第七季:用nginx搭建waf防火墙的实践

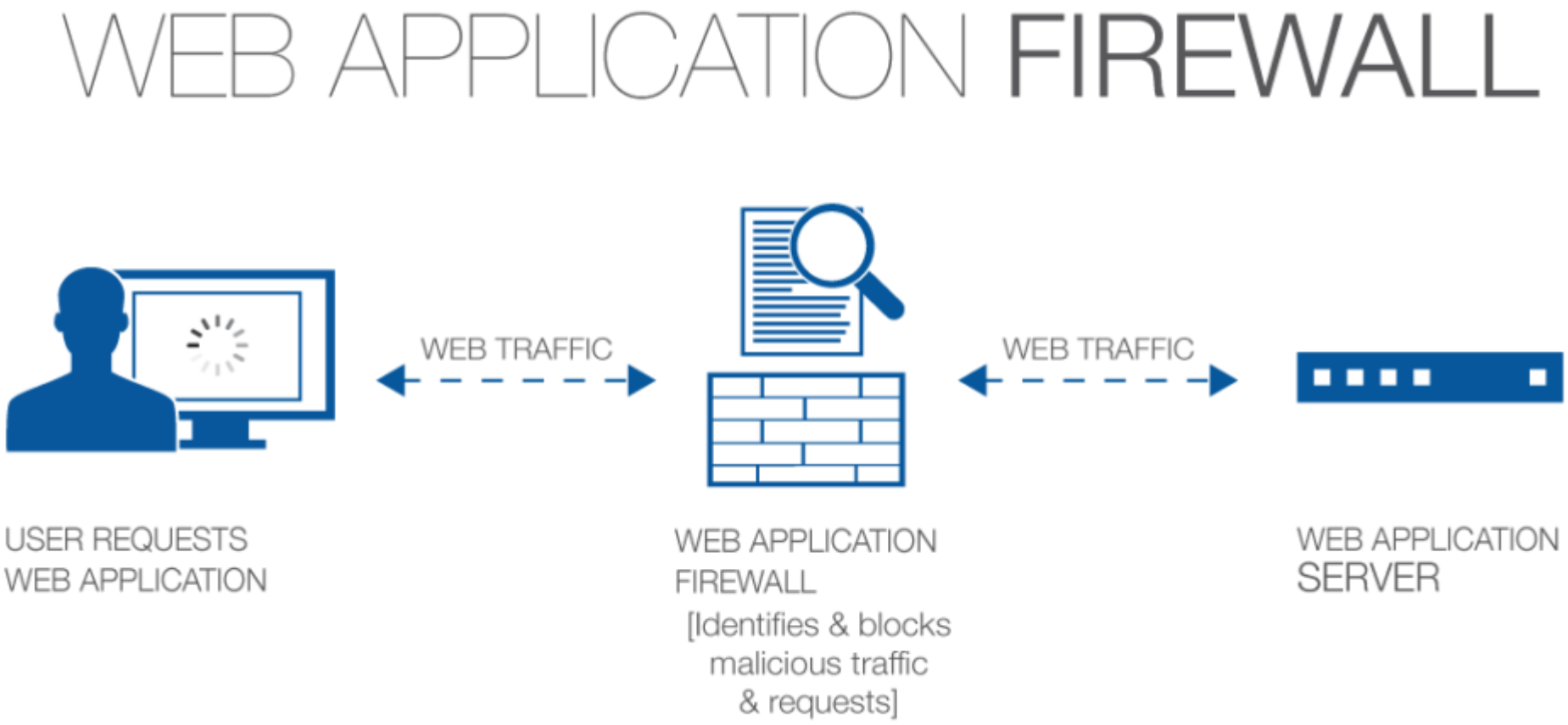
用nginx搭建waf防火墙的实践

- waf应用层攻击对抗详解
- 恶意http请求的甄别
- 恶意客户端的防御控制
- 降低waf防火墙的性能损耗

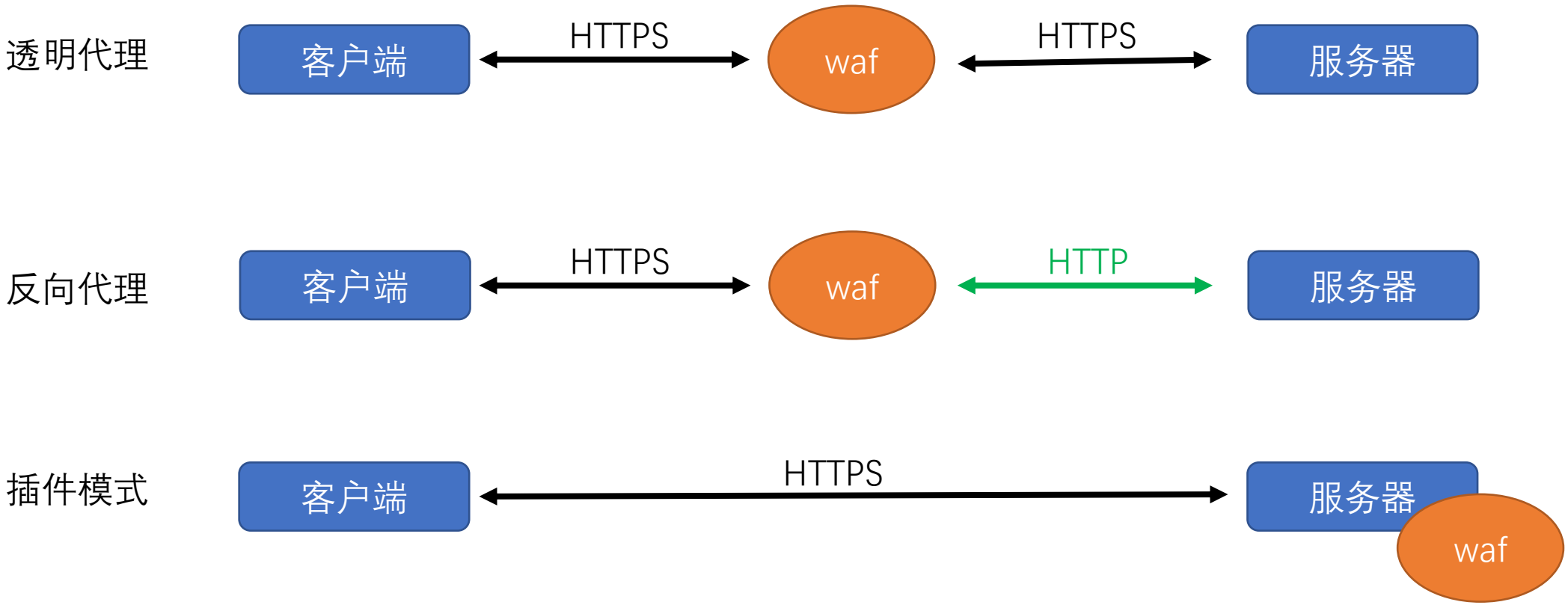
问题

- **什么是Waf防火墙？它应当具备什么能力？**
- **应用层的Web攻击有哪些？**
  - XSS、CSRF、点击劫持、SQL注入、代码注入、应用层DDos、正则ReDos、文件上传漏洞、CC攻击、XXE攻击等是如何进行的？
  - 哪些Web攻击适合用Waf防火墙防御？
- **为什么常用Nginx实现Waf防火墙？**
- **Nginx Waf防火墙有何优缺点？**

