



云服务技术课堂
Cloud Technology Class

阿里云开发者社区
ALIBABA CLOUD DEVELOPER COMMUNITY

云上安全产品 实战手册

云运维工程师从入门到精通

作者：枫凡



- 精解网络攻击原理分享防护解决方案
- 真实源码详解不同系统入侵的处理方式
- 结合问题背景深入分析5+安全案例



阿里云开发者电子书系列



云技术服务大学
云产品干货高频分享



云技术服务课堂
和大牛零距离沟通



阿里云开发者“藏经阁”
海量电子书免费下载

| 目录

DDoS 攻击缓解最佳实践	4
如何让千万级业务服务免于 CC 攻击危害	9
Linux 系统被入侵后处理方式介绍	15
Windows 系统被入侵后处理方式介绍	18
WAF+SLB 负载不均衡案例分享	22
链路问题？高防链接超时	24
WAF-HTTPS [Encrypted Alert]断开连接	28
某业务间接性获取不到数据	32
Linux 系统漏洞修复失败解析	35

DDoS 攻击缓解最佳实践

分布式拒绝服务攻击（DDoS 攻击）是一种针对目标系统的恶意网络攻击行为，DDoS 攻击经常会导致被攻击者的业务无法正常访问，也就是所谓的拒绝服务。

常见的 DDoS 攻击包括以下几类：

- 网络层攻击

比较典型的攻击类型是 UDP 反射攻击，例如 NTP Flood 攻击。这类攻击主要利用大流量拥塞被攻击者的网络带宽，导致被攻击者的业务无法正常响应客户访问。

- 传输层攻击

比较典型的攻击类型包括 SYN Flood 攻击、连接数攻击等。这类攻击通过占用服务器的连接池资源从而达到拒绝服务的目的。

- 会话层攻击

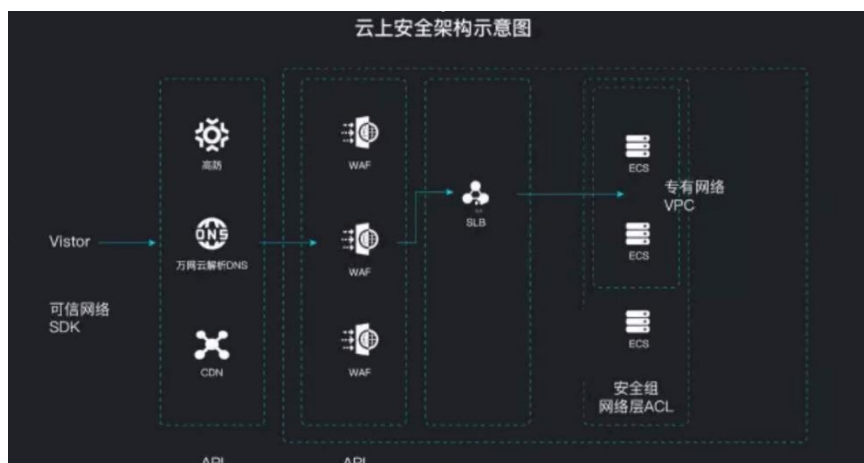
比较典型的攻击类型是 SSL 连接攻击。这类攻击占用服务器的 SSL 会话资源从而达到拒绝服务的目的。

- 应用层攻击

比较典型的攻击类型包括 DNS Flood 攻击、HTTP Flood 攻击（即 CC 攻击）、游戏假人攻击等。这类攻击占用服务器的应用处理资源，极大地消耗服务器计算资源，从而达到拒绝服务的目的。

一、DDoS 攻击缓解最佳实践

建议阿里云用户从以下几个方面着手缓解 DDoS 攻击的威胁：



- 缩小暴露面，隔离资源和不相关的业务，降低被攻击的风险。
 - 配置安全组

尽量避免将非业务必须的服务端口暴露在公网上，从而避免与业务无关的请求和访问。通过配置安全组可以有效防止系统被扫描或者意外暴露。

关于安全组的详细介绍，请参见[创建安全组](#)。
 - 使用专有网络 VPC (Virtual Private Cloud)

通过专有网络 VPC 实现网络内部逻辑隔离，防止来自内网傀儡机的攻击。

关于专有网络 VPC 的详细介绍，请参见[什么是专有网络](#)。
- 优化业务架构，利用公共云的特性设计弹性伸缩和灾备切换的系统。
 - 科学评估业务架构性能

在业务部署前期或运营期间，技术团队应该对业务架构进行压力测试，以评估现有架构的业务吞吐处理能力，为 DDoS 防御提供详细的技术参数指导信息。
 - 弹性和冗余架构

通过负载均衡或异地多中心架构避免业务架构中出现单点故障。如果您的业务在阿里云上，可以灵活地使用负载均衡服务 SLB (Server Load Balancer) 实现多台服务器的多点并发处理业务访问，将用户访问流量均衡分配到各个服务器上，降低单台服务器的压力，提升业务吞吐处理能力，这样可以有效缓解一定流量范围内的连接层 DDoS 攻击。

关于负载均衡的详细介绍，请参见[入门概述](#)。

- 。部署弹性伸缩

弹性伸缩（Auto Scaling）是根据用户的业务需求和策略，经济地自动调整弹性计算资源的管理服务。通过部署弹性伸缩，系统可以有效的缓解会话层和应用层攻击，在遭受攻击时自动增加服务器，提升处理性能，避免业务遭受严重影响。关于弹性伸缩的详细介绍，请参见[开通和授权服务](#)。

- 。优化 DNS 解析

通过智能解析的方式优化 DNS 解析，可以有效避免 DNS 流量攻击产生的风险。同时，建议您将业务托管至多家 DNS 服务商，并可以从以下方面考虑优化 DNS 解析。

- 屏蔽未经请求发送的 DNS 响应信息
- 丢弃快速重传数据包
- 启用 TTL
- 丢弃未知来源的 DNS 查询请求和响应数据
- 丢弃未经请求或突发的 DNS 请求
- 启动 DNS 客户端验证
- 对响应信息进行缓存处理
- 使用 ACL 的权限
- 利用 ACL、BCP38 及 IP 信誉功能

- 。提供余量带宽

通过服务器性能测试，评估正常业务环境下所能承受的带宽和请求数。在购买带宽时确保有一定的余量带宽，可以避免遭受攻击时带宽大于正常使用量而影响正常用户的情况。

- 服务器安全加固，提升服务器自身的连接数等性能。

对服务器上的操作系统、软件服务进行安全加固，减少可被攻击的点，增大攻击方的攻击成本：

- 。确保服务器的系统文件是最新的版本，并及时更新系统补丁。
- 。对所有服务器主机进行检查，清楚访问者的来源。
- 。过滤不必要的服务和端口。例如，对于 WWW 服务器，只开放 80 端口，将所有其他端口关闭，或在防火墙上设置阻止策略。

- 限制同时打开的 SYN 半连接数目，缩短 SYN 半连接的 timeout 时间，限制 SYN、ICMP 流量。
 - 仔细检查网络设备和服务器系统的日志。一旦出现漏洞或是时间变更，则说明服务器可能遭到了攻击。
 - 限制在防火墙外进行网络文件共享。降低黑客截取系统文件的机会，若黑客以特洛伊木马替换它，文件传输功能将会陷入瘫痪。
 - 充分利用网络设备保护网络资源。在配置路由器时应考虑针对流控、包过滤、半连接超时、垃圾包丢弃、来源伪造的数据包丢弃、SYN 阈值、禁用 ICMP 和 UDP 广播的策略配置。
 - 通过 iptable 之类的软件防火墙限制疑似恶意 IP 的 TCP 新建连接，限制疑似恶意 IP 的连接、传输速率。
- 做好业务监控和应急响应。
 - 关注基础 DDoS 防护监控

当您的业务遭受 DDoS 攻击时，基础 DDoS 默认会通过短信和邮件方式发出告警信息，针对大流量攻击基础 DDoS 防护也支持电话报警，建议您在接受到告警的第一时间进行应急处理。

关于配置告警消息接收人和语音告警方式，请参见[设置黑洞告警通知](#)。
 - 云监控

云监控服务可用于收集、获取阿里云资源的监控指标或用户自定义的监控指标，探测服务的可用性，并支持针对指标设置警报。

关于云监控的详细介绍，请参见[什么是云监控](#)。
 - 建立应急响应预案

根据当前的技术业务架构和人员，提前准备应急技术预案，必要时可以提前进行技术演练，以检验应急响应预案的合理性。
 - 选择合适的商业安全方案。阿里云既提供了免费的基础 DDoS 防护，也提供了商业安全方案。
 - Web 应用防火墙（WAF）

针对网站类应用，例如常见的 HTTP Flood 攻击，可以使用 WAF 可以提供针对

连接层攻击、会话层攻击和应用层攻击进行有效防御。

关于 WAF 的详细介绍，请参见[什么是 Web 应用防火墙](#)。

- 。 DDoS 原生防护

DDoS 原生防护为云产品 IP 提供针对 DDoS 攻击的共享全力防护能力，即时生效。

关于 DDoS 原生防护的详细介绍，请参见[什么是 DDoS 原生防护（防护包）](#)。

- 。 DDoS 高级防护

针对大流量 DDoS 攻击，建议使用阿里云 DDoS 高防服务。

关于 DDoS 高防的详细介绍，请参见[什么是 DDoS 高防](#)。

- 。 游戏盾

游戏盾是针对游戏行业常见的 DDoS 攻击、CC 攻击推出的行业解决方案。相比于高防 IP 服务，游戏盾解决方案的针对性更强，针对游戏行业的攻击防御效果更好、成本更低。

