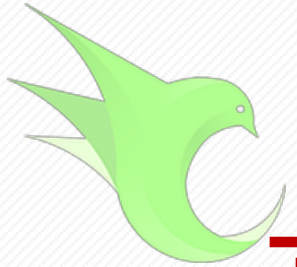
# CC攻击防护实践

基于OpenResty的CC攻击防护实践

https://github.com/starjun/openstar

周俊

# 为什么是 OpenResty



同 Nginx 一脉相承

高性能

成熟稳定

很多很多大型互联网公司都在用

社区友好

非常非常！！！

遇到问题，社区都积极解决处理了

OpenResty 最佳实践

易扩展

Nginx 可编程啦！不是C/C++

# 内容大纲

CC攻击概念

攻击分类

防护方法

其他

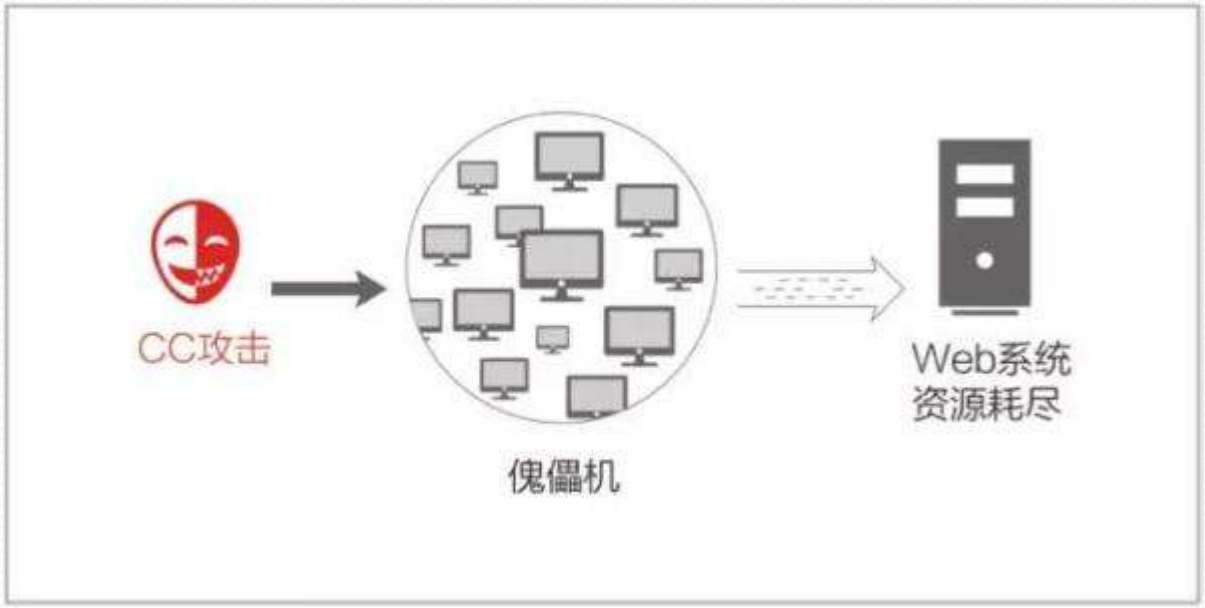
# PART 01

## CC攻击概念

什么是CC攻击?

# CC（挑战黑洞）

CC = Challenge Collapsar




1：来自7层的攻击（当时的 黑洞 设备仅仅可以防护4层攻击）
2：不断对网站发送连接请求
3：DDOS中的一种，攻击形式是发送http请求
4：带宽比4层的小太多太多

# PART 02

## CC攻击分类

CC攻击的形式是发送http请求，但仍然需要进行合理的分类

# 具体怎么分类?

基于攻击的方法:

GET/POST...

A

基于攻击消耗的资源:

CPU/数据库/连接数...

B

基于攻击的特征:

连接耗尽/慢速攻击...

C

基于被攻击的点:

URI/接口/验证码...

D

基于攻击方式:

直接/代理/分布式...

