ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.4, April 2010, pp.802−815
doi: 10.3724/SP.J.1001.2010.03514

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

推测网络蠕虫传播路径的在线聚积算法*

李 强[1,2+], 向 阳[1,2]

[1](吉林大学 计算机科学与技术学院,吉林 长春 130012)

[2](吉林大学 符号计算与知识工程教育部重点实验室,吉林 长春 130012)

Algorithm of Online Accumulation for Reconstructing the Path of Worm Propagation

LI Qiang[1,2+], XIANG Yang[1,2]

[1](College of Computer Science and Technology, Jilin University, Changchun 130012, China)

[2](Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China)

+ Corresponding author: li_qiang@jlu.edu.cn

Li Q, Xiang Y. Algorithm of online accumulation for reconstructing the path of worm propagation. *Journal of Software*, 2010,21(4):802−815. http://www.jos.org.cn/1000-9825/3514.htm

Abstract: Tracing online propagation paths when worm breaks out on a large scale can improve the network's anti-attackability. The existing tracing approaches to obtain worm propagation path are all based on off-line analysis and usually have a lower accuracy. This paper proposes an online Accumulation Algorithm with sliding detection windows, which can fleetly and efficiently trace the origin and initial causal edges of the worm. The algorithm solves the conflicts in choosing causal edges and tackles the problem of merging propagation paths in the consecutive reconstruction phase. The algorithm's accuracy and performance have been analyzed. Experimental results reveal that the online Accumulation Algorithm can dig out causal edge even at the initial stage, and the Accumulation Algorithm can achieve detection accuracy higher than 90% while its running time is only 1% of related works.

Key words: worm; propagation path; online tracing; detection window

摘 要: 在大规模网络蠕虫爆发时获取蠕虫的传播路径,可以提高网络的抗打击能力.现有的推测蠕虫传播路径方法只能运行于离线方式且准确率较低.提出了使用滑动检测窗口推测网络蠕虫传播路径的在线聚积算法,可快速获取网络蠕虫的传播源和初期传播路径.解决了传播路径选择冲突和相邻推测阶段传播路径合并等问题.分析了算法的准确率和运行性能.实验结果表明,在线聚积算法在蠕虫爆发初期即可检测出感染边,聚积算法具有 90%以上的准确率,所需路径推测时间只有同类工作的 1%.

关键词: 蠕虫;传播路径;在线追踪;检测窗口

中图法分类号: TP393 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60703023 (国家自然科学基金); the Science and Technology Development Plan of Jilin Province of China under Grant No.20080108 (吉林省科技发展计划资助项目)

Received 2008-02-29; Accepted 2008-10-27

自我繁殖是恶意代码的一个重要特征[1,2].网络蠕虫作为一类主要的恶意代码,拥有很强的自我繁殖能力,对网络和端系统的安全威胁正日益增加.新型智能蠕虫和多态蠕虫不断出现,多样化的传播途径和复杂的应用环境使网络蠕虫的发生频率增高,潜伏性变强,覆盖面更广,造成的损失也更大[1−3].网络蠕虫的爆发可以使攻击者在很短的时间内控制大量主机,发动 DDoS 攻击、窃取机密信息和破坏关键数据,因而每次网络蠕虫爆发必定会造成极大的危害.在传统的网络蠕虫防御方法中,对主机和网络的修补和加固通常需要预先获取蠕虫或漏洞的相关信息.而获取网络蠕虫的传播路径(即追踪蠕虫的攻击路径)[4−6]不仅可以推测最早的被感染的节点,还可以推测在传播过程中造成其他节点被感染的传播路径.即使只获取到部分路径,对抑制蠕虫继续传播和调查取证也具有重大意义.但迄今为止,即使蠕虫使用简单的扫描策略,获取它的感染序列也是一项极为困难的任务.

现有的蠕虫传播路径获取方法采用的都是离线分析方式,即在蠕虫攻击爆发之后,经过一段采集流量的时间,通过获取的网络流量数据进行分析得到推测结果,很难做到实时地追踪(即几乎可在网络蠕虫攻击过程中获得传播路径).这类离线方法虽然最后能够获取蠕虫的传播路径和攻击源,但不能在蠕虫爆发之时就获取传播路径,而且不能显示蠕虫传播路径时随时间的动态变化情况.因此,有必要研究在复杂网络环境下在线追踪网络蠕虫的方法,用于接近实时地追踪网络蠕虫初始传染源,抑制蠕虫的继续传播,保证更多主机不被网络蠕虫传染.

蠕虫在主机间和网络内传播,需要扫描目标和感染目标,产生通信的流量.对大规模蠕虫执行在线追踪,就是要获取被蠕虫传染的主机和网络设备的先后序列,这个序列不能由主机自身发现或报告,只能通过捕获的网络流量进行分析,推测蠕虫的传播路径.为了实现在线追踪网络蠕虫,必须解决以下几个问题:(1) 缩短蠕虫传播路径推测时间,减少计算复杂度;(2) 解决推测结果集中路径的冲突和合并问题;(3) 保证连续时间段的传播路径推测.

本文提出了使用滑动窗口在线追踪网络蠕虫的方法,分析了其准确率和运行性能,并通过大规模网络蠕虫爆发的模拟实验进行了验证.本文的主要工作包括:提出了蠕虫追踪的聚积算法,可快速获取网络蠕虫的传播源和初始传播路径,计算复杂度与数据规模成正比;解决了传播路径选择冲突和相邻推测阶段传播路径合并等问题,能够有效地提高准确率、降低误报率和漏报率;将追踪过程划分为连续时间的滑动检测窗口,提出了使用滑动检测窗口的在线聚积算法,可在网络蠕虫爆发时获取蠕虫传播源和初始传播路径;建立了蠕虫传播的模拟实验环境,验证算法运行性能.

本文第 1 节介绍蠕虫追踪的相关工作.第 2 节提出蠕虫追踪的聚积算法.第 3 节提出基于滑动检测窗口的在线聚积方法.第 4 节进行实验和分析.最后给出结论.

1 相关工作

现有的 IP 反向追踪技术[7]不适用于网络蠕虫传播路径的追踪.网络蠕虫在受害者间传播时,为了提高攻击成功率,通常使用受害者的真实地址发送攻击包.在分布式攻击中,绝大多数包的地址都不是攻击源,而仅仅是众多蠕虫受害者中的一个,所以需要寻找感染受害者的更早阶段的主机.跳板机反向追踪技术[8,9]也不适合蠕虫追踪.在现有跳板机检测方法中,基于内容的方法需要开销巨大的包内容分析,而且不能检测到多态蠕虫或者使用加密技术的蠕虫;基于会话的包间时序的关联方法也不适用网络蠕虫攻击,因为蠕虫并不使用交互式的会话.