关于游戏盾的详细介绍，请参见[什么是游戏盾](#)。

二、应当避免的事项

DDoS 攻击是业内公认的行业公敌，DDoS 攻击不仅影响被攻击者，同时也会对服务商网络的稳定性造成影响，从而对处于同一网络下的其他用户业务也会造成损失。

计算机网络是一个共享环境，需要多方共同维护稳定，部分行为可能会给整体网络和其他租户的网络带来影响，需要您注意：

- 避免使用阿里云产品机制搭建 DDoS 防护平台。
- 避免释放处于黑洞状态的实例。
- 避免为处于黑洞状态的服务器连续更换、解绑、增加 SLB IP、弹性公网 IP、NAT 网关等 IP 类产品。
- 避免通过搭建 IP 池进行防御，避免通过分摊攻击流量到大量 IP 上进行防御。
- 避免利用阿里云非网络安全防御产品（包括但不限于 CDN、OSS），前置自身有攻击的业务。
- 避免使用多个账号的方式绕过上述规则。

如何让千万级业务服务免于 CC 攻击危害

一、CC 攻击的原理

CC 攻击的原理就是攻击者控制某些主机不停地发大量数据包给对方服务器造成服务器资源耗尽，一直到宕机崩溃。CC 主要是用来消耗服务器资源的，每个人都有这样的体验：当一个网页访问的人数特别多的时候，打开网页就慢了，CC 就是模拟多个用户(多少线程就是多少用户)不停地访问那些需要大量数据操作(就是需要大量 CPU 时间)的页面，造成服务器资源的浪费，CPU 长时间处于 100%，永远都有处理不完的连接直至就网络拥塞，正常的访问被中止。

二、CC 攻击的种类

CC 攻击的种类一般分为三种，直接攻击，代理攻击，僵尸网络攻击，直接攻击主要针对有重要缺陷的 WEB 应用程序，一般说来是程序写的有问题的时候才会出现这种情况，比较少见。僵尸网络攻击有点类似于 DDoS 攻击了，从 WEB 应用程序层面上已经无法防御，所以代理攻击是 CC 攻击者一般会操作一批代理服务器，比方说 100 个代理，然后每个代理同时发出 10 个请求，这样 WEB 服务器同时收到 1000 个并发请求的，并且在发出请求后，立刻断掉与代理的连接，避免代理返回的数据将本身的带宽堵死，而不能发动再次请求，这时 WEB 服务器会将响应这些请求的进程进行队列，数据库服务器也同样如此，这样一来，正常请求将会被排在很后被处理，就像本来你去食堂吃饭时，一般只有不到十个人在排队，今天前面却插了一千个人，那么轮到你的机会就很小很小了，这时就出现页面打开极其缓慢或者白屏。

三、阿里云 CC 攻击防护解决方案

产品名称	简介	应用场景
Web 应用防火墙	云盾 Web 应用防火墙 (Web Application Firewall, 简称 WAF) 基于云安全大数据能力,用于防御 SQL 注入、XSS 跨站脚本、常见 Web 服务器插件漏洞、木马上传、非授权核心资源访问等 OWASP 常见攻击,并过滤海量恶意 CC 攻击,避免您的网站资产数据泄露,保障网站的安全与可用性。	既有 CC 防护需求,由于 web 应用攻击防护要求; Web 应用攻击防护: 通用 Web 攻击防护、Oday 漏洞虚拟补丁、网站隐身; 缓解恶意 CC 攻击: 过滤恶意的 Bot 流量,保障服务器性能正常; 业务安全保障: 提供业务风控方案,解决接口防刷、防爬等业务安全风险;
DDoS 高防 IP	阿里云提供的解决互联网服务器(包括非阿里云主机)遭受大流量 DDoS/CC 攻击的安全方案。通过配置 DDoS 高防,将攻击流量牵引至高防 IP,拦截恶意流量,确保源站服务器稳定可靠。	既有防 DDoS 攻击需求,存在被 CC 攻击的情况; 如: 重大线上直播、活动推广促销场景的 DDoS 攻击防护。业务遭竞争对手恶意攻击、勒索场景的安全防护。移动业务(A PP)遭恶意注册、刷单、刷流量场景的安全防护;
SCDN	SCDN 即拥有安全防护能力的 CDN 服务,提供稳定加速的同时,智能预判攻击行为,通过智能的调度系统将 DDoS 攻击请求切换至高防 IP 完成清洗,而真正用户的请求则正常从加速节点获取资源。加速节点的分布式架构还同时具备防 CC 攻击的能力,真正达到加速和安全兼顾。	既有 CDN 加速需求; 由于安全 CC 防护和 DDoS 防护需求的; 阿里云 SCDN 基于 阿里云 CDN 的优质边缘加速资源,深度集成阿里云盾的专业攻防策略,可一站式提供安全和加速的整体解决方案。

四、CC 攻击防护

当网站受到 CC 攻击时,第一优先级为想办法恢复业务;可以尝试直接在 Web 应用防火墙开启 CC 安全防护攻击紧急,需要注意的是,攻击紧急模式适用于网页/H5 页面,但不适用于 API/Native App 业务(会造成大量误杀),针对后面这个情况,我们建议使用 CC 自定义防护;根据攻击者所攻击的特征配置防护规则。



1. CC 防护思路

- 分析请求日志找出攻击者的特征;
- 针对攻击者的特征, 使用工具将恶意的请求进行封堵;

2. CC 攻击一般特征

- 攻击的目标 URL 异常集中;
- 攻击的源 IP 异常集中;
- 攻击的源 IP 在某几个 IP 段或者某几个省;
- 使用相同的 Referer 或者 User-Agent 等;

3. CC 防护案例

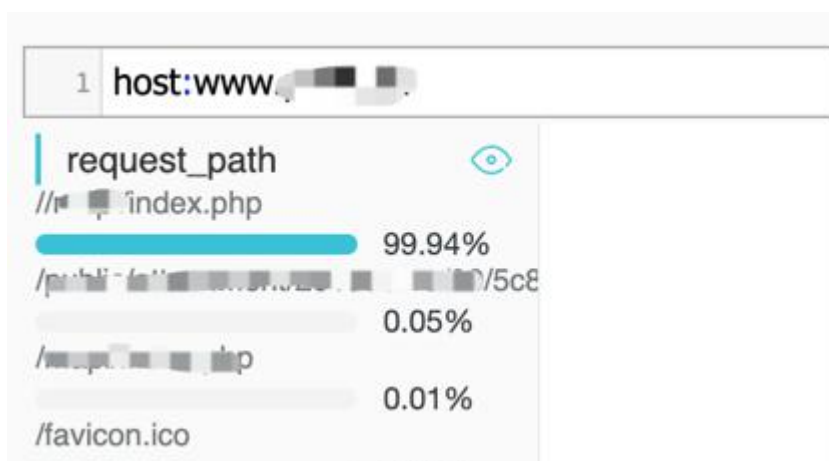
有了这些概念, 那么我们根据这几个特征去通过分析日志。我们以这个网站为例: www.xxxx.cn 在如图的这个时间段内, 峰值最高打了 13W 左右的 QPS。



碰到这个情况, 我们根据之前的 CC 攻击特征; 先分析被攻击的 URL 是哪个? 我们这里使用的是自动化的分析工具 sls 日志服务;

3.1 分析日志

使用 sls 日志服务查询得到结果绝大部分的请求都在访问 `//xxx/index.php` 这个 URL, 而正常情况下这个 URL, 不太可能有那么高的访问。



发现了这个攻击特征之后，那么在 WAF 的控制台上查到 CC 自定义防护中配置这个路径进行防护。

3.2 配置防护

配置如下：URI 使用我们通过日志分析出来的路径完全匹配；检测 10S，单一源 IP 访问 5 次，执行封禁，封禁 30 分钟；

新增规则

规则名称: 首页CC防护

URI: //index.php

匹配规则: ☒ 完全匹配 ☐ 前缀匹配

检测时长: 10 秒

单一IP访问次数: 5 次

阻断类型: ☒ 封禁 ☐ 人机识别

30 分钟

确定 取消

可以理解为，一个源 IP 在 10S 内访问超过 5 次，就触发封禁；这是一个比较严格的策略；这个频率可以根据自己业务的经验进行调整；

当然如果发现防护效果不好，可以初步的设置更加严格的策略。同时也可以选“人机识别”进行防护，人机识别是 WAF 返回一段特殊的代码给客户端（WAF 返回 200 状态码），判断客户端是否可能正常执行这个代码；如果能够执行则通过验证，然后放行；如果无法执行，这个客户端 IP 就会加黑对应的时间。如果在网站架构中，WAF 前端有高防或者 CDN 之类的产品，推荐使用人机识别进行防护。

3.3 深挖攻击特征

在这个时候网站业务正在慢慢恢复；我们继续分析日志，找到更加精确的攻击特征加以防护。

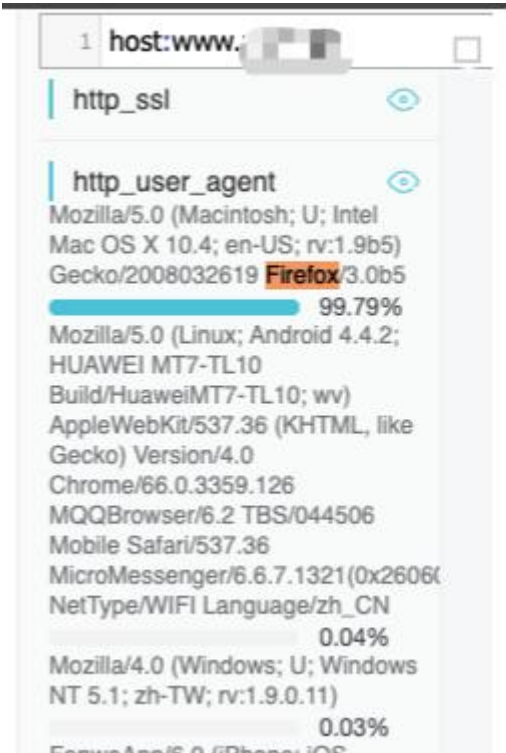
3.4 分析客户端 IP 分布

我们通过分析客户端真实 IP 地址时，TOP10 的 IP 地址以及地域分布情况如下，看不出明显的异常。



3.5 分析 user_agent 特征

有大量的请求使用到的 UA 是“Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.4; en-US; rv:1.9b5) Gecko/2008032619 Firefox/3.0b5”这个 UA，看着是一个 MAC OS 系统下 FireFox 的浏览器，由于业务是 APP 类型的；很少会有 MAC OS 系统下的访问，同时我们对比正常时间段访问记录的 user_agent 没有发现 Firefox 字样的 user_agent 记录；可以判断该 user_agent 是异常的；



