基于会话异常度模型的应用层分布式拒绝服务攻击过滤

肖 军[1),2)] 云晓春[1)] 张永铮[1)]

[1)](中国科学院计算技术研究所 北京 100190)

[2)](中国科学院研究生院 北京 100049)

摘 要 大量的网络攻击手段和可利用的网络资源大大增加了抵御分布式拒绝服务(Distributed Denial-of-Service, DDoS)攻击的难度.应用层 DDoS 建立在正常的网络层行为之上,当前网络层安全设备无法有效抵御攻击.文章提出了一种应用层 DDoS 攻击过滤模型.基于攻击请求的生成方式,文中将应用层 DDoS 攻击分为 5 类,分析了应用层 DDoS 攻击与正常访问行为的不同,提出了访问行为异常属性和 session 异常度模型.利用此模型,可以有效区分正常访问 session 和应用层 DDoS 攻击 session.将 First-Come First-Serve(FCFS)、Low Suspicion First(LSF)和 Round Robin 3 种转发策略与 session 异常度模型结合,采用真实网络日志,模拟分析合法请求返回时延随时间的变化关系.结果表明,转发速率为合法请求最大速率就可获得较好的转发性能,此外,FCFS 和 Round Robin 比 LSF 具有更低的合法请求返回时延.

关键词 DDoS;过滤;异常度;应用层;转发策略

中图法分类号 TP393 **DOI 号**: 10.3724/SP.J.1016.2010.01713

Defend Against Application-Layer Distributed Denial-of-Service Attacks Based on Session Suspicion Probability Model

XIAO Jun[1),2)] YUN Xiao-Chun[1)] ZHANG Yong-Zheng[1)]

[1)](*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

[2)](*Graduate University of Chinese Academy of Sciences, Beijing* 100049)

Abstract Mitigating Distributed Denial-of-Service (DDoS) attacks becomes more challenging with increasing available resources and techniques for attackers. Current network-layer security devices fail to counter application-layer DDoS (App-DDoS) attacks for the normal traffic feature on the network layer. In this paper, to handle App-DDoS attacks, a novel defense model is proposed. App-DDoS attack is divided into 5 types based on the attack URL generating way. Based on the differences between normal sessions and attack sessions, the paper proposes the session behavior suspicion parameters and the session suspicion model, which can be used to differentiate normal sessions from App-DDoS sessions accurately. The model is combined with 3 forwarding policies, including First-Come First-Serve (FCFS), Low Suspicion First (LSF) and Round Robin respectively to defend against 5 types of App-DDoS attacks. Simulation result with real Web trace shows that these forwarding policies perform well when the forwarding rate equals to the maximum normal request arrival rate, and FCFS and Round Robin perform better than LSF on the normal request response delay.

Keywords DDoS; filter; suspicion; application-layer; forwarding policy

---

收稿日期:2010-04-19;最终修改稿收到日期:2010-08-09.本课题得到国家自然科学基金(60703021)、国家"八六三"高技术研究发展计划项目基金(2007AA010501,2007AA01Z474,2007AA01Z467)资助.**肖 军**,男,1979 年生,博士研究生,主要研究方向为 DDoS 攻击检测、DDoS 攻击过滤.E-mail: jjjunxiao@hotmail.com. **云晓春**,男,1971 年生,博士,教授,博士生导师,主要研究领域为网络安全、互联网建模和网络测量. **张永铮**,男,1978 年生,博士,副教授,主要研究领域为网络安全和网络安全评估.

1 引 言

分布式拒绝服务(Distributed Denial-of-Service,DDoS)攻击是互联网安全的主要威胁之一,可在网络层或应用层实现.当前,对网络层 DDoS 攻击的检测和过滤研究已取得了相当的研究成果,基于网络层流量异常,防火墙等网络安全设备能够有效检测和抵御网络层 DDoS 攻击,如 SYN Flood、ICMP Flood 攻击等.应用层 DDoS 攻击首先提交正常的连接建立请求,在连接建立后,向目标服务器提交服务请求,消耗服务器计算资源.由于在网络层行为表现正常,应用层 DDoS 攻击能够有效逃避应用层级的检测和过滤.在 SYN Flood 攻击无法取得好的攻击效果时,攻击者可采用应用层 DDoS 攻击达到攻击意图[1].

应用层 DDoS 攻击可采用真实 URL 请求或伪造 URL 请求.除简单发送大量请求的 HTTP flood 攻击方式外,攻击者可提交如下请求,实现对服务器资源的有效消耗:(1)提交较长的请求消耗服务器缓冲区,使得大量合法请求无法进入缓冲区而被丢弃.为了达到较好的攻击效果,攻击者往往伪造一个较长的 URL 请求,提交给服务器.(2)下载大文件,这种攻击方式可以导致对磁盘的频繁访问.(3)请求较大的文件,如较大的图片和页面,此类攻击能够有效消耗网卡的计算能力和下行链路带宽.(4)提交大计算开销请求,包括计算复杂性高的操作,如加解密计算;或者复杂的数据库操作,如某个属性的最大值查询,此类攻击能有效消耗 CPU 或数据库服务器的计算能力.采用上述几种类型的请求,无需大量的攻击请求,攻击者可实现对服务器的有效攻击.同时,为了达到较好的攻击效果,攻击者可提高攻击请求的发送频率.

本文首先对应用层 DDoS 进行了分类,然后分析 DDoS 攻击与正常访问行为的不同点,提出访问异常属性,然后本文提出一种对服务和用户透明的应用层 DDoS 攻击 session 识别方法,无需了解请求具体内容、执行情况或资源消耗,只需获取请求 URL 和到达时间等信息,即可建立相应的模型,计算出的 session 的行为可疑度能有效区分攻击 session 和合法访问 session,为攻击过滤提供决策依据,由于 session 行为可疑度计算时间复杂性为 $O(1)$,可以实现对应用层 DDoS 攻击的实时过滤.本文的工作具体如下:

首先,基于攻击 URL 请求生成方式,本文将应用层 DDoS 攻击分为 5 类,包括 Single-URL Flood、Multi-URL Flood、Random-URL Flood、Session Flood 和 Forged-URL Flood 攻击.应用层 DDoS 攻击与正常访问行为相比,在 thinking time、main page 请求顺序、内嵌对象数目、URL 长度、main page 请求数目和请求序列循环数等方面存在多个明显区别.基于上述几个区别,本文提出访问行为异常属性,并基于异常属性,提出 session 异常度计算模型.对真实网站日志的分析表明,session 异常度模型具有较高的识别精度,由于应用层 DDoS 攻击 session 具有较高的异常度,因而能够与合法 session 有效区分.基于 session 异常度进行请求转发,分析 First-Come First-serve(FCFS)、Low Suspicion First(LSF)和 Round Robin 3 种转发策略下合法请求返回时延随时间的变化关系.结果表明,转发速率为合法请求最大速率,就可获得较好的转发性能,并且 Round Robin 和 FCFS 比 LSF 相比,具有较低的合法请求返回时延.

本文第 2 节介绍相关工作;第 3 节对应用层 DDoS 进行分类;第 4 节比较攻击 session 和正常访问 session 的不同,提出访问行为异常属性和 session 异常度模型,并对模型的有效性进行验证;第 5 节分析转发速率和转发调度策略;第 6 节对一些问题进行讨论;第 7 节总结全文.

