上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

BACHELOR'S THESIS

论文题目：基于 Bro 的 APT 检测系统设计与实现

学生姓名：肖佳伟

学生学号：515030910023

专　　业：信息安全

指导教师：邹福泰

学院(系)：电子信息与电气工程学院

基于 Bro 的 APT 检测系统设计与实现

摘要

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，高级持久威胁（Advanced Persistent Threat, APT）逐渐成为攻击事件的主要来源。基于对网络流量的直接观察和分析，对网络应用准确及时的识别与分类正变得日益重要。

本项目设计并实现了基于 Bro 入侵检测系统的 APT 检测系统 BroAPT 系统。这一系统将通过对网络流量的直接综合分析，对 APT 攻击进行检测。系统具有较高的性能和可拓展性，以及对高速流量进行实时分析和生成日志的功能，可对流量中传输的文件进行重组和提取，并通过针对性的恶意文件检测，以及对日志的分析，实现了对流量中 APT 攻击的检测。

本项目的工作对于 APT 检测的研究虽然较为前置，探究的目标比较宽泛，但是挖掘了 Bro 入侵检测系统对 APT 检测的效用，设计并实现了一个具备极强可拓展性且可用于高速流量处理的 APT 检测系统。这一系统在实际应用中已有一定前景，且可作为后续研究的基础的方向。

关键字： Bro 入侵检测系统，APT 检测，文件提取，安全检测

BROAPT: A SYSTEM FOR DETECTING APT ATTACKS IN REAL-TIME

ABSTRACT

Cybersecurity has long been a significant subject under discussion. With rapid evolution of new cyber attack methods, the threat of Internet is becoming more and more intense. Advanced persistent threat (APT) has become a main source of cybersecurity events. It is now even more important to identify and classify network traffic by direct analysis on the traffic itself in an accurate and timely manner.

We hereby describe BroAPT system, an APT detection system based on Bro IDS. The system monitors APT based on comprehensive analysis of the network traffic. It is granted with high performance and extensibility. It can reassemble then extract files transmitted in the traffic, analyse and generate log files in real-time; it can also classify extracted files through targeted malicious file detection configuration; and it detects APT attacks based on analysis of the log files generated by the system itself.

Although our research on APT detection is quite preceding, the BroAPT system utilised Bro IDS and works as an APT detection system which is compatible with high-speed network traffic. The system has been proved in practical scenarios and is the basis of follow-up researches on APT detection.

Keywords: Bro IDS, APT detection, file extraction, traffic reassembly, security detection

目录

第一章　绪论

互联网安全问题一直是全球范围内的主要议题。随着网络攻击方式和理论的快速发展迭代，互联网的安全威胁正在日益激化。基于对网络流量的直接观察和分析，对网络应用准确及时的识别与分类正变得日益重要。识别与分类，通常是指估算网络规划的需求趋势分析，自适应网络的服务质量（Quality of Service，QoS），以及基于自适应防火墙对监测到禁止的应用或攻击时的合法拦截或动态访问控制[1]。

随着网络攻击技术的发展，高级持久威胁（Advanced Persistent Threat, APT）逐渐成为攻击事件的主要来源。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识别并标记了了 8 千万个恶意软件，根据卡巴斯基发布的统计数据（图 1-1）显示，这一数量仍然在迅速增加。

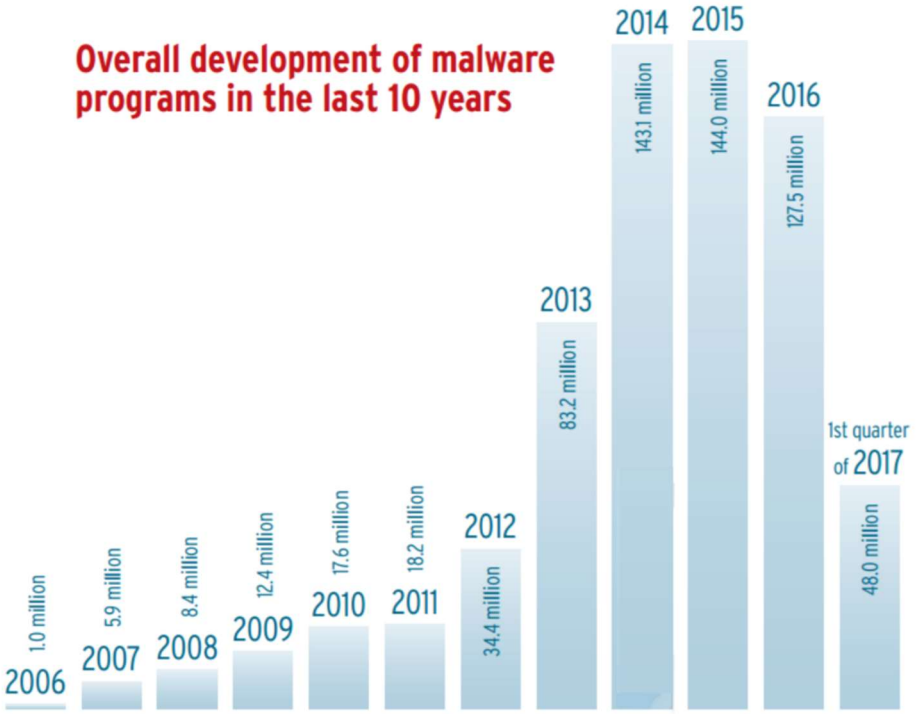

图 1-1 过去十年中恶意软件变化统计

本项目设计并实现了基于 Bro 入侵检测系统的 APT 监测系统 BroAPT 系统。这一系统将通过对网络流量的直接综合分析，对 APT 攻击进行检测。系统具有较高的性能和可拓展性，以及对高速流量进行实时分析和生成日志的功能，可对流量中传输的文件进行重组和提取，并通过针对性的恶意文件检测，以及对日志的分析，实现了对流量中 APT 攻击的检测。

具体来说，本项目的研究意义可以分为以下几点：

(1) 实现了对高速流量处理的 APT 检测系统

BroAPT 系统充分利用了 Bro 入侵检测系统的高性能优势，采用多进程的实现方式对实时抓取的网络流量 PCAP 文件进行处理，并重组和提取流量中传输的文件。由于 Bro 入侵检测系统对高速流量的适应性，因此 BroAPT 系统在设计中继承了 Bro 入侵监测系统的高效性与可拓展性，并通过 Python 对其进行了进一步地封装和优化。

(2) 实现了具备极强可拓展性的 APT 检测系统

BroAPT 系统在运行时，可通过自定义的 Bro 脚本和分析函数，对系统的原有功能进行拓展，生成定制的日志文件，产生特定的报警行为等；在对提取得到的文件进行检测时，BroAPT 系统可通过基于文件 MIME 类型的 API 配置文件，对不同类型的文件使用不同的检测方案，从而提高恶意文件检测的准确度。最后，通过系统所产生的日志文件和提取并检测的恶意文件，提出了基于 BroAPT 系统的 APT 检测方案。

本项目的工作对于 APT 检测的研究虽然较为前置，探究的目标比较宽泛，但是挖掘了 Bro 入侵检测系统对 APT 检测的效用，设计并实现了一个具备极强可拓展性且可用于高速流量处理的 APT 检测系统。这一系统在实际应用中已有一定前景，且可作为后续研究的基础的方向。

本文将对 BroAPT 系统的设计与实现进行详细介绍，其中共分为八个章节。各章节的主要内容如下：

(1) 绪论

绪论部分对本项目的研究背景进行了简要描述，说明了 APT 检测的主要方向和应用前景，以及当前研究中所遭遇的瓶颈。随后，对本项目的研究意义进行了简要阐述，说明了 BroAPT 系统的设计初衷和实现目标，即一个基于 Bro 入侵检测系统，具有对高速流量进行实时处理的高性能及可拓展性的 APT 检测系统。

(2) 国内外研究现状与相关背景介绍

由于本项目所实现的 BroAPT 系统核心主要基于 Bro 入侵检测系统，因此在本章节中，着重对 APT 研究现状和入侵检测系统进行了介绍。随后，对 Bro 入侵检测系统的架构与实现进行了简要描述，并介绍了 Bro 脚本语言及其日志系统。此外，本章还对相关背景进行了简要介绍，其中主要包含用于文件提取的 TCP 流量重组算法，用于针对性恶意文件检测的 MIME 类型分类，以及用于系统配置文件的 YAML 语言。

(3) 系统设计综述

本章着重介绍了 BroAPT 系统的架构设计，其核心模块由 BroAPT-Core 提取模块与 BroAPT-App 检测模块组成，核心模块在 Docker 容器中运行；以及 BroAPT-Daemon 服务进程，其作为 REST API 服务器在宿主机上运行。随后，介绍了 BroAPT 系统的运行逻辑与流程。本章还对 BroAPT 系统的具体实现进行了详细介绍，并对系统的可拓展性进行了简要描述，介绍了 BroAPT-Daemon 服务进程的实现背景。

(4) 基于 Bro 的文件提取与日志分析

本章对 BroAPT 系统的核心模块之一 BroAPT-Core 提取模块做了详细介绍，并对该模块的实现进行了说明，大体上可分为基于 Bro 脚本的事件处理与基于 Python 接口的日志分

析。同时，本章还对 BroAPT-Core 提取模块的拓展性设计做了详细介绍：模块可通过环境变量对系统的运行参数进行调整，可通过引入加载用户编写的 bro 脚本实现对 Bro 系统运行时功能的拓展，还可通过注册日志文件的分析函数，实现对 Bro 日志的自定义处理。

(5) 基于 API 的针对性文件检测

本章对 BroAPT 系统的核心模块之一 BroAPT-App 检测模块做了详细介绍，并对该模块的实现进行了说明，介绍了基于 BroAPT-Daemon 服务进程的客户端-服务器远程检测模式。同时，本章对 BroAPT-App 检测模块的拓展型设计做了详细介绍，并对系统所使用的 API 配置文件进行了详细介绍，对其实现特点和技术难点进行了描述。

(6) 系统评测

本章简要介绍了对 BroAPT 系统的性能测试和拓展功能测试。在对实验环境和软硬件环境进行了简要介绍后，对比了 Bro 入侵检测系统与 `tcpflow` 工具的性能和效果。此外，本章测试了第四章中介绍的 Bro 脚本对系统运行的负载情况，以及日志分析函数的功能，并对第五章中介绍的检测方案进行了测试。

(7) 相关工作与尝试

本章对项目实现过程中做出的尝试和相关工作进行了简要介绍。对 TCP 流量重组与文件提取的实现方案进行了深入讨论，并描述了 BroAPT 系统实现方案的选择原因。此外，本章讨论了关于系统架构的设计，并对两种设计进行了分析和解释。

(8) 总结与展望

在本章中，对本项目的 BroAPT 系统设计与实现进行了总结，讨论了其实现特点和应用前景，并简要阐述了当前系统中存在的不足与缺憾，最后对未来的工作方向进行了简要介绍。

第二章　国内外研究现状与相关背景介绍

2.1　APT 检测研究现状

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，这种基于端口及有效载荷等监测异常流量的方式效率越来越低。对于 APT 攻击而言，其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。这些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，这使得这种基于流量端口及有效载荷等监测异常流量的方式对其无效[2] 。

尽管如此，基于流量的分析，仍是 APT 检测的主要方式。根据分析方案的不同，针对 APT 攻击的检测主要可分为两种类型：

(1) 基于机器学习算法

许多学者基于机器学习（machine learning）算法与深度学习（deep learning）技术，对网络流量分类与检测进行了深入研究[3] 。如 Thomas 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[4]，或端口和主机之间的流量模式[5]；Matthew 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[6]；Yu 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[7]；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[8]；还有一些研究人员采用贝叶斯神经网络（Bayesian neural networks）[9]或支持向量机（Support Vector Machine，SVM）[10]等机器学习算法进行网络流量分类的研究。

(2) 基于对流量的综合分析

相较于通过机器学习算法的直接检测，对于网络流量的直接综合分析也是 APT 攻击检测的一大方向，例如 Bilge 等人通过大量流量分析检测僵尸网络[11]；Pavlos Lamprakis 等人所提出的网络请求图（web request graphs）算法[12]；Jasek R.等人基于蜜罐对 APT 攻击进行检测[13]；而 FireEye 公司的 Haq T.等人基于对流量对象的特征提取和匹配，以及概率分析算法实现了 APT 检测平台[14]。而目前的网络入侵检测系统，通常依靠签名匹配，和服务器端口作为主要的识别技术，例如本项目中所使用的 Bro 入侵检测系统[15]。此外，还有研究人员提出了基于入侵检测系统报告的检测方案[16]，本文中所实现的 BroAPT 系统即是采用了这一方案。

2.2　Bro 入侵检测系统

入侵检测系统（Intrusion Detection System，IDS）是用于检测网络中恶意流量的软件应用。其通常部署在网络边界 DMZ 处，对整个网络中通过的流量进行监控。入侵检测系统的检测方案主要有以下两种：

(1) 基于异常流量数据库

对通过的流量进行匹配。匹配的方式通常是对流量的签名进行查找，与杀毒软件的常用方案类似。系统需定期更新获取异常流量数据库，从而保证监测的准确性。然而，这种方式对于未知的攻击，例如 APT 攻击等，效果并不理想。

(2) 基于流量中的异常行为进行检测

系统可检测未知的攻击方式。通常，系统使用机器学习（machine learning）算法，构建可信行为模型，从而将流量中的行为与此进行比较，判断其是否存在异常行为。然而，这种方式存在误报（false positive）的问题，例如未知的正常行为也可能被系统认定为异常行为。

目前常见的入侵检测系统有 Snort、OSSEC、Suricata 等。其中，Snort 具备入侵检测系统和入侵防御系统（Intrusion Prevention System，IPS）的功能，实现了对 IP 协议的实时流量分析和日志记录[17]；OSSEC 是基于主机的入侵检测系统，其主要针对于系统的日志分析、完整性检查、Windows 注册表监测和 rootkit 攻击检测等[18]；Suricata 则是由开源信息安全基金会（Open Information Security Foundation，OISF）开发的入侵检测及防御系统，其功能较为完善，并与 Snort 相兼容[19]。这些入侵检测系统都是基于实际流量，并面向流量数据包的，即其对流量处理的基本单位为流量包（packet）。

Bro 是由美国伯克利大学开发的一个基于网络的入侵检测系统框架，现已更名为 Zeek[15][20] 。相较于上述其他入侵检测系统，Bro 是一个基于网络事件（event）的异步系统，其对流量处理的基本单位为连接（connection）[21] 。此外，由于设计上的完备性，Bro 框架适用于大流量的分析和处理，效率较高，基于连接，并能够在一定程度上抵御针对 Bro 框架本身的攻击。

表 2-1 Bro 与 Snort 功能对比[21]

特征	Bro	Snort
上下文签名分析	有	无
可拓展性	较高	一般
高速流量处理	较强	一般
用户社群	较小	较大
图形化配置	无	有
图形化分析	少量	许多
安装/部署	复杂	简单
操作系统兼容性	类 UNIX 系统	全平台支持

网络事件是 Bro 的基本概念，Bro 将内核中的流量数据过滤抽象为一系列的网络事件，并由处理函数（event handler）对这些事件进行响应，如更新状态信息、产生新的网络事件、将数据写入磁盘，以及通过 syslog 协议实现实时的日志通知和报警。此外，Bro 在其设计上更注重于高性能和可拓展性。Bro 更加注重于普适性的全局监测（monitoring），而不是针对性的异常流量检测和防控（blocking）[15] 。

Bro 的部署场景通常为一个网络区域的边际，即网关处，对该处通过的网络流量进行实时监听和检测。同时，Bro 通过限制检测规则的复杂度，避免了由于处理不够及时而导致 `libpcap` 内核丢包的发生。从设计上说，Bro 是具备实时性的，其可对网络中实际通过

的流量进行检测，并实时地通过日志系统进行记录和报警。在架构上，Bro 的实现机制和检测规则（policy）相互分离，可通过编写脚本实现高度个性化的监测配置。此外，Bro 在设计时还兼顾了鲁棒性，避免了如 SYN 洪流攻击、资源耗尽型拒绝服务攻击（Denial of Service，DoS）等对系统监测效果的影响。

2.2.1 系统架构

Bro 入侵检测系统主要分为两个概念层——网络事件层（event engine），将原始的网络流量简化为高层的网络事件，如 TCP 连接（connection）和 UDP 数据流（flow）等；脚本解释器（policy script interpreter），用于解析和运行用户编写、实现定制化监测方案的 Bro 脚本。系统架构如图 2-1 所示。

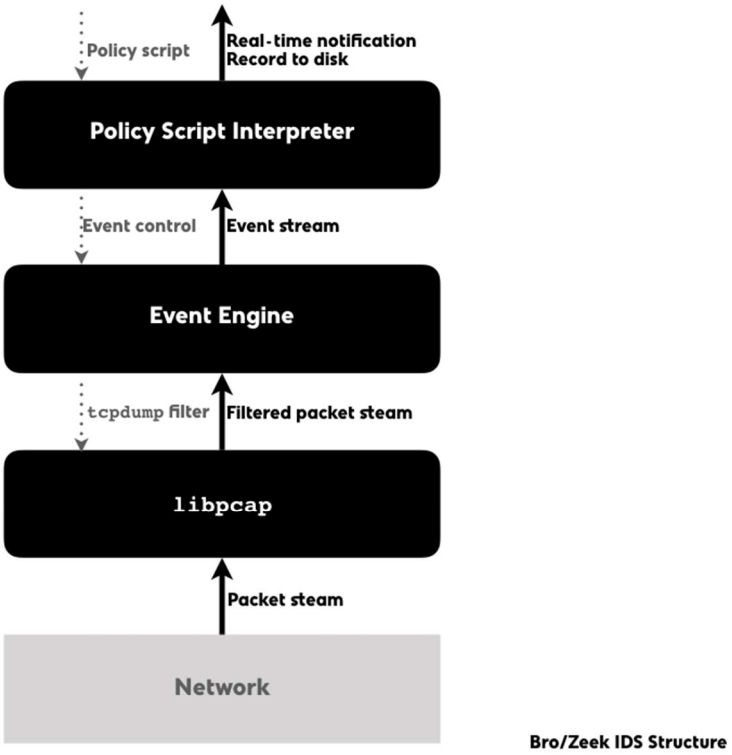

图 2-1 Bro 入侵检测系统架构

(1) libpcap 内核

Bro 入侵检测系统将 libpcap[22]作为内核，从而无需考虑底层（即链路层）协议及网络流量嗅探抓取；由于 libpcap 的通用性，也使得 Bro 可移植于不同的 UNIX/Linux 系统，并在条件允许时，通过 libpcap 的调用接口实现流量解析等的硬件加速。此外，利用 libpcap 读写 PCAP 文件的功能，Bro 还可实现离线监测、后期处理与分析的功能。

通常，嗅探抓包在操作系统的核心态进行，而事件的生成和处理在用户态进行。若系统存在内核级流量过滤工具，如 iptables 及 BPF 等，则可在内核中直接对流量进行分离和压缩，提高监测效率，减小系统开销。同时，由于减少了系统所需处理的流量，在很大程度上，也避免了内核丢包的情况发生。

Bro 通过 TCP 流量包中的 SYN（同步控制位）、FIN（结束控制位）以及 RST（重置控制位）标记并分离 TCP 连接，以 TCP 连接或 UDP 数据流为单位对网络事件进行生成和处理。此外，在 libpcap 的接口中，需设置 snaplen，即嗅探流量时每个网络包所抓取长度。由于 Bro 需要处理完整的网络流量，因此在实现中 Bro 将其设置为抓取完整流量。

(2) 网络事件层

在 libpcap 内核完成对流量数据包的抓取和处理解析后，Bro 将这些数据传递给网络事件层。网络事件层首先对流量包的完整性及校验和（checksum）进行检查，如果是异常流量包，则将其抛弃并报错。随后，Bro 将根据其传输层协议进行分流。