根据这个攻击特征，我们使用 WAF 的精准访问控制策略对该 user_agent 进行控制，配置如下，对于包含 Firefox 进行封禁：阻断的策略

新增规则

规则名称：

首页CC防护

匹配条件：

匹配字段	逻辑符	匹配内容
User-Agent	包含	Firefox
URL	包含	/index.php

+ 新增条件

匹配动作：

阻断

确定

取消

通过这种组合策略的防护策略能够更好保护业务正常对外提供服务。

Linux 系统被入侵后处理方式介绍

使用前一定先创建快照备份，否则不要使用本文方法。

一、将 ECS 断开网络连接

使用 ECS 安全组单独对该 ECS 进行隔离；出方向禁止所有协议。入方向只允许运维的端口和指定 IP 进入，其他均禁止。

二、shell 以 root 的权限运行

核心的点：将系统内部非正常的系统文件，杀掉进程；删除文件，删除注册表，删除计划任务，禁止/停止/删除服务。

下述脚本只做参考：

复制下述内容，放到 Linux 系统中，新建一个.sh 文件保存。使用 root 权限进行运行。木马病毒的文件名、服务名都有可能发生变化；脚本只是辅助。

主要是使用 busybox 来执行指令，因为该命令不会调用动态链接库，不回被劫持。

busybox 的具体介绍

```
service crond stop

busybox rm -f /etc/ld.so.preload
busybox rm -f /usr/local/lib/libioset.so
chattr -i /etc/ld.so.preload
busybox rm -f /etc/ld.so.preload
busybox rm -f /usr/local/lib/libioset.so

# 清理异常进程

busybox ps -ef | busybox grep -v grep | busybox egrep 'ksoftirqds' | busybox
awk '{print $1}' | busybox xargs kill -9
```

```
busybox ps -ef | busybox grep -v grep | busybox egrep 'watchdogs' | busybox  
awk '{print $1}' | busybox xargs kill -9
```

```
busybox rm -f /tmp/watchdogs
```

```
busybox rm -f /etc/cron.d/tomcat
```

```
busybox rm -f /etc/cron.d/root
```

```
busybox rm -f /var/spool/cron/root
```

```
busybox rm -f /var/spool/cron/crontabs/root
```

```
busybox rm -f /etc/rc.d/init.d/watchdogs
```

```
busybox rm -f /usr/sbin/watchdogs
```

```
ldconfig
```

```
# 再次清理异常进程
```

```
busybox ps -ef | busybox grep -v grep | busybox egrep 'ksoftirqds' | busybox  
awk '{print $1}' | busybox xargs kill -9
```

```
busybox ps -ef | busybox grep -v grep | busybox egrep 'watchdogs' | busybox  
awk '{print $1}' | busybox xargs kill -9
```

```
# 清理开机启动项
```

```
chkconfig watchdogs off
```

```
chkconfig --del watchdogs
```

```
service crond start
```

计划任务文件处理

```
/etc/crontab
```

```
/etc/cron.d/
```

```
/var/spool/cron/
```

```
/var/spool/cron/crontabs/
```

```
/etc/cron.hourly
```

```
/etc/cron.daily
```

```
/etc/cron.weekly
```

```
/etc/cron.monthly
```

```
/var/spool/cron/crontabs/root
```

这里面有没有异常的计划任务，如有者进行清理

三、Redis 业务特殊处理

如无业务必要，修改 Redis 只监听 127.0.0.1，并为 Redis 设置认证密码。编辑 Redis 配置文件/etc/redis.conf 以下行保存后使用 `service redis restart` 重启 Redis 服务：

```
bind 127.0.0.1    #配置只监听本地回环地址 127.0.0.1
```

```
requirepass xxx  #去掉行前注释，修改密码为所需的密码。
```

四、登录密钥处理

不影响业务的情况下，建议临时删除机器上.ssh/known_hosts 和登录密钥文件。(蠕虫常常利用这个进行关联传播，会在短时间内扩散到所有内网机器)

五、附录

常见问题处理

- Docker 服务安全加固
- FTP 匿名登录或弱口令漏洞及服务加固
- Jenkins 服务安全加固
- Kubernetes 服务安全加固
- Hadoop 环境安全加固
- FileZilla FTP Server 安全加固
- Elasticsearch 服务安全加固
- phpMyadmin 服务安全加固
- 数据库加固

Windows 系统被入侵后处理方式介绍

使用前一定先创建快照备份，否则不要使用本文方法。

一、将 ECS 断开网络连接

使用 ECS 安全组单独对该 ECS 进行隔离；出方向禁止所有协议。入方向只允许运维的端口和指定 IP 进入，其他均禁止。

二、脚本的 cmd 以管理员权限运行。

核心的点：将系统内部非正常的系统文件，杀掉进程；删除文件，删除注册表，删除计划任务，禁止/停止/删除服务。

下述脚本只做参考：

复制下述内容，放到 windows 系统中，新建一个 txt 文件保持，然后修改后缀为 bat。使用管理员权限进行运行。

木马病毒的文件名、服务名都有可能会发生变化；脚本只是辅助。

```
taskkill /F /im JVIBH.exe
taskkill /F /im TsRMf.exe
taskkill /F /im 3PwJLGwf.exe
taskkill /F /im oydr2006.exe
taskkill /F /im lzjp.exe
taskkill /F /im nIaX.exe
taskkill /F /im kCXsjb.exe
taskkill /F /im cymdxn.exe
taskkill /F /im vuoj.exe
taskkill /F /im mAc4c9G0.exe
taskkill /F /im EsH7Fc5F.exe
```

```
taskkill /F /im cM0vRGi9.exe
taskkill /F /im fzary.exe
taskkill /F /im 80EDN6nD.exe
del C:\Windows\JVIBH.exe /a:h /f
del c:\windows\TsRMf.exe /a:h /f
del C:\Windows\3PwJLGwf.exe /a:h /f
del C:\Windows\oydr2006.exe /a:h /f
del C:\Windows\TEMP\lzjp.exe /a:h /f
del C:\Windows\nIaX.exe /a:h /f
del c:\windows\kCXsjb.exe /a:h /f
del C:\Windows\TEMP\cymdxn.exe /a:h /f
del C:\Windows\TEMP\vuoj.exe /a:h /f
del C:\Windows\mAc4c9G0.exe /a:h /f
del C:\Windows\Esh7Fc5F.exe /a:h /f
del C:\Windows\cM0vRGi9.exe /a:h /f
del C:\Windows\TEMP\fzary.exe /a:h /f
del C:\Windows\80EDN6nD.exe /a:h /f
schtasks /delete /TN JVIBH /F
schtasks /delete /TN mhLpn85RiQ /F
schtasks /delete /TN IZY8bgnbEO /F
schtasks /delete /TN nIaX /F
schtasks /delete /TN DFPDYg2Dyw /F
schtasks /delete /TN 7bZAylWFwu /F
schtasks /delete /TN sRr47NEs9l /F
schtasks /delete /TN p6Mc2efsyB /F
schtasks /delete /TN "\OqpUpVgk" /F
schtasks /delete /TN "\hGKBet" /F
schtasks /delete /TN \Microsoft\Windows\lzjp /F
schtasks /delete /TN "\pwXs" /F
schtasks /delete /TN \Microsoft\Windows\cymdxn /F
schtasks /delete /TN \Microsoft\Windows\vuoj /F
schtasks /delete /TN \Microsoft\Windows\fzary /F
sc config Ddriver start= disabled
```

```
sc config WebServers start= disabled

net stop Ddriver

net stop WebServers

taskkill /im wmiex.exe

netsh interface portproxy delete v4tov4 listenport=65532
netsh interface portproxy delete v4tov4 listenport=65531

netsh advfirewall firewall del rule name=UDP dir=in
netsh advfirewall firewall del rule name=UDP2 dir=in
netsh advfirewall firewall del rule name=ShareService dir=in

del c:\windows\syswow64\wmiex.exe /a:h /f
del c:\windows\system32\wmiex.exe /a:h /f
del C:\Windows\Temp\_MEI5697402\*. * /f /q
del C:\Windows\Temp\m.ps1 /f /q
del C:\Windows\Temp\*.tmp /f /q
del C:\Windows\Temp\*.sqm /f /q
del C:\Windows\Temp\*.vbs /f /q
del C:\Windows\Temp\mkatz.ini /f /q
del C:\Windows\Temp\svchost.exe /f /q
del c:\windows\syswow64\drivers\svchost.exe /a:h /f
del c:\windows\system32\drivers\svchost.exe /a:h /f
del c:\windows\syswow64\drivers\taskmgr.exe /a:h /f
del c:\windows\system32\drivers\svchost.exe /a:h /f
del c:\windows\syswow64\drivers\svchost.exe /a:h /f
del c:\windows\system32\drivers\svchost.exe /a:h /f
del c:\windows\syswow64\drivers\taskmgr.exe /a:h /f
del c:\windows\system32\drivers\svchost.exe /a:h /f

REG DELETE HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Ddriver /f
REG DELETE HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\WebServers /f

schtasks /delete /TN Ddrivers /F
schtasks /delete /TN DnsScan /F
schtasks /delete /TN WebServers /F
```


三、除了使用清理脚本自动清理外，还建议最好在本机查看下：

1. 病毒木马可以利用了系统自带的服务 RemComSvc，创建管道横向传播。如果业务不需要该服务，建议停掉。

具体操作命令如下，在 cmd 中执行即可。

```
sc config RemComSvc start= disabled
```

2. 病毒木马可能会有扫描的 smb 的 445 端口，如果业务不需要开放 445 端口，建议关闭。

关闭命令如下，同样在 cmd 下运行即可。

```
ipseccmd -w REG -p "HFUT_SECU" -r "Block TCP/445" -f *+0:445:TCP -n BLOCK -x
```

```
ipseccmd -w REG -p "HFUT_SECU" -r "Block UDP/445" -f *+0:445:UDP -n BLOCK -x
```