2 相关工作

应用层 DDoS 攻击导致访问流量大幅度增加,与 Flash Crowd[2] 极为相似,因此应用层 DDoS 攻击的检测研究主要是如何区分这两者.Jung[3] 等人认为 Flash Crowd 发生时,大量的地址 cluster 重复出现,而 DDoS 攻击时,会出现大量新的地址 cluster;在 Flash Crowd 时,与正常访问相比,每个用户对应的请求数变小,而 DDoS 时则变大;Flash Crowd 的访问地址分布不均匀,而 DDoS 攻击时,访问地址分布比较均匀;在文件请求方面,Flash Crowd 发生时被请求文件呈 Zipf-like 分布,而 DDoS 发生时则集中在少数文件.Xie[4] 等人基于 document populairty 来区分应用层 DDoS 攻击和 Flash Crowd,Flash Crowd 对应的 aggregate access behavior 熵无明显变化,而 DDoS 对应的 aggregate access behavior 的熵值有较大下降.Li[5] 等人利用 variation metric 和 Bheattacharyya metric,采用概率测量的方法来区

分 DDoS 攻击和 Flash Crowd.

攻击流由固定的程序生成,与正常访问流相比,攻击流之间呈现明显的相似性,Yu[6]等人利用这一特点,实现对 DDoS 攻击流和 Flash Crowd 流的区分.基于请求的动态变化、请求的语义和具备对可视对象处理能力等 3 个正常访问的特征,Oikonomou[7]等人构建了正常行为模型,用来区分攻击 bot 和正常访问者.

为了区分正常访问者和攻击 robot,Park[8]等人采用图灵测试,将行为探测程序传到客户端,分析是否有鼠标移动等正常用户行为,同时分析用户的访问请求是否符合正常浏览的行为模式,判断是正常用户还是 robot. Ranjan[9]等人根据 session 的参数,包括 session 建立速率、请求速率和请求消耗,把应用层 DDoS 攻击分为 Request Flooding 攻击、Asymmetric Workload 攻击和 Repeated one-shot 攻击,基于这 3 种攻击,提出了一种 session 可疑度计算模型.依据 session 可疑度,进行请求转发.构建合法用户行为模型需用 system log 来获取请求对 CPU、带宽以及磁盘的消耗,但 system log 并不易获得,并且在服务器进行更新后,无法及时获得新请求的资源消耗.图灵测试[10]能够有效区分攻击者和正常访问者.由于合法用户能够正确地完成测试,而攻击主机不具备完成测试的能力,可以准确地区分两者,但是图灵测试往往会干扰访问者对服务器的正常访问. Walfish[11]等人提出了一种 speak-up 方法来抵御应用层 DDoS,与以往减慢或削弱攻击者的过滤方法相反,speak-up 方法让所有客户端提高发送速率,但攻击者为了达到较好的攻击效果,通常采取尽最大能力攻击原则,在攻击开始时就会采用最大发送速率,所以增加发送速率的均为合法用户,因此能够识别合法流量. Kruegel[12]等人结合多种异常检测技术,实现对 Web 服务器攻击的检测.以服务器日志为输入,计算 request inquiry 的异常指数.此方法针对 query 进行研究,对 CC 攻击具有一定的检测效果,没有针对应用层 DDoS 攻击进行检测,并且以日志为输入无法实现对攻击的实时检测. Yu[13]等人结合 K-means Clutering 异常检测方法和 Offense 方法,识别攻击端,过滤应用层 DDoS 攻击,但无法有效抵御慢速的应用层 DDoS 攻击. Xie[14]等人提出了一种基于用户浏览行为的统计异常检测,根据 Web 页面的链接特性和各级 cache 对用户请求的响应,采用了隐马尔可夫模型描述服务器端观察的用户访问行为,如果一个用户的访问行为偏离了正常用户的行为特征,则认为此用户为攻击端.此方法不足在于训练和计算过程比较繁琐.

3　应用层 DDoS 攻击分类

Gavrilis[15]等人将 Web DDoS 攻击分为 3 类,在此基础上,本文对应用层 DDoS 攻击的分类进行了扩充.依据攻击主机提交请求的真实性,应用层 DDoS 攻击可分为真实 URL 请求攻击和伪造 URL 请求攻击(Forged-URL Flood).真实 URL 攻击请求可进一步分为重复使用单一 URL 请求(Single-URL Flood)攻击、重复使用多个 URL 请求(Multi-URL Flood)攻击、基于页面链接随机选择 URL 请求(Random-URL Flood)攻击以及为了更好地隐藏攻击者,利用已有合法 session 的请求,按照 session 中各个请求的间隔时间和请求顺序反复提交请求(Session Flood)进行攻击.同时,为了达到更好的攻击效果,在 Forged-URL Flood、Single-URL Flood、Multi-URL Flood、Random-URL Flood 攻击时,攻击者往往会增加请求发送速率.

伪造 URL 攻击方式是一类常见的应用层 DDoS 攻击方式,虽然目前尚未有此类攻击的报道,但在实际中有不少攻击者采用了这种攻击方式.此类攻击具有两个特点:(1)适用性强且易于实现,无需修改 URL 生成方式,可对任何 Web 网站发起攻击;(2)伪造的 URL 往往较长,可以有效消耗服务器的缓冲区,导致合法请求被丢弃.

Single-URL Flood 和 Multi-URL Flood 攻击是最常见且易于实施的攻击方式.攻击者可反复提交一个或数个页面请求,或者反复请求一个或几个大字节对象,如图片或音乐等,达到消耗服务器资源的目的.由于请求的单一性,此类攻击也最易于检查和发现,开源工具 Mod_evasive① 就具备检测此两类攻击的能力.

为了有效地隐藏自己,攻击者可采用 Random-URL Flood 攻击方式,从上一个访问的页面中随机选择一个链接作为下一个请求,由于每次请求均随机选取,增加了检查的难度,无法从请求重复出现的频率和次数角度检测攻击.

为了更好地隐藏自己,攻击者可采用 Session

① http://www.zdziarski.com/blog/wp-content/uploads/2010/02/mod_evasive_1.10.1.tar.gz

Flood 攻击方式，利用一个真实 session(seed session)，依据此 session 的请求 URL、请求顺序和每个请求的间隔时间，循环提交请求进行攻击. 由于此攻击采用了真实的 session 请求，与真实访问行为较为接近，请求间隔时间也符合真实情况，因此这类攻击不易检测. Session Flood 攻击至今未见报道，但实现此类攻击难度不大，攻击者只需捕获一个正常访问 session 即可，攻击者甚至可以利用自己对目标服务器的正常访问 session 来实现攻击.

4 Session 异常度模型

4.1 正常访问行为 vs. 攻击行为

与文献[14]类似，本文通过一个状态转换图描述正常访问行为，包含 REQUEST、VIEW 和 SESSION OVER 3 个状态，如图 1 所示. 在客户端开始向服务器发送请求时，进入 REQUEST 状态，直到所有的返回请求均被客户群接受. 在用户开始浏览请求页面时，进入 VIEW 状态，VIEW 状态的持续时间即为用户阅读页面的时间，称为“thinking time”. 用户结束会话，进入 SESSION OVER 状态.