WAF：Web Application Firewall



waf的3种工作模式

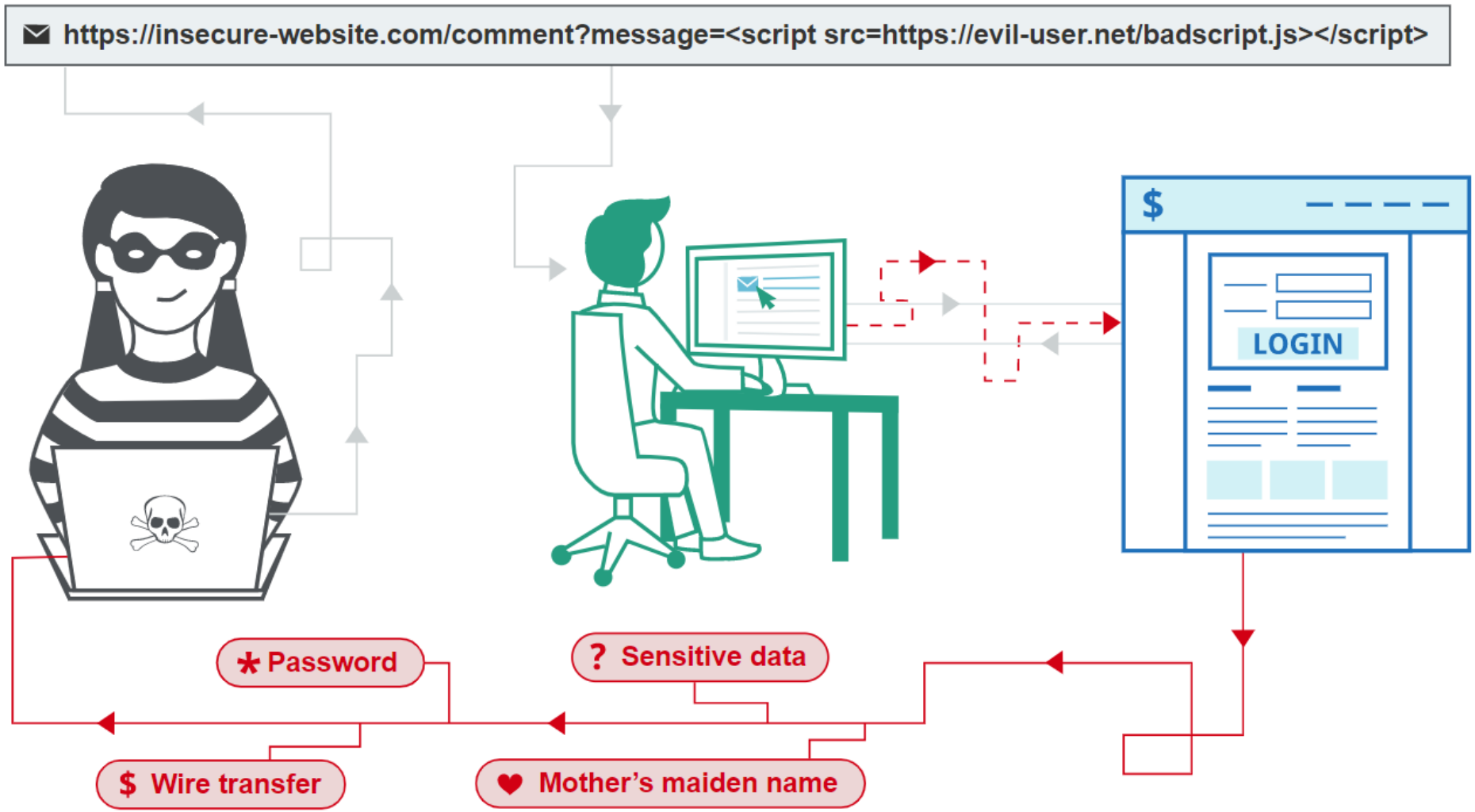

透明代理
客户端
HTTPS
waf
HTTPS
服务器
反向代理
客户端
HTTPS
waf
HTTP
服务器
插件模式
客户端
HTTPS
服务器
waf

反向代理模式waf防火墙的必备能力

- 解析HTTPS协议的能力
- 解析URL参数
- 解析HEADER头部
- 解析BODY包体，及其承载的JSON、XML等格式消息

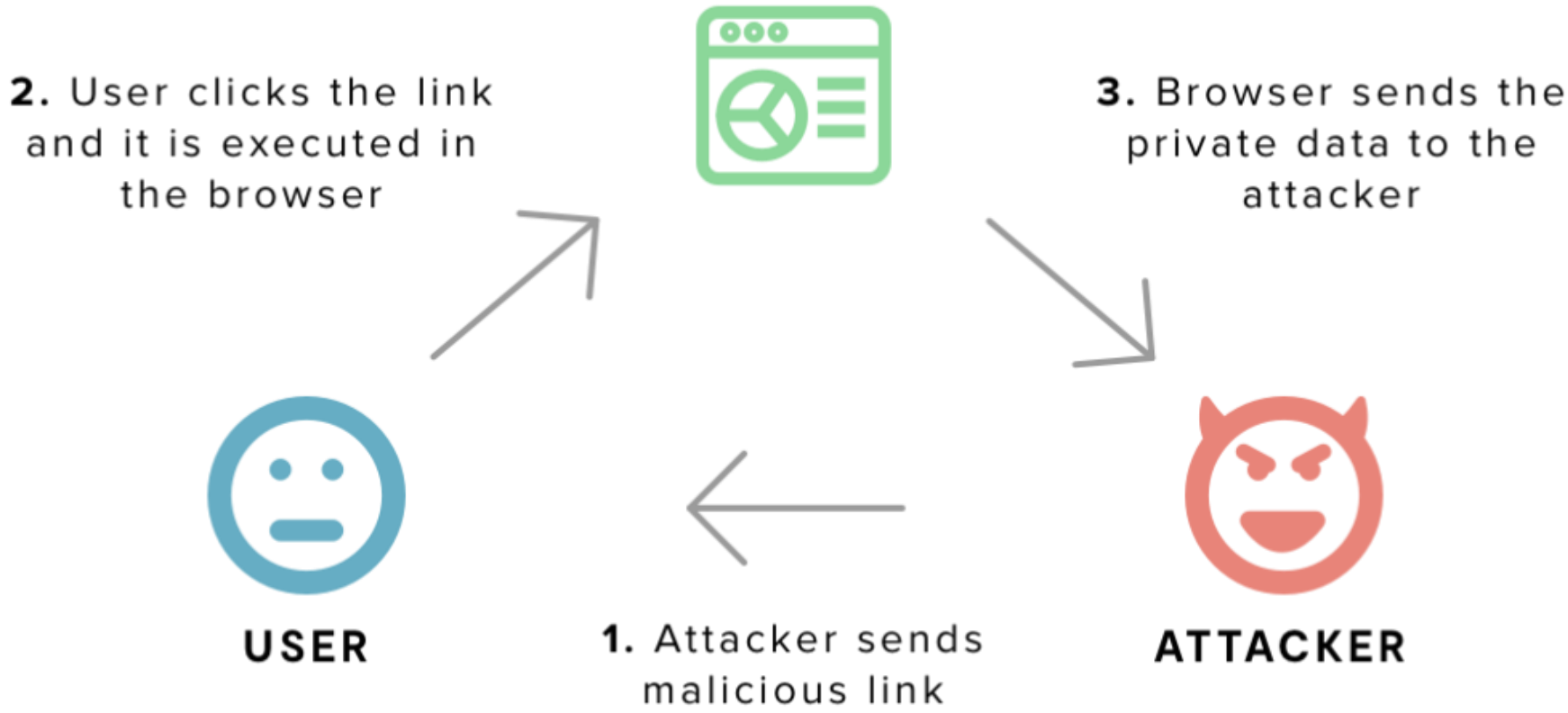
XSS攻击

- Cross Site Script
  - 避免与CSS重名
  - 跨站脚本攻击

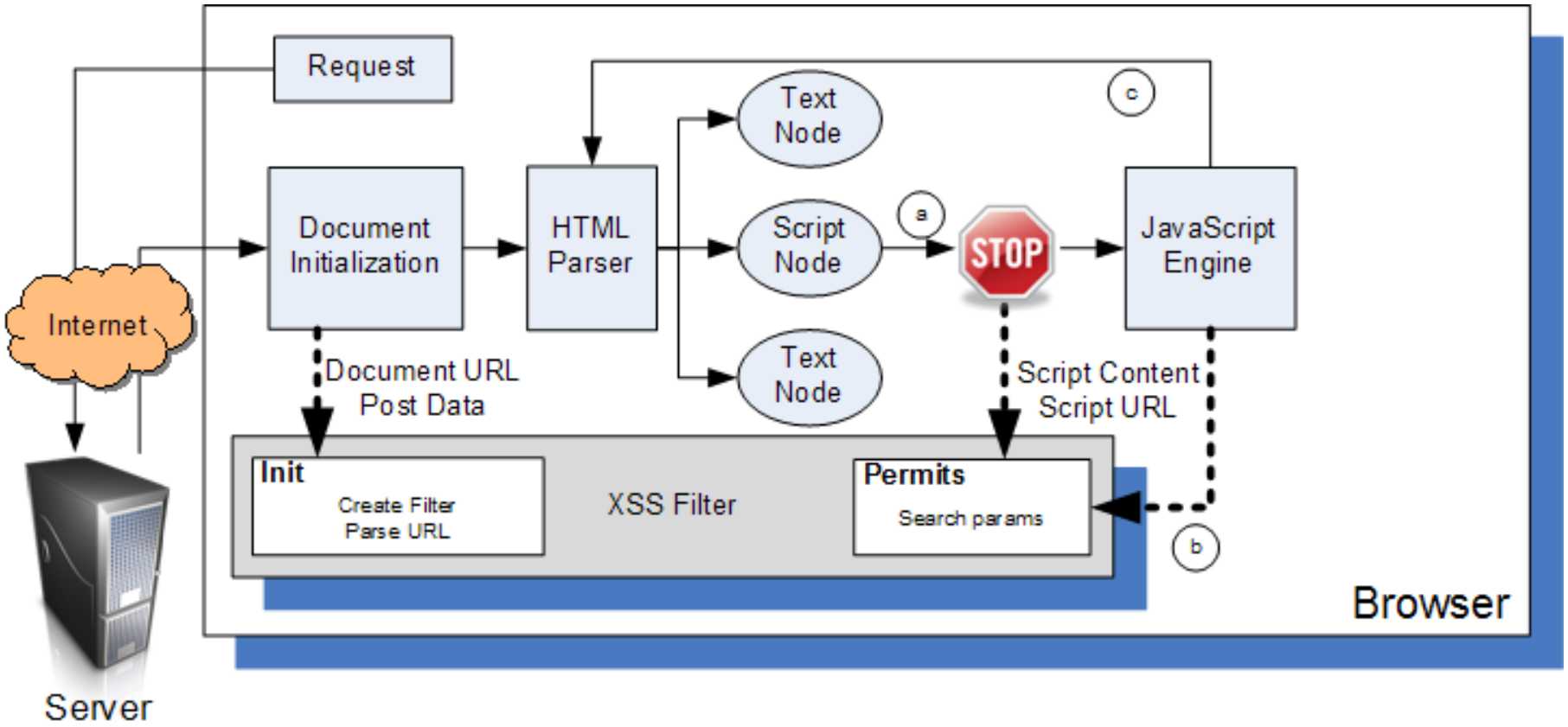


XSS类型

- 反射型
- 存储型
- DOM型

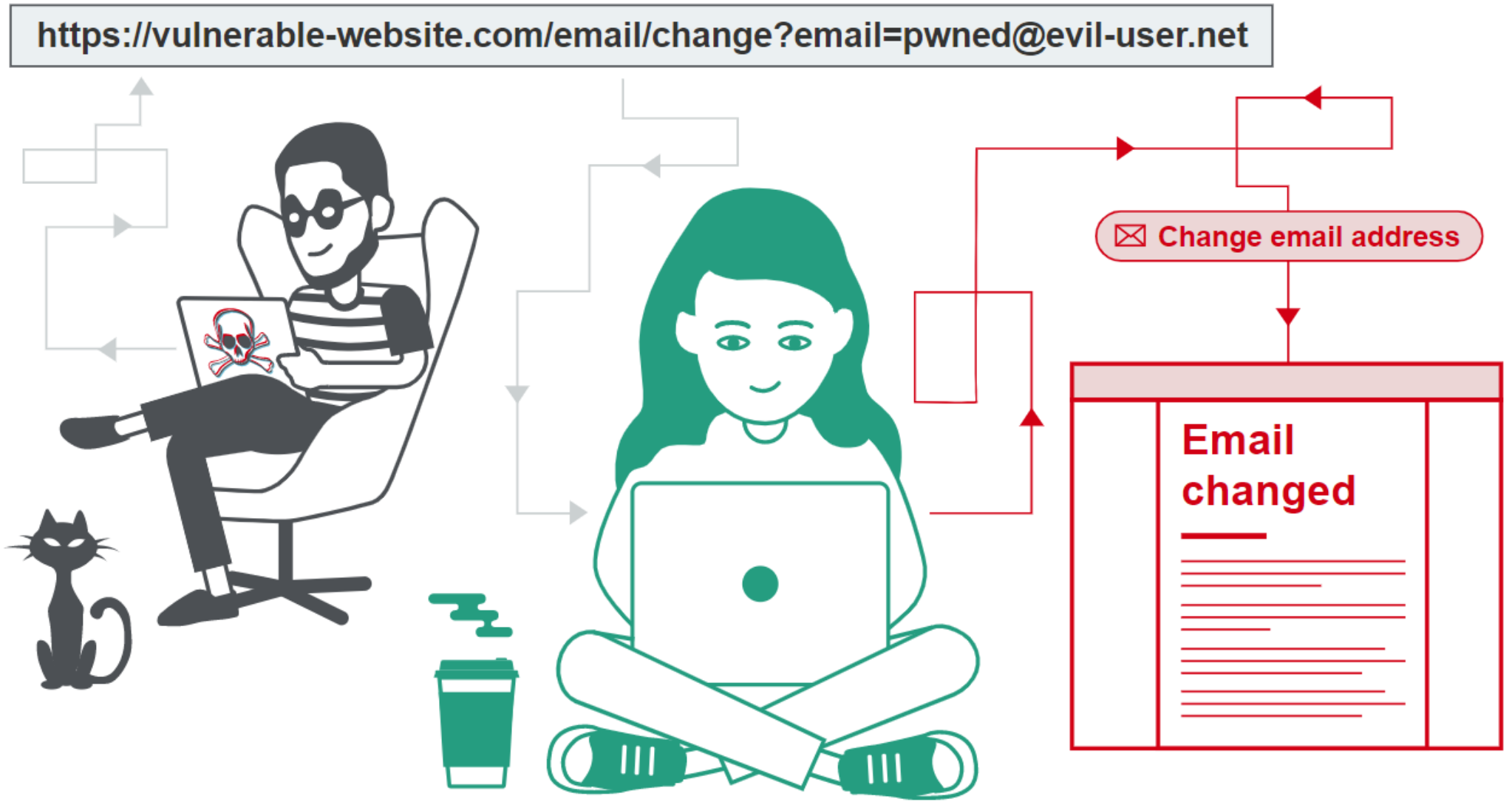


XSS filter过滤



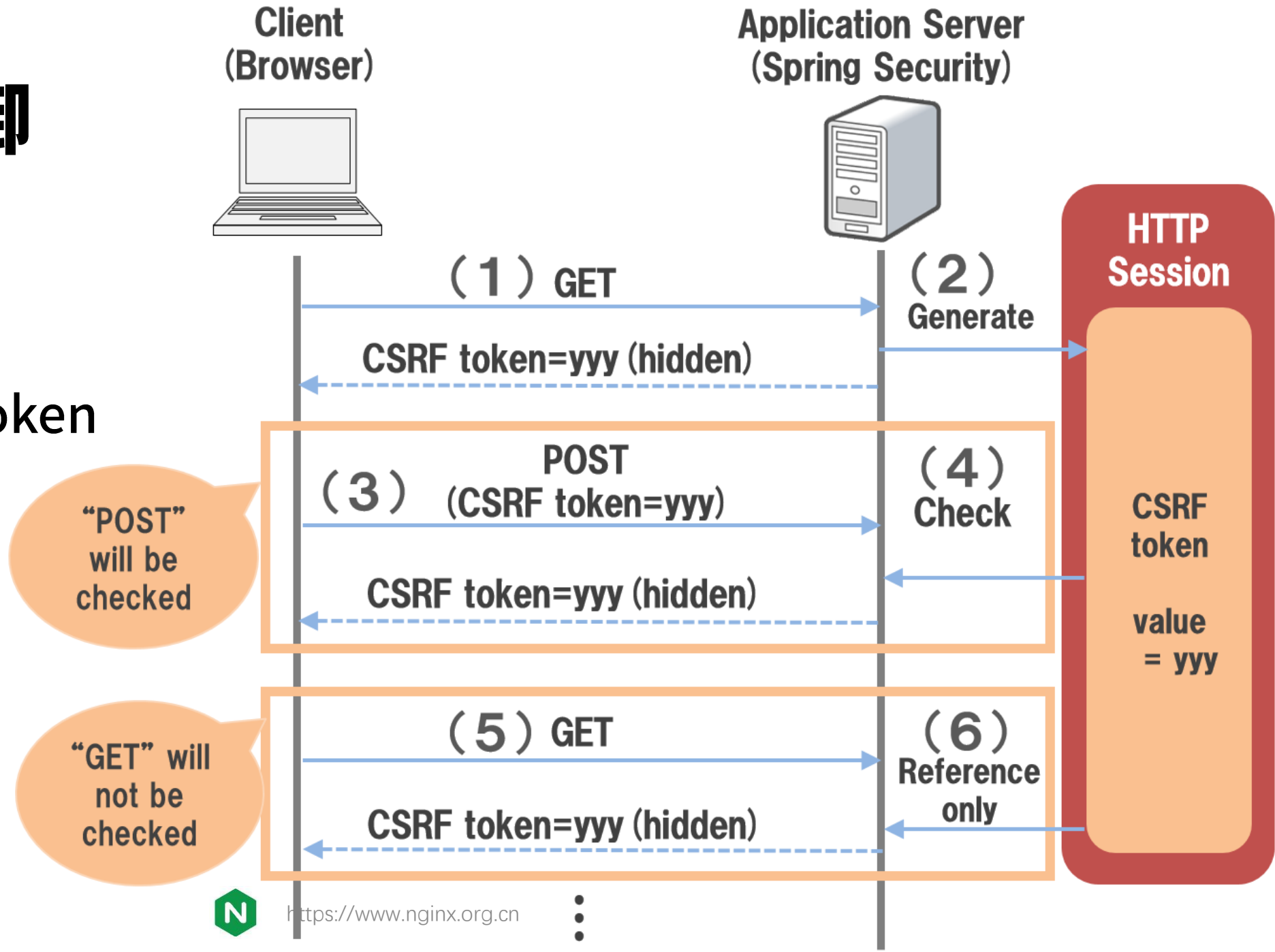
CSRF攻击

- Cross Site Request Forgery跨站点请求伪造
  - 根因：浏览器支持跨站点发送Cookie

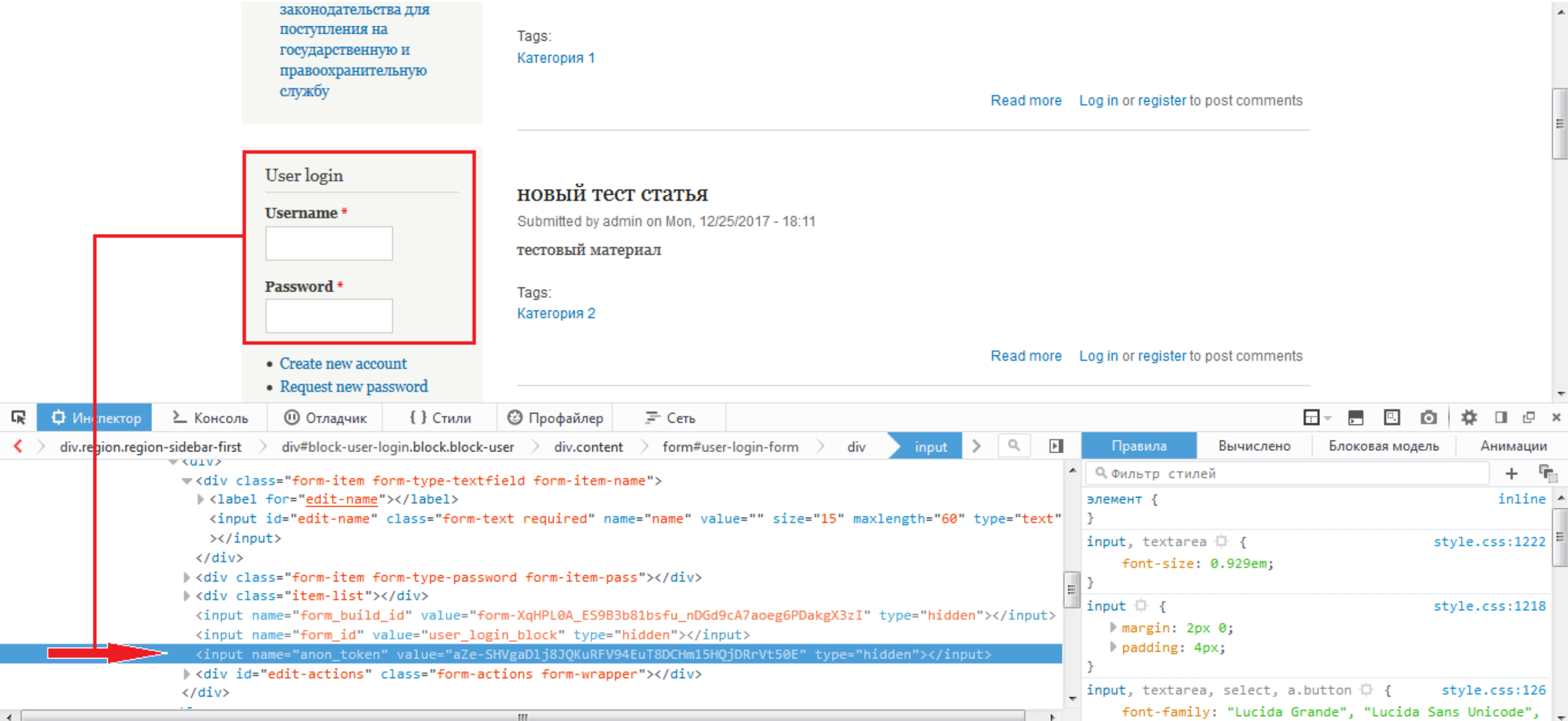


CSRF防御

- 验证码
- Referer
- 无法预测的Token



CSRF token


законодательства для поступления на государственную и правоохранительную службу
Tags:
Категория 1
Read more
Log in or register to post comments
User login
Username *
Password *
Create new account
Request new password
новый тест статья
Submitted by admin on Mon, 12/25/2017 - 18:11
тестовый материал
Tags:
Категория 2
Read more
Log in or register to post comments
Инспектор
Консоль
Отладчик
Стили
Профайлер
Сеть
div.region.region-sidebar-first
div#block-user-login.block.block-user
div.content
form#user-login-form
div
input
Правила
Вычислено
Блоковая модель
Анимации
Фильтр стилей

<label for="edit-name"></label>
<input id="edit-name" class="form-text required" name="name" value="" size="15" maxlength="60" type="text"></input>

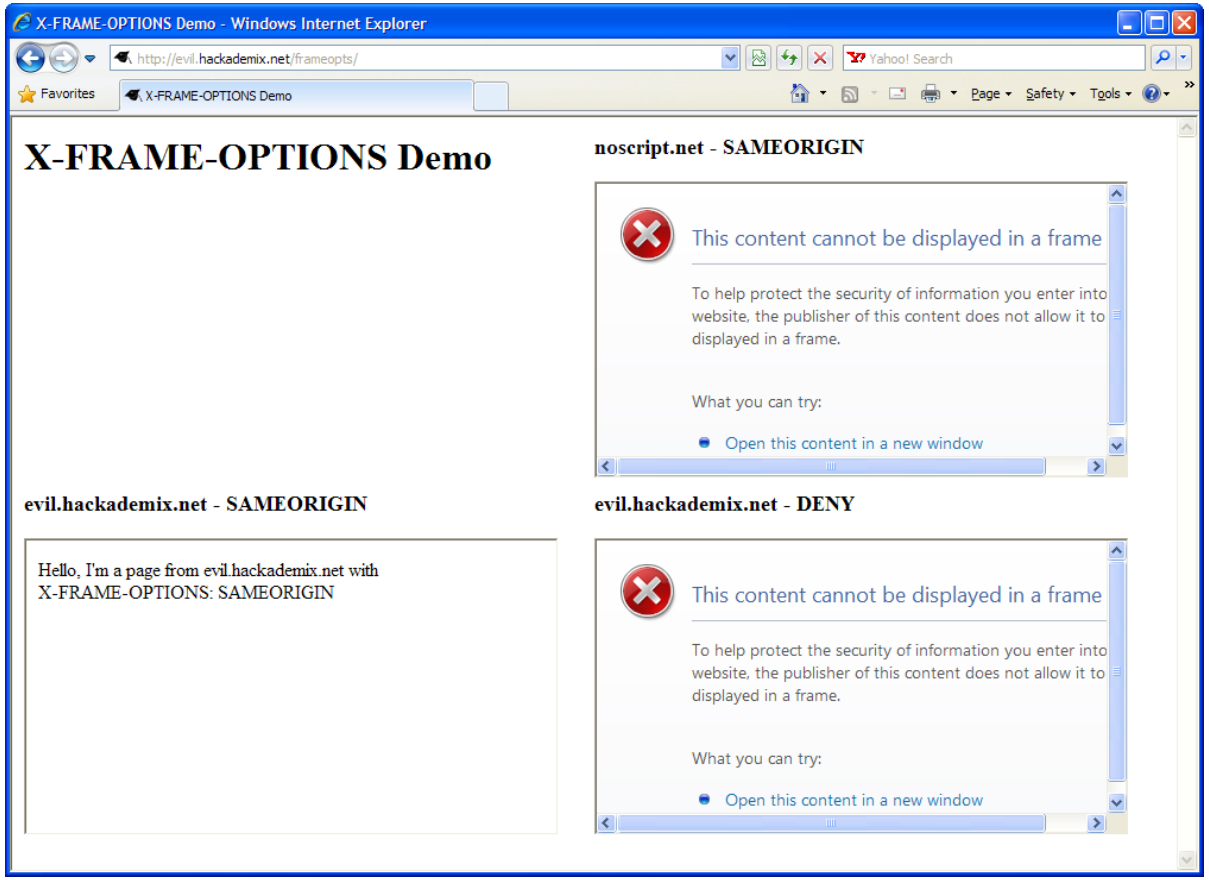
<input name="form_build_id" value="form-XqHPL0A_ES9B3b81bsfu_nDGd9cA7aoeg6PDakgX3zI" type="hidden"></input>
<input name="form_id" value="user_login_block" type="hidden"></input>
<input name="anon_token" value="aZe-SHVgaDlj8JQKuRFV94EuT8DCHm15HQjDRrVt50E" type="hidden"></input>

