NGINX、HTTPGUARD 与 FAIL2BAN
技术架构说明书

撰写：NGX.HK
版本：v1.0
日期：2018 年 7 月 21 日

一、 目录

二、 架构与应用环境 3

1. 应用环境 3

2. 架构图 3

3. 防护逻辑 4

三、 验证测试 5

4. 测试环境 5

5. 测试环境架构 5

6. 测试过程 6

四、 生产环境 9

7. 部署前准备 9

8. 准备操作系统 9

9. 安装 NGINX 11

10. NGINX 预配置 13

11. HttpGuard 配置文件 17

12. HttpGuard 配置建议 19

13. HttpGuard 测试 21

14. Fail2ban 23

五、 结语 29

二、 架构与应用环境

1. 应用环境

该架构主要实现抵御 CC 攻击的功能，选用 nginx 作为高性能反向代理器，在后端应用服务器前实现防护功能，该架构无法抵御 DDOS 攻击与任何渗透攻击。

目前市面上的 CDN 服务绝大部分都不包含抗 CC 攻击的服务或者属于付费服务，当防护量急剧增大的情况下及其容易导致费用暴涨，而该架构也适用于服务器处于 CDN 之后的主机，也支持部署在负载均衡器之后，但需要确认各个节点是否能正常传递“x-forwarded-for”这个 HTTP header，如果无法正常传递，则不适用此防护架构。

2. 架构图

下图为该架构的简单示意图：

3. 防护逻辑

CC 攻击一般有以下几个特点：

- 同一 IP 的连接数突然增大或持续保持在高位；
- 不断访问同一 URL 或加入随机查询字段；
- 不支持 301、302 或 js 跳转；
- 不支持传递 cookie。

因为前端服务无法分辨 URL 的有效性，所以无法通过 URL 判断访客的善恶与否。所以针对第二条，在这里不做判断，也不针对其进行拦截。

在攻击开始之初，iptables 内并没有拦截规则，所有 IP 均可通过 iptables 到达 nginx。当攻击开始后，大量请求到达 nginx，HTTP 请求次数急剧增大，当某个 IP 的请求书超过 HttpGuard 的阈值后，HttpGuard 将对其启用设定的防护动作。如果该 IP 无法通过 HttpGuard 的测试，将通知 nginx 返还指定的 HTTP Code。

而 Fail2Ban 一直在背后实时读取 nginx 访问日志，通过设定的正则匹配规则，寻找包含指定 HTTP Code 的 nginx 访问日志条目，而后提取该条目中的访客 IP 地址，通过预设的命令将其加入 iptables 中。

当已被加入 iptables 的 IP 再次访问服务器时，将被 iptables 阻挡，因为请求未能到达 nginx，所以 nginx 的负载也会逐渐下降，从而使服务保持正常。

HttpGuard 的防护测试正是利用 CC 攻击的特点，可以选用或同时启用以下测试功能：

- 验证码验证；
- 302 跳转验证；

- Js 跳转验证；
- Cookie 传递验证。

三、 验证测试

4. 测试环境

- 操作系统：Centos 7.5
- Nginx 版本：1.15.1
- HttpGuard：https://github.com/centos-bz/HttpGuard
- Fail2Ban 版本：0.9.7

5. 测试环境架构

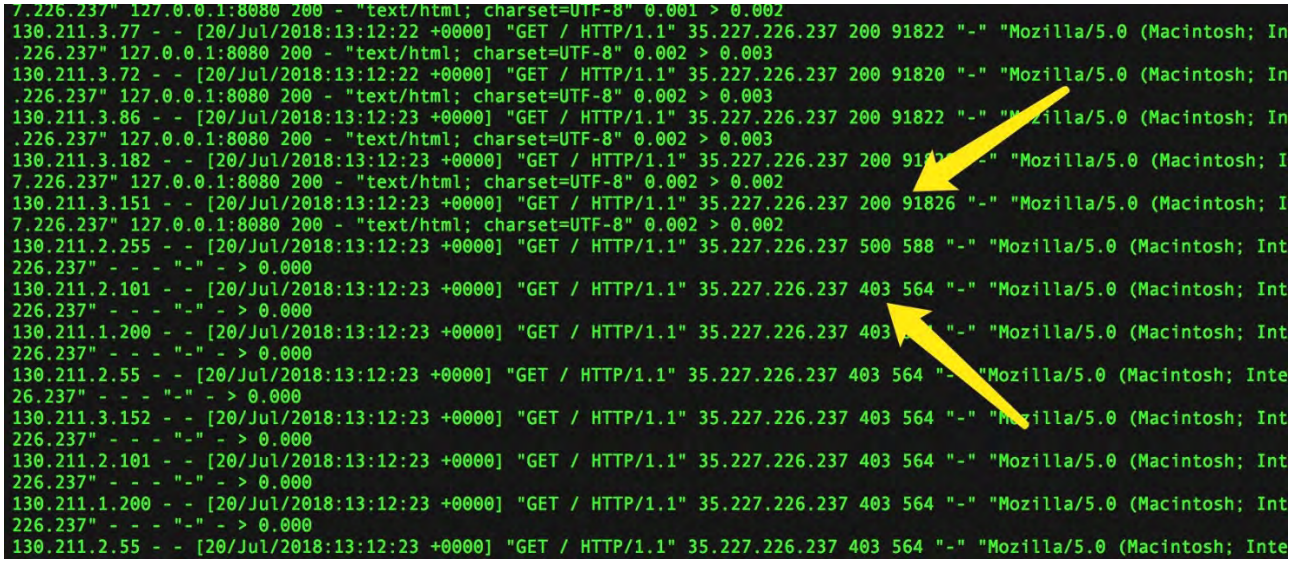
使用 10 台 Google Cloud 主机模拟攻击肉鸡，攻击 Google CDN 节点，而该 Google CDN 节点反向代理 Nginx-Main 这台主机。

攻击肉鸡使用 Apache ab 命令模拟高并发环境；使用 GoldenEye 模拟 CC 攻击；Google CDN 节点并未做任何防护，仅仅用于流量分发；Nginx-Main 运行本防御架构与 PHP-FPM 并运行着一个 vhost，该 vhost 托管着一个 phpinfo 页面用于测试。

6. 测试过程

测试过程分为两部分，首先关闭 HttpGuard，使用 10 台攻击偷鸡分别使用 Apache ab 与 GoldenEye 对 Google CDN 进行攻击，；而后启用 HttpGuard，再次实施攻击。