对于 TCP 流量，Bro 首先对流量包的检验和进行检查；随后根据其源 IP 地址端口及目的 IP 地址端口，将流量以 TCP 连接为单位分离；而后根据 SYN/FIN/RST 控制位状态，更新相应连接的状态，如连接请求（connection_attempt）、连接建立(connection_established)、连接拒绝（connection_rejected）和连接断开（connection_finished）等；最后，Bro 将处理流量包中的数据确认信息（data acknowledgement），并生成对应的事件（event）以处理数据包中的载荷（payload）。

对于 UDP（User Datagram Protocol，用户数据报文协议）流量，Bro 将根据其源 IP 地址端口及目的 IP 地址端口，将流量以 UDP 数据流为单位分离；随后根据其通信方向生成 UDP 请求（udp_request）和 UDP 响应（udp_reply）事件，并生成对应的事件以处理数据包中的载荷。

(3) 脚本解释器

当网络事件层完成对流量包的处理后，其将生成一系列的网络事件。这些事件将被缓存在一个 FIFO 队列中，并会立刻进行处理；当这些事件处理完毕后，Bro 将回到内核层，读入下一个流量包，进行上述处理过程。

Bro 脚本实质上是基于事件驱动的异步编程语言。用户可通过编写脚本，定义对事件的处理方式；同时，还可通过脚本，实现对应用层协议处理的拓展。例如，如下脚本可使得 Bro 计算流量中，嗅探到所有文件的 MD5 哈希值，并记录在对应的日志文件 files.log 中：

```
1  event file_new(f: fa_file)
2      {
3      Files::add_analyzer(f, Files::ANALYZER_MD5);
4      }
```

2.2.2 系统实现

Bro 入侵检测系统采用了单线程设计。这是由于考虑到了如果为每一个网络事件的处理函数（event handler）创建一个线程，则可能最终导致事件处理函数之间的资源竞争等问题。然而，在系统正式开始运行前，其将展开 Bro 脚本中的域名（hostname）常量，采用异步请求的方式，通过 DNS 查询得到这些域名常量所对应的 IP 地址。

由于系统中使用了大量的计时器，如 TCP 连接超时的计时器等，考虑到插入和删除操作时间复杂度都是 O(log(N)) 的单优先级堆（single priority heap），在元素数量较多时的性能表现较差，因此使用了日历队列（calendar queue）[23]以实现对计时器的管理。同时，由于计时器超时分布并不均匀，以及系统载荷有限，因此系统为了负载均衡，放弃了对计

时器超时处理的精确性，即——限制每次处理超时计时器的数量；当系统空闲时处理计时器，而不是等待至下一流量包到达。

在 Bro 脚本的处理上，Bro 入侵检测系统采取了类似 Python 语言的处理方案：将 Bro 脚本编译为抽象语法树（Abstract Syntax Tree，AST），从而在执行时按需调用。由于计时器等交互性元素的存在，脚本解释器在抽象语法树之外，另有一条用于维护运行现场的执行栈。

如上文所述，Bro 入侵检测系统采用了 `libpcap` 作为系统内核，因此其除支持对网络流量的实时监听和检测外，还可对抓取流量得到的 PCAP 文件进行“离线”分析。同时，通过 Bro 的日志系统，用户还可对流量中的事件进行进一步分析和处理。

2.2.3 Bro 脚本语言

如上文所述，Bro 入侵检测系统是基于网络事件（event）的异步系统。Bro 通过 `libpcap` 内核完成对流量的初步解析后，将原始的流量数据转化为一系列的网络事件，并传递至网络事件层中，由 Bro 脚本中定义的事件处理函数（event handler）对这些事件进行响应。

表 2-2 Bro 语言中的常见类型

类型	描述
`bool`	布尔值
`count/int/double`	数值
`time/interval`	时间类型
`string`	字符串
`pattern`	正则表达式
`port/addr/subnet`	网络类型
`enum`	枚举类型
`table/set/vector/record`	容器类型
`function/event/hook`	可执行函数
`file`	文件对象

Bro 脚本实质上是基于事件驱动的异步编程语言。Bro 语言是静态类型语言，其定义了多种不同的类型，如下表所示。其中，有三种可执行（executable）的函数类型，即线性函数、事件函数和谓词函数。

线性函数，由 `function` 定义声明，与 Python 等传统编程语言中的函数类似，是对一定功能代码的封装，以便需要时进行调用。事件函数，由 `event` 定义声明，为 Bro 语言中基于网络事件的特有概念，是对其所绑定网络事件的响应——当某一网络事件生成时，Bro 系统的事件处理层将调用所有与该事件相绑定的事件函数——因此，事件函数是异步执行的函数。谓词函数，由 `hook` 定义声明，可视为一种语法糖（syntactic sugar）——对于同一函数名，可定义多个不同的谓词函数，这些函数在被调用时将被同时执行，但如若其中一个函数以 `break` 语句强制退出，则这些函数将退出执行，并返回执行失败（布尔值 `F`）；反之，若所有函数均正常退出，则返回执行成功（布尔值 `T`）——在语法上，等同于以循环结构反复调用这些函数，并检查执行情况，如果其中一个函数以 `break` 语句强制退出，则退出循环，并返回执行失败；反之，返回执行成功。

2.2.4 日志系统

Bro 入侵检测系统的离线分析和实时通知均是基于其日志系统而实现的。Bro 日志系统在对网络流量进行解析和分析时，会生成多种日志文件，常用日志文件如下表所示：

表 2-3 常用的 Bro 日志文件

日志	描述
conn.log	TCP/UDP/ICMP 连接记录
dns.log	DNS 查询记录
ftp.log	FTP 连接记录
http.log	HTTP 请求与响应记录
files.log	文件分析记录
pe.log	Windows 可执行文件 PE（Portable Executable）记录

在 Bro 系统中，这些日志可通过 Bro 脚本对其实现拓展和修改，还可自定义日志文件。在第四章中，将对此进行详细介绍。

2.3 相关背景介绍

2.3.1 TCP 流量重组

TCP/IP 协议族（Internet Protocol Suite, IPS）是当前互联网网络通信的基本模型，由链路层、网络层、传输层和应用层构成。本项目主要关注于应用层中所传输的数据和文件，如 FTP 协议，HTTP 协议与 SMTP 协议等应用层协议所交换、通信、传输的数据和文件，并将其提取并通过一定程序进行分析和检测。然而，由于上层协议数据窗口大小有限，因此在传输较大的应用层协议数据时，其将会在传输层被 TCP 协议分为数段进行传输。

TCP 分段，是指应用层数据包分段，将数据包分割为更小的单位。当数据包的长度大于最大分段大小（Maximum Segmentation Size，MSS）时，为避免 TCP 包在网络层因长度大于最大传输单元（Maximum Transmission Unit，MTU）被 IP 协议分片，在数据包的发送端，TCP 会将其分解为较小的数个区段，以使其能够在当前网络中正确地传输。同时，在数据包的接收端，TCP 需将其重组还原，得到原始的应用层数据包，这一过程被称为 TCP 重组。

在 RFC791[24]中，描述了 IP 协议的分片与重组算法。其中重组算法以源 IP 地址、端口及目的 IP 地址、端口四元组作为同一原始数据包的标识位（identifier），随后将 IP 协议数据中的流量根据其标识位进行组合，通过已接收比特位表（fragment received bit table）记录重组状态和进度，最终得到原始的数据包。而在 RFC815[25]中，则提出了空缺描述符表（hole descriptor list），用于替代空间开销较大的已接收比特位表，以进行效率更高的数据包重组。

根据上述算法，以及 TCP 协议数据包的结构，本项目设计了一种 TCP 流量重组的算法。[26]算法描述如下：

表 2-4 TCP 协议数据包结构

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1								
Source Port								Destination Port
Sequence Number								
Acknowledgement Number								
Data Offset	Reserved		URG	ACK	PSH	RST	SYN	FIN	Window
Checksum								Urgent Pointer
Options								Padding
Payload (Application Layer)								

表 2-5 算法中使用到的术语定义

名称	定义
DSN	数据序列号
ACK	确认号
SYN	同步标识位
FIN	结束标识位
RST	重置标识位
BUFID	缓存标识符
HDL	空缺描述符表
ISN	初始序列号
src	源 IP 地址
dst	目的 IP 地址
srcport	源 TCP 端口
dstport	目的 TCP 端口

表 2-6 TCP 流量重组算法（伪代码）

```
1   DO {
2       BUFID <- src|dst|srcport|dstport|ACK;
3       IF (SYN is set) {
4           IF (buffer with BUFID is allocated) {
5               flush all reassembly for this BUFID;
6               submit datagram to next step;
7           }
8       }
9
10      IF (no buffer with BUFID is allocated) {
11          allocate reassembly resources with BUFID;
12          ISN <- DSN;
13          put data from fragment into data buffer with BUFID
                [from octet fragment.first to octet fragment.last];
```

```
        update HDL;
    }

    IF (FIN is set or RST is set) {
        submit datagram to next step;
        free all reassembly resources for this BUFID;
        BREAK.
    }
} give up until (next fragment);

update HDL: {
    DO {
        select the next hole descriptor from HDL;

        IF (fragment.first >= hole.first) CONTINUE.
        IF (fragment.last <= hole.first) CONTINUE.

        delete the current entry from HDL;

        IF (fragment.first >= hole.first) {
            create new entry "new_hole" in HDL;
            new_hole.first <- hole.first;
            new_hole.last <- fragment.first - 1;
            BREAK.
        }

        IF (fragment.last <= hole.last) {
            create new entry "new_hole" in HDL;
            new_hole.first <- fragment.last + 1;
            new_hole.last <- hole.last;
            BREAK.
        }
    } give up until (no entry from HDL)
}
```

2.3.2 MIME 类型

多用途互联网邮件拓展（Miltipurpose Internet Mail Extensions，MIME）原是用于在电子邮件中对其传输数据及内容的一种拓展格式，由数据头和数据体组成。其中，数据头中的“Content-Type”字段，描述了数据体的媒体类型，亦即数据格式。[27]这些媒体类型（media type）在互联网中，广泛用于描述数据文件的类型和格式，如 HTTP 协议、SMTP 协议及 SIP 协议中的相关字段，因而又被称为 MIME 类型。

MIME 类型由两部分组成：媒体类型（type）和子类型（subtype）。前者表示其所属大类，后者表示其具体类型。标准格式为

```
type / subtype
```

以 HTML 文件为例，其 MIME 类型应为 `text/html`；而 PDF 文件的 MIME 类型则为 `application/pdf`。

目前，在互联网号码分配局（Internet Assigned Numbers Authority，IANA）中，已有十个媒体类型被注册，约有 2071 项不同的 MIME 类型[28] 。

表 2-7 已在 IANA 中注册的媒体类型

媒体类型	描述	代表 MIME 类型	代表文件类型
application	应用文档	application/zip	ZIP 压缩文件
audio	音频文档	audio/mp4	MP4 音频文件
font	字体文档	font/otf	OpenType 字体文件
example	样例		
image	图片文档	image/jpeg	JPEG 图片文件
message	消息/信息	message/rfc822	电子邮件信息
model	三维图形文档	model/vrml	VRML 文件
multipart	混合类型	multipart/encrypted	加密信息
text	文本文档	text/x-python	Python 脚本
video	视频文档	video/x-matroska	MKV 视频文件

在 UNIX/Linux 操作系统中，命令行工具 `file` 是常用的文件类型探测工具。其基于对 `libmagic` 数据库的封装，对各种文件的 MIME 类型和内容特征（signature 或 magic number）进行了整理和关联，从而实现基于文件内容的文件类型探测。Bro 入侵检测系统的文件分析模块也采用了 `libmagic` 数据库，并在其基础上进行了优化和修改，以适配网络流量中的文件探测和分析。

在本项目中，BroAPT 系统对网络流量中传输的文件，以 MIME 类型为单位，进行针对性的检测。一方面，系统避免了对所有类型的文件采用相同的算法进行检测，这种笼统的检测方式在实际应用中的检测准确率并不理想；另一方面，通过配置文件，用户可对不同 MIME 类型的文件采用不同的检测方式，即可自行定制调整，提高检测准确率的同时，也提高了 BroAPT 系统的可拓展性和普适性。

2.3.3 YAML 语言

YAML 语言[29]，全称为“YAML Ain't Markup Language”，是一种可读的数据序列化（data-serialization）语言。其常用于配置文件，也见于在应用程序中用于存储数据和传输通信等。BroAPT 系统采用了 YAML 语言编写系统中的配置文件。

YAML 语言编写的配置文件，在一定程度上可替代 XML，尽管其语法与标准通用标记语言（Standard Generalized Markup Language，SGML）中的规范并不兼容。在 YAML 语言中，其使用了 Python 风格的空格缩进表示逻辑层级，并在 YAML 1.2 中使用 `[]` 表示列表、`{}` 表示字典以实现对 JSON 语法的完全兼容。

2.4 本章小结

本章首先对当前 APT 检测的研究现状进行了简要介绍，对基于机器学习算法和流量综合分析两种方式的相关研究进行了讨论，并简要概述了本项目所采用的方案。随后，对本文所使用的 Bro 入侵检测系统进行了详细描述，介绍了该系统的设计架构和实现特点，并对 Bro 脚本语言的语法特点进行了举例和描述，同时介绍了 Bro 日志系统及其中常用的日志文件。

此外，本章还对文中使用到的背景知识进行了介绍——TCP 流量重组算法，文件的 MIME 类型，以及用于配置文件的 YAML 语言。在下一章中，本文将对课题所实现的 BroAPT 系统在设计和实现上进行简要介绍。

第三章　系统设计综述

本项目实现了 BroAPT 系统。BroAPT 是一个基于 Bro 的 APT 监测系统，具有较高的性能和可拓展性。为了达成这一目的，BroAPT 系统充分利用了 Bro 入侵检测系统的高性能优势，采用多进程的方式对抓取的 PCAP 文件进行处理，并提取流量中传输的文件；同时，BroAPT 系统在运行时，还支持自定义的脚本和函数，可对 BroAPT 系统的原有功能进行拓展，生成定制的日志文件或产生特定的报警行为等；在对提取得到的文件进行检测时，BroAPT 系统还可通过配置文件，调整使用特定的检测方式或算法，以处理不同类型的文件。以下，将对 BroAPT 系统的设计和架构进行简要介绍。

3.1　系统架构

BroAPT 系统在设计上继承了 Bro 入侵监测系统的高效性与可拓展性，并通过 Python 对其进行了进一步地封装和优化。其核心主要由两个模块组成——BroAPT-Core，提取模块：通过编写 Bro 脚本，运行 Bro 读取处理 PCAP 文件，并提取中其中所传输的文件；BroAPT-App，检测模块：基于用户定义的 API 配置文件，对特定类型的文件执行特定的检测命令，找出其中的恶意文件。系统架构如图 3-1 所示。

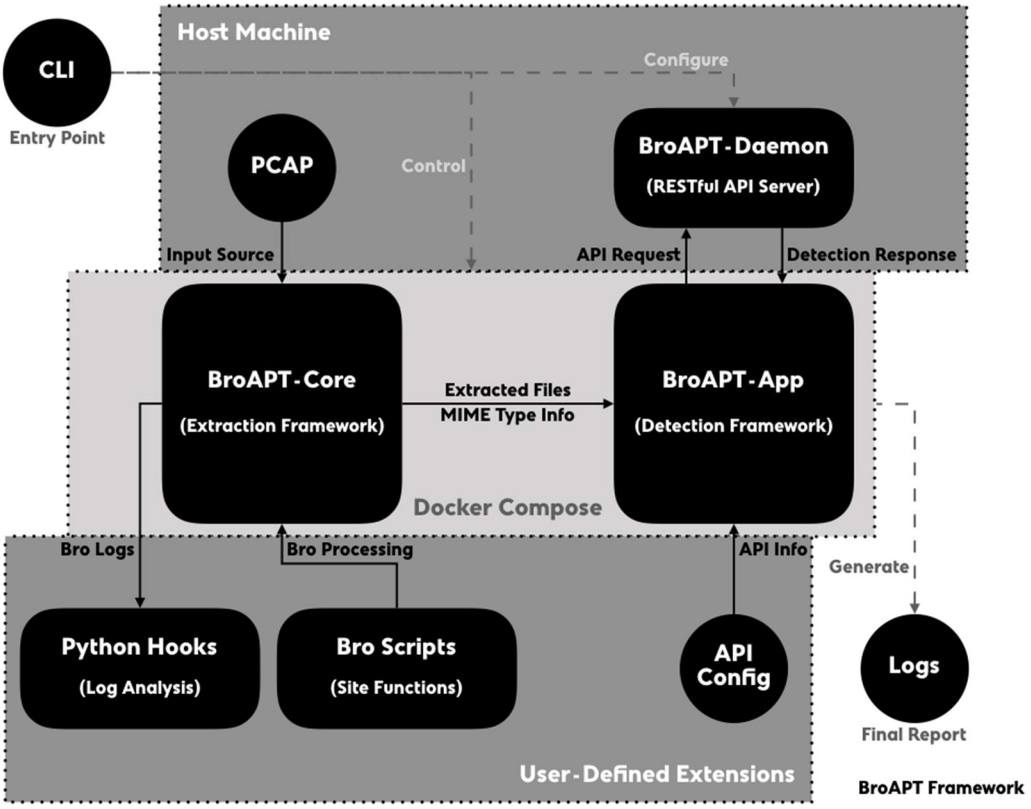

图 3-1 BroAPT 系统架构

根据运行环境的不同，可将系统分为三个逻辑层面。最底层是系统所运行的宿主机，其作为网关实时抓取并产生 PCAP 文件；中间层是 Docker 容器[30]，系统的核心模块，即提取和检测模块均在此运行；第三层则是对系统的配置和拓展，用户可根据实际需求进行调整。

BroAPT 系统的运行流程如下：

(1) 通过命令行接口（command line interface，CLI），配置 BroAPT-Daemon 系统服务进程（service）并将其启动，同时开始运行 Docker 容器中的系统核心模块。CLI 的参数列表如下：

```
usage: broapt-appd [-t HOST] [-p PORT] [-c DOCKER_COMPOSE]
                   -d DUMP_PATH -l LOGS_PATH  -r API_ROOT
                   -a API_LOGS  [-i INTERVAL] [-m MAX_RETRY]

BroAPT-App Daemon

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

server arguments:
  -t HOST, --host HOST  the hostname to listen on
  -p PORT, --port PORT  the port of the webserver

compose arguments:
  -c DOCKER_COMPOSE, --docker-compose DOCKER_COMPOSE
                        path to BroAPT-App docker-compose.yml
  -d DUMP_PATH, --dump-path DUMP_PATH
                        path to extracted files
  -l LOGS_PATH, --logs-path LOGS_PATH
                        path to log files

API arguments:
  -r API_ROOT, --api-root API_ROOT
                        path to detection APIs
  -a API_LOGS, --api-logs API_LOGS
                        path to API runtime logs

runtime arguments:
  -i INTERVAL, --interval INTERVAL
                        sleep interval
  -m MAX_RETRY, --max-retry MAX_RETRY
                        command retry
```

其中，服务进程在启动时，首先将系统的 Docker 镜像启动，运行其中的系统核心模块，随后进入主循环（mainloop），等待来自 Docker 镜像的检测请求；在服务进程退出时，将退出主循环并关停核心模块。

(2) BroAPT-Core 提取模块以宿主机抓取的 PCAP 流量文件作为输入，通过 Bro 脚本进行处理，提取出流量中传输的文件，同时生成 Bro 日志文件。此外，模块调用分析函数对这些日志进一步处理和分析，得到精简后的日志。

(3) BroAPT-App 检测模块则获取从流量中提取的文件，根据这些文件的 MIME 类型信息，以及 API 配置文件中的设置，对文件进行针对性的检测，并生成最终的检测日志。在检测时，模块根据其 API 配置，在 Docker 容器中运行检测命令，或向服务进程发送请求，在宿主机执行相应命令，如调用其他 Docker 镜像或运行虚拟机等。

通过 BroAPT 系统生成的检测日志和其他系统日志，如 Bro 日志和用户通过分析函数定义并生成的日志等，可得出其中的恶意流量；同时，还可使用大数据及机器学习算法等进行分析，从而检测流量中潜在的恶意攻击。

