论软件架构的选择

# 摘要：

2016年3月，我公司承担了国家某安全中心漏洞挖掘系统的开发工作，我在该项目中担任系统架构设计师职务，主要负责系统的架构设计。该项目是分析互联网流量，进行漏洞挖掘，主要包括流量捕获、流量存储、流量分析等。

本文以漏洞挖掘系统为例，针对不同架构风格的使用场景及优缺点，讨论了软件架构的选择。整个系统采用了面向服务的架构风格。在各个子服务中，又根据具体情况选用了不同的架构风格，例如在流量分析子服务中，采用了数据流风格；在样本分析子服务中，使用了微服务的架构风格；在各个子服务之间使用了进程通信架构风格；在漏洞靶场服务中使用了虚拟机风格。整个项目开发工作历时6个月，目前已稳定运行1年。实践证明，选用合适的架构风格提高了项目的开发效率，提高了系统的可用性、性能、可扩展性、可重用性和可移植性，从而保证了项目的顺利完成。

# 正文：

随着互联网的快速发展，网络安全问题越来越收到国家的重视。2016年3月我公司承接了国家某安全中心漏洞挖掘系统的开发工作。该项目通过对互联网中的流量进行特征分析，从中提取出相关的攻击内容，并将这些内容存储到大数据平台，结合大数据分析技术，对攻击者进行跟踪分析，从而捕获出未知漏洞。按照合同规定该项目必须在一年内完成，开发漏洞挖掘系统6个月，分析高质量的漏洞6个月。根据客户需求，该项目主要分为流量捕获、流量存储和流量分析三部分。经过分析，我将系统拆分为了流量抓取系统、文件存储系统、流量分析系统、数据库系统及web管理系统。

软件架构是一系列相关的抽象模式，是系统架构师对系统各个方面进行利弊权衡的结果，是总体设计的体现。架构选择的好坏直接决定了项目的成败，在一个项目中，好的架构风格往往能减少开发测试的难度、提高系统的稳定性，使项目开发达到事半功倍的效果。错误的架构风格，会增加项目的复杂性，降低系统的性能，增加测试的难度，给项目带来很大的风险。在进行架构选择的时候，应充分考虑影响架构选择的因素，比如项目中的参与的客户，用户，程序员，测试人员、当前的技术环境等等。软件架构风格主要分为以下几种：数据流风格、调用返回风格、独立构建风格、虚拟机风格、仓库风格、c/s、b/s及混合风格、面向服务的架构风格、微服务风格等等。下面我将具体来讨论本项目中架构风格的选择。

整体架构的选择

在系统整体架构选择上，我主要考虑了整个系统的可用性、性能及可扩展性，采用了面向服务的架构风格（SOA）。根据系统的功能和面向服务架构的特点将整体系统拆分为流量捕获服务（该服务通过流量抓取设备根据规则或定向ip抓去网络中的流量）、流量存储服务（该服务将通过流量抓取设备抓取到的流量进行分布式存储）、文件自动发现服务（该服务通过监控流量存储文件，自动发现新的流量文件）、文件服务器服务（该服务可通过http请求下载存储文件）、消息队列服务（该服务用于各个服务之间进行消息的收发）、流量分析服务（该服务实时分析网络流量并将分析结果存储在数据库中）。各个服务之间通过简单精确定义的接口进行通信。通过使用面向服务的架构，增加了系统的可重用性，使得一个服务创建后能应用于多个应用和业务流程。通过使用面向服务的架构，使得系统具有跨平台特性，各个服务根据自身的特点选择合适自己的平台，样本分析服务搭建在了linux平台，其他服务搭建在windows平台。