E

# 基于被攻击的点(我选择的维度):

| 序号 | 被攻击的点 | 说明 | EG |
| --- | --- | --- | --- |
| 1 | 可直接访问的URI | 直接暴露出来，被访问的地址 | 搜索请求[数据库查询]，高计算[cpu计算]等 |
| 2 | 浏览器AJAX接口 | 前端页面异步调用使用的接口 | ajax判断用户是否存在的uri[数据库查询]等 |
| 3 | 嵌入类型的URI | 前端页面嵌入引用的地址 | 嵌入的验证码url[CPU计算]等 |
| 4 | 非浏览器的接口 | 移动端访问使用到的地址 | 动态REST API接口等 |
| 5 | 特定类型攻击 | 慢速攻击(Slow headers/body/read)<br>PHP Multipart/form-data remote dos | 针对特定web server，特定语言/漏洞 等 |
| 6 | 随机URI | 随机地址 | 任意页面地址[连接数] |
| | | | |

# 黑客攻击寻找潜在的弱点

验证码生成：http://www.xxxx.com/captcha/new.html?height=38&width=160&font_size=20

全文搜索：http://blog.xxxx.net/?s=CC攻击

高计算：http://www.xxx.com/rsa?dec=xxxxxx

应用漏洞：php dos (代号69364) ; WordPress (CVE-2018-6389) ; Apache POI Dos (CVE-2017-5644)
OpenSSL DoS (CVE-2018-0739)...

连接耗尽：纯粹的高频率的连接请求

利用外部热门网站漏洞（XSS - DDOS/反弹），利用代理，利用肉鸡，直接攻击...

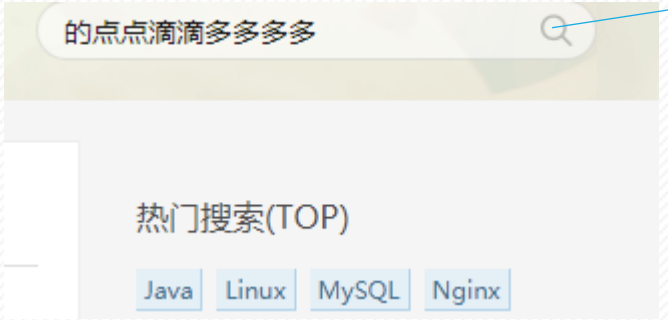
...


！！！ 异步判断是否注册过 [数据库操作]

！！！ 嵌入的验证码uri [CPU/内存消耗]

！！！ 短信发送接口 [短信炸弹]

！！！ 用户注册 [数据库操作]
RSA解密 [CPU/内存消耗]




！！！ 内容搜索操作 [数据库操作(全文索引)]

# PART 03

## 防护算法

针对CC攻击提供的防御手段（7层）

# OpenResty 实践

lua_shared_dict limit_ip_dict 100m;  用于频率计数器存放
lua_shared_dict config_dict 50m;  用于 配置规则的存放
lua_shared_dict ip_dict 50m;  用于 ip 黑白名单的存放
…

set_by_lua_file

rewrite_by_lua_file

content_by_lua_file  API操作使用

init_by_lua_file  初始化 配置规则 (从本地的JSON文件中读取到 config_dict 和 ip_dict 等 dict 中)

init_worker_by_lua_file 定时器操作(集群模式下,定时推送到redis/从redis中拉取配置/推送计数信息)
定时将config规则进行同步到各个worker中.

access_by_lua_file  核心的操作都在这里完成(规则匹配,动作拦截等)

header_filter_by_lua_file  拦截操作时,修改HTTP状态码等操作

body_filter_by_lua_file  该阶段一般用于 页面 hook 操作时使用(就是对返回页面进行内容替换操作)

log_by_lua_file  日志记录操作/一些常规计数操作

# 初级阶段：频率限制

ip频率限制：单位时间内访问次数超过阈值对ip进行拦截操作(验证码等...)