到目前为止,只有少数方法可以用来离线检测网络蠕虫的感染路径.Xie 等人[5,10]提出了在主机关联图中进行随机行走的方法来检测蠕虫的攻击源.主机关联图是通过在蠕虫传播时收集到各个潜在受害者之间的流量而得到的.尽管该算法能够得到蠕虫传播的详细信息,包括初始的感染源以及互相感染的关系,但是它需要在记录相当长时间的网络全部流量之后才能开始追踪计算.此外,针对无线节点中传播的移动蠕虫的流量特点,Sarat 等人[11]改进了随机行走算法,使之能继续有效地工作.Rajab 等人[4]利用从网络望远镜得到的蠕虫监测历史数据来推断蠕虫的感染序列.如果一个被感染机器的扫描包比另一个被感染机器的扫描包提前到达监视器,从直觉上可以推断出前者比后者更早地被感染.然而,扫描过程中所固有的随机性、相对于易感染机器数量的望远镜的大小以及不同状态下蠕虫传播的混杂性等因素,会直接影响到推断出来的结果的可靠性.而且,当到达望远镜的扫描包的时间间隔减少的时候,将会更加难以准确地检测到实际的感染序列.Kumar 等人[6]提出了另一种方

法,主要通过对 witty 蠕虫的逆向工程方法来分析它的随机扫描算法和相对应的随机种子.如果有了目标的选择算法,就可以重现检测序列,并得到蠕虫传播的具体视图以及被感染机器的一些特征.尽管该算法所需的信息可以在本地复原,但是该算法很难推广到其他的蠕虫,因为每次检测都要经过艰巨的逆向工程过程.Collins 等人[12]使用协议图,通过监控各种协议图的异常变化,检测目标列表扫描蠕虫和发起该蠕虫攻击的主机.

2 传播路径快速推测

2.1 假设和定义

参考文献[5]的部分定义,将网络中的主机间通信定义为一个有向图 $G=\langle V,E\rangle$,称其为主机联系图.其中图的点集 $V=H\times T$,H 是网络中主机的集合,T 表示时间;图的边集 E 是 $V\times V$ 的一个子集.图 G 中的一条有向边 $e=\langle u,t^s,v,t^e\rangle$表示网络中的一个流.其中,$\langle u,t^s\rangle\in H\times T$ 表示流的源主机和开始时间,$\langle v,t^e\rangle\in H\times T$ 表示流的目的主机和结束时间.如果一条边带有蠕虫攻击性,无论它是否成功地感染了目的主机,则均被称为攻击边.如果一条攻击边成功地感染了一台以前未被感染的目的主机,则被称为感染边.G 中除去攻击边之外的所有边称为正常边.

G 中两条边 $e_1=\langle u_1,t_1^s,v_1,t_1^e\rangle$,$e_2=\langle u_2,t_2^s,v_2,t_2^e\rangle$,若 $u_2=v_1$ 且 $t_1^e<t_2^s<t_1^e+\Delta t$($\Delta t$ 是预先设定的时间间隔参数),则称 e_2 是 e_1 的后继,e_1 是 e_2 的前驱.e 的前驱分别记为 $e_{pre}^1,...,e_{pre}^j,...,e_{pre}^{PRE(e)}$,其中,$PRE(e)$表示 e 的前驱个数.类似地,e 的后继分别记为 $e_{suc}^1,...,e_{suc}^k,...,e_{suc}^{SUC(e)}$,其中,$SUC(e)$表示 e 的后继个数.前驱和后继描述了边之间的相邻关系.

一般情况下,我们假定网络中只有一个蠕虫源,蠕虫的传染过程形成了一棵树(称为感染树).感染树中从树根到树叶的一条路径被称为感染链.感染树由 G 中所有的感染边构成.感染树的根代表蠕虫攻击源,位于树顶层的边表示蠕虫传染初期的感染路径.获取关于攻击源和初始传染序列的相关信息,对于抑制蠕虫传播和调查取证具有重大意义.在已知主机联系图 G 的情况下,我们的算法能够以较高的准确率找出感染树顶层中的一些边(即蠕虫的初始攻击序列).网络蠕虫的爆发可以在很短的时间内感染大量主机,这就要求蠕虫追踪算法能尽快地获取蠕虫传播路径以降低危害.同时,由于网络流量产生时速度快、规模大,所以要求追踪算法的时空复杂性至少是呈线性的.下面介绍蠕虫追踪的聚积算法,同时借鉴动态规划的思想对算法实现中的关键代码进行优化,即使对于百万级数据,算法也能在很短的时间内完成,为防御措施的部署争取更多的时间.

2.2 聚积算法

相比被感染之前,一个主机被蠕虫感染后,为了继续感染其他主机,会发出更多的攻击流,但接收到的流数量不会有显著增加[5].从平均意义上讲,感染边和正常边的前驱数量相似,但感染边有更多的后继.基于这样的差别,我们提出了蠕虫传播路径的推测方法——聚积算法.首先赋予每条边相同的权值;然后,算法通过 K 次权值“聚集-累积”过程(聚积过程),使得较多的权值聚积到感染边上;最后挑选权值最大的 Z 条边推测出蠕虫传播的初始序列,从而推测感染树的顶层部分.

设 $p(e,i)$为边 e 第 i 次聚积过程中的权增量,算法结束后,e 的总权值为 $p(e)=\sum_{i=1}^{K}p(e,i)$.每一次权值聚积的过程就是对上一次的权增量进行重新分配的过程.具体地,将上一次的权增量 $p(e,i-1)$均匀地分配给 e 的所有前驱 $e_{pre}^1,...,e_{pre}^j,...,e_{pre}^{PRE(e)}$,构成这些前驱边本次权增量的一部分.如此迭代 K 次,边的权增量不断地分配给它们的前驱.对权增量的重新分配过程,实际上也是权值在感染链上的逆聚积过程.算法执行过程如下:

1) $i=0$;$p(e)=0.0$;$p(e,0)=1.0$;

2) $i=i+1$;

$$p(e,i)=\sum_{j=1}^{SUC(e)}\frac{p(e_{suc}^j,i-1)}{PRE(e_{suc}^j)} \tag{1}$$

$p(e)=p(e)+p(e,i)$;

3) 如果 $i<K$,则转到第 2)步,否则,转到第 4)步;

4) 挑选出 $p(e)$最大的 Z 条边作为结果集(记为 ANS),推测蠕虫传播初期的感染树.

由公式(1)可知,若 e 没有前驱边,它的权增量 $p(e,i)$在下一次不会分配给其他边.也就是说,在权值重分配过程中,仅有少数没有前驱的边丢失了权增量,权增量的总和基本保持不变.即对于某确定的 i, $\sum_{e\in E} p(e,i)\approx|E|$.

每一次对权增量的调整只是一种重新分配,并没有产生新的权增量.我们将权增量重分配过程称为聚集,将权增量的加入称为累积.随着聚积过程的进行,权值沿着感染链逆向聚集到感染树的顶层,成为我们发现蠕虫初始攻击序列的主要依据.

2.3 算法实现与计算复杂性

在 K 次迭代过程中,依据第 i 次的权增量即可完全生成第 i+1 次的权增量.即“将来”的权增量仅依赖于“现在”而不依赖于“过去”.于是,权值重分配部分的算法实现可以借鉴动态规划的思想,依次计算出所有的 $p(e,i)(e\in E,1\le i\le K)$.算法的复杂度瓶颈主要存在于公式(1)的计算中.若能快速得到边 $e=\langle u,t^s,v,t\rangle$的权增量 $p(e,i)$,将会大为降低总的时间复杂度.e 的后继具有两个属性:源主机是 v;发生于之后Δt 秒内.