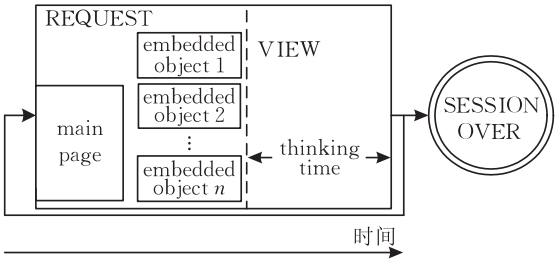


图 1 访问过程状态图

正常访问行为与攻击行为有如下几点不同：

(1) REQUEST 过程中，用户请求的主页面返回后，浏览器根据主页面的内嵌对象(embedded object)链接，如广告条、图片或音乐等，向服务器提交这些内嵌对象请求. 在服务器端观察，从内嵌对象数量上看，一个 main page 通常对应多个内嵌对象，由于服务器不会频繁更新，可以认为在一定时间内每个 main page 对应的内嵌对象数是一定的；从请求提交顺序看，内嵌对象请求会在 main page 请求之后被提交到服务器端；从间隔时间看，内嵌对象请求会在主页面请求后较短时间内被提交到服务器端，通常都是 1s、2s 内. Single-URL Flood 攻击往往只提交 main page，因而内嵌对象数目为 0，或者无 main page，只有内嵌对象，且内嵌对象数目超过了正常内嵌对象数；Multi-URL Flood 攻击往往只提交 main page 或只有内嵌对象，或 main page 与内嵌对象数不符.

(2) 一个页面的 thinking time 长度与页面的内容有关，不同页面对应着不同的 thinking time 分布. Thinking time 通常分布在数秒至数十秒之间. 攻击者为了达到较好的攻击效果，往往增加请求发送频率，如在 1 秒发送多个数据包，相应地 thinking time 值较小.

(3) 用户通常会按照兴趣选择 main page，选取的 main page 往往属于同一话题，而不会随机选取一个页面进行访问. 一个属于兴趣 A 的页面，后面跟随一个同属于兴趣 A 的页面概率较高，而后接一个话题 B 页面的概率较小，并且用户不会反复多次阅读一个或几个页面. Single-URL Flood、Multi-URL Flood 和 Random-URL Flood 攻击提交的 main page 与真实阅读习惯不符.

(4) 对一个网站而言，短时间内不会出现大量 URL 的增加或消失，其 URL 长度分布也是稳定的. 为了有效消耗服务器缓冲区，攻击者采用长度较大的伪造 URL，相应地，攻击者提交的 URL 长度分布异于正常分布.

(5) 正常用户请求数通常在一定的范围之内，在获得了感兴趣的信息后，用户通常会结束访问. 而攻击者为了达到较好的攻击效果，往往发送大量请求到服务器，因而在请求数量上超过正常 session 的请求数.

(6) 正常用户不会反复访问一个或数个页面，而 Session Flood 攻击、Single-URL Flood 和 Multi-URL Flood 攻击反复循环提交一系列的请求.

4.2 Session 异常度模型

基于上述正常访问行为与攻击行为的差异，本节首先提出 session 异常属性和属性训练方法，然后提出 session 异常度模型(session suspicion model).

4.2.1 Session 异常属性

(1) 页面转移异常度(main page transition suspicion). 正常用户请求的相邻两个页面通常属于同一话题. 而 Single-URL Flood、Multi-URL Flood 和 Random-URL Flood 攻击不符合这一规律. 对一个 main page URL_i，下一个 main page 为 URL_j 的次数为 X_j，所有次数的最大值为 M，将页面转移过程视为马尔可夫过程，定义从 URL_i 转移到 URL_j 的异常度为

$$f_{\text{transit}}(i,j)=1-\frac{X_j}{M}.$$

(2) 内嵌对象请求数异常度(embedded objects number suspicion). 由于服务器不会频繁更新,可以认为一个 main page 包含的内嵌对象数在一定时间内(如 6h)是稳定的. 网络 cache 通常会对静态请求进行缓存,如果内嵌目标在 cache 有缓存,则请求不会到达服务器端,因而一个 main page 对应内嵌对象数在一定范围内分布. 由于 Single-URL Flood 攻击、Multi-URL Flood 攻击或 Random-URL Flood 攻击对应的内嵌请求为 0,或者超过正常范围,因此通过内嵌请求数,可以区分攻击和正常访问. 对一个 main page URL_i,对应 n 个内嵌对象的次数为 X_n,不同内嵌对象次数的最大值为 M,定义 URL_i 对应 n 个内嵌对象的异常度为

$$f^i_{\text{eobj}}(n)=1-\frac{X_n}{M}.$$

(3) Thinking time 异常度(Thinking time suspicion). 为了获得较好的攻击效果,攻击者往往增加请求发送频率,相应地,攻击请求的 thinking time 不符合正常请求的 thinking time 分布. 一个 main page URL_i,thinking time 等于 n 秒的次数为 X_n,不同长度的 thinking time 次数最大值为 M,定义 URL_i 对应 thinking time 为 n 秒的异常度为

$$f^i_{\text{tt}}(n)=1-\frac{X_n}{M}.$$

(4) URL 长度异常度(URL length suspicion). 对一个网站而言,短时间内不会出现大量 URL 的增加或消失,其 URL 长度分别也是稳定的. Forged-URL Flood 攻击采用较长 URL 消耗缓冲区,基于 URL 长度可以检测此类攻击. 训练中,长度为 i 的 URL 次数为 X_i,不同长度 URL 次数的最大值为 M,定义长度为 i 的 URL 长度异常度为

$$f_{\text{urllen}}(i)=1-\frac{X_i}{M}.$$

(5) Main page 请求数异常度(main page number suspicion). 攻击者通常会发送尽可能多的请求来实现攻击目的,且对服务器造成的危害与攻击请求数目成正比. Main page 数可用于区分攻击 session 和合法访问 session. 采用累加分布函数定义 main page 请求数异常度,定义 i 个请求的异常度为

$$f_{\text{mreq}}(i)=\sum_{t=1}^{i}X_t\Big/\sum_{t=1}^{N}X_t,\ \ i\leq N \tag{1}$$

其中,N 为训练中合法 session 的请求最大值,X_t 表示为训练中请求数为 t 的 session 数.

(6) 请求循环次数异常度(request repeat times suspicion). 正常用户不会反复请求一个或数个页面,相比之下,Single-URL Flood 攻击、Multi-URL Flood 攻击和 Session-URL Flood 攻击反复循环提交请求. 因此,请求循环次数可作为一个指标识别攻击 session.

首先定义请求循环次数. 如果一个 session 的请求序列 s 由一个请求子序列循环 s' 组成,其各自包含的请求数为 $|s|$ 和 $|s'|$,则请求循环次数为 $|s|/|s'|$. 例如,一个攻击 session 的请求序列为 a,b,c,d,a,b,c,d,a,b,c,d,相应地子序列为 a,b,c,d,$|s|=12$,$|s'|=3$,请求循环次数 $c=3$.

