## **1.关于广告劫持目前能想到的方案有两个**：

1. 在内核的netfilter上挂钩子函数，对于来往的http报文进行修改。这种方式实现难度太大，大多数http报文都是经过gzip压缩过的，没办法在里面插入内容，同时很多报文都是分片报文，修改难度大，一不小心就可能导致网页打不开甚至更严重的问题。好处就是效率比较高。
2. 在路由器上加入web服务器做透明代理，对用户与web服务器之间的交互做中转，同时在中间插入我们的广告内容。这种方式可靠性比较高，不会出现网页打不开之类的大问题，同时可以灵活的对各个网站采取不同的措施，控制更加灵活方便。不过由于工作在用户态，需要把报文从内核收上来修改后再发送，效率可能比较低。

做透明代理的话一般使用nginx，比较小而且替换功能强大。下面主要介绍nginx方案。

## 2.nginx

### 2.1 ngin工作原理

Nginx由内核和模块组成，其中，内核的设计非常微小和简洁，完成的工作也非常简单，仅仅通过查找配置文件将客户端请求映射到一个location block（location是Nginx配置中的一个指令，用于URL匹配），而在这个location中所配置的每个指令将会启动不同的模块去完成相应的工作。

Nginx的模块从结构上分为核心模块、基础模块和第三方模块：

核心模块：HTTP模块、EVENT模块和MAIL模块

基础模块：HTTP Access模块、HTTP FastCGI模块、HTTP Proxy模块和HTTP Rewrite模块，

第三方模块：HTTP Upstream Request Hash模块、Notice模块和HTTP Access Key模块。

用户根据自己的需要开发的模块都属于第三方模块。正是有了这么多模块的支撑，Nginx的功能才会如此强大。

Nginx的模块从功能上分为如下三类。

Handlers（处理器模块）。此类模块直接处理请求，并进行输出内容和修改headers信息等操作。Handlers处理器模块一般只能有一个。

Filters （过滤器模块）。此类模块主要对其他处理器模块输出的内容进行修改操作，最后由Nginx输出。

Proxies （代理类模块）。此类模块是Nginx的HTTP Upstream之类的模块，这些模块主要与后端一些服务比如FastCGI等进行交互，实现服务代理和负载均衡等功能。

图1-1展示了Nginx模块常规的HTTP请求和响应的过程。

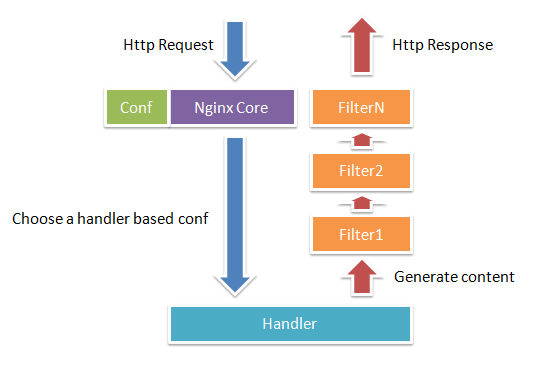


图1-1展示了Nginx模块常规的HTTP请求和响应的过程。

Nginx本身做的工作实际很少，当它接到一个HTTP请求时，它仅仅是通过查找配置文件将此次请求映射到一个location block，而此location中所配置的各个指令则会启动不同的模块去完成工作，因此模块可以看做Nginx真正的劳动工作者。通常一个location中的指令会涉及一个handler模块和多个filter模块（当然，多个location可以复用同一个模块）。handler模块负责处理请求，完成响应内容的生成，而filter模块对响应内容进行处理。

Nginx的模块直接被编译进Nginx，因此属于静态编译方式。启动Nginx后，Nginx的模块被自动加载，不像Apache，首先将模块编译为一个so文件，然后在配置文件中指定是否进行加载。在解析配置文件时，Nginx的每个模块都有可能去处理某个请求，但是同一个处理请求只能由一个模块来完成。

在工作方式上，Nginx分为单工作进程和多工作进程两种模式。在单工作进程模式下，除主进程外，还有一个工作进程，工作进程是单线程的；在多工作进程模式下，每个工作进程包含多个线程。Nginx默认为单工作进程模式。