对所有的边按源主机分类,将源主机相同的边用链表连接.若要找出所有源主机为 v 的边,只需扫描 v 所对应的链表.按时间逆序依次计算每个 $p(e,i)$.维护一组指针 $end[]$,$end[v]$指向当前时间往后Δt 秒内最后一条以 v 为源主机的边.计算 $p(e,i)$时,只需知道 $end[v]$之前所有以 v 为源主机的边对 e 的权值贡献(设为 $sum[v]$).已知 $CurP[]$表示现有的权值分配,求下一次权增量重分配 $NextP[]$的算法如下:

1) $sum[]=\{0.0\}$;
2) 赋值 $end[]$,使 $end[v]$指向最后一条以 v 为源主机的边;
3) 逆序扫描所有的边,对于第 j 条边 e_j:$(u\to v)$,循环执行第 4)~第 6)步;
4) 更新 $end[v]$,将 $end[v]$沿源主机为 v 的边链表逆序向前移动,直到 $end[v]$指向的边成为当前边 e_j 的最后一个后继;
5) 更新 $sum[v]$,对第 4)步 $end[v]$前移过程中跨过的每一条边 e':$sum[v]-=CurP[e']/PRE[e']$,边 e_j 的下一次全增量 $NextP[j]$即为 $sum[v]$;
6) 更新 $sum[u]$,加入当前边 e_j 对其前驱的下一次权增量贡献:$sum[u]+=CurP[j]/PRE[j]$,其中 $PRE[j]$是边 e_j 的前驱个数.

首先,算法中用到的所有数组均为线性数组,因此,算法的空间复杂度为 $O(|E|)$.其次,在整个算法中,$end[v]$只向前移动,第 4)步和第 5)步中的更新过程只扫描了一次边表.另外,第 6)步中的 $PRE[]$可以由一个预处理过程在线性时间复杂度内全部计算出.于是,上述权增量重分配算法所需的时空复杂度均为 $O(|E|)$.聚积算法进行 K 次权值重分配,总时间复杂度为 $O(K\times|E|)$.实验中,K 取到很小的值即可得到很好的结果.因此可以认为,聚积算法拥有线性时空复杂度.

2.4 理论分析

若在 t^e 时(或此之前)目标主机被蠕虫感染,则称 $e=\langle u,t^s,v,t^e\rangle$为可疑边,记为 e_m.除此之外的边是普通边,记为 e_n.普通边一定是正常边,但可疑边中除了所有的感染边,还包括部分正常边和非感染性攻击边(v 在 t^e 之前已被感染).聚积算法试图以较高的准确率挑选出蠕虫传染初期的感染边并构建感染树.为了说明算法的准确率及其可行性,需要对网络中的流量和蠕虫的攻击建立简单的模型.

在统计平均意义下:一台正常主机Δt 时间内发出的流量数设为 A;一台被感染主机Δt 时间内发出的流量数设为 B(包括 A 条正常边和 $B-A$ 条攻击边,显然 $B>A$,且对于快速传播蠕虫,主机被感染后发出的流量数会显著增加,$B>>A$);感染前后一台主机在Δt 时间内接收到的流量数没有太大变化,均设为 C;一台主机在Δt 时间内发出的正常边中,可疑边的条数设为 X.

下面使用数学归纳法证明.

当 $B>A$ 时,对于任意 $1\le i\le K$ 都有 $E[p(e_m,i)]>E[p(e_n,i)]$.其中, $E[p(e_m,i)]=\dfrac{\sum_{e_m\in E} p(e_m,i)}{\sum_{e_m\in E}1}$, $E[p(e_n,i)]=\dfrac{\sum_{e_n\in E} p(e_n,i)}{\sum_{e_n\in E}1}$,

分别表示可疑边和普通边权增量的期望值.

1. 首先,当 i=1 时,$E[p(e_m,1)]>E[p(e_n,1)]$:

$$E[p(e_m,1)]=\frac{\sum_{e_m\in E}p(e_m,1)}{\sum_{e_m\in E}1}=\frac{\sum_{e_m\in E}\left(\sum_{j=1}^{SUC(e_m)}\frac{p((e_m)_{suc}^{j},0)}{PRE((e_m)_{suc}^{j})}\right)}{\sum_{e_m\in E}1}\approx\frac{\sum_{e_m\in E}\left(\frac{B}{C}\right)}{\sum_{e_m\in E}1}=\frac{B}{C},$$

$$E[p(e_n,1)]=\frac{\sum_{e_n\in E}p(e_n,1)}{\sum_{e_n\in E}1}=\frac{\sum_{e_n\in E}\left(\sum_{j=1}^{SUC(e_n)}\frac{p((e_n)_{suc}^{j},0)}{PRE((e_n)_{suc}^{j})}\right)}{\sum_{e_n\in E}1}\approx\frac{\sum_{e_n\in E}\left(\frac{A}{C}\right)}{\sum_{e_n\in E}1}=\frac{A}{C}.$$

因为$E[p(e_m,1)]-E[p(e_n,1)]\approx\frac{B-A}{C}>0$,所以$E[p(e_m,1)]>E[p(e_n,1)]$.

2. 假设在 $1\le i<K$ 时,$E[p(e_m,i)]>E[p(e_n,i)]$,下面证明 $E[p(e_m,i+1)]>E[p(e_n,i+1)]$:

$$E[p(e_m,i+1)]=\frac{\sum_{e_m\in E}p(e_m,i+1)}{\sum_{e_m\in E}1}=\frac{\sum_{e_m\in E}\left(\sum_{j=1}^{SUC(e_m)}\frac{p((e_m)_{suc}^{j},i)}{PREV((e_m)_{suc}^{j})}\right)}{\sum_{e_m\in E}1}$$

$$\approx\frac{\sum_{e_m\in E}\left(\frac{A-X}{C}\bullet E[p(e_n,i+1)]+\frac{X+(B-A)}{C}\bullet E[p(e_m,i+1)]\right)}{\sum_{e_m\in E}1}$$

$$=\frac{A-X}{C}\bullet E[p(e_n,i+1)]+\frac{X+(B-A)}{C}\bullet E[p(e_m,i+1)],$$

$$E[p(e_n,i+1)]=\frac{\sum_{e_n\in E}p(e_n,i+1)}{\sum_{e_n\in E}1}=\frac{\sum_{e_n\in E}\left(\sum_{j=1}^{SUC(e_n)}\frac{p((e_n)_{suc}^{j},i)}{PREV((e_n)_{suc}^{j})}\right)}{\sum_{e_n\in E}1}$$

$$\approx\frac{\sum_{e_n\in E}\left(\frac{A-X}{C}\bullet E[p(e_n,i+1)]+\frac{X}{C}\bullet E[p(e_m,i+1)]\right)}{\sum_{e_n\in E}1}$$

$$=\frac{A-X}{C}\bullet E[p(e_n,i+1)]+\frac{X}{C}\bullet E[p(e_m,i+1)].$$

因为$E[p(e_m,i+1)]-E[p(e_n,i+1)]\approx\frac{B-A}{C}\bullet E[p(e_m,i)]>0$,所以$E[p(e_m,i+1)]>E[p(e_n,i+1)]$.

综上所述,对于任意 $1\le i\le K$,总有 $E[p(e_m,i)]>E[p(e_n,i)]$.