3.2 系统实现

BroAPT 系统采用了多进程（multiprocessing）设计。由于系统使用 Python 语言进行实现，而 Python 的主流实现 CPython 中由于 GIL（Global Interpreter Lock，全局解释器线程锁）技术[31]，其多线程（multithreading）实际上没有并行，即同一时刻至多有一个线程运行，故无法达到提高效率的目标；并且 BroAPT 系统中，进程间的通信和协调并无较大开销，对系统整体运行的影响可忽略不计，因此 BroAPT 系统采用了多进程的设计方案。

在 BroAPT 系统的核心模块中，主要有三组进程，如图 3-2 所示。

首先，在 BroAPT-Core 提取模块中，系统为每一个输入的网络流量 PCAP 文件创建一个子进程；在这些子进程中，Bro 载入用户提供的 Bro 脚本，与系统中的文件提取脚本一起，处理和分析 PCAP 文件，得到流量中传输的文件和 Bro 日志。

完成对 PCAP 文件的 Bro 初步分析后，系统通过同步队列（queue），向另一组进程告知生成的 Bro 日志信息。这一组进程主要负责对 Bro 日志的进一步分析和处理。用户使用编写分析函数（hook），并将这些函数添加注册到 Python 进程中。随后，每当系统接收到一条 Bro 日志信息，其便将为每一个分析函数创建一个子进程，由这些子进程完成对 Bro 日志的进一步分析和处理。

同时，对于从流量中提取得到的文件，系统将通过另一同步队列，向第三组进程告知文件的相关信息，如文件路径和 MIME 类型等。这一组进程即是 BroAPT-App 检测模块的组成部分。系统将为每一个提取得到的文件创建一个子进程；这些子进程将从 API 配置文件中，根据待检测文件的 MIME 类型，选择对应的检测配置，并执行配置中的检测脚本，生成恶意文件检测日志。

考虑到 BroAPT 系统的核心模块是运行在 Docker 容器中的，其执行权限有所限制。然而，在 BroAPT-App 检测模块中，API 配置可能使用到了需要完整权限和运行环境的命令，如调用另一 Docker 镜像进行检测，因此系统设计了 BroAPT-Daemon 系统服务进程。这一服务进程将在宿主机上以完整权限运行，通过 REST API 与系统核心模块，即 BroAPT-App 检测模块，进行通信，从而在宿主机上执行检测程序，并生成恶意文件检测日志。

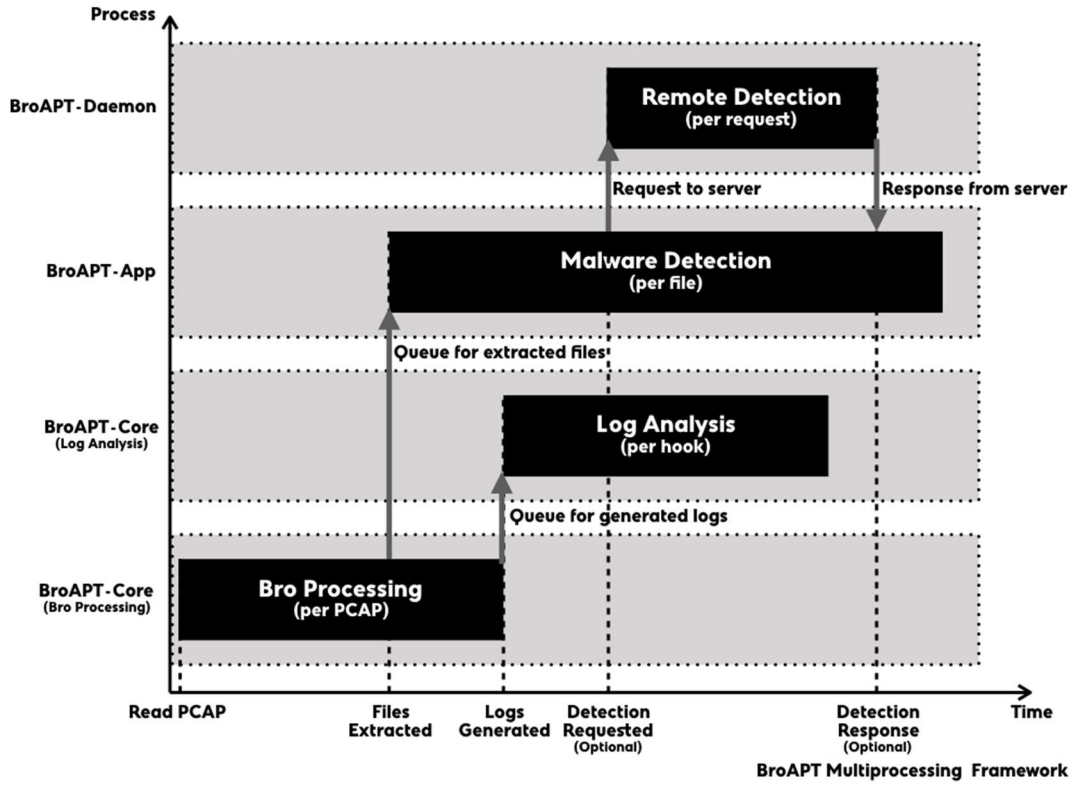

图 3-2 BroAPT 进程逻辑

在延展性上，BroAPT 系统充分利用了 Bro 入侵检测系统的可拓展性——执行提取文件的 Bro 脚本时，同时加载用户定义的拓展函数（site function）；并在 BroAPT-Core 提取模块中，引入分析函数，对 Bro 生成的日志文件进行自定义的分析处理。此外，在 BroAPT-App 检测模块中，系统还可通过 API 配置文件，调整针对不同 MIME 类型文件的检测方案。

如上文所述，BroAPT 系统结合了 Bro 入侵检测系统的日志系统，实现了对日志的离线分析，以便用户实施进一步的分析和处理。

3.3 本章小结

本章首先介绍了 BroAPT 系统的架构设计，其核心模块由 BroAPT-Core 提取模块与 BroAPT-App 检测模块组成，核心模块在 Docker 容器中运行；以及 BroAPT-Daemon 服务进程，其作为 REST API 服务器在宿主机上运行。随后，介绍了 BroAPT 系统的运行逻辑与流程。

本章还对 BroAPT 系统的具体实现进行了详细介绍。系统采用了多进程的实现方式，进程间的协调与并行逻辑如上文图 3-2 所示。此外，本章对系统的可拓展性进行了简要描述，介绍了 BroAPT-Daemon 服务进程的实现背景。在第四章中，本文将对 BroAPT 系统的核心模块之一 BroAPT-Core 提取模块作详细介绍。

第四章　基于 Bro 的文件提取与日志分析

4.1　BroAPT-Core 提取模块

本项目所设计的 BroAPT 系统是一个基于 Bro 的 APT 监测系统，系统核心由 BroAPT-Core 提取模块和 BroAPT-App 检测模块构成。其中，BroAPT-Core 提取模块基于 Bro 入侵检测系统，着手于对网络流量 PCAP 文件进行分析和处理，生成日志并提取出流量中传输的文件。

4.1.1　模块架构与实现

BroAPT-Core 提取模块是 BroAPT 系统的核心模块之一，主要功能为基于 Bro 入侵检测系统的文件提取和日志分析。模块大致可分为三个阶段，如图 4-1 所示。

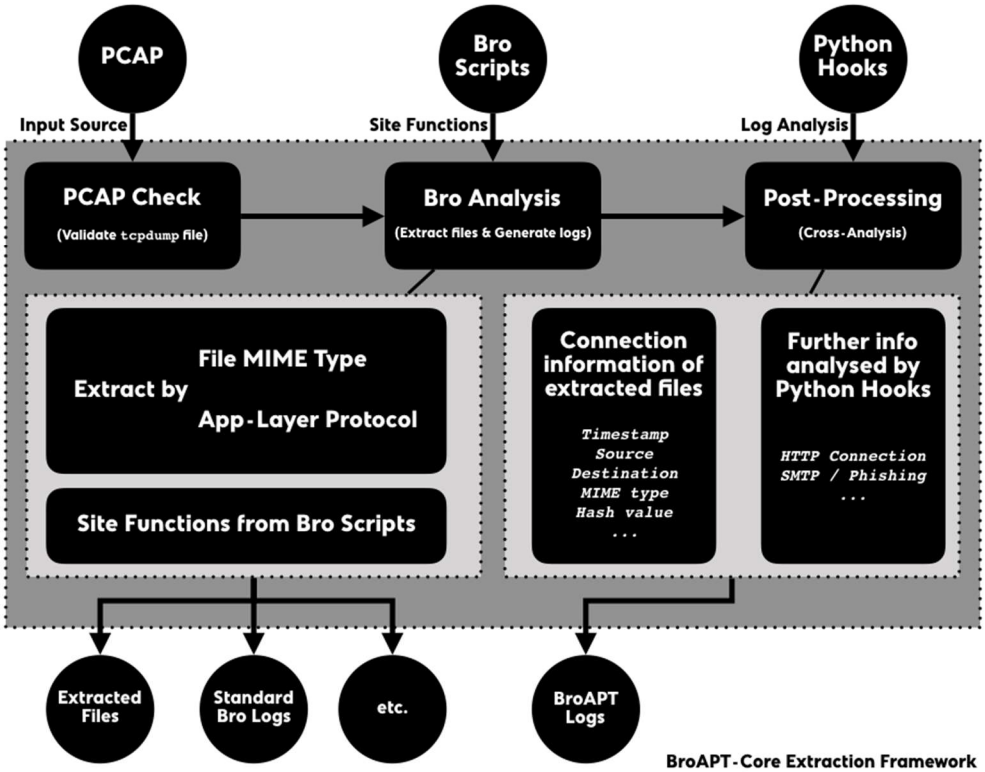

图 4-1 BroAPT-Core 提取模块架构

首先，模块对输入源进行扫描，获取尚未处理的网络流量 PCAP 文件，即新产生的 PCAP 文件。模块为每一个 PCAP 文件创建一个子进程，如上文所述。在子进程中，模块加载内置的文件提取脚本，以及由用户编写的 Bro 功能拓展脚本，调用 Bro 入侵检测系统对 PCAP 文件进行分析和处理，得到从流量中提取的文件，与由 Bro 日志系统生成的日志

文件。随后，模块调用内置的分析函数对日志文件进行解析，从原始日志文件中提取信息，形成分析日志。

4.1.2 Bro 事件与处理

在 BroAPT-Core 提取模块中，内置的文件提取脚本主要使用到了 `file_sniff` 事件。每当 Bro 系统事件处理层从流量中发现传输的文件后，Bro 系统将生成 `file_new` 事件，即探测到新文件。随后，Bro 系统根据 `default_file_bof_buffer_size` 常量设置的缓存大小，对文件的 MIME 类型进行检测，并生成 `file_sniff` 事件，即嗅探到某种类型的文件。通过 `file_sniff` 事件，可以获得由 Bro 系统文件分析模块探测的 MIME 类型，从而可以实现基于 MIME 类型的文件分类和提取。

考虑到 BroAPT 系统的应用场景，并不需要提取网络流量中传输的所有文件，而只需要对特定的文件类型或特定的网络协议进行提取，因此，BroAPT-Core 提取模块在内置的文件提取脚本中还使用了谓词函数（`hook`）。这些函数在模块中以谓词（predication）的形式定义，分为提取（extract）和忽略（ignore）两种。顾名思义，提取谓词用于判断特定的文件是否需要提取，而忽略谓词则用于判断是否对特定的文件进行处理。

模块在使用时，可通过环境变量设置需要提取的文件类型以及协议类型，并生成对应的提取谓词函数。在运行时，Bro 系统将根据这些谓词函数，提取网络流量中对应 MIME 类型的文件，或对应应用层协议中传输的文件。例如，如下谓词函数可提取网络流量中传输的所有 PDF 文件：

```
hook extract(f: fa_file, meta: fa_metadata)
    {
    if ( meta?$mime_type and meta$mime_type == "application/pdf" )
        break;
    }
```

而如下的谓词函数则可提取所有由 HTTP 协议传输的文件：

```
hook extract(f: fa_file, meta: fa_metadata)
    {
    if ( f$source == "HTTP" )
        break;
    }
```

4.1.3 Bro 日志与分析

在 BroAPT 系统中，基于 `pandas` 数据分析库实现了对 Bro 日志文件的解析接口。通过接口函数，BroAPT-Core 解析模块在 Bro 系统完成对网络流量 PCAP 文件的解析和处理后，对 Bro 系统生成的日志进行进一步的分析。例如，通过对 `conn.log` 和 `files.log` 的交叉分析，模块生成了提取文件的相关信息，如传输该文件连接的时间戳、源与目的、文件的 MIME 类型和哈希（hash）值等，如下表所示。

表 4-1 BroAPT 系统生成的提取文件信息日志

字段	Bro 类型	描述
timestamp	float	传输该文的 TCP/UDP/ICMP 连接的时间戳
log_uuid	string	Bro 日志源的 UUID
log_path	string	Bro 日志源的绝对路径（在 Docker 容器内）
log_name	string	Bro 日志源的相对路径（相对于系统日志路径）
dump_path	string	该文件的绝对路径（在 Docker 容器内）
local_name	string	该文件的相对路径（相对于系统提取路径）
source_name	string	该文件在流量中的原始文件名称（如有）
hosts	vector	传输该文件的主机名称（发送端与接收端）
text	vector	传输该文件的连接（源与目的 IP 地址和端口）
bro_mime_type	string	Bro 文件分析模块检测得到的 MIME 类型
real_mime_type	string	BroAPT 系统由 libmagic 检测得到的 MIME 类型
hash	table	该文件的哈希值（包括 MD5、SHA1 和 SHA256）

4.2 模块拓展型设计

如前文所强调的，BroAPT 系统是一个具有较强可拓展性的 APT 检测系统。其拓展性一方面来源于 Bro 入侵检测系统，另一方面则来源于系统核心模块中的架构设计。以下，将对 BroAPT-Core 提取模块中的拓展性设计作详细介绍。

4.2.1 模块配置和接口

由于 BroAPT 系统的核心模块均是在 Docker 容器中运行，因此 BroAPT-Core 提取模块采用了环境变量的配置接口。通过环境变量，可配置模块的日志路径、文件提取路径、文件提取 MIME 类型白名单和传输协议白名单等，如下表所示。

表 4-2 BroAPT-Core 提取模块环境变量表

变量名	描述
BROAPT_CPU	系统 Bro 处理并行进程数
BROAPT_INTERVAL	模块运行周期间隔
BROAPT_DUMP_PATH	文件提取路径
BROAPT_PCAP_PATH	网络流量 PCAP 文件源路径
BROAPT_LOGS_PATH	系统日志路径
BROAPT_MIME_MODE	提取文件以 MIME 类型聚类
BROAPT_JSON_MODE	Bro 系统以 JSON 格式生成日志
BROAPT_BARE_MODE	以极简模式运行 Bro 系统
BROAPT_NO_CHKSUM	Bro 系统运行时不检查检验和
BROAPT_HASH_MD5	计算提取文件的 MD5 值
BROAPT_HASH_SHA1	计算提取文件的 SHA1 值
BROAPT_HASH_SHA256	计算提取文件的 SHA256 值
BROAPT_X509_MODE	在 Bro 日志中加入 X509 证书信息
BROAPT_ENTROPY_MODE	在 Bro 日志中加入文件熵值信息
BROAPT_LOAD_MIME	文件提取模块 MIME 类型白名单

续表 4-2

变量名	描述
BROAPT_LOAD_PROTOCOL	文件提取模块传输协议白名单
BROAPT_FILE_BUFFER	Bro 文件分析模块重组缓存大小
BROAPT_SIZE_LIMIT	Bro 文件分析模块提取大小限制
BROAPT_HOOK_CPU	系统日志分析并行进程数

其中，BROAPT_LOAD_MIME 和 BROAPT_LOAD_PROTOCOL 环境变量均是以逗号（,）或分号（;）分隔的字符串，例如：

```
BROAPT_LOAD_MIME="\
    application/vnd.microsoft.portable-executable; \
    application/pdf; \
    application/vnd.android.package-archive \
"
```

表示系统将提取 application/vnd.microsoft.portable-executable、application/pdf 和 application/vnd.android.package-archive 三种 MIME 类型的文件；通俗地说，即是 Windows 可执行文件 PE、PDF 文件和 Android 应用文件 APK。模块会根据它们的值，在内置 Bro 脚本中生成对应的提取谓词函数，如上文所述。

而在 BroAPT-Core 提取模块的 Python 代码中，模块还提供了对 Bro 日志文件的解析函数 logparser.parse，和一些工具函数，如判断日志解析后某一字段是否为空（None）的检查函数 utils.is_nan，以及大量全局常量，如系统日志路径 const.LOGS_PATH 等。对于日志解析函数 logparser.parse，其支持 ASCII 模式与 JSON 模式下生成的 Bro 日志文件，保留其中的附加信息，如日志的生成时间段和文件格式等。以如下日志文件为例：

```
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   http
#open   2019-05-23-06-33-10
#fields scrip   ts  url dstip   src_port    method
#types  addr    time    string  addr    port    string
128.3.150.243   1504716966.628123   http://104.16.18.111/redirect
.aspx?linkID=231078&eid=52837  104.16.18.111   61272   GET
202.120.15.33   1504716966.722234   http://www.cnki.net
103.227.81.121  50025   GET
19.19.100.143   1504716966.842344   http://10.23.12.1/upload.html
   10.23.12.1  62212   POST
```

通过 logparser.pars 解析得到的日志对象表述（repr）如下：

```
TEXTInfo(
    format='text',
    path='http',
    open=datetime.datetime(2019, 5, 23, 6, 33, 10),
    close=datetime.datetime(2019, 5, 23, 6, 39, 29),
    context=
           dstip method         scrip  src_port
ts                                              url
0   104.16.18.111    GET  128.3.150.243     61272 2017-09-07
00:56:06.628123
http://104.16.18.111/redirect.aspx?linkID=2310...
1  103.227.81.121    GET  202.120.15.33     50025 2017-09-07
00:56:06.722234                                 http://www.cnki.net
2     10.23.12.1   POST  19.19.100.143     62212 2017-09-07
00:56:06.842344                       http://10.23.12.1/upload.html,
    exit_with_error=False
)
```

如日志为 JSON 格式，则返回的日志对象为 JSONInfo 对象。其中，两种对象的 context 成员都是 pandas.DataFrame 数据对象，以便于进行较为复杂的交叉分析，如上文中交叉分析 conn.log 和 files.log 得到提取文件的连接信息等。

4.2.2 基于 Bro 的事件处理脚本

在 BroAPT 系统中，用户可自定义 Bro 脚本，BroAPT-Core 解析模块在运行 Bro 系统对网络流量 PCAP 文件进行解析时，将加载这些脚本，从而提供拓展分析。在运行时，系统会将用户自定义的 Bro 脚本映射至核心模块中。假设 BROAPT_LOAD_MIME 环境变量设置如上文所述，则文件树结构如下：

```
/broapt/scripts/
|   # load FileExtraction module
├── __load__.bro
|   # configurations
├── config.bro
|   # MIME-extension mappings
├── file-extensions.bro
|   # protocol hooks
├── hooks/
|   |   # extract DTLS
|   ├── extract-dtls.bro
|   |   # extract FTP_DATA
|   ├── extract-ftp.bro
```

```
14  │   │   # extract HTTP
15  │   ├── extract-http.bro
16  │   │   # extract IRC_DATA
17  │   ├── extract-irc.bro
18  │   │   # extract SMTP
19  │   └── extract-smtp.bro
20  │   # core logic
21  ├── main.bro
22  │   # MIME hooks
23  ├── plugins/
24  │   │   # extract all files
25  │   ├── extract-all-files.bro
26  │   │   # extract APK
27  │   ├── extract-application-vnd-android-package-archive.bro
28  │   │   # extract PDF
29  │   ├── extract-application-pdf.bro
30  │   │   # extract PE
31  │   ├── extract-application-vnd-microsoft-portable-executable.bro
32  │   │   # extract by BRO_MIME
33  │   └── extract-white-list.bro
34  │   # site functions by user
35  └── sites/
36      │   # load site functions
37      ├── __load__.bro
38      └── ...
```