注意：

1：有触发条件

2：当在CDN后面时，需要从header头中获取用户ip

X-Forwarded-For/X-Real-IP 【自定义header头】

realIpFrom_Mod

```
"ic.test.com":{
    "ips":[["36.110.148.1/24","123.12.32.12/24"],"cidr"],
    "realipfrom":"X_Forwarded_For"
}
```

network_Mod

```
[
    {
        "state": "on",
        "id":"1-host_list",
        "network":{"maxReqs":30,"pTime":10,"blackTime":600},
        "hostname": [["101.200.122.200","localhost"],"list"],
        "uri": ["/api/time",""]
    },
    {
        "state": "on",
        "id":"2-test",
        "network":{"maxReqs":1000,"pTime":10,"blackTime":120},
        "hostname": ["*",""],
        "uri": ["*",""]
    }
]
```

频率限制进阶:

使用业务属性进行频率限制（session/cookie）

**! openstar 暂不支持（后续会增加）**

# OpenResty 实践

```lua
--- 访问频率检查 并且计数
-- _tb_network 频率规则  _uid 唯一标识
-- true:触发 频率限制  false:未触发 计数++
local function network_ck(_tb_network,_uid)
    if type(_tb_network) ~= "table" then return end
    local pTime = _tb_network.pTime or 10
    local maxReqs = _tb_network.maxReqs or 50
    local ip_count = limit_ip_dict:get(_uid)
    if ip_count == nil then
        limit_ip_dict:set(_uid,1,pTime)
        return
    else
        if ip_count >= maxReqs then
            return true
        else
            limit_ip_dict:incr(_uid,1)
            return
        end
    end
end
```

使用 limit_ip_dict 进行计数操作

效率更好的：lua-resty-lrucache
Worker之间不共享，暂时就没有使用这个API

# 中级阶段：算法防护

1：JS 跳转 √
[服务端：静态/动态验证]

2：SET COOKIE 跳转（302/307） √
[服务端：动态验证]

3：JS SET COOKIE √
[服务端：静态验证]

4：Meta 跳转 √

5：Iframe 嵌套 √

6：验证码认证 ×
[服务端：动态验证]

7：flash动作 ×
[服务端：动态验证]

8：强制静态缓存 √

9：请求参数（header/args/posts）算法验证 √
[服务端：验签/算法碰撞]

>1 app_Mod：rehtml/refile

```
<html>
```

>2 rewrite_Mod

```
[
    {
        "state": "on",
        "id":"1-test",
        "action": "set_cookie",
        "set_cookie":["asjldisdafpopliu8909jk34jk","diy_name"],
        "hostname": ["localhost",""],
        "uri": ["/rewrite",""]
    }
]

</script>
</head>
</html>
```

```
}

</script>
</head>
</html>
```

# OpenResty 实践

```
if v.action == "set_cookie" then
    local token = ngx.md5(v.set_cookie[1] .. ip)
    local token_name = v.set_cookie[2] or "token"
    -- 没有设置 tokenname 默认就是 token
    if ngx_var["cookie_"..token_name] ~= token then
        ngx.header["Set-Cookie"] = {token_name.."=" .. token}
        if method == "POST" then
            return ngx.redirect(request_uri,307)
        else
            return ngx.redirect(request_uri)
        end
    end
elseif v.action == "set_url" then
```

更加IP和配置的常量进行MD5

307 对应POST方法

这个是需要注意到，否则对业务功能都会有影响，浏览器会直接进行set cookie跳转，使用302方式跳转，原来的POST方法变成了GET方法，POST的数据同样都丢失了，不是我们想要的效果

# 前置页面的HOOK


http://xx/ajax/checktel?tel=18801180400

http://xx/lzcaptcha?key=$key

```
{
    "state": "on",
    "uri": ["/register.html",""],
    "hostname": ["www.abc.com",""],
    "replace_list":
        [
         ["<script type='text/javascript' src='#youjs_url'></script>","",
            "<script type='text/javascript' src='#'></script>\n<script type='text/javascript' src='my_jsurl'></script>"]
         ["allow","","allowPASS"],
         ["lzcaptcha\\?key='\\s*\\+ key","jio","lzcaptcha?key='+key+'&keytoken=@token@'"]
        ]
}
```

app_Mod  rehtml/refile.