测试环境配置完成后，在预测试时确认功能正常并能实现预设的效果。Nginx 访问日志截图如下：

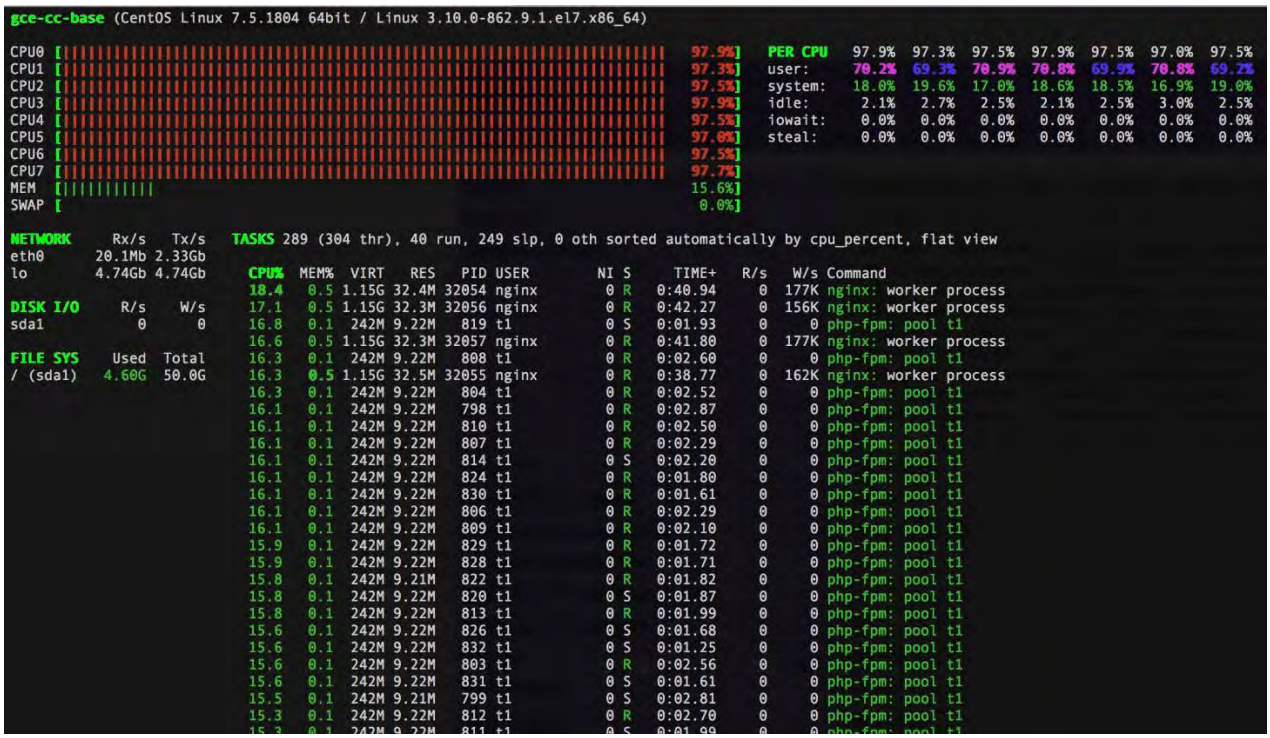


当访客正常访问时，Nginx 返还了 200，而随着访问量持续上涨，使得 HttpGuard 动作，Nginx 返还了 403。而两种状态转换之间的 500 是因为 Nginx 中断了与 proxy server 的通讯导致的。

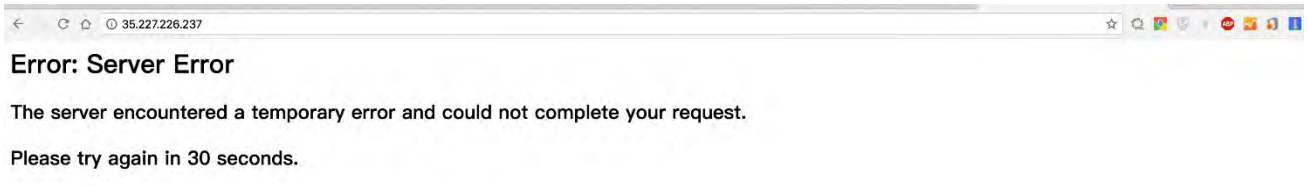
在关闭 HttpGuard 的状态下，Google CDN 转发到 Nginx-Main 服务器中的流量高达 2Gbps，而此时的连接数在 3 万左右：

```
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
3
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
20510
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
13752
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
30946
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
32140
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
32953
```