3. 建议查看相关注册表启动项，看看是否还有可疑启动项。

```
HKEY_CURRENT_USER/Software/Microsoft/Windows/CurrentVersion/Run
```

```
HKEY_CURRENT_USER/Software/Microsoft/Windows/CurrentVersion/RunOnce
```

```
HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Run
```

```
HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/RunOnce
```

WAF+SLB 负载不均衡案例分享

一、问题演变过程

时间点 1: 高防+WAF+SLB+2 台 ECS

时间点 2: 高防+WAF+SLB+4 台 ECS

二、问题描述

在时间点 1 时，没有发现明显的负载不均衡的情况。在时间点 2 时，出现大部分请求都打到了其中一台 ECS 上。需要定位问题原因。

三、问题梳理

- 问题链路

是 SLB 后端的 ECS 出现负载不均衡的请求，那么直接影响这个转发算法，是 WAF 以及 SLB。那么和高防没有关系了。



- 配置情况

SLB: TCP 监听, WRR 转发算法, 开启会话保持

WAF: 无特殊配置, 域名直接回源负载均衡 IP

四、问题点 1：轮询算法+会话保持

措施：尝试修改轮询算法为 WLC，会话保持时间调短。

然而这个优化措施效果并不明显，由于开启了会话保持，那原有负载不均衡的情况下，调整 WRR 算法到 WLC 的算法，没有实现预期的 WLC。

但是从另外一个角度来说，如果源 IP 非常分散的场景下，即使有会话保持，理论上还是应该在经过一个较长的时间段之后，依然能够到达均衡。

这里由于是使用 WAF 的回源地址进行访问，所以对负载均衡来说，客户端的公网 IP 地址是固定的，一直是固定的几个；从而调整 WLC+会话保持的调整收效甚微。

五、问题点 2：会话保持模板刷新问题

措施：尝试关闭会话保持。

稍有成效：关闭会话保持后，经过一段时间的通信，4 台 ECS 初步的开始均衡，但是到了一个固定值之后；没有继续均衡，一直保持着 1:2 的状态。

这里有 2 个知识点：

1. WLC 算法的计数开始是从调整为这个算法的时间点开始的；那么如果历史开始就出现不均衡，那么开启后还是会不均衡的。
2. 由于 WAF 的回源地址与 SLB 的通信一直在，没有断过所以历史的会话保持的效果依然存在，已经会话保持的 IP，依然会发给对应负载均衡的 RS，导致不均衡。

推荐的解法为：使用负载均衡的权重功能，将连接数多的机器的权重调低，待 4 台机器的连接数基本均衡后，将 RS 的权重都调整为一致。

链路问题？高防链接超时

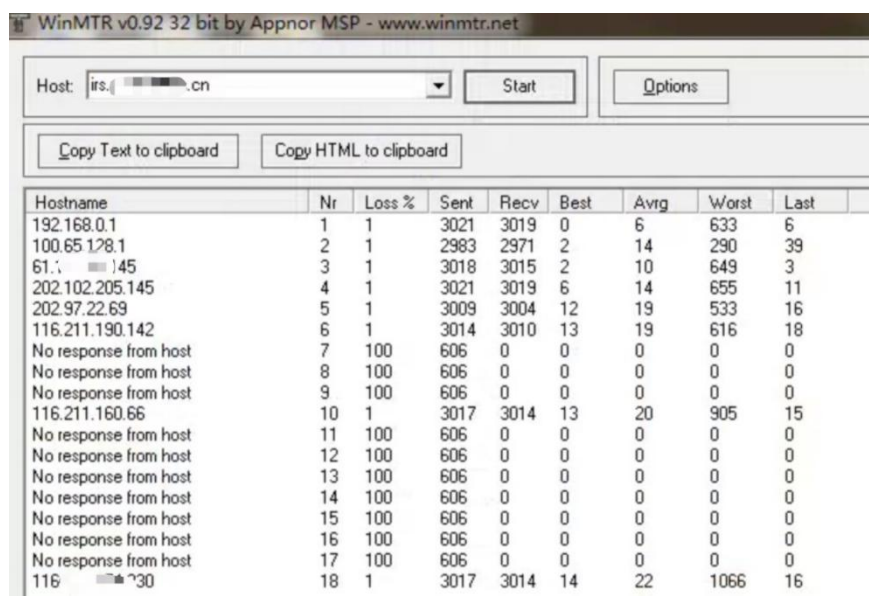
一、问题简述

某用户反馈安徽电话经常出现链接高防 IP 超时的情况，判断是安徽电信链接高防存在链路问题，期望我们协助解决。

二、问题分析

1. 网络测试

用户在存在有问题的客户端上协助执行 ping 以及 mtr，从 MTR 上看，客户端就出现丢包，是本地客户端网络的问题。但是这个丢包率其实很低，同时 TCP 协议是有重传机制的，理论上不会出现明显链接中断。



Hostname	Nr	Loss %	Sent	Recv	Best	Avrg	Worst	Last
192.168.0.1	1	1	3021	3019	0	6	633	6
100.65.128.1	2	1	2983	2971	2	14	290	39
61.1.1.145	3	1	3018	3015	2	10	649	3
202.102.205.145	4	1	3021	3019	6	14	655	11
202.97.22.69	5	1	3009	3004	12	19	533	16
116.211.190.142	6	1	3014	3010	13	19	616	18
No response from host	7	100	606	0	0	0	0	0
No response from host	8	100	606	0	0	0	0	0
No response from host	9	100	606	0	0	0	0	0
116.211.160.66	10	1	3017	3014	13	20	905	15
No response from host	11	100	606	0	0	0	0	0
No response from host	12	100	606	0	0	0	0	0
No response from host	13	100	606	0	0	0	0	0
No response from host	14	100	606	0	0	0	0	0
No response from host	15	100	606	0	0	0	0	0
No response from host	16	100	606	0	0	0	0	0
No response from host	17	100	606	0	0	0	0	0
116.211.160.30	18	1	3017	3014	14	22	1066	16

2. 进一步进行分析抓包

用户反馈了 2 个抓包，发现了 2 个场景：

2.1 客户端直接连接负载均衡也会出现问题

No.	d	Time	Source	TTL	Destination	Seq	ACK	Sport	Protocol	Length	Info
846841	0.000229	2018-07-29 12:20:14.998053	192.168.1.30	64	115.28.115.126	263	1	51968	TCP	71	51968 → 80 [PSH, ACK] Seq=263
846842	0.000117	2018-07-29 12:20:14.998178	192.168.1.30	64	115.28.115.126	280	1	51968	TCP	76	51968 → 80 [PSH, ACK] Seq=280
846843	0.000047	2018-07-29 12:20:14.998217	192.168.1.30	64	115.28.115.126	302	1	51968	HTTP	150	POST /irs-iface/on/inf/pay/v...
846864	0.040887	2018-07-29 12:20:15.831184	115.28.115.126	51	192.168.1.30	1	209	80	TCP	60	80 → 51968 [ACK] Seq=1 Ack=2...
846878	0.003963	2018-07-29 12:20:15.835067	115.28.115.126	51	192.168.1.30	1	223	80	TCP	60	80 → 51968 [ACK] Seq=1 Ack=2...
846871	0.000000	2018-07-29 12:20:15.835067	115.28.115.126	51	192.168.1.30	1	241	80	TCP	60	80 → 51968 [ACK] Seq=1 Ack=2...
846875	0.004177	2018-07-29 12:20:15.839244	115.28.115.126	51	192.168.1.30	1	263	80	TCP	60	80 → 51968 [ACK] Seq=1 Ack=2...
846876	0.000001	2018-07-29 12:20:15.839245	115.28.115.126	51	192.168.1.30	1	280	80	TCP	60	80 → 51968 [ACK] Seq=1 Ack=2...
846877	0.000000	2018-07-29 12:20:15.839245	115.28.115.126	51	192.168.1.30	1	302	80	TCP	60	80 → 51968 [ACK] Seq=1 Ack=3...
846878	0.000091	2018-07-29 12:20:15.839246	115.28.115.126	51	192.168.1.30	1	309	80	TCP	60	80 → 51968 [ACK] Seq=1 Ack=3...
846886	0.005569	2018-07-29 12:20:15.844815	115.28.115.126	51	192.168.1.30	1	398	80	HTTP	384	HTTP/1.1 200 OK (text/plain)
846132	0.199332	2018-07-29 12:20:15.244147	192.168.1.30	64	115.28.115.126	398	251	51968	TCP	54	51968 → 80 [ACK] Seq=398 Ack=...
848957	14.663872	2018-07-29 12:20:29.988019	192.168.1.30	64	115.28.115.126	398	251	51968	TCP	262	51968 → 80 [PSH, ACK] Seq=39...
848958	0.000060	2018-07-29 12:20:29.988187	192.168.1.30	64	115.28.115.126	686	251	51968	TCP	60	51968 → 80 [PSH, ACK] Seq=68...
848959	0.000060	2018-07-29 12:20:29.988167	192.168.1.30	64	115.28.115.126	620	251	51968	TCP	72	51968 → 80 [PSH, ACK] Seq=62...
848960	0.000047	2018-07-29 12:20:29.988214	192.168.1.30	64	115.28.115.126	638	251	51968	TCP	76	51968 → 80 [PSH, ACK] Seq=63...
848961	0.000244	2018-07-29 12:20:29.988458	192.168.1.30	64	115.28.115.126	660	251	51968	TCP	71	51968 → 80 [PSH, ACK] Seq=66...
848962	0.000061	2018-07-29 12:20:29.988519	192.168.1.30	64	115.28.115.126	677	251	51968	TCP	76	51968 → 80 [PSH, ACK] Seq=67...
848963	0.000054	2018-07-29 12:20:29.988573	192.168.1.30	64	115.28.115.126	699	251	51968	HTTP	150	POST /irs-iface/on/inf/pay/v...
849836	0.293805	2018-07-29 12:20:30.205878	192.168.1.30	64	115.28.115.126	398	251	51968	TCP	451	(TCP Retransmission) 51968 →
849899	0.000064	2018-07-29 12:20:30.080522	192.168.1.30	64	115.28.115.126	398	251	51968	TCP	451	(TCP Retransmission) 51968 →
849234	1.203463	2018-07-29 12:20:32.080977	192.168.1.30	64	115.28.115.126	398	251	51968	TCP	451	(TCP Retransmission) 51968 →
849235	0.031750	2018-07-29 12:20:32.038727	115.28.115.126	51	192.168.1.30	251	415338	80	TCP	60	80 → 51968 [RST] Seq=251 Win=...

