2018 年全国大学生信息安全竞赛

作品报告

作品名称：基于流量的自反馈恶意软件监测系统

电子邮箱：369150573@qq.com

提交日期：2018.06.05

填写说明

1. 所有参赛项目必须为一个基本完整的设计。作品报告书旨在能够清晰准确地阐述（或图示）该参赛队的参赛项目（或方案）。

2. 作品报告采用 A4 纸撰写。除标题外，所有内容必需为宋体、小四号字、1.5 倍行距。

3. 作品报告中各项目说明文字部分仅供参考，作品报告书撰写完毕后，请删除所有说明文字。(本页不删除)

4. 作品报告模板里已经列的内容仅供参考，作者可以在此基础上增加内容或对文档结构进行微调。

5. 为保证网评的公平、公正，作品报告中应避免出现作者所在学校、院系和指导教师等泄露身份的信息。

目　　录

摘要

近年来，互联网安全问题一直是全球范围内的重要议题。恶意软件作为网络环境的重大威胁之一，具有难侦查，变体多的特点。由于用户的安全意识的疏忽，恶意软件一旦进入个人设备，会对其所在的机构的整个网络环境带来威胁，造成隐私数据泄露、远程控制等损害。因此，对于各类组织机构而言，如何有效检测并定位其网络系统中的恶意软件已经成为了维护信息安全的首要任务。而 HTTP 作为绝大多数程序进行通信的方式，其流量特征具有很好的辨识度，通过 HTTP 流量的检测能有效识别恶意软件。

目前，传统的网络流量实时检测系统往往仅使用黑名单、签名匹配等机制来对流量进行检测过滤，其对变化的网络环境缺乏适应性，导致精确度不够理想，并且系统在检测到恶意流量后，无法对内部源头进行精确定位溯源；当前恶意软件的最主要检测方式——单机检测，需要在个人机器上部署，针对单个设备进行检测，其精确度虽高，但对于一个大型组织而言非常难以控制和管理，缺乏效率。总体而言，现有系统在检测范围、工作效率、精确度以及对于环境变化的适应性上难以做到兼备，同时传统的检测方式往往需要人工对于特征进行分析，成本较高。针对该问题，我们提出了基于 HTTP 流量的自反馈恶意软件监测系统。

我们的系统是一个可通过自反馈持续学习的，高精度的实时恶意软件监测系统，其工作在网关，按时间窗对 HTTP 流量进行抓取并进行分析。在数据预处理阶段，为提高系统工作效率，确保检测的实时性，我们将流量以流(stream)的形式处理，使用 WebGraphic 请求图算法对 HTTP 流量中大量的来自浏览器的部分进行清洗，并使用指纹算法检测识别同类流量，避免数据的重复处理。在检测阶段，我们采用深度学习算法，使用采集的标记数据集训练模型，使其自动提取特征进行学习。系统在实际工作的前期，我们引入迁移学习的机制，提高模型检测效果；在工作后期，我们引入反馈学习机制，使系统持续学习并适应变化的流量环境。

相比同类产品，我们的系统兼备以下特点：检测范围广，能对整个网络系统中的大多数恶意软件进行检测；数据的预处理使得数据量大大减少，确保了系统工作的效率；使用深度学习算法相对传统的 HTTP 恶意流量检测方式免去了人工提取特征的工作，减少成本；通过迁移学习和反馈学习机制，系统能够不断适应流量环境与

恶意软件的变化。此外，IP 加上 User-Agent 字段的流聚类处理方式使粒度更小，能在最终的检测结果中提供更多的信息，帮助在系统中进行恶意软件的定位与溯源。

Abstract

In recent years, cyber security has long been a major subject worldwide. With difficulties on investigation and countless variations, malware is one of the primary threats to network environment. Once malware plants on personal devices since user’s misbehavior and neglect, it endangers the whole network of the organization where the devices connect, and may cause leakage on privacy data and intrusive remote control, etc. It is, therefore, to detect and locate malware in its network system that has been the primary duties to maintain information security for an organization. Since high visibility in traffic characteristics, HTTP, the most popular networking protocol for programs, can be of great efficiency on malware detection.

Traditional real-time network monitoring system, which uses blacklist, signature matching and other mechanisms to monitor and filter traffic, is lack of adaptability to the changing environment, not ideal on accuracy and unable to precisely locate its internal source after malware detected. And stand-alone detection, the most popular method, requires distribution on PCs. Though high accuracy, it is inefficient for a large organization to control and manage such system. Overall, existing systems cannot archive all following, such as monitoring scope, adaptability, efficiency and accuracy, etc. Traditional monitoring methods, meanwhile, usually requires artificial feature extraction with higher cost. Along with the issues above, we have proposed a self-feedback malware monitoring system based on HTTP traffic.

It is designed as a real-time malware monitoring system with

sustainable feedback on training and high precision on prediction. The system works at the gateway, capture and analyze HTTP traffic by time scales. At preprocessing stage, we process the traffic as streams for efficiency, adopt WebGraphic algorithm to purge portions of traffic from web browsers, then extract fingerprints to recognize and eliminate similar traffic. At detection stage, we use neural network to produce detection model. Before distribution, we introduce transfer learning to improve the model; and later, we use feedback learning to adapt the model to environment changes.

Compared to existing products, the system has following features: wide monitoring scope, is able to detect most malware in the entire network; preprocessing, greatly reduces data scale and ensures the efficiency of system; neural network, omits artificial feature extraction; transfer and feedback learning, adapts the system to vicissitude of network environment. Besides, cluster using IP and User-Agent field diminish the granularity and ensures more information on final detection report, which helps locate and trace malware in the system.

第一章 作品概述

1.1 项目背景

近年来，互联网安全问题一直是全球范围内的重要议题。随着新型网络攻击方式的快速发展，互联网的安全威胁正在日益激化。恶意软件作为主要网络威胁之一，极可能导致其所在的整个网络系统遭受攻击，造成敏感信息的泄露，这也使得在网络系统中，对恶意软件准确及时的识别与分类变得日益重要。卡巴斯基已经在2017 年第一个季度为世界各地 190 个国家的用户检测并阻止了 4.8 亿次恶意攻击。目前，卡巴斯基网络防病毒组件已将 8 千万个有唯一的 URL 的软件标记为恶意软件，根据卡巴斯基发布的统计数据（图 1）显示，这一数量仍然在迅速增加。

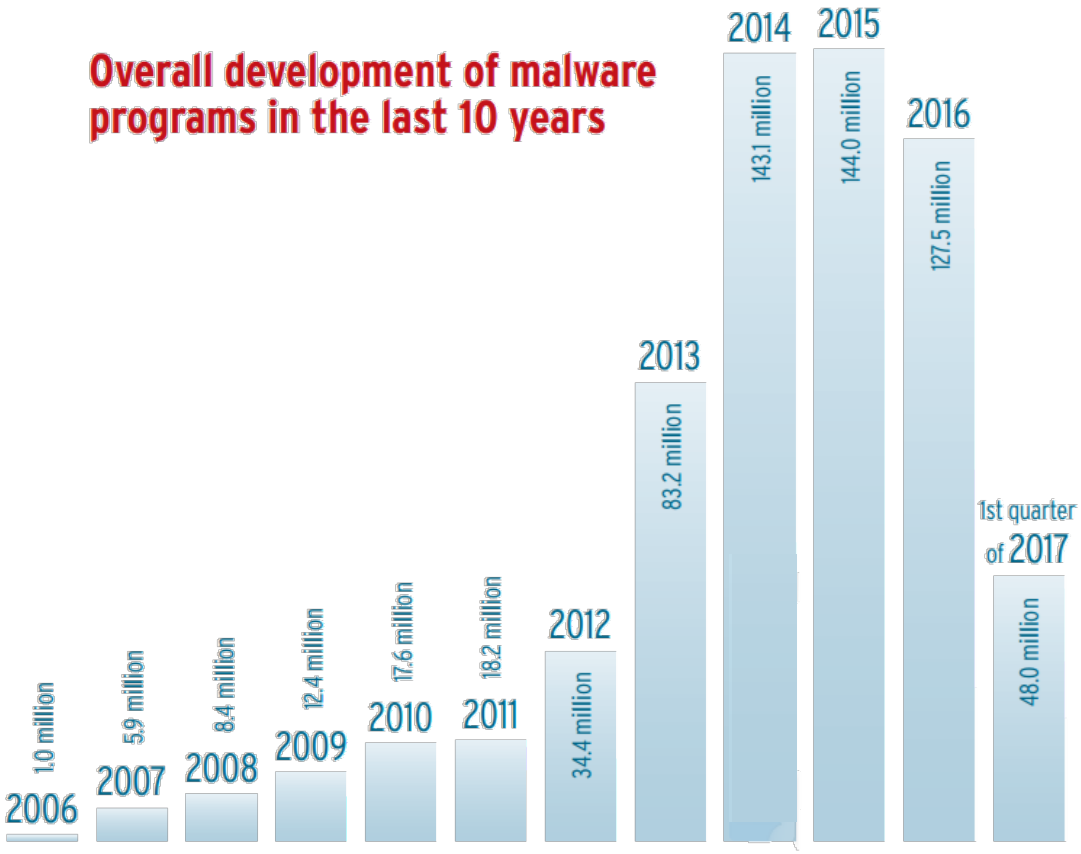

图 1 2006 年以来恶意软件增长统计

对于组织机构而言，一个内部人员的疏忽而引入的恶意软件可能最终造成不可预料的损失。如何在其整个网络系统环境中有效检测恶意软件已经成为了共同的话题。然而目前的恶意软件的检测的主要方式仍为单机检测，即检测系统需要部署在单个设备上进行工作。这种方式对于拥有较大网络系统的机构来说缺乏可行性，且

难以保证全面部署。而企业当中常用的网络流量检测方式通常是针对来自外部的一些攻击，其对于恶意软件的检测往往为了确保实时性只是使用了一些黑名单、签名匹配等机制进行判别，因此精确度和对变化的适应性不高。对于这些系统而言，需要一个在检测范围、实时性和精确性上做到兼备的恶意软件检测方案。

1.2 技术背景

1.2.1 软件 HTTP 流量

HTTP 流量是软件进行通信的最常用的一种方式。卡巴斯基报告显示，近年来，虽然越来越多的应用程序已经从 HTTP 切换到 HTTPS，截至 2018 年 1 月，有近 63%的应用程序正在使用 HTTPS，但是其中大多数仍没有放弃使用 HTTP。据图 2 统计数据所示，目前仍有将近 90%的应用程序正在使用 HTTP，其中许多人的敏感数据正在被未加密地传输。