采用累加分布函数计算请求循环次数异常度,定义 c 个请求循环次数异常度为

$$f_{\text{repeat}}(c)=\sum_{t=0}^{c}X_t\Big/\sum_{t=0}^{T}X_t,\ \ c\leq T \tag{2}$$

其中,T 为训练中合法 session 的请求最大循环次数,X_t 为循环次数为 t 的 session 数.

4.2.2　Session 异常度属性

本节首先给出 main page 异常度计算方法,然后在其基础上,给出 session 的异常度计算方法.

(1) Main page 异常度

在收到新的 main page URL_j 时,计算其前一个 main page URL_i 的异常度.

从 URL_i 到 URL_j 的转移异常度为 $f_{\text{transit}}(i,j)$;thinking time $intv$ 为 URL_j 和 URL_i 的最后一个内嵌对象请求的到达时间间隔;URL_i 的长度为 l,相应地,URL 长度异常度为 $f_{\text{urllen}}(l)$;URL_i 的内嵌对象请求数为 n,相应的内嵌对象请求数异常度为 $f_{\text{eobj}}(n)$. 基于上述 4 个异常度,计算 URL_i 的异常度

$$\begin{aligned}f_{\text{page}}(i)=&\alpha\times f_{\text{urllen}}(l)+\beta\times f_{\text{tt}}(intv)+\\&\gamma\times f_{\text{eobj}}(n)+\delta\times f_{\text{transit}}(i,j),\\&\alpha+\beta+\gamma+\delta=1.\end{aligned}$$

每个参数分配一个权重,权重参数的选取依赖于 4 个异常度的大小分布,哪个异常度较大,就赋予其较大的权重. 例如,如果 URL 长度异常度较大($f_{\text{urllen}}(l)>0.95$),则令 $\beta=\gamma=\delta=0.02$,$\alpha=0.94$. 如果 4 个异常度均较小(均≤ 0.95),则令 $\alpha=\beta=\gamma=\delta=0.25$.

(2) Session 异常度计算

在收到一个 session s 的第 $r+1$ 个 main page 请求时,利用其前 r 个 main page 的异常度,计算 s 的异常度. 如果 s 的 main page 请求数 r 小于或等于训练中合法 session 的请求最大值 N,则按照式(1)计算 $f_{\text{mreq}}(r)$,否则令 $f_{\text{mreq}}(r)=1.0$. 请求循环次数 c 小于等于训练中合法 session 的请求最大循环次

数 T,则按照式(2)计算 $f_{\text{repeat}}(c)$,否则令 $f_{\text{repeat}}(c)=c-T+1$. s 的第 i 个($0<i\leq N$) main page 的异常度为 p^i_{page}. 定义 s 异常度为

$$f^r_{\text{session}}(s)=f_{\text{repeat}}(c)\times f_{\text{mreq}}(r)\times\frac{\sum_{i=1}^{r}f_{\text{page}}(i)}{r}.$$

如果 $f^r_{\text{session}}(s)>1$,令 $f^r_{\text{session}}(s)=1$.

4.3 Session 异常度计算方法

Session 的异常度计算过程如图 2 表示,包含两个状态. S_1 对应 main page 处理过程,接收到第 i 个($i>1$) main page 请求后进入 S_1,计算其 $i-1$ 个 main page 异常度,并计算 session 异常度. S_2 对应内嵌对象处理过程. 具体计算过程如下:

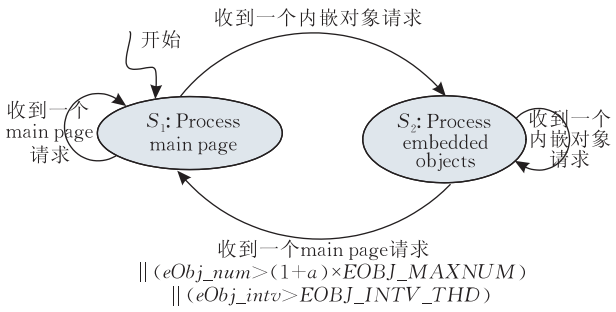


图 2 Session 异常度计算过程

1. 收到一个来自于新 session 的请求,进入 S_1 状态. 如果收到一个 main page URL_1,则转换异常度概率为 $f_{\text{tt}}(\text{NULL},URL_1)$,即此时转移异常度为 URL_1 作为第一个 main page 的异常度. 如果没有收到 main page 而直接收到了内嵌对象,则令此 main page 为 NULL,转换异常度概率为 $f_{\text{tt}}(\text{NULL},\text{NULL})$,即第一个请求不是 main page 的异常度,并且令 main page 长度为 0,然后转入状态 S_2,如图 3(a)所示.

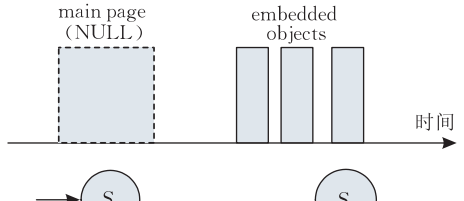
2. 处于 S_1,收到内嵌对象后,转入状态 S_2;如果在 S_1 收到一个 main page,则表示上一个 main page 无内嵌对象,两个 main page 的到达时间间隔为 thinking time,仍处于状态 S_1,如图 3(b)所示.

3. 在 S_2 中,直到收到一个 main page,转入 S_1. 或者出现如下两种情况,也由 S_2 转入 S_1,然后由 S_1 转入 S_2 中.

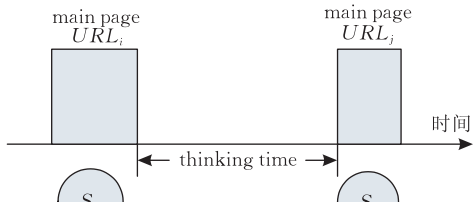
3.1. 相邻内嵌对象的时间间隔大于 EOBJ_INTV_THD(EOBJ_INTV_THD 为前后两个内嵌对象的间隔时间阈值),转入 S_1,如图 3(c)所示. 令 main page 为 NULL,thinking time 为此两个相邻内嵌对象的时间间隔,如果上一个 main page 为 URL_i,则转发异常度概率为 $f_{\text{tt}}(URL_i,\text{NULL})$. 在网络状况良好的情况下,同一个 main page 的内嵌对象请求到达 defense proxy 的时间接近. 如果相邻两个内嵌对象时间间隔大于 EOBJ_INTV_THD,则认为分属于两个 main page,且第 2 个 main page 为 NULL. 在静态 main page 被网络 cache 缓存时,部署于服务器前端的 proxy 接受不到 main page,导致内嵌对象归属识别失败,引入 EOBJ_INTV_THD 可以降低误差. EOBJ_INTV_HTD 的大小通过学习确定.

3.2. 一个 main page 内嵌对象总数大于 $(1+\alpha)\times$ EOBJ_MAXNUM($0<\alpha<1$),转入 S_1,如图 3(d)所示. EOBJ_MAXNUM 为训练时一个 main page 的最大的内嵌对象数. 令 main page 为 NULL,thinking time 为此两个相邻内嵌对象的时间间隔,如果上一个内嵌对象的 mian page 为 URL_i,则转发异常度概率为 $f_{\text{tt}}(URL_i,\text{NULL})$. 攻击者可以采用请求内嵌对象请求进行攻击,引入 EOBJ_MAXNUM 可以将内嵌对象分配到多个为空的 main page,然后利用 session 异常度模型识别出攻击 session.

