软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(7):1676−1689 [doi: 10.3724/SP.J.1001.2011.03858]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

恶意软件网络协议的语法和行为语义分析方法*

应凌云[1,2,3+], 杨 轶[1], 冯登国[1,2], 苏璞睿[1]

[1](中国科学院 软件研究所 信息安全国家重点实验室,北京 100190)

[2](中国科学院 研究生院 信息安全国家重点实验室,北京 100049)

[3](信息安全共性技术国家工程研究中心,北京 100190)

Syntax and Behavior Semantics Analysis of Network Protocol of Malware

YING Ling-Yun[1,2,3+], YANG Yi[1], FENG Deng-Guo[1,2], SU Pu-Rui[1]

[1](State Key Laboratory of Information Security, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Information Security, Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

[3](National Engineering Research Center for Information Security, Beijing 100190, China)

+ Corresponding author: E-mail: yly@is.iscas.ac.cn

Ying LY, Yang Y, Feng DG, Su PR. Syntax and behavior semantics analysis of network protocol of malware. *Journal of Software*, 2011,22(7):1676−1689. http://www.jos.org.cn/1000-9825/3858.htm

Abstract: Network protocol reverse analysis is an important aspect of malware analysis. There are many different network protocols and every protocol contains different types of fields that result in various malware behaviors. Without the protocol syntax and filed semantics, analyzers cannot understand how malware interacts with the outside network. This paper presents a syntax and a behavior semantics analysis method of the network protocol. By monitoring the way malware parse the network data and by using different fields in a virtual execution environment, this method can identify protocol fields, extract protocol syntax and correlate each syntax with malware behaviors, accordingly. This paper designs and implements the prototype Prama (protocol reverse analyzer for malware analysis). Experimental results show that this method can correctly infer protocol syntax and tag fields with meaningful malware behaviors.

Key words: malware analysis; network protocol reverse analysis; dynamic analysis; network security

摘 要: 网络协议逆向分析是恶意软件分析的一项重要内容.现有的网络协议逆向分析方法主要考虑获取消息格式和协议语法,缺少数据的行为语义,导致分析者难以在网络数据和恶意软件行为之间建立起对应关系.提出一种网络协议的语法规范和字段行为语义分析方法,该方法利用基于虚拟执行环境的动态程序分析技术,通过分析恶意软件对网络数据的解析过程提取协议语法信息,并根据恶意软件对协议字段的使用方式获取字段的程序行为语义.通过结合 API 拦截和指令执行监控,该方法降低了分析复杂度,提高了分析效率.在所设计和实现的原型系统 Prama(protocol reverse analyzer for malware analysis)上的实验结果表明,该方法能够较为准确地识别字段,提取协议语法规范,并能在命令字段与其引起的程序行为之间建立起有效的对应关系.

关键词: 恶意软件分析;网络协议逆向分析;动态分析;网络安全

* 基金项目: 国家自然科学基金(60703076, 61073179); 国家高技术研究发展计划(863)(2009AA01Z435, 2007AA01Z451)
收稿时间: 2009-04-29; 修改时间: 2009-10-23; 定稿时间: 2010-04-14

中图法分类号: TP393 文献标识码: A

随着网络应用的普及,网络通信已成为各种软件系统的重要基础功能之一,网络协议分析也成为软件分析工作的重要一环.识别软件使用的通信协议、分析软件网络交互操作是软件安全性分析、流量控制和网络安全策略制定等工作的重要内容.尤其是对于迅速传播的蠕虫、僵尸程序、木马等恶意软件,快速分析恶意软件的网络协议、掌握其命令控制方式是对其做出及时反应的关键步骤.

现有的网络协议分析方法主要通过抓取网络通信流量,利用流量的端口、重复出现的字节序列等统计特征来进行分析.由于恶意软件在实现协议标准时的差异、有些甚至使用非标准的私有通信协议,导致流量分析只能获取标准协议相关的结构信息,难以获取流量的语义信息,具有较大的局限性.并且,单纯的网络流量分析也无法处理恶意软件中常见的混淆或加密流量问题.

利用程序分析方法从软件代码实现中分析协议是网络协议分析的另一种思路,主要分为静态分析和动态分析两大类.静态分析方法主要是利用各种反汇编工具对软件进行反汇编,人工分析软件的执行流程和代码逻辑,进而提取出网络协议的相关结构信息.由于静态分析难以处理程序的动态行为,如以寄存器值为目标地址的跳转等[1],并且需要大量的人工参与,因此通常作为网络协议分析的辅助分析方法.动态分析方法则通过动态跟踪程序的运行过程、分析程序对各种数据的处理方式来揭示程序的执行逻辑,能够较好地处理恶意软件的间接跳转和自修改代码等动态行为,因而具有较高的准确性.

传统的分析方法把恶意软件及其产生的流量视为独立的分析目标,使得程序分析和流量分析割裂开来,忽视了恶意软件行为和网络流量之间的内在联系.我们将恶意软件运行过程中产生的各种系统调用视为其行为,将恶意软件在处理接收的网络数据时的行为作为该网络数据的行为语义,在分析过程中还原恶意软件行为和网络流量之间的对应关系.

基于上述思想,本文提出了一种恶意软件网络协议的语法规范提取和行为语义关联方法,该方法利用动态程序分析技术跟踪恶意软件对网络数据的解析过程,通过分析恶意软件如何识别和切割数据,提取出网络协议的语法信息.同时,通过记录恶意软件的行为信息并标记网络数据的传播和使用过程,在网络数据使用点将字段关联到其导致的软件行为序列,提取出行为语义信息.最后,利用语法和行为语义相结合的结果归类方法对提取的信息进行确认,实现网络协议分析.

本文的主要贡献如下:(1) 提出了一种基于动态程序分析的网络协议格式自动分析方法.通过动态监控软件对网络数据的解析过程,根据软件提取字段的方式,我们实现了协议字段的自动划分和具有特殊协议语义字段的识别,并可处理加密通信协议;(2) 提出一种在网络协议字段与程序行为之间建立关联的方法.利用网络数据的传播和使用记录,结合软件的行为信息,我们在命令字段与其触发的程序行为之间联立关联,实现了字段行为语义的提取;(3) 实现了一套基于语法和行为语义分析的网络协议分析工具原型系统 Prama(protocol reverse analyzer for malware analysis).对不同的恶意软件样本的分析表明,本文提出的方法能够较为准确地提取协议语法规范;同时,以 API 序列表示的程序行为语义也能较好地揭示字段的真实操作含义.

本文第 1 节介绍相关工作.第 2 节阐述本文研究针对的问题和采取的方法.第 3 节论述原型系统实现中的关键点和相关的实验分析结果.最后,第 4 节对本文的工作进行总结.

1 相关工作

根据分析目标的不同,网络协议的自动解析和逆向分析方法主要有基于流量分析的方法和基于软件分析的方法两种.基于流量分析的方法有 Cui 等人提出的通过观察网络数据流中不同协议字段的出现次数,推断协议字段结构的方法[2].由于这种方法不考虑具体协议内容,因此分析效果依赖于样本数据流中字段重复出现的次数和每次出现的形式是否一致.Justin 等人提出了通过对网络流量的内容进行统计建模,识别和区分流量对应的不同协议的方法[3].但该方法只能用于识别协议类型,无法解析具体的协议结构.Small 等人则提出了利用自然语言处理和字符串对齐算法对网络数据流进行学习,自动地对收到的网络请求进行应答的方法[4].该方法能

够根据数据内容的排列模式从样本数据中选择类似的会话用于自动应答,但无法分析具体协议语法和语义.这类基于统计分析的方法需要大量的网络数据作为分析样本,同时无法处理混淆、加密流量,更无法获取协议字段的语义信息.