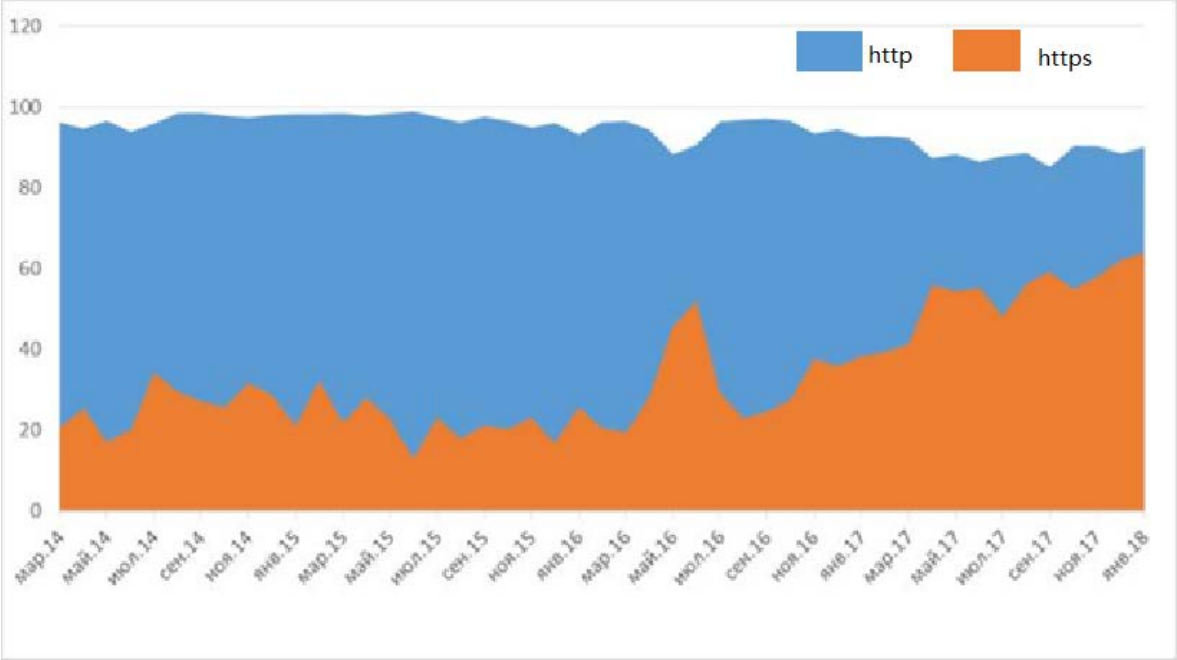

图 2 2014 年以来使用 HTTP/HTTPs 的软件比例

卡巴斯基在报告中也指明，许多恶意应用程序选择使用 HTTP 来对窃取到的用户数据进行传输。而这些恶意软件所产生的 HTTP 流量具有一定的特征，通过对这些特征进行识别分析，我们可以检测出恶意的 HTTP 流量，从而达到反过来识别恶意软件的目的。

同时，对于每个软件程序而言，其在 HTTP 请求中所使用的 User-Agent 字段往

往具有较高的辨识度，恶意软件多数也是如此。因此通过对恶意 HTTP 流量的检测，加上 User-Agent 所提供的信息，能够有效检测和定位系统中的恶意软件。

1.2.2 流量分析方式

在传统方式上，网络攻击的监测和拦截主要依赖于对已知攻击方法的分析，即提取可获得的关键字段和特征参数与流量数据进行模式匹配。然而，随着网络攻击技术的发展，这种基于 TCP 和 UDP 端口及有效载荷等监测异常流量的方式效率越来越低。以高级持久威胁（Advanced Persistent Threat, APT）为例，APT 利用攻击目标的某些未知漏洞部署长期任务，与正常的用户网络访问流量交织在一起。APT 攻击通常在网络流量上并没有一定的特征，或明显的行为模式，这使得传统的分类方法对其无效。

机器学习（Machine Learning, ML）技术[1]则提供了另外一种稳定而可靠的流量分析方法。其基于可靠且客观独立的统计学特征，而不是传统方法中的应用协议（或有效载荷）信息，如包长度和接收时间等。目前，基于 ML 的流量分类研究主要侧重于两种形式，一种是监督学习，即训练前实现对数据模型进行了分类和定义，另一种是无监督学习，也称为聚类。

而深度学习（deep learning）通常基于 ML 过程与现有数据配额进行比较，开发对网络情况的监测系统，生成用于区分正常和恶意数据流量的决策网络。由于数据传输与访问间差异性的存在，我们可以据此区分恶意攻击或正常访问，例如在网络流量中攻击原型和正常数据，并通过深度学习算法来帮助监测未知的 APT 攻击行为，该方式相比而言往往具有更好的效果。

1.2.3 现有检测技术

随着互联网的发展和防御方法的进步，恶意软件也不断更新换代，使人们不断寻找新方法进行恶意软件的检测。但主要的方式仍为以下两类：

第一类，传统的检测方式，即单机检测方式。该方法基于恶意软件的签名和特征码[2]，但这种方式无法检测未知的恶意软件。随着近年机器学习相关技术的不断成熟，研究者也将机器学习和数据挖掘技术应用于恶意代码检测。如 Saxe 等人介绍了一种使用二维二进制软件特征的基于深度神经网络的恶意软件检测方法[3]；Tobiyama 等人提出了使用软件行为特征的基于机器学习的恶意软件检测方法[4]；

Yuan 等人将机器学习应用于 Android 平台的恶意软件检测[5]。目前而言，基于机器学习的恶意软件的检测主要依靠恶意样本本身的特征，其检测也是针对系统内部软件的扫描。在检测上适应性还不够高，并且在网络系统中的检测范围十分受限

第二类检测方法是对网络流量的处理，许多学者都曾对此作过深入研究。如 Karagiannis 等人曾提出一种分类模型，其需要可访问的端口和（IP 地址）信息[7]，或端口和主机之间的流量模式[8]；Roughan 等人利用最近邻居模型（nearest neighbour）进行聚类以提供所需的分类[9]，而 McGregor 等人进行类似研究时由于使用了未预先标记的数据集进行分类，即无监督学习的方式，最终导致无法确定其模型的分类依据和模式[10]——这也是在研究中，无监督学习较少有人采用的主要原因；Gu 等人则采用了熵最大方法（entropy maximization method），通过流量预测判断网络状态[11]；一种特别设计的神经网络（neural networks）则由 Atiya 等人提出，使用稀疏选择（sparse-basis selection）预测视频流量的利用率[12]；此外，还有一些研究人员采用贝叶斯神经网络（Bayesian neural networks）[13] 或支持向量机（support vecter machine, SVM）[14] 等新兴技术进行网络流量分类的研究。而目前的网络入侵检测系统，通常还是依靠签名匹配，和服务器端口跟踪的传统方法作为主要的识别技术[15][16]。虽然其能够实时检测网络流量，但该方式缺乏灵活度，精度有待提升。

总结而言，现有最常见的恶意软件相关检测技术及其特点如表 1 所示。

表 1 现有恶意软件相关检测技术及其特点

检测方式	所用技术	特点	代表产品
单机检测	恶意样本本身特征码、签名、代码检测	检测精确度较高，但检测范围小，效率不高	360、电脑管家、金山毒霸等
传统网络流量检测	URL 黑名单、签名匹配、端口跟踪等	具有实时性，但对变化缺乏适应性，精度有待提升，需人工分析特征	卡巴斯基、f-secure、一些防火墙产品等

异常流量、行为的机器学习检测	机器学习模型检测	精度较高，但检测粒度只到 IP 层面，无法进一步对恶意软件精确定位溯源，模型面对特定应用场景和环境变化时仍有提升空间	蓝盾等企业网关安全产品

1.3 特色描述

针对以上的各主流恶意软件检测方式中所存在的不足与可优化之处，我们设计并实现了凭借对网关恶意 HTTP 流量的分析处理来识别并定位网络系统内的恶意软件的自反馈监测系统。

相比同类软件，我们的系统最大的特点为在检测范围、检测效率以及对变化的适应性上做到了较好的兼备，同时能够自动提取流量特征，并通过对流量的 IP 加上 User-Agent 字段聚类的精细粒度处理，能够在整个网络系统中帮助定位恶意软件具体所在。系统的各特点及对应实现如表 2 所示。

表 2 系统特点及对应实现

特点	实现方式
范围广	从网关 HTTP 流量检测整个系统
高效率	流方式处理数据、清洗浏览器流量、指纹判别同类流量
变化适应性	对深度学习检测模型引入迁移学习、反馈学习的机制
更精确定位	IP+User-Agent 进行 HTTP 流聚类，提高

	定位和溯源的精度
自动化	应用卷积神经网络，免人工特征提取，方便自动化更新模型

在数据预处理阶段，我们将 HTTP 流量以流的方式进行处理，使用 WebGraphic 请求图算法清洗大量来自浏览器的流量，并使用指纹算法判别同类流量，避免相似数据重复处理，从而大大减少了数据处理量，保证了系统工作效率及实时性。

在流量检测阶段，我们首先使用深度学习算法，在标记好的数据集上自动进行特征学习，训练基本检测模型。在系统工作前期，利用迁移学习机制引入人工干预，调整模型适应环境；在系统工作后期，通过反馈学习机制使得系统自动更新优化检测模型，保证其面对变化的流量环境和发展的恶意软件而保持有效性。

第二章 作品设计与实现

2.1 系统概述

我们的系统以在局域网的网关处按时间窗捕获的 HTTP 流量作为输入，经过分析处理后，输出结果为检测到的该网络系统中存在的恶意软件的信息。检测结果信息以源 IP、目的 IP、使用的 User-Agent 以及端口号的四元组的形式给出。根据该结果信息，网络系统维护人员能够通过一些如端口扫描的方式，来具体定位系统中某台设备上的恶意软件。整个系统结构如图 3 所示。

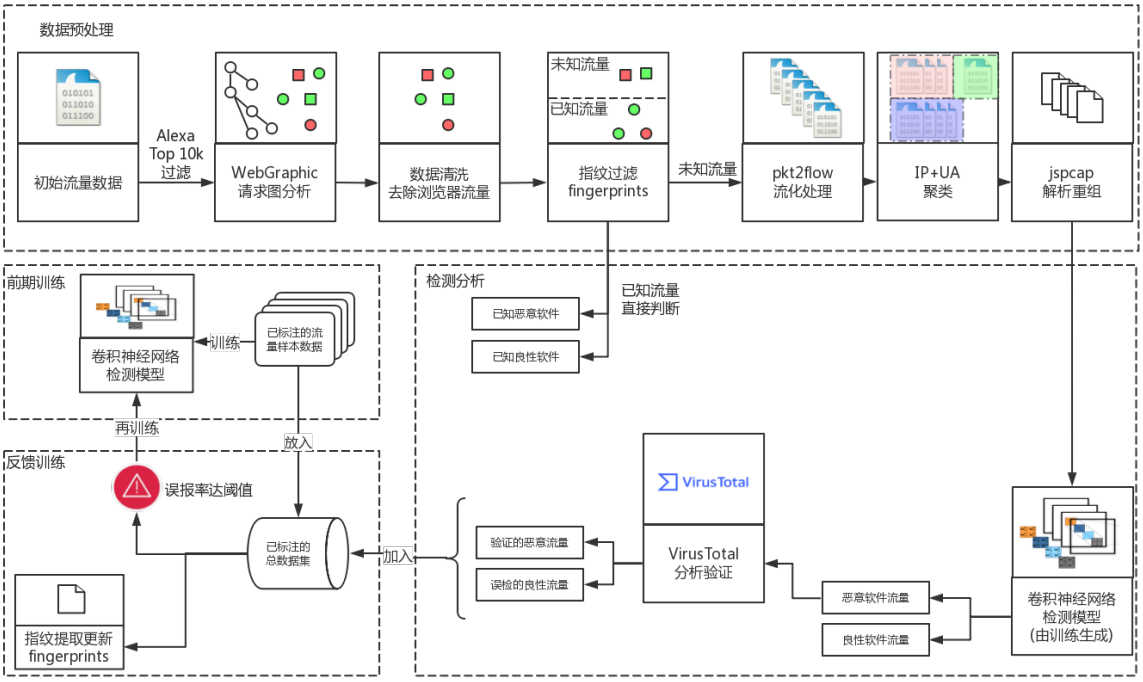

图 3 系统整体架构图

系统的结构主要分为数据的预处理和检测分析两个部分。

A. 数据预处理

对于数据预处理，其主要有两点目的：第一是对数据初步处理，减小数据量，提高系统的工作效率从而尽量保证检测的实时性；第二是减小后续预测步骤中数据的粒度，以便在检测结果中给出更精确的信息，从而能对恶意软件进行溯源定位。