由于证得可疑边的权增量期望值大于正常边的权增量期望值,因此可以认为,绝大多数可疑边的权值都会大于正常边的权值.由证明过程可知,$E[p(e_m,i)]-E[p(e_n,i)]$与 $B-A$ 成正比.对于快速传播的蠕虫,存在 $B>>A$,因此有 $E[p(e_m,i)]>>E[p(e_n,i)]$.随着聚积过程的进行,$E[p(e_m,i)]-E[p(e_n,i)]$逐渐增大,可疑边的权值优势逐渐增大.同时,因为聚积过程是沿着感染链的逆向聚积,使得蠕虫传播初期的感染边将会得到更多的权值,从而在所有边中凸显出来.但是可疑边不仅仅包括感染边,还包括一部分正常边和非感染性攻击边.在蠕虫传染的初期阶段,被感染的主机不多,可疑边的绝大多数属于感染边.蠕虫传染后期,网络中的易感主机几乎全部被感染,导致可疑边中的正常边和非感染性攻击边增多.因此,在蠕虫传播初期检测漏报率较低,有利于及早发现蠕虫,为采取措施抑制蠕虫的进一步传播争取了时间.

2.5 性能衡量

聚积算法的目的是找出蠕虫传播初始阶段的攻击序列,除了需要了解聚积算法对于攻击边的检测准确率

和感染边的检测准确率以外,还希望评价算法找出初始感染边的能力.首先给出一些符号的解释:E 为所有边的集合;ANS 为返回的结果边集合;CAU 为所有感染边的集合;INI 为按时间排序前 10%的感染边集合(初始感染边);A_{ans} 为结果集中的攻击边数量;C_{ans} 为结果集中的感染边数量.

本文首先从以下两个方面评价聚积算法的性能:1) 攻击边准确率:考察算法得出的结果集中攻击边的比率;2) 感染边准确率:考察算法得出的结果集中感染边的比率.另外,聚积算法的目的是找出蠕虫传播初始阶段的攻击序列,为了重构感染树,除了需要了解聚积算法对于攻击边的检测准确率和感染边的检测准确率以外,还希望评价算法找出初始感染边的能力.通过对漏报率和误报率的如下定义来进行考察:1) 漏报率:未检测出的初期感染边占所有感染边的比率;2) 误报率:误检测出的初期感染边占所有非感染边的比率.对于初期感染边,实验中我们定义为按时间排序的前 10%感染边.

具体地,在实验中我们从以下 4 个方面衡量算法性能:

1) 攻击边准确率,即结果集中攻击边的比率: $AA=\dfrac{A_{ans}}{|ANS|}$;

2) 感染边准确率,即结果集中感染边的比率: $CA=\dfrac{C_{ans}}{|ANS|}$;

3) 漏报率,即未检测出的初始感染边比率: $FN=\dfrac{|\{e\,|\,e\in INI\text{且}e\notin ANS\}|}{|CAU|}$;

4) 误报率,即误检测出的初始感染边比率: $FP=\dfrac{|\{e\,|\,e\in ANS\text{且}e\notin INI\}|}{|E|-|CAU|}$.

3 在线聚积算法

聚积算法必须在蠕虫爆发之后,采集相当长时间(如 3~5 小时)的网络全部流量才能开始运行.而此时,蠕虫已经感染了网络中的大部分甚至全部的易感主机.如果能够实现算法在线化,则可以在蠕虫爆发初期(如 30 分钟)检测出蠕虫的传播,相关的抑制和防御措施可以尽快地展开,蠕虫造成的损失也能被尽量降低.在相关工作中,提高检测算法的实时性通常采用滑动窗口的方法[9,13].聚积算法能够快速地得出结果,为在线化提供了很好的条件.基于离线的聚积算法,我们提出了使用滑动窗口的在线聚积算法.算法执行过程如下:

1) i=0;
2) 采集最近 S 秒时间内的网络流量,构成当前窗口的主机联系图 G_i;
3) 对于任意的 $e\in E_i$,$p(e,0)$=1.0;
   若 i>0,对于任意的 $e\in E_i$ 且 $e\in ANS_{i-1}$,$p(e,0)$=2.0;
4) 在 G_i 上运行聚积算法,计算出权值最大的 Z 条边构成当前结果集 ANS_i;
5) 合并相邻时间窗口的结果集 ANS_{i-1} 和 ANS_i,从中选出权值最大的 Z 条边重新赋给 ANS_i,根据新的 ANS_i 重构感染树;
6) i=i+1,R 秒之后转到第 2)步.

实际的网络蠕虫攻击事件之间是存在关联性的.蠕虫传播的感染树实际上构成了一棵因果树,感染链上相邻两条边存在着因果关系.上述在线聚积算法将历史上得出的结果采纳进来一起分析:对于 ANS_{i-1} 中的每条边,它们是感染边的可能性非常大,因此,在当前窗口检测时,首先赋予这些边更多的初始权值.

滑动窗口算法具有很多优点:R 秒执行一次聚积算法,能够尽早地发现蠕虫;每次执行只需采集最近 S 秒时间内的流量数据,降低了数据规模,减少了一次算法的运行时间.当然,只采用局部数据会导致算法的准确率较采用全局数据有所降低.

但算法的在线化依然存在一些问题.为了对连续时间阶段产生的结果进行合并,希望当前得到的结果集 ANS_i 可以构成一棵树(至少能构成森林).设 $e_1=\langle u_1,t_1^s,v_1,t_1^e\rangle$,$e_2=\langle u_2,t_2^s,v_2,t_2^e\rangle$是结果集中的两条边.如图 1 所示,当 $v_1=v_2$ 时,结果集不能构成一棵树;当 $u_1=v_2$ 且 $v_1=u_2$ 时,结果集中形成环.在这两种情况下,e_1,e_2 组成一对冲突边.e_1, e_2 都在结果集中,它们都是攻击边的可能性非常大,但此时真正的感染边只有一个.

定义 $p(e_1)$,$p(e_2)$的近似度: $app|_{p(e_1),p(e_2)}=\frac{|p(e_1)-p(e_2)|}{\max\{p(e_1),p(e_2)\}}$.当 $p(e_1)$,$p(e_2)$非常相近,即 $app|_{p(e_1),p(e_2)}\le$阈值λ时,保留时间早的边为感染边,否则,保留权值大的边为感染边.

当合并相邻两个时间窗口内得到的结果集时可能会丢失某些真正的感染边.图 2 描述了这样的情形.S_i,S_{i+1}表示相邻的两个时间窗口,完整的感染树包括图 2 中的所有边,在 S_i 和 S_{i+1} 内各获得了一棵感染树,但丢失了边 e,简单地合并二者会使感染树更加残缺.e 的丢失不能在 S_i 内纠正,因为算法不能预测将来,这样的不足只能在 S_{i+1} 内加以弥补.当 ANS_i 和 ANS_{i+1} 不能合并在一起时,从剩下的边中选取权值最大的连接两棵子树,形成当前的感染树.