问题原因：分析这个报文看到，这个客户端的 request，是过了 15S 再次发出的。由于负载均衡的默认 keepalive 的时间为 15S 的，如果超过 15S，就断开链接了，导致通信有问题。

解决方法：建议在负载均衡的控制台调整到最长 60S 观察下的。

2.2 客户端链接高防出现问题

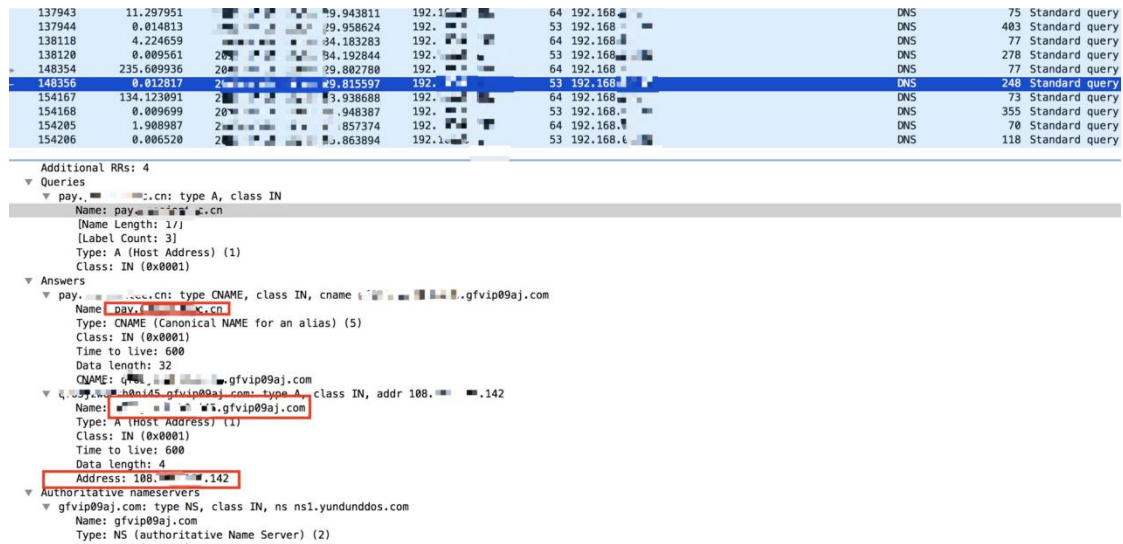
20674	12:44:11.519	queryResult arg="..."	Httpclient
20675	12:44:11.519	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20676	12:44:18.845	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20677	12:44:18.845	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20678	12:44:18.845	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20679	12:44:18.846	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20680	12:44:18.846	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20681	12:44:18.882	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20682	12:44:18.882	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20683	12:44:18.925	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20684	12:44:18.925	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20685	12:44:18.930	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20686	12:44:18.990	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20687	12:44:18.990	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20688	12:44:19.293	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20689	12:44:19.295	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20690	12:44:19.295	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20691	12:44:24.212	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20692	12:44:27.053	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20693	12:44:29.299	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20694	12:44:29.301	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20695	12:44:29.301	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20696	12:44:30.023	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20697	12:44:33.098	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20698	12:44:33.098	java.net.SocketException: Broken pipe (Write failed)	Httpclient

如图为 7 月 29 日 11:44:23 到 7 月 29 日 12:46:33 的抓包。但是与高防 IP 的 443 端口的通信，在 7 月 29 日 12:23:13 之后就没有了，而给的日志是时间是 7 月 29 日 12:44:11 这个很诡异。

继续分析报文，理论上当时在抓包，所有的通信都会保留在报文中，同时客户端的报错日志为：

20674	12:44:11.519	queryResult arg="..."	Httpclient
20675	12:44:11.519	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20676	12:44:18.845	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20677	12:44:18.845	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20678	12:44:18.845	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20679	12:44:18.846	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20680	12:44:18.846	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20681	12:44:18.882	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20682	12:44:18.882	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20683	12:44:18.925	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20684	12:44:18.925	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20685	12:44:18.930	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20686	12:44:18.990	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20687	12:44:18.990	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20688	12:44:19.293	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20689	12:44:19.295	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20690	12:44:19.295	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20691	12:44:24.212	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20692	12:44:27.053	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20693	12:44:29.299	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20694	12:44:29.301	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20695	12:44:29.301	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20696	12:44:30.023	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20697	12:44:33.098	java.net.SocketException: Broken pipe (Write failed)	Httpclient
20698	12:44:33.098	java.net.SocketException: Broken pipe (Write failed)	Httpclient

最终定位是由于客户端的原因，查询 dns 的请，如下图存在 dns 劫持。对应的 cname 被劫持到了一个海外的地址。



No.	Time	Source	Destination	Seq	ACK	Sport	Protocol	Length	Info
137943	11.297951	192.168.0.240	192.168.0.240	64	0	49851	DNS	75	Standard query
137944	0.014813	192.168.0.240	192.168.0.240	53	0	49851	DNS	403	Standard query
138118	4.224659	192.168.0.240	192.168.0.240	53	0	49851	DNS	77	Standard query
138120	0.009561	192.168.0.240	192.168.0.240	53	0	49851	DNS	278	Standard query
148354	235.609936	192.168.0.240	192.168.0.240	53	0	49851	DNS	77	Standard query
148356	0.012817	192.168.0.240	192.168.0.240	53	0	49851	DNS	248	Standard query
154167	134.123091	192.168.0.240	192.168.0.240	64	0	49851	DNS	73	Standard query
154168	0.009699	192.168.0.240	192.168.0.240	53	0	49851	DNS	355	Standard query
154205	1.908987	192.168.0.240	192.168.0.240	64	0	49851	DNS	70	Standard query
154206	0.006520	192.168.0.240	192.168.0.240	53	0	49851	DNS	118	Standard query

Additional RRs: 4

Queries

pay.com: type A, class IN

Name: pay.com

[Name Length: 11]

[Label Count: 3]

Type: A (Host Address) (1)

Class: IN (0x0001)

Answers

pay.com: type CNAME, class IN, cname gfvip09aj.com

Name: pay.com

Type: CNAME (Canonical Name for an alias) (5)

Class: IN (0x0001)

Time to live: 600

Data length: 32

CNAME: gfvip09aj.com

Name: gfvip09aj.com

Type: A (Host Address) (1)

Class: IN (0x0001)

Time to live: 600

Data length: 4

Address: 108.160.166.142

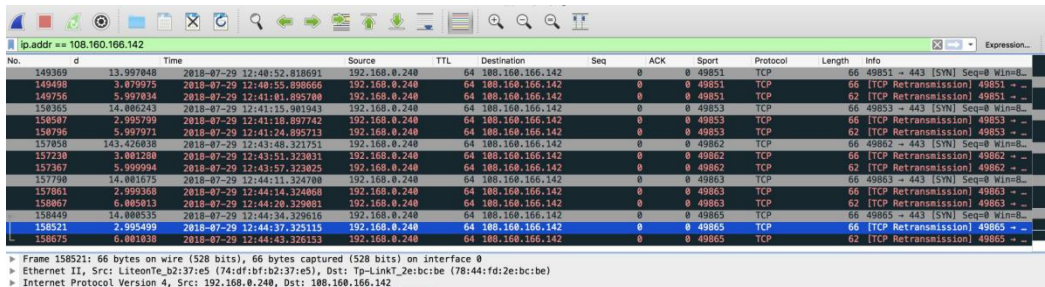
Authoritative nameservers

gfvip09aj.com: type NS, class IN, ns ns1.yundunddos.com

Name: gfvip09aj.com

Type: NS (authoritative Name Server) (2)

同时查询与 IP 地址的通信全部超时了。



No.	Time	Source	Destination	Seq	ACK	Sport	Protocol	Length	Info
149369	13.957048	192.168.0.240	108.160.166.142	0	0	49851	TCP	66	49851 → 443 [SYN] Seq=0 Win=0
149403	3.832919	108.160.166.142	192.168.0.240	0	0	49851	TCP	66	49851 → 443 [SYN] Seq=0 Win=0
149756	5.997034	192.168.0.240	108.160.166.142	0	0	49851	TCP	62	TCP Retransmission] 49851 →
158365	14.006243	192.168.0.240	108.160.166.142	0	0	49853	TCP	66	49853 → 443 [SYN] Seq=0 Win=0
158587	2.955799	192.168.0.240	108.160.166.142	0	0	49853	TCP	66	TCP Retransmission] 49853 →
158706	5.997271	192.168.0.240	108.160.166.142	0	0	49853	TCP	62	TCP Retransmission] 49853 →
157058	143.426038	192.168.0.240	108.160.166.142	0	0	49862	TCP	66	49862 → 443 [SYN] Seq=0 Win=0
157238	3.001288	192.168.0.240	108.160.166.142	0	0	49862	TCP	66	TCP Retransmission] 49862 →
157267	5.999994	192.168.0.240	108.160.166.142	0	0	49862	TCP	62	TCP Retransmission] 49862 →
157290	16.001015	192.168.0.240	108.160.166.142	0	0	49863	TCP	66	49863 → 443 [SYN] Seq=0 Win=0
157861	2.999388	192.168.0.240	108.160.166.142	0	0	49863	TCP	66	TCP Retransmission] 49863 →
158867	6.005013	192.168.0.240	108.160.166.142	0	0	49863	TCP	62	TCP Retransmission] 49863 →
158849	14.000535	192.168.0.240	108.160.166.142	0	0	49865	TCP	66	49865 → 443 [SYN] Seq=0 Win=0
159521	2.955409	192.168.0.240	108.160.166.142	0	0	49865	TCP	66	TCP Retransmission] 49865 →
158875	6.001038	192.168.0.240	108.160.166.142	0	0	49865	TCP	62	TCP Retransmission] 49865 →

