc	unsigned int	0x00000264	文件大小
[40,43]	sscanf,malloc	unsigned int	0x00000040	传输单位长度

6.3 僵尸程序样本 SDBot

SDBot 是一类知名的僵尸程序.实验采用的样本是

SDBot 的一个变种,它集 IRC 后门、蠕虫功能于一体,可以通过网络共享和操作系统漏洞进行传播,并通过 IRC 服务器接受执行攻击者发出的指令,例如安装/卸载后门、下载并运行文件、结束进程、运行代理服务器、发动 DoS 攻击等.

实验中,我们向该 SDBot 变种样本发送 download 命令,让它从一个 Web 服务器上下载名为 DropperA.exe 的恶意程序.对 download 命令消息的逆向分析结果,如附表 8 所示.实验结果正确识别出了消息中的分隔符、关键字和数据字段,并给出了各字段的语义理解.

表 8　对 SDBot 变种样本 download 命令消息的逆向分析结果

范围 Range	语法类型 Syntax	数据类型 Type	值 Value	语义 Semantic
[0,8]	Keyword	– –	.download	.download
Separator	– –	0x20(空格)	空格	
[10,48]	Data	string	"http://192.168.5.4/samples/DropperA.exe"	下载目标地址
49	Separator	– –	0x20(空格)	空格
[50,57]	Data	string	"c:\1.exe"	本地文件名
58	Separator	– –	0x20(空格)	空格
59	Keyword	– –	1	1

6.4　讨论

通过上述恶意代码样本的实验,验证了我们的协议逆向分析方法可以准确识别消息数据的各种协议语法元素,并给出相应字段的语义理解.表 9 显示了实验过程的性能数据.其中,分析耗时包含了预留 60s 的进程动态执行信息提取时间.从性能数据可见我们的分析方法记录了进程动态执行的大量底层信息,并且具有较高的执行效率.

表 9　协议逆向分析性能数据

通信数据	InsNode 节点数量	APINode 节点数量	分析产生 的数据量	分析 耗时
Trojan 木马样本下载命令消息 1	14236	25	53.5MB	1m16s
Trojan 木马样本下载命令消息 2	1947	14	26MB	1m7s
SDBot 变种样本下载命令	2561	19	31.8MB	1min14s

与基于进程行为的分析方法[7]比较,我们可以获取目标进程的指令级执行流程,从而更细致的刻画行为特征,因此语法划分结果更准确.另外,我们借助函数调用序列可以给出数据字段的数据类型和语义理解,这是[6]所不能提供的.与基于网络数据包的协议逆向的方法[4]比较,它们对关键字等语法元素的识别往往需要借助经验假设,但我们不依赖任何经验假设,结果更加可靠.例如,网络恶意代码的通信协议可能并不使用常见分隔符而使用任意字符,关键字也可能通过编码变换变得不可辨识.我们的方法则不受此影响.因为不论分隔符或关键字的形式如何,恶意进程对这些协议语法要素的底层处理过程是相对稳定的.

7　总结

本文提出了一种利用动态污点分析监控恶意进程的执行流程,通过识别进程对协议数据的处理过程的行为特征实现恶意代码通信协议逆向分析的方法.我们设计实现了针对通信协议逆向分析的动态污点分析框架,构建指令级和函数级相结合的 ETPG 记录进程动态执行信息,并在 ETPG 的基础上识别分隔符、关键字、定向字段等协议语法要素和提取数据字段语义信息.本文着重讨论了如何对恶意代码接收到的单条通信数据进行协议逆向分析.下一步,我们将研究存在多消息会话的情况下,如何建立不同消息之间的关联关系实现完整的通信协议的逆向分析.

参考文献

[1] W Cui, M Peinado, H J Wang, and M E Locasto. Shieldgen: Automatic data patch generation for unknown vulnerabilities with informed probing[A]. IEEE Symposium on Security and Privacy[C]. Oakland, USA, May 20 – 23, 2007. 252 – 266

[2] The SNORT network intrusion detection system[OL]. http://www.snort.org.

[3] 胡振宇,刘在强,苏璞睿,冯登国.基于协议分析的 IM 阻断策略及算法分析[J].电子学报,2005 33 (10):1830 – 1834.
HU Zhen-yu, LIU Zai-qiang, SU Pu-rui, FENG Deng-guo. On the Policy and Algorithm to Block Instant Messaging[J]. Acta Electronica Sinica. 2005 33 (10):1830 – 1834. (in Chinese)

[4] W Cui, J Kannan, and H J Wang. Discoverer: Automatic Protocol Reverse Engineering from Network Traces[A]. Proceedings of the 16th USENIX Security Symposium[C]. Boston, MA, August. 2007.

[5] Higgins D G, Sharp P M. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer[J]. Gene 73 (1):237 – 44. 1988

[6] J Caballero and D Song. Polyglot: Automatic Extraction of Protocol Format using Dynamic Binary Analysis[A]. In ACM Conference on Computer and Communications Security (CCS), 2007[C]. Singapore, March 20 – 22, 2007.

[7] Z Lin, X Jiang, D Xu, X Zhang. Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution[A]. Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)[C]. Feb. 2008.

[8] G Wondracek, P M Comparetti, C Kruegel, and E Krida. Automatic Network Protocol Analysis[A]. Proceedings of the 15th Annual Network and Distributed System Secrurity Symposium (NDSS'08)[C]. Feb. 2008.

[9] The Protocol Information Projects[OL]. http://baselineresearch.net/PI/

[10] Weidong Cui, Marcus Peinado, Karl Chen, Helen J Wang, and

Luis Irun-Briz. Tupni: Automatic Reverse Engineering of Input Formats[A]. Proceedings of the 15th ACM Conferences on Computer and Communication Security[C]. CCS 2008 (October 2008).

[11] J Newsome and D Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[A]. Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS05)[C]. Feb. 2005.

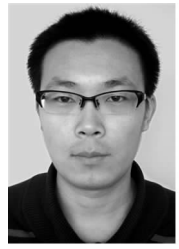
[12] G Ramalingam. Identifying loops in almost linear time[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), v.21 n.2, p.175 – 188, March 1999

[13] WooKon 恶意代码分析平台[OL]. http://www.isec.ac.cn/group_detail.jsp? gid = 2&id = 62

[14] Udis86[OL]. http://udis86.sourceforge.net/

作者简介

刘 豫 男,1983 年出生于四川眉山,博士研究生,主要研究方向为网络信息安全与恶意代码分析. E-mail: liuyu@chinamobile.com



王明华 男,1986 年出生于吉林长春,硕士研究生,主要研究方向为网络信息安全与恶意代码分析. E-mail: wangminghua@is.iscas.ac.cn

苏璞睿 男,1976 年出生于湖北宜昌,博士,副研究员,主要研究方向为恶意代码分析与防范.

冯登国 男,1965 年出生于陕西靖边,博士,研究员,博士生导师,主要研究方向为密码学与信息安全.