首先，系统读入在网关处捕获的 HTTP 流量数据，使用 Alexa Top 10000 的名单对 HTTP 流量进行过滤。Alexa Top 名单中将世界网站按访问流量进行排名，因此这些域名能够被视作可信任的。该步处理往往能过滤掉超过一半的捕获的 HTTP 流量。

然后，考虑到捕获的 HTTP 流量中包含大量来自浏览器的部分，而浏览器的 HTTP 流量是由用户直接交互所产生的，是良性流量，这意味着我们只需要对后台软件产生的流量进行判别来检测恶意软件。因此，我们根据浏览器的 HTTP 请求相互之间存在关联的特点，使用 WebGraphic 请求图算法来对浏览器 HTTP 流量进行清洗。

在去掉了浏览器流量后，在剩下的后台软件 HTTP 流量中，由于同类软件的相似 HTTP 流量会多次在网关处出现，对这些具有一定相同特征的流量进行重复处理会消耗不必要的资源，因此，通过对流量生成指纹，可在之后对流量的处理中直接判别

相似流量，从而提高系统检测效率。

最后，我们将过滤得到的 HTTP 流量进行流化处理，并对这些流按照其 IP 和使用的 User-Agent 进行聚类。最终将同一类组中的 HTTP 流利用 jspcap 进行重组，对重组后的数据进行下一步的检测分析。这样处理的好处在于，首先通过将 HTTP 流量生成流并进行重组，一定意义上减少了数据处理量，保证了系统的效率；同时 IP 和 User-Agent 聚类的方式减小了粒度，使得在最后的检测结果中能够给出更精确的系统中的恶意软件的信息。

B. 分析检测

在检测分析阶段中，我们通过使用深度学习进行流量的分析，并引入迁移学习和反馈学习的机制，来提高检测精度：

首先，我们提前使用采集的良性与恶意的 HTTP 流量样本集，采用深度学习中的卷积神经网络来训练基本的检测模型。工作时，模型以进行重组后的 HTTP 流作为输入，给出判别的结果。

考虑到该检测模型极其依赖于训练所使用的 ground truth 样本数据集，而该样本集是从别处采集进行标注获取的，往往不能涵盖实际应用的新环境当中所有可能的流量特征，这就会导致检测系统精度不够理想，特别是误检率较高。因此，我们在该部分引入了迁移学习和反馈学习的机制。检测模型在应用到实际环境的初期，在人工指导下进行调整并适应新环境；然后在之后的工作阶段中，根据判断，自主更新数据集并进行重新学习调整，保证模型持续有效。检测过程的流程大致示意图如图 4。

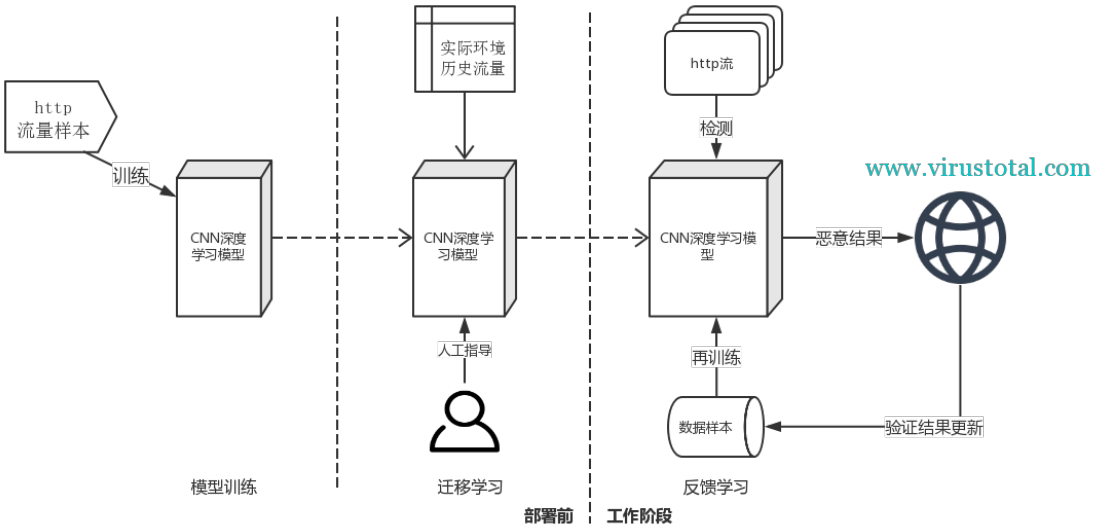

图 4 检测分析流程大致示意

2.2 设计与实现

2.2.1 数据预处理模块

2.2.1.1 浏览器流量清洗

A. 概述

经过网关的 HTTP 流量当中包含大量来自浏览器的部分，这系列流量与用户的直接交互相关，比如某个页面的加载，某个图片的点击请求等，是由用户亲自所操作产生，可视为良性流量。而那些由后台软件所发出的 HTTP 请求不包含用户的操作，是软件自行进行的一些比如更新查询、与服务器通信之类的行为，这也是恶意软件使用 HTTP 协议进行通信的行为方式，所以恶意软件的检测识别依赖于对后台软件产生的 HTTP 请求的处理。因此，为保证系统检测的实时性，我们选择使用 WebGraphic 网络图算法,对数据中的浏览器 HTTP 流量进行清洗，减小处理的数据量，使后续步骤集中于对后台软件的 HTTP 流量进行分析。

同时，因为浏览器 HTTP 请求和后台软件 HTTP 请求具有明显不同的特征（如浏览器 HTTP 请求的格式和内容变化频繁，而后台软件 HTTP 请求的格式相对稳定，具有固定的模式，其 HTTP 请求头具有固定的结构，HTTP 请求的大小固定，或者具有相同的数据内容等[22]），该处理也使得后续步骤中使用深度学习生成的检测模型更具有针对性。

B. WebGraphic 请求图算法

浏览器在用户的交互下产生 HTTP 请求，这些由用户操作产生的一系列请求一般来说具有明显的相关性。比如用户在浏览网页时，从一个站点点击链接跳转到另一个站点，或者在同一个主域名下不同页面间浏览，这些请求都存在明显的关系。而后台软件所产生的每个 HTTP 请求一般具有各自明确目标和任务，并不相互关联。根据这个特性，我们将捕获到的每个 HTTP 请求视为一个节点，依次按数据包的时间戳读入请求，将其置于设定为一定大小的时间窗之中。然后通过预定的规则不断在时间窗中发现并连接具有联系的节点，构建 WebGraphic 网络请求节点图。在所有请求读入并处理完毕后，成功建立起联系的节点即为来自浏览器类型软件的流量，而孤独结点则为来自后台软件的流量[23]。处理流程如图 5 所示：

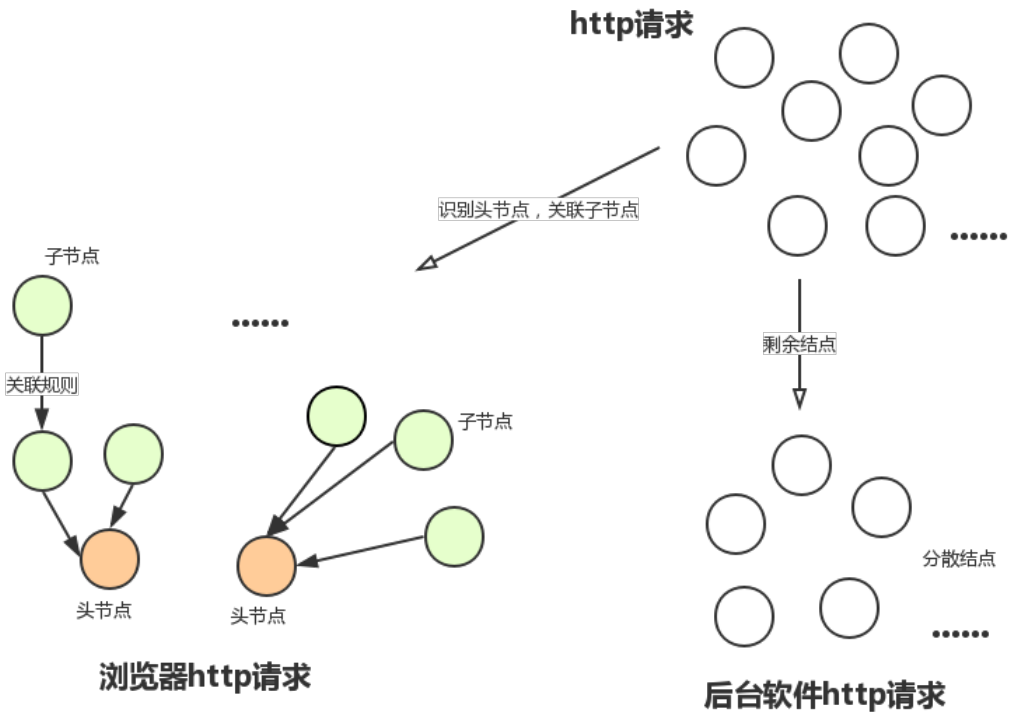

图 5 WebGraphic 算法

在处理过程当中，我们使用以下规则来识别 HTTP 请求的头节点并连结相关的子节点：

I. 根据 HTTP 请求的 Referer 字段在设定好的时间窗中找出前向节点，将其设置为头节点后，将所有指向相同 Referer 的子节点连结在一起；

II. 根据 HTTP 请求的 URL 包含的关键字，寻找对于常见的可视化页面相关资源的请

求，将其设置为头节点。目标请求的寻找采用关键字匹配的方法，所设置的关键字包括：html，php，htm，css，json，js，asp，aspx，jsp，shtml，stm，shtm[21]；

III. 对于包含 Referer 字段但在当前时间窗内找不到前向节点的 HTTP 请求，将其连接到与其 Referer 字段指向的头节点的域名相似的节点组当中。判断域名相似的方法为使用 Levenshtein 算法计算主域名的相似度，若相似度大于字段长度的 2/3 则判断为 True（注：对于给定的两个字符串 str1 和 str2，Levenshtein 算法能够计算 str1 修改为 str2 需要的最少字符变动次数）；