同时 PHP-FPM 也使得 Nginx-Main 服务器的 CPU 处于满负荷状态：



经过多次压力测试，在其中一次测试中把 Nginx-Main 中的 Nginx-Main 线程打崩溃，但随后主进程自动重建，服务经历短暂不可用后恢复正常：

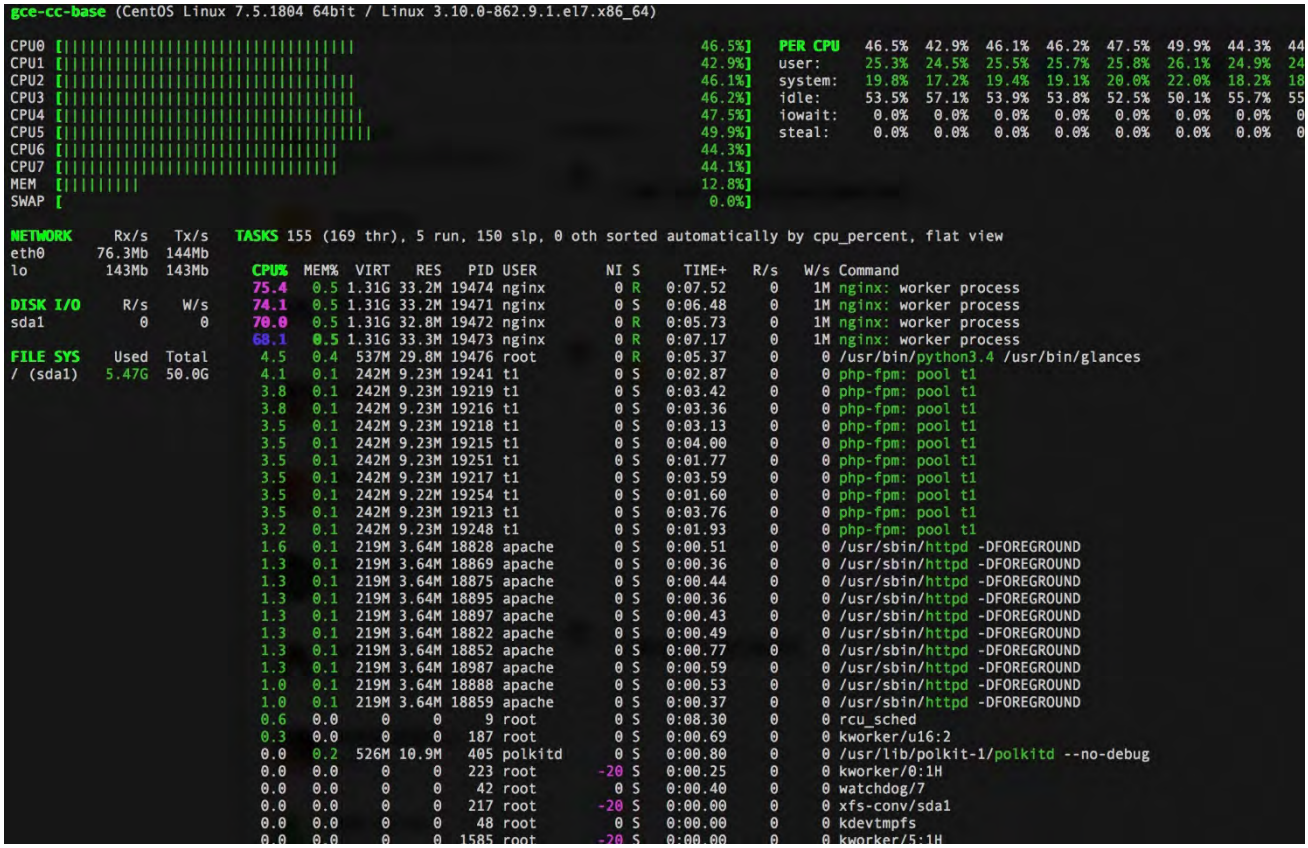


完成对照测试后，重新配置 HttpGuard 并启用，再重新进行测试，经检测，连

接数保持在 3 万左右：

```
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
3
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
40383
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
41049
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
41841
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
39898
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
27265
[[root@gce-cc-base HttpGuard]# netstat -anp | grep 80 | grep ESTABLISHED | wc -l
48924
```

在启用 HttpGuard 后，因为 Nginx 自动对恶意请求返还 403，请求并未到达后端的 PHP-FPM，所以 CPU 负载较未启用 HttpGuard 前有很大改善：



与此同时，因为无需返还 HTML 数据，所以带宽的使用也维持在 150Mbps 以下。但因 Nginx 需要处理的内容较未启用 HttpGuard 前要多，所以相应地，Nginx 的 CPU 使用率也有所上升。

四、 生产环境

7. 部署前准备

在部署前有一些参数与内容需要提前确认。首先是业务正常运行期间，单 IP 的连接数与请求数均值；确定需要对恶意访客返还的 HTTP Code；确定检测到多少特定的 HTTP Code 后需要该访客的 IP 使用 iptables 屏蔽等。

因为需要用到第三方模块,所以建议自行编译 Nginx 并选择最新版本的 Nginx。而对操作系统则无特定要求，但建议使用 Centos 7.5。

8. 准备操作系统

在这里，我是用 Centos 7.5，系统信息如下：

```
[root@ngx-main ~]# lsb_release  -a
LSB Version:  :core-4.1-amd64:core-4.1-noarch:cxx-4.1-amd64:cxx-4.1-noarch:desktop-4.1-amd64:desktop-4.1-noarch:languages-4.1-amd64:languages-4.1-noarch:printing-4.1-amd64:printing-4.1-noarch
Distributor ID: CentOS
Description:  CentOS Linux release 7.5.1804 (Core)
Release:  7.5.1804
Codename:  Core
```

本说明不涉及系统安全方面的设置，为方便配置，将关闭 selinux：

```
# 打开 selinux 配置文件
# 将 SELINUX= enforcing 修改为 SELINUX=disabled
# 完成后保存退出
[root@ngx-main ~]# vim /etc/selinux/config
```

我的测试网络中不支持 IPv6，为此，我将通过修改引导文件的方式关闭 IPv6：

```
# 打开默认的 grub 文件
[root@ngx-main ~]# vim /etc/default/grub
```

```
# 在 GRUB_CMDLINE_LINUX 变量中添加以下内容
ipv6.disable=1

# 例如：
GRUB_CMDLINE_LINUX="ipv6.disable=1 rd.lvm.lv=centos_t1/root rd.lvm.lv=centos_t1/swap rhgb quiet"

# 完成后保存退出，并执行以下命令
[root@ngx-main ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

为提高内核性能，建议针对内核进行调整，以便提高对 TCP 队列的处理能力：

```
# 打开 sysctl.conf 文件
[root@ngx-main ~]# vim /etc/sysctl.conf

# 在文件最后追加以下内容
#修改超时时间为 30 秒，某些情况可以进一步降低该值
net.ipv4.tcp_fin_timeout = 30
#将 keepalive 的发送频率降低为 20 分钟一次
net.ipv4.tcp_keepalive_time = 1200
#开启 SYN Cookies，当 SYN 队列溢出时启用 Cookies
net.ipv4.tcp_syncookies = 1
#开启 TIME-WAIT sockets 重用功能
net.ipv4.tcp_tw_reuse = 1
#开启 TIME-WAIT sockets 快速回收功能
net.ipv4.tcp_tw_recycle = 1
#加大 SYN 队列
net.ipv4.tcp_max_syn_backlog = 8192
#最大 TIME_WAIT 保持数，超过将全部清除
net.ipv4.tcp_max_tw_buckets = 5000
```

完成修改后，重启系统，以便让修改生效。而后重新通过 SSH 登入并执行以下命令，检查修改是否生效：

```
# 检查 selinux 状态
[root@ngx-main ~]# sestatus
SELinux status:                 disabled

# 检查是否存在 IPv6 地址
[root@ngx-main ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:50:56:b6:12:a5 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.122/24 brd 10.1.1.255 scope global noprefixroute dynamic ens192
       valid_lft 6433sec preferred_lft 6433sec
```

9. 安装 NGINX

在这里我选用主线 Nginx 的最新版本：1.15.1，可以通过以下地址下载：

- https://nginx.org/en/download.html

在下载前建议新建一个目录，用于存放源码：

```
# 新建文件夹
[root@ngx-main ~]# mkdir -p codex/nginx

# 进入相应的文件夹
[root@ngx-main ~]# cd codex/nginx/

# 下载文件
[root@ngx-main nginx]# wget https://nginx.org/download/nginx-1.15.1.tar.gz

# 解压文件
[root@ngx-main nginx]# tar zxvf nginx-1.15.1.tar.gz

# 克隆 lua-nginx-module
[root@ngx-main nginx]# git clone https://github.com/openresty/lua-nginx-module.git
```

因为需要用到 LUA，所以在编译之前需要安装 LUA 的链接库：