Frame 158521: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0

Ethernet II, Src: LiteonTe_b2:37:e5 (74:df:bf:b2:37:e5), Dst: Tp-LinkT_2e:bc:be (78:fd:4d:2e:bc:be)

Internet Protocol Version 4, Src: 192.168.0.240, Dst: 108.160.166.142

最终的结论为由于该域名对应的 cname 地址被劫持了，导致与服务端通信异常。

3. 处理建议

- 最终客户端修改 dns 地址为 223.5.5.5 223.6.6.6 然后观察的。
- 最终客户端绑定 HOST 到 高防 IP 上进行使用的。
- 修改域名解析为 A 记录，然后使用的。

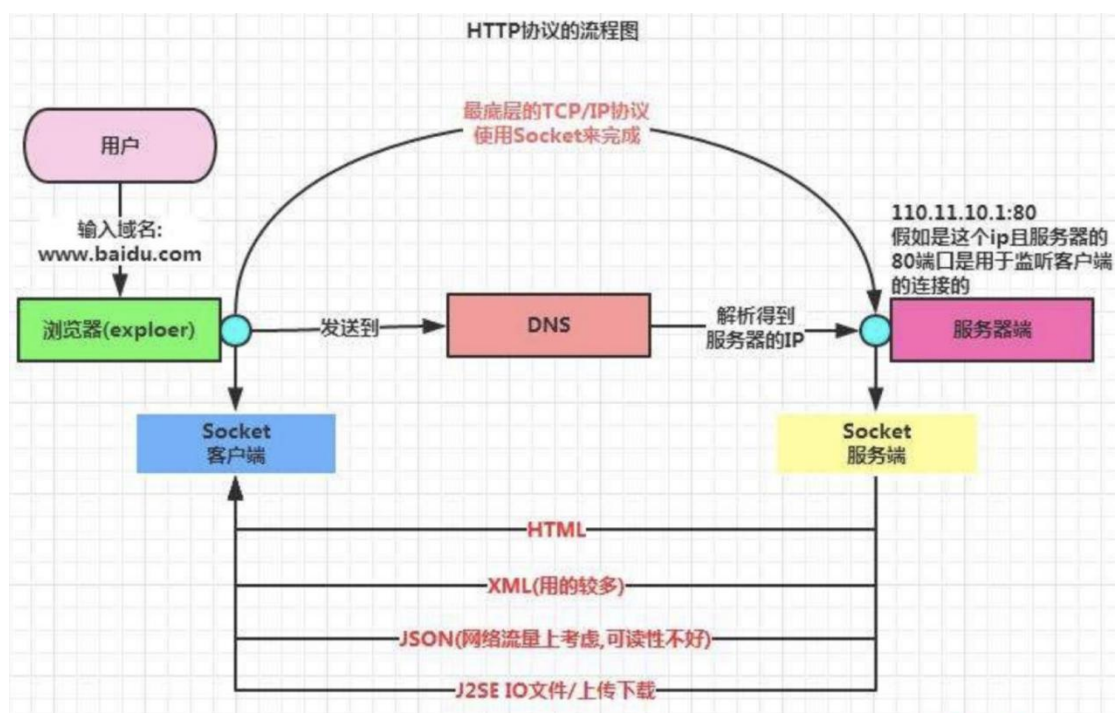
3 个方法，影响面不一样，可以进行评估下选择。

最终通过联系运营商刷新 dns 缓存后解决。

三、总结

这个问题能够分析出来有 2 个核心点:

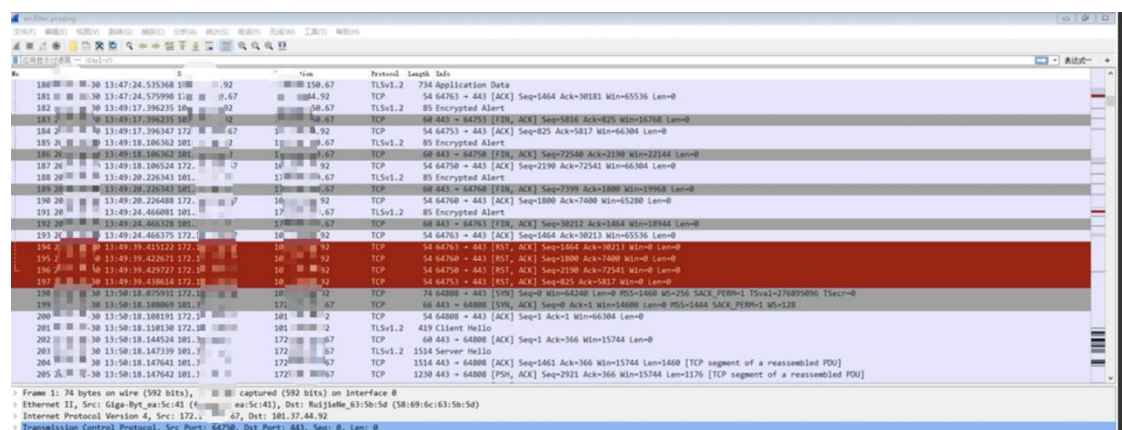
1. 抓包时抓取了所有的报文, 而非与高防 IP 进行通信的报文。不是所有问题都是表面看到的。
2. 熟悉以及了解 HTTP 的通信全过程。



WAF-HTTPS [Encrypted Alert]断开连接

一、问题背景

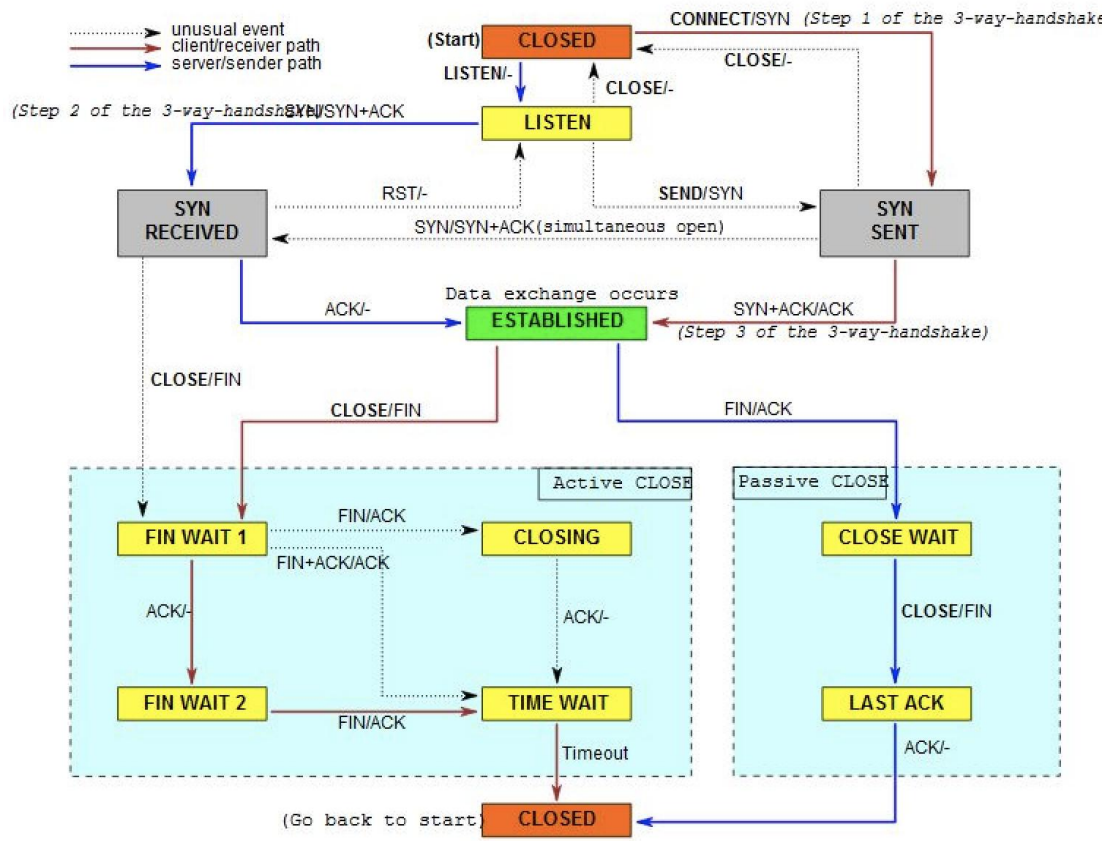
使用阿里云的 WAF 时，HTTPS 访问，偶尔会出现 SSL 建立连接失败,抓包发现 WAF 返回了 Encrypted Alert，然后发了 FIN 包结束，内容里头序号 182~197 的包都有异常。