IV. 若 HTTP 请求的 Host 字段与已经存在的某个节点组中的域名相同，则将该 HTTP 请求加入该组。

2.2.1.2 指纹判断相似流量

A. 概述

在一个网络系统内部，由于软件种类总数总体来看处于相对稳定状态，同时，同类软件在工作时往往会产生具有类似特征的 HTTP 流量，在网关捕获的所有 HTTP 流量数据中，很多流量之间具有相似性，且在长时间段内，出现过的流量的类型总量变化不大。因此，通过对这些出现过的相似流量进行识别，我们的系统就能够在工作过程当中利用已有检测结果直接做出判断，在很大程度上提高了系统的工作效率。

由于依靠传统方式所使用的根据 User-Agent 来对同类流量进行识别并不够可靠[17]，同时恶意软件也可能伪造自身的 User-Agent[19][20] 影响识别结果。因此，我们选择根据每种软件所发出的 HTTP 请求在格式、大小以及某些字段所具有的特点，针对这些特征使用指纹生成算法来实现对相似的流量的识别。

B. 指纹提取算法

在该部中，我们根据 IP+UA 对 HTTP 流量(尚未进行流化处理)进行分组，以分组为单位，根据每组内包含的所有 HTTP 流量来进行该类流量的指纹计算。针对后台软件的流量在 HTTP 请求的格式和内容方面具有的特征，我们使用以下规则来提取其指纹。

我们使用以下字段来生成后台软件流量的指纹：

- User-Agent 字段
- IP 字段
- Constant Header（组内所有 HTTP 请求头部的公共字段）[18]
- Average Size（组内所有 HTTP 请求头部平均大小）
- Language 字段

C. 指纹识别算法

在指纹识别阶段，若之前生成的流量指纹列表为{FP_1, FP_2, FP_3, …}，通过遍历该列表，将新生成的流量指纹与每个指纹计算相似度，若与某个已有指纹相似度达到所设定的阈值，则该流量被识别为已知流量[22]。

识别过程中关键在于各个特征字段相似度分数以及相似度阈值大小的设定。若阈值设定过小，则会出现把新的流量识别为已知流量的情况；若设定过大，则会影响识别出相似流量的效果。但考虑到在本系统中指纹识别主要用途为提升系统效率，因此阈值宁愿设置较高，以避免在检测出现遗漏的情况。经过反复实验，最终确定分数阈值设定如下：

- 后台软件流量指纹相似度计算方式及分数阈值设置如表 3 所示。

表 3 后台软件流量指纹相似度分数

符号	参数	分数	
		相同	不同
F_1	User-Agent 字段	1.0	0.0
F_2	IP 字段	1.0	0.0
F_3	Constant Header	$\begin{cases} 0.5, \text{ if const\_headers1} \supseteq \text{const\_headers2} \\ \quad\quad \text{or const\_headers1} \subseteq \text{const\_headers2} \\ 0.0, \text{ otherwise} \end{cases}$	

F_4	Average Size	$\begin{cases} 1.0, \text{ if } \frac{2}{3}size1 < size2 < \frac{4}{3}size1 \\ 0.5, \text{ if } \frac{1}{3}size1 < size2 < \frac{2}{3}size1 \text{ or } \frac{4}{3}size1 < size2 \\ 0.0, \text{ otherwise} \end{cases}$	
F_5	Language 字段	0.5	0.0
$F = F_1 + F_2 + F_3 + F_4 + F_5$			

- 相似度阈值：2.0

$$is\_similar = \begin{cases} True, \ if \ F >= threshold \\ False, if \ F < threshold \end{cases}$$

2.2.1.3 其它处理

A. HTTP 流化处理

我们使用 pkt2flow 工具生成以 TCP 连接为单位的网络数据流，在后续步骤中以流而并非数据包为单位来进行数据处理，这在一定程度上减少了系统的处理消耗，有利于系统检测的实时性。

B. IP+UA 流聚类处理

以生成的 HTTP 流作为输入，通过流的源 IP 字段以及使用的 User-Agent 字段进行聚类形成组。由于对于软件来说，其所使用的 User-Agent 往往具有对其自身的辨识度，因此通过 IP+UA 聚类的方式，我们减小了检测的粒度，从而在检测结果中给出更精确的信息。

C. 空 User-Agent 情况处理

HTTP 请求中会存在 User-Agent 字段为空的情况，这会影响到 IP+UA 流聚类的处理。经过分析，这种情况大多数是一些后台软件在向服务端反馈某些数据信息，或者在请求一些 XML 配置文件时发出的 HTTP 请求，而这些后台软件大多数还会发出其他的包含有 User-Agent 字段的请求。因此，对于每一个 User-Agent 字段为空的 HTTP 请求，如果存在与之具有相同域名，但 User-Agent 不为空的请求，我们则将该空 UA 请求忽略；若不存在，我们则将其单独进行区分，在 IP+UA 聚类阶段仅根据 IP 进行聚类。

2.2.2 检测分析模块

2.2.2.1 深度学习模型

A. 输入数据

如前文所述，我们发现使用 HTTP 请求头部作为输入所训练的模型具有最好的分类效果。因此，我们将一次对话中所有的 HTTP 请求头部抽取，并截取前 1024 字节作为输入。我们认为，使用 HTTP 协议进行通信的恶意软件其主要特征存在于前期的通信过程，而后期的 HTTP 信息传输主要为信息窃取和恶意软件部署的过程，因此，我们只截取了前 1024 字节作为卷积神经网络的输入。

B. 卷积神经网络原理

卷积神经网络是一种深度前馈人工神经网络，与传统的前馈式神经网络不同的是，卷积神经网络中的神经元只与上一层部分神经元直接相连，这使得卷积神经网络在运行时具有更小的开销。

一个卷积神经网络主要包含一个输入层和一个输出层，以及若干隐藏层，隐藏层主要包括卷积层、ReLU 层、池化层和全连接层。其中，卷积层由一组科学系的过滤器组成，它们可接受范围较小，但延伸至输入的所有区域。ReLU 层用于增加整个网络的非线性特征，线性整流与其他方法相比速度更快，适用于大规模流量数据的处理。

卷积神经网络在图像处理方面表现良好，虽然我们的卷积神经网络输入并不是传统意义上的图像，但我们认为 HTTP 头部特征是具有此类特征并可以使用卷积神经网络进行提取。

我们所构建的卷积神经网络模型如图 6 所示，整个模型使用 Google 开发的开源工具 Tensorflow 进行搭建。

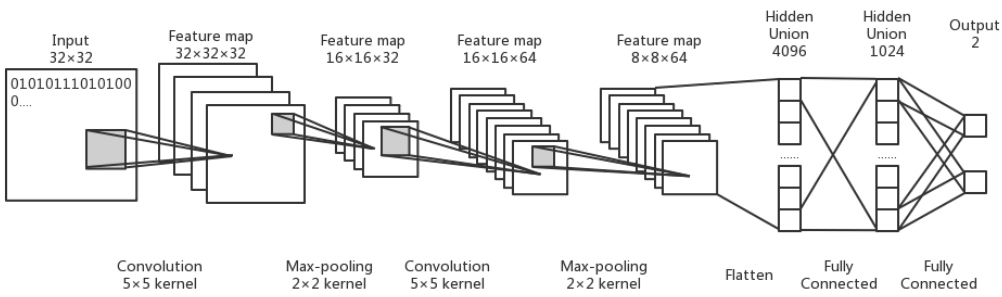

图 6 卷积神经网络模型

I. 输入图像通过一个窗口大小为 5×5，窗口数为 32 的卷积层进行初次特征提取，使用线性整流；

II. 通过 2×2 的池化层进行特征的汇集；

III. 通过窗口大小为 5×5，窗口数为 32 的卷积层再次进行特征提取，使用线性整流，并通过 2×2 的池化层进行特征的汇集；

IV. 通过两个全连接层获得输出结果。

2.2.2.2 迁移学习

在训练得到基本的检测模型后，在实际环境中正式部署运行该系统之前，由于该检测模型的检测特性取决于训练时使用的 ground truth 数据集，而此部分数据集是在别的环境中采集并进行标记得到的，对于不同特性的 http 流量的涵盖具有局限性。因此当其在应用到新环境中时，检测的效果可能不佳，特别是可能出现误报率较高的情况。

而深度学习模型检测与传统的规则匹配检测相比，其优势在于深度学习能够在工作当中对数据进行不断获取，并对这些数据当中的知识进行归纳、综合以及应用，使自己的性能得到不断提升，而并非单单只是对已有知识的直接应用。对应这一点，我们在系统中引入了迁移学习的机制。

在深度学习中，迁移学习的概念简单而言，是指将在一个问题上训练好的模型通过简单的调整使其适用于一个新的场景。在初期工作时，利用已有基础模型对实际环境中的流量进行检测，在检测出恶意对象后，通过引入前期人工指导干预——

使用某些方式对模型的检测结果进行判断处理，并利用处理后的结果对检测模型进行训练调整。在该过程反复进行一段时间后，检测模型对真实的网络流量环境进行了适应，增强了对当前环境中流量特性的识别，从而能够在后期检测工作中表现出更好效果。迁移学习的示意图如图 7：

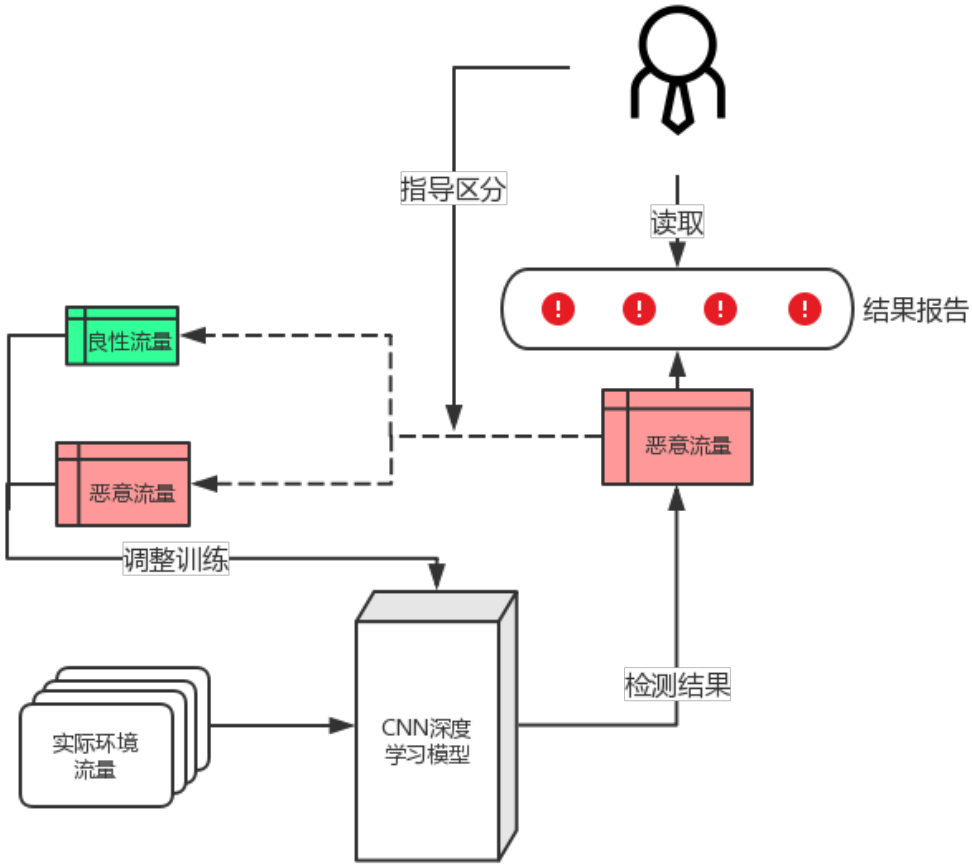

图 7 迁移学习过程

2.2.2.3 反馈学习

在一个局域网内，其软件环境会随着时间推移而发生缓慢变化，这会造成网关的流量特征也处于不断变化当中，同时随着恶意软件发展变化，其流量特征也在发生改变。当我们的系统部署到实际环境时，即使进行了一定时间的迁移学习，在接下来的一段时间内能够保持较好的效果，但随着时间的推移，其检测能力也难免下降，这就意味着需要对模型不断进行调整。但继续使用迁移学习的方法会不停对模型进行训练，这在系统的长期工作中可行性并不高。