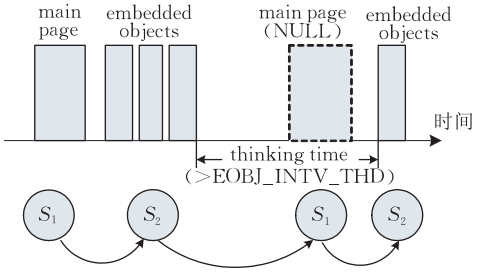
4. 除上述 3 种情况,在 S_2 收到内嵌对象,仍然处于 S_2 状态,统计对应 main page 内嵌对象数.



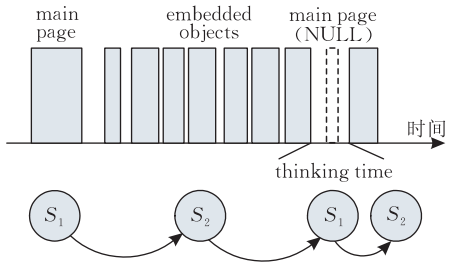
(a) 无main page处理



(b) 连续main page处理



(c) 相邻内嵌对象间隔超过EOBJ_INTV_HTD处理

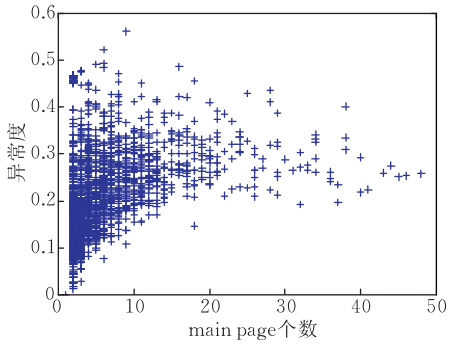


(d) 连续内嵌对象数超过EOBJ_MAXNUM处理

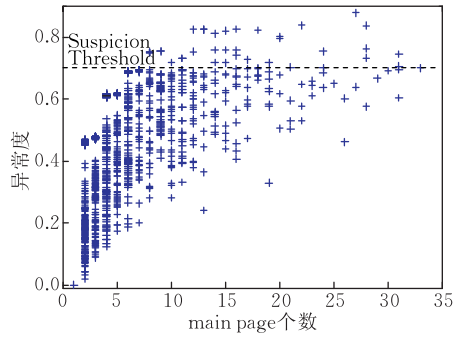
图 3 Session 异常度计算状态转换处理

4.4　异常度模型性能

采用真实日志 EPA-HTTP[①],共 15 小时 21 分 52 秒的数据作为训练集,进行异常度模型训练. 剩下的 8h37min58s 日志作为正常请求验证异常度模型的有效性. 通过一个预先设定的阈值来区分攻击 session 和合法 session,如果一个 session 异常度高于阈值,则认为其为攻击 session. 训练中 session 的最终异常度如图 4(a)所示. 为了减小误报率,本文将异常度阈值设为 0.7. 合法 session 的最终异常度如图 4(b)所示,错报率为 1.2%.



(a)训练集中session最终异常度分布



(b)合法请求session最终异常度分布

图 4　Session 最终异常度

4.4.1　平均异常度

分析正常 session 和上述 5 种攻击 session 对应的异常度,攻击具体情况如下:

(1) Single-URL Flood 采用训练集中的所有 main page URL 作为攻击请求,计算所有攻击 session 的平均异常度.

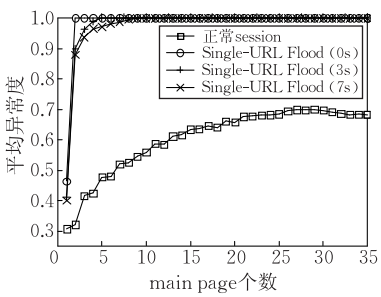
(2) Multi-URL Flood 随机选取训练集中的 URL 请求(包括 main page URL 和内嵌对象 URL)构成攻击序列.

(3) Random-URL Flood 的所有请求均来自训练集,每次依据当前请求的 main page 的超链接,随机选取一个新的 main page 作为新的攻击请求,考察攻击 session 在不同攻击间隔时间下的平均异常度.

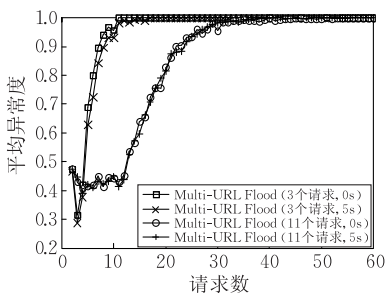
(4) Session Flood 的所有 seed session 均来自训练集,分别采用了包含 7 个请求,23 个请求和 37 个请求的 session 作为 seed session.

(5) Forged-URL Flood 的所有伪造 URL 长度为 800 字节,大于所有合法的 URL 长度.

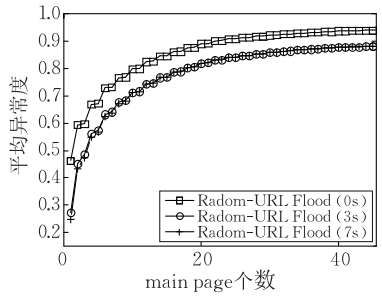
合法 session 和各种攻击 session 的平均异常度随 main page 个数的变化关系如图 5 所示,可以得



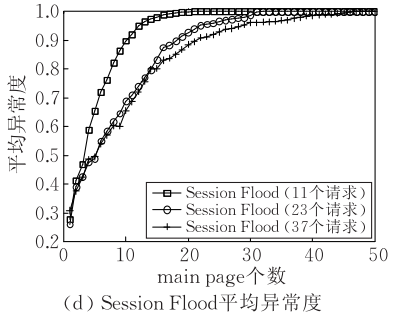
(a) Single-URL Flood 平均异常度



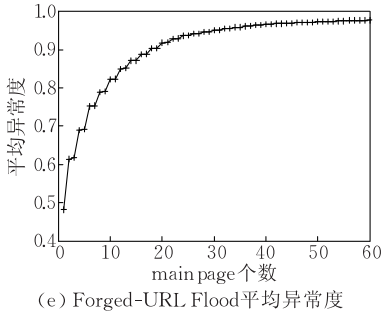
(b) Multi-URL Flood 平均异常度



(c) Random-URL Flood 平均异常度



(d) Session Flood平均异常度



(e) Forged-URL Flood平均异常度

图 5　正常访问 session 和 5 类攻击 session 平均异常度

① http://ita.ee.lbl.gov/html/contrib/EPA-HTTP.html

到如下结论：

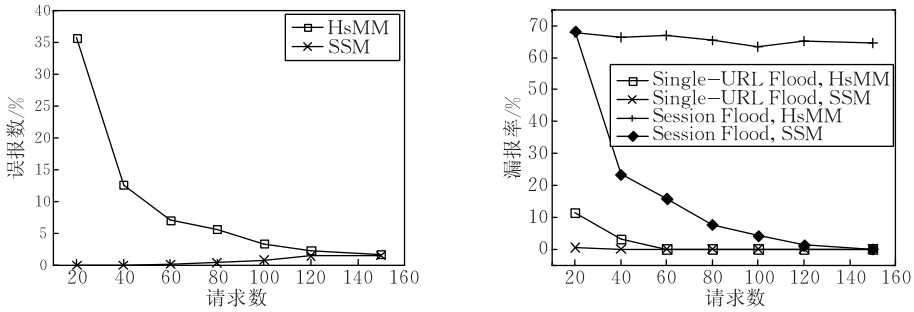
(1) 从图 5(a)可以看出，真实 session 平均异常度低于异常度阈值，同时，异常度阈值能够识别所有的攻击 session，因而漏报率为 0.