```
# 安装 LUA 链接库
[root@ngx-main nginx]# yum install lua-devel -y
```

然后使用以下命令进行配置（configure）、编译并安装：

```
# 进入 Nginx 源码目录
[root@ngx-main nginx]# cd /root/codex/nginx/nginx-1.15.1/

# configure
```

```
[root@ngx-main nginx-1.15.1]#  ./configure --prefix=/usr/local/nginx --sbin-path=/usr/sbin/nginx --conf-path=/usr/local/nginx/nginx.conf --pid-path=/var/run/nginx.pid --error-log-path=/var/log/nginx/error.log --http-log-path=/var/log/nginx/access.log --lock-path=/var/lock/nginx.lock --http-client-body-temp-path=/var/tmp/nginx/client_body --http-proxy-temp-path=/var/tmp/nginx/proxy --http-fastcgi-temp-path=/var/tmp/nginx/fastcgi --http-uwsgi-temp-path=/var/tmp/nginx/uwsgi --http-scgi-temp-path=/var/tmp/nginx/scgi --with-http_gunzip_module --with-pcre --with-pcre-jit --with-http_perl_module --with-ld-opt="-Wl,-E" --with-http_realip_module --with-http_secure_link_module --with-http_slice_module --with-http_ssl_module --with-http_stub_status_module --with-http_sub_module --with-http_v2_module --with-http_addition_module --with-http_xslt_module --with-http_image_filter_module --with-http_geoip_module --with-http_dav_module --with-http_flv_module --with-http_mp4_module --with-http_gzip_static_module --with-http_auth_request_module --with-http_random_index_module --with-select_module --with-poll_module --with-file-aio --with-http_degradation_module --with-libatomic      --add-module=../lua-nginx-module

# 编译
[root@ngx-main nginx-1.15.1]# make

#安装
[root@ngx-main nginx-1.15.1]# make install
```

编译完成后执行以下命令测试 Nginx 是否成功编译安装：

```
# 检查 Nginx 版本与编译参数
[root@ngx-main nginx-1.15.1]# nginx -V
nginx version: nginx/1.15.1
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-28) (GCC)
built with OpenSSL 1.0.2k-fips  26 Jan 2017
TLS SNI support enabled
configure arguments: --prefix=/usr/local/nginx --sbin-path=/usr/sbin/nginx --conf-path=/usr/local/nginx/nginx.conf --pid-path=/var/run/nginx.pid --error-log-path=/var/log/nginx/error.log --http-log-path=/var/log/nginx/access.log --lock-path=/var/lock/nginx.lock --http-client-body-temp-path=/var/tmp/nginx/client_body --http-proxy-temp-path=/var/tmp/nginx/proxy --http-fastcgi-temp-path=/var/tmp/nginx/fastcgi --http-uwsgi-temp-path=/var/tmp/nginx/uwsgi --http-scgi-temp-path=/var/tmp/nginx/scgi --with-http_gunzip_module --with-pcre --with-pcre-jit --with-http_perl_module --with-ld-opt=-Wl,-E --with-http_realip_module --with-http_secure_link_module --with-http_slice_module --with-http_ssl_module --with-http_stub_status_module --with-http_sub_module --with-http_v2_module --with-http_addition_module --with-http_xslt_module --with-http_image_filter_module --with-http_geoip_module --with-http_dav_module --with-http_flv_module --with-http_mp4_module --with-http_gzip_static_module --with-http_auth_request_module --with-http_random_index_module --with-select_module --with-poll_module --with-file-aio --with-http_degradation_module --with-libatomic --add-module=../lua-nginx-module
```

确认编译安装成功后即可进入 Nginx 目录克隆 HttpGuard：

```
# 进入 nginx 目录
[root@ngx-main nginx-1.15.1]# cd /usr/local/nginx/
```

```
# 克隆 HttpGuard
[root@ngx-main nginx]# git clone https://github.com/centos-bz/HttpGuard.git
```

此时，Nginx 目录中有以下文件与目录：

```
# 列出 Nginx 目录下的文件与目录
[root@ngx-main nginx]# ll -h
total 68K
-rw-r--r-- 1 root root 1.1K Jul 21 19:04 fastcgi.conf
-rw-r--r-- 1 root root 1.1K Jul 21 19:04 fastcgi.conf.default
-rw-r--r-- 1 root root 1007 Jul 21 19:04 fastcgi_params
-rw-r--r-- 1 root root 1007 Jul 21 19:04 fastcgi_params.default
drwxr-xr-x 2 root root   40 Jul 21 19:04 html
drwxr-xr-x 7 root root  163 Jul 21 19:09 HttpGuard
-rw-r--r-- 1 root root 2.8K Jul 21 19:04 koi-utf
-rw-r--r-- 1 root root 2.2K Jul 21 19:04 koi-win
-rw-r--r-- 1 root root 5.2K Jul 21 19:04 mime.types
-rw-r--r-- 1 root root 5.2K Jul 21 19:04 mime.types.default
-rw-r--r-- 1 root root 2.6K Jul 21 19:04 nginx.conf
-rw-r--r-- 1 root root 2.6K Jul 21 19:04 nginx.conf.default
-rw-r--r-- 1 root root  636 Jul 21 19:04 scgi_params
-rw-r--r-- 1 root root  636 Jul 21 19:04 scgi_params.default
-rw-r--r-- 1 root root  664 Jul 21 19:04 uwsgi_params
-rw-r--r-- 1 root root  664 Jul 21 19:04 uwsgi_params.default
-rw-r--r-- 1 root root 3.6K Jul 21 19:04 win-utf
```

10. NGINX 预配置

首先需要修改 HttpGuard 的配置内容：

```
lua_package_path "/usr/local/nginx/HttpGuard/?.lua";
lua_shared_dict guard_dict 100m;
lua_shared_dict dict_captcha 70m;
init_by_lua_file '/usr/local/nginx/HttpGuard/init.lua';
access_by_lua_file '/usr/local/nginx/HttpGuard/runtime.lua';
lua_max_running_timers 1;
```

请注意将“lua_package_path”、“init_by_lua_file”与“access_by_lua_file”中

的路径修改为正确的文件路径。

而后将以上内容添加至“nginx.conf”文件中的“http”块中，如：