其中，extract-application-vnd-android-package-archive.bro、extract-application-pdf.bro 和 extract-application-vnd-microsoft-portable-executable.bro 中是根据 BROAPT_LOAD_MIME 环境变量分别生成的提取谓词函数；/broapt/scripts/sites/文件夹是从宿主机中，映射进入 Docker 容器的用户自定义 Bro 脚本路径。用户在/broapt/scripts/sites/__load__.bro 文件中，载入（@load）自定义的 Bro 脚本，在系统运行时便会自动加载并运行。目前，系统中已引入了六类 Bro 脚本，对系统的功能进行了拓展。

(1) 常用协议的常量定义和声明

在 Bro 入侵检测系统中，有大量系统预设的常量，如 FTP 协议的命令（command）列表，HTTP 协议的请求方法（method）列表等。BroAPT 系统通过爬虫技术，从 IANA 的数据库中获取了相关数据，并生成或更新相应的 Bro 常量，如 HTTP 协议请求头字段的注册表 HTTP::header_names 等。

(2) 记录 HTTP 请求头中的 COOKIE 字段

在针对 HTTP 流量的分析中，HTTP 请求头的 COOKIE 字段是一个重要因素，从中或可提取得到密码等敏感信息，同时也是伪造和重放请求的重要依据。脚本从 HTTP 连接的

流量中提取得到请求头，即 http_header 事件，并将请求头中的 COOKIE 字段添加到 http.log 的记录对象 HTTP::Info 中。从而在 Bro 日志系统生成的 http.log 日志文件中，将含有 COOKIE 字段的信息。

(3) 记录 HTTP 请求头中的自定义字段

根据 RFC2616[32]与 RFC7230[33]的定义以及 IANA 的注册声明[34]，HTTP 请求头中的字段是具有注册认证的。但同时，在 HTTP 协议的实现过程中，也可根据需要自行定义字段，而这些字段也可能包含关于 HTTP 流量的重要信息。因此与上一脚本相似，通过 http_header 事件，脚本从请求头中提取出自定义的字段，即不在上述 HTTP::header_names 列表中注册的字段，并将其记录在 http.log 日志文件中。

(4) 记录 HTTP 连接中 POST 请求的数据

POST 是 HTTP 请求命令之一，用于向目标服务器上传数据。RFC2616[32]在对 HTTP 流量进行分析时，通过对 POST 命令中传输的数据进行分析，了解到网络中上载的数据；结合从 HTTP 响应中得到的下载数据，从而可对 HTTP 流量进行深入的分析。因此如上文所述，http_entity_data 事件，即 HTTP 连接中传输的协议包实体数据，脚本选择将 POST 命令传输的数据内容取出，并记录在 http.log 日志文件中。

(5) 计算流量中所有文件的哈希值

在对恶意文件检测时，常需要通过其哈希值与安全厂商提供的样本值进行匹配。因此，在对网络流量中的文件进行分析时，通过 file_new 事件，即 Bro 系统在流量中嗅探发现新文件，可使得 Bro 系统在完成对该文件的分析后计算其哈希值，并记录在 files.log 中。从而，用户可通过这些日志文件，对网络流量中传输的文件进行上述的安全检查。

(6) 基于 Bro 的 SMTP 钓鱼邮件分析

由于 SMTP 流量中传输的文件较为零散，无法直接从提取得到的文件获取钓鱼邮件（phishing）信息。因此，对流量的直接分析是检测 SMTP 钓鱼邮件的有效方案。因此，在 BroAPT 系统中，引入了基于 Bro 的分析脚本。

A. Phishing 模块[35]

该模块主要实现了以下功能：检测向大量邮箱发送的相同邮件及附件，可配置邮箱数量的阈值和监测附件的类型等；计算邮件发送方地址之间的莱文斯坦距离（Levenshtein distance）[36]，从而判断是否为大规模发送的钓鱼邮件。最终，模块生成 phishing_link.bro 日志文件，记录这些恶意的连接和其中传输的 URL 链接等信息。

B. Phish 模块[37]

该模块基于 Ho G.等人对在企业设置中检测凭证性网络钓鱼攻击的研究[38]，主要实现了以下功能：从邮件中提取 URL，并将其记录在 smtpurl_links.log 日志文件中；使用 HTTP 访问这些链接，并检查这些链接是否被点击访问过，即产生了相关流量，如是则记录在 smtp_clicked_urls.log 日志文件中；根据特征数据库中的记录，检查流量中是否存在与之匹配的 SMTP 流量，如有则通过 Bro 系统产生报警；如在上述 URL 中发现可疑字符串，则产生报警；如上述 URL 被点击，并产生了文件下载的流量，则产生报警。

4.2.3 基于日志的交叉分析函数

在 BroAPT 系统中，用户还可自定义对日志文件的分析函数，BroAPT-Core 解析模块在完成 Bro 系统对网络流量 PCAP 文件的解析后，将调用这些分析函数，从而对 Bro 系统生成的日志做进一步交叉分析。在运行时，系统会将用户自定义的 Python 分析函数映射至核心模块中。文件树结构如下：

```
/broapt/python/
│   # setup PYTHONPATH
├── __init__.py
│   # entry point
├── __main__.py
│   # config parser
├── cfgparser.py
│   # Bro script composer
├── compose.py
│   # global constants
├── const.py
│   # Bro log parser
├── logparser.py
│   # BroAPT-Core logic
├── process.py
│   # multiprocessing support
├── remote.py
│   # BroAPT-App logic
├── scan.py
│   # Python hooks
├── sites
│   │   # register hooks
│   ├── __init__.py
│   └── ...
│   # utility functions
└── utils.py
```

其中，`/broapt/python/sites/`文件夹是从宿主机中，映射进入 Docker 容器的用户自定义 Bro 脚本路径。用户在`/broapt/python/sites/__init__.py`文件中，载入（import）自定义的分析函数，并添加至 `HOOK` 和 `EXIT` 列表中，在系统运行时便会运行。`HOOK` 列表中，是用户自定义的分析函数，将在每一个网络流量 PCAP 文件处理完成后进行调用，对 Bro 系统生成的日志文件进行分析；`EXIT` 列表中，则是对上述分析函数的清理函数，在主进程退出前调用，完成对上述分析函数的清理收尾工作。目前，系统中已引入了两组分析函数，对系统的功能进行了拓展。其中一组为上文中所提及的对提取文件信息进行处理，交叉分析 `files.log` 和 `conn.log`，并生成如表 4-1 所示日志文件的分析函数

当前系统中引入的另一组分析函数 `http_parser.generate` 主要针对 HTTP 日志，即 `http.log` 的分析。结合上文中描述的对 `http.log` 进行拓展的 Bro 脚本，分析函数可从中提取出如下表所示的信息，并生成一新日志文件，文件格式参照 Bro 日志系统。该文件中，将包含所有已处理的网络流量 PCAP 文件中的 HTTP 流量信息，从而可用于进一步的大数据分析。在系统退出前，将调用清理函数 `http_parser.close`。该函数将在新生成的 HTTP 流量日志中加入系统退出时间。

表 4-3 新 HTTP 流量日志的描述

字段	描述
srcip	用户所在客户端 IP
ts	用户请求的时间戳（毫秒）
url	用户请求的当前页面 URL
ref	用户请求的来源页面 URL（以 base64 编码）
ua	用户浏览器的 User-Agent（以 base64 编码）
dstip	用户访问的服务器 IP
cookie	用户 Cookie（以 base64 编码）
src_port	用户客户端源端口
json	自定义的 header
method	HTTP 请求命令（method）
body	POST 请求传输的数据（以 base64 编码）

4.3 本章小结

本章对 BroAPT 系统的核心模块之一 BroAPT-Core 提取模块做了详细介绍。BroAPT-Core 提取模块主要实现了基于 Bro 的文件提取与日志分析，其模块结构共可分为三个阶段，如图 4-1 所示。此外，本章对该模块的实现进行了说明，大体上可分为基于 Bro 脚本的事件处理与基于 Python 接口的日志分析。

同时，本章还对 BroAPT-Core 提取模块的拓展性设计做了详细介绍。模块可通过环境变量对系统的运行参数进行调整，例如配置系统提取文件的 MIME 类型、系统运行时的并行进程数等；可通过引入加载用户编写的 Bro 脚本实现对 Bro 系统运行时功能的拓展，并介绍了当前系统中引入的六类 Bro 脚本；还可通过注册日志文件的分析函数，实现对 Bro 日志的自定义处理，并介绍了当前系统中引入的 `http_parser.generate` 分析函数。

第五章　基于 API 的针对性文件检测

5.1　BroAPT-App 检测模块

如前文所述，BroAPT 系统是基于 Bro 的 APT 监测系统，其核心模块由 BroAPT-Core 提取模块和 BroAPT-App 检测模块构成。其中，BroAPT-App 检测模块基于 API 配置文件，着手于对从流量中提取得到的文件进行检测，生成恶意文件检测报告。

5.1.1　模块架构与实现

BroAPT-App 检测模块是 BroAPT 系统的核心模块之一，主要功能为基于 API 的针对性恶意文件检测。模块大致可分为三个阶段，如图 5-1 所示。

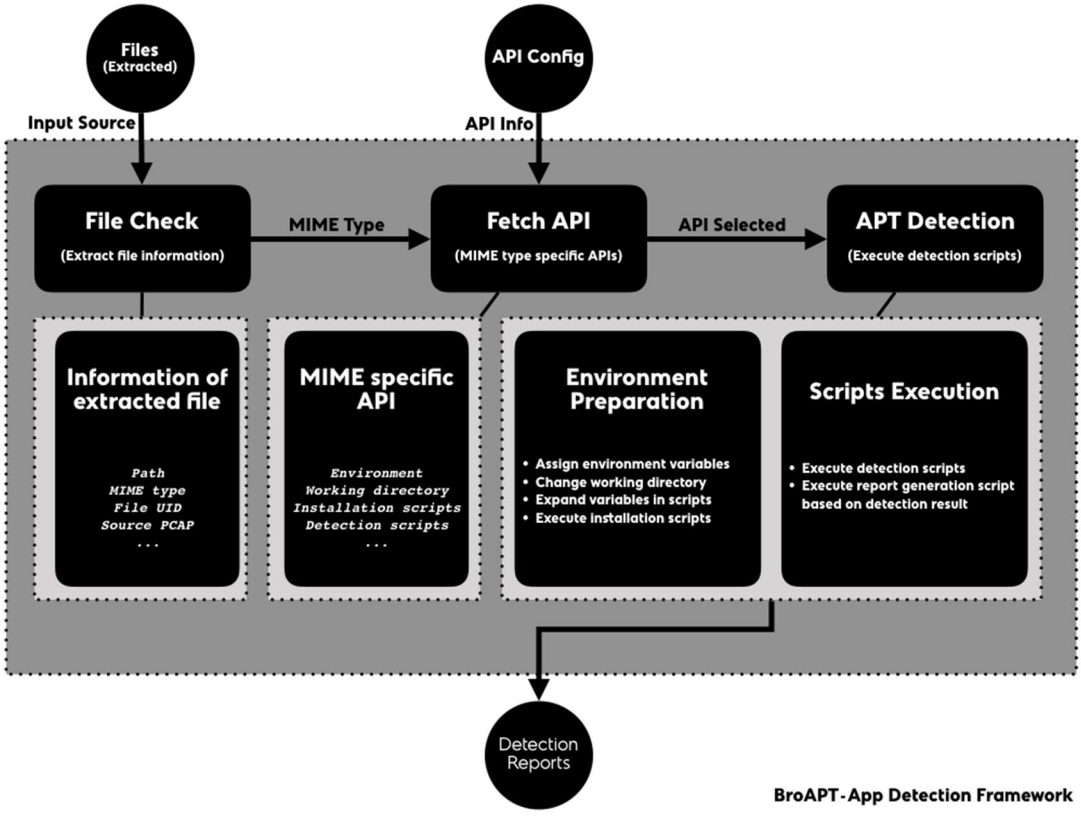

图 5-1 BroAPT-App 检测模块架构

如本文在第三章中所介绍的，BroAPT 系统采用了多进程的运行逻辑架构。BroAPT-Core 提取模块通过同步队列（queue），向 BroAPT-App 检测模块传递提取文件的信息。因此，每当模块从同步队列中，获得提取文件的信息，其首先对该文件进行检查，并获取文件的 MIME 类型、UID 等信息。随后，根据文件的 MIME 类型，模块从 API 配置文件中选择对应的检测 API 配置。最后，模块执行 API 配置中的检测命令进行检测并生成报告。

BroAPT 系统在提取文件时，会以

PROTOCOL-FUID-MIMETYPE.EXT

的格式命名。从而，BroAPT-App 检测模块可从提取文件名中，得到该文件在 Bro 系统中所使用的 UID、传输该文件的协议类型、libmagic 识别该文件的 MIME 类型等信息。以文件 applicaion/vnd.openxmlformats-officedocument/HTTP-F3Df5B3z9UI3yi5J03.application.msword.docx 为例，这些信息在系统中的对象表述（repr）如下：

```
Entry(
    path='applicaion/vnd.openxmlformats-officedocument/HTTP-F3Df5B3z9UI3yi5J03.application.msword.docx',
    uuid='F3Df5B3z9UI3yi5J03',
    mime=MIME(
        media_type='application',
        subtype='msword',
        name='application/msword'
    )
)
```

而根据 MIME 类型，模块从 API 配置文件中获取检测 API 配置，配置中将指定工作路径、环境变量以及安装脚本和检测命令等。随后，在对文件进行检测时，模块首先配置检测环境——赋值环境变量，调整工作路径，展开脚本中的变量名称，并执行安装脚本——而后，执行检测脚本，对文件进行检测，最终执行报告脚本生成检测报告。

在执行脚本时，由于系统是以多进程架构并行的，为避免安装脚本的重复执行，模块采用了共享内存的方式为每一种 API 配置文件设置了已安装标识位。如已置位，则略过安装步骤；反之，则在开始执行安装脚本时，获取进程锁，避免安装脚本在多个进程中并行执行。此外，考虑到脚本可能在运行过程中受到环境影响，如网络通信中断等，如脚本执行失败，即执行进程退出时返回值非零，则将该脚本重复执行一定次数，直至该脚本执行成功，或全部失败；若为后者，则模块将输出报错信息，并中止后续逻辑，放弃对该文件的检测。

5.1.2 客户端-服务器检测架构

由于 BroAPT 系统的核心模块在 Docker 容器中运行，其运行权限是受限状态，而某些检测工具则可能需要在完整权限下运行，如使用 Docker 镜像或虚拟机等。因此，BroAPT-App 检测模块设计了客户端-服务器检测架构，即通过 API 配置文件的配置，选择在 Docker 容器内进行检测或在宿主机环境中进行检测，如图 5-2 所示。

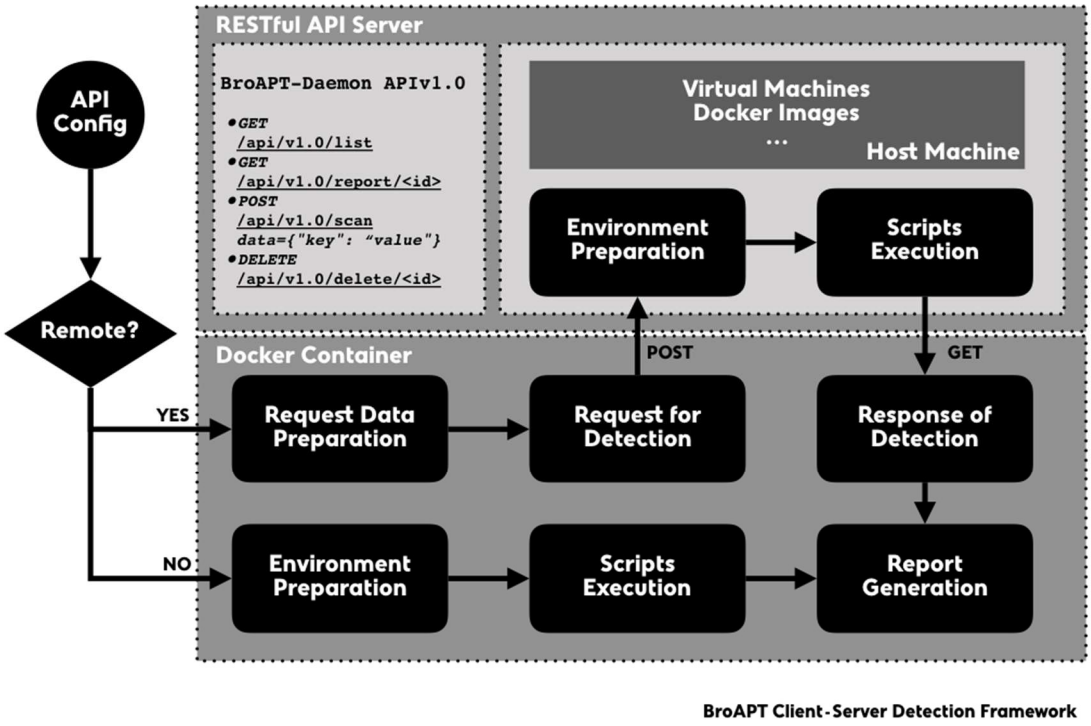

图 5-2 BroAPT 客户端-服务器检测架构

系统实现了 BroAPT-Dameon 系统服务进程。该服务进程是基于 `Flask` 框架的 REST 服务器，主要实现了对从网络流量中提取的文件进行异常文件检测。BroAPT-App 检测模块将检测请求发送给服务进程后，服务进程根据请求配置环境，即——赋值环境变量，调整工作路径，展开脚本中的变量名称，并执行安装脚本——而后，执行检测脚本，对文件进行检测，并向 BroAPT-App 检测模块返回检测结果，生成检测报告。此外，服务器还设计了获取检测结果、枚列检测运行状态和删除检测记录等 API 接口，如下表所示。

表 5-1 BroAPT 系统生成的提取文件信息日志

HTTP 请求方法	URI 地址	描述
GET	/api/v1.0/list	查询检测状态列表
GET	/api/v1.0/report/<id>	查询检测结果
POST	/api/v1.0/scan data={"key": "value"}	请求恶意文件检测
DELETE	/api/v1.0/delete/<id>	删除检测记录

5.2 基于 MIME 类型的模块配置与拓展性

如前文所强调的，BroAPT 系统是一个具有较强可拓展性的 APT 检测系统。本文在第四章中介绍了 BroAPT-Core 提取模块的可拓展性设计，以下将对系统的另一核心模块，BroAPT-App 检测模块的可拓展性作详细介绍。

与 BroAPT-Core 检测模块相同，在 BroAPT-App 检测模块中，可通过一些环境变量进行配置，如下表所示。

表 5-2 环境变量

变量名	描述
BROAPT_SCAN_CPU	并行检测文件进程数
BROAPT_MAX_RETRY	命令最大重试次数
BROAPT_API_ROOT	API 根目录
BROAPT_API_LOGS	API 日志文件路径
SERVER_NAME_HOST	BroAPT-Daemon 服务进程地址
SERVER_NAME_PORT	BroAPT-Daemon 服务进程端口

模块通过 API 配置文件实现了可拓展性。在 API 配置文件中，基于 MIME 类型对恶意文件的检测方案进行了独立的设置。配置文件采用了 YAML 语言，参考了 Docker Compose 和 Travis CI 的配置文件格式，以 MIME 类型为单位进行配置。API 配置文件及其脚本文件的文件树结构如下：

```
/api/
│   # API configuration file
├── api.yml
│   # MIME: application/*
├── application/
│   └── ...
│   # MIME: audio/*
├── audio/
│   └── ...
│   # default API
├── example/
│   └── ...
│   # MIME: font/*
├── font/
│   └── ...
│   # MIME: image/*
├── image/
│   └── ...
│   # MIME: message/*
├── message/
│   └── ...
│   # MIME: model/*
├── model/
│   └── ...
│   # MIME: multipart/*
├── multipart/
```

```
27 |   └── ...
28 |   # MIME: text/*
29 ├── text/
30 |   └── ...
31 |   # MIME: video/*
32 └── video/
33     └── ...
```