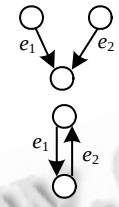

Fig.1 Conflict edge

图 1 冲突边

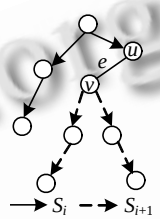

Fig.2 Missing edge

图 2 缺失边

4 实验和分析

在蠕虫检测工作中,通常混合使用真实网络流量和人为生成的蠕虫流量来构造实验数据[5,11−13].为了方便重复实验,使用预先捕获的网络真实流量作为背景流量.背景流量常取自网络主交换或路由处.在真实背景流量的基础上,人为地加入蠕虫攻击流量开展实验.我们使用 WAND 的 NZIX II 数据[14]作为实验背景流量.数据是在新西兰 Internet 交换中心捕获的长 2.5 小时经 GPS 时间同步的 IP 包头部流量,包括 6 个网络的交换流量,共有 0.1 百万个主机,3 百万个流.这些流量通过路由器的 SPAN 端口捕获,只有流的摘要信息,而不包括具体包内容.背景流量由 TCP,UDP 和 ICMP 流量组成.通过匿名化,所有地址分布在 10.0.0.018 范围内.

我们设定蠕虫在 900s 时开始爆发.一台主机被感染后,每隔一定时间,它会在所有其他主机中随机选择一台作为攻击对象,并发出一条攻击流.若目标主机是易感的,则会被感染,否则不会.我们以被感染主机向外发出攻击边的周期(蠕虫扫描周期)作为评价蠕虫侵略性的标准.在下面的实验中,设定网络中的易感主机比率为 0.1.为了研究不同传播速率的蠕虫对算法性能的影响,设定了 3 种不同速率的蠕虫(Worm-x 表示每秒发出 1/x 次扫描).从表 1 和图 3 可以看出 3 种蠕虫的差别:Worm-30 攻击性非常强,在 5 000s 时已经感染了全部易感主机, 41%的边是攻击边;相比而言,Worm-90 的行踪非常隐蔽,攻击流量只占总流量的 3%,最后时刻也只感染了 54%的易感主机;而 Worm-60 介于前两者之间,在 9 000s 时感染了绝大部分易感主机.

Table 1 Three different warm scanning rates

表 1 3 种不同的蠕虫扫描速率

Worm scanning rate	Worm-30	Worm-60	Worm-90
Worm scan duration (s)	30	60	90
Total flows (million)	5.08	3.59	3.09
Fraction of attack edges	0.41	0.16	0.03
Fraction of causal edges	0.002 4	0.003 2	0.002 0
Fraction of infected hosts	0.100	0.099	0.063

在考虑某个参数对算法性能的影响时,只允许相应一个参数的变化.实验中若无特殊说明,各参数初始值见表 2.第 4.1 节讨论各参数对聚积算法性能的影响.第 4.2 节讨论各参数对在线算法性能的影响.第 4.3 节给出算法得到的一棵感染树.第 4.4 节比较聚积算法、随机行走算法[5]、在线聚积算法、在线随机行走算法的性能差别.第 4.5 节实验验证算法在真实网络环境中的检测能力.

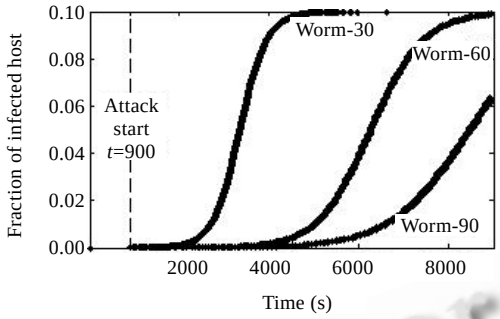

Fig.3 Fraction of infected host along with time

图 3 蠕虫在某时刻已感染的主机

Table 2 Parameters' initial value

表 2 参数的初始值

Number of accumulation process: K	10
Time interval on the definition of precursor: Δt seconds	1 000
Number of edges in the result set: Z	100
Running duration of online algorithm: R seconds	480
Size of sliding window: S seconds	2 400
Threshold when choosing conflict edge: λ	0.1

4.1 聚积算法的性能

4.1.1 K 对性能的影响

图 4 给出了聚积次数 K 对算法准确率的影响.对于各种扫描速率的蠕虫,K=9 时,算法性能最优,攻击边和感染边的准确率均达到了最高.由此可以看出,仅需要较少的聚集次数就能得到很好的结果.从图 4 中还可发现,当 K 继续增大时,准确率有所下降.因为聚积次数过多时,沿着感染链的逆向聚积更有可能将权值聚集到蠕虫攻击开始之前的正常边,让更多的非感染性可疑边加入了结果集.图 4 还显示了 3 种不同速率的蠕虫对检测准确率的影响.可以清晰地看出,快速传播蠕虫更容易被检测,但即使是传播非常隐蔽的 Worm-90,感染边检测的准确率依然能够达到 30%.

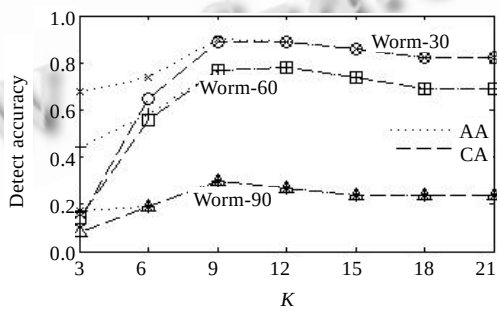

Fig.4 K vs. AA (attack edge accuracy), CA (causal edge accuracy)

图 4 K 对 AA,CA 的影响

4.1.2 Δt 对性能的影响

图 5 中给出了Δt 对 AA,CA 的影响.当Δt 增大时,准确率首先升高然后略有降低.Δt 很小时,准确率很低,因为逆向聚积更有可能到达一个在以前Δt 时间内没有前驱的边,使得权增量丢失,权值聚集到感染树上端的可能性也更低.较大的Δt 使得权值有更大的机会聚积到感染树的高层,于是准确率会随着Δt 的升高而升高.但当Δt 继续增大时,准确率反而降低了.这是因为更大范围的Δt 意味着有更多的前驱,权值更有可能聚积到蠕虫爆发之前的

正常边上.从图 6 和图 7 也可看出这一点,当Δ*t* 继续升高时,FP(false positive),FN(false negative)均有所升高,更多的正常边被选入了结果集.

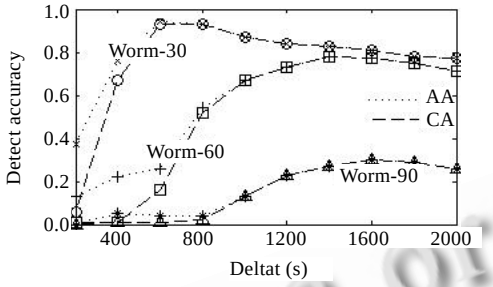

Fig.5 Δ*t* vs. AA, CA

图 5 Δ*t* 对 AA,CA 的影响

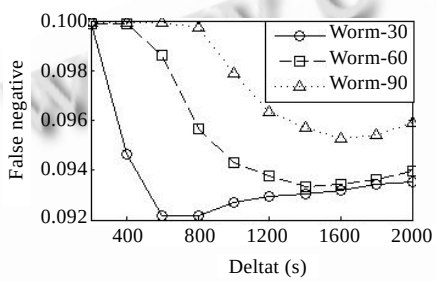

Fig.6 Δ*t* vs. FN

图 6 Δ*t* 对 FN 的影响

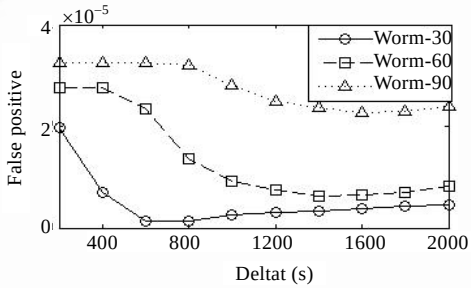

Fig.7 Δ*t* vs. FP

图 7 Δ*t* 对 FP 的影响

结合图 5~图 7 可以看出,存在一个Δ*t*,使得算法准确率达到最高的同时漏报率和误报率都达到最低.表 3 列出了不同蠕虫传播速率下Δ*t* 的最优取值.