二、问题分析

问题方向确定

问题描述为:抓包发现 WAF 返回了 Encrypted Alert，然后发了 FIN 包结束，分析 FIN 包是 TCP 链接中断开时使用的，如图：



确定该问题不是用户描述的 SSL 建立链接失败；而是断开链接有问题。

三、分析报文

定位用户描述的报文，如下确实有看到描述的报文，但是该分析方式无法有效的判断问题：

No.	D	Time	Source	Len	Protocol	Length	Info
172	0.000021	2017-03-13 13:47:24.470977	172.17.0.67	54	TCP	54	64763->443 [ACK] Seq=1464 Ack=20741 Win=66304 Len=0
173	0.063815	2017-03-13 13:47:24.534792	10.10.1.92	1514	TLSv1.2	1514	Application Data
174	0.000271	2017-03-13 13:47:24.535063	101.101.92	1514	TLSv1.2	1514	Application Data[TCP segment of a reassembled PDU]
175	0.000001	2017-03-13 13:47:24.535064	101.101.92	1514	TLSv1.2	1514	Application Data[TCP segment of a reassembled PDU]
176	0.000005	2017-03-13 13:47:24.535069	101.101.92	1514	TLSv1.2	1514	Application Data[TCP segment of a reassembled PDU]
177	0.000000	2017-03-13 13:47:24.535069	101.101.92	1514	TLSv1.2	1514	Application Data[TCP segment of a reassembled PDU]
178	0.000001	2017-03-13 13:47:24.535070	101.101.92	1514	TLSv1.2	1514	Application Data[TCP segment of a reassembled PDU]
179	0.000000	2017-03-13 13:47:24.535158	172.17.0.67	54	TCP	54	64763->443 [ACK] Seq=1464 Ack=29501 Win=66304 Len=0
180	0.000218	2017-03-13 13:47:24.535368	10.10.1.92	734	TLSv1.2	734	Application Data
181	0.000030	2017-03-13 13:47:24.575998	172.17.0.67	54	TCP	54	64763->443 [ACK] Seq=1464 Ack=30181 Win=65536 Len=0
182	112.820237	2017-03-13 13:49:17.396235	172.17.0.67	85	TLSv1.2	85	Encrypted Alert
183	0.000000	2017-03-13 13:49:17.396235	172.17.0.67	60	TCP	60	443->64753 [FIN, ACK] Seq=5816 Ack=825 Win=16768 Len=0
184	0.000112	2017-03-13 13:49:17.396347	10.10.1.92	54	TCP	54	64753->443 [ACK] Seq=825 Ack=5817 Win=66304 Len=0
185	0.710015	2017-03-13 13:49:18.106362	172.17.0.67	85	TLSv1.2	85	Encrypted Alert
186	0.000000	2017-03-13 13:49:18.106362	172.17.0.67	60	TCP	60	443->64750 [FIN, ACK] Seq=72540 Ack=2190 Win=22144 Len=0
187	0.000162	2017-03-13 13:49:18.106524	10.10.1.92	54	TCP	54	64750->443 [ACK] Seq=2190 Ack=72541 Win=66304 Len=0
188	2.119819	2017-03-13 13:49:20.226343	172.17.0.67	85	TLSv1.2	85	Encrypted Alert
189	0.000000	2017-03-13 13:49:20.226343	172.17.0.67	60	TCP	60	443->64760 [FIN, ACK] Seq=7399 Ack=1800 Win=19968 Len=0
190	0.000145	2017-03-13 13:49:20.226488	10.10.1.92	54	TCP	54	64760->443 [ACK] Seq=1800 Ack=7400 Win=65280 Len=0
191	4.235593	2017-03-13 13:49:24.466081	172.17.0.67	85	TLSv1.2	85	Encrypted Alert
192	0.000247	2017-03-13 13:49:24.466328	172.17.0.67	60	TCP	60	443->64763 [FIN, ACK] Seq=30212 Ack=1464 Win=18944 Len=0
193	0.000047	2017-03-13 13:49:24.466375	10.10.1.92	54	TCP	54	64763->443 [ACK] Seq=1464 Ack=30213 Win=65536 Len=0
194	14.502747	2017-03-13 13:49:39.415122	172.17.0.67	54	TCP	54	64763->443 [RST, ACK] Seq=1464 Ack=30213 Win=0 Len=0
195	0.007549	2017-03-13 13:49:39.522671	172.17.0.67	54	TCP	54	64760->443 [RST, ACK] Seq=1800 Ack=7400 Win=0 Len=0
196	0.007056	2017-03-13 13:49:39.429727	172.17.0.67	54	TCP	54	64750->443 [RST, ACK] Seq=2190 Ack=72541 Win=0 Len=0
197	0.008887	2017-03-13 13:49:39.438614	172.17.0.67	54	TCP	54	64753->443 [RST, ACK] Seq=825 Ack=5817 Win=0 Len=0
198	38.637317	2017-03-13 13:50:18.075931	172.17.0.67	74	TCP	74	64808->443 [SYN] Seq=0 Win=64240 Mss=1460 WS=256 S...
199	0.032138	2017-03-13 13:50:18.108069	10.10.1.92	66	TCP	66	443->64808 [SYN, ACK] Seq=0 Ack=1 Win=14608 Len=0 MSS=14...

使用 wireshark 功能 Follow TCP Stream(追踪 TCP 流)观察条有问题的流报文

The image shows a Wireshark packet capture window titled 'tcp stream eq 3'. It displays a list of packets with columns for No., Time, Source, Destination, Protocol, Length, and Info. The packets are numbered 158 to 194. Packet 181 is an 'Encrypted Alert' (85 bytes) from 10.13.49.24 to 10.13.49.24. Packet 191 is a 'FIN, ACK' (60 bytes) from 10.13.49.24 to 10.13.49.24. Packet 193 is an 'ACK' (54 bytes) from 10.13.49.24 to 10.13.49.24. Packet 194 is an 'RST, ACK' (54 bytes) from 10.13.49.24 to 10.13.49.24. The 'Info' column for packet 181 shows '85 Encrypted Alert'. The 'Info' column for packet 191 shows '60 443->64763 [FIN, ACK] Seq=30212 Ack=1464 Win=18944 Len=0'. The 'Info' column for packet 193 shows '54 64763->443 [ACK] Seq=1464 Ack=30213 Win=65536 Len=0'. The 'Info' column for packet 194 shows '54 64763->443 [RST, ACK] Seq=1464 Ack=30213 Win=0 Len=0'.

No.	Time	Source	Destination	Protocol	Length	Info
158	0.052890	10.13.47.24	10.13.47.24	TLSv1.2	1483	Application Data
159	0.000244	10.13.47.24	10.13.47.24	TLSv1.2	1483	Application Data
160	0.000006	10.13.47.24	10.13.47.24	TLSv1.2	1483	Application Data
161	0.000037	10.13.47.24	10.13.47.24	TCP	54	64763->443 [ACK] Seq=1464 Ack=10738 Win=66304 Len=0
162	0.000196	10.13.47.24	10.13.47.24	TLSv1.2	1483	Application Data
163	0.000001	10.13.47.24	10.13.47.24	TLSv1.2	1483	Application Data
164	0.000026	10.13.47.24	10.13.47.24	TCP	54	64763->443 [ACK] Seq=1464 Ack=13596 Win=66304 Len=0
165	0.000191	10.13.47.24	10.13.47.24	TLSv1.2	1483	Application Data
166	0.000000	10.13.47.24	10.13.47.24	TLSv1.2	1483	Application Data
167	0.000026	10.13.47.24	10.13.47.24	TCP	54	64763->443 [ACK] Seq=1464 Ack=16454 Win=66304 Len=0
168	0.000186	10.13.47.24	10.13.47.24	TLSv1.2	1483	Application Data
169	0.000015	10.13.47.24	10.13.47.24	TCP	54	64763->443 [ACK] Seq=1464 Ack=17883 Win=64768 Len=0
170	0.000628	10.13.47.24	10.13.47.24	TLSv1.2	1483	Application Data
171	0.000219	10.13.47.24	10.13.47.24	TLSv1.2	1483	Application Data
172	0.000021	10.13.47.24	10.13.47.24	TCP	54	64763->443 [ACK] Seq=1464 Ack=20741 Win=66304 Len=0
173	0.063815	10.13.47.24	10.13.47.24	TLSv1.2	1514	Application Data
174	0.000271	10.13.47.24	10.13.47.24	TLSv1.2	1514	Application Data[TCP segment of a reassembled PDU]
175	0.000001	10.13.47.24	10.13.47.24	TLSv1.2	1514	Application Data[TCP segment of a reassembled PDU]
176	0.000005	10.13.47.24	10.13.47.24	TLSv1.2	1514	Application Data[TCP segment of a reassembled PDU]
177	0.000000	10.13.47.24	10.13.47.24	TLSv1.2	1514	Application Data[TCP segment of a reassembled PDU]
178	0.000001	10.13.47.24	10.13.47.24	TLSv1.2	1514	Application Data[TCP segment of a reassembled PDU]
179	0.000000	10.13.47.24	10.13.47.24	TCP	54	64763->443 [ACK] Seq=1464 Ack=29501 Win=66304 Len=0
180	0.000218	10.13.47.24	10.13.47.24	TLSv1.2	734	Application Data
181	0.040630	10.13.47.24	10.13.47.24	TCP	54	64763->443 [ACK] Seq=1464 Ack=30181 Win=65536 Len=0
191	119.890083	10.13.49.24	10.13.49.24	TLSv1.2	85	Encrypted Alert
192	0.000247	10.13.49.24	10.13.49.24	TCP	60	443->64763 [FIN, ACK] Seq=30212 Ack=1464 Win=18944 Len=0
193	0.000047	10.13.49.24	10.13.49.24	TCP	54	64763->443 [ACK] Seq=1464 Ack=30213 Win=65536 Len=0
194	14.948747	10.13.49.39	10.13.49.24	TCP	54	64763->443 [RST, ACK] Seq=1464 Ack=30213 Win=0 Len=0

可以看到：

- A、其中 Encrypted Alert 与 FIN 是同时从服务端发送的
- B、Encrypted Alert 的报文与上一条报文的间隔时间为 119.89S
- C、No.193 的报文客户端马上发送了 ACK
- D、No.194 过了 14.948 客户端发送了一个 RST,ACK 报文，重置链接
- E、No.181 与 No.191 直接没有任何报文通信

四、问题定位

众所周知 WAF 是一个 Web 应用层面的防护产品，目前 WAF 的实现访问是使用 Tengine 做方向代理。即也会有会话保持时长限制；而 WAF 的会话保持时长为 120S，与 No.181 与 No.191 直接的间隔时间非常相近；建议用户根据[链接](#)内容进行调整，然后观察。

五、附录

1. 关于 SSL-Encrypted Alert 提示。

SSL 通信在断开连接时均为发送 Encrypted Alert 信息给客户端告知要关闭 ssl 会话了，同步会发生 FIN,ACK 报文从 TCP 层面断开链接：

Since we are already **in** an encrypted connection, the only way **to** really know what is being sent within packets is **to** make Wireshark **or** similar tools aware of the keys used **in** the transmission. Even though this is possible, I think **for** the purpose of this analysis it is enough **to** know that the **client** sends an alert message when the **connection** is asked **to** be closed actively by the **client** **or** server. The **type** of this Alert message should be CloseNotify (type 0), but we won't be able **to** see it **from** the **raw** data. **In** this case, the **client** is the sender of the following Alert message:

Secure Sockets Layer

TLSv1.2 Record Layer: Encrypted Alert

Content Type: Alert (21)

Version: TLS 1.2 (0x0303)

Length: 26

Alert Message: Encrypted Alert

2. 关于会话超时时间

`proxy_read_timeout 60s;`

默认: 60s

配在: http 中、server 中、location 中

在将请求发送给 upstream 的 server 后, 后端 server 就会回传数据, 这个时间是两次收取数据的时间差, 不是整个的接收时间。比如说负载大、网络卡, 在第 1 次收到请求的数据时断了, 然后过了 60s 后才收到后面的数据, 这两个时间差(其实就是两次 read 的时间差)超过了设置的 60s, `tengine (nginx)` 就会超时报错, 我当前走的是默认设置 60s。

Defines a timeout **for** reading a response **from** the proxied server. The timeout is set only between two successive read operations, **not for** the transmission of the whole response. **If** the proxied **server** does **not** transmit anything within this time, the **connection** is closed.