在系统运行时，Docker 容器将宿主机文件系统的对应文件夹映射为此处的/api/文件夹，即 API 根路径 BROAPT_API_ROOT。/api/api.yml 是上文所述的 API 配置文件，在/api/*/文件夹中的文件则是以 MIME 类型分类的检测脚本。其中，/api/example/文件夹中的文件是默认的恶意文件检测脚本，即当某一 MIME 类型在配置文件中没有定义检测配置时，系统将调用 example 这一 MIME 类型下配置的检测配置。

为了简化配置文件的编写，系统在 API 配置文件中加入了全局环境变量 environment 字段。在该字段中，以字典形式可设置在 BroAPT-App 检测模块对文件进行检测时，使用的全局环境变量。如下所示的配置中，将 API_ROOT 赋值为系统环境变量 BROAPT_API_ROOT，为常用的 Python 和 Shell 可执行文件设置了环境变量。

```yaml
environment:
  # API root path
  API_ROOT: ${BROAPT_API_ROOT}
  # Python 3.6
  PYTHON36: /usr/bin/python3.6
  # Python 2.7
  PYTHON27: /usr/bin/python
  # Shell/Bash
  SHELL: /bin/bash
```

以针对 PDF 文件（MIME 类型为 application/pdf）的检测为例。假设 API 跟路径 BROAPT_API_ROOT 为/api/，该检测配置的工作路径为./pdf_analysis/，需要设定 ENV_FOO=1 和 ENV_BAR=cliche 两个环境变量，检测脚本使用 Python 语言编写，依赖库在./pdf_analysis/requirements.txt 中描述，则对 application/pdf 这一 MIME 类型的 API 配置段如下所示：

```yaml
application:
  pdf:
    remote: false
    workdir: pdf_analysis
    environ:
      ENV_FOO: 1
      ENV_BAR: cliche
    install:
      - apt-get update
```

```
      - apt-get install -y python python-pip
      - ${PYTHON27} -m pip install -r requirements.txt
      - rm -rf /var/lib/apt/lists/*
      - apt-get remove -y --auto-remove python-pip
      - apt-get clean
    scripts:
      - ${PYTHON27} detect.py [...]
      - ...
    report: ${PYTHON27} report.py
```

在 API 配置文件中，report 字段是必须的，如缺失则模块在解析配置文件时将报错。而如将 remote 字段设置为“真”（true），则表示针对该 MIME 类型文件的检测需在宿主机上通过 BroAPT-Daemon 服务进程完成。此外，考虑到对于不同的 MIME 类型文件可能使用了相同的检测方案，因此在配置文件中系统还加入了 shared 字段，从而在系统运行时，将在这些 MIME 类型的 API 对象中使用相同的已安装置位和进程锁。

通过系统配置文件解析接口 fgparser.parse 处理后，上述 API 配置信息可表述（repr）如下：

```
API(
    workdir='pdf_analysis',
    environ={
        'API_ROOT': '${BROAPT_API_ROOT}',
        'PYTHON36': '/usr/bin/python3.6',
        'PYTHON27': '/usr/bin/python',
        'SHELL': '/bin/bash',
        'ENV_FOO': '1',
        'ENV_BAR': 'cliche'
    },
    install=[
        'apt-get update',
        'apt-get install -y python python-pip',
        '${PYTHON27} -m pip install -r requirements.txt',
        'rm -rf /var/lib/apt/lists/*',
        'apt-get remove -y --auto-remove python-pip',
        'apt-get clean'
    ],
    scripts=[
        '${PYTHON27} detect.py [...]',
        ...
    ],
    report='${PYTHON27} report.py',
    remote=False,
    shared='application/pdf',
```

```
    inited=<Synchronized wrapper for c_ubyte(0)>,
    locked=<Lock(owner=unknown)>
)
```

其中，API.inited 是安装标识位，通过共享内存在多个检测子进程间同步当前环境中是否已完成对该 MIME 类型检测方案的安装；API.locked 是共享进程锁，用于避免重复安装和资源竞争；API.shared 是共享标识位，如 API 配置文件中未设置，则默认为其 MIME 类型。需要指出的是，对于具有相同 API.shared 的 API 配置对象，其 API.inited 和 API.shared 也将分别是相同，从而系统可实现对检测子进程的调控。

假设待检测文件路径为/dump/application/pdf/test.pdf，则在模块对上述文件进行检测时，模块的工作流程如下：

(1) 设置以下环境变量：

```
API_ROOT="/api/"
PYTHON36="/usr/bin/python3.6"
PYTHON27="/usr/bin/python"
SHELL="/bin/bash"
ENV_FOO=1
ENV_BAR="cliche"
BROAPT_PATH="/dump/application/pdf/test.pdf"
BROAPT_MIME="application/pdf"
```

其中，BROAPT_PATH 与 BROAPT_MIME 是系统向检测命令传递的参数。

(2) 将当前工作路径更换至/api/application/pdf/pdf_analysis/。

(3) 若 API 配置中的安装标识位尚未置位，即此时尚未执行过安装脚本，则向系统获取共享进程锁，并执行以下命令：

```
apt-get update
apt-get install -y python python-pip
python -m pip install -r requirements.txt
rm -rf /var/lib/apt/lists/*
apt-get remove -y --auto-remove python-pip
apt-get clean
```

完成上述命令后，将安装标识位置位，随后解除进程锁。

(4) 执行如下检测命令：

```
/usr/bin/python detect.py [...]
...
```

(5) 完成上述所有步骤后，执行日志生成脚本/usr/bin/python report.py。

目前，系统中已引入了六种检测方案，对系统的功能进行了拓展。以下将对这些方案进行简要介绍。

5.2.1 默认检测方案

如上文所述，`example` 这一 MIME 类型下的 API 检测方案为默认检测方案。当针对某 MIME 类型的检测方案尚未在 API 配置文件中配置时，BroAPT-App 检测模块将调用默认检测方案对文件进行检测。

当前默认检测方案是使用 VirusTotal 的恶意文件扫描 API，对文件进行检测。VirusTotal 是谷歌公司旗下一款免费提供针对文件和 URL 的蠕虫、木马、僵尸网络等各种安全威胁检测的安全工具。VirusTotal 综合了 Baidu-International、Google Safebrowsing、Tecent 等 67 个知名检测平台的检测结果，以给出最终的判断，在检测结果上具有很强的可靠性。配置代码段如下：

```
example:
  environ:
    ## sleep interval
    VT_INTERVAL: 30
    ## max retry for report
    VT_RETRY: 10
    ## percentage of positive threshold
    VT_PERCENT: 50
    ## VT API key
    VT_API: ...
    ## path to VT file scan reports
    VT_LOG: /var/log/bro/tmp/
  report: ${PYTHON36} virustotal.py
```

其中，使用 VirusTotal 的 API 时需要使用其专用口令（token），因此需在环境变量或代码中设置。由于 VirusTotal 对检测频率和访问频率有一定限制，因此可通过设置 `VT_INTERVAL` 环境变量和 `VT_RETRY` 环境变量进行一定程度上的频率调控和访问重试。在 VirusTotal 返回的检测结果中，包括总体检测结果和 67 个工具单独检测结果等在内的信息，从而可通过 `VT_PERCENT` 环境变量调整认定为该文件为恶意文件的比例阈值。此外，还可通过 `VT_LOG` 指定检测命令运行日志的存储路径。

5.2.2 对 Andriod 应用 APK 文件的检测方案

APK 文件在 IANA 注册表中对应的 MIME 类型为 `application/vnd.android.package-archive`。系统选用了 AndroPyTool 作为对 APK 文件的检测工具。AndroPyTool 是基于对 APK 文件的动态和静态特征进行分析，检测是否为恶意应用的工具，集合了如 DroidBox、FlowDroid、Strace、AndroGuard 和 VirusTotal 等在内的多种 APK 文件检测工具[39][40]。配置代码段如下：

```
application:
  vnd.android.package-archive:
    remote: true
    workdir: AndroPyTool
    environ:
      APK_LOG: /home/traffic/log/bro/tmp/
    install:
      - docker pull alexmyg/andropytool
    report: ${SHELL} detect.sh
```

由于 AndroPyTool 的环境配置过于复杂，因此本系统直接使用了其 Docker 镜像。BroAPT-App 检测模块在对 APK 文件进行检测时，将根据上述检测配置生成检测请求，并发送至 BroAPT-Daemon 服务进程，由其使用 AndroPyTool 的 Docker 镜像进行检测后，生成检测结果返回至模块中，得到检测报告。

5.2.3 对 Office 文档的检测方案

Office 文档是一种基于 XML（extensible markup language，可拓展描述语言）开发的文档，主要有 Microsoft Office 和 OpenOffice 两类，其 MIME 类型主要有 `application/msword`、`application/ms-excel`、`application/vnd.ms-powerpoint` 和 `application/vnd.openxmlformats-officedocument.*`等。系统选用了 MaliciousMacroBot 作为对该类文件的检测工具。MaliciousMacroBot 是基于机器学习（machine learning）算法对 Office 文档中的恶意宏（macro）进行检测的工具，其使用到了随机森林（random forest）和 TF-IDF（term frequency－inverse document frequency）算法[41]等。配置代码段如下：

```
application:
  vnd.openxmlformats-officedocument: &officedocument
    workdir: ${API_ROOT}/application/vnd.openxmlformats-
officedocument/
    environ:
    MMB_LOG: /var/log/bro/tmp/
    install:
    - yum install -y git
    - git clone https://github.com/egaus/MaliciousMacroBot.git
    - ${PYTHON36} -m pip install ./MaliciousMacroBot/
    - yum clean -y all
    report: ${PYTHON36} MaliciousMacroBot-detect.py
    shared: officedocument
  msword: *officedocument
  vnd.ms-excel: *officedocument
  vnd.ms-powerpoint: *officedocument
  ...
```

由于 MaliciousMacroBot 可对多种 MIME 类型的文件进行检测，因此在配置文件中使用 YAML 语言的引用语法&和*。此外，为避免重复安装和资源冲突，配置文件中将 shared 字段进行了置位，从而系统在对不同 MIME 类型的 Office 文档进行检测时，BroAPT-App 模块将能够通过这一字段对相关进程进行规约和管理。

5.2.4 对 Linux 系统 ELF 可执行文件的检测方案

ELF（可执行与可链接格式）可执行文件是类 UNIX 操作系统中的二进制文件标准格式，其 MIME 类型为 application/x-executable。系统选用了 ELF Parser 作为对该类文件的检测工具。ELF Parser 主要实现了以下功能：通过对 ELF 文件中的函数和签名进行分类，分析其数据结构，如符号表（symbol table）、动态段（dynamic segment）等，了解其功能，并尝试对已知的恶意应用，如 Kaiten、Elfknot 和 BillGates 等进行检测，最终对该文件进行打分[42]。配置代码段如下：

```
application:
  x-executable:
    ## ELF Parser
    remote: true
    environ:
      ELF_LOG: /home/traffic/log/bro/tmp/
      ELF_SCORE: 100
    workdir: ELF-Parser
    install:
      - docker build --tag elfparser:1.4.0 --rm .
    report: ${SHELL} detect.sh
```

5.2.5 对常见 Linux 恶意文件的检测方案

LMD（Linux Malware Detect）是一个基于签名匹配的恶意文件检测工具，其主要实现了对文件系统的自动扫描以及清理等功能，并支持对 HTML 钓鱼网页，Perl 与 PHP 恶意脚本，以及常见二进制恶意文件的检测[43] 。配置代码段如下：

```
application:
  octet-stream: &lmd
    ## LMD
    workdir: ${API_ROOT}/application/octet-stream/LMD
    environ:
      LMD_LOG: /var/log/bro/tmp/
    install:
      - yum install -y git which
      - test -d ./linux-malware-detect/ ||
        git clone https://github.com/rfxn/linux-malware-detect.git
      - ${SHELL} install.sh
    report: ${SHELL} detect.sh
    shared: linux-maldet
```

```
text:
  html: *lmd
  x-c: *lmd
  x-perl: *lmd
  x-php: *lmd
```

5.2.6 对恶意 JavaScript 脚本的检测方案

JavaScript 是常用的 web 语言，在攻击中常可用于窃取用户 cookie、伪造钓鱼页面、向页面中植入广告，甚至可利用操作系统或浏览器漏洞对系统进行攻击。系统采用了 JaSt 作为对该类文件的检测工具。JaSt 是一个基于语法分析对混淆的恶意 JavaScript 代码进行检测的工具，通过机器学习算法实现了对 JavaScript 代码的高精度检测[44]。配置代码段如下：

```
application:
  javascript: &javascript
    workdir: ${API_ROOT}/application/javascript/JaSt
    environ:
      JS_LOG: /var/log/bro/tmp/
    install:
      - yum install -y epel-release
      - yum install -y git nodejs
      - test -d ./JaSt/ ||
        git clone https://github.com/Aurore54F/JaSt.git
      - ${PYTHON3} -m pip install
          matplotlib
          plotly
          numpy
          scipy
          scikit-learn
          pandas
      - ${PYTHON3} ./JaSt/clustering/learner.py
          --d ./sample/
          --l ./lables/
          --md ./models/
          --mn broapt-jast
    scripts:
    - ${PYTHON3} ./JaSt/clustering/classifier.py
        --f ${BROAPT_PATH}
        --m ./models/broapt-jast
    report: ${PYTHON3} detect.py
    shared: javascript
text:
  javascript: *javascript
```

5.3 本章小结

本章对 BroAPT 系统的核心模块之一 BroAPT-App 检测模块做了详细介绍。BroAPT-App 检测模块主要实现了基于 API 的针对性文件检测，其模块结构共可分为三个阶段，如图 5-1 所示。此外，本章对该模块的实现进行了说明，介绍了基于 BroAPT-Daemon 服务进程的客户端-服务器远程检测模式，如图 5-2 所示。

同时，本章还对 BroAPT-App 检测模块的拓展型设计做了详细介绍。模块可通过环境变量对系统的运行参数进行调整，例如配置服务进程地址信息、检测文件时并行进程数等。同时，本章还对系统所使用的 API 配置文件进行了详细介绍，并对其实现特点和技术难点进行了描述。此外，本章对当前系统 API 配置文件中所使用的六种检测方案进行了简要介绍。在下一章中，本文将对 BroAPT 系统的性能和拓展功能进行测试。

第六章　系统评测

6.1　实验环境及配置

本文中的实验均在 Linux 服务器上完成。实验中所使用的流量来自于对某学校网关处流量的镜像，为上述网关处的实时流量，其日行流量变化趋势如下图所示：

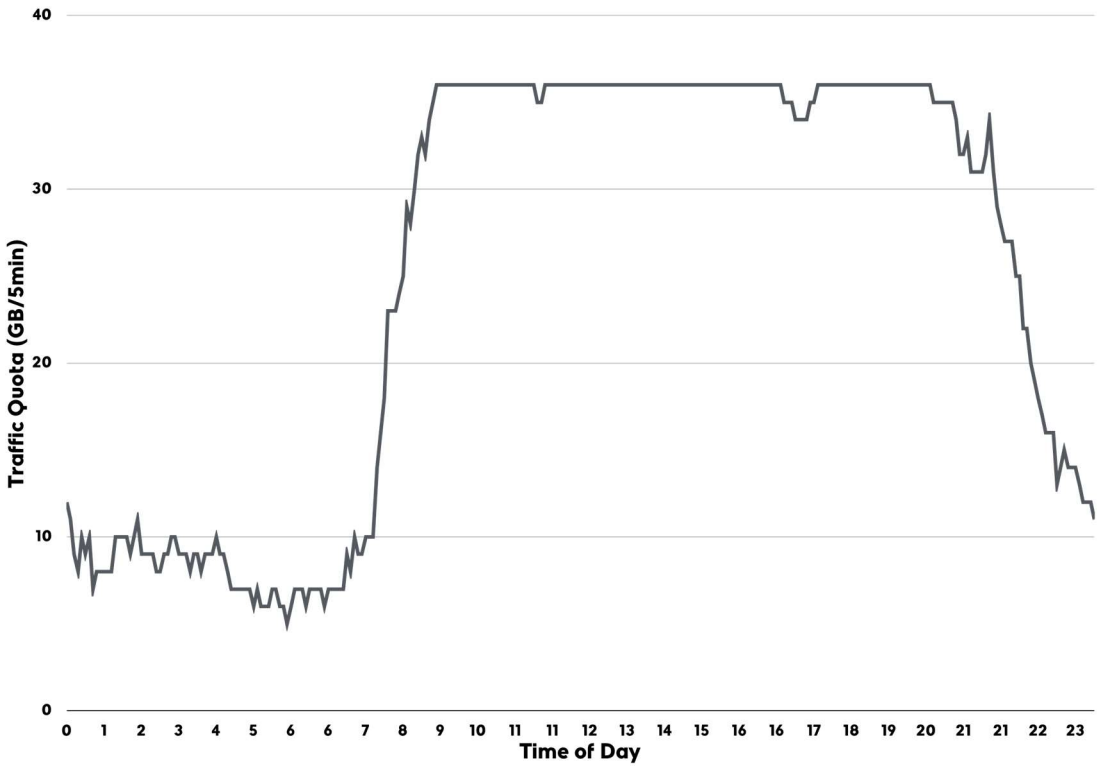

图 6-1 流量变化趋势图

服务器的软硬件配置如下：

- 操作系统：CentOS 7
- 内核版本：Linux 3.10.0
- 内存：252G
- CPU：40 核
- Docker 版本：17.05.0
- Docker Compose 版本：1.23.2

而系统核心模块在 Docker 环境中运行，本文采用了 CentOS 操作系统作为系统核心模块的底层镜像，其它软硬件配置如下：

- 操作系统：CentOS 7
- 内核版本：Linux 3.10.0
- Python：3.6.8
- Bro：2.6.1
- C/C++库：
  - `libmagic`：5.11（探测文件的 MIME 类型信息）
  - `libyaml`：0.1.4（对 YAML 语言的支持）
- Python 依赖：
  - `dataclasses`：0.6（Python 3.7 标准库 `dataclasses` 的后向支持）
  - `file-magic`：0.4.0（对 `libmagic` 的 Python 封装）
  - `pandas`：0.24.2（Python 数据分析库）
  - `PyYAML`：5.1（对 `libyaml` 的 Python 封装）
  - `requests`：2.22.0（HTTP 访问库）

而对于 BroAPT-Daemon 服务进程，考虑到在部署环境中安装相关依赖较为复杂，因此本文使用了 `PyInstaller` 对其进行打包，生成无需安装依赖环境的可执行文件。而由于 `PyInstaller` 在打包时只能生成其当前运行环境下的可执行文件，因此本文使用 CentOS 的 Docker 镜像，对 Linux 环境下的可执行文件进行打包。BroAPT-Daemon 服务进程在开发环境中所使用的相关软硬件环境如下：

- 操作系统：CentOS 7
- 宿主机操作系统：macOS Mojave 10.14.6 Beta
- Docker 版本：18.09.2
- Docker Compose 版本：1.23.2
- Docker 镜像中的软硬件环境：
- 操作系统：CentOS 7
- 内核版本：Linux 4.9.125
- Python：3.6.7
- Python 依赖：
  - `PyInstaller`：3.4（Python 项目打包工具）
  - `Flask`：1.0.2（HTTP 服务器库）
  - `dataclasses`：0.6（Python 3.7 标准库 `dataclasses` 的后向支持）

6.1.1 系统配置

BroAPT 系统提供了环境变量作为参数接口。在实验中，对这些环境变量配置如下：

表 6-1 环境变量配置

变量名	变量值
BROAPT_CPU	5
BROAPT_INTERVAL	10
BROAPT_DUMP_PATH	/dump/
BROAPT_PCAP_PATH	/pcap/
BROAPT_LOGS_PATH	/var/log/bro/