针对这一问题，我们提出了反馈学习的方式。该方式下，系统在处理完一个时

间片的流量后，会自主使用在线检测工具 VirusTotal 对检测结果中的恶意流量进行验证，但系统不会随时根据验证的检测结果对模型进行训练，而是并将验证后的数据结果加入到样本库当中。通过设定误报率阈值，当计算出误报率超过了该阈值时，再使用样本数据对模型进行训练调整，同时更新指纹库。以这种方式，系统在工作过程中避免了不停训练模型，同时能持续学习，不断提高检测精度，降低误报率，在不断变化的网络环境中保持有效性。反馈学习具体示意图如图 8。

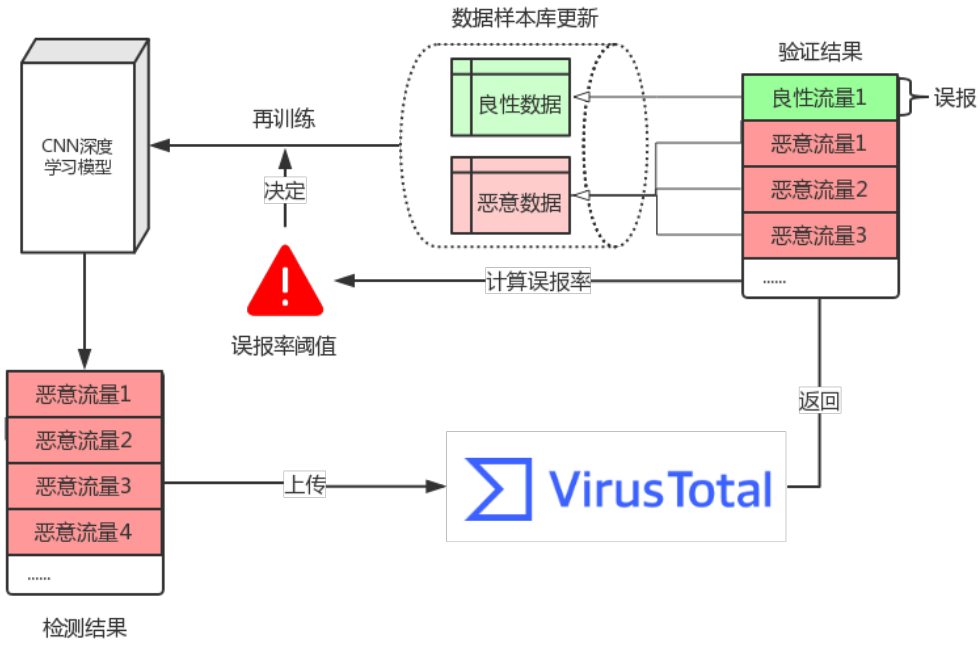

图 8 反馈学习过程

2.2.3 使用的工具

A. Alexa Top 10K 白名单

Alexa Top 10K 是由 Alexa 公司所提供的，对世界网址根据访问流量大小进行排名的前 10000 域名列表。Alexa 公司是一家专门发布网站世界排名的网站，其每天在网上搜集超过 1,000GB 的信息，给出多达几十亿的网址链接，并为其中的每一个网站进行了排名，是当前拥有 URL 数量最庞大，排名信息发布最详尽的网站。由于该列表具有公认性，且恶意域名的访问流量规模往往不大，因此我们可以认为，访问流量世界排名处于该列表当中的域名为正规安全域名，与该域名进行通信的 HTTP 流量为良性 HTTP 流量。据此可对捕获的 HTTP 流量进行初步过滤处理，减小系

统处理的数据量。

B. pkt2flow 流式转换工具

pkt2flow 是一个能够将 PCAP 文件拆分为流的简单工具。其根据源 IP 地址，目标 IP 地址，源端口号，目标端口号四个字段所组成的四元组把捕获的网络数据包拆分为 TCP 或者 UDP 流。每个拆分得到的流按照数据包读入的顺序，以四元组的四个字段加上时间戳作为文件名，依次保存为 PCAP 文件。使用该工具我们可将 HTTP 流量转化为流进行处理。

C. VirusTotal 流量标记网站

VirusTotal 是一个在线的恶意文件、恶意域名检测网站。针对单个杀毒软件无法做到 100%的检测率的问题，其集成了数十种当下知名安全厂商的扫描引擎，能够对用户上传的文件进行扫描检测并返回结果报告，同时也能够进行恶意域名的检测查询。VirusTotal 向用户提供了 API，通过该 API 能够对数据文件进行批量的检测和标记，进而得到训练所需要的数据集。同时使用该网站可对最终检测的恶意结果进行验证，为反馈学习提供数据。

D. jspcap 流量解析工具

jspcap 是由一个使用 Python 编写的 PCAP 文件解析工具，可将 PCAP 流量文件进行解析，并将其中的 TCP 流量载荷重组还原应用层协议报文。与常用的解析工具如 Scapy、dpkt 和 pyshark 等不同的是，jspcap 采取了流式读写的方案，通过密集 I/O 降低对内存的占用和消耗，能够在一定程度上显著降低本项目对部署目标的内存压力，使其具有更好的兼容性。此外，jspcap 还提供了前文所述 TCP 报文重组接口，为本项目提供了数据集生成的基本工具。

第三章 作品测试与分析

3.1 测试方案

我们的测试主要包括以下几个方面：

- 模型在原始训练数据集的性能评估；

  我们对比了对流量特征不同提取方式对性能的影响，并评估了模型在原始训练数据集的表现，将标记完成的数据集分为测试集和训练集，根据模型准确率的增长趋势判断应选择的迭代次数。

- 对比系统迁移学习前后准确率提升；

  通过对比系统在迁移学习前后，对同一个数据集的判断的效果，我们对迁移学习的效果进行评价，并进行总结以及提出了改进方案。

- 对比指纹过滤对系统性能的影响；

  我们对比了采用指纹过滤与不采用过滤时，系统在统一数据集上判断的准确率和误报率、运行的时间、运行中临时文件所占用的空间。

- 反馈算法对系统表现的提升；

  测试了系统在一段时间运行后性能下降的情况，并实验了反馈算法对性能的改善。

- 与同类产品对比，检测系统的相对准确率。

3.2 测试环境搭建

3.2.1 硬件环境

CPU：Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz

内存：16GB

3.2.2 软件环境

Ubuntu 16.04.2 LTS

Python 3.6.5; SCons 3.0.1

3.3 测试数据

本项目所使用的训练数据分为良性流量和恶意流量两部分。其中，恶意流量来自某安全厂商发布的一万个样本，我们使用脚本在虚拟机中运行恶意软件样本并在外部抓取恶意流量，将其中通过 HTTP 协议通信的流量保存为 PCAP 文件；良性流量为实际抓取的网络通信流量，使用 Wireshark 在组员个人电脑环境下抓取，并使用 VirusTotal 验证为良性流量。训练数据的具体信息如表 4 所示。