Table 3 Optimal Δ*t* of different worm scanning rates

表 3 不同速率蠕虫的最优Δ*t*

Worm scanning rate	Optimal Δ*t* (s)
Worm-30	600
Worm-60	1 400
Worm-90	1 600

4.1.3 *Z* 对准确率的影响

聚积算法使得权值逐渐聚集到了初始感染边之上.图 8 显示,随着结果集的增大,AA,CA 的变化情况.算法在 Worm-30 数据上检测出的 TOP-10 达到了 100%的准确率.随着 *Z* 的增大,越来越多的非感染边被选入结果集.但是这种变化是平缓的,即使是在 Worm-60 数据上检测出的 TOP-600,准确率也达到了 73%.同时也注意到,当 *Z* 很小时,对 Worm-90 的检测准确率很低,传播缓慢的蠕虫能够更好地隐藏它们的行踪,适当增大 *Z* 有助于检测出隐蔽性高的蠕虫.从图中可以看出,*Z*=100 时,对 Worm-90 的检测达到了最高的 30%.从图 4、图 5、图 8 中还可以发现,感染边准确率通常相当接近攻击边准确率.这是因为随着聚积过程的进行,权值逐渐聚积到了蠕虫爆发初期的可疑边之上.而在蠕虫爆发初期,被感染的主机较少,可疑边中的绝大多数属于感染边.

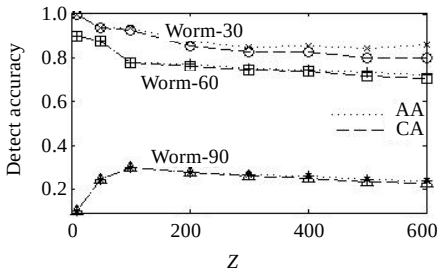

Fig.8 Z vs. AA, CA

图 8 Z 对 AA,CA 的影响

4.2 在线算法的性能

由聚积算法的执行过程可知,算法运行的时间复杂度为 $O(K\times|E|)$.算法运行时间主要受网络中流量规模的影响,但这种影响只是线性的.图 9 直观地显示了算法运行时间和数据规模之间的关系.算法运行在一台奔腾 4 的 PC 机上(2.8GHZ CPU,512MB 内存).从图中可以看出,即使百万级的流量,算法也能在数秒之内完成.这为算法的在线化提供了很好的条件.

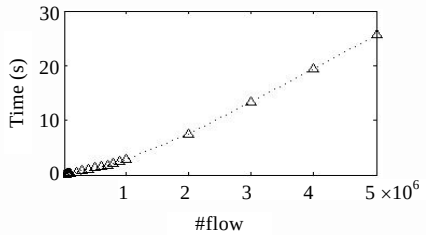

Fig.9 Data magnitude vs. execution time

图 9 数据规模和算法运行时间的关系

4.2.1 性能随时间的变化

图 10 显示了在线算法的性能随时间的变化(图例中 30,60,90 表示蠕虫扫描周期).结合图 3 可以看出,对于快速传播蠕虫 Worm-30,算法在 2 500s 即能检测出部分感染边,而此时只有 0.5%的主机被感染.在蠕虫爆发中期,准确率已能达到 50%.即使是非常隐蔽的 Worm-90,算法也能在 0.3%的主机被感染之前探测到它的行踪.对不同速率的蠕虫,算法均能在爆发初期探测到蠕虫,在爆发中期就能达到较高的检测准确率.从图 11 还可以看出,蠕虫爆发初期算法的漏报率非常低.及早探测出蠕虫的行踪,有助于部署防御,降低损失.

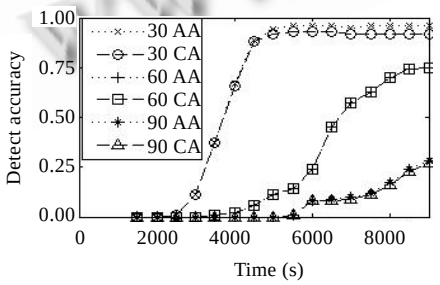

Fig.10 Online detection accuracy along with time

图 10 在线检测准确率随时间的变化

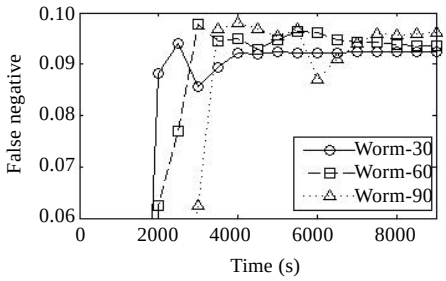

Fig.11 Online detection FN rate along with time

图 11 在线检测漏报率随时间的变化

4.2.2 *R*,*S* 对性能的影响

对于一种在线算法,我们希望算法执行的周期非常地短,这样能够及早地检测出蠕虫.图 12、图 13 显示了窗口大小 *S* 和算法执行周期 *R* 对准确率的影响(Worm-30).只采用局部数据,这既是在线算法的必然要求和优点(降低运行时间和内存开销),也是它的不足(准确率有所降低).从图中可以看出,窗口越大时,算法的准确率越高,但在 *S*=2 400 和 *S*=3 600 时的准确率相差很小.随着执行周期的增大,算法的准确率有所增加.这是因为在相邻窗口的感染树合并时,重叠部分的边越多,造成的冲突边会越多.但准确率随执行周期的变化是平缓的,每 60s 执行一次聚积算法即能达到近 80%的准确率.同时观察到,图 12 和图 13 的曲线非常相近,再一次显示了算法检测出的攻击边绝大多数都是感染边这一结论.

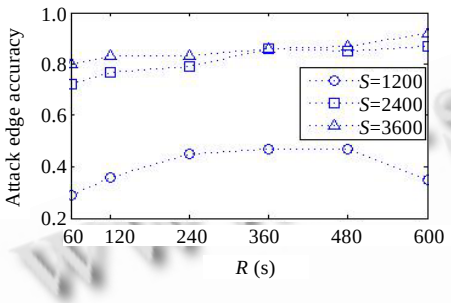

Fig.12 *R*, *S* vs. AA

图 12 *R*,*S* 对 AA 的影响

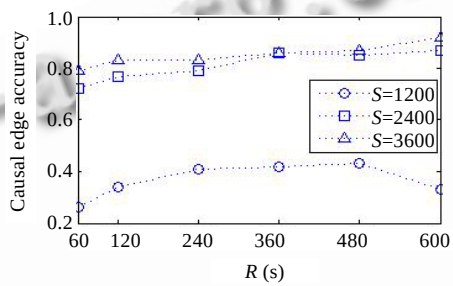

Fig.13 *R*, *S* vs. CA

图 13 *R*,*S* 对 CA 的影响

4.2.3 λ对准确率的影响

在线算法试图构造一棵感染树,推测蠕虫传播的初始阶段.如何解决构造感染树时遇到的冲突问题,我们引入了阈值λ.第 3.3 节已经指出,$E[p(e_m,i)]-E[p(e_n,i)]$与 $B-A$ 成正比.对于快速扫描蠕虫,由于其较高的扫描速率,导致受感染主机发出的流量较大,于是 $E[p(e_m,i)]>>E[p(e_n,i)]$.当结果集中两条边的权值比较接近时,它们更有可能同为攻击边,但是感染树中不可能同时包含产生冲突的两条边.从图 14 中可以看出(Worm-30),λ作为影响最终结果的阈值,太大或太小都会影响算法准确率,一般来说,λ=0.1 是一个较好的取值.