下面我着重从系统的可扩展性及性能两个方面讨论对整体系统使用面向服务架构风格的必要性。在可扩展性方面：通过将一个大系统拆分为不同功能的模块，而模块的构建采用了面向服务进行封装，这符合软件工程“高内聚、低耦合”的思想，使得一个模块只专注于一种服务。如果用户增加了新的需求，由于将系统各个功能拆分为了各个子服务，我们只需要修改或者更换相应的子服务即可。比如该项目是捕获并分析http流量，若我们要进一步捕获并分析ftp流量，只需要修改流量捕获服务的规则及替换新的流量分析服务就能达到目标，而其他服务不需要做任何修改。在系统的性能方面：考虑到木桶理论，经过对各个子服务分别进行性能测试，找出整体系统的性能瓶颈。针对存在性能瓶颈的子服务，增加服务器数量，提高服务器配置。针对不存在性能瓶颈的服务，通过将多个服务部署在同一台机器上更好的节约了资源。通过采用这样的策略，在相同的服务器资源情况下，使得系统性能达到最大化。在该项目中，我们将流量分析服务部署在多个高性能服务器，将消息队列服务部署在普通服务器。事实证明该项目整体架构采用面向服务的架构风格是十分正确的。下面我将进一步讨论下各子服务架构风格的选择。

各子服务的架构选择

在流量分析服务中，采用了数据流的架构风格。主要考虑到将流量捕获服务中的数据包进行拆解分析，对于http请求要经过一系列的处理过程，针对该特点采用了管道－过滤器的架构风格。使用该架构风格，使得构件具有良好的隐蔽性。将http的处理是一个个过滤器的处理行为，每个过滤器完成自己的功能，通过并发执行，提升程序的运行效率，增大子系统的吞吐量。在各个服务之间进行通信的过程中，通过使用ucmq消息队列进行进程间通信，ucmq消息队列是一款轻量级的http协议级消息队列服务组件，支持http协议、支持长链接，支持多队列。由于各个子服务都是相对队列的个体，他们之间不直接进行通信，降低了系统的耦合性，提升了灵活性。

在样本分析服务中，我们对数据库存储的http请求对处理，根据五元组进行机器学习分析，这里使用了微服务的架构风格，通过将该样本分析服务进行进一步的拆解，将其拆解为获取高危ip服务、IP定向分析服务。采用微服务的架构风格，通过将服务进行进一步的细力度划分，使得服独立部署。增加了系统的可扩展性，使得服务的部署更加简单。针对各个服务出现的问题，便于轻易的重写服务或者删除无用的服务。

对于客户提出的搭建漏洞靶场要求，通过根据不同的漏洞搭建不同的漏洞靶场，我采用了虚拟机的架构风格，将漏洞靶场所需的各个环境部署在不同的虚拟机上。漏洞靶场演示需要还原黑客的攻击过程，通过把不同的漏洞靶场搭建成不同的虚拟机，在演示过程中，用户只需要通过telnet终端命令选择不同的漏洞就会连接不同的漏洞靶场服务器，在项目演示阶段很好的演练了漏洞攻击的过程。

该项目开发工作于2016年8月完工，我们的安全分析人员和客户的安全分析人员使用该系统进行漏洞挖掘的分析。根据系统的运行情况和他们的使用反馈，验证了当初架构选择的正确性。当然软件工程“没有银弹”，任何架构都会带来一定的负面作用，对于面向服务的架构，如果盲目的对系统进行拆分，会导致整体维护成本上升。过多的服务及分布式部署，需要更大的运维投入，且需要批量化脚本部署及维护，增加了系统的维护成本。

虽然项目顺利得通过了验收，但是在项目架构设计也遇到了一些问题。其一：在采用面向架构架构风格时，由于将部分服务拆分过于微小，一方面对系统的部署带来了一些困难，增加了服务的部署及运维成本。另外过多的服务对服务之间的通信又带来了额外的通信开销。其二：在采用开源消息队列服务的时候，由于前期对其调研不深刻，满足不了系统的需求，后期通过更换为ucmq消息队列才使问题得到解决。

通过承担该项目的架构师工作，使我更加深刻的体会到了系统架构选择的好坏对项目的成败的重要性。

转自：https://blog.csdn.net/feitianxuxue/article/details/78918793