表 4 训练数据具体信息

良性流量数量	恶意流量数量
6500	5265

用于测试的实际数据集来自某大学网关抓取的流量数据，并使用 VirusTotal 进行标记，并将其按通信时间分为组，各组情况如表 5 所示。

表 5 测试数据具体信息

	良性流量数量	恶意流量数量	良性软件数量	恶意软件数量
实际数据集一	292	57	152	1
实际数据集二	1285	24	340	1

在实验中，我们将模型的识别率和误判率作为系统评价的首要指标。另外，我们也考虑了系统运行时间和临时文件占用空间大小等指标作为次要评价标准。

3.4 结果分析

3.4.1 模型在原始训练数据集的性能评估

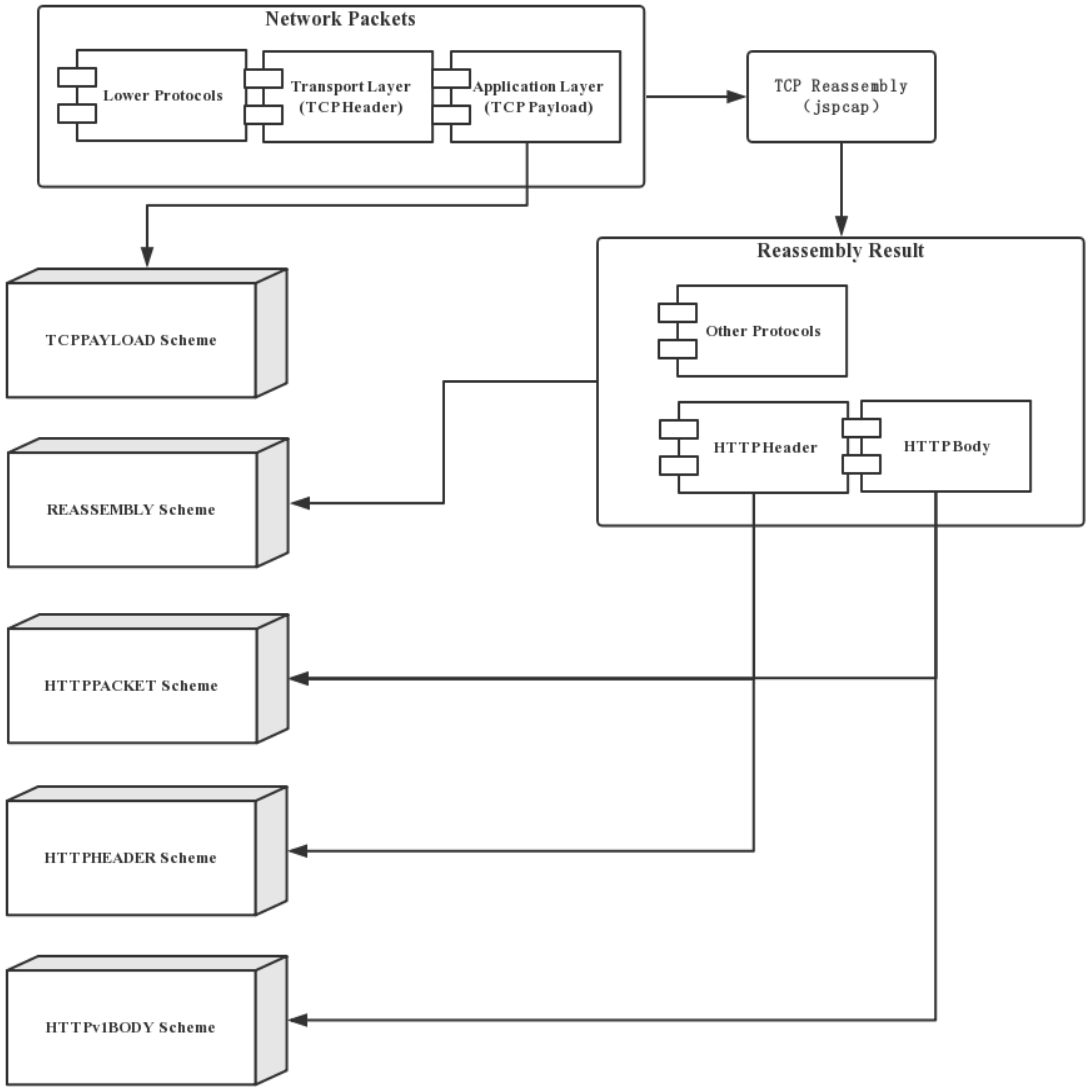

图 9 不同数据集生成方式

由于我们关注于 HTTP 流量，因此我们需要从原始的流量数据中提取出相应的流量生成数据集。而如上章所述，传输层 TCP 协议可能对应用层数据进行分片处理，需要将其重组之后才能获取完整数据[28]。于是，我们考虑了如图 9 所示的五种不同的数据集生成方式：

- tcppayload，提取 TCP 载荷，并直接相加

- reassembly，对 TCP 载荷进行重组，并将重组结果直接相加
- httppacket，对 TCP 载荷进行重组，取其中 HTTP 流量相加
- httpheader，对 TCP 载荷进行重组，取其中 HTTP 流量的报文头部相加
- httpv1body，对 TCP 载荷进行重组，取其中 HTTP 流量的消息正文相加

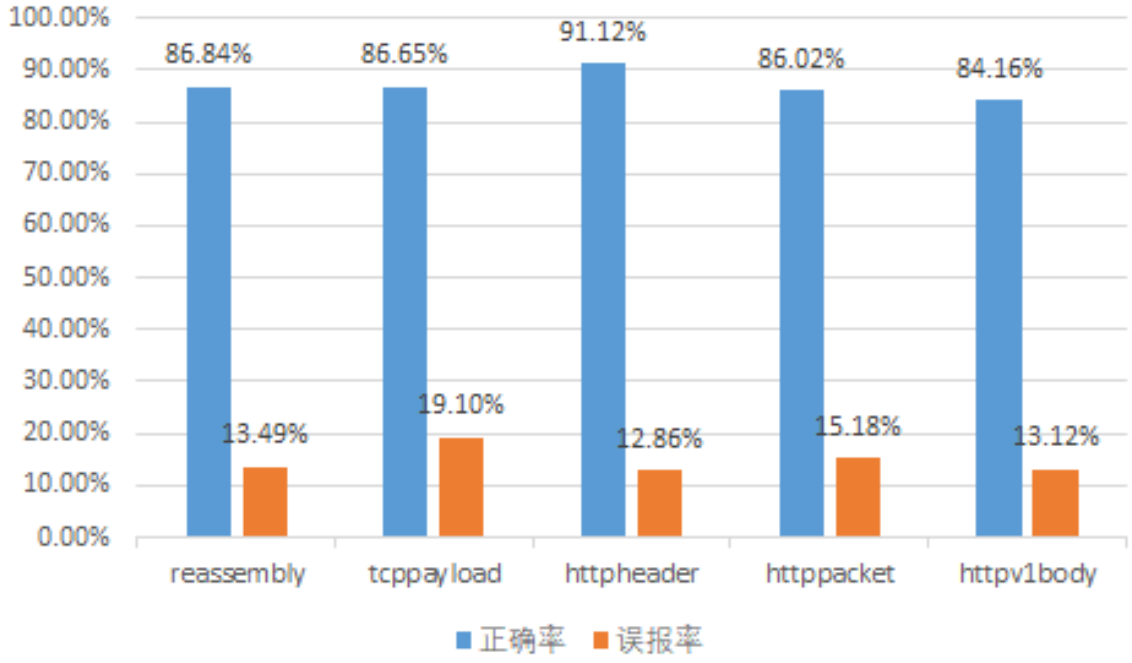

图 10 数据集生成方案测试结果

数据集生成方案测试结果如图 10 所示。通过实验我们发现，httpheader 类数据集的效果最好，恶意软件流量检测的正确率为 91.12%，显著高于其它几类；误报率为 12.86%，明显低于其它几类。因此，我们最终选用 httpheader 类数据集生成方案产生本项目的数据集。

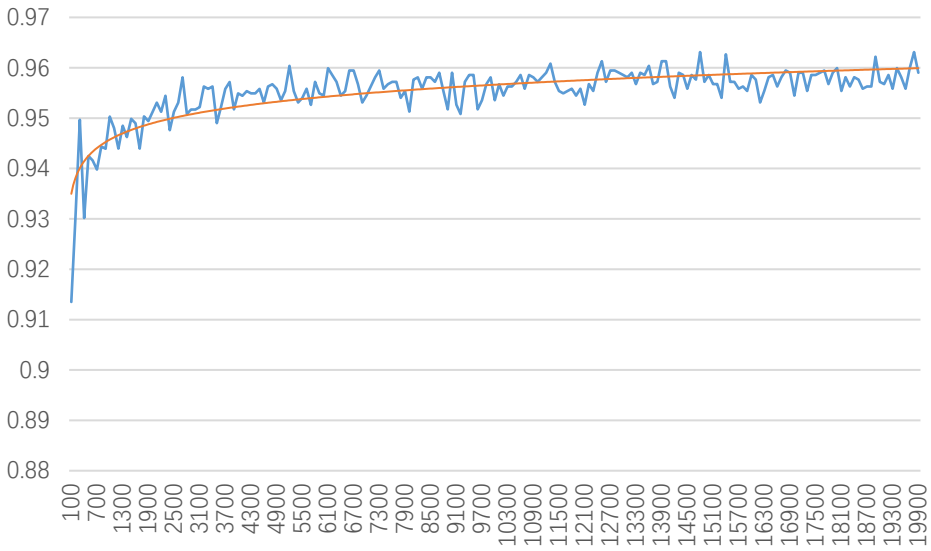

图 11 训练过程中精确度的提升

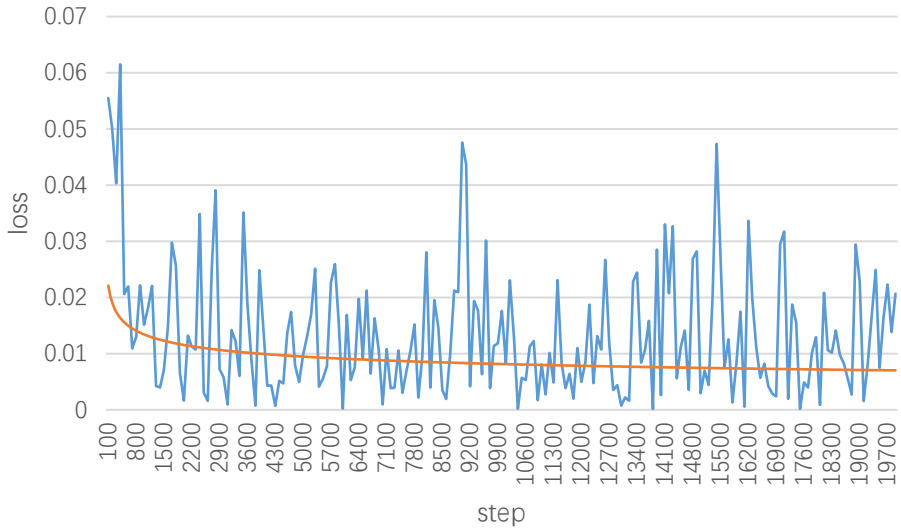

图 12 训练过程中损失的下降

在测试中，使用梯度下降法，每次抓取 100 个数据对模型进行训练。图 11 展示了训练过程中精确度的提升，图 12 展示了训练过程中损失的下降，其中横坐标代表步数，每 100 步记录一次数据，纵坐标代表模型在测试集上的准确率或损失。蓝色图像代表准确率或损失的实际值，橙色线反映了其变化趋势。

可以看出，在迭代至 18000 步左右时，准确度的增长不再明显，因此，我们选择 18000 步的迭代作为训练的终止点。最终，模型在测试集的对恶意流量的识别率率为 95.9%。

3.4.2 迁移学习实验结果

迁移学习主要影响指纹提取的效果。由于目前模型仍然具有10%左右的误检率，在整体看来，良性流量数量巨大，10%的误检率仍然会在验证时对网络造成一定压力，并且会降低系统运行的效率。而使用历史流量进行迁移学习，不仅仅会提高CNN网络对本环境的针对性，同时也会生成已知流量的指纹。