基于软件分析的网络协议分析方法又可以进一步分为静态分析和动态分析两类.静态分析包括利用 IDA Pro,W32Dasm 等工具对程序代码进行静态反汇编分析的方法[5]、Vigna 和 Christodorescu 等人提出的利用静态反汇编技术提取程序中包含的信息,还原代码高级语义的方法[6,7]以及 Cavadini 提出的利用切片技术进行静态数据流分析,发现程序中的可能执行路径的方法[8].这类方法需要先静态分析提取程序信息,再人工分析网络协议的结构及其功能.由于二进制代码分析的复杂性随着程序规模的变大而迅速增长,这些方法难以有效提取恶意软件中的网络协议信息.此外,Moser 和 Linn 等人还指出了静态分析技术在处理恶意软件代码时面临的反汇编困难等问题[1,9].

动态分析包括利用 OllyDbg,WinDbg 和 SoftICE 等工具对程序进行跟踪分析的方法.由于恶意软件能够检测到调试器的存在,进而故意隐藏各种行为.为此,人们提出了 Valgrind[10],Pin[11]和 DynamoRIO[12]等能够进行指令级别程序控制的专用分析框架,并先后提出了 Prospex[13],Polyglot[14]和 Tupni[15]等基于不同动态程序分析平台的协议格式推断、协议规范提取方法.Newsome,Egele 和 Saxena 等人则提出了利用污点传播技术开展动态数据流分析,通过追踪程序的数据处理过程,进而分析程序行为的方法[16−18].同时,随着虚拟机技术的不断发展,Huang,Bayer 和 Dinaburg 等人分别提出了基于软件和硬件虚拟技术,构造可控虚拟运行环境用于动态分析程序行为的方法[19−21].基于此类虚拟环境构造技术,Wondracek 和 Lin 等人提出了通过动态监控程序执行过程,从程序运行记录中提取协议信息的分析方法[22,23].这类方法提取的协议格式精确度较高,并且能够识别一些常见字段的协议语义,甚至能够部分重构协议状态机[22].

此外,软件测试领域也有一些网络协议分析相关的工作,如 Kaksonen 等人从黑盒测试的角度提出利用协议规范作为指导,产生测试输入,开展基于语法的软件协议实现安全性分析方法[24].这类方法的目标与网络协议逆向分析刚好相反.由于恶意软件采用的协议规范通常无法获知,因此我们需要通过分析其网络数据处理过程来逆向产生网络协议规范.在此基础上,我们可以考虑采用类似的黑盒测试方法查找更多可能的消息类型与字段组合方式,挖掘恶意软件的协议实现漏洞.

动态分析方法的优点在于,能够精确地跟踪程序的执行过程,但同时也导致分析粒度过细,使得整个分析过程十分复杂.为此,我们采取基于虚拟运行环境的动态分析为主、静态分析为辅的方式开展分析.同时我们发现,软件在处理网络数据时往往涉及到大量的操作系统库函数,因此,我们通过拦截相关库函数的调用获取其参数和返回值,避免对这些库函数内部的指令进行分析,从而简化了分析过程.

我们的分析方法与 Polyglot[14]等方法不同,上述分析方法都只关注协议格式的解析和语法信息的提取,而我们注意到,在恶意软件分析过程中不仅需要解析网络协议的语法格式,更需要了解其中各个字段的语义以及字段与恶意软件行为之间的对应关系.没有协议的程序行为语义信息,分析人员就不能掌握不同语法格式的数据对应的不同应用场景,也难以开展像 Botspy 这样的网络渗透分析.我们的分析方法在消息格式解析的基础上,实现了字段的程序行为语义关联分析,不仅获取字段在协议规范中的含义,还要推导字段的程序行为解释.例如一个字段的协议语法解释是 4 个字节的文本字符串,协议语义解释是命令字段,而程序行为含义则是运行参数字段指定的可执行文件.了解字段在协议规范中和程序行为中的含义,对于分析恶意软件的用途和破坏力、提取恶意软件的通信特征和行为特征都具有重要价值.

2 分析方法

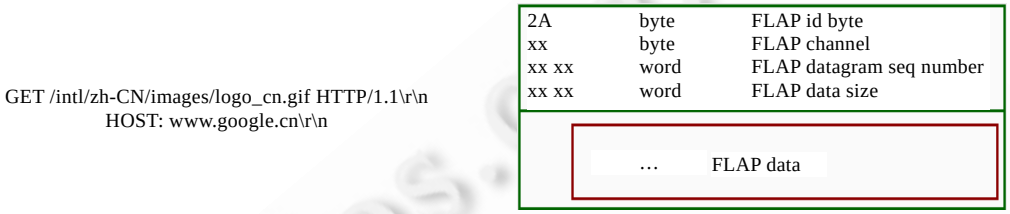
本节我们首先介绍本文研究的问题,对其中涉及的概念进行说明和定义,然后根据恶意软件对接收的网络数据的处理过程,分为协议规范分析、行为语义分析和结果归并 3 个阶段,分别阐述协议字段分割、字段语义提取和关联以及多个消息的分析结果归并过程中的具体分析方法.

2.1 问题描述

网络行为分析是恶意软件分析中十分重要的部分.典型地,如发现僵尸程序时,我们不仅需要分析僵尸程序的代码特征,还需要分析僵尸程序如何与僵尸网络交互,僵尸网络控制者如何从僵尸网络收集信息、如何向僵尸程序发送命令以及如何对僵尸主机进行控制等.这就要求我们不仅要掌握恶意软件通信协议的字段组成等格式信息,还需要了解恶意软件如何处理各个字段并在程序行为上做出响应,也就是字段对应的程序行为语义.

根据被分析的网络协议使用的字符集,我们可以将协议分为如下 3 种类型:(1) 文本型协议,如HTTP,SMTP,IRC 协议;(2) 二进制协议,如 DNS,NFS 协议;(3) 混合型协议,如 SMB 协议.

同时,根据协议中各个字段的标识和区分方式的不同,我们可以将字段分割方式分为如下 3 种:(1) 分隔符划分字段,如图 1(a)所示,HTTP 协议以"\r\n"分割不同行,并以空格分割不同字段;(2) 固定格式字段,如图 1(b)所示,OSCAR 协议中前 3 个字段由约定的字段值类型大小直接确定;(3) 由其他字段确定的字段,包括由长度指示字段直接确定的字段,如图 1(b)中的"FLAP data"字段即由它前面的"FLAP data size"字段确定,以及由其他字段间接确定的字段,如填充字段.



(a) HTTP protocol structure in Wireshark
(a) Wireshark 显示的 HTTP 协议结构

(b) OSCAR/ICQ protocol message structure
(b) OSCAR/ICQ 协议消息结构

Fig.1 Different type of fields separation methods

图 1 不同字段分割方式示意图

协议类型和字段分割方式之间没有必然的联系,但是,为了避免内容和分隔符相混淆,文本型协议普遍采用分隔符划分字段,二进制协议多数采用固定格式和长度标识字段,混合型协议则在不同部分采用不同的字段划分方式.