续表 6-1

变量名	变量值
BROAPT_MIME_MODE	true
BROAPT_JSON_MODE	false
BROAPT_BARE_MODE	false
BROAPT_NO_CHKSUM	true
BROAPT_HASH_MD5	true
BROAPT_HASH_SHA1	true
BROAPT_HASH_SHA256	true
BROAPT_X509_MODE	false
BROAPT_ENTROPY_MODE	false
BROAPT_LOAD_MIME	application/vnd.microsoft.portable-executable;application/x-dosexec;application/msword;application/pdf;application/vnd.android.package-archive;message/rfc822
BROAPT_LOAD_PROTOCOL	smtp
BROAPT_FILE_BUFFER	0xffffffffffffffff
BROAPT_SIZE_LIMIT	0
BROAPT_HOOK_CPU	5
BROAPT_SCAN_CPU	10
BROAPT_MAX_RETRY	3
BROAPT_API_ROOT	/api/
BROAPT_API_LOGS	/var/log/bro/api/
SERVER_NAME_HOST	localhost
SERVER_NAME_PORT	5000

其中，BROAPT_LOAD_MIME 环境变量设为提取所有的 Widnows 可执行文件（包括 PE 和 DOS 可执行文件）、Microsoft Word 文档、PDF 文件、Android 应用 APK 文件以及在 RFC0822 中描述的邮件文件；BROAPT_LOAD_PROTOCOL 环境变量设为提取所有 SMTP 协议传输的文件；BROAPT_FILE_BUFFER 环境变量设为 Bro 语言中 count 类型的最大值，即 C/C++中 uint64 类型的最大值 UINT64_MAX；BROAPT_SIZE_LIMIT 环境变量设为重组文件大小不限。

6.1.2 测试文件

在性能测评时，本文采用了一个文件大小为 35G 的网络流量 PCAP 文件。该文件的抓取时间为 2019 年 5 月 20 日中午 12 时，流量窗口为 5 分钟，共有约 41,000,000 个流量包，该文件的其他信息（通过 capinfos 获得）如下表所示：

表 6-2 文件的其他信息

项目	数值
文件格式	Wireshark/tcpdump - PCAP 文件
底层协议	Ethernet（以太网）协议
流量包大小限制	262144 字节
流量包数量	41M
文件大小	45G
窗口时长	300 秒
开始时间	2019 年 5 月 20 日 12 时 01 分 52 秒
结束时间	2019 年 5 月 20 日 12 时 06 分 51 秒
数据带宽	117Mbps
平均流量包大小	849.97 字节
平均抓包速率	每秒 138,000 个
流量包严格时序	否

6.1.3 测试流量

在对系统功能和拓展插件与检测方案进行测评时，本文采用了从上述服务器抓取的 525 个网络流量 PCAP 文件（时间窗口为 5 分钟）。这些流量从 2019 年 5 年 8 日 1 时 1 分 41 秒起，至 2019 年 5 月 9 日 20 时 46 分 40 秒结束；共 1 天 19 小时 45 分钟的流量，总流量大小为 12.14T，共约有 210 亿个流量包。

6.2 功能评测

以上述配置环境下的 BroAPT 系统对这些流量进行处理，系统总耗时约 1 天 2 小时 15 分钟。系统共处理了 525 个网络流量 PCAP 文件，BroAPT-Core 提取模块从中提取了 11,183 个文件，包括 Office 文档、PDF 文件、Windows 可执行文件等多种类型（图 6-2）；经 BroAPT-App 检测模块检测后未发现恶意文件。其中，约有 9,703 个文件由 BroAPT-Daemon 系统服务进程检测。

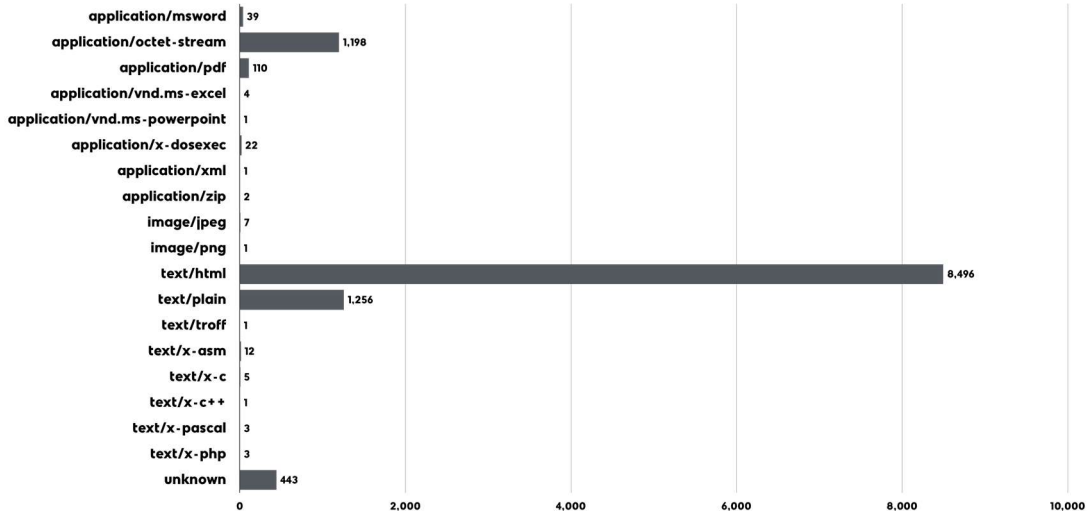

图 6-2 测试流量中提取文件类型数量

通过对 Bro 日志 conn.log 分析可知，流量中共有 2,121,027 次网络连接（图 6-3），其中 19,751 次 HTTP 连接（图 6-4）和 110 次 SMTP 连接（图 6-5）等；此外，流量中共传输了 34.004 个文件（图 6-6）。同时，通过对 SMTP 协议分析的 Bro 拓展脚本，系统还发现了 11 次可疑的钓鱼邮件记录。

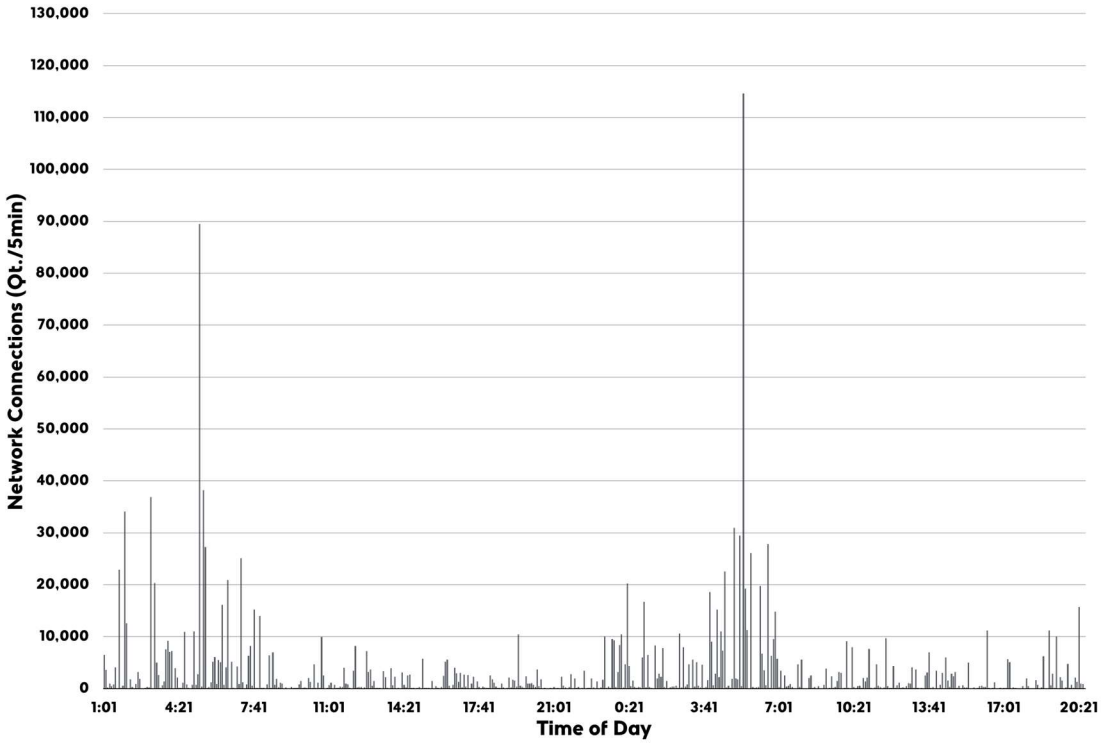

图 6-3 测试流量中网络连接随时间变化

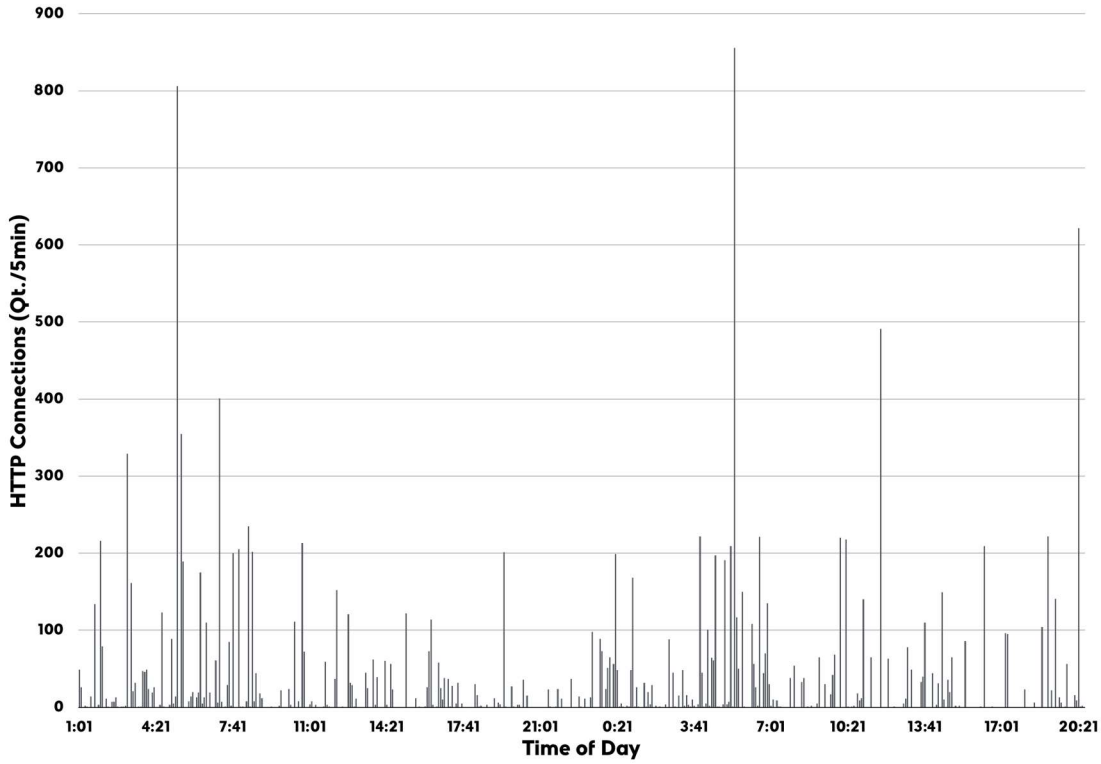

图 6-4 测试流量中 HTTP 连接随时间变化

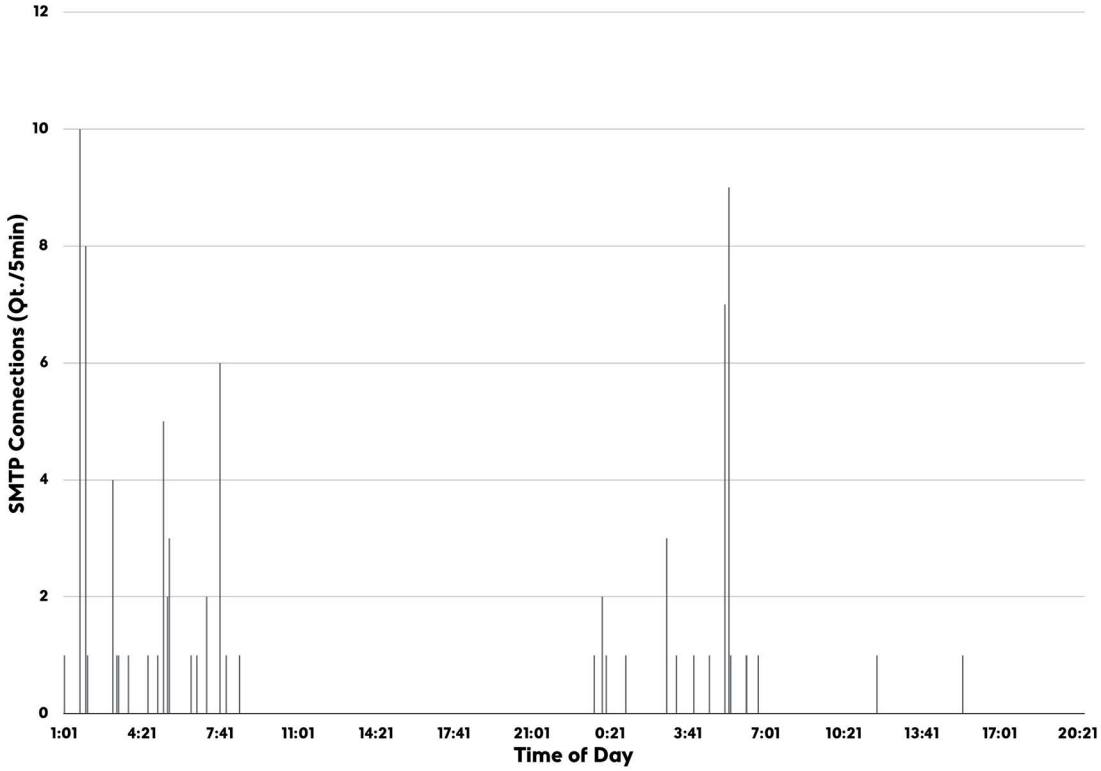

图 6-5 测试流量中 SMTP 连接随时间变化图

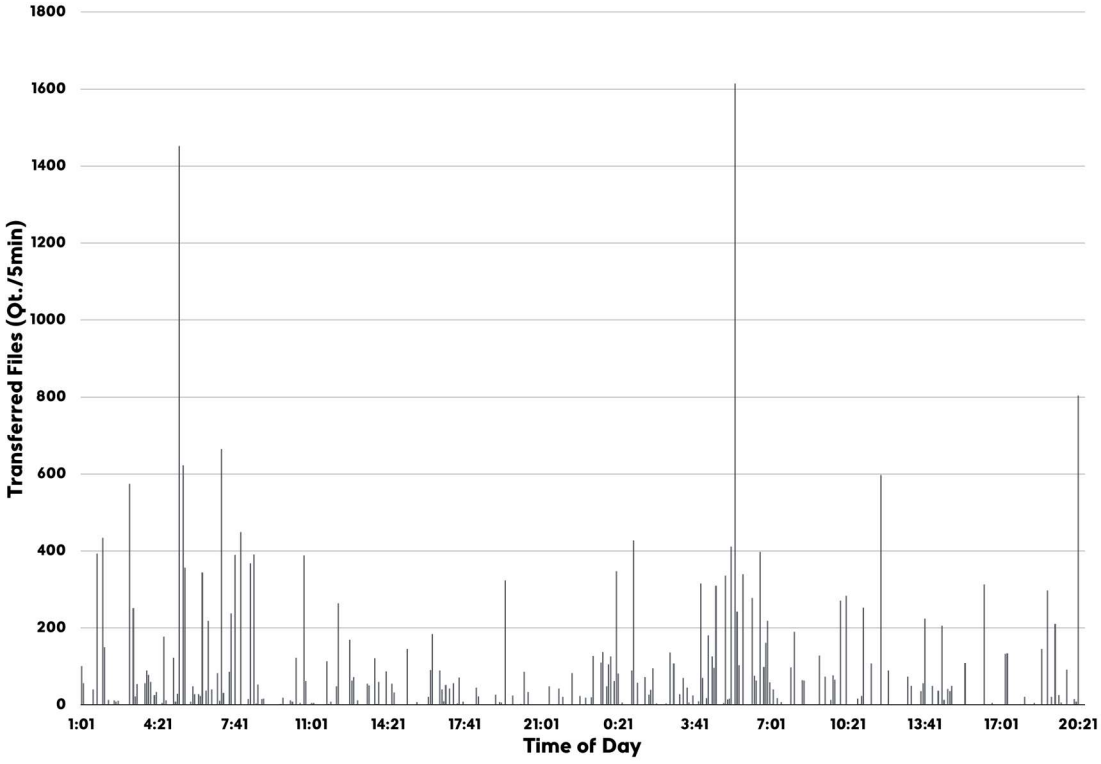

图 6-6 测试流量中传输文件数量随时间变化

6.3 性能评测

如上文所述，实验时所部署的网络环境中流量极大，因此系统需要以较高的效率完成对流量的处理。相较于其他常用系统，BroAPT 系统所使用的 Bro 入侵检测系统在流量处理和文件提取上效率极高。

本文采用 tcpflow（版本为 1.4.5）与 BroAPT 系统核心模块中的 Bro 入侵检测系统进行性能比较。tcpflow 是使用 C/C++语言编写的流量重组工具，其重组算法与第二章中介绍的 TCP 重组算法相似。然而需要指出的是，tcpflow 在完成对 TCP 流量的重组后，并未解析其中的应用层协议并提取文件，因此本文在测试时使用 Python 编写了对 HTTP 协议的解析脚本，使用其对 HTTP 流量中的文件进行了提取。而在实验中，Bro 入侵检测系统使用了如下脚本：

```
1 event file_sniff(f: fa_file, meta: fa_metadata)
2     {
3     Files::add_analyzer(f, Files::ANALYZER_EXTRACT,
  [$extract_filename=fmt("%s-%s", f$source, f$id)]);
4     }
```

即提取流量中出现的所有文件，并以其传输协议（如 HTTP、SMTP 等）与文件 UID 命名。二者运行时间和效果如下表所示：

表 6-3 运行时间和效果比较

项目	Bro	tcpflow 与 Python 文件提取脚本
运行时间	3 分 5 秒	20 分 42 秒
文件提取（总）	102,780	76,959
文件提取（DTLS）	201	0
文件提取（FTP_DATA）	0	0
文件提取（HTTP）	76,959	76,959
文件提取（IRC_DATA）	0	0
文件提取（SMTP）	47	0
文件提取（SSL）	25573	0
日志文件	18	0

通过对上述日志文件 conn.log 的分析，可以得到该 PCAP 文件中包含了 1,733,505 组连接，其中有 769,809 组 TCP 连接（而 tcpflow 仅提取出 332,720 组），950,933 组 UDP 连接，以及 12,763 组 ICMP 连接。此外，对其他日志文件的分析还可得到，该 PCAP 文件中共有 882,157 次 DNS 查询，95,138 次 HTTP 连接，612 次 RDP（remote desktop，远程桌面）连接，90 次 SMTP 连接，9 次 SSH 连接。其中，由 Bro 入侵检测系统检测到 112,684 次文件传输，其提取得到的文件分布于 56 种不同的 MIME 类型。

此外，当对文件提取的 MIME 类型或传输协议进行一定限制时，系统的运行效率将显著提高。以上节中所描述的 BROAPT_LOAD_MIME 和 BROAPT_LOAD_PROTOCOL 环境变量为例，此时 Bro 入侵检测系统只需约 1.35 秒即可完成对上述网络流量 PCAP 文件的处理和提取。

然而，需要指出的是，由于 Bro 系统的性能消耗较大，因此系统的运行效率在一定程序上会受到环境的影响。例如，对于上述 PCAP 文件，在当前实验环境中，Bro 系统只需约 1.35 秒即可完成处理，而在上文所述的开发环境中，则需要至少 11 分 27 秒才能完成处理。