элемент { inline
}
input, textarea { style.css:1222
font-size: 0.929em;
}
input { style.css:1218
margin: 2px 0;
padding: 4px;
}
input, textarea, select, a.button { style.css:126
font-family: "Lucida Grande", "Lucida Sans Unicode",

click jacking点击劫持

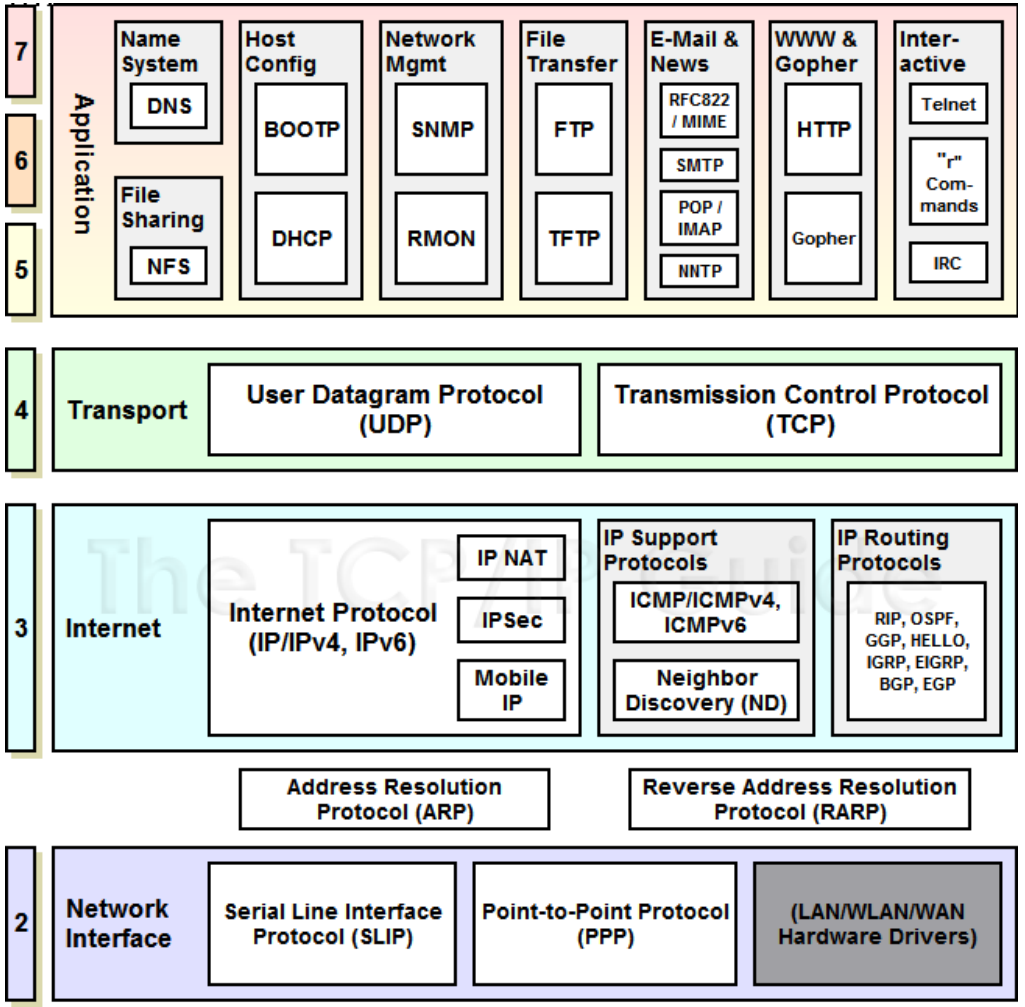
防御点击劫持

- 通过JS禁止跨域iFrame
- 增加X-Frame-Options头部
  - deny
  - sameorigin
  - allow-from

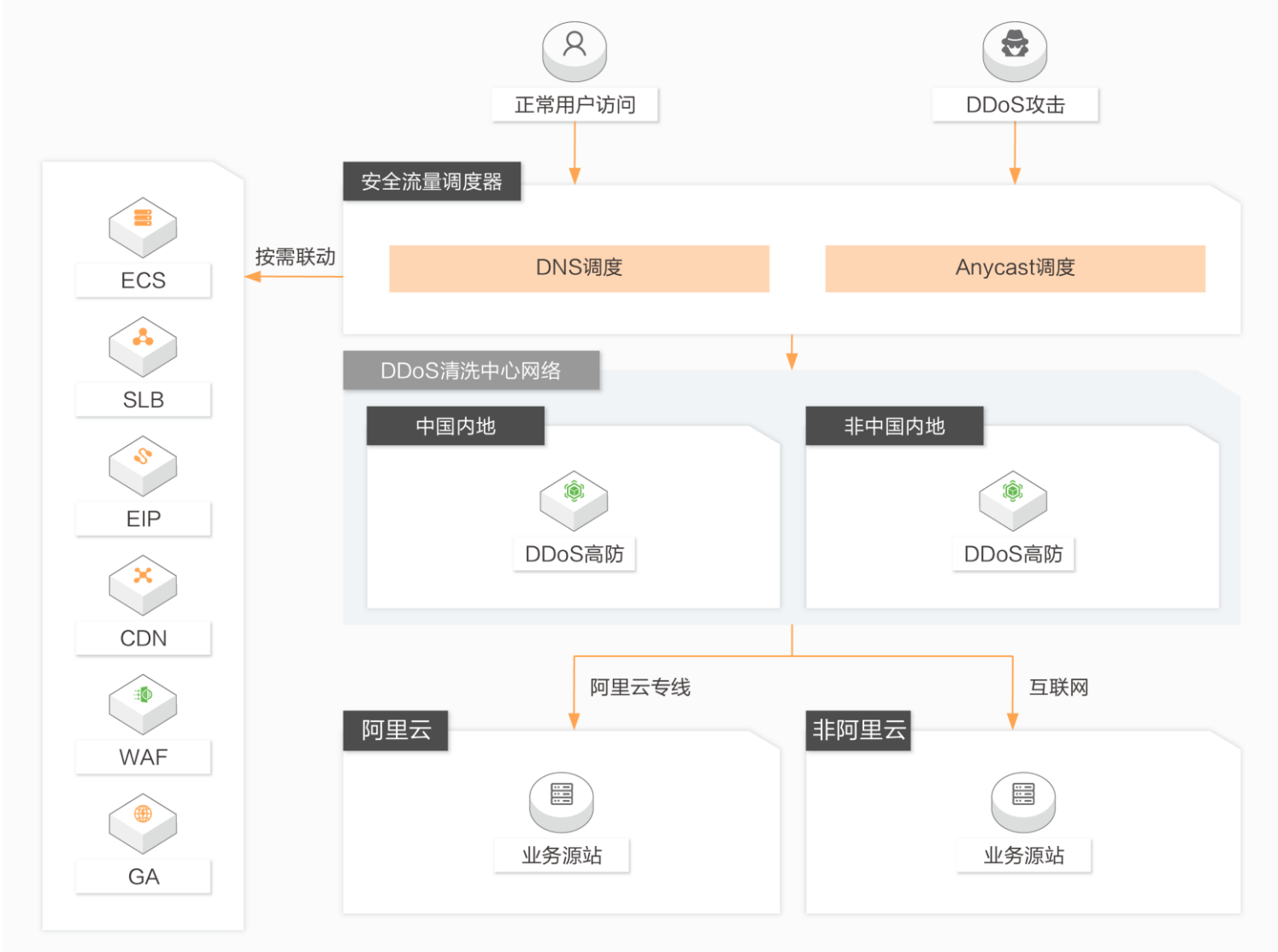


DDos拒绝服务攻击

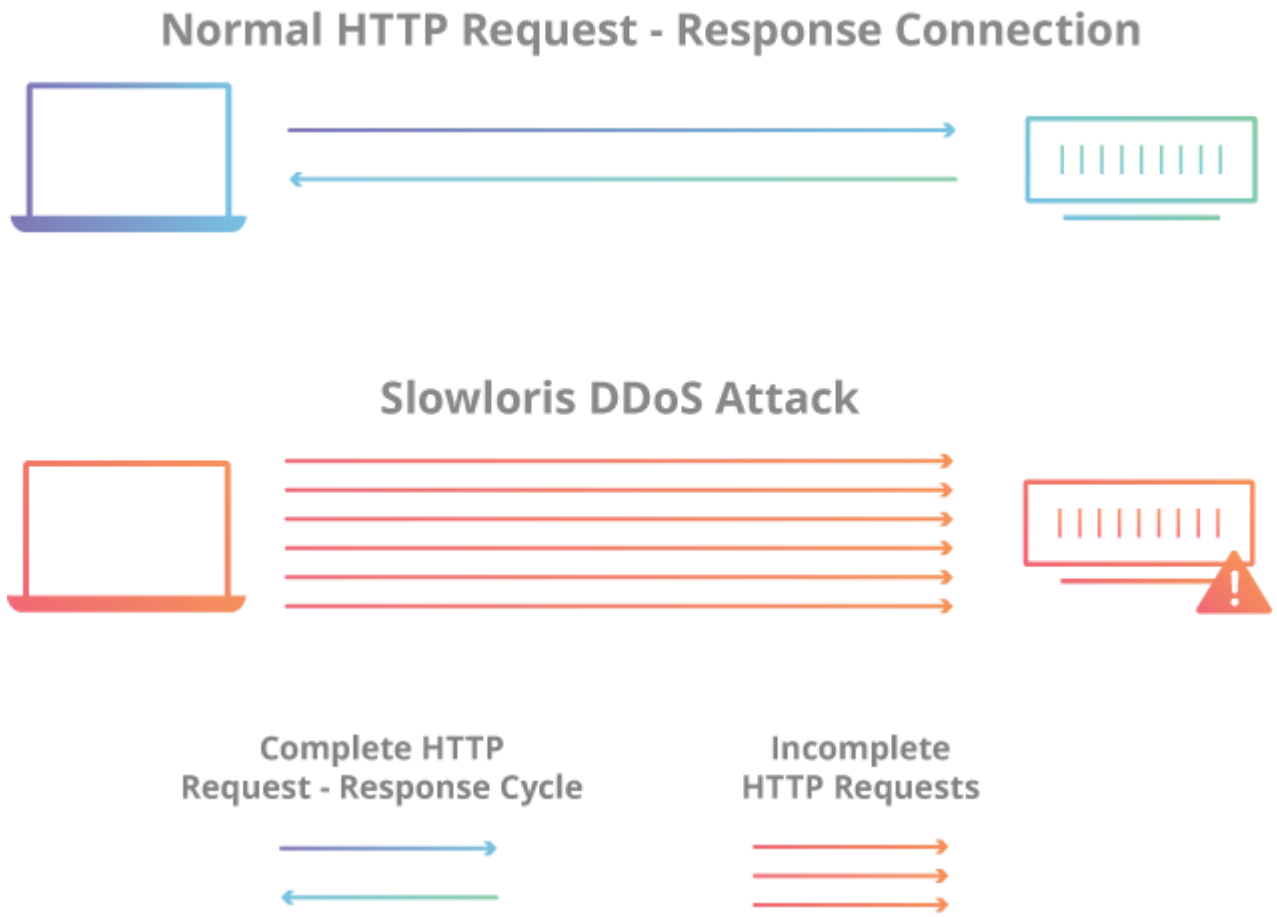
- distributed denial-of-service attack
  - 带宽消耗型
  - 资源消耗型
    - 传输层
      - SYN Flood
    - 应用层
      - POST超大文件
      - Distributed HTTP flood
        - 控制代理
        - 对高流量站点XSS攻击成功
      - 正则ReDOS



网络、传输层DDoS攻击的流量清洗

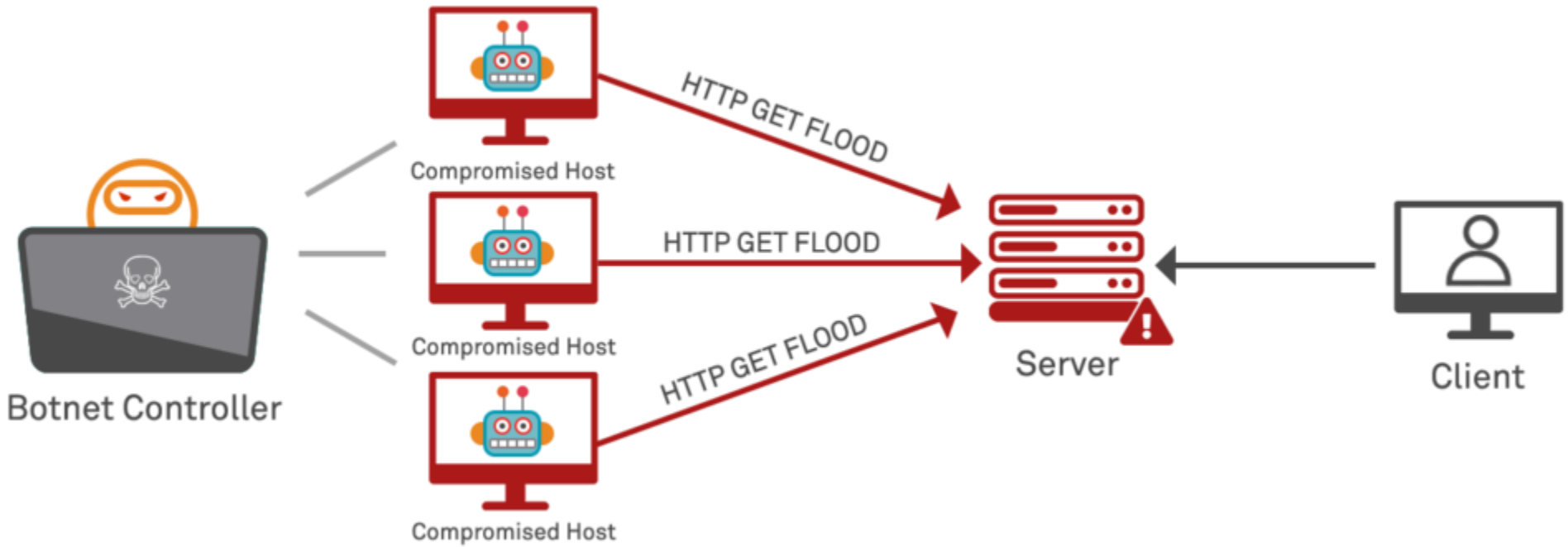


slowloris慢连接攻击



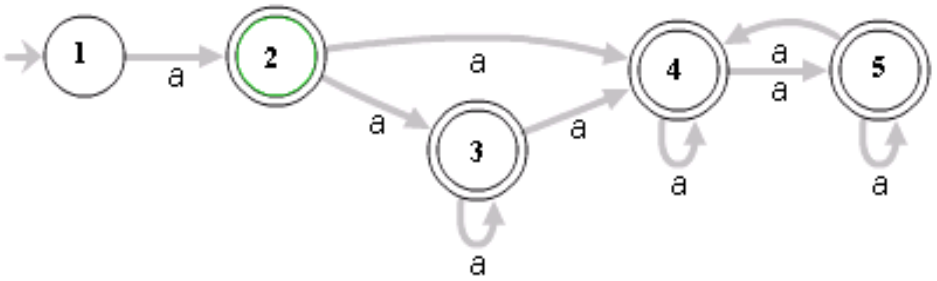
CC攻击

- Challenge Collapsar



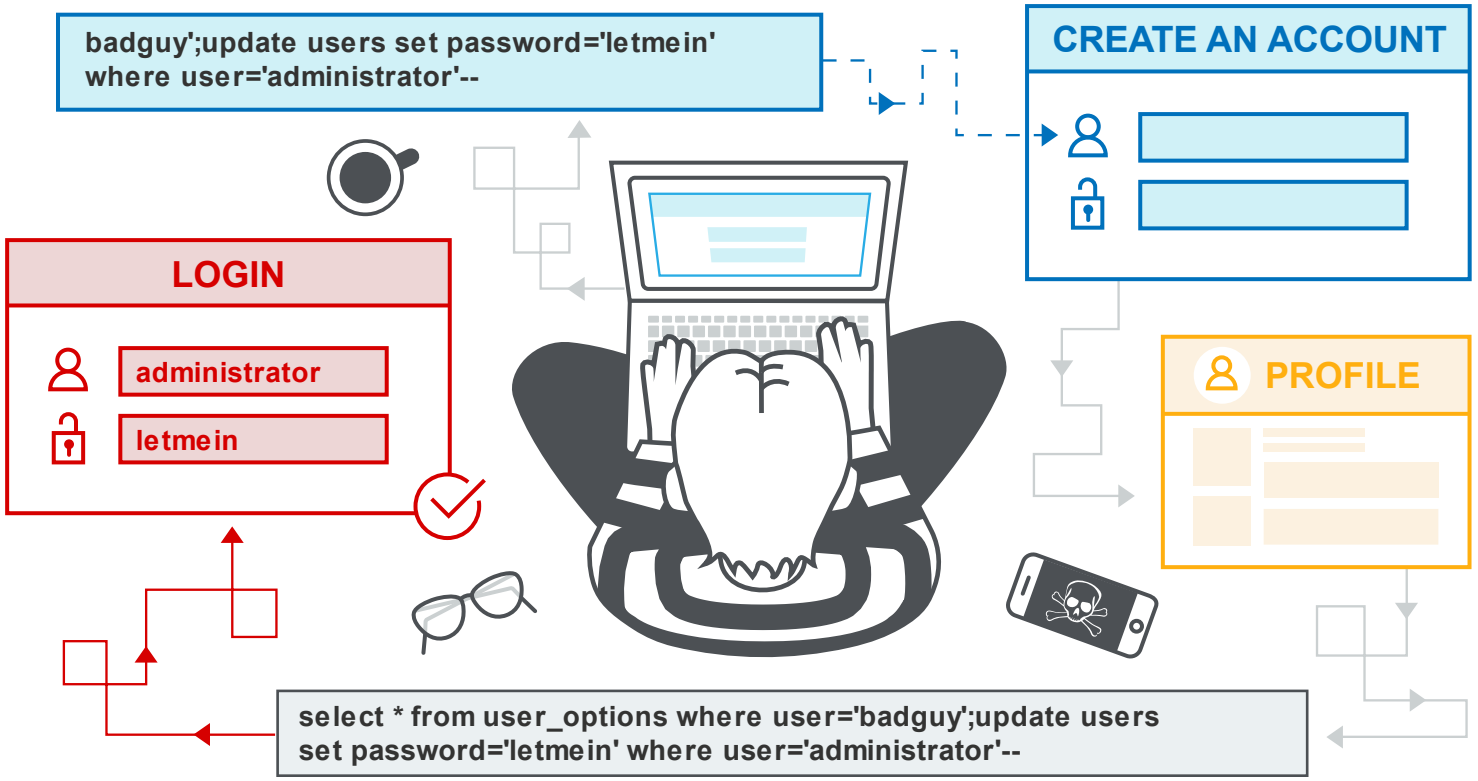
正则 ReDOS

- **Regular expression Denial of Service**
  - DFA(Deterministic Finite Automaton)确定性有限状态自动机
  - NFA(Nondeterministic Finite Automaton)非确定性有限状态自动机
    - 例：^(a+)+$匹配字符串aaaab需要进行16次匹配
    - 有缺陷的正则
      - (a+)+
      - ([a-zA-Z]+)*
      - (a|aa)+
      - (a|a?)+
      - (.*a){x} | for x > 10