本文以常见的文本型协议为分析目标,针对不同的字段分割方式,识别和提取协议中的不同字段.为了叙述方便,我们把分析过程中涉及的相关概念作如下定义,并据此对研究目标进行描述:

- 原子字段:在恶意软件解析网络数据时的最小不可分割的连续字节序列,记为 f;
- 分隔符:用于将网络数据划分成不同字段的特定原子字段,记为 d;
- 复合字段:由若干个前后相继的原子字段组成的字段,记为 F;
- 原子字段集 Γ:$\Gamma=\{f_i|i=1,\ldots,n\}$;
- 分隔符集 Λ:$\Lambda=\{d_j|j=1,\ldots,k\}$;
- 消息集合 M: $M=\left\{m_t=\sum_{i=1}^{n} format_t(f_i)\,|\,t=1,...,s,\ f_i\in\Gamma\right\}$.其中,$format(x)$为符号 x 在序列中出现位置的映射函数,$\sum$表示符号的串接操作.

根据上述定义,在图 1(a)所示例子中,GET 字段是原子字段,"\r\n"和空格是分隔符,而每一行都是一个复合字段.且由定义可知,$\Lambda\subset\Gamma$.

为了用统一且可比较的方式表示恶意软件行为,我们以恶意软件的 API 调用作为恶意软件行为的标识.

- API 监控集合 A:$A=\{a_i|i=1,\ldots,m\}$.其中,a_i 为我们监控的 API 调用;
- 单一行为:在恶意软件运行过程中监控到的一次调用点为 s 的 a_i 调用,也是行为标识的最小单元,记为 $b_s=logging(a_i)$.其中,$logging(a_i)$表示记录 a_i 调用相关的数据;
- 字段使用点:字段 f_i 参与的比较操作中所有比较成功的操作的位置,记为 $P(f_i)=\{p_1,p_2,\ldots,p_n\}$;

- 行为序列:在 API 调用记录中,从 s 处开始的长度为 l 的单一行为的序列,记为 $b(s,l)=\{b_s,b_{s+1},\ldots,b_{s+l}\}$;
- 字段行为语义:字段 f_i 的所有使用点对应的行为序列的集合,记为 $B(f_i)=\{b(p_j,l)|p_j\in P(f_i)\}$.

利用上述定义,本文的研究目标可以描述为:

(1) 对于恶意软件接收到的网络协议消息 m_t,根据恶意软件对协议字段的解析过程识别并提取 f_i,进而获取消息 m_t 对应的字段映射函数 $format_t(x)$,解析消息的字段组成格式;

(2) 同时记录恶意软件运行过程中产生的行为,并根据恶意软件对各个字段 f_i 的使用方式,将其关联到对应的程序行为序列集合 $B(f_i)$,获取字段的行为语义;

(3) 结合前两步分析的结果,对消息格式按照字段类型进行抽象,然后利用格式信息和行为语义信息进行归类,缩减和确认分析结果.

2.2 协议规范分析

协议语法分析阶段主要是完成协议字段的识别和划分工作.由于恶意软件本身就是其采用的网络协议的一个实现,代码中通常包含协议中使用的固定字符串.因此,我们对软件的可执行文件进行静态扫描分析,提取其中包含的字符串常量作为协议字段的候选集 F_c 以简化分析.

由于协议解析和字段内容比较中都大量使用了循环结构,我们利用 Sreedhar 等人提出的循环识别算法[25]对软件进行预处理,标记其中的循环结构区块.通过这种方法,我们在动态分析过程中能够区分当前指令是否属于某个循环结构,从而识别连续相继的数据操作,进而将涉及的数据区块作为一个整体来处理.

与 Prospex[13]等同样基于动态程序分析的协议规范提取方法不同,我们注意到,单纯的指令分析粒度过细,导致分析过程过于复杂.而实际上我们并不需要了解程序的所有执行过程,只需关注其中程序自身代码的行为.因此,我们采用 API 分析和指令分析相结合的方式,略过所有拦截到的 API 内部的指令,只记录相关 API 的调用点、参数和返回值.同时,利用污点传播方法对其他指令开展分析,确保不同 API 调用之间数据的相关性,维持污点数据的传播关系.

我们将 *recv*()等接收操作获取的网络数据 buff 标记为污点源$\langle i,p\rangle$,其中,i 是全局唯一污点标签值,p 为原始网络数据包标记.在程序处理 buff 中的数据时,所有源为带有污点标记的数据的移动和赋值操作都将导致目标数据被污染,同时附加污点标记$\langle i',p,off,opc,opr\rangle$.其中,*off* 是当前数据在原始污点源中的偏移,*opc* 为当前指令操作码,*opr* 为操作数.通过这种方式,我们能够回溯传播过程,根据字段 f 最终解析后的状态找出 f 对应的原始数据包的位置.

由于污点传播过程包含了数据来源信息,因此需要对加密流量作特殊处理.注意到,流量加密通常以数据包或应用层消息为单位,恶意软件在处理加密流量时需要先解密接收到的数据,然后再按照处理非加密流量的方式逐个字段进行处理.因此,我们通过一种启发式算法对加密流量进行处理:如果污点源$\langle i,p\rangle$被作为一个整体进行处理,那么我们将其视为解密操作,同时把处理结果作为新的污点源$\langle i',p\rangle$.这使得我们可以在不破解加密算法和提取密钥的情况下,实现对协议数据的分析.

通过拦截数据处理过程中的字符处理相关 API 调用,记录每个 API 的名字 N、参数 $P_i(i=1,\ldots,k)$、调用点 C、调用栈快照 S 和返回值 R,我们采用如算法 1 所示的方法识别字段.同时,我们将与污点数据进行连续相继的比较操作的常量视为分隔符.通过这两种方式,首先识别分隔符和直接与常量比较的原子字段.对于由长度指示确定的字段,得益于 API 监控所具有的高层语义,我们将污点数据操作相关 API 中所有作为长度参数传入的字段视为长度指示字段,并将被操作污点数据区块视为其指示的字段.在识别字段的同时,根据字段是否参与算术操作,我们将字段分为数值型和字符型两种类型,用于后续的结果归并.

算法 1. 协议字段识别算法.

输入:候选字段集 $F_c=\{f_i|i=1,\ldots,s\}$,API 拦截记录 $La=\{a_t=\langle N_t,C_t,S_t,R_t,P_{t1},P_{t2},\ldots,P_{tk}\rangle|t=1,\ldots,n\}$,污点传播记录 $Lt=\{T_j=\langle i_j,p_j,off_j,opc_j,opr_j\rangle|j=1,\ldots,m\}$;

输出:协议字段集Ω.

FieldIdentify(F_c,*La*,*Lt*) {

```
Ω={};
bTaint=bContain=false;
for (i=1; i≤s; i++;) {
  for (t=1; t≤n; t++;) {
    if (N_t 的返回值 R_t 含义==false ) continue;
    for (q=1; q≤k; q++;) {
      if (P_tq 是污点数据) bTaint=true;
      if (f_i==P_tq) bContain=true;
    }
    if (bTaint && bContain) Ω=f_i∪Ω;
  }
}
return Ω;
}
```

对于固定格式字段的识别比较困难,我们只对直接针对污点源的数据操作进行了处理.若循环结构的循环条件未被污染,则视对应的连续相继的字节序列为一个固定格式字段,这导致我们无法识别无需用循环处理的机器字宽范围内的字段.由于文本协议很少使用这种方式分割字段,所以,这种简化处理是可以接受的,并且可以避免错误识别,实验分析也证实了这一点.