在接下来的工作中，对于输入的流量，系统将会首先对其进行指纹的识别，验证流量是否在之前出现过。系统仅会将之前没有出现过的流量输入CNN模型进行识别并验证。

对迁移学习效果的测试我们使用实际数据集一，该数据集大小为1GB，包含57条恶意流量和252条良性流量，使用迁移学习前后系统对恶意流量的识别效果如表6所示。

表6 迁移学习前后系统对恶意流量的识别效果对比

	识别率	**误判率**
未使用迁移学习	91.23%	9.25%
使用迁移学习	94.74%	6.16%

在使用迁移学习后，良性流量的误判率降低了33.41%，同时恶意流量的识别率略有提升，但效果并不明显。

最终，我们的系统在实际数据中获得了94.74%的识别率和6.16%的误判率，结果的ROC图像如图13所示，图像的AUC = 0.9923。

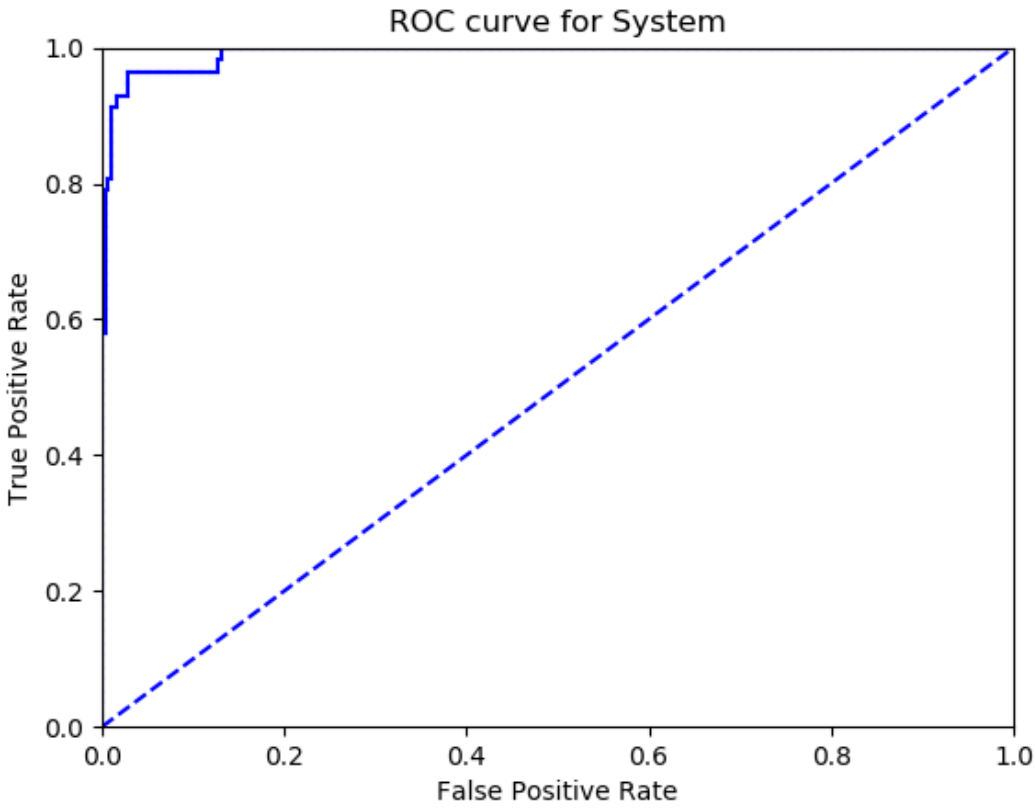

图 13 系统在实际数据上的 ROC 图

以上数据表明，使用迁移学习的系统可以有效减少对良性流量的误判，但同时指纹识别算法也有待优化，仍然有较大的提升空间。

3.4.3 指纹过滤对系统性能的影响

我们同样选择实际数据集一对系统性能进行测试，该数据集中总共包含 153 个软件，其中存在 1 个恶意软件，152 个良性软件。

测试结果如表 7 所示。

表 7 性能测试结果

不使用指纹过滤的系统性能			
正确检测数	1	**识别率**	100%
误报数	16	**误判率**	10.53%
运行时间	57min 52sec	**临时文件大小**	852KB
使用指纹过滤的系统性能			

正确检测数	1	识别率	100%
误报数	9	误判率	5.92%
运行时间	41min 35sec	临时文件大小	652KB

在实验所用数据集上，指纹过滤可以使误判率降低约 43.78%，运行时间降低约 28.14%，临时文件大小减少约 23.47%。实际性能提升会由于系统硬件环境、流量数据组成等情况有所变化。

3.4.4 反馈算法对系统表现的提升效果

我们首先使用实际数据集一的流量数据对系统进行初次部署的迁移学习。进行迁移后，系统的识别率为 94.74%，误判率为 6.16%。接下来，我们选取了在实际数据集一 24 小时后抓取的实际数据集二对我们的系统进行了测试，经过 VirusTotal 的标记，该测试数据详细信息如表 8 所示。

表 8 测试数据基本信息

软件总数	恶意软件数		良性软件数
340	1		339
恶意软件 User-Agent	目的 IP	源 IP	恶意软件对话流数
Mozilla/5.0	123.59.68.172	172.16.201.137	67
Unknown	123.59.68.172	172.16.201.137	2

可以看出在实际数据中，IP 地址为 172.16.201.137 的主机上存在一个恶意软件，它与 IP 地址为 123.59.68.172 的服务器进行了通信，通过 VirusTotal 验证发现该地址为恶意服务器。软件通信过程中，大部分通信使用了伪造的 User-Agent 将自身伪装成浏览器，小部分使用了空 User-Agent 字段。对这一组实际数据的检测结果如表 9 所示。

表 9 实际数据集二检测结果

反馈训练前检测结果			
正确检测数	1	识别率	100%

误报数	32	误判率	9.41%
反馈训练后检测结果			
正确检测数	1	识别率	100%
误报数	18	误判率	5.29%

可以看出，随着系统的运行，由于局域网中软件环境有所变化，系统的误判率会上升，而采用迁移学习的策略，可以使系统的误判率稳定在一个较低的水平。

3.4.5 与现有产品的对比

对于软件的测试，我们随机挑选了 15 个恶意软件样本，分别使用 Maltrail、Network Total 与我们的系统进行比较。

Maltrail 利用公开的黑名单来检测恶意和可疑的通信流量。对于 Maltrail 的测试，我们使用程序模拟软件与服务器进行通信。

Network Total 是一个在线 PCAP 文件分析工具，使用预设的规则集对 PCAP 文件中记录的恶意行为进行检测。对于 Network Total 的测试，我们将对话流上传进行检测。

实验结果如表 10（表中略去 UA 详细信息）和图 14 所示。

表 10 系统与现有产品检测效果对比

恶意软件流量的 User-Agent 字段	恶意软件流量的 Host 字段	本系统检测结果	Maltrail 检测结果	Network Total 检测结果
Mozilla/5.0	123.59.68.172	检出	未检出	未检出
Mozilla/5.0	palmerlevellll931.ru	检出	检出	未检出
Unknown UA	gocgleclicks.com	检出	检出	未检出
Mozilla/4.0	report.3w793gm9g17aaaa9ku.com	未检出	未检出	未检出
NSISDL/1.2	download.phpnuke.org	检出	检出	未检出
IE9	download.u-tab.co.kr	检出	检出	未检出

Mozilla/4.0	totdisseny.net	检出	未检出	未检出
Mozilla/4.0	install.blamcity.com	检出	检出	检出
Mozilla/4.0	shake.youknowtheanswers.com	检出	未检出	未检出
test	up1.give2sms.com	检出	未检出	未检出
Mozilla/4.0	cdnus.webfilescdn.com	检出	检出	检出
Mozilla/4.0	www.ddho.com.br	未检出	未检出	未检出
Mozilla/4.0	116.255.235.9	检出	未检出	未检出
Mozilla/4.0	nuolaidos.lsas.lt	检出	未检出	未检出
NSISDL/1.2	download.ircfast.com	检出	检出	未检出

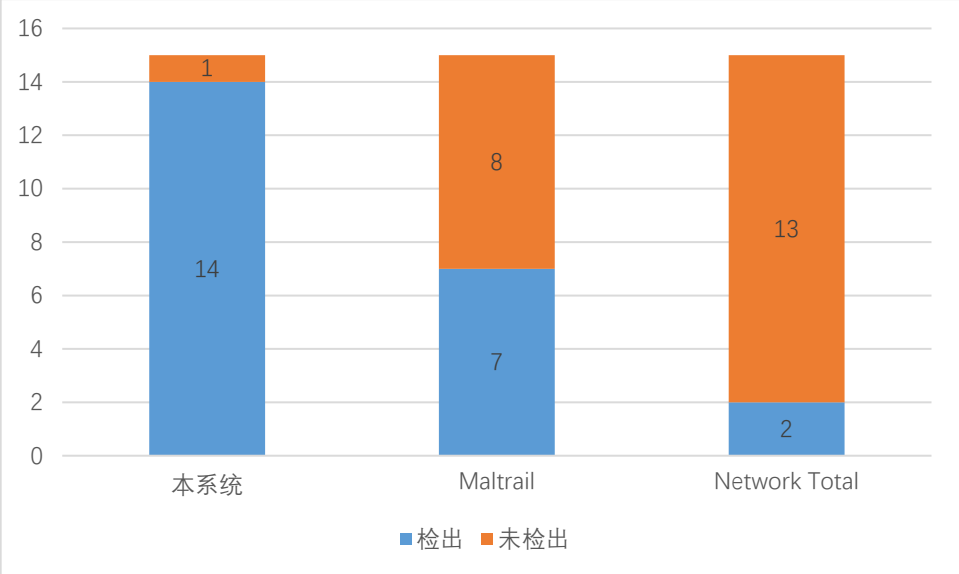

图 14 检测效果对比

由于 Maltrail 使用第三方发布的黑名单对通信流量进行检测，对恶意软件的检测也利用已知的 Command & Control 服务器（C&C 服务器）名单，因此其无法检测未被发现的 C&C 服务器通信的恶意软件或不通过 C&C 服务器工作的恶意软件。由于 Network Total 所使用的 EmergingThreats PRO Rules 中恶意软件规则尚未完善，对恶意软件的 HTTP 通信流量检测效果较差。

通过与已有产品的对比，可以看出我们的系统具有更高的检出率和适应性，相比于黑名单检测能够适应当前网络环境。

3.4.6 总结

我们通过多项实验，对系统输入预处理、模型训练参数等变量进行了测试和对比，获得了较好的实验结果。最终，我们的系统在某高校网关流量上的测试结果获得了 100%的识别率和 5.29%的误判率。

通过与他人工作的对比，我们的系统具有较好的检测率，实验结果较为理想。但误判率仍有较大提升空间，考虑可以改进特征提取算法以降低误判率。

第四章 创新性说明

4.1 精细粒度帮助定位溯源

目前市面上通过流量检测来识别网络中恶意程序的系统，在发现恶意软件的存在时，往往无法精确定位到网络系统中具体哪个设备上的哪个软件存在问题。而我们通过对 HTTP 流按照 IP+UA 的方式进行聚类处理，使得检测粒度更精细，能够在最终的检测结果中以源 ip，目的 ip，User-Agent，以及端口号的四元组的形式给出信息。

在获取到检测结果后，根据相应的源 ip 及端口信息，维护人员可利用如端口扫描等手段，来精确地定位内部网络系统中具体某台设备上所存在的某恶意软件。同时，对于软件来说，User-Agent 往往具有较高辨识度，通过对检测出的 User-Agent 信息进行分析处理，还可以对机器上的恶意软件的变化发展进行细粒度的追踪。

4.2 深度学习优化特征提取

在对网络流量进行分析检测时，提取的特征的好坏直接影响到最终的检测结果。在传统的方法当中，通过人工的方式对流量的特征进行分析和提取，成本花费较高，同时在某些数据中下人工分析提取特征的难度较大，提取的特征模式固定，难以适应更大范围以及更多不同特性的输入，需要人为针对特定输人对算法进行修改

而应用卷积神经网络进行流量分析时，特征提取完全由计算机自动进行，其相比于人工提取，更充分地利用了深度学习能够挖掘深层特征关联的优点，能够对于流量特性的抓捕更加全面。同时，通过应用深度学习，系统在工作时能方便地对检测模型进行自动更新调整，无需人为对使用的特征进行分析修改，很大程度优化了工作过程。

4.3 迁移学习、反馈训练提高适应性

市面上的异常流量检测的软件大多采取出厂即用的策略，即仅仅只是对其所包含的知识直接进行应用，而忽略了工作过程中对环境数据的归纳、综合及再应用。这些软件模型所包含的规则或者其训练时所使用的数据集并非针对某一特定的实际应用场景而设计，因此无法涵盖各种场景下流量的特性。另外由于网络流量环境处于不断变化之中，恶意软件也在不断发展变化，因此检测难以保持长期有效。