(2) 不同攻击频率的 Single-URL Flood 攻击在两个 main page 后平均异常度大于阈值；Multi-URL Flood 攻击最多在 18 个 request 后(包括 main page 请求和内嵌对象请求)，平均异常度大于阈值；Random-URL Flood 和 Forged-URL Flood 攻击在 6 个 main page 之后，平均异常度大于阈值.

(3) 利用包含 7 个 request 的 seed session 进行 Session Flood，在 6 个 main page 后平均异常度大于阈值；相应地，seed session 包含 23 个请求，11 个 main page 后平均异常度大于阈值；seed session 包含 37 个请求，12 个 main page 后，平均异常度超过阈值. Seed session 规模越大，异常度增加越慢，需要接收到更多的 main page 来与合法 session 相区分.

4.4.2 识别误差

分析识别误差随请求数的变化关系(与文献[14]工作相比较)如图 6 所示. 分析 Single-URL Flood 和 Session Flood 攻击下的误报率，Multi-URL Flood、Random-URL Flood 和 Forged-URL Flood 攻击的误报率与 Single-URL Flood 类似(SSM 为本文所提方法，HsMM 为文献[14]中的方法).



(a) 误报率 (b) 漏报率

图 6 Session 识别的误报率和漏报率

与文献[14]相比，本文所提方法的误报率和漏报率均较低，且文献[14]方法需要接收到足够多的请求才能降低误报率和漏报率，因而本文所提方法在接受到较少的请求时，就可以识别攻击 session，降低了转发时延. 同时，文献[14]无法有效识别 Session Flood 攻击的攻击 session.

5 转发模型

采用与文献[9]相似的转发机制，如图 7 所示. 每个 session 对应一个请求队列，转发策略每次选择一个 session 的请求队列，转发其头部请求. 如果队列满，则丢弃相应到达请求. 本节为转发机制选择合适的转发调度策略和转发速率.

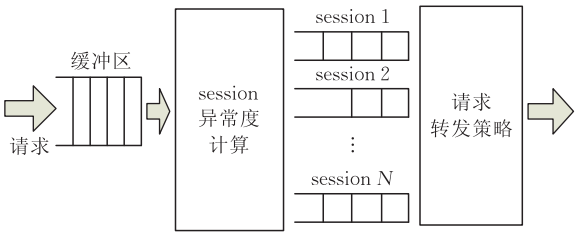


图 7 转发模型

5.1 转发调度策略

限制请求转发速率，依据 session 异常度进行请求转发，考察如下 3 种转发策略：

(1) 最小异常度优先转发(Lowest Suspicion First (LSF) Scheduler)

每次选择异常度大于 0，且最小的 session，转发其最早到来的请求. 如果不存在，选取一个异常度等于 0 的 session，转发其最早到来的请求. 依据上文，接受到第 i 个 main page 时，计算第 $i-1$ main page 的异常度. 异常度为 0 表明此 session 还没有接受到第 2 个 main page，无法判断此 session 是否异常，为了及时转发合法请求，在转发速率许可的情况下，转发尚无法判断是否异常的请求.

(2) First-Come First-Serve (FCFS)

每次转发所有异常度小于识别阈值且异常度大于 0 的 session 的最早请求. 如果不存在，选取一个异常度为 0 的 session，转发其最早到来的请求.

(3) Round Robin

逐次转发所有异常度小于识别阈值且异常度大于 0 的 session 的最早请求. 如果不存在，选取一个异常度为 0 的 session，转发其最早到来的请求.

5.2 实验设置

与上文相同，仍然利用 EPA-HTTP 日志进行模拟. 考察本文提出的 session 异常度模型分别结合上述 3 种转发策略对本文所提 5 种应用层攻击的过

滤性能. 每一个 session 对应的 queue 最多能存储 50 个 request,服务器的处理能力为 20MB/s,服务器缓冲区最多能存储 1000 个 requests.

用 15h21min52s 的日志作为训练集进行模型训练,8h37min58s 的日志作为正常请求,正常请求被切分成 2227 个 session,构成合法session 集. 从合法 session 集中随机选取 session 构成合法访问 session,每秒开始 15 个 session,最大请求速率为 103 个/s. 共有 200 个攻击session,从模拟开始发送请求直到模拟结束,除 Session flood 攻击,每个攻击 session请求发送速率均为 1 个/s. 考察不同的转发策略在合法请求返回时延随时间的变化关系.

定义合法 session 为真实合法访问 session,定义正常 session 为异常度小于识别阈值的 session.

5.3　性能分析

本节考察 Session Flood 时,LSF、FCFS 和 Round Robin 的转发速率. 3 种转发策略下最大转发速率对合法请求返回时延的影响分别如图 8(a)～(c)所示. 可以得到如下结论:

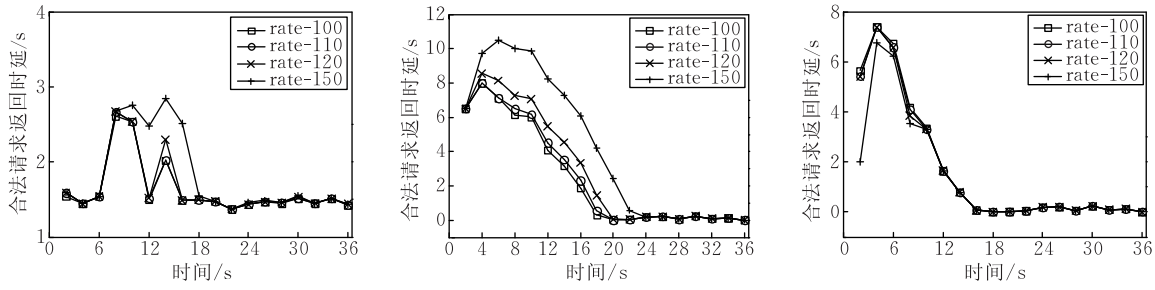
(1) 约 20s,3 种转发策略在不同转发速率下对应的返回时延均下降到稳定值,且不同转发速率下,合法请求返回时延相同. 此时,攻击 session 已被识别,且因攻击 session 未识别而在队列中等待的合法请求已经被全部转发.

(2) 在返回时延下降到稳定值前,LSF 对应的合法请求返回时延随转发速率的增加而增加. 在时延达到稳定后,LSF 仍然维持了较高的平均时延(约 1.4s).

(3) 在返回时延下降到稳定值前,FCFS 对应返回时延随最大转发速率的增加而增加. 在时延达到稳定后,FCFS 维持了较低的平均时延(<0.2s).