由于并不是所有的字段都会被程序处理,为此还需进行字段推导.我们将所有分隔符和已识别字段间的连续字节区域视为一个单独的字段加入结果字段集合Ω中,这是一种保守的处理,推导获得的字段可能仍然是复合字段,但是由于程序将其作为一个字段来处理,这种推导不会造成错误的划分.并且,与 Wondracek 等人提出的协议规范提取方法不同[22],我们的分析方法能够处理长度为一个字节的分隔符.事实上,HTTP,IRC 等协议都大量采用了空格、回车等单字节分隔符.

2.3 行为语义分析

行为语义分析的目的是在协议字段和软件行为之间建立起对应关系,揭示不同字段所蕴含的程序行为含义.由于不是所有的协议字段都有程序行为语义,像分隔符字段、协议控制字段等都只有协议规范上的含义,只有应用层协议命令字和参数等才具有对应的程序行为语义,因此,我们同样利用 API 调用分析和指令执行分析相结合的方式进行行为语义分析.

首先,我们需要识别协议命令字段.由于恶意软件需要对收到的不同命令做出不同的响应,因此我们可以利用恶意软件代码实现中所包含的命令字段解析过程进行识别.对于文本协议,这个过程一般通过各种字符串比较操作完成.我们将所有除分隔符之外直接参与字符串比较的字段作为候选命令字段加入命令字段集Ψ.对于$\forall f_i \in \Psi$,将调用点 s 满足 $s \leq p_j, p_j \in P(f_i)$的行为 b_s 视为字段 f_i 引起的恶意软件行为.其中,$\leq$为监控到的程序执行流上的偏序关系.

以图 2(a)所示的消息为例,Agobot 在响应这条消息时,将产生如下 API 调用序列:

gethostbyname(),*socket*(),*connect*(),*send*(),*fopen*(),*recv*(),*fputc*(),*recv*(),…,*fclose*(),*CreateProcess*().

由上面序列中各个 API 的作用和调用顺序,我们可以推断出该条消息会导致软件执行网络数据收发操作,创建文件的同时将数据写入文件,并创建新的进程以执行相关操作.

然后,我们根据字段是否直接作为非字符串来操作 API 的参数使用识别参数字段,这可以通过遍历字段的污点传播流图,查找是否存在满足要求的节点来实现.根据这一规则,在图 2(a)所示例子中,host.evil.com 字段被 *gethostbynam*()作为参数使用,/bugfix.exe 被 *send*()作为参数使用,%TEMP%\sysupdater.exe 被 *fopen*()作为参数使用.将所有识别的参数字段排除后,可得命令字段集为Ψ={http,update,v3}.这里,v3 字段实际上是一个命令参数.但是由于恶意软件对该字段的处理类似于命令字段,因此从程序行为语义上被判定为命令字段.由于同一个命令字段搭配不同的参数字段可能具有不同的行为,因此在命令字段比较成功之后,即使用点之后,还可能有参数

字段的比较操作,也就是在命令字段的使用点和引起的程序行为之间还存在参数字段的使用点.通过识别参数字段,不仅可以缩减命令字段集规模,还可以避免将参数字段关联到其后的程序行为,以减少错误的行为关联.

同时,为了避免不同字段行为语义之间的相互干扰,我们采用长度为 l 的滑动窗口从调用序列中截取字段行为语义序列.l 设为

$$l=\min\{H,P-P'\},$$

其中:P 为当前分析的命令字段使用点;P'为记录中距离 P 最近的下一个命令字段使用点;阈值 H 根据 Forrest 等人的研究成果[26],取为 11.并且,为了避免典型的 *send*(),*recv*()循环等重复操作的干扰,我们从 API 拦截记录中提取字段行为语义序列时不考虑 API 的重复出现,忽略当前序列中已提取的 API,如图 2(a)所示的消息调用序列,提取后为

gethostbyname(),*socket*(),*connect*(),*send*(),*fopen*(),*recv*(),*fputc*(),*fclose*(),*CreateProcess*().

最后,作为 API 行为语义的补充,我们还在指令分析过程中提取和记录了命令字段对应的代码块地址,以便需要进一步分析时进行人工分析和确认.由于恶意软件等通常会对自身代码进行加壳等保护,导致无法在真实执行的指令与静态代码之间直接对应,因此需要特殊处理.对于无法静态分析的软件,当我们在其运行过程中第 1 次调用网络相关 API 时,从虚拟运行环境中直接抓取程序在虚拟内存中已经脱壳的代码镜像,并据此镜像进行第 2.2 节中所述的字符串常量提取等静态分析操作.

PRIVMSG #Bot :.http.update http://host.evil.com /bugfix.exe %TEMP%\updater v3

(a) Agobot .http.update message sample
(a) Agobot .http.update 消息实例

PRIVMSG #Bot :.ftp.update UpUsEr UpPaSs ftp.evil.com /bugfix.exe %TEMP%\sysupdater.exe v3.3

(b) Agobot .ftp.update message sample
(b) Agobot .ftp.update 消息实例

Fig.2 Agobot communication messages
图 2 Agobot 通信消息实例

2.4 结果归并

在完成单个消息的语法分析和语义分析之后,我们尝试着关联多个不同消息的分析结果,将相似的消息归为一类,以便利用多条消息结果相互验证,确认某些特殊字段的含义.与 Prospex[13]等只关注协议语法规范的分析方法不同,我们在结果归并时不仅考虑字段的语法信息,还考虑了语义信息,以体现恶意软件在接收和处理不同消息时的不同特征.

首先,我们将字段分析结果抽象成以字段类型表示的形式,避免字段的具体内容对归并过程造成影响.仍以图 2 所示的两条消息为例,经过抽象后,其格式表达如图 3(a)所示.

由于网络协议消息抽象后的字段序列较短,我们采用 Needleman-Wunsch 算法[27]对抽象后的字段序列进行全局匹配分析.通过最大化相同类型的字段对齐数,捕捉不同数据包之间语法结构上的相似性.采用表 1 所示的相似性计分矩阵,设缺口扣分(gap penalty)为−1,则图 3(a)所示的两条抽象消息序列将如图 3(b)所示那样对齐.从对齐结果可以看出,两条消息在语法结构上确实十分类似,只是中间包含的参数个数不同,说明两条消息很可能属于同一类型.

在表 1 所示的对齐算法相似性矩阵中,各项的取值基于如下考虑:即已经明确类型的字段需与同类型字段匹配,未明确类型的字段可以与任意类型字段匹配.同时,为了过滤过短的匹配,我们取匹配符号百分比 Pm>80%作为成功匹配与否的阈值.通过字段类型抽象和序列匹配,我们可以处理由于可选字段的出现与否以及参数数量不同导致的具体消息格式上的差异,只根据消息的宏观结构进行归类.由于语法格式上相近的消息,其字段蕴含的程序行为语义仍然可能存在巨大差异,因此,对于语法上判断为同一类的消息,我们再利用语义信息作进一步的判定.如算法 2 所示,根据命令和参数字段关联的程序行为语义,我们将行为相似的消息归为一类.