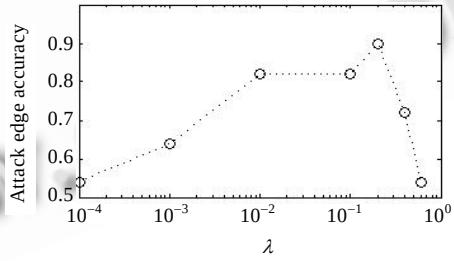

Fig.14 λ vs. AA

图 14 λ对 AA 的影响

4.3 构造感染树

聚积算法试图检测出蠕虫传播初期的感染序列,保证返回的结果可以构成森林,并尽量重构感染树.图 15 是由算法得到的结果集(TOP-40)去掉冲突边、加入缺失边之后绘制的感染树.椭圆代表主机,用 IP 的后 3 节区分.图中所有的边中只有 6 条是非感染边.算法正确地检测到了蠕虫的传染源,并以较高的准确率检测出了蠕虫爆发初期的感染边.

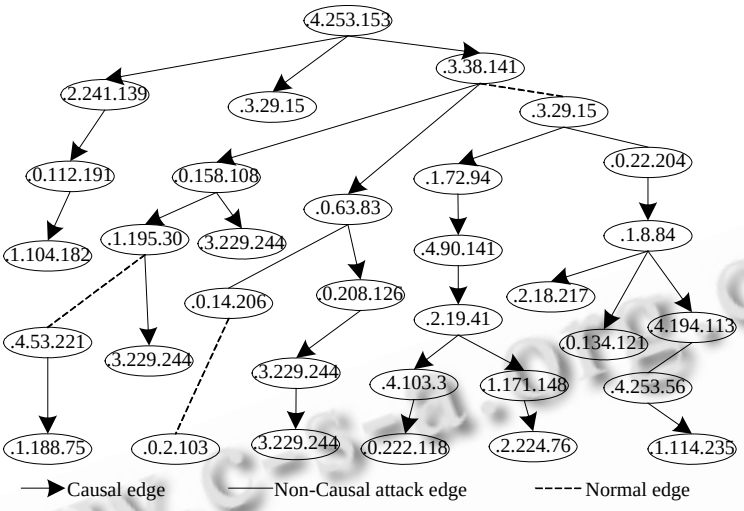

Fig.15 A causal tree reconstructed by accumulation algorithm

图 15 聚积算法重构的感染树

4.4 与随机行走算法的比较

4.4.1 离线算法性能比较

Xie 等人提出了随机行走[5,10]的方法来检测蠕虫的攻击源和初始感染序列.该算法重复 W 次从主机联系图中进行路径采样.采样中出现最频繁的 Z 条边被选为蠕虫爆发初期的感染边.算法产生一条路径时从随机选取的一条边开始,每一次从可能的前驱边中随机选取一条进行逆向行走.

图 16 比较了各种算法检测出感染边的准确率(Worm-30):聚积算法(accumulation,简称 ACC)、随机行走算法(W-x,其中,x 表示取样次数,取样次数越多,准确率越高).图 16 显示,聚积算法获得了最高的 94%的攻击边准确率(AA)以及 91%的感染边准确率(CA).从图中还可以发现,随机行走算法虽然能够检测出较多的攻击边,但其对感染边的检测准确率并不高,这方面聚积算法表现良好.CA 率越高,构造的感染树越精确,更有利于防御的部署.图 17 是对各种算法执行时间的比较($|E|=5\times10^6$).当数据规模达到百万级流量时,聚积算法的执行时间只有 500 次随机行走算法的 10%、4 000 次取样随机行走算法的 1%.离线聚积算法的高效执行,为其在大规模网络中的在线化打下了良好的基础.

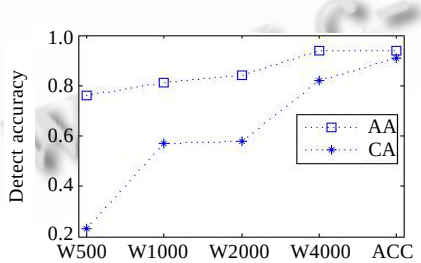

Fig.16 Accuracy of offline algorithms

图 16 离线算法检测准确率的比较

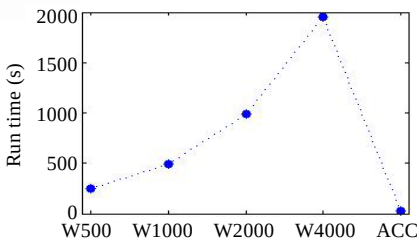

Fig.17 Execution time of offline algorithm

图 17 离线算法执行时间的比较

4.4.2 在线算法性能比较

为了比较在线算法的性能,采用类似的滑动窗口方法将原始的随机行走算法在线化.在窗口大小和执行周期相同的情况下,图 18(Worm-30)和图 19($|E|=5\times10^6$)比较了各种在线算法的性能:在线聚积算法(online accumulation,简称 OACC)、在线随机行走算法(OW-x,x 表示取样次数).同样可以看出,在线聚积算法达到了最高的 80%的攻击边准确率和 76%的感染边准确率,执行时间也远小于 4 000 次取样的在线随机行走算法.

虽然在线算法的准确率相对离线算法有一定损失,但在线算法拥有更高的实时性,使得蠕虫能够尽早地检测出来.而且,在线聚积算法已经达到了将近 80%的准确率,这样的结果比较理想.

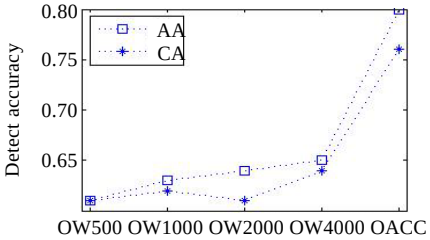

Fig.18 Accuracy of online algorithms

图 18 在线算法检测准确率的比较

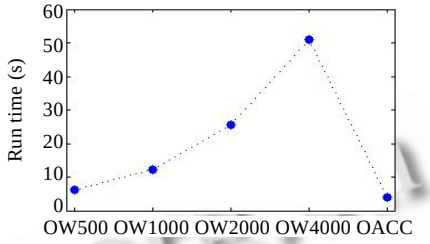

Fig.19 Execution time of online algorithm

图 19 在线算法执行时间的比较

4.5 真实网络环境中的实验

为了检验算法在真实网络环境中的性能,参考同类工作[15]的实验过程,组建了实际网络环境和虚拟机环境相结合的实验环境.实际网络环境包括教育网中的 5 个 C 类网段,不同网段采用网关和路由器连接,含有 Windows 主机 656 台,Linux 主机 251 台.其中包括 4 台邮件服务器和 6 台 Web 服务器.首先从教育网中捕获 1 小时内 5 个 C 类网段内部的所有网络流量,包括 TCP,UDP,ICMP 等流量,作为实验的背景流量.然后采用 UML 虚拟机技术[15],虚拟易感节点,添加蠕虫流量.