某业务间接性获取不到数据

一、引言

问题往往出现在意想不到的地方；问题分析时，我们需要更多的数据作为支撑。

二、问题现象

某天晚上用户反馈业务经过高防产品后间接性的无法获取到数据；判断是高防的问题导致的，由于是线上业务需要我们紧急处理。

三、第一优先级恢复业务

用户反馈由于源站证书问题，无法进行域名解析回源的。当时懵了，还有这种操作；随后的排查：

1. 请用户处理源站证书，全部处理正确；随后全部域名解析回源。
2. 分析高防 4 层&7 层日志 是否有拦截情况。

四、分析高防日志

在高防的 4 层和 7 层日志上，没有任何的异常返回码以及拦截记录。即高防没有啥问题。

五、源站处理

用户将源站证书处理正常；将域名全部解析回源，但是依然发现问题。

反馈的业务架构为：Client->cms..cn(slb-ECS)->content.x.cn

其中 b.com 是在高防上的。但是均已经解析回源负载均衡了依然有问题，苦于联系不到用户的开发同学；与客户达成共识暂定白天继续分析。

六、深入分析

1. 业务恢复

白天开发操作，由原来的请求访问 b.com 修改为内网负载均衡之后，问题现象消失。需要我们配合查询原因，同时不认可是应用问题；判断的依据是真正的通信域名一直在高防上。

2. Review 业务结构

针对这个情况拉上用户的运维，开发，我们的同学与用户对齐信息；了解到如下信息用户的运维和开发是 2 个部门，操作完全是分开的。晚上操作的 b.com 回源其实没有效果，因为当时 cms.x.cn(slb-ECS)访问的域名是 content.x.com；content.x.com 这个域名一直解析在高防上没有解析走过。

3. Review 分析过程

根据用户的数据：

cms.x.cn(slb-ECS)这 5 个 ECS 的公网地址是需要大量的访问 content.x.com 这个域名的。

海外SLB实例详情

实例信息

监听信息

操作日志

请输入过滤内容

监听ID

协议

前端端口

后端端口

slb-cn-xxxxxx000

https

443

80

slb-cn-xxxxxx001

https

443

80

slb-cn-xxxxxx002

https

443

80

slb-cn-xxxxxx003

http

80

80

slb-cn-xxxxxx004

http

80

80

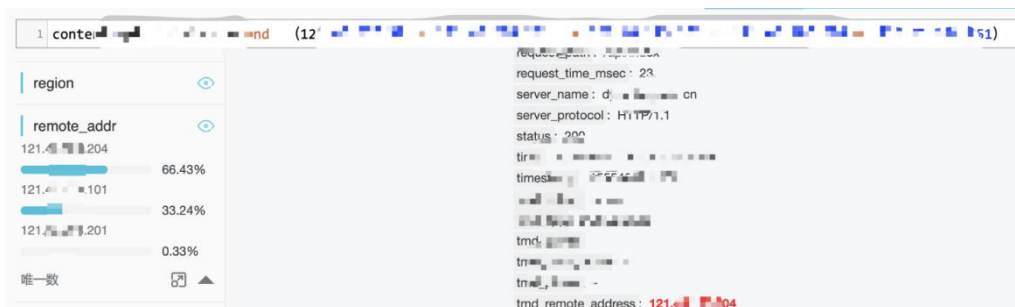
后端服务器/健康检查

请输入过滤内容

IP 地址	后端接口	健康检查状态	健康检查配置时间	健康检查历史日志	权重	Site Id	VPC Tunnel ID	ECS ID	创建时间	修改时间	健康检查变化趋势
10.10.10.89	80	正常	2018-07-18 10:06	点击查看	100	0	非VPC实例	无	2018-07-18 10:06	2018-07-18 10:06	查看
10.10.10.199	80	正常	2018-05-24 10:03	点击查看	100	0	非VPC实例	无	2018-05-24 10:03	2018-05-24 10:03	查看
10.10.10.252	80	正常	2018-05-23 10:50	点击查看	100	0	非VPC实例	无	2018-05-23 10:50	2018-05-23 10:50	查看
10.10.10.180	80	正常	2018-08-22 10:13	点击查看	100	0	非VPC实例	无	2018-08-22 10:13	2018-08-22 10:13	查看
10.10.10.798	80	正常	2018-03-11 10:25	点击查看	100	0	非VPC实例	无	2018-03-11 10:25	2018-03-11 10:25	查看

4. 分析高防 7 层日志

分析过程中发现共 5 台 ECS 的公网 IP，但是在高防的 7 层日志里只有 3 台的数据量是正常的；其他 1 台没有日志，1 台非常少。这个数据是符合间接性的失败的情况。



5. 分析高防 4 层日志

分析过程中发现用户的 5 台 ECS 的公网 IP，在 4 层日志都是有的，且量还不小。

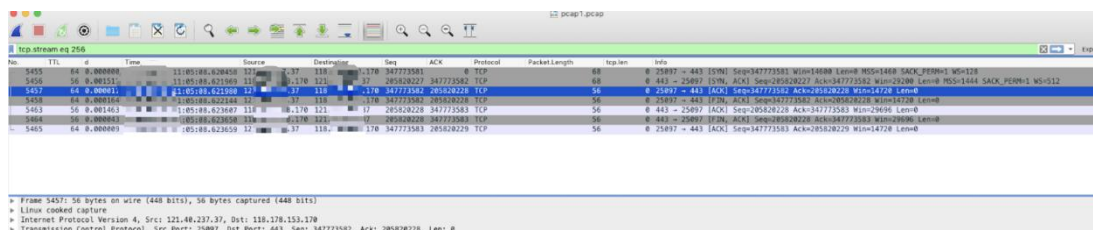
6. 分析初步结论

开始认为可能为高防的 VIP 到 7 层代理之间，或者 7 层代理的问题。但是经过与开发同学的讨论：

1. LB 流的日志是正常的。所以当时的网络以及 Tengine 不太可能有问题。
用户是走的 https 协议，那么三次握手完成之后，应该是要进行 ssl 握手。
2. 最终判断比较大的可能是由于当时 ssl 握手的时候，出现了问题。

7. 验证问题

将其中一台请求基本没有的 ECS 的链接地址修改为高防/SLB，然后在该 ECS 上进行抓包分析。看到的抓包如下，肯定完成三次握手后，没有进行 ssl 握手的动作直接断开了链接；这个行为是不符合预期的。



随后用户对比正常与不正常的服务配置。定位到是由于 php 的配置文件中，没有开启模块 extension=php_openssl.dll；开启后问题现象消失。

Linux 系统漏洞修复失败解析

一、漏洞修复过程

1. 在控制台点击漏洞修复按钮
2. 云安全中心控制中心将修复的命令下发到 ECS 中 Agent ([agent 介绍](#))
3. Agent 运行修复命令等待运行结果
4. Agent 返回运行结果，并进行检测漏洞是否修复
5. 如果为内核漏洞会提醒需要重启，如果为软件漏洞显示为修复。

二、漏洞修复的依赖

1. yum 源的配置

可能存在的问题：

1. yum 源配置错误，导致修复失败。
2. yum 源配置地址网络不可达；比如配置了 yum 源为 ipv6 的，但是 ECS 本身没有启用 ipv6,导致网络无法联通。
3. 配置的第三方 yum 源，内容更新不及时，不存在需要的软件包，导致修复失败。
4. yum 源配置为海外的，由于网络传输过慢，导致修复超时。

2. yum 命令可用性

yum 本身命令需要能够正常运行，yum 命令是依赖 python 的，所以 python 本身也需要保持正常；可能存在的问题：

```
1) yum 命令运行报错: sre_constants.error: bad character range
2) yum 缓存状态不对，比如: Repodata is over 2 weeks old. Install yum-cron? Or run:
yum makecache fast
```

3. 系统限制

可能存在的问题：

- 1) 系统文件打开数: Too many open files: '///var/lib/yum/rpmdb-indexes/obsoletes.tmp'
- 2) 无权限运行: cannot open Packages database in /var/lib/rpm
- 3) DNS 配置: yum 源更新时一般是使用域名进行访问的, 如果 dns 配置错误, 使域名无法正常解析出正确的 IP, 会导致连接超时、连接失败等。14: PYCURL ERROR 6 - "Couldn't resolve host 'www.xxx.com' "
- 4) 系统磁盘空间不够了, 比如 Could not create lock at /var/run/yum.pid: [Errno 28] No space left on device

三、常见问题修复

1. 使用 yum 命令报 “File "/usr/bin/yum", line 30” 的错误
2. 在使用 yum 安装软件时提示 “The requested URL returned error: 404”
3. 使用 yum 安装时出现 “error: unpacking of archive failed on file /usr/bin/xxx
x: cpio: open Failed” 报错
4. linux 更新软件源为阿里云
5. Linux 安装软件-dkpg 安装中断问题



云技术服务大学
云产品干货高频分享



云技术服务课堂
和大牛零距离沟通



阿里云开发者“藏经阁”
海量电子书免费下载