图 2(a) .http.update message⇒*ZDZDCDCDPDPDPDC*
图 2(b) .ftp.update message⇒*ZDZDCDCDPDPDPDPDPDC*
C:命令,*D*:分隔符,*P*:参数,*Z*:其他字段类型

ZDZDCDCDPDPDPDPDPDC
ZDZDCDCDPDPDPD - - - - C

(a) Agobot .http.update and .ftp.update abstract message structures (b) Alignment of field type sequence
(a) Agobot .http.update 和.ftp.update 抽象消息格式 (b) 字段类型序列对齐示例

Fig.3 Agobot abstract messages

图 3 Agobot 抽象消息实例

Table 1 Similar matrix of Needleman-Wunsch algorithm

表 1 Needleman-Wunsch 算法相似性矩阵

---	*C*	*D*	*P*	*Z*
C	5			
D	–5	4		
P	1	–3	3	
Z	0	0	1	2

算法 2. 行为语义相似性判定算法.

输入:语法结构相似的消息集合 $M=\{m_i|i=1,\ldots,s\}$;

输出:行为予以相似的消息类别集合$\Theta=\{\theta_k=\{m_k,\ldots,m_j,\ldots\}|1\leqslant k,j\leqslant s,k\neq j\}$.

```
Classifier(M) {
  Θ={};
  for (i=1; i≤n; i++) θ_i={m_i};
  for (i=1; i<n; i++) {
    if (m_i 没有相关的行为语义) continue;
    for (j=i+1; j≤n; j++) {
      if (m_j 没有相关的行为语义) continue;
      λ=m_i 的行为语义 API 序列集合;
      μ=m_j 的行为语义 API 序列集合;
      if (|λ∩μ|>|λ∪μ|/2) {θ_i=θ_i∪m_j;θ_j=θ_j∪m_i;}
    }
  }
}
```

通过语法和行为语义相结合的结果归类,我们可以利用同类消息相互对照,对类中出现的字段语义作进一步的确认.同时,由于归类方法与字段具体内容无关,归类结果还可以用来确认由长度指示确定的可变长度字段识别是否正确.

3 实验评估

为了评估上述分析方法的有效性,我们在硬件模拟工具 QEMU[28]的基础上实现了原型系统 Prama,并在此基础上对若干恶意软件样本进行了实例分析,对分析结果进行了相应的说明,并对分析过程中发现的问题进行了讨论.

3.1 实 现

我们的原型系统由虚拟运行环境构建模块、分析控制模块和数据分析引擎 3 个模块组成,如图 4 所示.我们扩展了虚拟 CPU 模块,增加了虚拟 CPU 监视器用于识别当前执行的指令,提取 CPU 寄存器的内容.同时,还扩展了虚拟内存模块,用于监视虚拟内存的读写操作,提取指定虚拟内存区域的内容.控制模块按照监控配置和数据流分析逻辑,根据当前虚拟 CPU 的指令执行情况更新虚拟 CPU 和虚拟内存监视器的状态,标记数据并抓取相关内容.最后,数据分析引擎对获取的 API 拦截记录和污点传播记录进行分析,输出分析结果.

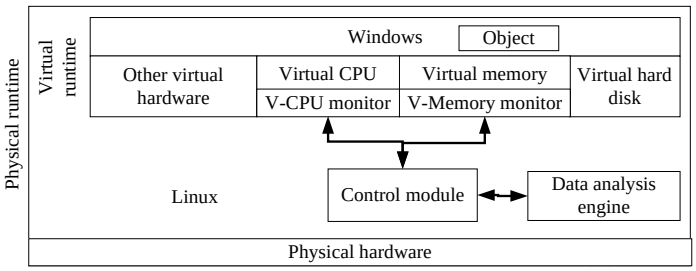


Fig.4 Structure of prototype system

图 4 原型系统结构示意图

由于指令分析过程需要CPU以单步模式运行,为了提高原型系统的运行效率,我们利用虚拟CPU监视器检测虚拟CPU的当前执行环境,在分析目标运行时动态切换虚拟CPU到模拟单步执行模式,从而避免了使用虚拟CPU内部的单步标志造成的分析性能下降问题的出现.

其次,虚拟CPU监视器还用于实现API拦截机制.我们在操作系统将程序和依赖库映射进内存之后,执行入口指令之前,分析并获得该程序导入的所有 API 地址,然后在运行时根据指令的 EIP,判断是否需要调用对应的API数据提取过程,提取所需数据,并在该API调用返回时提取返回值.通过这种方式,我们实现了在虚拟系统外部拦截虚拟系统内部的 API 调用.目前,我们的原型系统能够拦截字符串、文件、网络、进程、内存、注册表和系统服务等操作的主要API函数.

此外,在进行如第2.3节所述的自保护软件的内存镜像获取时,由于受操作系统内存分页、换页机制的影响,某些页面可能不在虚拟内存中.因此,我们在镜像获取时通过在当前指令执行流中强行插入读取已换出页面内容的指令,迫使操作系统换入相关页面,从而避免抓取的镜像不完全.然后从插入点之前重新开始执行,以确保执行流的正确和完整.与传统的通过查找和拦截原始入口点(OEP)进行脱壳处理的方式相比,我们的方法更简单,并能用统一的方式处理其他软件保护机制.

通过直接操作虚拟CPU、虚拟内存等虚拟设备获取虚拟运行环境中的数据的方法,我们的分析工具无需对被分析软件进行任何修改,也不依赖于虚拟系统内部的各种硬件和软件调试功能,这使得我们可以分析包含反调试、反跟踪功能的软件.对于恶意软件分析,分析工具对分析目标的高透明性可以确保恶意软件无法感知到外部监控的存在,能够表现出在物理环境中的真实行为,避免发生现有的基于调试器等的方法容易被恶意软件察觉并误导的情况.

3.2 实例分析

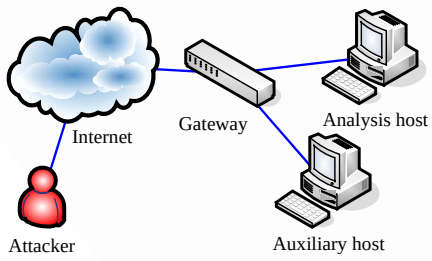


Fig.5 Network topology of case study

图 5 实验网络环境示意图

我们利用 Prama 原型系统对若干软件进行了实例分析.具体分析环境配置如下:运行Prama的分析主机配备了Intel Core2 Duo 3.00 GHz的E8400 CPU,4 GB内存和64位的Fedora Core 10 Linux 系统,同时在 Prama 的虚拟机器上安装了 32 位的Windows XP系统作为分析目标的运行环境.实验网络环境如图5所示,为了防止外部网络对分析过程产生干扰,我们将分析主机产生的除 DNS 查询之外的所有连接重定向到实验辅助主机,并在辅助主机上开启了 HTTP,SMTP,FTP,TFTP,IRC,TELNET,SSH等网络服务以配合分析.

根据协议类型及通信流量是否加密,我们挑选不同的恶意软件样本进行分析.由于恶意软件的特殊性,没有可供对照的协议规范和软件说明文档,只能通过人工确认对分析结果的准确性进行判断.结果表明,我们的方法能够正确识别大多数协议字段,提取的程序行为语义也基本符合实际操作含义.在此,我们详细分析 Win32/ Agobot3.BJ 和 Win32/FTP.Mini.A(样本名经 ESET Smart Security

v3.0.669.0 识别获得)两个样本的实验结果.

A) Win32/Agobot3.BJ

样本 A 是一个典型的基于 IRC 协议的僵尸程序.现有资料显示,Agobot 族的恶意软件有数量繁多的变种,提供数十种控制命令,具有多种方式的 DDoS 和网络扫描功能以及 HTTP 和 FTP 两种方式的程序下载执行功能,包含多种漏洞攻击代码,并且支持通信流量加密.该样本经 PeiD(PEiD 0.95,http://peid.has.it/)识别为 UPX 0.89.6-1.02/1.05-2.90 加壳程序,其具体分析结果见表 2.