(4) 在返回时延下降到稳定值前,Round Robin 对应的合法请求返回时延随最大转发速率的增加而减小. 在时延达到稳定后,Round Robin 维持了较低的平均时延(<0.2s).

(5) 为及时转发合法请求,转发速率应不低于合法请求最大速率. 过大的转发速率易导致服务器过载,另一方面,转发速率的增加,并不会降低合法请求返回时延. 因此,转发速率为合法请求最大速率可获得较好的转发效果.



(a) LSF合法请求返回时延　(b) FCFS合法请求返回时延　(c) Round Robin合法请求返回时延

图 8　LSF、FCFS 和 Round Robin 策略下合法请求返回时延

Single-URL Flood, Multi-URL Flood, Random-URL Flood 和 Forged-URL Flood 攻击下的分析与 Session Flood 类似,区别仅在于返回时延下降到稳定值的时间比 Session Flood 短.

6　讨　论

上文已对 App-DDoS 攻击 session 的识别和过滤进行了详细阐述,下面对几个相关问题进行讨论.

6.1　异常度模型更新

采用上文所述方法建立的模型为静态模型,网站内容的更新和用户访问行为的改变,会导致识别误差逐渐增加. 本节提出了模型在线更新方法,如图 9 所示.

模型在线更新算法:
1. 通过训练建立 session 异常度模型;
2. 定义异常度阈值 $T_{anomaly}$ 和模型更新时隙 Δ;
3. 统计合法 session 的可疑度参数;
4. 重复步骤 3,直到模型更新时隙 Δ 结束;
5. 建立更新模型 update-model,更新 session 可疑度模型;
6. 重新设置模型更新时隙 Δ,重复步 3～5.

图 9　异常度模型更新算法

由于网站不会频繁更新,步 2 中的模型更新间隔 Δ 无需设置过长,比如 10min. 而异常度阈值 T_{anomaly} 即为 3.4 节所提的识别阈值,如果一个 session 的最后异常度度小于 T_{anomaly},则视为合法 session.

依据当前时间和此 session 最后一个请求的时间间隔,识别步 3 的 inactive session,如果时间间隔

大于阈值 T_{inactive}，则认为此 session 已经结束，不再活跃. 依据一个 session 最终异常度，判断一个 session 是否合法，如果最终异常度大于识别阈值，则 session 为合法. 每个 session 维持一个数据结构，记录此 session 的 main page，main page 的内嵌对象数，thinking time. 并且统计在一个 model update interval 内的 URL 长度分布统计值、main page 请求统计值以及各 main page 对应的转移页面统计值、内嵌对象数统计值和 thinking time 统计值.

在步 5 中利用上述的各统计值建立 update-model，建立方法与异常度模型建立方法相同. 然后利用 update-model 和已有异常度模型，对异常度模型进行更新. 更新方法如下：

URL 长度分布异常度和 main page 请求数异常度更新方法相似，以 URL 长度异常度为例. 假设异常度模型中，长度为 l 的 URL 的异常度为 f_{urllen}，update-model 中长度为 l 的 URL 的异常度为 $f_{\text{urllen}}^{\text{upd}}$，则

$$f_{\text{urllen}}=\alpha\times f_{\text{urllen}}+(1-\alpha)\times f_{\text{urllen}}^{\text{upd}},\ 0<\alpha<1.$$

一个 main page，其页面转移异常度、内嵌对象数异常度和 thinking time 异常度更新方法相似，以页面转移异常度为例. 在异常度模型中，一个 main page URL_i 转移到 URL_j 的异常度为 $f_{\text{transit}}(i,j)$，在 update-model对应的转移异常概率为 $f_{\text{transit}}^{\text{upd}}(i,j)$，则

$$f_{\text{transit}}(i,j)=\alpha\times f_{\text{transit}}(i,j)+(1-\alpha)\times f_{\text{transit}}^{\text{upd}}(i,j),\quad 0<\alpha<1,$$

α 的大小决定了模型的敏感性和稳定性. α 越大，模型更新速度越慢，也越稳定. α 越小，模型越敏感，越不稳定.

6.2 计算复杂性分析

6.2.1 异常度模型建立的复杂性

在线学习或离线学习建立模型的计算复杂性分析相似. 假设训练集包含 N 个请求，其中有 M 个不同的 main page. 对请求的处理为两个过程. (1) 查找请求对应的session，同时，为了计算异常属性值，如果请求的是 main page，需对转移页面数和各 thinking time 值进行统计累加，如果是内嵌对象，则需对内嵌对象数进行统计累加，此外需对 URL 长度和请求数进行累加；(2) 计算各异常属性值.

过程(1)中，对请求的源 IP 地址做 Hash，利用 Hash 值查找相应的 session，时间复杂性为$O(1)$. 过程(2)中，对 main page 的 URL 做 Hash，利用 Hash 值查找对应的 main page，时间复杂性为$O(1)$. 所以建立异常度模型的时间复杂性为 $O(1)$.

假设内嵌对象最大数目为 E，thinking time 最大长度为 T，每个 main page 维持一个内嵌对象分布大小数组和 thinking time 分布大小数组. 采用矩阵保存转移概率信息，则内存消耗为 $O(M^2+M\times E+M\times T)$，通常 $M\gg E$，$M\gg T$，因此，模型建立后，保存上述信息，内存消耗为 $O(M^2)$. 另一方面，如果在模型建立时，某一时刻最多有 S 个完成的 session，则内存消耗为 $O(S)$. 所以计算异常度模型的空间复杂性为 $O(M^2+S)$. 判断一个 session 是否完成或 session 是否活跃，依据当前时间和其最后一个请求的时间间隔. 可见训练集对用的流量速率越大，S 越大，相应的内存消耗越大.

6.2.2 session 异常度计算复习性

收到一个请求，需查找对应的 session 和 main page，用请求的源 IP 地址和 main page URL 的 Hash 值做查找，相应的计算复杂性为 $O(1)$.

如果进入 S_1，则进行 main page 异常度计算，查找相应的异常属性值和计算 session 异常度的时间复杂性均为 $O(1)$. 如果进入状态 S_2，需进行内嵌对象数累加，相应的时间复杂性为 $O(1)$. 可见，session 异常度计算时间复杂性为 $O(1)$.

对空间复杂性的分析与模型建立相同，为 $O(M^2+S)$，其中 M 为不同的 main page 数，S 为同时活跃的 session 最大数目.

6.3 相关工作比较

本文与文献[9]均提出一个 session 行为可疑度计算模型，然后根据可疑度进行请求转发. 但有如下几点不同：

(1) 本文所提异常度模型基于请求长度、请求数量、页面转换概率、请求循环次数、请求间隔时间和 main page 的内嵌对象数分布，上述信息均可在服务器外获得，因此本文所提可疑度模型可以在服务器外建立和在线更新；文献[9]中 session 可疑度模型基于 system log 建立，包括 session 传递时间、请求传递时间和 session 资源消耗轮廓，因此，不能实现对服务器的透明和在线更新.

(2) 本文通过模拟发现，在转发速率为最大合法请求转发速率时，就可获得较好的转发性能；而文献[9]采用了一种随可疑度之和变化的转发速率. 本文的转发机制更易于部署和实现.