```
#user  nobody;
worker_processes  1;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;


events {
    worker_connections  1024;
}


http {
    lua_package_path "/usr/local/nginx/HttpGuard/?.lua";
    lua_shared_dict guard_dict 100m;
    lua_shared_dict dict_captcha 70m;
    init_by_lua_file '/usr/local/nginx/HttpGuard/init.lua';
    access_by_lua_file '/usr/local/nginx/HttpGuard/runtime.lua';
    lua_max_running_timers 1;

    include       mime.types;
    default_type  application/octet-stream;
```

然后需要修改 HttpGuard 目录下的“config.lua”文件：

```
# 打开 config.lua 文件
[root@ngx-main nginx]# vim HttpGuard/config.lua


# 修改第二行的 baseDir 内容
baseDir = '/usr/local/nginx/HttpGuard/'
```

至此，即可完成 nginx 的配置，再次尝试测试 Nginx 的配置文件：

```
[root@ngx-main nginx]# nginx -t
nginx: the configuration file /usr/local/nginx/nginx.conf syntax is ok
nginx: [emerg] mkdir() "/var/tmp/nginx/client_body" failed (2: No such file or directory)
nginx: configuration file /usr/local/nginx/nginx.conf test failed
```

出现“nginx: [emerg] mkdir() "/var/tmp/nginx/client_body" failed (2: No such file or directory)”是因为 Nginx 权限在“tmp”目录下新建名为“nginx”的目录，

此时需手动建立：

```
[root@ngx-main nginx]# mkdir /var/tmp/nginx
```

随后再次测试 Nginx 配置文件：

```
[root@ngx-main nginx]# nginx -t
nginx: the configuration file /usr/local/nginx/nginx.conf syntax is ok
ls: cannot access /usr/local/nginx/HttpGuard/captcha/*.png: No such file or directory
nginx: configuration file /usr/local/nginx/nginx.conf test is successful
```

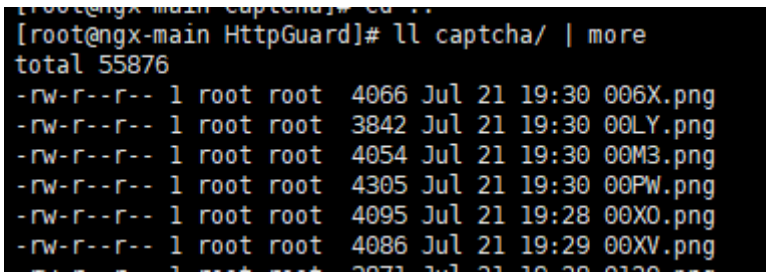
出现“ls: cannot access /usr/local/nginx/HttpGuard/captcha/*.png: No such file or directory”是因为在 HttpGuard 的配置文件中，对恶意 IP 的测试设置为“captcha”，因为尚未生成验证码图片，所以出现以上错误。

如果不计划启用验证码，则可以修改配置文件并将“blockAction”的值修改为“forbidden”，此时将返还 444 这个 HTTP Code 给恶意访客。

如果计划启用验证码，则需要安装 PHP，并使用 PHP 脚本生成验证码图片：

```
# 安装 PHP
[root@ngx-main captcha]# yum install php php-gd –y

# 生成验证码图片
[root@ngx-main captcha]# php /usr/local/nginx/HttpGuard/captcha/getImg.php
```

生成验证码图片需要用到 GD 库，所以需要安装“php-gd”，生成验证码图片所需要的时间较长，请耐心等待。完成后即可发现 HttpGuard 目录下的“captcha”目录新建了许多验证码图片：



此时再测试 Nginx 的配置文件：

```
[root@ngx-main HttpGuard]# nginx -t
nginx: the configuration file /usr/local/nginx/nginx.conf syntax is ok
nginx: configuration file /usr/local/nginx/nginx.conf test is successful
```

可以发现已经通过了测试，而后直接运行 “nginx” 即可启动 nginx：

```
# 启动 Nginx
[root@ngx-main HttpGuard]# nginx

# 检查监听情况
[root@ngx-main HttpGuard]# netstat -anp | grep nginx
tcp        0      0 0.0.0.0:80          0.0.0.0:*          LISTEN      12570/nginx: master
unix  3      [ ]         STREAM     CONNECTED     84432    12570/nginx: master
unix  3      [ ]         STREAM     CONNECTED     84433    12570/nginx: master
```

然后通过浏览器检查 Nginx 是否处于正常服务状态：

最后检查日志是否正常：

```
[root@ngx-main HttpGuard]# tail -f -n 1 /var/log/nginx/access.log
10.1.2.73 - - [21/Jul/2018:19:41:19 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:23 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
10.1.2.73 - - [21/Jul/2018:19:41:24 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
```

11. HttpGuard 配置文件

HttpGuard 的配置文件在 HttpGuard 目录下，名为“config.lua”的文件。

配置文件中主要的配置信息如下：

- keyDefine：定义 keySecret，用于生成验证的 token；
- limitReqModules：限制请求模块：
  - state：控制模块的开关；
  - amongTime：计数时间；
  - maxReqs：在计数时间内的最大请求数，超过将启用验证功能；
  - urlProtect：需要保护的 URL 列表，支持正则；
- redirectModules：302 响应头跳转模块：
  - state：控制模块的开关；
  - verifyMaxFail：最大失败次数，超过将启用验证功能；
  - keySecret：用于生成 token 的 key，token 将添加在 URL 中返还给访客，如果访客跳转的 URL 中没有该 token，则判定为验证失败；
  - amongTime：计数时间；
  - urlProtect：需要保护的 URL 列表，支持正则；
- JsJumpModules：发送 js 跳转代码模块：
  - state：控制模块的开关；
  - verifyMaxFail：最大失败次数，超过将启用验证功能；
  - keySecret：用于生成 token 的 key，token 将添加在 js 中返还给访客，如果访客跳转的 URL 中没有该 token，则判定为验证失败；

- amongTime：计数时间；
- urlProtect：需要保护的 URL 列表，支持正则；

- cookieModules：发送 cookie 验证模块：
  - state：控制模块的开关；
  - verifyMaxFail：最大失败次数，超过将启用验证功能；
  - keySecret：用于生成 token 的 key，token 将添加在 cookie 中返还给访客，如果访客传递的 cookie 中没有该 token，则判定为验证失败；
  - amongTime：计数时间；
  - urlProtect：需要保护的 URL 列表，支持正则；
- autoEnable：自动开启主动防御模块
  - state：控制模块的开关；
  - protectPort：监控的端口；
  - interval：计数时间；
  - normalTimes：连续巡查发现正常次数；
  - exceedTimes：连续巡查发现异常次数；
  - maxConnection：最大连接数；
  - ssCommand：ss 命令的可执行文件的绝对路径；
  - enableModule：需要自动启动的模块；
- captchaKey：当 keyDefine 设置为 static 时，需要手动配置此项；
- blockAction：默认的拦截动作，可设置为 captcha、forbidden 与 iptables；
- captchaToIptables：IP 在黑名单中且默认拦截动作为 captcha 时可触发

iptables 拦截功能：

- state：控制模块的开关；
- maxReqs：在计数时间内的最大请求数，将启用 iptables 封禁；
- amongTime：计数时间；

- sudoPass：调用 iptables 时所需要的 sudo 密码；
- blockTime：封禁时间；
- whiteTime：通过验证后的白名单时间；
- keyExpire：token 的 key 失效时间；
- urlMatchMode：匹配 url 模式：
  - requestUri：url-protect 目录下的正则匹配的是原始地址并可带参数；
  - uri：url-protect 目录下的正则匹配的是已 decode 的地址且不可带参数；
- captchaPage：验证码页面路径；
- reCaptchaPage：输入验证码错误时显示的页面路径；
- whiteIpModules：白名单 ip 文件，支持正则；
- realIpFromHeader：用于设定获取真实访客 IP 的 HTTP header；
- captchaDir：验证码图片目录；
- debug：用于开启除错模式；
- logPath：指定日志路径。

12. HttpGuard 配置建议

默认情况下，所有防御模块都应该被关闭，因为这有可能为真实访客带来影响，

而任何跳转模块都不利于 SEO。如果线上服务处于 24 小时有人值守的状态，建议在紧急情况下按需手动启用防御模块。

当然，也可以保持在启动状态，但不建议将跳转模块设为启动状态，而仅仅将限制请求模块（limitReqModules）启动即可，例如：