Table 2 Analysis result of Win32/Agobot3.BJ

表 2 Win32/Agobot3.BJ 分析结果

Received network data							
No. of packets	1 377	No. of messages	148	No. of Agobot commands (manual analysis result)			65
Result of static analysis							
No. of candidate fields	7 055	No. of overlapped fields with dynamic analysis result			76	Percentage	1.02%
Result of dynamic analysis							
No. of fields	1 022	No. of different fields		126	No. of command fields		78
No. of message fields							
Average No. of fields per message		3.89 (exclude separators)			7.32 (include separators)		

在分析过程中,我们总共捕捉到了 1 377 个分析样本相关的网络数据包,其中包括样本的命令控制协议数据包、HTTP 和 FTP 协议数据包以及 TCP 应答等数据包.对程序内存镜像的静态分析总共获得了 7 055 条字串,其中有 76 条出现在了动态分析结果中,占静态分析结果数的 1.02%.动态分析过程中总共记录了 1 022 个字段,其中不同的字段数有 126 个,排除分隔符和参数后的命令字段集包含 78 个字段.通过人工检查 Prama 产生的分析记录数据,我们发现其中包括 65 个样本私有协议命令字段,此外还包含了 PING,NOTICE,PRIVMSG,KICK,NICK,PART 和 QUIT 共 7 个标准的 IRC 协议命令字段,.EDU..Edu..edu 和 id 共 4 个固定的参数字段以及 v3 和昵称共 2 个可变的参数字段.命令字段识别的正确率为 92.3%,说明我们的分析方法具有很高的准确性.这主要是由于,当样本接收到错误的命令时会将该命令与所有实现的命令字进行穷举比较,使得我们可以识别出未知的命令字段.

同时还发现,静态分析提取的字符串中包含了大量 API 名、格式化输出控制字符串和注册表项等字符串常量,但不包含空格“ ”和空格冒号“:”等短分隔符字符串.这是由于我们用来提取字符串的工具 strings(strings version 2.18.50.0.9-8.fc10 20080822,GNU Binutils,http://www.gnu.org/software/binutils/)没有将这类短字符序列作为字符串提取,但它们作为参数对污点源数据的分割操作仍然被 API 调用分析所截获,从而被识别.

该样本的分析结果还显示出,与直观感受不同,程序将形如“http.update”的命令作为一个字段统一处理,而没有进一步拆分.在如图 6 所示的消息中,字段“#Bot”没有被使用,字段“.”(点号)被提取但没有被使用,因此被标记为未确定类型字段.另外,我们还发现,http.xxx 和 ftp.xxx 类别命令的 API 调用栈与 bot.xxx 类别命令的调用栈明显不同,并且前两者具有类似的执行环境,只在最上层调用点地址上有小的偏移.因此,两个系列的命令很可能由同一块代码处理,人工对获取的程序内存镜像文件的分析证实了这一点.

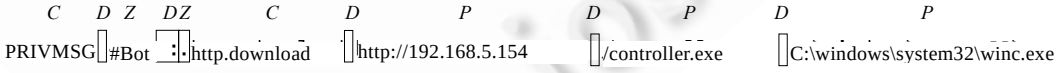


Fig.6 Message parsing of Win32/Agobot3.BJ

图 6 Win32/Agobot3.BJ 消息解析示意图

通过实验,我们还发现,分析记录中有大量的全局唯一污点标识 i 不同但数据包标记 p 相同的污点源标记 $\langle i,p\rangle$,说明样本对接收到的数据包 p 的内容进行了多次的解密操作.通过人工分析样本代码可知,程序没有存储解密后的数据,而是在每次需要使用时重新解密原始数据,因此触发了大量的重复解密操作,导致产生大量对应于同一个原始网络数据包的不同污点源标记.

此外还注意到,命令字段 ddos.httpflood 和 ddos.synflood 都有关联的网络相关行为,而语法判定相似的 ddos.pingflood,ddos.udpflood,ddos.spudpflood 和 ddos.stop 字段却没有,导致行为语义分析将其判定为两类.直接

检查 API 拦截记录确认,样本在处理前两个命令时产生了大量的 *send*(),*recv*()调用,而处理后 4 个命令时却没有,表明两类命令确实具有不同的行为语义.人工分析样本代码后发现,ddos.stop 只会导致程序修改某个状态字,而另外 3 个命令将导致处理过程直接返回.也就是说,该样本没有实现这 3 个命令的功能.

B) Win32/FTP.Mini.A

样本 B 是一个具有 FTP 服务器功能的后门软件,支持非加密的 FTP 协议,具有常见的文件上传、下载、更名、删除以及目录创建、删除等功能.借助标准的 FTP 客户端软件,我们对该恶意软件样本实现的 FTP 控制协议进行了分析.

从表 3 的分析结果中我们可以看出,样本 B 实现的 FTP 控制协议消息主要由命令字段构成(31/46).并且与样本 A 的分析结果相比,样本 B 的静态分析提取的候选字段数与动态分析结果重叠比例较高.我们认为,这主要与该样本功能较为单一,包含较少的导入 API 名、格式化输入/输出等控制字符串有关.

Table 3 Analysis result of Win32/FTP.Mini.A

表 3 Win32/FTP.Mini.A 分析结果

Received network data							
No. of packets	413	No. of messages	162	No. of FTP commands (manual analysis result)			33
Result of static analysis							
No. of candidate fields	415	No. of overlapped fields with dynamic analysis result		31	Percentage		7.46%
Result of dynamic analysis							
No. of fields	723	No. of different fields	46	No. of command fields			31
No. of message fields							
Average No. of fields per message		1.99 (exclude separators)		3.98 (include separators)			

由于 FTP 协议比较规范,我们的分析工具正确识别出了实验过程中观察到的所有协议字段.通过与 FTP 协议标准 RFC 959(RFC 959-FILE TRANSFER PROTOCOL(FTP),http://tools.ietf.org/html/rfc959)中给出的 33 个命令进行对比,我们发现分析结果中出现了 26 个标准给出的命令,有 7 个标准给出的命令没有实现.同时,该样本实现了 5 个 RFC 959 协议标准中未给出的命令,详细结果见表 4.

虽然我们在分析过程中只向分析样本发送了符合 RFC 959 协议标准的消息,但由于样本程序在识别命令字段时的一系列比较操作,我们的分析工具成功地识别出了该样本的 FTP 协议实现中的非标准命令,从而可以利用分析结果评估样本的协议实现情况.

通过分析字段的行为语义关联结果,我们获得了该样本实现的协议标准命令字段的行为语义序列,见表 5.从中可以发现,以 API 序列表示的行为特征确实可以用于表达命令字段的操作含义,并且符合外部观察到的程序行为表现.另外,从样本使用的 *PostThreadMessageA*(),*PeekMessageA*()等 API 还可以看出,该样本是基于 Windows 消息循环架构的多线程程序.