如上文所述，服务器上以 5 分钟为流量窗口，对网络流量进行实时抓取并生成 PCAP 文件。在使用上文描述的配置运行时，BroAPT 系统平均每 0.9 秒完成对一个网络流量 PCAP 文件的处理，平均每天可提取得到 6,241 个文件。其中，约有 61 个 PDF 文件，37 个 Office 文档，14 个 Windows 可执行文件，另有 73 个使用 SMTP 协议传输的邮件及其附件等。

6.4 拓展插件与检测方案评测

如前文所述，BroAPT 系统具有极强的可拓展性。其中，在 BroAPT-Core 提取模块，可通过编写 Bro 脚本，在系统运行 Bro 入侵检测系统对网络流量 PCAP 文件进行处理时，进行额外的分析；还可注册分析函数，在系统完成对 PCAP 文件的处理后，对 Bro 系统生成的日志进行综合分析。此外，在 BroAPT-App 检测模块中，可通过 API 配置文件对不同 MIME 类型的文件配置恶意文件检测方案。当前，系统已集成了六类 Bro 脚本、一个分析函数以及四种检测方案，为 BroAPT 系统的功能做出了极大拓展。以下将对这些功能拓展进行测评。

6.4.1 Bro 脚本评测

本文在第四章中介绍了当前系统中所引入的六类 Bro 脚本，这些脚本主要用于拓展 Bro 系统对 FTP 和 HTTP 协议的检测范围，以及通过 Bro 日志系统生成日志和拓展已有系统日志的信息。从性能上看，对于上述网络流量 PCAP 文件，加载脚本与不加载脚本的运行时间几乎没有差别。然而，加载 Bro 脚本后，系统将生成新的日志文件，为后续的离线分析提供了极大便利。

表 6-4 对上述 PCAP 文件处理后的 Bro 日志文件比较

日志文件	加载 Bro 脚本	不加载 Bro 脚本	备注
conn.log	有	有	TCP/UDP/ICMP 等连接记录
dns.log	有	有	DNS 查询记录
files.log	有	有	文件分析记录
http.log	有	有	HTTP 连接记录，加载 Bro 脚本后添加了对请求头 COOKIE 字段、请求头中自定义字段以及请求方法（method）和 POST 请求所上传的数据等信息
notice.log	有	无	Bro 通知系统报警信息
packet_filter.log	有	有	Bro 系统在 libpcap

			内核中所使用的过滤器

续表 6-4

日志文件	加载 Bro 脚本	不加载 Bro 脚本	备注
smtp_clicked_urls.log	有	无	SMTP 流量中被点击的 URL 信息
reporter.log	有	无	Bro 通知系统日志信息
smtp.log	有	有	SMTP 连接信息
smtpurl_links.log	有	有	SMTP 流量传输信息文本中 URL 信息
ssl.log	有	有	SSL 连接信息
weird.log	有	有	Bro 系统认定可疑流量信息
x509.log	有	有	X509 认证信息

6.4.2 分析函数评测

如本文在第四章中所介绍的，当前系统中仅加入了一个日志分析函数 http_parser.generate，用于从 http.log 中提取信息构成新的日志文件，以便进行后续的大数据处理。对于上节中所使用约 35G 的网络流量 PCAP 文件，共生成 150,506 次 HTTP 连接，而这一函数的运行时间约为 20.04 秒。其中一部分记录如下：

```
1  #separator \x09
2  #set_separator  ,
3  #empty_field    (empty)
4  #unset_field    NoDef
5  #path   http
6  #open   2019-05-23-06-33-10
7  #fields scrip   ts  url ref ua  dstip   cookie  src_port    json
      method  body
8  #types  addr    string  time    string  string  string  addr    s
   tring  port    vector[string]  string  string
9  ...
10 198.128.28.216  1302203839314   http://208.96.29.104/login.php  a
   HR0cDovL2xibC53ZWJkYW1kYi5jb20vbG9naW4ucGhw    TW96aWxsYS81LjAgKE
   1hY2ludG9zaDsgVTsgSW50ZWwgTWFjIE9TIFggMTBfNl83OyBlbi1VUykgQXBwbGV
   XZWJLaXQvNTM0LjE2IChLSFRNTCwgbGlrZSBHZWNrbykgQ2hyb21lLzEwLjAuNjQ4
   LjIwNCBTYWZhcmkvNTM0LjE2    208.96.29.104   UEhQU0VTU0lEPWtuZzVoM
   G1jbXEyNW51MG10YzkwYWxnZDU1    49424   (empty) POST    dXNlcm5hbW
   U9dGVzdGluZzljJnBhc3N3b3JkPXRlc3RpbmcmbWV0aG9kPUxvZ2lu
11 ...
```

```
12  128.3.150.243  1504717710140  http://104.16.19.111/s/1314/images/gid25/editor/emails/data-science/AI-in-Healthcare_92.png NoDef  TW96aWxsYS81LjAgKE1hY2ludG9zaDsgSW50ZWwgTWFjIE9TIFggMTBfMTFfNikgQXBwbGVXZWJLaXQvNjAxLjcuOCAoS0hUTUwsIGxpa2UgR2Vja28p  104.16.19.111  NoDef  61756  CF-BGJ,CF-POLISHED,X-POWERED-BY,X-SERVER,X-XSS-PROTECTION,CF-CACHE-STATUS,CF-RAY  GET NoDef
13  ...
```

6.4.3 检测方案评测

在本文第五章中，详细介绍了系统当前引入的六种检测方案。其中包括：针对 Andriod 应用 APK 文件的检测方案 AndroPyTool，针对 Office 文档的检测方案 MaliciousMacroBot，针对 Linux 系统 ELF 可执行文件的检测方案 ELF Parser，针对常见 Linux 恶意文件的检测方案 LMD，针对 JavaScript 恶意脚本的检测方案 JaSt，以及默认检测方案 VirusTotal。

表 6-6 评测使用的样本集

文件类型	恶意样本数量	良性样本数量	样本总量
APK 文件	22,000	22,000	44,000
Office 文档	100	250	350
ELF 可执行文件	1,000	1,000	2,000
Linux 二进制文件及脚本	3,000	3,000	6,000
JavaScript 脚本	85,059	20,246	105,305

在实验中，本文使用从实际运行过程中收集提取的文件对上述其他检测方案进行评测。其中，由于部分文件类型提取得到的样本较少，因此本文额外加入了一些其它样本集，如 APK 文件样本集 OmniDroid[39]等。本文使用 VirusTotal 对上述样本进行了验证，并作为评测基准（ground truth）——若 VirusTotal 检测结果中，有 1 组报告认为该样本为恶意文件，或 2 组报告认为该样本为可疑文件，则认为该样本为恶意样本。

表 6-7 上述检测方案的准确率

检测方案	针对文件类型	准确率	召回率	F1 分数
AndroPyTool	Android 应用 APK 文件	89.92%	95.11%	92.44%
MaliciousMacroBot	插入恶意宏的 Office 文档	78.40%	91.16%	84.30%
ELF Parser	Linux 系统 ELF 可执行文件	87.50%	71.60%	78.76%
LMD	常见 Linux 恶意文件	85.27%	90.48%	87.80%
JaSt	恶意 JavaScript 脚本	99.43%	87.80%	99.54%

实验结果如表 6-7 及图 6-7 所示。针对 Android 应用 APK 文件的检测方案 AndroPyTool，和针对恶意 JavaScript 脚本的检测方案 JaSt 均有较好的检测效果及准确率；而针对 Linux 系统 ELF 可执行文件的检测方案 ELF Parser 则较差，这可能是由于其对恶意文件的评判标准较为简单，恶意文件的签名数据较为陈旧，故其对恶意文件的检测准确率较低。总体而言，五种检测方案的检测效果符合预期。

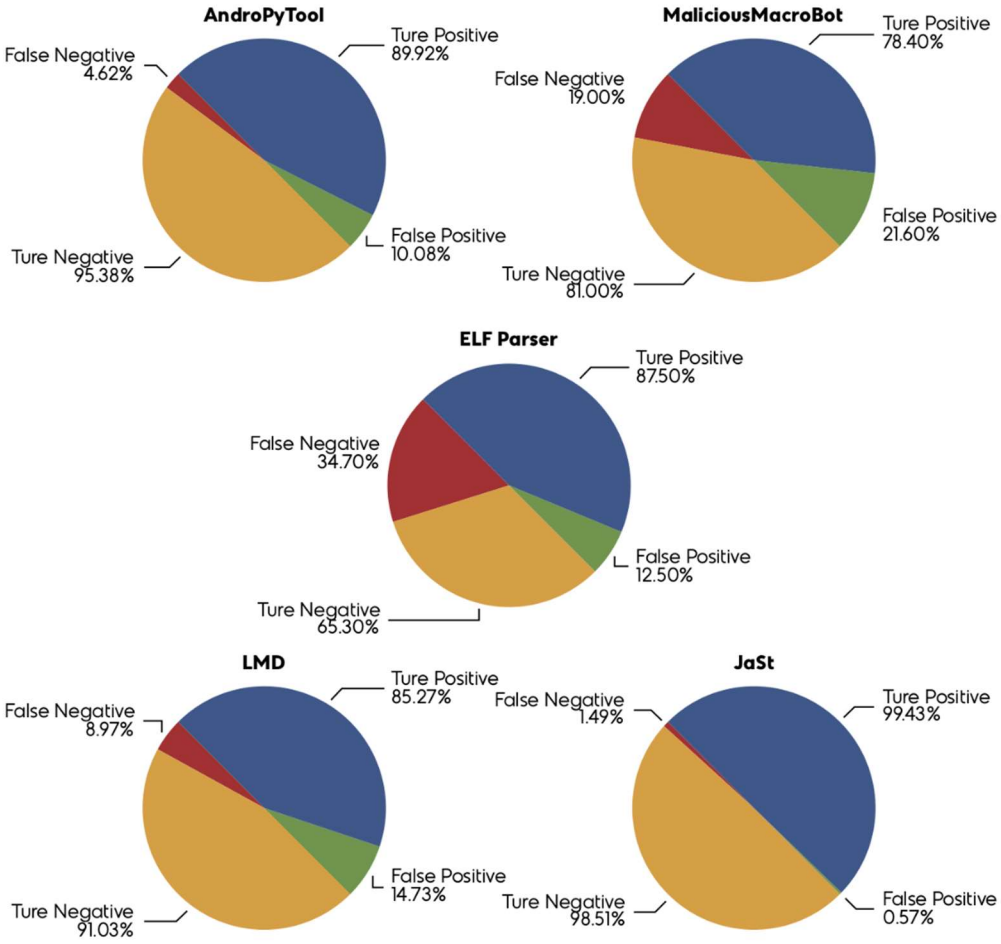

图 6-7 上述检测方案的检测效果

6.5 本章小结

本章简要介绍了对 BroAPT 系统的性能测试和拓展功能测试。首先，对实验环境和软硬件环境进行了简要介绍，并描述了实验中所使用的网络流量 PCAP 文件信息。随后，本章针对从网络流量中重组并提取文件，对比了 Bro 入侵检测系统与 `tcpflow` 工具的性能和效果，得出了 Bro 系统明显优于后者的结论。此外，本章测试了第四章中介绍的 Bro 脚本对系统运行的负载情况，以及日志分析函数的功能，发现前者对系统运行效率几乎没有影响，并能够拓展系统的日志信息，以便离线处理。同时，本章使用 VirusTotal 作为评测基准，对第五章中介绍的三种检测方案进行了测试。在下一章中，本文将对项目实现过程中，所进行的尝试和相关工作做简要分析和介绍。

第七章　相关工作与尝试

在系统的实现过程中，笔者曾做出过许多尝试和调整。本章将对这些相关工作和尝试进行简要介绍。

7.1　TCP 流量重组与文件提取

本项目的目标是对网络中的流量进行重组，并从中提取流量传输的文件，因此在实现上存在多种方案。考虑到实现难度和系统复杂度，本项目尝试了以下两种不同的实现策略。

7.1.1　基于 PyPCAPKit 的实现

PyPCAPKit 是一个多内核的网络流量 PCAP 文件分析工具库，由笔者在业余时间开发。其中实现了对 PCAP 文件及主流协议的解析，如 IP、TCP、HTTP、FTP 等，以及对 IP 和 TCP 流量的重组接口。PyPCAPKit 中使用的重组算法即是在第二章中所描述的算法。由于 PyPCAPKit 仅使用了 Python 语言进行实现，因此其对 PCAP 文件的解析速度较慢，故其还提供了对其他 PCAP 文件解析工具的支持，如 Scapy、DPKT 和 PyShark 等，即为 PyPCAPKit 的内核。上述内核在实际解析时的性能比较如下：

表 7-1 内核在实际解析时的性能比较

内核	处理 1,000,000 个包所需时间
PyPCAPKit	4 小时 52 分 3 秒
PyShark	22 小时 1 分 4 秒
Scapy	40 分 43 秒
DPKT	6 分 1 秒

然而，在处理大流量的重组时，PyPCAPKit 的效率和内存开销对系统的实现造成了极大阻碍。对于上节所使用的网络流量 PCAP 文件，使用 DPKT 作为内核，调用 PyPCAPKit 重组接口实现的 TCP 流量重组和文件提取程序在运行极长时间后将因为内存耗尽 `MemoryError` 退出，且尚未完成对所有流量的重组和提取。

7.1.2　基于 Bro 语言的实现

在纯 Python 语言的实现失败后，本项目尝试用使用 Bro 语言对 TCP 流量的重组进行直接实现。在这一方案中，本项目采用 Bro 语言实现了在第二章中所描述的 TCP 重组算法。系统使用 tcp_packet 事件，获取原始的 TCP 包，随后利用 TCP 重组算法对流量进行重组。然而，在实际应用时，也存在内存消耗过大，最终 Bro 系统底层无法申请内存空间后报错的问题。

因此，本项目又尝试了通过 `tcp_packet` 事件，将 TCP 流量提取并写入日志文件。随后，分别使用基于 DPKT 内核的 PyPCAPKit 和 C/C++实现了对这些日志文件的流量重组

以及文件提取。尽管这一实现在实际应用时不存在内存消耗过大的问题，然而由于 `tcp_packet` 事件较为底层，对 Bro 系统的负载影响较大，因此这一实现对于大文件的处理速度并不理想。

而在 Bro 语言中，可通过编写 BiF（Bro internal Function）代码实现对 Bro 功能的拓展。在 BiF 文件中，使用了 Bro 语言和 C/C++语言的混编，使用 Bro 系统提供的工具 `bifcl` 编译后，可得到动态链接库和 Bro 接口代码。然而，这一方案并未解决直接使用 `tcp_packet` 事件所带来的系统负载问题，故最终并未实现。

最后通过资料查找，发现可通过 Bro 系统内置的文件分析模块 `FileAnalysis` 实现对流量的重组和文件的提取，最终形成了 BroAPT 系统核心模块中 BroAPT-Core 提取模块的 Bro 脚本。

上述五种实现方案，处理一个大小约 1.4M 的网络流量 PCAP 文件（共 1,601 个流量包）的时间消耗如下表所示：

表 7-2 处理 1.4M 网络流量文件的时间消耗

实现方案	处理时间
纯 Bro 实现（自行实现重组算法）	14 分 36 秒
Bro 与 Python（PyPCAPKit）的混合实现	20.4 秒
纯 Python 实现（基于 DPKT 内核的 PyPCAPKit）	18.7 秒
Bro 与 C/C++的混合实现	2.4 秒
当前系统实现（基于文件分析模块）	1.8 秒

7.2 系统架构的设计与选择

如前文所述，本项目所实现的 BroAPT 系统由 BroAPT-Core 提取模块与 BroAPT-App 检测模块两个核心模块构成。在系统的原始设计中，曾采用两个不同的镜像对运行这两个模块。然而，在这一设计方案中，两个模块中的通信协调问题较为突出——如对某一 MIME 类型的文件检测 API 存在错误，该文件的检测进程在抛出后异常退出，而在下一次轮询待检测文件时，则会再次将此文件列入待检测文件中。因此，系统最终选择了 BroAPT-Core 提取模块与 BroAPT-App 检测模块在同一 Docker 镜像中运行的方案，并采用同步队列（queue）实现两个模块之间的通信协调。

7.3 本章小结

本章对项目实现过程中做出的尝试和相关工作进行了简要介绍。其中，对 TCP 流量重组与文件提取的实现方案进行了深入讨论，并在实际测试后得出上述实现方案并不理想的结论，最终采用了当前系统的实现方案。

此外，本章讨论了关于系统架构的设计，对使用单个 Docker 镜像实现核心模块，或使用两个 Docker 镜像分别实现核心模块的方案进行了分析，并最终选择了前者。

第八章　总结与展望

8.1　总结

本项目实现了 BroAPT 系统。BroAPT 系统是一个基于 Bro 的 APT 监测系统，具有较高的性能和可拓展性，以及对高速流量进行实时分析和生成日志的功能，可对流量中传输的文件进行重组和提取，并通过针对性的恶意文件检测，以及对日志的分析，实现了对流量中 APT 攻击的检测。

BroAPT 系统充分利用了 Bro 流量监测系统的高性能优势，采用多进程的实现方式对实时抓取的网络流量 PCAP 文件进行处理，并重组和提取流量中传输的文件；同时，BroAPT 系统在运行时，可通过自定义的 Bro 脚本和分析函数，对系统的原有功能进行拓展，生成定制的日志文件，产生特定的报警行为等；在对提取得到的文件进行检测时，BroAPT 系统可通过基于文件 MIME 类型的 API 配置文件，对不同类型的文件使用不同的检测方案，从而提高恶意文件检测的准确度。最后，通过系统所产生的日志文件和提取并检测的恶意文件，提出了基于 BroAPT 系统的 APT 检测方案。

通过对某学校网关处的实际流量进行分析，共使用了 1 天 19 小时 45 分钟的实际流量。这些流量大小约 12.14T，约共有 210 亿个流量包；通过对 Bro 日志 `conn.log` 分析可知，有 2,121,027 次网络连接，其中 19,751 次 HTTP 连接和 110 次 SMTP 连接等；此外，流量中共传输了 34.004 个文件，系统提取了其中 11,183 个文件，检测后并未发现恶意文件。同时，通过对 SMTP 协议分析的 Bro 脚本，系统还发现了其中有 11 次可疑的钓鱼邮件记录。

本文对 BroAPT 系统的架构设计与大体实现进行了详细介绍，并提出了基于该系统的 APT 检测方案。本文的实验结果表明，BroAPT 系统在实际应用场景中的表现优异，可实现对网关处告诉流量的实时高效处理，而基于该系统的 APT 检测方案的具体实现，则值得广大安全工作者做进一步研究，实现对高速流量 APT 检测的针对性解决方案。

8.2　目前存在的不足

本项目的目标是基于 Bro 入侵检测系统，设计并实现对高速流量的 APT 检测系统。尽管实验结果在一定程度上证明了本文中所设计的 BroAPT 系统的可用性，但本项目仍有一些缺陷和不足。

首先，在 BroAPT 系统的接口上，当前系统通过命令行接口（CLI）作为入口点，对整个系统进行控制，然而这一接口的配置功能并不完善，且存在一定入侵风险，例如若将命令行中的`--docker-compose` 参数指定为错误的配置文件（compose file），或其他路径参数配置错误，则系统将无法正常运行，且可能导致被恶意攻击利用。

同时，在当前实现中 BroAPT-Daemon 服务进程仅用于对某些 MIME 类型文件进行检测，而事实上该服务进程还可用于日志记录的协调和核心模块的调控等。例如，当前系统中通过文件锁解决日志文件的读写竞争问题，而这一解决方案也存在难以协调的潜在问题。

对于 BroAPT-App 检测模块中所使用的 API 配置文件，如用户在配置中存在错误，则其将无法对相应 MIME 类型的文件进行检测。并且，该配置文件受限于核心模块所运行的操作系统。例如，当前系统中引入的针对 Linux 系统 ELF 文件检测方案，由于 ELF Parser 无法在 CentOS 环境中正确编译，因此系统不得不通过请求 BroAPT-Daemon 服务进程，由其在另一 Ubuntu 环境的 Docker 镜像中编译和检测。此外，在本项目中，仅实现了 BroAPT 系统的设计与实现，并未对 APT 检测的具体方案进行深入研究。