```
limitReqModules = { state = "On" , maxReqs = 200 , amongTime = 60, urlProtect = baseDir.."url-protect/limit.txt" },
```

以上配置的意义为：在 60 秒内，访客访问的 URL 如果匹配“limit.txt”中的正则且请求次数超过 200 次，则调用拦截动作（blockAction)。

建议根据实际情况调整“amongTime”与“maxReqs”的数值，“maxReqs”的值设置在业务繁忙期请求数均值的 80%会比较好。

需要注意的是，限制请求模块（limitReqModules）所统计的并不是连接数，而是请求数，且不区分 vhost 域名与端口，将对真个 Nginx 服务生效。

与此同时，建议配置自动开启防御模块功能（autoEnable)，实现自动启动防御模块。例如：

- 监控 80 端口，自动开启 cookieModules：

```
autoEnable = { state = "On", protectPort = "80", interval = 10, normalTimes = 3,exceedTimes = 2,maxConnection = 50, ssCommand = "/usr/sbin/ss" ,enableModule = "cookieModules"},
```

以上配置的意义为：调用 ss 命令每 10 秒检查一次 80 端口的连接数，如果连续 2 次超过 50，则启动 cookieModules，如果连续 3 次小于 50，则关闭 cookieModules。

同样的，可以采用同样的配置方法监听其他端口并启用防御模块：

```
autoEnable = { state = "On", protectPort = "80", interval = 10, normalTimes = 3,exceedTimes = 2,maxConnection = 100, ssCommand = "/usr/sbin/ss" ,enableModule = "redirectModules"},
autoEnable = { state = "On", protectPort = "443", interval = 10, normalTimes = 3,exceedTimes = 2,maxConnection = 50, ssCommand = "/usr/sbin/ss" ,enableModule = "cookieModules"},
```

```
autoEnable = { state = "On", protectPort = "443", interval = 10, normalTimes = 3,exceedTimes = 2,maxConnection = 100, ssCommand = "/usr/sbin/ss" ,enableModule = "redirectModules"},
```

这样配置对搜索引擎比较友好，不需要一直启用跳转。另外需要注意的是，自动开启防御模块功能（autoEnable）并不监控 Nginx 的请求数，而是通过 ss 命令监控系统的连接数。

如果服务器处于负载均衡器或 CDN 之后，必须要启用 realIpFromHeader 功能，以便从“x-forwarded-for”HTTP header 中获取真实的用户 IP，否则将导致负载均衡器或 CDN 节点的 IP 被拦截甚至屏蔽。

如果 blockAction 设置为 forbidden，那么在多次验证失败后，Nginx 将返还 444 给访客，强制访客断开连接。如果需要修改为其他 HTTP Code，则需要修改“guard.lua”文件：

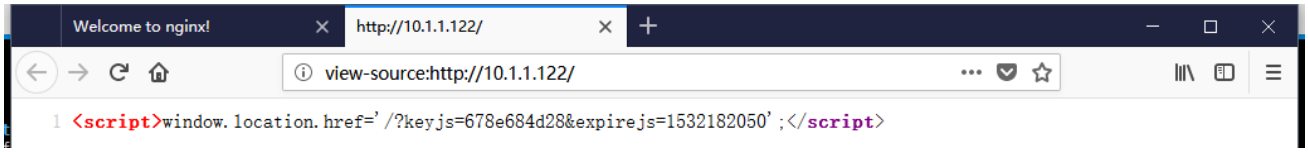
```
# 打开文件
[root@pub-ngx HttpGuard]# vim guard.lua