Table 4 Analysis result of command fields of Win32/FTP.Mini.A

表 4 Win32/FTP.Mini.A 命令字段分析结果

Types	Commands
Supported commands in RFC 959	ABOR, APPE, BYE, CDUP, CWD, DELE, HELP, LIST, MKD, NLST, NOOP, PASS, PASV, PORT, PWD, QUIT, REST, RETR, RMD, RNFR, RNTO, SIZE, STOR, SYST, TYPE, USER
Supported commands not in RFC 959	P@SW, XCWD, XMKD, XPWD, XRMD
Unsupported commands in RFC 959	ACCT, MODE, REIN, SITE, SMNT, STAT, STRU

Table 5 Behavior semantics of command fields of Win32/FTP.Mini.A (partial)

表 5 Win32/FTP.Mini.A 命令字段行为关联(部分)

Commands	API sequences	Explanation
LIST	*FindFirstFileA, FindNextFileA, FindClose, lstrcpyA, PostThreadMessageA, PeekMessageA, TranslateMessage, DispatchMessageA, send*	Get a list of files and subdirectories under current directory
STOR	*CreateFileA, recv, WriteFile, PostThreadMessageA, PeekMessageA, TranslateMessage, DispatchMessageA, send*	Upload a file
RNTO	*MoveFileA, PostThreadMessageA, PeekMessageA, TranslateMessage, DispatchMessageA, send*	Rename or move a file

3.3 讨 论

由于无法获得 Prospex[13],Polyglot[14]和 Tupni[15]的协议规范逆向分析工具,因此无法利用这些方法对我们的恶意软件样本进行分析.但根据相关文献的描述可以发现,这些方法与本文的方法有两个重要的区别:首先,本文的方法针对的是恶意软件,因此在设计和实现中都考虑了恶意软件包含自我保护、分析环境检测等特点.而上述方法都显式或隐式地将分析对象定为不包含此类对抗技术的普通软件,因此可以预见,这类方法难以全面和有效地分析僵尸程序等恶意软件的通信协议;其次,我们的分析结果不仅包含协议消息格式、字段类型等协议语法和语义信息,还包含协议字段对应的恶意软件行为.由于程序混淆和代码保护等反分析技术的干扰,恶意软件的分析难度要远大于普通软件.因此,我们的分析方法获取的消息格式和协议语法的精度可能会低于上述方法;但我们的方法能够关联字段到对应的程序行为语义,能够揭示恶意软件的网络命令-程序响应关系,是网络协议理解和恶意软件行为分析的重要基础,而这些信息是上述分析方法所无法提供的.

其次,我们的方法具有较高的分析效率.分析效率是衡量动态分析方法的一项重要指标,由于没有可供对比分析的数据,我们以代码执行时间来表示该时间段内执行的指令数,通过分别记录样本在受监控的 API 中执行的时间和在其他代码中的执行时间来与单纯的指令分析方法进行比较,衡量分析效率的提高程度.在上一节的两个样本分析过程中,样本 A 约有 58%的代码运行时间消耗在了受监控的 API 中,样本 B 有 27.5%.由于与指令数增长导致的分析开销增加相比,引入 API 拦截机制的开销很小,因此本文的分析方法比单纯的指令分析方法在效率上提高了 20%~50%.其中,样本 B 的分析效率提高较少,我们认为是该样本采用的 MFC 框架对底层 API 做了各种封装,引入了大量中间代码导致.

在分析能力方面,由于针对可执行代码的软件分析方法可能受到不同编译器的代码生成策略的影响,我们对利用 MSVC 6.0-9.0,MinGW GCC v3.4.5,LCC-Win32 v3.8 和 Borland BCC v5.5.1 这 4 类主流的 C/C++编译器产生的程序进行了分析实验.结果表明,我们的分析方法能够处理大多数这些编译环境产生的程序,但分析效率容易受到编译器种类和编译选项的影响.虽然我们的方法主要基于程序代码运行逻辑和底层的操作系统机制,但像 MSVC 编译器的/MT 选项会造成部分运行库函数被静态编译进入程序,使得 API 拦截机制无法监控到这些库函数,导致指令分析复杂度提高.

我们也注意到,恶意软件作者可以通过自己实现一些 API 函数的功能,减少 API 调用以躲避分析.由于一些核心功能,如网络操作、进程创建等很难通过不利用系统 API 来实现,并且会导致软件规模膨胀,因此难以完全绕过我们的分析方法.同时,我们也将在后续研究中加强指令分析系统,尝试通过指令分析提取和还原更多的信息以应对这种情况.另外,由于受到分析过程中观察到的网络活动种类和数量的限制,恶意软件的一些网络数据解析和字段使用过程有可能未被触发,可能导致我们的分析结果不够全面,这是网络协议逆向分析中普遍存在的问题.我们可以利用已有分析结果,通过 Fuzz 方法主动触发网络交互过程来提高分析的覆盖率,弥补现有分析模式的不足.

目前,我们的分析主要针对恶意软件接收的网络数据的处理过程,利用同样的方法分析网络交互对端,则可以获取双向通信过程的信息.并且,由于恶意软件接收网络数据和读取本地文件都是处理外部输入数据,文件格式也具有类似通信协议的规范.因此我们相信,经过适当的扩展,我们的方法也可以用于处理恶意软件的配置文件解析等本地文件数据处理过程.

4 总 结

网络协议逆向分析和程序行为分析是恶意软件分析的重要组成部分,分析人员不仅需要掌握恶意软件通信协议的语法规范,更需要了解协议字段对应的程序行为语义,只有这样才能制定出有效的恶意软件防治措施.本文提出了一种利用动态程序分析技术进行网络协议逆向分析的方法,该方法通过在可控的虚拟执行环境中运行和监视恶意软件执行过程,跟踪恶意软件对网络数据的处理流程,识别和划分通信协议字段,提取协议语法信息,并关联协议字段到对应的程序行为.在我们实现的原型系统上的实例分析表明,本文提出的方法能够较好地提取文本协议的语法信息,同时,字段关联的程序行为语义也能够准确地反映其真实含义.利用本文方法获取

的结果,可以作为恶意软件行为特征提取、网络流量特征提取和开展恶意软件网络渗透的基础.下一步,我们计划进一步加强现有的指令分析系统,提高污点传播实现的精度,以提高复杂字段的识别能力,并研究针对二进制协议和本地文件的相关分析方法.

References:

[1] Moser A, Kruegel C, Kirda E. Limits of static analysis for malware detection. In: Proc. of the 23rd Annual Computer Security Applications Conf. (ACSAC 2007). Miami Beach: IEEE Computer Society, 2007. 421−430. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4413008&tag=1 [doi: 10.1109/ACSAC.2007.21]

[2] Cui WD, Kannan J, Wang HJ. Discoverer: Automatic protocol reverse engineering from network traces. In: Proc. of the 16th USENIX Security Symp. (Security 2007). Berkeley: USENIX Association, 2007. 199−212. http://portal.acm.org/citation.cfm?id=1362917

[3] Ma J, Levchenko K, Kreibich C, Savage S, Voelker GM. Unexpected means of protocol inference. In: Proc. of the 6th ACM SIGCOMM Conf. on Internet Measurement (IMC 2006). New York: ACM, 2006. 313−326. http://portal.acm.org/citation.cfm?id=1177123 [doi: 10.1145/1177080.1177123]

[4] Small S, Mason J, Monrose F, Provos N, Stubblefield A. To catch a predator: A natural language approach for eliciting malicious payloads. In: Proc. of the 17th USENIX Security Symp. (Security 2008). Berkeley: USENIX Association, 2008. 171−183. http://portal.acm.org/citation.cfm?id=1496723

[5] Kruegel C, Robertson W, Valeur F, Vigna G. Static disassembly of obfuscated binaries. In: Proc. of the 13th Conf. on USENIX Security Symp. (Security 2004). Berkeley: USENIX Association, 2004. 18. http://portal.acm.org/citation.cfm?id=1251393

[6] Vigna G. Static disassembly and code analysis. In: Christodorescu M, ed. In: Christodorescu M, Jha S, Maughan D, Song D, Wang C, eds. Proc. of the Malware Detection. Washington: Springer-Verlag, 2007. 19−41.