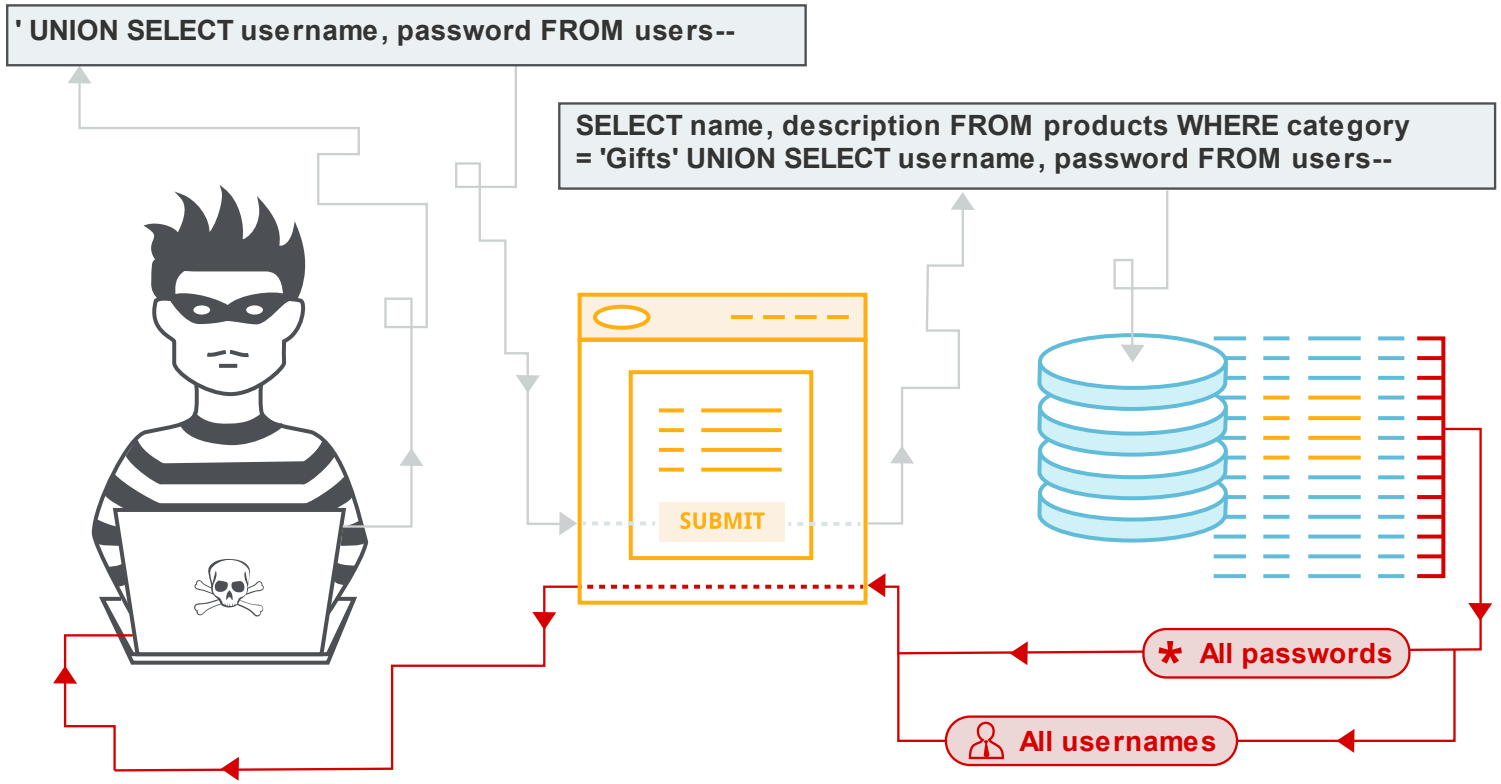


SQL拼接用户输入数据

select * from user_options where user=@用户输入名

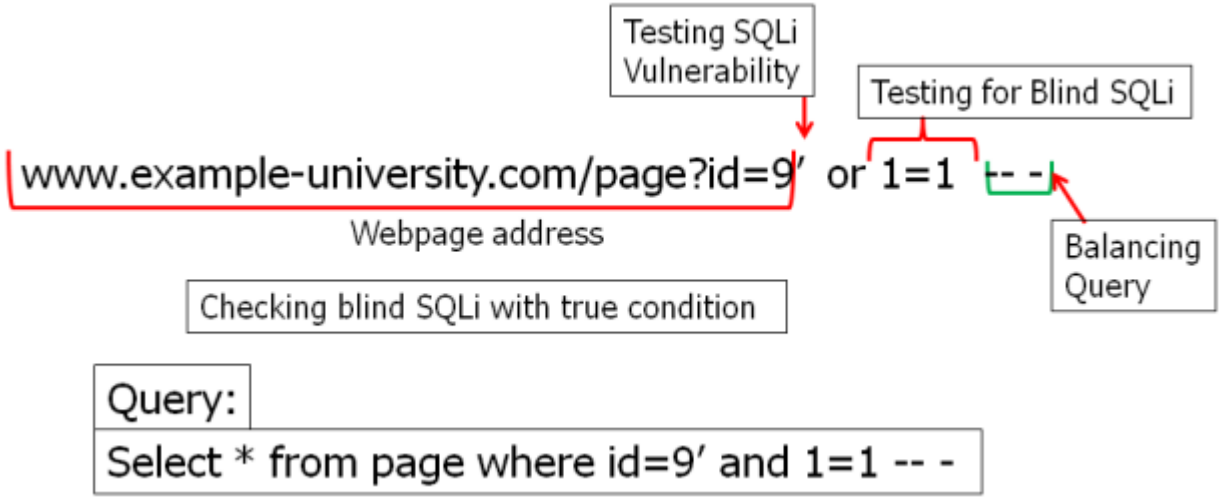


SQL注入攻击


' UNION SELECT username, password FROM users--
SELECT name, description FROM products WHERE category = 'Gifts' UNION SELECT username, password FROM users--
SUBMIT
All passwords
All usernames

SQL盲注

- 无法获得错误回显



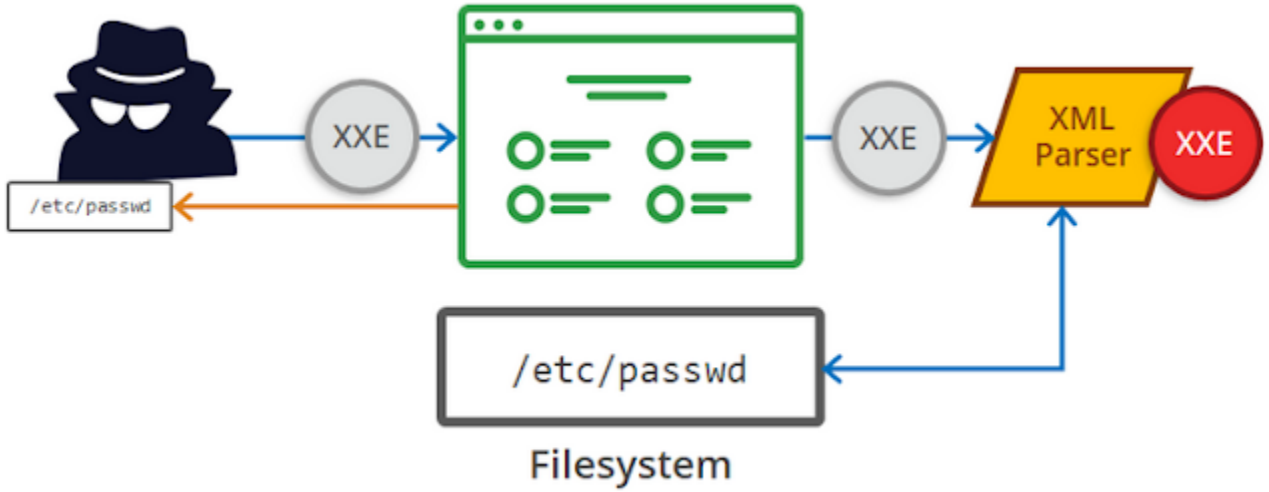
代码注入

- asp
  - execute
- php
  - eval
- aspx
  - eval
- Java
  - ScriptEngine.eval
- C/C++
  - system

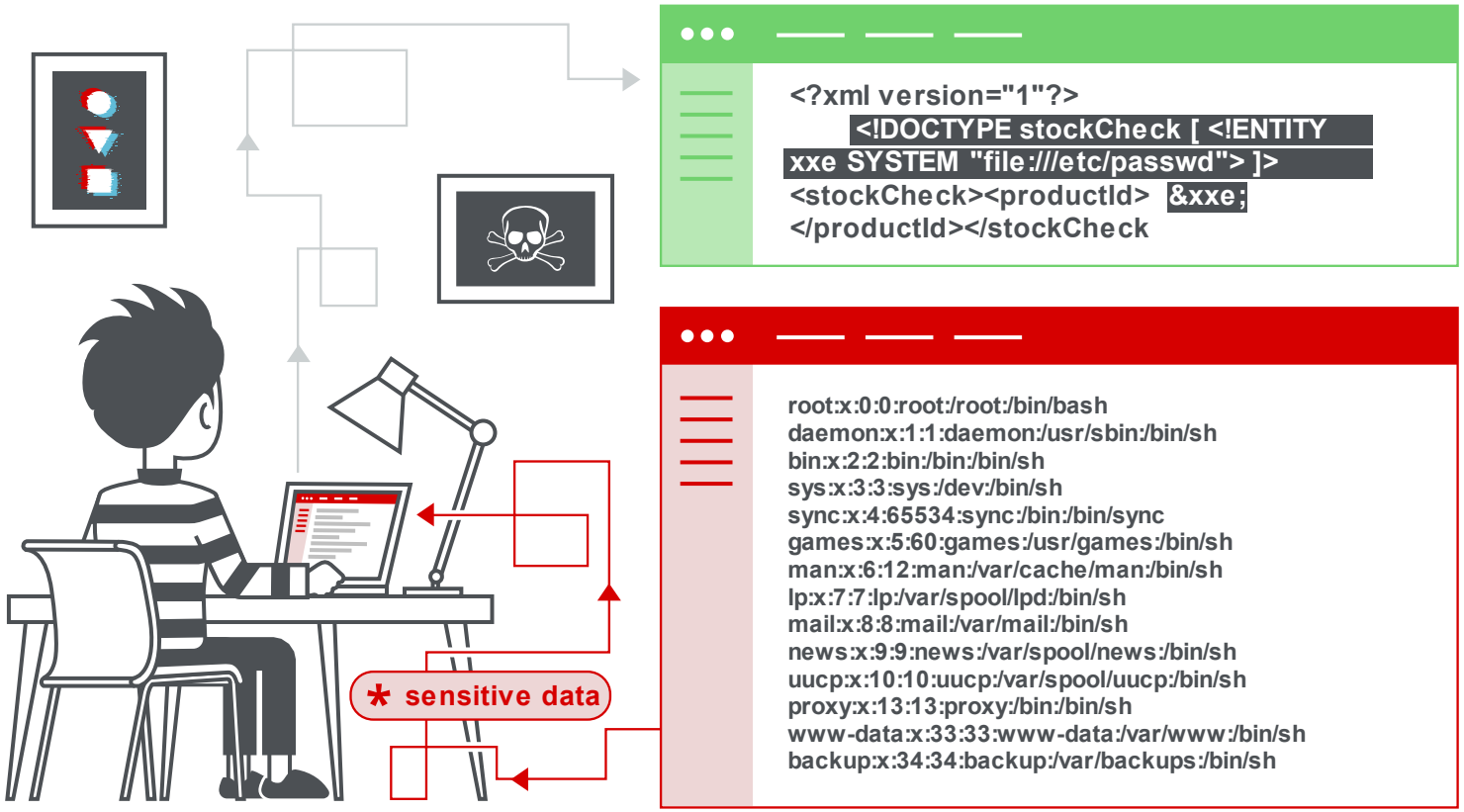
CODE
INJECTION

XML注入

- XXE：XML External Entiy



XXE攻击(DTD部分引用外部资源)



CRLF注入

- \r\n的分隔符效果

```
HTTP/1.1 302 Found
Content-Type: text/plain
Location: /username=\r\n
Content-Type: text/html \r\n\r\n  <-- CRLF two times will separate headers and HTML

<html><h1>hacked!</h1></html>
Content-Type: text/plain
Date: Thu, 13 Jun 2019 16:12:20 GMT
```

CRLF注入Cookie

```
https://example.com/%0d%0aSet-Cookie%3A%20username%3D%3Cscript%3Ealert
(%27hacked%27)%3C%2Fscript%3E
```

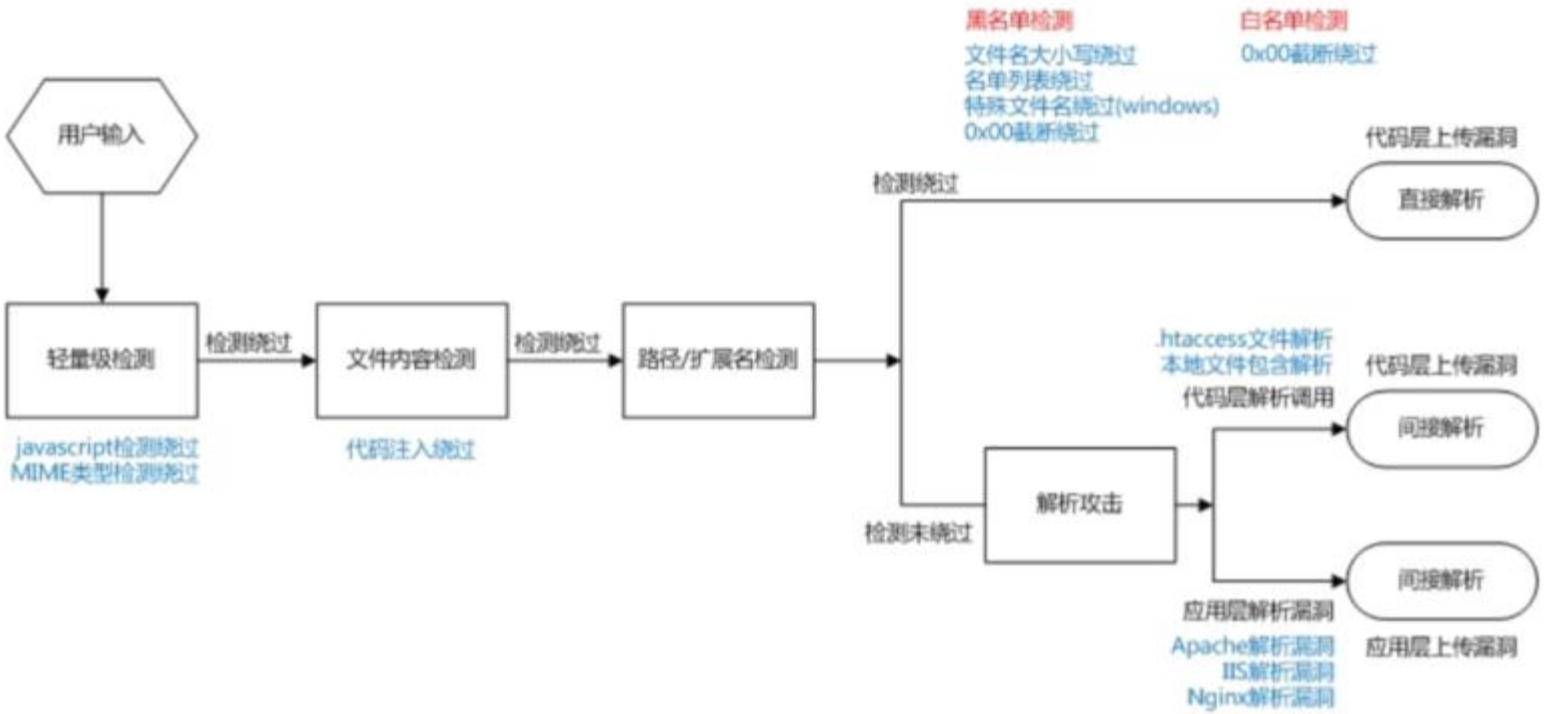
```
HTTP/1.1 200 OK
Location: profile \r\n
Set-Cookie: username=<script>alert('hacked')</script>
```