# body_filter_by_lua_file

```
--- STEP 14
if config_is_on("replace_Mod") and action_tag == "" then
    local Replace_Mod = getDict_Config("replace_Mod")
    for _,v in ipairs(Replace_Mod) do
        if v.state =="on" and host_uri_remath(v.hostname,v.uri) then
            ngx.req.clear_header("Accept-Encoding")-- 取消浏览器要求gzip操作)
            next_ctx.replace_Mod = v
            --ngx_ctx.body_mod = v
            break
        end
    end
end
```

没有彻底解决!!! 当后端的服务器强制进行gzip了,不通过浏览器的标记判断时,
内容替换阶段就失效了!








**客户端 大有可为!**

JS:
鼠标轨迹验证 JS随机延迟 键盘鼠标事件验证 [浏览器方言]

计算浏览器指纹(长期积累指纹信誉库)

SDK:
Header头增加签名数据 [ 服务器验签有性能瓶颈 ]
增加碰撞算法字符串（md5($uri+$time+约定常量)）

IP:
交换ip信誉库

控件/浏览器:
展望：未来终端的控件/浏览器和WAF联动

# 小结

| | 频率控制 | JS跳转 | SET COOKIE | 验证码认证 | 强制静态缓存 | 参数算法验证 | JS SET COOKIE | Meta跳转 | Iframe嵌套 | flash验证 |
|---|---|---|---|---|---|---|---|---|---|---|
| openstar | network_Mod | app_Mod rehtml. | rewrite_Mod | | app_Mod rehtml. | app_Mod func. | app_Mod rehtml. | app_Mod rehtml. | app_Mod rehtml. | |
| 直接访问的URI | | | | | | | | | | |
| 浏览器AJAX接口 | | | | | | | | | | |
| 嵌入类型的URI | | | | | | | | | | |
| 非浏览器的REST接口 | | | | | | | | | | |
| 特定类型攻击 | | | | | | | | | | |
| 随机URI | | | | | | | | | | |
| 攻击软件情况 | 无法绕过 | 需要JS引擎<br>效率低<br>不易绕过 | 多数可绕过 | 需要识别验证码<br>效率低<br>不易绕过 | 无法绕过 | 需要分析算法<br>JS：分析难度中<br>SDK：分析难度高<br>不易绕过 | 需要分析JS<br>易绕过 | 需要解析HTML<br>多数容易绕过 | 需要分析<br>容易绕过 | 需要解析Flash<br>不易绕过 |
| 缺点 | 有触发条件，在海量IP分布式攻击时，效果要差一些 | 不够通用<br>影响业务 | 容易被绕过 | 不通用，影响体验 | 影响业务 | 算法变更成本高<br>SDK有业务入侵 | 不通用<br>影响业务 | 不通用 | 不通用 | 验证复杂<br>不通用<br>浏览器支持情况 |
| 优点 | 简单有效，实用 | 可以扩展（JS） | 效率高 | 可以扩展<br>静态/动态 验证码 | 效率高 | 效果好 | 可以扩展（JS） | 可以扩展（JS） | 可以扩展（JS） | 效果好 |
| 增加门槛 | 鼠标轨迹验证、JS随机延迟、键盘鼠标事件验证［浏览器方言］、浏览器指纹信誉库(风控) | | | | | | | | | |