[7] Christodorescu M, Kidd N, Goh WH. String analysis for x86 binaries. In: Proc. of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE 2005). New York: ACM, 2005. 88−95. http://portal.acm.org/citation.cfm?id=1108814 [doi: 10.1145/1108792.1108814]

[8] Cavadini S. Secure slices of insecure programs. In: Proc. of the 2008 ACM Symp. on Information, Computer and Communications Security (ASIACCS 2008). New York: ACM, 2008. 112−122. http://portal.acm.org/citation.cfm?id=1368329 [doi: 10.1145/1368310.1368329]

[9] Linn C, Debray S. Obfuscation of executable code to improve resistance to static disassembly. In: Proc. of the 10th ACM Conf. on Computer and Communications Security (CCS 2003). New York: ACM Press, 2003. 290−299. http://portal.acm.org/citation.cfm?id=948149 [doi: 10.1145/948109.948149]

[10] Nethercote N, Seward J. Valgrind: A framework for heavyweight dynamic binary instrumentation. In: Proc. of the ACM Conf. on Programming Language Design and Implementation (PLDI 2007). New York: ACM, 2007. 89−100. http://portal.acm.org/citation.cfm?id=1250746 [doi: 10.1145/1250734.1250746]

[11] Luk CK, Cohn R, Muth R, Patil H, Klauser A, Lowney G, Wallace S, Reddi VJ, Hazelwood K. Pin: Building customized program analysis tools with dynamic instrumentation. In: Proc. of the 2005 ACM Conf. on Programming Language Design and Implementation (PLDI 2005). New York: ACM, 2005. 190−200. http://portal.acm.org/citation.cfm?id=1065034 [doi: 10.1145/1065010.1065034]

[12] Aaraj N, Raghunathan A, Jha NK. Dynamic binary instrumentation-based framework for malware defense. In: Proc. of the 5th Int'l Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2008). Berlin, Heidelberg: Springer-Verlag, 2008. 64−87. http://portal.acm.org/citation.cfm?id=1428328 [doi: 10.1007/978-3-540-70542-0_4]

[13] Comparetti PM, Wondracek G, Kruegel C, Kirda E. Prospex: Protocol specification extraction. In: Proc. of the IEEE Symp. on Security & Privacy. Washington: IEEE Computer Society, 2009. 110−125. http://www.computer.org/portal/web/csdl/doi/10.1109/SP.2009.14 [doi: 10.1109/SP.2009.14]

[14] Caballero J, Yin H, Liang ZK, Song D. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In: Proc. of the 14th ACM Conf. on Computer and Communications Security (CCS 2007). New York: ACM, 2007. 317−329. http://portal.acm.org/citation.cfm?id=1315286 [doi: 10.1145/1315245.1315286]

[15] Cui WD, Peinado M, Chen K, Wang HJ, Irun-Briz L. Tupni: Automatic reverse engineering of input formats. In: Proc. of the 15th ACM Conf. on Computer and Communications Security (CCS 2008). New York: ACM, 2008. 391−402. http://portal.acm.org/citation.cfm?id=1455820 [doi: 10.1145/1455770.1455820]

[16] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Proc. of the Network and Distributed System Security Symp. (NDSS 2005). San Diego, 2005. http://www.valgrind.org/docs/newsome2005.pdf [doi: 10.1.1.62.8372]

[17] Egele M, Kruegel C, Kirda E, Yin H, Song D. Dynamic spyware analysis. In: Proc. of the USENIX Annual Technical Conf. Berkeley: USENIX Association, 2007. 233−246. http://portal.acm.org/citation.cfm?id=1364403

[18] Saxena P, Sekar R, Puranik V. Efficient fine-grained binary instrumentation with applications to taint-tracking. In: Proc. of the 6th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO 2008). New York: ACM, 2008. 74−83. http://portal.acm.org/citation.cfm?id=1356069 [doi: 10.1145/1356058.1356069]

[19] Huang Y, Stavrou A, Ghosh AK, Jajodia S. Efficiently tracking application interactions using lightweight virtualization. In: Proc. of the 1st ACM Workshop on Virtual Machine Security (VMSec 2008). New York: ACM, 2008. 19−28. http://portal.acm.org/citation.cfm?id=1456486 [doi: 10.1145/1456482.1456486]

[20] Bayer U, Kruegel C, Kirda E. TTAnalyze: A tool for analyzing malware. In: Proc. of the 15th Annual Conf. of the European Institute for Computer Antivirus Research (EICAR 2005). Hamburg, 2006. http://www.iseclab.org/papers/ttanalyze.pdf

[21] Dinaburg A, Royal P, Sharif M, Lee W. Ether: Malware analysis via hardware virtualization extensions. In: Proc. of the 15th ACM Conf. on Computer and Communications Security (CCS 2008). New York: ACM, 2008. 51−62. http://portal.acm.org/citation.cfm?id=1455779 [doi: 10.1145/1455770.1455779]

[22] Wondracek G, Milani P, Kruegel C, Comparetti P, Kirda E. Automatic network protocol analysis. In: Proc. of the 15th Annual Network & Distributed System Security Symp. (NDSS 2008). San Diego, 2008. http://www.cs.ucsb.edu/~chris/research/doc/ndss08_protocol.pdf

[23] Lin ZQ, Jiang XX, Xu DY, Zhang XY. Automatic protocol format reverse engineering through context-aware monitored execution. In: Proc. of the 15th Annual Network & Distributed System Security Symp. (NDSS 2008). San Diego, 2008. http://www.isoc.org/isoc/conferences/ndss/08/papers/14_automatic_protocol_format.pdf [doi: 10.1.1.120.2651]

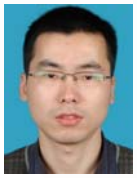
[24] Kaksonen R, Laakso M, Takanen A. Vulnerability analysis of software through syntax testing. 2000. https://www.ee.oulu.fi/research/ouspg/PROTOS_WP2000-robustness

[25] Sreedhar VC, Gao GR, Lee YF. Identifying loops using DJ graphs. ACM Trans. on Programing Language Systems, 1996,18(6): 649−658. [doi: 10.1145/236114.236115]

[26] Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA. A sense of self for Unix processes. In: Proc. of the '96 IEEE Symp. on Security and Privacy (S&P'96). Washington: IEEE Computer Society, 1996. 120−128. http://portal.acm.org/citation.cfm?id=884258 [doi: 10.1109/SECPRI.1996.502675]

[27] Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 1970,48(3):443−453. [doi: 10.1016/0022-2836(70)90057-4]

[28] Bellard F. QEMU, a fast and portable dynamic translator. In: Proc. of the 2005 USENIX Annual Technical Conf. (ATEC 2005). Berkeley: USENIX Association, 2005. 41−46. http://portal.acm.org/citation.cfm?id=1247401



应凌云(1982－),男,浙江永康人,博士,助理研究员,主要研究领域为恶意代码分析与防范.

冯登国(1965－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为密码学,信息安全.

杨轶(1982－),男,博士生,主要研究领域为恶意代码分析与防范.

苏璞睿(1976－),男,博士,副研究员,主要研究领域为恶意代码分析与防范.