HTTP响应拆分


Request
Raw
Params
Headers
Hex
1 GET /%%0D%%0ASet-Cookie:crlfinjection=POCbyPreritPathak HTTP/1.1
2 Host: [redacted]
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125
Safari/537.36
5 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/a
png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
6 Accept-Encoding: gzip, deflate
Response
Raw
Headers
Hex
1 HTTP/1.0 307 Temporary Redirect
2 Server: glass/1.0 Python/2.6.4
3 Date: Thu, 20 Aug 2020 11:30:47 GMT
4 Location: https://[redacted]
5 Set-Cookie:crlfinjection=POCbyPreritPathak
6
7

文件上传漏洞

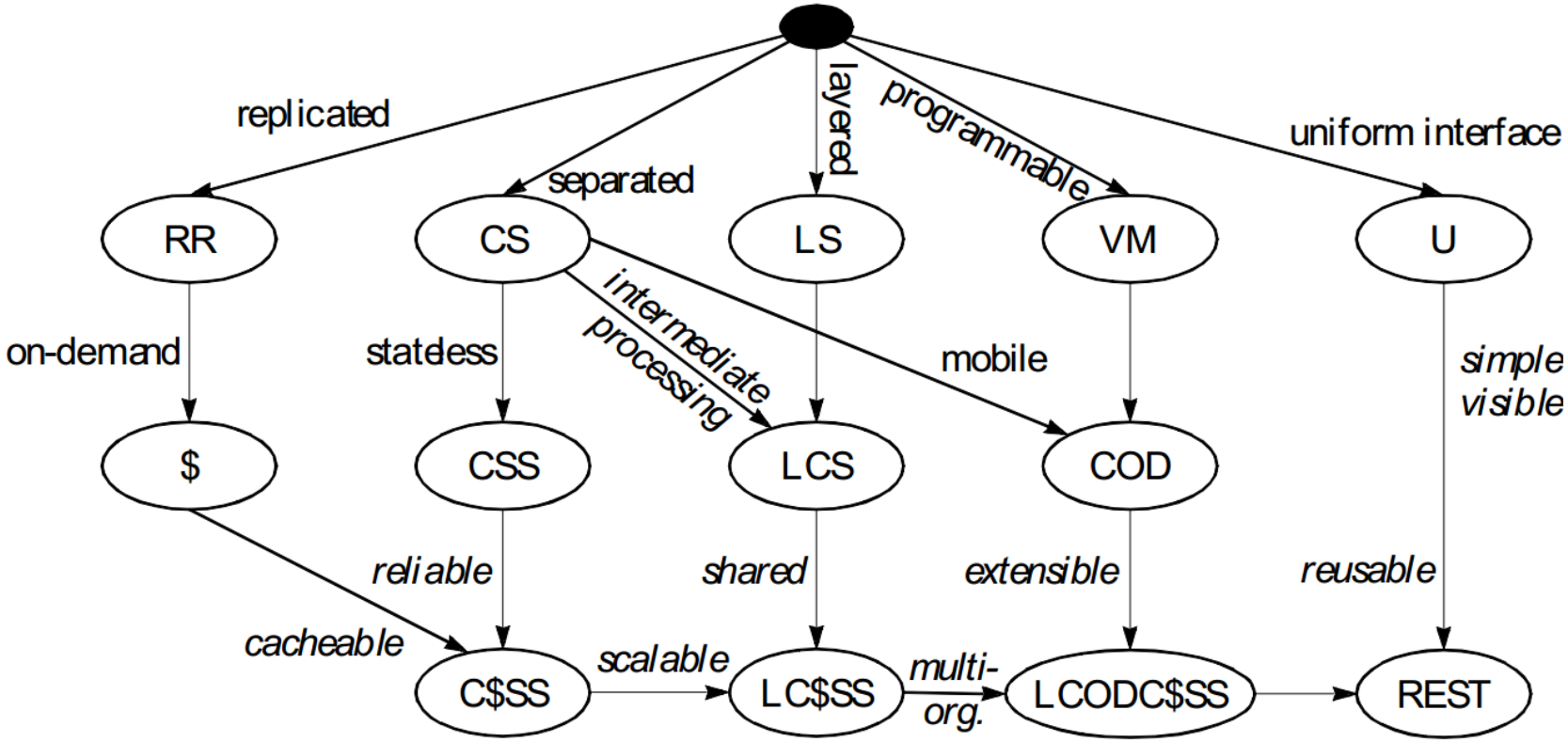
- 上传文件会被Server执行
  - 或者对客户端有害
- 用户可以访问上传文件
- 文件未被转换



用nginx搭建waf防火墙的实践

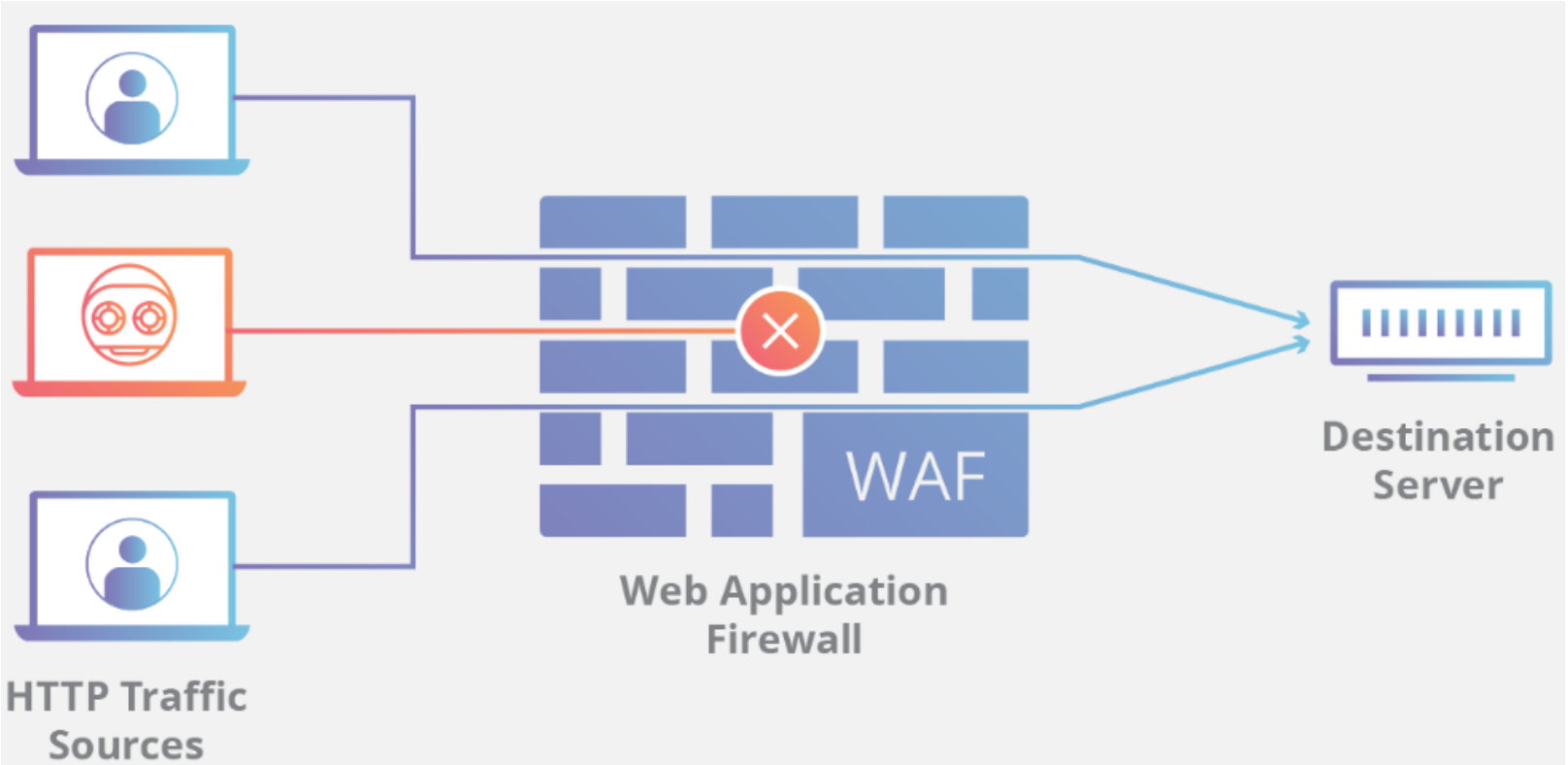
- waf应用层攻击对抗详解
- 恶意http请求的甄别
- 恶意客户端的防御控制
- 降低waf防火墙的性能损耗

REST架构带来的安全问题

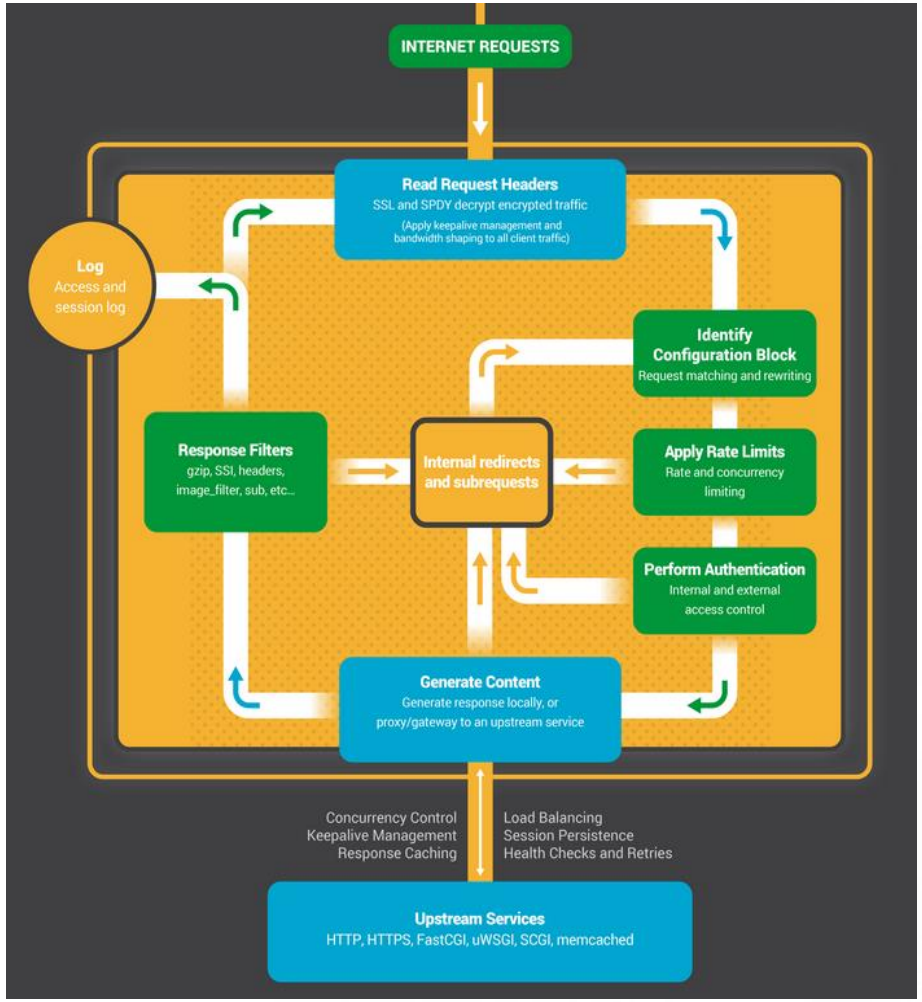


Nginx为什么适合做反向代理型Waf?

- **IP协议处理**
  - IP地址处理
- **TCP/IP协议处理**
  - TCP并发连接管理
  - TCP传输速度管理
- **TLS协议处理**
  - 机密、完整、身份验证
- **HTTP协议解析**
  - URI处理
  - Header处理
  - Body处理
- **第三方HTTP模块(Lua、JS块)**
  - 11个阶段的请求处理模块
  - 响应过滤模块
  - 负载均衡模块



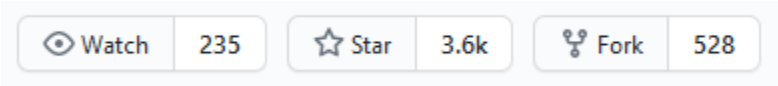
常用于Waf的处理阶段



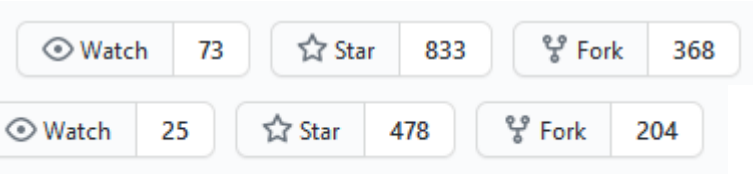
POST_READ	realip
SERVER_REWRITE	rewrite
FIND_CONFIG	
REWRITE	rewrite
POST_REWRITE	
PREACCESS	limt_conn, limit_req
ACCESS	auth_basic, access, auth_request
POST_ACCESS	
PRECONTENT	try_files
CONTENT	index, autoindex, concat
LOG	access_log

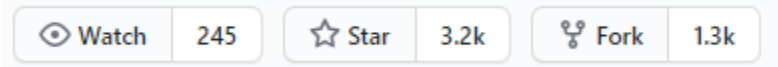
Nginx开源Waf

- Nginx C模块
  - https://github.com/nbs-system/naxsi



- Lua模块
  - https://github.com/alexazhou/VeryNginx/
  - https://github.com/xsec-lab/x-waf
  - https://github.com/unixhot/waf
  - https://github.com/titansec/OpenWAF
  - https://github.com/loveshell/ngx_lua_waf



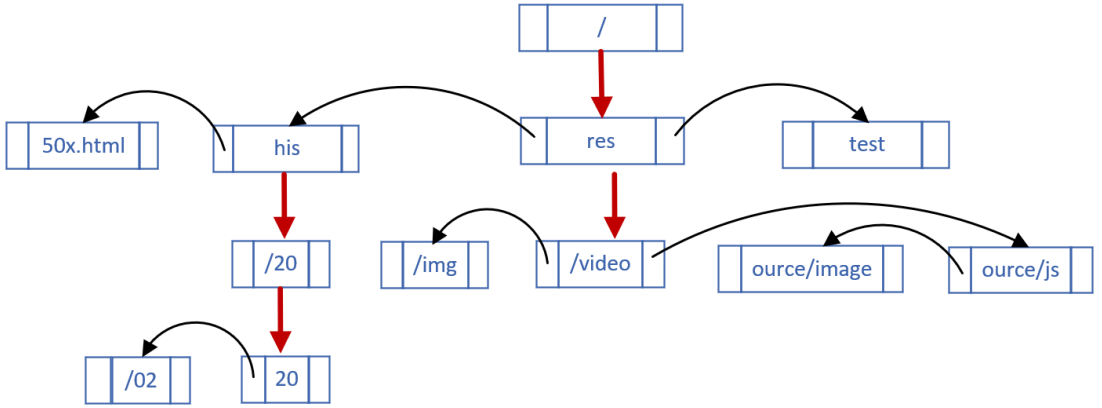


检测恶意请求

- **nginx.conf语法**
  - HTTP语义
    - URL：$uri（location匹配）、$args（$args_）
    - HEADER：$http_（server_name与Host头部匹配）
    - BODY：结构化数据匹配
      - 表单匹配
  - 变量
  - if指令
- **C模块**
- **Lua代码**