(3) 文献[9]在攻击 session 和合法访问 session 同时长时间活跃的前提下分析不同转发策略的性能，而本文在攻击 session 保持活跃，合法访问 session 在一定时间后停止的前提下分析转发性能，

因而本文的实验更接近真实情况.

(4) 文献[9]表明 Proportional to Suspicion Share(PSS)也具有较好的转发性能,合法请求返回时延与 FCFS 和 Round Robin 接近.然后,由于无法准确预测活跃 session 的数量,大量的转发速率分配给已不再活跃的 session,导致转发速率大量浪费 Single-URL Flood 攻击下,合法请求转发速率约为 FCFS 和 Round Robin 的 74%.

文献[14]提出的模型与本文所提的 session 行为可疑度模型均可实现对服务器的透明.但有如下不同:

(1) 文献[14]从请求速率、观察时间和请求序列建立模型,而本文模型进一步包含了请求伪造、请求数分布和请求循环次数等攻击特征参数.

(2) 本文的计算时间复杂性为 $O(1)$,而文献[14]时间复杂性为 $MN'+N'D$.

(3) 文献[14]无法有效抵御 Session Flood 这种较隐蔽的攻击.

7 总　结

本文首先将 App-DDoS 攻击分为 5 类,分析了此 5 类攻击与正常访问行为的区别,提出了 session 异常属性和这些属性的计算方法,并基于此,提出了 session 异常度模型和 session 异常度计算过程.分析了异常度模型的最终识别误差和识别误差随请求的变化关系,比较了合法 session 和 5 类 App-DDoS 攻击 session 的异常度,验证了 session 异常度模型的有效性.采用真实网络日志进行模拟,将异常度模型与 3 种转发策略相结合,分析各种转发策略下合法请求返回时延随时间的变化关系.结果表明,转发速率为最大合法请求速率,就可及时转发合法请求.在识别出攻击 session 后,FCFS 和 Round Robin 的合法请求返回时延较低(<0.25s),具有较好的转发性能,LSF 对应的返回时延仍然较高(约 1.4s).

Session 异常度模型无需了解请求对服务器的资源消耗,只需获得请求 URL 以及请求时间等信息,可在服务器外部署.同时,session 异常度计算时间复杂性为 O(1),具有较快的计算速度,可实现对 session 异常度的在线计算.

参　考　文　献

[1] MyDoom virus [online]. http://www.us-cert.gov/cas/techalerts/ta04-028a.html

[2] Chen X, Heidemann J. Flash crowd mitigation via adaptive admission control based on application-level observations. ACM Transactions on Internet Technology, 2005, 5(3): 532-569

[3] Jung J, Krishnamurthy B, Rabinovich M. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites//Proceedings of the 11th IEEE International World Wide Web Conference. Honolulu, Hawaii, USA, ACM, 2002: 252-262

[4] Xie Y, Yu S. Monitoring the application-layer DDoS attacks for popular websites. IEEE/ACM Transactions on Networking, 2009, 17(1): 15-25

[5] Li K, Zhou W, Li P, Hai J, Liu J. Distinguishing DDoS attacks from flash Crowds using probability metrics//Proceedings of the 3rd International Conference on Network and System Security. Gold Coast, Queensland, Australia, 2009: 9-17

[6] Yu S, Thapngam T, Liu J, Wei S, Zhou W. Discriminating DDoS flows from flash crowds using information distance//Proceedings of the 3rd International Conference on Network and System Security. Gold Coast, Queensland, Australia, 2009: 351-356

[7] Oikonomou G, Mirkovic J. Modeling human behavior of defense against flash-crowd attacks//Proceedings of the IEEE International Conference on Communications. Dresden, Germany, 2009: 14-18

[8] Park K, Pai V, Lee K, Calo S. Securing Web service by automatic robot detection//Proceedings of the Annual Conference on USENIX'06 Annual Technical Conference. Boston, USA, 2006: 23-28

[9] Ranjan S, Swaminathan R, Uysal M, Knightly E. DDoS-Shield: DDoS-resilient scheduling to counter application layer attacks. IEEE/ACM Transactions on Networking, 2009, 17(1): 26-39

[10] Kandula S, Katabi D, Jacob M, Berger A. Botz-4-scale: Surviving organized DDoS attacks that mimic flash crowds//Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation. Kyoto, Japan, 2005: 287-300

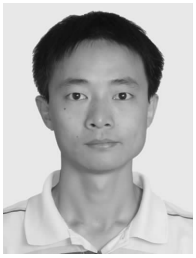
[11] Walfish M, Vutukuru M, Balakrishnan H, Karger D, Shenker S. DDoS defense by offense//Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. Pisa, Italy, 2006: 303-314

[12] Kruegel C, Vigna G. Anomaly detection of Web-based attacks//Proceedings of the 10th ACM Conference on Computer and Communications Security. Washington, DC, USA, 2003: 251-261

[13] Yu J, Chen H, Chen X. A detection and offense mechanism to defense against application layer DDoS attacks//Proceedings of the 3rd International Conference on Networking and Services. Athens, Greece, 2007: 251-261

[14] Xie Y, Yu S. A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors. IEEE/ACM Transactions on Networking, 2009, 17(1): 54-65

[15] Gavrilis D, Chatzis I, Dermatas E. Flash crowd detection using decoy hyperlinks//Proceedings of the 2007 IEEE International Conference on Networking, Sensing and Control. London, UK, 2007: 466-470



XIAO Jun, born in 1979, Ph. D. candidate. His research interests include network security and overload control.

YUN Xiao-Chun, born in 1971, Ph. D., professor, Ph. D. supervisor. His research interests include the network security, Internet model and Internet measurement.

ZHANG Yong-Zheng, born in 1978, Ph. D., associate professor. His research interests include network security and network security evaluation.

Background

Distributed Denial-of-Service (DDoS) attack is a major threat to the current cyber system. Current defense based on network-layer traffic feature is effective to detect network-layer DDoS attacks like SYN flood attacks. However, these defense methods are not useful for DDoS attacks based on application layer, because the network-layer traffic feature seems normal when App-DDoS attacks happen. Turing tests can differentiate normal users from bots, however, these methods may annoy users. Speak-up method is not feasible for its more overload impact to the server. Other methods are not workable because the detection model can not be updated online or their computational complexities are very high, which make their real time application unpractical.

In order to defend against App-DDoS attacks, the authors propose a session suspicion model and analyze the performances of 3 different forwarding policies. This suspicion model can differentiate normal sessions from App-DDoS sessions accurately. The computational complexity of this model is $O(1)$, which makes the online identification practical. Besides, this model is independent of Web servers. In addition, the proper forwarding rate and policy are also analyzed. So, the work is practical and can be implemented in an application firewall or a defense proxy.

This paper is supported by the National Natural Science Foundation of China under grant No. 60703021, the National High Technology Research and Development Program (863 Program) of China under grant Nos. 2007AA010501, 2007AA01Z474, 2007AA01Z467.

The authors aim at a practical defense method against the application-layer DDoS attacks. Several security systems have run in real Internet environment and several papers have always been published, including the normal behavior model, scheduler policy under attacks, and so on.