我们利用分布于不同网段的 5 台 PC 机,使用 UML 虚拟机建立了一个包含 136 个虚拟易感节点的实验环境.每个虚拟机运行 Redhat Linux 6.1 操作系统,宿主机运行 Redhat Linux 9.0 操作系统.每台宿主机上运行的若干虚拟机组成一个局域网络,不同局域网内的虚拟机通过宿主机网关和实际网络网关互相通信.该环境可以独立地反复使用,能够灵活地配置每个节点和网络拓扑.在蠕虫爆发后可获取用于分析的网络数据和感染情况.虚拟实验环境搭建完成之后,人为地初始启动一个蠕虫传染源,爆发 Lion 蠕虫攻击[16],并记录真实的感染情况和攻击流量.表 4 和表 5 分别给出了实验数据信息和实验结果.由表 5 可以看出,算法在实际网络环境中仍然能够得出令人满意的结果.

Table 4 Information of experiment data

表 4 实验数据信息

Total flows	14 356
Total hosts	907
Fraction of attack edges	7 966
Fraction of causal edges	135
Fraction of vulnerable hosts	0.15
Fraction of infected hosts	0.15

Table 5 Experiment result (%)

表 5 实验结果(%)

	AA	CA
ACC	92	90
OACC	83	80

5 结 论

在线获取网络蠕虫的传播路径不仅可以推测最早的被感染的节点,而且还可以推测在传播过程中造成其他节点被感染的传播路径.本文提出的基于滑动窗口的在线聚积算法,可以推测随时间变化的蠕虫传播路径.通过实验验证和分析各项参数对算法结果的影响,可以选择恰当的参数以提高算法的准确率,并降低误报率和漏报率.

实验结果表明,对于快速传播蠕虫,在线聚积算法可在蠕虫爆发 1 600s 后开始得到感染边,进而获得了近 80%的准确率.即使对于非常隐蔽的蠕虫,在线算法也能在 0.3%的主机被感染之前检测出蠕虫的一些踪迹.在得到相同准确率结果的情况下,聚积算法所需要的路径推测时间只是典型的同类工作随机行走方法的 1%.本文工作可作为其他恶意代码,如木马、网络僵尸等追踪研究工作的基础,并对它们产生启发作用,也是网络蠕虫检测

和防治的一种重要和必备的补充手段.本文提出的在线聚积算法侧重于最基本的扫描式蠕虫,进一步的工作是建立更加复杂和真实的恶意代码实验环境,研究复杂网络环境下的恶意代码在线追踪.

References:

[1] Wen WP, Qing SH, Jiang JC, Wang YJ. Research and development of Internet worms. Journal of Software, 2004,15(8):1208−1219 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1208.htm

[2] Chen ZS, Ji CY. An information-theoretical view of network-aware malware attacks. IEEE Trans. on Information Forensics and Security, 2009,4(3):530−541.

[3] Kienzle DM, Elder MC. Recent worms: A survey and trends. In: Staniford S, Savage S, eds. Proc. of the 2003 ACM Workshop on Rapid Malcode. New York: ACM Press, 2003. 1−10.

[4] Rajab MA, Monrose F, Terzis A. Worm evolution tracking via timing analysis. In: Atluri V, Keromytis AD, eds. Proc. of the 2005 ACM Workshop on Rapid Malcode. New York: ACM Press, 2005. 52−59.

[5] Xie YL, Sckar V, Maltz DA, Reiter MK, Zhang H. Worm origin identification using random moonwalks. In: Martin DC, ed. Proc. of the IEEE Symp. on Security and Privacy. Los Alamitos: IEEE Computer Society, 2005. 242−256.

[6] Kumar A, Paxson V, Weaver N. Exploiting underlying structure for detailed reconstruction of an Internet scale event. In: Padmanabhan VN, Veitch D, Greenberg A, eds. Proc. of the 5th Conf. on Internet Measurement. Berkeley: USENIX Association, 2005. 351−364.

[7] Savage S, Wetherall D, Karlin A, Anderson T. Practical network support for IP traceback. ACM/IEEE Trans. on Networking, 2001, 9(3):226−237. [doi: 10.1109/90.929847]

[8] Zhang Y, Paxson V. Detecting stepping stones. In: Bellovin S, Rose G, eds. Proc. of the 9th USENIX Security Symp. Denver: USENIX Association, 2000. 171−184.

[9] Peng P, Ning P, Reeves DS, Wang XY. Active timing-based correlation of perturbed traffic flows with chaff packets. In: Martin DC, ed. Proc. of the 25th Int'l Conf. on Distributed Computing Systems Workshops. Los Alamitos: IEEE Computer Society, 2005. 107−113.

[10] Xie YL, Sckar V, Reiter MK, Zhang H. Forensic analysis for epidemic attacks in federated networks. In: Martin DC, ed. Proc. of the 14th IEEE Int'l Conf. on Network Protocols. Los Alamitos: IEEE Computer Society, 2006. 43−53.

[11] Sarat S, Terzis A. On the detection and origin identification of mobile worms. In: Kruegel C, ed. Proc. of the 2007 ACM Workshop on Recurring Malcode. New York: ACM Press, 2007. 54−60.

[12] Collins MP, Reiter MK. Hit-List worm detection and bot identification in large networks using protocol graphs. In: Krugel C, Lippmann R, Clark A, eds. Recent Advances in Intrusion Detection, 10th Int'l Symp., RAID 2007. LNCS 4637, Berlin, Heidelberg: Springer-Verlag, 2007. 276−295.

[13] Stafford S, Li J, Ehrenkranz T. Enhancing SWORD to detect 0-day-worm-infected hosts. Simulation: Trans. of the Society for Modeling and Simulation Int'l, 2007,83(2):199−212. [doi: 10.1177/0037549707080753]

[14] WAND Network Research Group. WAND WITS: NZIX-II trace data. 2000. http://wand.cs.waikato.ac.nz/wits/nzix/2/nzix-ii.php

[15] Jiang X, Xu D, Wang HJ, Spafford EH. Virtual playgrounds for worm behavior investigation. In: Valdes A, Zamboni D, eds. Recent Advances in Intrusion Detection, the 8th Int'l Symp., RAID 2005. LNCS 3858, Berlin, Heidelberg: Springer-Verlag, 2005. 1−21.

[16] Linux Lion Worms. 2001. http://www.symantec.com/security_response/writeup.jsp?docid=2001-032311-2042-99&tabid=1

附中文参考文献:

[1] 文伟平,卿斯汉,蒋建春,王业君.网络蠕虫研究与进展.软件学报,2004,15(8):1208−1219. http://www.jos.org.cn/1000-9825/15/1208.htm

李强(1975－),男,博士,副教授,主要研究领域为网络攻击源追踪.

向阳(1986－),男,博士生,主要研究领域为网络路由,网络安全.