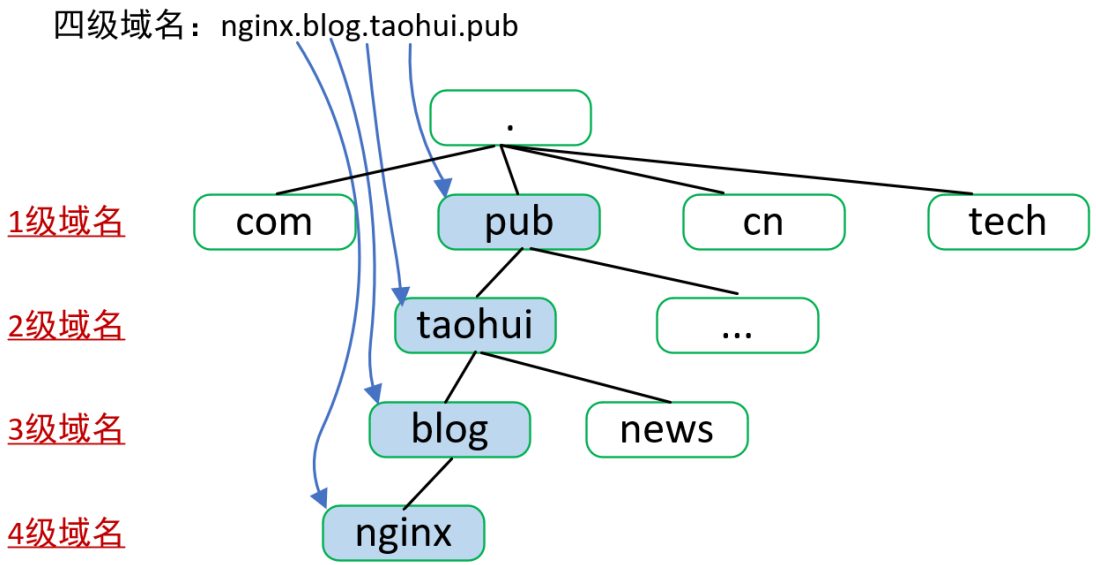
URI与location的匹配

- location的匹配优先级
  - 精确匹配
  - 正则表达式匹配
  - 最长前缀匹配



server与虚拟主机

- 多级域名匹配
- server_name
  - 精确匹配
  - 前缀通配符匹配
  - 后缀通配符匹配
  - 正则表达式匹配



rewrite 模块的 if 指令

Syntax:	**if** (*condition*) { ... }
Default:	—
Context:	server, location

规则

条件 condition 为真，则执行大括号内的指令；遵循值指令的继承规则

if 指令的条件表达式

规则

01 检查变量为空或者值是否为0，直接使用

02 将变量与字符串做匹配，使用=或者 !=

03 将变量与正则表达式做匹配

- 大小写敏感，~或者!~
- 大小写不敏感，~*或者!~*

04 检查文件是否存在，使用-f 或者 !-f

05 检查目录是否存在，使用-d或者!-d

06 检查文件、目录、软链接是否存在，使用-e或者!-e

07 检查是否为可执行文件，使用-x或者!-x

HTTP包体的传输方式

- **chunk包体**
  - chunked_transfer_encoding on;
- **content-length明确的包体**
  - 以 boundary分隔的multipart多消息包体

Nginx转发HTTP包体的方式

- 即时转发
  - proxy_request_buffering on | off;
- 缓存至内存
  - client_body_max_size = client_body_buffer_size
  - client_body_in_single_buffer
    - $request_body
- 缓存至磁盘延后转发

ngx_http_form_input_module

- **通过--without-http_form_input_module禁用模块**
  - 源码及文档：https://github.com/calio/form-input-nginx-module
  - 依赖ndk_http_module模块：ngx_devel_kit
- **rewrite阶段读取请求包体，将POST和PUT请求中的FORM表单内容，解析成nginx变量**
  - client_body_max_size必须与client_body_buffer_size一致
  - 仅接受Content-Type： application/x-www-form-urlencoded
    - curl localhost/form -X POST -d "filename=1.txt&filename=2.jpg&name=john"

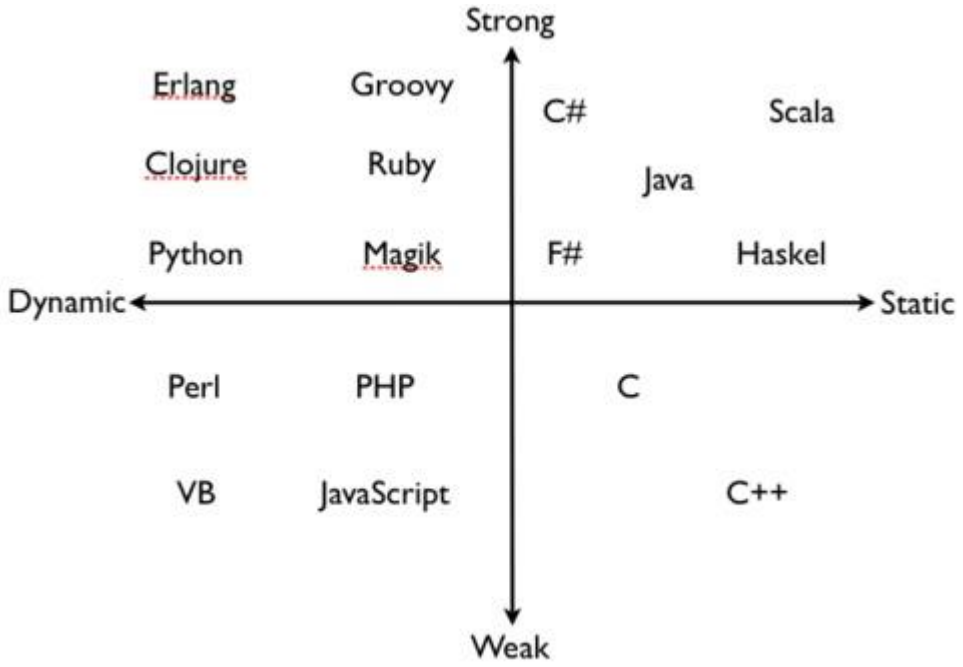
```
..!...!.POST /form HTTP/1.1
User-Agent: curl/7.29.0
Host: localhost
Accept: */*
Content-Length: 39
Content-Type: application/x-www-form-urlencoded

filename=1.txt&filename=2.jpg&name=john
```

C模块检测恶意请求

- 从ngx_http_request_t中获取请求信息
  - URL及HEADER头部
    - 结构化key/value头部
      - headers_in
    - 原始字符流
      - header_in
        - uri_start
        - args_start
        - header_start
  - 包体
    - request_body
      - temp_file
      - bufs
      - buf

Lua

- https://www.lua.org/
- 诞生于1993年，一种动态、弱类型脚本语言
  - 支持垃圾回收
- Lua + Nginx
  - TEngine
  - OpenResty
    - Kong

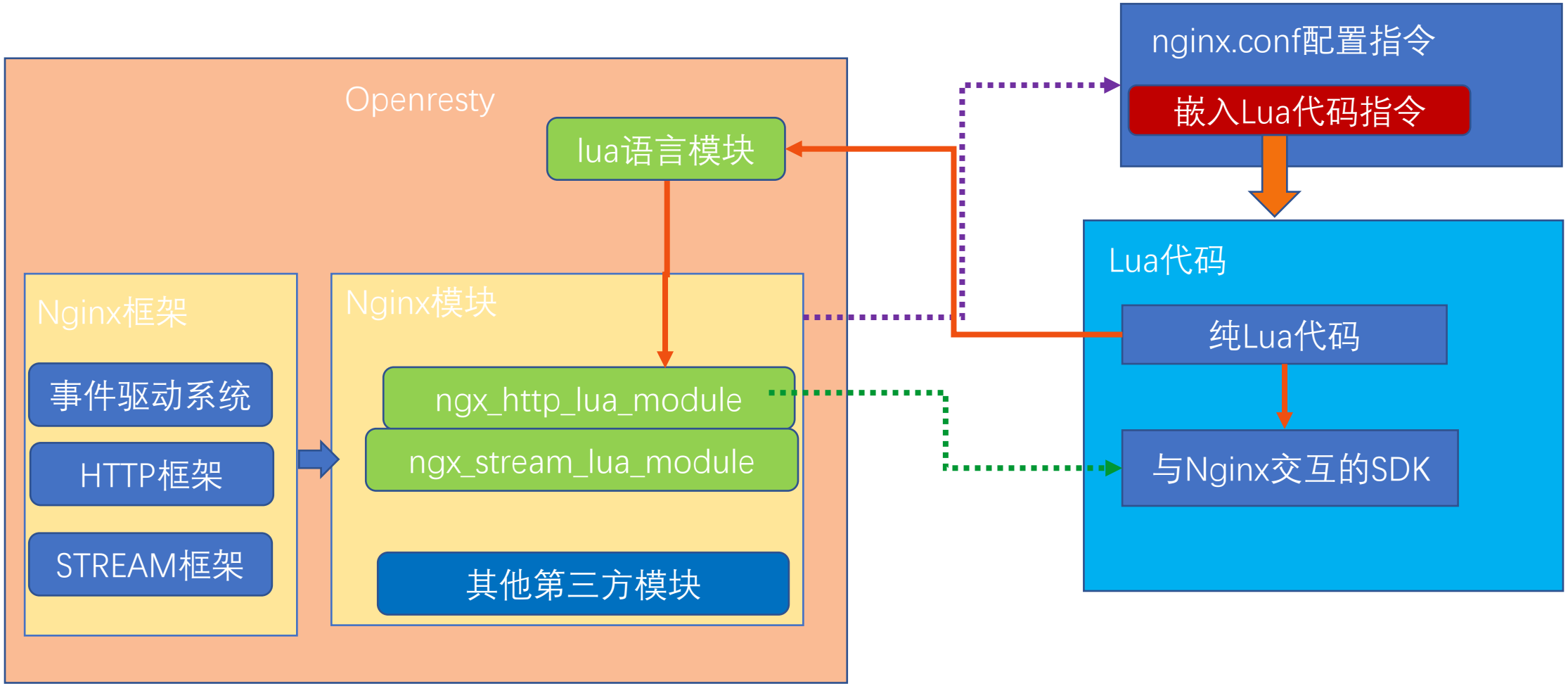


Lua特点

- 跨平台
- 性能高
- 嵌入式
- 体积小

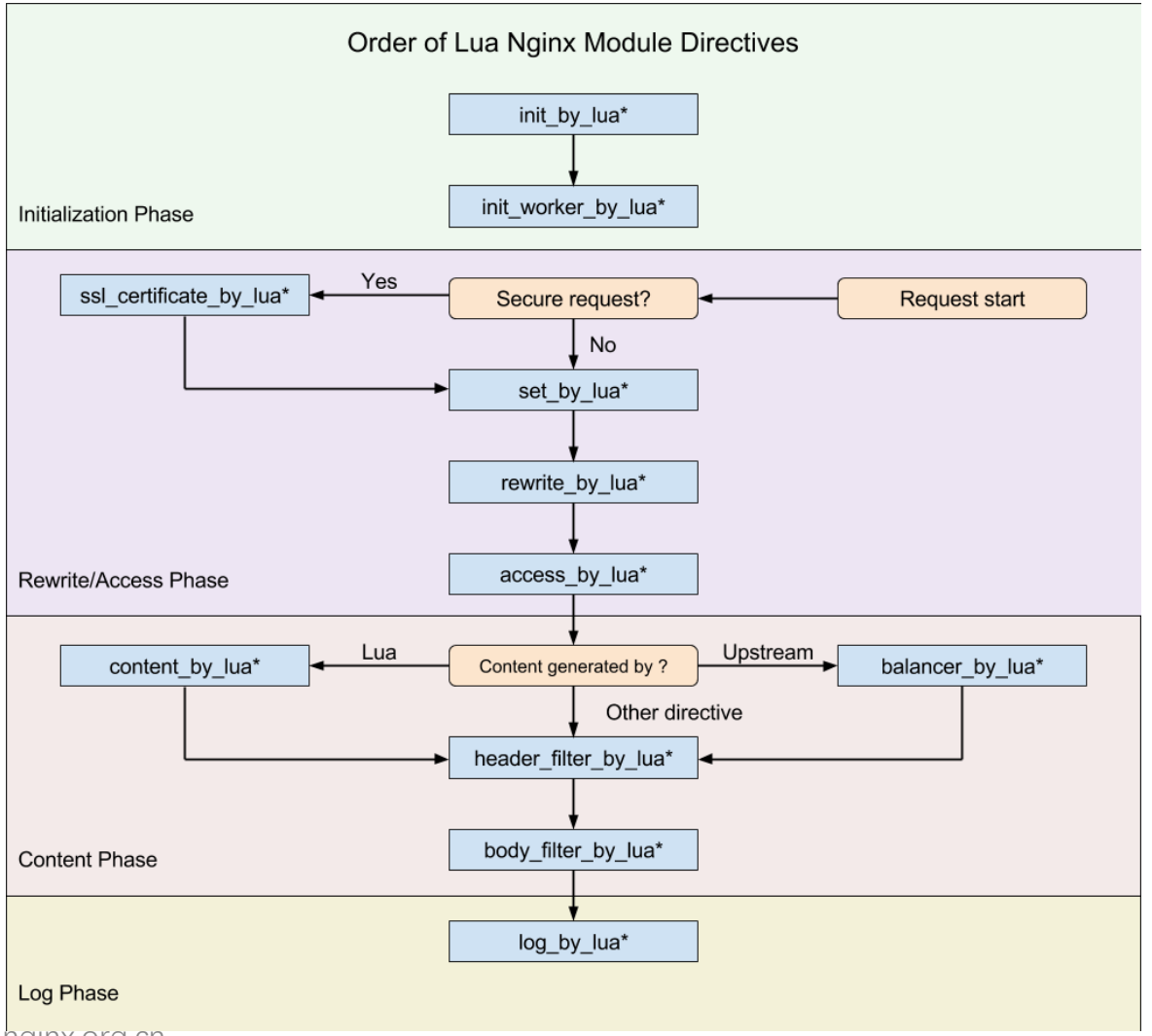
lua-5.4.0.tar.gz
2020-06-29, 342K
md5: dbf155764e5d433fc55ae80ea7060b60
sha1: 8cdbffa8a214a23d190d7c45f38c19518ae62e89

Openresty的运行机制



Nginx+Lua对业务层的处理

- Nginx启动时的预处理
- TLS证书认证
- 访问控制
- 生成HTTP响应
- 修改HTTP响应
- 负载均衡算法
- 记录审计日志



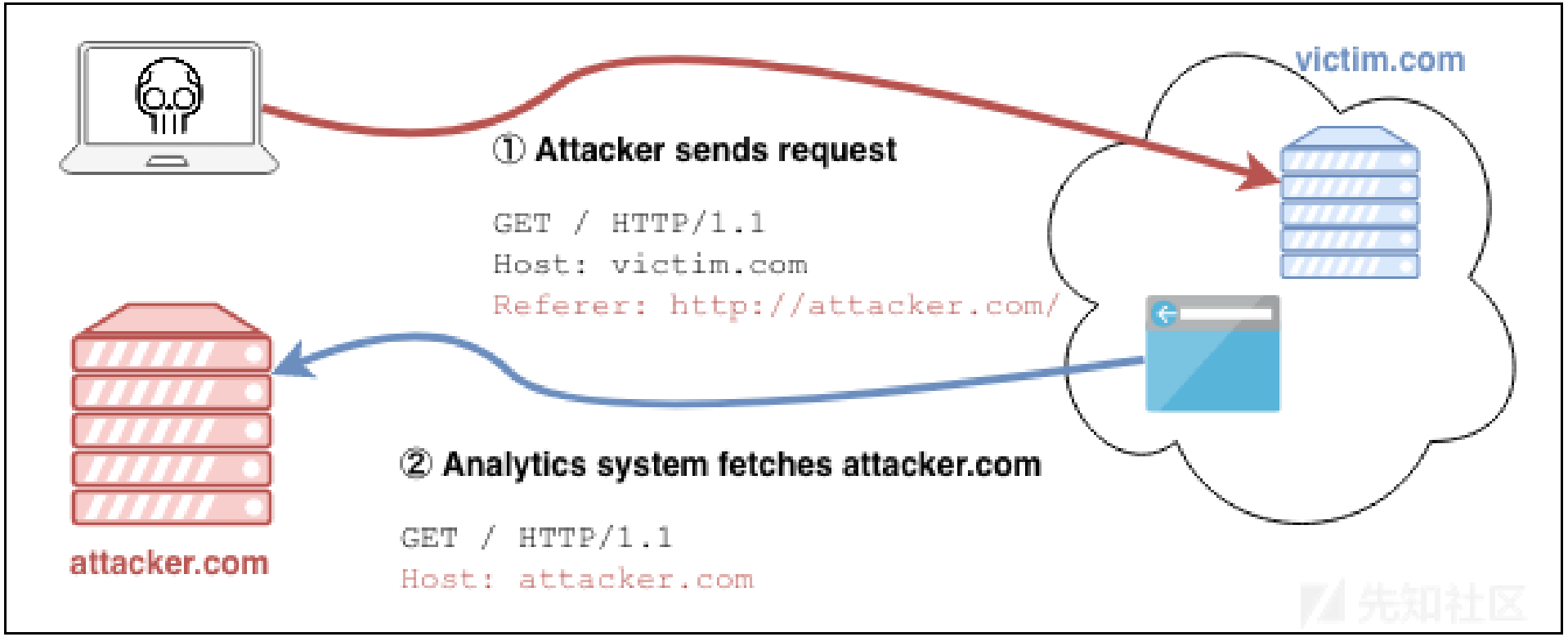
Lua检测恶意请求

- access_by_lua_file
  - ngx.var.取变量
    - ngx.req.get_headers()
  - ngx.re做正则匹配
  - 读取包体
    - 确保已经读取到包体
      - ngx.req.read_body()
      - lua_need_request_body on;
        - 在rewrite/access/content_by_lua前接收完包体
    - 基于文件
      - 获取文件名：ngx.req.get_body_file ()
    - 基于内存：
      - 获取包体
        - 变量：ngx.var.request_body
        - 函数：ngx.req.get_body_data ()、ngx.req.get_post_args ()
      - client_body_max_size = client_body_buffer_size
    - ngx.req.socket()取TCP套接字
      - receive读取包体