需要指出的是，由于系统文件提取，即 BroAPT-Core 提取模块的核心功能，主要基于 Bro 入侵检测系统，因此其受到 Bro 系统能力本身的限制。当前，已知系统可支持对 HTTP、FTP、IRC、SMTP、DTLS 等主要应用层协议的分析与文件提取，对未知的协议或加密流量均无法进行提取。

8.3 未来的工作

基于本项目的研究成果，本系统在应用方面还可做进一步优化和拓展。

首先，系统可完善入口点接口的功能和完备性，并拓展 BroAPT-Daemon 服务进程的功能；由于其实现基于 `Flask` 这一 web 框架，因此还可在当前基础上实现系统的可视化，如数据的集成展示及查询等。而系统当前的日志系统均以文本文件形式储存，因此还可将日志系统迁移至数据库中。参考 Bro 入侵检测系统，本系统还可引入 `syslog` 或邮件发送功能，实现对攻击情报的实时报警功能。

同时，考虑到系统的可拓展性，在未来的工作和应用中，还可对系统的功能进行拓展，如引入 Bro 脚本，以支持更多的应用层协议，和其他对流量的直接分析；引入分析函数，对日志文件进行更深入的处理和分析，以便后续的 APT 检测；同时，在对恶意文件检测的 API 配置文件中，还可加入更多的检测工具，以实现更加全面的恶意文件检测功能。由于上述 API 配置文件存在操作系统的限制和依赖性，可考虑参考 Travis CI 配置文件，加入对配置运行环境（操作系统）进行指定的字段，以简化和避免配置错误。

此外，尽管当前系统已部署在实际应用环境中，但其对后续的 APT 检测并未深入研究。因此，在未来的工作中，还可进行基于 BroAPT 系统所提取的文件和生成的日志，如何有效检测 APT 攻击的研究，例如：对传输恶意文件的网络连接特征进行分析，并通过这些连接间的关联性，实现对 APT 攻击的检测。

参考文献

[1] Williams N , Zander S , Armitage G . A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification[J]. ACM SIGCOMM Computer Communication Review, 2006, 36(5):5-0.
[2] Chen P , Desmet L , Huygens C . A Study on Advanced Persistent Threats[M]. Communications and Multimedia Security. Springer Berlin Heidelberg, 2014.
[3] Lakhina A , Crovella M , Diot C . [ACM Press the 2005 conference - Philadelphia, Pennsylvania, USA (2005.08.22-2005.08.26)] Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications, - SIGCOMM "05 - Mining anomalies using traffic feature distributions[J]. 2005:217.
[4] Karagiannis T , Broido A , Faloutsos M , et al. [ACM Press the 4th ACM SIGCOMM conference - Taormina, Sicily, Italy (2004.10.25-2004.10.27)] Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, - IMC "04 - Transport layer identification of P2P traffic[J]. 2004:121.
[5] Karagiannis T . Blinc : Multilevel traffic classification in the dark[J]. ACM SIGCOMM, 2005, 2005.
[6] Roughan M , et al. Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification[C]. Proceedings of the 4th ACM SIGCOMM conference on Internet measurement. ACM, 2004.
[7] Gu Y , Mccallum A , Towsley D F . Detecting anomalies in network traffic using maximum entropy[C]. 2005.
[8] Atiya A F , Aly M A , Parlos A G . Sparse basis selection: new results and application to adaptive prediction of video source traffic[J]. IEEE Transactions on Neural Networks, 2005, 16(5):1136-1146.
[9] Auld T , Moore A W , Gull S F . Bayesian Neural Networks for Internet Traffic Classification[J]. IEEE Transactions on Neural Networks, 2007, 18(1):223-239.
[10] Li X , Qi F , Xu D , et al. An Internet Traffic Classification Method Based on Semi-Supervised Support Vector Machine[C]. 2011 IEEE International Conference on Communications (ICC). IEEE, 2011.
[11] Bilge L , Balzarotti D , Robertson W , et al. Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis[C]. Proceedings of the 28th Annual Computer Security Applications Conference. ACM, 2012.
[12] Lamprakis P , Dargenio R , Gugelmann D , et al. Unsupervised Detection of APT C&C Channels using Web Request Graphs[C]. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham, 2017.
[13] Roman J , Martin K , Tomas V. APT detection system using honeypots[J]. Proceeding of the 13th International Conference on Applied Informatics and Communications (AIC'13), WSEAS Press. 2013:25-29.

[14] Haq T , Jinjian Z , and Vinay P . Advanced persistent threat (APT) detection center: US, 9,628,507[P]. 2017-04-18.
[15] Paxson V . Bro: a system for detection network intruders in real-time[J]. Computer networks, 1999, 31(23-24):2435-2463.
[16] Kim Y H , Park W H . A study on cyber threat prediction based on intrusion detection event for APT attack detection[J]. Multimedia Tools and Applications, 2014, 71(2):685-698.
[17] Roesch M . Snort - Lightweight Intrusion Detection for Networks[J]. Proc.usenix System Administration Conf, 1999:229--238.
[18] Hay A, Bray R, Cid D. OSSEC Host-Based Intrusion Detection Guide[C]. 2008.
[19] Open Information Security Foundation. Suricata. [2019-05-28]. https://suricata-ids.org.
[20] Paxson, V. Renaming Bro project[OL]. 2018-10-11. [2019-05-28]. http://blog.bro.org/2018/10/renaming-bro-project_11.html.
[21] Mehra P . A brief stufy and comparison of snort and bro open source network intrusion detection systems. International Journal of Advanced Research in Computer and Communication Engineering. 2012. 1(6):383-386.
[22] The Tcpdump team. Tcpdump & Libpcap[OL]. [2019-05-28]. https://www.tcpdump.org.
[23] Brown R. Calendar queues: a fast O(1) priority queue implementation for the simulation event set problem[J]. Communications of the Acm, 1988, 31(10):1220-1227.
[24] Information Sciences Institute, University of Southern California. Internet Protocol: IETF RFC 791[J/OL], September 1981. [2019-05-28]. https://tools.ietf.org/html/rfc791.
[25] David D. Clark. IP datagram reassembly algorithms: IETF RFC 815[J/OL], July 1982. [2019-05-28]. https://tools.ietf.org/html/rfc815.
[26] Jarry Shaw. PyPCAPKit: Python multi-engine PCAP analysis kit[OL]. [2019-05-28]. https://github.com/JarryShaw/PyPCAPKit.
[27] Freed N , Norenstein N . Multipurpose Internet Mail Extensions: IETF RFC 2045[J/OL], November 1996. [2019-05-28]. https://tools.ietf.org/html/rfc2045.
[28] Freed N , Kucherawy M . Media Types[OL]. [2019-05-28] https://www.iana.org/assignments/media-types/media-types.xhtml.
[29] Evans C . YAML: YAML ain't markup language[OL]. [2019-05-28]. https://yaml.org.
[30]Boettiger C . An introduction to Docker for reproducible research, with examples from the R environment[J]. Acm Sigops Operating Systems Review, 2014, 49(1):71-79.
[31] Beazley D . Understanding the Python GIL[C]. PyCON Python Conference. 2010.
[32] Fielding, R , Gettys, J , Mogul J , et al. Hypertext Transfer Protocol -- HTTP/1.1: IETF RFC 2616[J/OL], 1999. [2019-05-28]. https://tools.ietf.org/html/rfc2616.
[33] Fielding, R , Reschke, R . Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing: IETF RFC 7230[J/OL], June 2014. [2019-05-28]. https://tools.ietf.org/html/rfc7230.
[34] Graham Klyne. Message Headers[OL]. [2019-05-28]. https://www.iana.org/assignments/message-headers/message-headers.xml#perm-headers%7CMessage.
[35] Hosom. Bro-phishing: Detecting phishing with Bro IDS[OL]. [2019-05-28]. https://github.com/hosom/bro-phishing.
[36] Levenshtein, Vladimir I. Binary codes capable of correcting deletions, insertions, and reversals[J]. Soviet physics doklady. 1966, 10(8): 707-710.

[37] Initconf. Smtp-url-analysis: Extracting and analyzing URLs from Emails for phishing events[OL]. [2019-05-28]. https://github.com/initconf/smtp-url-analysis.
[38]Ho, G , Sharma, A , Javed, M , Paxson, V , Wagner, D . Detecting credential spearphishing in enterprise settings[C]. 26th {USENIX} Security Symposium ({USENIX} Security 17). 2017:469-485.
[39] Martín, Alejandro, Raúl Lara-Cabrera, and David Camacho. Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset[J]. Information Fusion. 2019. 52:128-142.
[40] Martín, Alejandro, R. Lara-Cabrera, and David Camacho. A new tool for static and dynamic android malware analysis[J]. Data Science and Knowledge Engineering for Sensing Decision Support. 2018:509-516.
[41] Egaus. MaliciousMacroBot: Applied machine learning[OL]. [2019-05-28]. https://maliciousmacrobot.readthedocs.io.
[42] Jacob Baines. ELF Parser: Static ELF Analysis[OL]. [2019-05-28]. http://www.elfparser.com.
[43] R-fx Networks. LMD: Linux malware detection[OL]. [2019-05-28]. https://www.rfxn.com/projects/linux-malware-detect.
[44]Giuffrida C , Bardin, Sébastien, Blanc G . [Lecture Notes in Computer Science] Detection of Intrusions and Malware, and Vulnerability Assessment Volume 10885 || JaSt: Fully Syntactic Detection of Malicious (Obfuscated) JavaScript[J]. 2018, 10.1007/978-3-319-93411-2(Chapter 14):303-325.

谢辞

从看着题目一脸懵逼，到“速成”Bro 语言；从花式实现尝试，到最终全部放弃；从第一行 Bro 代码，到最终数千行的完整系统；在过去的五个月里，我真的经历了很多。就像过去四年来的本科生涯一样，从一个菜鸟小白，一路摸爬滚打，实现了不少项目，自认为成为了一位 Pythonista，终于“混”到了毕业。在这里，我衷心向所有曾经帮助过我、陪伴过我的人们表示感谢。

首先，要感谢我的毕业设计导师邹福泰老师。在邹老师的指导下，我曾和几位同学一起获得了信息安全作品赛的二等奖。这对于并不擅长科研的我而言，是对科研的莫大鼓励。在本次毕业设计的过程中，邹老师也一如既往，耐心地与我讨论系统设计与实现方案，并为系统测试提供了软硬件设施和大量流量作为实验环境。在我进行多种实现尝试的过程中，邹老师每天都一直与我讨论实现的效果和细节直到深夜。在这里，我非常感激邹老师的指导和付出。

我要感谢本科四年中所有的授课老师们，感谢他们授予我专业知识，让我在编程的世界中找到乐趣，让我对未来有了清晰的认知和发展方向。感谢上海交通大学网络空间安全学院为我提供了良好的学习环境，和丰富的学习材料。我尤其要感谢本科教务办的王佳力澜老师，感谢她对毕业工作的投入和关心。

同时，我要感谢在这四年里一直陪伴我的朋友们，感谢与我熟识快十年还没把我拉黑的老王医生，感谢在过去两年里和我一起实习一起学习一起比赛的小王同学，感谢他们与我一起进步，一起玩耍，一起约饭，一起约自习，让我的大学生活充满了快乐的回忆。特别地，我要感谢我的室友苟神，尽管他经常在我的项目中戳 bug，甚至在做毕设期间也不消停，但他时常与我讨论技术问题，对我的知识水平提升起到了极大作用；希望未来我们能够继续合作，实现更多有趣的项目。

我还要感谢我的实习公司暨“下家”TekID，探恪软件科技有限公司，感谢 Maxime 先生在我“走投无路”之时收下了我，感谢他让我对信息安全领域的应用和工作有了直接的了解，让我对电子取证和合规审计有了初步了解，对未来的工作方向有了亲身体验。

此外，我要郑重感谢我的父母，一直以来他们为我提供了良好的生活环境，让我能够成长为一个有才日后也能有为的人。在文章的最后，以向泰戈尔老先生（也不记得到底是哪一部诗集）的致敬作为本文的结束语吧：

写到这里，可算写完了。我说。

BROAPT: A SYSTEM FOR DETECTING APT ATTACKS IN REAL-TIME

Cybersecurity has long been a significant subject under discussion. With rapid evolution of new cyber-attack methods, the threat of Internet is becoming more and more intense. Advanced persistent threat (APT) has become a main source of cybersecurity events. It is now even more important to identify and classify network traffic by direct analysis on the traffic itself in an accurate and timely manner.

We hereby describe BroAPT system, an APT detection system based on Bro IDS. The system monitors APT based on comprehensive analysis of the network traffic. It is granted with high performance and extensibility. It can reassemble then extract files transmitted in the traffic, analyse and generate log files in real-time; it can also classify extracted files through targeted malicious file detection configuration; and it detects APT attacks based on analysis of the log files generated by the system itself.

The BroAPT system consists of two major parts. One is the core functions. This part runs in a Docker container, which currently is based on CentOS 7 image. The core functions can be described by two different components: an extraction framework BroAPT-Core and a detection framework BroAPT-App. The other is the command line interface (CLI) and a daemon server BroAPT-Daemon, which is a RESTful API server based on `Flask` framework. This part runs on the host machine of the Docker container.

CLI is the entry point for the whole BroAPT system. When running, the CLI configures the daemon server and bring it up, then start the Docker container with core functions. Within BroAPT-Core extraction framework, it will read in a PCAP file and process it with Bro IDS, which will reassemble then extract files transmitted by the traffic and generate log files from its logging system. Afterwards, BroAPT-App detection framework will take the extracted file, parse its file name to extract MIME type information of this file. Then the framework will fetch specific detection API of such MIME type and process it to detect if the file is malicious. If needed, the framework will generate a request to BroAPT-Daemon server to process a remote (privileged) detection on such file.

Of BroAPT-Core extraction framework, it mainly has three steps. First, file check. the system will scan for new PCAP files and send them to the BroAPT-Core extraction framework. Second, Bro analysis. The system will process the PCAP with file extraction scripts, reassemble then extract files transmitted through the traffic. The extraction can be grouped by MIME type of files or application layer protocol which transmitted the file. Also, the user may load external Bro scripts as site functions to process along with the main extraction scripts. Third, post-processing and cross-analysis. After processing the PCAP file with Bro IDS, the system will have several extracted files and a bunch of log files. Besides those standard Bro logs, there will be logs defined by the site functions and generated by the logging system of Bro IDS. Then the system, by default, will generate connection information of the extracted files through Bro logs, which includes timestamp, source and destination, MIME type, as well as hash values. Plus, the user may also

register Python hooks to the system, as they will be called every time a PCAP is processed. These hooks can to be used to provide further investigation upon the logs generated by Bro IDS.

To work along with Bro intrusion detection system (IDS), the system is implemented in a multi-processing manner. Since CPython's multi-threading is not working as expected -- cannot perform parallel processing -- we implemented BroAPT system with full support of multi-processing to accelerate the main processing logic. Synchronised queues are used to communicate and coordinate processes within the system: in BroAPT-Core extraction framework, we used a queue to send basic information about the extracted files to BroAPT-App detection framework, and another queue to proceed Python hooks with the generated log files.

Currently, we have introduced several site functions and Python hooks to BroAPT system. There are six bunches of Bro scripts. Constant definitions for common application layer protocols, such as HTTP and FTP, these constants are fetched from IANA registry. Extend standard Bro log `http.log` with new entry of COOKIE information and data in POST request. Calculate hash values of all files transmitted through network traffic. And two Bro modules to perform phishing emails based on cross-analysis of SMTP and HTTP traffic. The two Python hooks currently included are: one, to parse `http.log` then extract information of HTTP connections and generate a new log file; two, to cross-analyse `files.log` and `conn.log` then generate a new log file for information of extracted files.

As for BroAPT-App detection framework, we genetically designed the client-server remote detection framework based on the support of BroAPT-Daemon server. Briefly, the BroAPT-App detection framework will take the extracted files as input source. The system will perform file check to extract information from it. This information includes path to the file, MIME type and unique identifier (UID) of such file, etc. Then the system will parse an API configuration file to obtain a mapping of MIME type specific malicious file detection APIs. Based on the MIME type we had from the file, the system will perform APT detection with the selected API. When detection, the system will firstly prepare the working environment according to the API configuration: it will assign environment variables, change working directory, accordingly, expand variables defined in scripts then execute installations scripts. Afterwards, the system will execute detection scripts, then report generation script to generate detection results for the target file. If a remote detection is required, the system will prepare the request data, then post it to the BroAPT-Daemon server running on the host machine. The BroAPT-Daemon server will process the detection ibid.

Speaking of installation, we introduced several attributes to manage and avoid resource competition. We used a shared memory space to indicate whether such API has been proceeded with installation. This indicator will avoid reinstallation of APIs. It is shared with all MIME type specific APIs that sharing the same detection process, not just processes using the same API. Additionally, we have a synchronised process lock to prevent parallel installation for the same APIs. However, considering the APIs might fail due to network connection issue, we will try to rerun the script if it fails.

We have by far introduced; six different APIs targeted for dozens of MIME types. We used VirusTotal as the basic general detection method for BroAPT, which will detect any MIME types that have no registered API; VirusTotal aggregates many antivirus products and online scan engines to check for ciruses that the user's own antivuris may have missed, or to verfy against any false positives. We used AndroPyTool to detect APK files (MIME type:

`application/vnd.android.package-archive`); AndroPyTool is a tool for extracting statis and dynamic features from Android APKs, which combines different well-known Android application analysis tools such as DroidBox, FlowDroid, Strace, AndroGuard or VirusTotal analysis. We used MaliciousMacroBot to detect Office documents (MIME type: `application/vnd.openxmlformats-officedocument`, or `application/msword`, `application/vnd.ms-excel`, `application/vnd.ms-powerpoint`, etc.); MaliciousMacroBot provides a powerfule malicious file triage tool through clever feature engineering and applied machine learning techniques like Random Forest and TF-IDF. We used ELF Parser to detect Linux ELF binaries (MIME type: `application/x-executable`); ELF Parser is a static ELF analysis tool to quickly determine the capabilities of an ELF binary through static analysis. We used LMD to detect other common Linux exploitable files (MIME type: `application/octet-stream`, `text/html`, `text/x-c`, `text/x-perl`, `text/x-php`, etc.); LMD is a malware scanner for Linux systems based on threat data from network edge intrusion detection systems to extract malware that is actively being used in attacks and generates signatures for detection. And we used JaSt to detect JavaScript files (MIME type: `application/javascript` or `text/javascript`); JaSt is a tool to syntactically detect malicious (obfuscated) JavaScript files based on machine learning and clustering algorithms.

As described above, BroAPT is an APT detection system based on Bro IDS with high extensibility and compatibility with high-speed traffic. We tested BroAPT system with real-time traffic collected from the network edge of a college. The system will extract all targeted files from an approximately 35G PCAP file within one minute. And the Bro site functions introduced within BroAPT-Core extraction framework has no significant impact on performance of the system, whilst the Python hook functions will smoothly work along and generate new log files as it intended to. Also, the detection APIs we used in BroAPT-App detection system has proved that they are working perfectly with reasonable false-positive rates. In a word, the BroAPT system is working as expected in the real network environment.

However, besides the implementation above, we have tried several other implementations during the project. We used pure Python scripts based on PyPCAPKit (a multi-engine PCAP file analysis tool) with support of DPKT to reassemble and extract files transmitted through the traffic, but the process efficiency was not quite good. Not to mention hybrid implementation with Bro scripts logging TCP traffic data and Python or C/C++ programs to reassembly then extract the traffic, and the miserable pure Bro implementation of TCP reassembly. At last, File Analysis framework of Bro IDS proved its worthiness to the BroAPT system. And thus we adopted the current implementation.

Although our research on APT detection is quite preceding, the BroAPT system utilised Bro IDS and works as an APT detection system which is compatible with high-speed network traffic. The system has been proved in practical scenarios and is the basis of follow-up researches on APT detection.