# 修改 561 行中的内容
--拒绝访问动作
function Guard:forbiddenAction()
                ngx.header.content_type = "text/html"
                ngx.exit(444)
end
```

13. HttpGuard 测试

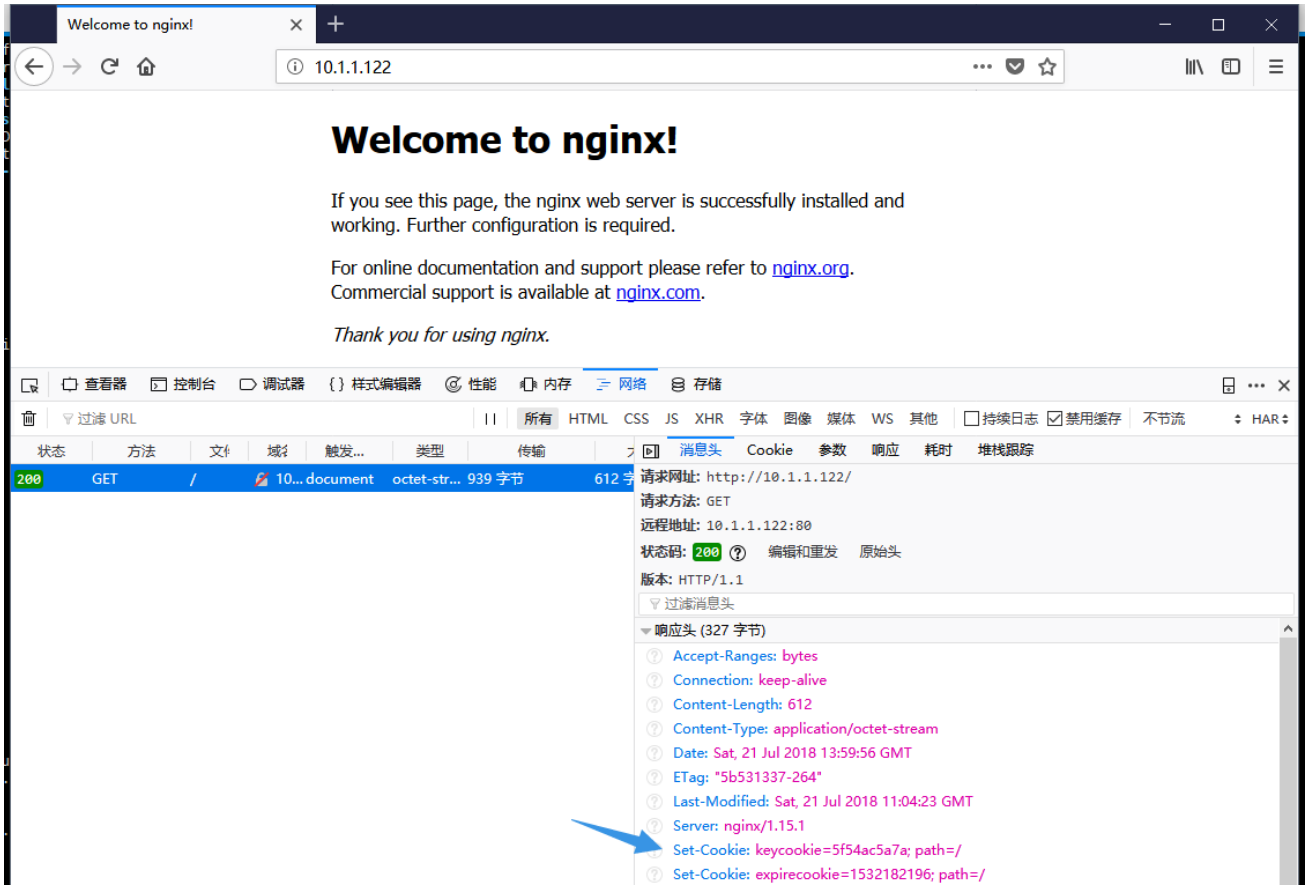
配置完成后，将 blockAction 设置为“captcha”，然后持续刷新页面，将出现验证码验证页面：

当启用 302 响应头跳转模块后，多次刷新页面将出现 302 跳转的响应：

当启用 js 跳转代码模块，多次刷新页面将出现带有 js 代码的跳转页面：



当启用 cookie 验证模块时，多次刷新页面将出现“Set-Cookie”的响应头：



14. Fail2ban

为了实现 IP 地址的自动封禁，建议使用功能较为成熟高效的 fail2ban。在 Centos 系统下，通过以下命令即可完成安装：

```
[root@ngx-main ~]# yum install fail2ban
```

Fail2ban 是通过正则匹配文本日志中的条目，从匹配的条目中找出指定的 IP 地址，而后调用相应的命令，对 IP 进行相应的动作。

在这里主要用 iptables 对 IP 实施丢包的动作。

首先需要准备好正则匹配的语句，以下是 Nginx 的默认日志格式：

```
10.1.2.73 - - [21/Jul/2018:21:59:56 +0800] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0"
```

在这里我需要匹配所有 HTTP 响应码为 444 的日志条目，并且提取出第一个字段的 IP 地址。简单的正则如下：

```
<HOST> - - \[.*\] \".*\" (403|444) \d*\s\".*
```

IP 地址的匹配正则用“<HOST>”代替，fail2ban 将自动提取该 IP 地址，为了防止日志的其他地方也出现 444 等字符，务必将规则写得更加详细，防止匹配错误的日志条目。

而后进入 fail2ban 的程序目录并复制一份配置文件用于配置软件：

```
# 进入相应的目录
[root@ngx-main ~]# cd /etc/fail2ban/


# 复制配置文件
[root@ngx-main fail2ban]# cp jail.conf jail.local
```

因为后缀名为“local”的配置文件的优先级高于后缀名为“conf”的配置文件，所以无需将“jail.conf”删除。

而后再“jail.local”文件的最后追加以下内容：

```
[ngx-log-40x]
enabled = true
port = http,https
filter = ngx-40x
logpath = /var/log/nginx/access.log
action     = %(banaction)s[name=%(__name__)s-tcp, port="%(port)s", protocol="tcp", chain="%(chain)s",
actname=%(banaction)s-tcp]
                %(mta)s-whois[name=%(__name__)s, dest="%(destemail)s"]
maxretry = 10
findtime = 1800
bantime = 86400
ignoreip = 127.0.0.1 10.1.1.122
```

第一行为该规则的名称，其余的配置含义如下：

- Enabled：是否启用该规则；
- Port：指定端口；
- Filter：指定筛选器；
- Logpath：需要检查的日志的绝对路径；
- Action：实施的动作；
- Maxretry：检测到的最大次数；
- Findtime：计数时长；
- Bantime：封禁时长；
- Ignoreip：忽略的 IP。

请务必将本地环回与本机上的其他 IP 地址加入“ignoreip”中，这些 IP 一旦被封禁，很有可能造成网络或系统异常。

上面的配置信息的意义是：使用筛选器（filter）中的正则实时匹配指定的日志文件（logpath），如果在指定的时间内（findtime）被匹配超过阈值（maxretry），则调用封禁动作（action）中的“ban”动作，直至封禁时间超过设定值（bantime），随后使用封禁动作（action）中的“unban”动作解封。

完成 jail 文件的修改后，在“filter.d”目录下建立名为“ngx-40x.conf”的筛选器配置文件：

```
# 新建文件
[root@ngx-main fail2ban]# vim filter.d/ngx-40x.conf

# 填入内容
[Definition]
failregex = <HOST> - - \[.*\] \".*\" (403|444) \d*\s\".*
ignoreregex =
```

将匹配的正则语法填写在“failregex”字段中即可，完成后使用以下命令测试该筛选器：