DDoS：统计请求频率

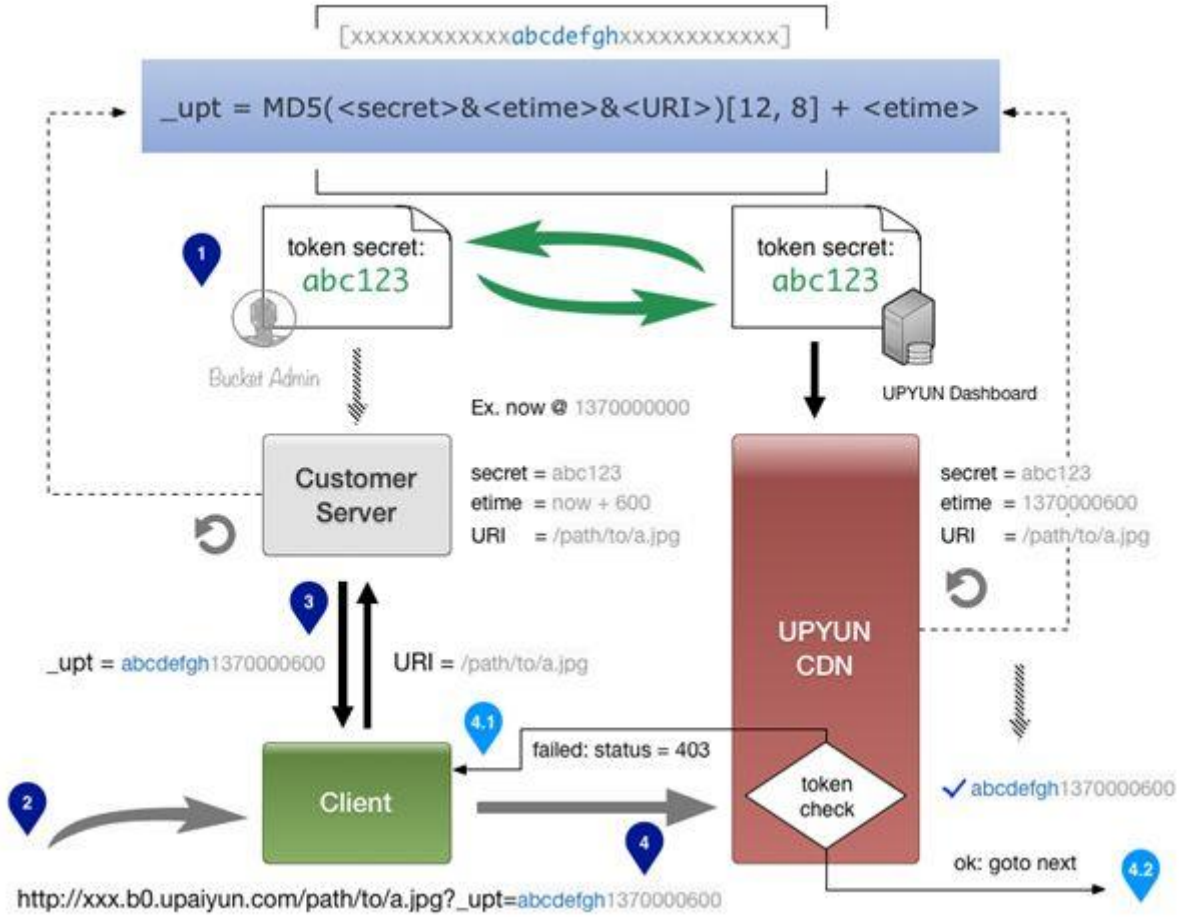
- 单进程统计
- 跨进程统计
  - lua_shared_dict
- 跨主机统计
  - redis
    - lua-resty-core

Referer防盗链


victim.com
① Attacker sends request
GET / HTTP/1.1
Host: victim.com
Referer: http://attacker.com/
② Analytics system fetches attacker.com
GET / HTTP/1.1
Host: attacker.com
attacker.com

Token防盗链

- 时间戳哈希摘要



用nginx搭建waf防火墙的实践

- waf应用层攻击对抗详解
- 恶意http请求的甄别
- **恶意客户端的防御控制**
- 降低waf防火墙的性能损耗

问题

- 如何配合浏览器提升安全性?
- 如何降低CC攻击的资源消耗?
- limit_rate是如何通过滑动窗口控制发送速度的?
- limit_req是如何基于leacky bucket控制QPS的?
- limit_conn是如何限制并发连接的?
- allow与deny是如何设置IP黑、白名单的?
- 如何基于地理位置限制客户端?

ngx_http_referer_module模块

Syntax:	**referer_hash_bucket_size** size;
Default:	referer_hash_bucket_size 64;
Context:	server, location

Syntax:	**referer_hash_max_size** size;
Default:	referer_hash_max_size 2048;
Context:	server, location

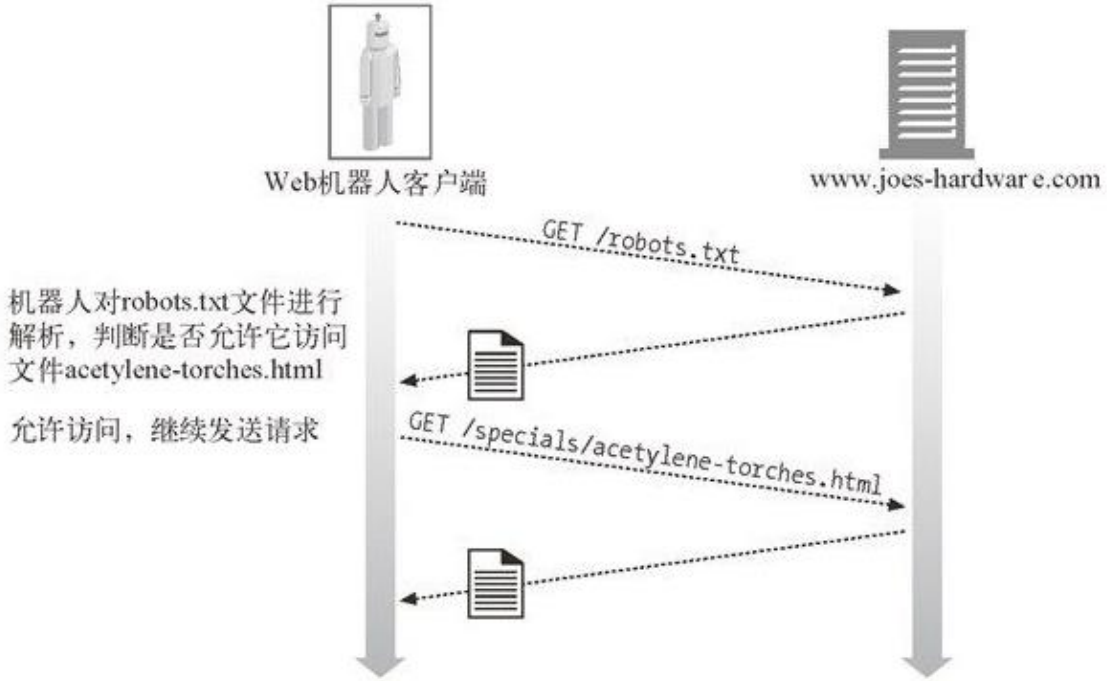
Syntax:	**valid_referers** none | blocked | server_names | string ...;
Default:	—
Context:	server, location

$invalid_referer变量

- $invalid_referer 为空时，valid_referrers与Referer头部
  - none：不传Referer头部
  - blocked：Referer头部存在，但由于防火墙的存在，值为空
  - server_names：Referer值与server_name指令中的域名匹配
  - 含有*通配符的URL
  - ~打头的正则表达式

网络爬虫

- 合规的爬虫
  - HTTP头部
    - User-Agent：识别是哪类爬虫
    - From：提供爬虫机器人管理者的邮箱地址
    - Accept：爬虫对哪些资源类型感兴趣
    - Referer：包含了当前请求的页面 URI
  - Robots exclusion protocol：http://www.robotstxt.org/orig.html
  - robots.txt 文件内容
    - User-agent：允许哪些机器人
    - Disallow：禁止访问特定目录
    - Crawl-delay：访问间隔秒数
    - Allow：抵消 Disallow 指令
    - Sitemap：指出站点地图的 URI
- 恶意爬虫



Web框架漏洞

- 不安全的HTTP方法
  - TRACE
    - $request_method
- Apache Expect漏洞
- 限制Cookie场景
  - http only

ngx_http_headers_module模块

Syntax:	**add_header** name value [always];
Default:	—
Context:	http, server, location, if in location

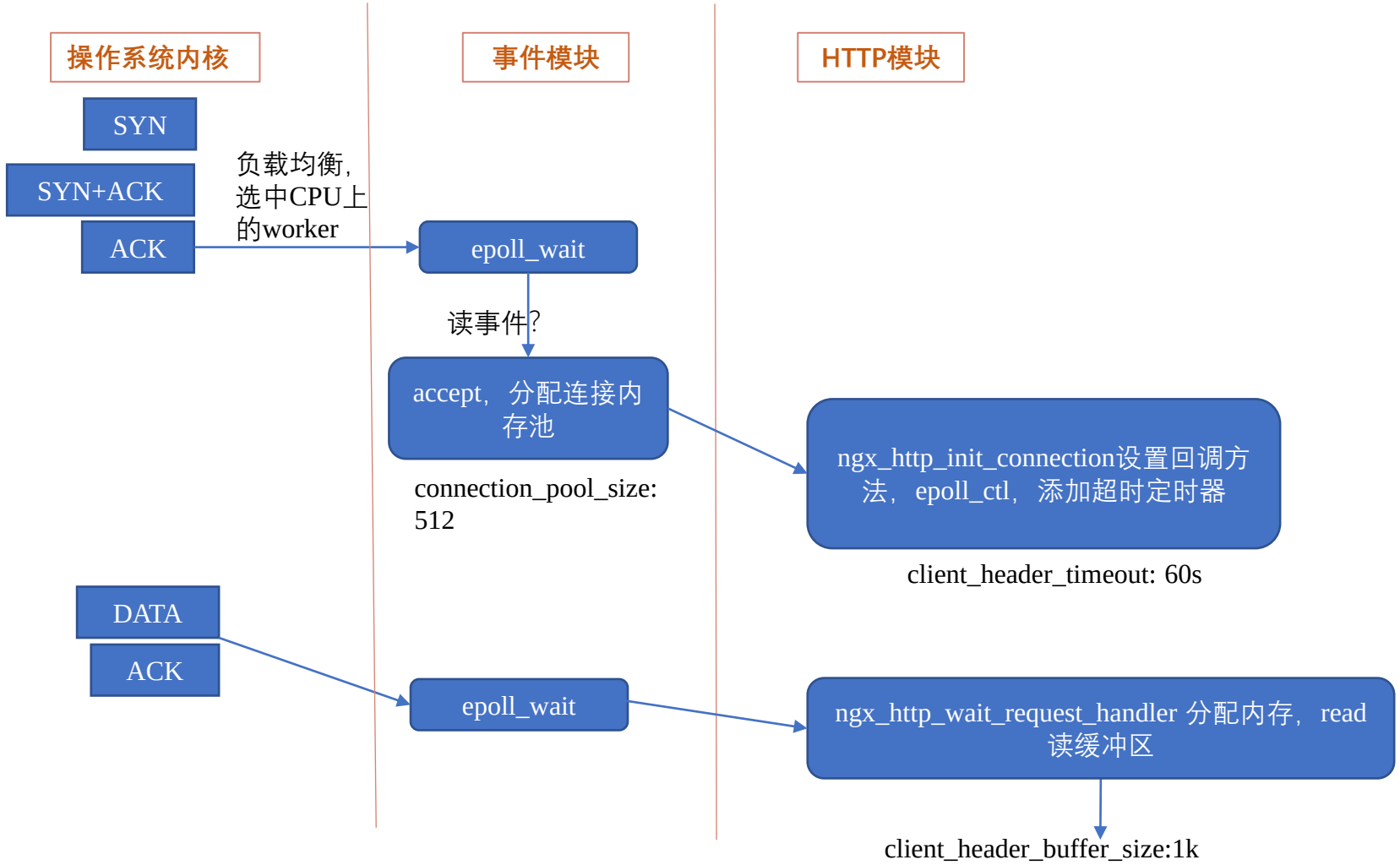
- **对以下响应添加头部**
  - 200, 201 , 204, 206, 301, 302, 303, 304, 307 , 308
  - always：对所有响应添加头部
- **应用场景举例：**
  - 防御点击劫持
    - add_header X-Frame-Options sameorigin always;
  - 阻止非法的跨域访问
    - add_header Access-Control-Allow-Origin a.b.c;
  - 阻止XSS中内联JS脚本加载
    - add_header X-XSS-Protection "1; mode=block";

配合浏览器限制Cookie

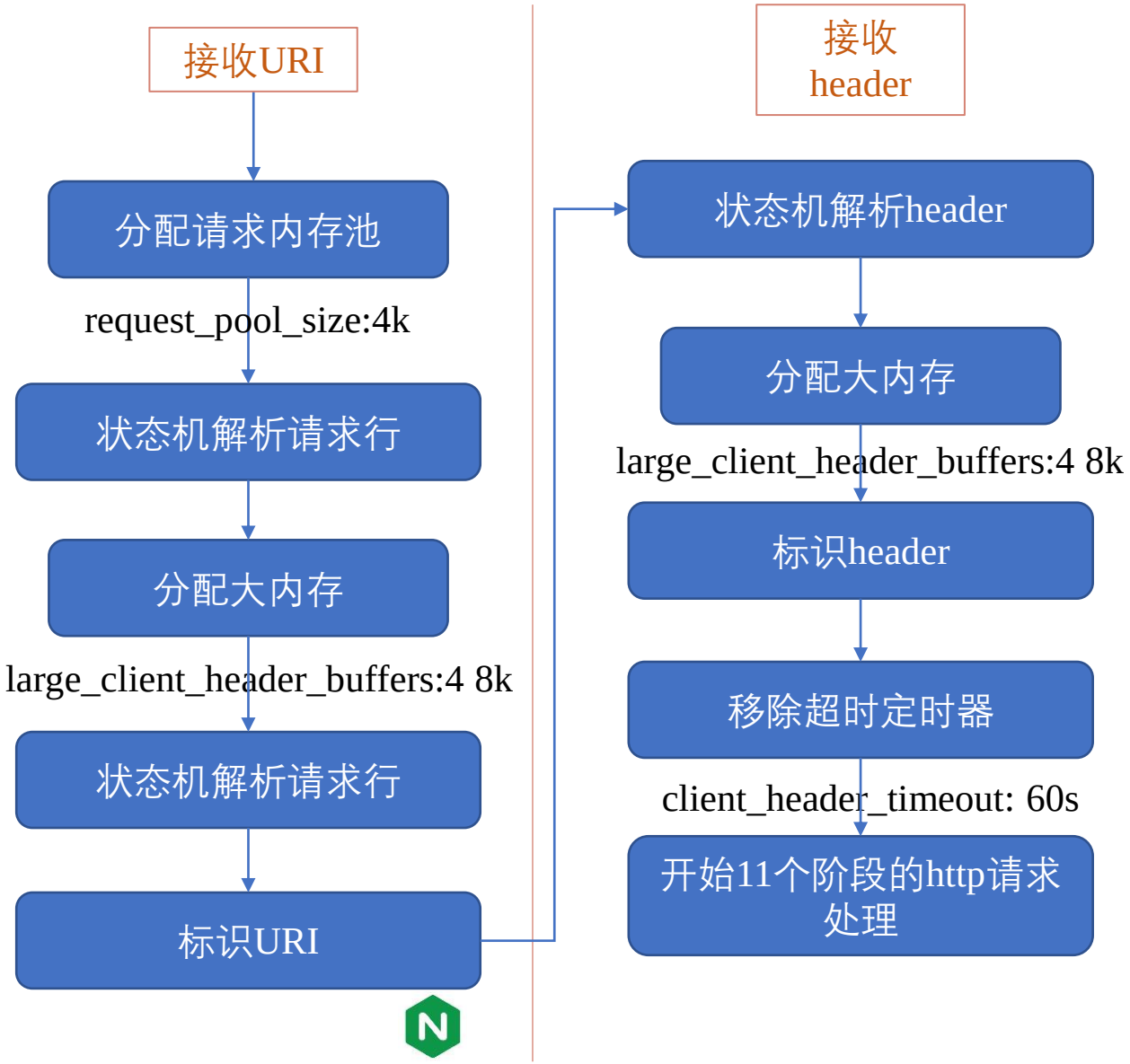
- **Set-Cookie属性**
  - Secure：仅当使用TLS协议时，才能将Cookie头部发送给服务器
  - HttpOnly：JS不能读取Cookie
  - SameSite=Strict：跨站请求不会携带 Cookie
  - Path=：仅对相应URL请求才携带Cookie
  - Domain：仅当访问相应域名时才携带Cookie
- **proxy_cookie_flags（1.19.3）**
  - proxy_cookie_flags secure httponly;