我们的系统在使用 ground truth 数据集训练好基本检测模型后，在实际工作的

初期，通过迁移学习引入主动干预指导——通过某种方式对模型检测出来的结果进行评判，并将评判后的正确结果作为数据重新对模型进行调整训练。如此进行一段周期后，模型对实际环境有了很好的适应能力。同时在后期工作时，系统也会对检测结果进行自行验证，并将正确数据存入数据集。通过设定阈值，当误报率超过该阈值时，系统再自动使用更新的数据集进行模型重训练。该反馈学习方式既避免了迁移学习中需要对模型不停训练，也保证了系统持续学习。

通过初期的迁移学习与后期工作过程中的反馈学习，系统健壮性得到提高，从而可适应不断变化的动态流量环境，能够应对当前恶意软件不断变化发展的趋势。

4.4 浏览器清洗、指纹判别提高效率

在 HTTP 恶意流量检测上，由于在规模稍大的网络系统中，实时通过网关的流量的数据量很大，要对如此多的流量进行高精度的检测，巨大的工作量使实时性成为最主要的瓶颈。对此，我们采用了浏览器流量清洗的方法：先用 Alexa 白名单进行初步过滤，然后根据 HTTP 流量中有大量来自浏览器的部分的特点，通过 WebGraphic 请求图算法对浏览器流量进行清洗，后续仅对后台软件流量进行处理，从而很大程度地减少了数据量，提高了系统效率。

另外对于剩下的来自后台软件的流量，由于大多数流量相互之间存在相似性，且出现过的类型的流量往往会在将来重复出现，因此我们通过提取流量指纹，直接对已知流量进行判别，减少系统的工作负担，进一步提高了检测的效率。

4.5 网关检测扩大范围

市面上的恶意软件检测软件往往是针对个人设备进行检测，而对于一个组织机构而言，在 IT 系统内部所有设备上进行部署的可行性不高，因此对于整个系统内部的恶意软件检测难以进行。而我们的系统部署于网关上，通过处理经过的 HTTP 流量来识别恶意软件，因此对于内部所有活动设备均能够进行有效检测。

第五章 总结

5.1 优缺点总结

经过查阅多方资料与数次尝试，我们最终选择根据本文中所阐述的算法来实现恶意软件检测系统。在检测范围上，通过对网关 HTTP 流量的分析，我们能够对整个网络系统进行检测；在实时性上，我们在数据预处理的过程中通过浏览器清洗、流量指纹判别，很大程度地减小了需进行检测数据量，提高了系统的效率；在精度上，事实证明通过卷积神经网络对 HTTP 报文头部进行识别这一方案在实践中可以取得令人满意的准确率。同时我们使用迁移学习保证系统初次部署到实际环境中的效果，并利用反馈学习的机制使得系统在工作中持续自主学习，不断适应流量环境与恶意软件的变化，保持长期有效。

由于各方面的限制，本系统目前仍存在一定不足，其中主要有如下几个方面：

I. 原始训练阶段尚可进一步完善，使用更加广谱的数据集训练，提高其识别精确度和准确率；

II. 在数据集的生成阶段，还可通过多进程加速和使用 C 等效率更高的语言实现等方式提高效率，同时还可使用一些算法对数据进行进一步特征提取，以期更为具有代表性；

III. 目前缺少友好的交互界面，暂为后台驻留服务的形式进行工作，不便用户进行调试和查询；

IV. 虽然目前超过 90%的软件在使用 HTTP 协议通信，但是这也意味着仍有部分恶意软件并不使用 HTTP，对于这些恶意软件，我们的系统无法进行检测。同时，即使系统中存在使用 HTTP 的恶意软件，当其不产生 HTTP 流量时，也无法对其进行检测。

5.2 未来展望

5.2.1 使用特征广谱数据集进行基本模型优化

在现阶段的系统开发中，我们采集了足量的数据集，并使用这些数据集进行了模型训练。尽管测试结果较为理想，但由于数据集并不具备广谱特征，因此现阶段所得到的基本模型识别精确度和准确率仍有提高区间。期望后续可以投入更多的设备和精力，抓取更多的数据集，对基本模型进行进一步的优化，以便于进一步的提高准确率和精确度。

5.2.2 加速流量解析

在本系统中，我们提出了 jspcap 流量解析工具，对 PCAP 文件进行解析，并将其中的 TCP 流量载荷重组还原应用层协议报文。与传统的解析工具相比 jspcap 采取了流式读写的方案，通过密集 I/O 降低对内存的占用和消耗，能够在一定程度上显著降低本项目对部署目标带来的内存压力。虽然实际运行中，流量解析速度已经取得比较不错的效果，但我们认为仍能够进一步提升解析速度，如通过多进程并发处理，或使用 C 语言转写底层代码，降低流量解析的延迟，对边界网关的流量进行更高精度、更低延迟的解析，从而进一步提高系统检测的实时性。

5.2.3 多种模型组合

虽然本系统在实际数据上的表现较好，但其检测率和误报率仍然有一定的优化空间。本系统目前仅使用卷积神经网络来训练检测模型，我们认为在该方面还可存在改进。可以考虑通过加入多种机器学习模型，结合多模型的检测结果进行综合判断的方式，使得对恶意流量的预测取得更好的效果。

5.2.4 系统的用户友好

目前，整个系统架构已基本完成，但其目前的部署方式为后台驻留服务，在使用中仍然需要通过命令行接口，我们认为系统需要提高模块化，并提供更加友好的用户接口或图形界面。同时，当前系统在监测到恶意流量后，将其记

录在后台日志中。这对用户而言并不十分便捷。因此，我们还将实现一个静态网页，作为监测记录和状态的展示平台。

参考文献

[1] Mitchell, Tom M. "Machine Learning (McGraw-Hill International Editions Computer Science Series)." (1997).

[2] Christodorescu M, Jha S. Static analysis of executables to detect malicious patterns[C]. *USENIX Security Symposium*, 2003: 12-12.

[3] Saxe J, Berlin K. Deep neural network based malware detection using two dimensional binary program features[C]. *International Conference on Malicious and Unwanted Software*, 2015: 11-20.

[4] Yuan Z, Lu Y, Wang Z, et al. Droid-Sec: deep learning in android malware detection[C]. *ACM Special Interest Group on Data Communication*, 2015, 44(4): 371-372.

[5] Tobiyama S, Yamaguchi Y, Shimada H, et al. Malware Detection with Deep Neural Network Using Process Behavior[C]. *Computer Software and Applications Conference*, 2016: 577-582.

[6] Lakhina, Anukool, Mark Crovella, and Christophe Diot. "Mining anomalies using traffic feature distributions." *ACM SIGCOMM Computer Communication Review*. Vol. 35. No. 4. ACM, 2005.

[7] Karagiannis, Thomas, Andre Broido, and Michalis Faloutsos. "Transport layer identification of P2P traffic." *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004.

[8] Karagiannis, Thomas, Konstantina Papagiannaki, and Michalis Faloutsos. "BLINC: multilevel traffic classification in the dark." *ACM SIGCOMM Computer Communication Review*. Vol. 35. No. 4. ACM, 2005.

[9] Roughan, Matthew, et al. "Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification." *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004.

[10] McGregor, Anthony, et al. "Flow clustering using machine learning techniques." *Passive and Active Network Measurement* (2004): 205-214.

[11] Gu, Yu, Andrew McCallum, and Don Towsley. "Detecting anomalies in network traffic using maximum entropy estimation." *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005.

[12] Atiya, Amir F., Mohamed A. Aly, and Alexander G. Parlos. "Sparse basis selection: New results and application to adaptive prediction of video source traffic." *IEEE Transactions on Neural Networks* 16.5 (2005): 1136-1146.

[13] Auld, Tom, Andrew W. Moore, and Stephen F. Gull. "Bayesian neural networks for internet traffic classification." *IEEE Transactions on neural networks* 18.1 (2007): 223-239.

[14] Li, Xiang, Feng Qi, Dan Xu, and Xue-song Qiu. "An internet traffic classification method based on semi-supervised support vector machine.". *IEEE International Conference on Communications (ICC)*. IEEE, 2011.

[15] Paxson, Vern. "Bro: a system for detecting network intruders in real-time." *Computer networks* 31.23 (1999): 2435-2463.

[16] Roesch, Martin. "Snort: Lightweight intrusion detection for networks." *Lisa*. Vol.

99. No. 1. 1999.
[17] Nizar Kheir. 2012. Analyzing HTTP User Agent Anomalies for Malware Detection. In *Data Privacy Management and Autonomous Spontaneous Security*, 7th International Workshop, DPM 2012, Pisa, Italy, September 13-14, 2012. Springer, 187-200.
[18] Terry Nelms, Roberto Perdisci, And Mustanque Ahamad. 2013. ExecScent: Mining for New C&C Domains in Live Networks with Adaptive Control Protocol Templates. In *Proceeding of the 22th USENIX Security Symposium*, Washington DC, USA, August 14-16, 2013.USENIX Association, 589-604.
[19] Christian Rossow, Christian J Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten Van Steen, Felix C Freiling, and Norbert Pohlmann. 2011. Sandnet: Network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. ACM, 78–88.
[20] Apostolis Zarras, Antonis Papadogiannakis, Robert Gawlik, and Thorsten Holz. 2014. Automated Generation of Models for Fast and Precise Detection of HTTP-based Malware. In *2014 Twelfth Annual International Conference on Privacy, Secu-rity and Trust*, Toronto, ON, Canada, July 23-24, 2014. 249–256.
[21] Christopher Neasbitt, Roberto Perdisci, Kang Li, and Terry Nelms. 2014. Click-Miner: Towards Forensic Reconstruction of User-Browser Interactions from Network Traces. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, Scottsdale, AZ, USA, November 3-7, 2014. 1244–1255.
[22] Riccardo Bortolameotti, Thijs van Ede, Marco Caselli, Maarten H. Everts,Pieter Hartel, Rick Hofstede, Willem Jonker, and Andreas Peter. 2017. DE-CANTeR: DEteCtion of Anomalous outbouNd HTTP TRaffic by Passive Application Fingerprinting. In *Proceedings of ACSAC 2017*. ACM, New York, NY, USA, 373-386.
[23] Lamprakis, Pavlos, et al. "Unsupervised Detection of APT C&C Channels using Web Request Graphs." *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, Cham, 2017.
[24] Pascanu R, Stokes J W, Sanossian H, et al. Malware classification with recurrent networks[C]. *International Conference on Acoustics, Speech, and Signal Processing*, 2015: 1916-1920.
[25] Jon Postel (ed.). "Transmission Control Protocol". *RFC 793*. IETF, 1981.
[26] Jon Postel (ed.). "Internet Protocol". *RFC 791*. IETF, 1981.
[27] David D. Clark. "IP Datagram Reassembly Algorithms". *RFC 815*. IETF, 1982.
[28] Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee, Tim. "Hypertext Transfer Protocol – HTTP/1.1". *RFC 2616*. IETF, 1999.