Nginx支持如下处理连接的方法（I/O复用方法），这些方法可以通过use指令指定。

* select – 标准方法。 如果当前平台没有更有效的方法，它是编译时默认的方法。你可以使用配置参数 –with-select\_module 和 –without-select\_module 来启用或禁用这个模块。
* poll – 标准方法。 如果当前平台没有更有效的方法，它是编译时默认的方法。你可以使用配置参数 –with-poll\_module 和 –without-poll\_module 来启用或禁用这个模块。
* kqueue – 高效的方法，使用于 FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0 和 MacOS X. 使用双处理器的MacOS X系统使用kqueue可能会造成内核崩溃。
* epoll – 高效的方法，使用于Linux内核2.6版本及以后的系统。在某些发行版本中，如SuSE 8.2, 有让2.4版本的内核支持epoll的补丁。
* rtsig – 可执行的实时信号，使用于Linux内核版本2.2.19以后的系统。默认情况下整个系统中不能出现大于1024个POSIX实时(排队)信号。这种情况 对于高负载的服务器来说是低效的；所以有必要通过调节内核参数 /proc/sys/kernel/rtsig-max 来增加队列的大小。可是从Linux内核版本2.6.6-mm2开始， 这个参数就不再使用了，并且对于每个进程有一个独立的信号队列，这个队列的大小可以用 RLIMIT\_SIGPENDING 参数调节。当这个队列过于拥塞，nginx就放弃它并且开始使用 poll 方法来处理连接直到恢复正常。
* /dev/poll – 高效的方法，使用于 Solaris 7 11/99+, HP/UX 11.22+ (eventport), IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+.
* eventport – 高效的方法，使用于 Solaris 10. 为了防止出现内核崩溃的问题， 有必要安装[这个](http://sunsolve.sun.com/search/document.do?assetkey=1-26-102485-1) 安全补丁。

### nginx广告劫持方案

方案如下：

1. 在iptables中设置nat端口转发，将通过路由器的80端口要转发的http报文上送。

iptables -t nat -A PREROUTING -p tcp -m tcp --dport 80 -j DNAT \  
--to-destination lan口地址:nginx绑定的端口

1. 使用nginx在路由器上开启透明代理，劫持http流量，利用nginx的替换功能在http源文件中插入js广告代码。

location / {

proxy\_pass $scheme://$host$request\_uri;

proxy\_set\_header Host $http\_host;

sub\_filter </body> ' js广告代码</body>';

sub\_filter\_types \*;

sub\_filter\_once on;

}

1. 不同的网站可能有不同的广告策略。nginx能对url进行识别，可以针对不同的url做不同的处理。

附nginx的url匹配规则：

location匹配命令

~ #波浪线表示执行一个正则匹配，区分大小写

~\* #表示执行一个正则匹配，不区分大小写

^~ #^~表示普通字符匹配，如果该选项匹配，只匹配该选项，不匹配别的选项，一般用来匹配目录

= #进行普通字符精确匹配

@ #"@" 定义一个命名的 location，使用在内部定向时，例如 error\_page, try\_files

location 匹配的优先级(与location在配置文件中的顺序无关)

= 精确匹配会第一个被处理。如果发现精确匹配，nginx停止搜索其他匹配。

普通字符匹配，正则表达式规则和长的块规则将被优先和查询匹配，也就是说如果该项匹配还需去看有没有正则表达式匹配和更长的匹配。

^~ 则只匹配该规则，nginx停止搜索其他匹配，否则nginx会继续处理其他location指令。

最后匹配理带有"~"和"~\*"的指令，如果找到相应的匹配，则nginx停止搜索其他匹配；当没有正则表达式或者没有正则表达式被匹配的情况下，那么匹配程度最高的逐字匹配指令会被使用。

location 优先级官方文档

1.Directives with the = prefix that match the query exactly. If found, searching stops.

2.All remaining directives with conventional strings, longest match first. If this match used the ^~ prefix, searching stops.

3.Regular expressions, in order of definition in the configuration file.

4.If #3 yielded a match, that result is used. Else the match from #2 is used.

1.=前缀的指令严格匹配这个查询。如果找到，停止搜索。

2.所有剩下的常规字符串，最长的匹配。如果这个匹配使用^〜前缀，搜索停止。

3.正则表达式，在配置文件中定义的顺序。

4.如果第3条规则产生匹配的话，结果被使用。否则，如同从第2条规则被使用。

例如

location = / {

# 只匹配"/".

[ configuration A ]

}

location / {

# 匹配任何请求，因为所有请求都是以"/"开始

# 但是更长字符匹配或者正则表达式匹配会优先匹配

[ configuration B ]

}

location ^~ /images/ {

# 匹配任何以 /images/ 开始的请求，并停止匹配 其它location

[ configuration C ]

}

location ~\* .(gif|jpg|jpeg)$ {

# 匹配以 gif, jpg, or jpeg结尾的请求.

# 但是所有 /images/ 目录的请求将由 [Configuration C]处理.

[ configuration D ]

}请求URI例子:

•/ -> 符合configuration A

•/documents/document.html -> 符合configuration B

•/images/1.gif -> 符合configuration C

•/documents/1.jpg ->符合 configuration D