```
[root@ngx-main fail2ban]# fail2ban-regex /var/log/nginx/access.log /etc/fail2ban/filter.d/ngx-40x.conf
```

结果如下：

```
Running tests
=============

Use      failregex filter file : ngx-40x, basedir: /etc/fail2ban
Use           log file : /var/log/nginx/access.log
Use           encoding : UTF-8


Results
=======

Failregex: 1476 total
|-   #) [# of hits] regular expression
|    1) [1476] <HOST> - - \[.*\] \".*\" (403|444) \d*\s\".*
`-

Ignoreregex: 0 total

Date template hits:
|- [# of hits] date format
|   [2248] Day(?P<_sep>[-/])MON(?P=_sep)Year[ :]?24hour:Minute:Second(?:\.Microseconds)?(?: Zone offset)?
`-

Lines: 2248 lines, 0 ignored, 1476 matched, 772 missed
[processed in 0.46 sec]

Missed line(s): too many to print.   Use --print-all-missed to print all 772 lines
```

默认情况下，fail2ban 中的 iptables 动作为“reject”，该动作会主动通知访客请求被拒绝，该动作会给网络上行造成压力，所以建议改成“drop”。

打开以下文件并修改：

```
# 打开文件
[root@ngx-main fail2ban]# vim action.d/iptables-common.conf

# 修改 blocktype
blocktype = DROP
```

完成后将 fail2ban 设为开机启动并立即启动：

```
# 设为开机启动
[root@ngx-main fail2ban]# systemctl enable fail2ban.service
Created symlink from /etc/systemd/system/multi-user.target.wants/fail2ban.service to
/usr/lib/systemd/system/fail2ban.service.

# 立即启动
[root@ngx-main fail2ban]# systemctl start fail2ban.service

# 检查启动状态
[root@ngx-main fail2ban]# systemctl status fail2ban.service
● fail2ban.service - Fail2Ban Service
   Loaded: loaded (/usr/lib/systemd/system/fail2ban.service; enabled; vendor preset: disabled)
   Active: active (running) since Sat 2018-07-21 22:46:08 CST; 4s ago
     Docs: man:fail2ban(1)
  Process: 22368 ExecStart=/usr/bin/fail2ban-client -x start (code=exited, status=0/SUCCESS)
 Main PID: 22371 (fail2ban-server)
   CGroup: /system.slice/fail2ban.service
           └─22371 /usr/bin/python2 -s /usr/bin/fail2ban-server -s /var/run/fail2ban/fail2ban.sock -p
/var/run/fail2ban/fail2ban.pid -x -b

Jul 21 22:46:08 ngx-main systemd[1]: Starting Fail2Ban Service...
Jul 21 22:46:08 ngx-main fail2ban-client[22368]: 2018-07-21 22:46:08,674 fail2ban.server [22369]: INFO
Starting Fail2ban v0.9.7
Jul 21 22:46:08 ngx-main fail2ban-client[22368]: 2018-07-21 22:46:08,674 fail2ban.server [22369]: INFO
Starting in daemon mode
Jul 21 22:46:08 ngx-main systemd[1]: Started Fail2Ban Service.
```

因为 fail2ban 依赖 iptables 服务，所以请确认该服务是否处于正常运行的状态：

```
[root@ngx-main fail2ban]# systemctl status iptables.service
● iptables.service - IPv4 firewall with iptables
   Loaded: loaded (/usr/lib/systemd/system/iptables.service; disabled; vendor preset: disabled)
```

```
   Active: inactive (dead)
```

如果未安装或未运行，则需要手动安装或运行：

```
# 安装
[root@ngx-main fail2ban]# yum install iptables-services –y

# 设为开机启动
[root@ngx-main fail2ban]# systemctl enable iptables.service
Created symlink from /etc/systemd/system/basic.target.wants/iptables.service to
/usr/lib/systemd/system/iptables.service.

# 立即启动
[root@ngx-main fail2ban]# systemctl start iptables.service

# 重启 fail2ban
[root@ngx-main fail2ban]# systemctl restart fail2ban.service
```

此时检查 iptables 的规则可以发现：

```
[root@ngx-main fail2ban]# iptables -L -vn
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 f2b-ngx-log-40x-tcp  tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           multiport dports 80,443
  157 10923 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
    0     0 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0
    1    35 ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           state NEW tcp dpt:22
   29  2205 REJECT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           reject-with icmp-host-prohibited

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 REJECT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           reject-with icmp-host-prohibited

Chain OUTPUT (policy ACCEPT 6 packets, 658 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain f2b-ngx-log-40x-tcp (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0
```

随后多次刷新页面，可以发现测试用的访客 IP 已经被 iptables 封禁：

```
[root@ngx-main fail2ban]# iptables -L -vn --line
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1      992 69047 f2b-ngx-log-40x-tcp  tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           multiport dports 80,443
2      613 56877 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
3        1    84 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0
4        2    72 ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0
5        0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           state NEW tcp dpt:22

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 4 packets, 318 bytes)
num   pkts bytes target     prot opt in     out     source               destination

Chain f2b-ngx-log-40x-tcp (1 references)
num   pkts bytes target     prot opt in     out     source               destination
1       33  3806 DROP       all  --  *      *       10.1.2.73            0.0.0.0/0
2      959 65241 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0
```

然后通过以下命令检查 fail2ban 的封禁状态：

```
[root@ngx-main fail2ban]# fail2ban-client status ngx-log-40x
Status for the jail: ngx-log-40x
|- Filter
|  |- Currently failed:      0
|  |- Total failed:    30
|  `- File list:   /var/log/nginx/access.log
`- Actions
   |- Currently banned:  1
   |- Total banned: 1
   `- Banned IP list: 10.1.2.73
```

可以发现，jail 名称为“ngx-log-40x”的规则下共检测到 30 次匹配的条目，并且已经封禁了“10.1.2.73”这个 IP。

该 IP 将在 86400 秒后自动解封，如果需要手动解封，则可以使用以下命令：

```
[root@ngx-main fail2ban]# fail2ban-client set ngx-log-40x unbanip 10.1.2.73
10.1.2.73
```

五、 结语

Nginx、HttpGuard 与 fail2ban 三者配合使用对 CC 攻击的防御效果更好，HttpGuard 可以提供多种高效的检测方式，而 fail2ban 则提供实时监控日志文件的功能，不但可以调用 iptables，还可以调用 email 等实现提前预警。

无论何种跳转验证都会对搜索引擎的爬虫造成影响，绝大部分爬虫不会跟随跳转，尤其是 js 跳转，同时也不会传递 cookie。如果不是出于攻击的状态下，请不要启用相关的模块，以免对收录造成影响。