使用CDN，隐藏服务器真实IP！！！
拦截国外所有IP（虽然粗暴、效果不错）

# 主动出击（题外话）

分析攻击IP

1：直接攻击

2：利用代理

有的代理是非匿名的

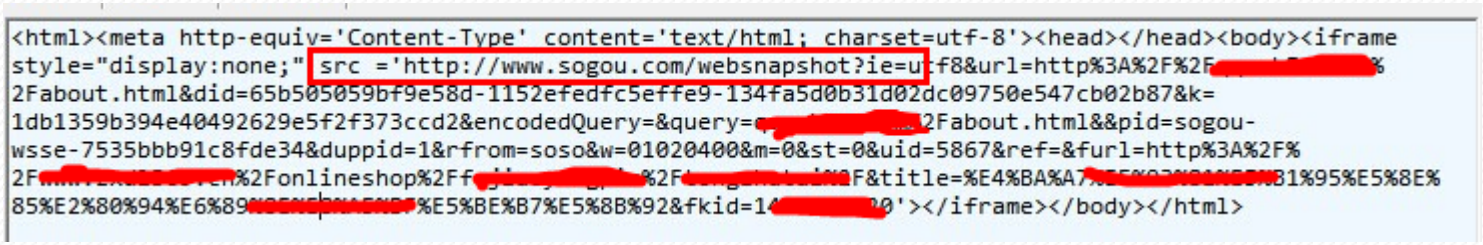
3：肉鸡攻击

反向入侵肉鸡服务器

扩展：

获取QQ号/手机号

利用sougou网页快照（失效）




xxx营销系统

# PART 04

## 其他

攻击软件、项目说明等...

# 日常压力测试工具

1：HULK (Http Unbearable Load
https://github.com/grafov/hull
python hulk.py http://www.cc

2：GoldenEye
https://github.com/jseidl/Gold
python goldeneye.py [-m] htt

…

```
GoldenEye v2.1 by Jan Seidl <

Hitting webserver in mode 'ge
571 GoldenEye strikes hit. (0
2693 GoldenEye strikes hit. (
4277 GoldenEye strikes hit. (
5277 GoldenEye strikes hit. (
5852 GoldenEye strikes hit. (
7065 GoldenEye strikes hit. (
9018 GoldenEye strikes hit. (
9542 GoldenEye strikes hit. (
9542 GoldenEye strikes hit. (
        Server may be DOWN!
```

| 工具名称 | 描述 | 攻击类型 |
| --- | --- | --- |
| LOIC | LOIC是一款专善于web应用程序的Dos/DDOS攻击工具，它可以用TCP数据包、UDP数据包、HTTP请求于对目标网站进行DDOS/DOS测试 | UDP/TCP/HTTP GET |
| Xoic | 和LOIC相比，工具主打的还是流量型攻击，不过相比前者增加了Testmode模式，可以测试攻击主机的性能 | TCP/UDP/ICPM/HTTP GET |
| HOIC | HOIC是High Orbit Ion Cannon(高轨道离子炮)的缩写，一款用RealBasic开发的DoS工具。 | HTTP GET |
| HULK | HULK是一种web的拒绝服务攻击工具。它能够在web服务器上产生许多单一的伪造流量，能绕开引擎的缓存，因此能够直接攻击服务器的资源池 | HTTP GET |
| glodeneye | 主打应用层（http flood）攻击的工具，从Hulk项目发展而来 | HTTP GET/POST |
| srDOS | 该工具在与服务端建立连接后，在等待服务端发送数据，然后才会发送自己的攻击报文，这也是为什么没有后续的攻击报文通过查看反汇编的代码发现，该工具里有https握手相关的攻击 | TCPSSL |
| 骷髅头 | 国人编写的4层DOS软件 | TCP/UDP/PING/碎片... |
| Slowhttptest | 它能发起诸如slowloris、Slow http post、slow read、slow range等工具实现的低带宽应用层拒绝服务攻击工具利用了http协议的一个特点 | HTTP |
| DirtyJumper | 通过botnet发动DDoS 攻击的toolkit | HTTP GET/POST syn flood |
| Darkddoser | 通过僵尸网络发起的http 攻击 | HTTP GET/POST/TCP/UDP |
| sslhamer | 模拟僵尸网络的攻击行为,与客户端建立三次握手之后，发送大量的垃圾报文 | TCP/SSL |
| hyenae | 灵活指定IP和发包速率，并且支持TCP/UDP/HTTP等多种协议;支持IPv6 | TCP/UDP/HTTP |
| killemall | 作者宣称可以绕过ADS设备的HTTP redirect、HTTP Cookie、Javascript、CAPTCHA（图片验证码攻击）防护算法该攻击工具对HTTP redirect、HTTP Cookie的绕过速度比较快；该攻击工具需要借助v8引擎才能绕过javascript防护 | HTTP |
| dossim | Ddosim可以模拟几个僵尸主机 (局域网内随机IP地址)。 | HTTP |
| pyloris | PyLoris是一个测试服务器的脆弱性和连接拒绝服务（DoS）攻击的脚本工具 | HTTP漏洞型 |

# Thank you