Syntax:	**proxy_cookie_flags** off | cookie [flag ...];
Default:	proxy_cookie_flags off;
Context:	http, server, location

接收请求事件模块


操作系统内核
事件模块
HTTP模块
SYN
SYN+ACK
ACK
负载均衡，选中CPU上的worker
epoll_wait
读事件?
accept，分配连接内存池
connection_pool_size: 512
ngx_http_init_connection设置回调方法，epoll_ctl，添加超时定时器
client_header_timeout: 60s
DATA
ACK
epoll_wait
ngx_http_wait_request_handler 分配内存，read读缓冲区
client_header_buffer_size:1k

接收请求 HTTP 模块


接收URI
分配请求内存池
request_pool_size:4k
状态机解析请求行
分配大内存
large_client_header_buffers:4 8k
状态机解析请求行
标识URI
接收
header
状态机解析header
分配大内存
large_client_header_buffers:4 8k
标识header
移除超时定时器
client_header_timeout: 60s
开始11个阶段的http请求处理

应对CC攻击：408超时

- 传输层超时
  - client_body_timeout　默认60秒
  - send_timeout　默认60秒
- 应用层超时设置
  - client_header_timeout　默认60秒
  - auth_delay　默认不加限制
  - http2_idle_timeout　默认3分钟
  - lingering_time　默认30秒
  - lingering_timeout　默认5秒

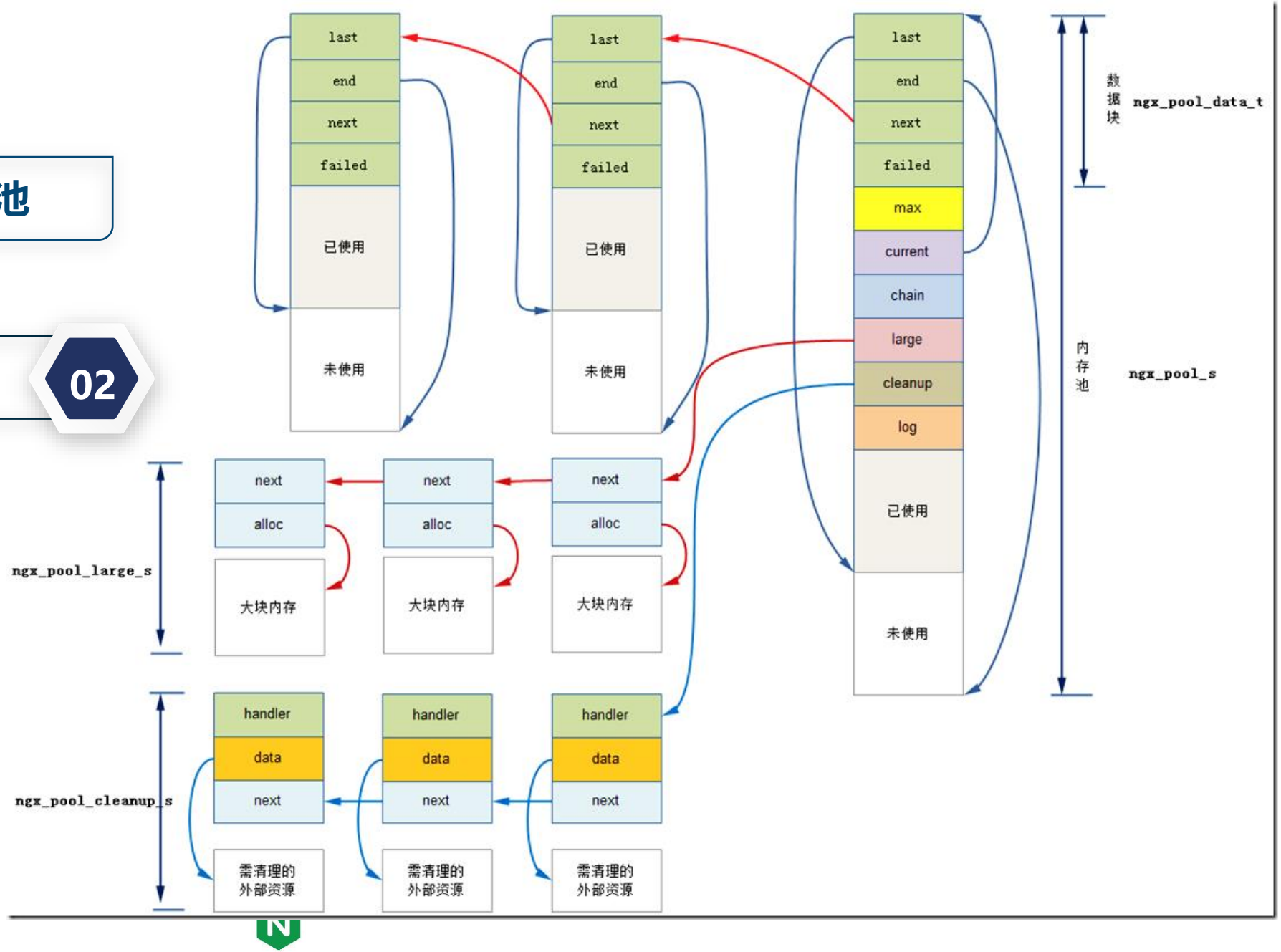
应对CC攻击：控制内存

- 内存占用
  - worker_connections 默认512
  - client_header_buffer_size 默认1KB
  - large_client_header_buffers 默认4*8KB
  - client_body_buffer_size 默认16KB
  - request_pool_size 默认4KB
  - connection_pool_size 默认512B
- 磁盘占用
  - client_max_body_size 默认1MB

内存池

01 连接内存池

02 请求内存池



limit_rate限速：仅针对1个连接

- limit_rate指令
  - limit_rate_after
- $limit_rate变量
  - set指令
- X-Accel-Limit-Rate上游头部

Syntax:	**limit_rate** rate;
Default:	limit_rate 0;
Context:	http, server, location, if in location

Syntax:	**limit_rate_after** size;
Default:	limit_rate_after 0;
Context:	http, server, location, if in location

Syntax:	**set** $variable value;
Default:	—
Context:	server, location, if

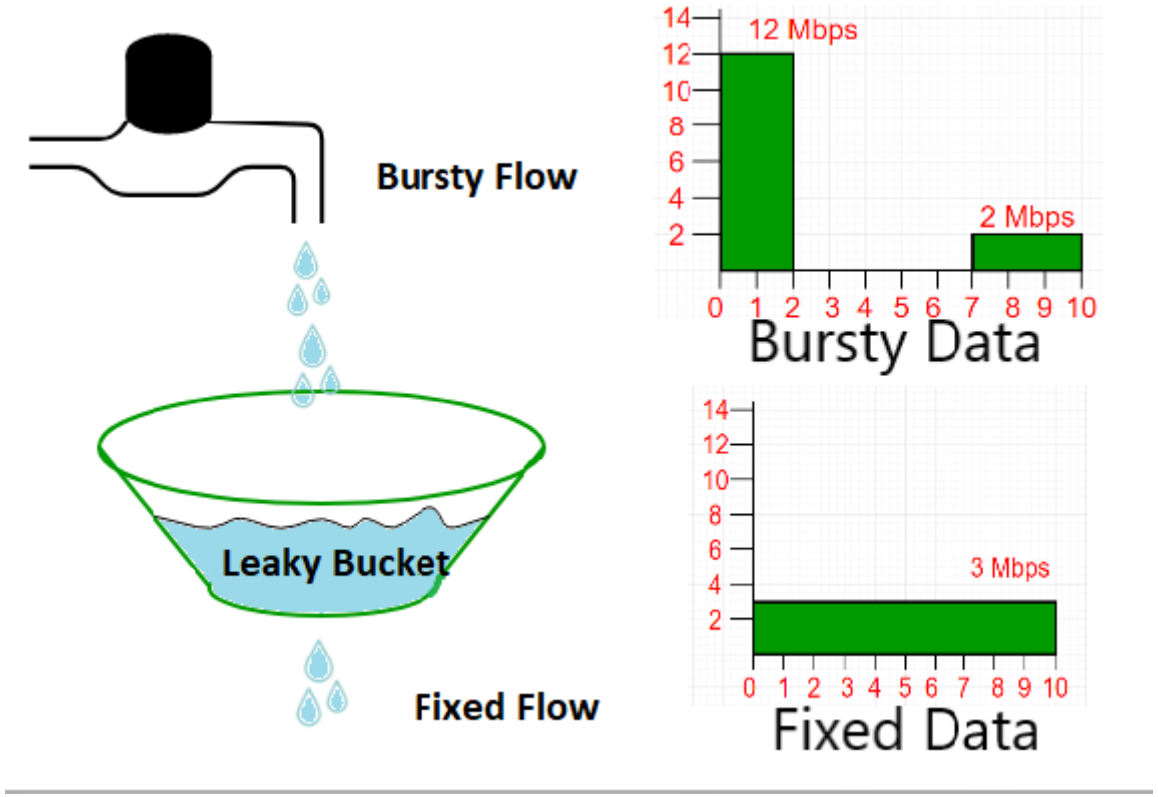
limit_rate的工作原理

- 在ngx_http_write_filter_module过滤模块中生效
  - 无论配置的r/s或者r/m，都会以毫秒为单位延迟发送
  - r->start_sec是从接收请求算起

```
limit = (off_t) r->limit_rate * (ngx_time() - r->start_sec + 1)
        - (c->sent - r->limit_rate_after);

if (limit <= 0) {
    c->write->delayed = 1;
    delay = (ngx_msec_t) (- limit * 1000 / r->limit_rate + 1);
    ngx_add_timer(c->write, delay);

    c->buffered |= NGX_HTTP_WRITE_BUFFERED;

    return NGX_AGAIN;
}
```

limit_req与leaky bucket

- **平滑限速**
  - 保持固定的出流量
  - 设定bucket容量
  - 漫过bucket的策略



limit_req指令（1）

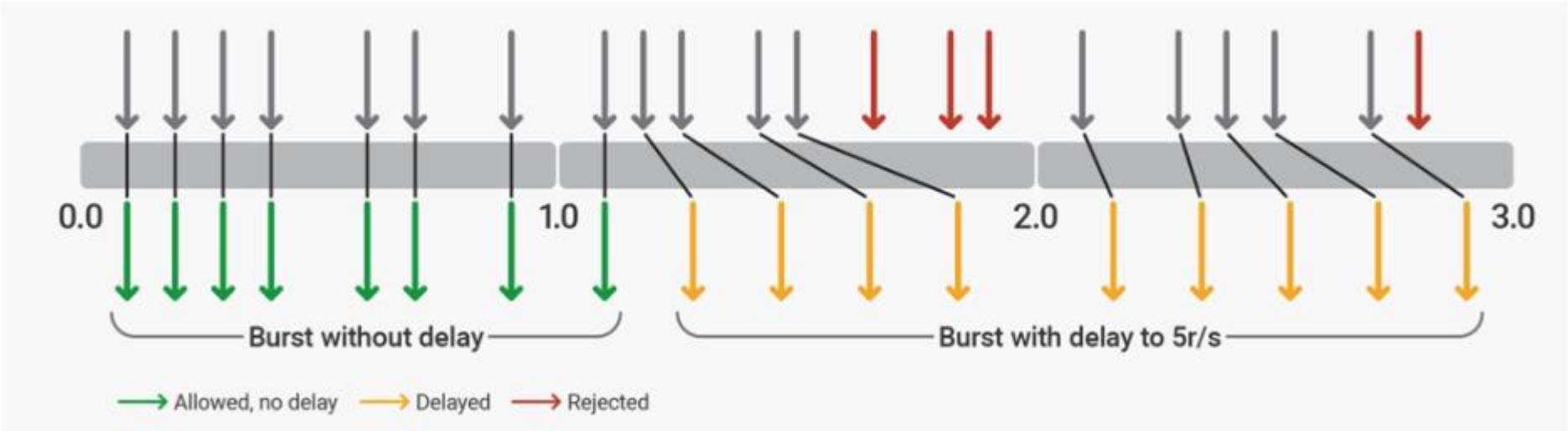
- limit_req_zone
  - zone
    - LRU算法淘汰
      - 60秒以上未使用节点
- limit_req
  - 增加桶容量
    - burst
      - 默认0
  - delay VS nodelay

Syntax:	**limit_req** zone=name [burst=number] [nodelay | delay=number];
Default:	—
Context:	http, server, location

Syntax:	**limit_req_zone** key zone=name:size rate=rate [~~sync~~];
Default:	—
Context:	http

匀速VS效率

```
limit_req_zone $binary_remote_addr zone=ip:10m rate=5r/s;
server {
        listen 80;
        location / {
                limit_req zone=ip burst=12 delay=8;
                proxy_pass http://website;
        }
}
```



limit_req指令（2）

- **客户端返回错误码**
  - limit_req_status
- **对某些请求不加限制**
  - limit_req_dry_run
- **是否记录日志**
  - limit_req_log_level

Syntax:	**limit_req_dry_run** on | off;
Default:	limit_req_dry_run off;
Context:	http, server, location

Syntax:	**limit_req_status** code;
Default:	limit_req_status 503;
Context:	http, server, location

Syntax:	**limit_req_log_level** info | notice | warn | error;
Default:	limit_req_log_level error;
Context:	http, server, location

limit_req的zone需要多大?

- 红黑树节点
- state节点
  - 56+key长度
    - $binary_remote_addr
      - IPv4：4字节
      - IPv6：16字节

```
struct ngx_rbtree_node_s {
    ngx_rbtree_key_t       key;
    ngx_rbtree_node_t      *left;
    ngx_rbtree_node_t      *right;
    ngx_rbtree_node_t      *parent;
    u_char                 color;
    u_char                 data;
};

typedef struct {
    u_char                 color;
    u_char                 dummy;
    u_short                len;
    ngx_queue_t                queue;
    ngx_msec_t                 last;
    ngx_uint_t                 excess;
    ngx_uint_t                 count;
    u_char                 data[1];
} ngx_http_limit_req_node_t;
```


limit_conn与限速

- **共享内存**
  - 无LRU淘汰
  - 并发连接数为0时淘汰
- **HTTP/2与SPDY**
  - HTTP请求代替TCP连接

Syntax:	**limit_conn_zone** key zone=name:size;
Default:	—
Context:	http

Syntax:	**limit_conn** zone number;
Default:	—
Context:	http, server, location

limit_conn指令（2）

- 客户端返回错误码
  - limit_conn_status
- 对某些请求不加限制
  - limit_conn_dry_run
- 是否记录日志
  - limit_conn_log_level

Syntax:	**limit_conn_dry_run** on | off;
Default:	limit_conn_dry_run off;
Context:	http, server, location

Syntax:	**limit_conn_status** code;
Default:	limit_conn_status 503;
Context:	http, server, location

Syntax:	**limit_conn_log_level** info | notice | warn | error;
Default:	limit_conn_log_level error;
Context:	http, server, location

limit_conn的zone需要多大?

- 红黑树节点
- state节点
  - 4+key长度
    - $binary_remote_addr
      - IPv4：4字节
      - IPv6：16字节

```
struct ngx_rbtree_node_s {
    ngx_rbtree_key_t       key;
    ngx_rbtree_node_t      *left;
    ngx_rbtree_node_t      *right;
    ngx_rbtree_node_t      *parent;
    u_char                 color;
    u_char                 data;
};

typedef struct {
    u_char                 color;
    u_char                 len;
    u_short                conn;
    u_char                 data[1];
} ngx_http_limit_conn_node_t;
```

access模块：IP访问控制

- **白名单**
  - allow
- **黑名单**
  - deny
- **与或关系**
  - satisfy

Syntax:	**allow** address | CIDR | unix: | all;
Default:	—
Context:	http, server, location, limit_except

Syntax:	**deny** address | CIDR | unix: | all;
Default:	—
Context:	http, server, location, limit_except

Syntax:	**satisfy** all | any;
Default:	satisfy all;
Context:	http, server, location

基于地理位置的访问控制

- geo模块：基于IP网段
  - ngx_http_geo_module
- geoip模块：基于MaxMind数据库
  - ngx_http_geoip_module

根据客户端地址创建新变量：geo 模块

Syntax: **geo** [*$address*] *$variable* { ... }

Default: —

Context: http

功能

根据IP地址创建新变量

模块

ngx_http_geo_module,

默认编译进nginx,

通过--without-http_geo_module禁用

根据客户端地址创建新变量：geo 模块

规则

- **如果geo指令后不输入$address，那么默认使用$remote_addr变量作为IP地址**
- **{} 内的指令匹配：优先最长匹配**
  - 通过IP地址及子网掩码的方式，定义IP范围，当IP地址在范围内时新变量使用其后的参数值
  - default指定了当以上范围都未匹配上时，新变量的默认值
  - 通过proxy指令指定可信地址（参考realip模块），此时remote_addr的值为X-Forwarded-For头部值中最后一个IP地址
  - proxy_recursive允许循环地址搜索
  - include，优化可读性
  - delete删除指定网络

geo 模块示例

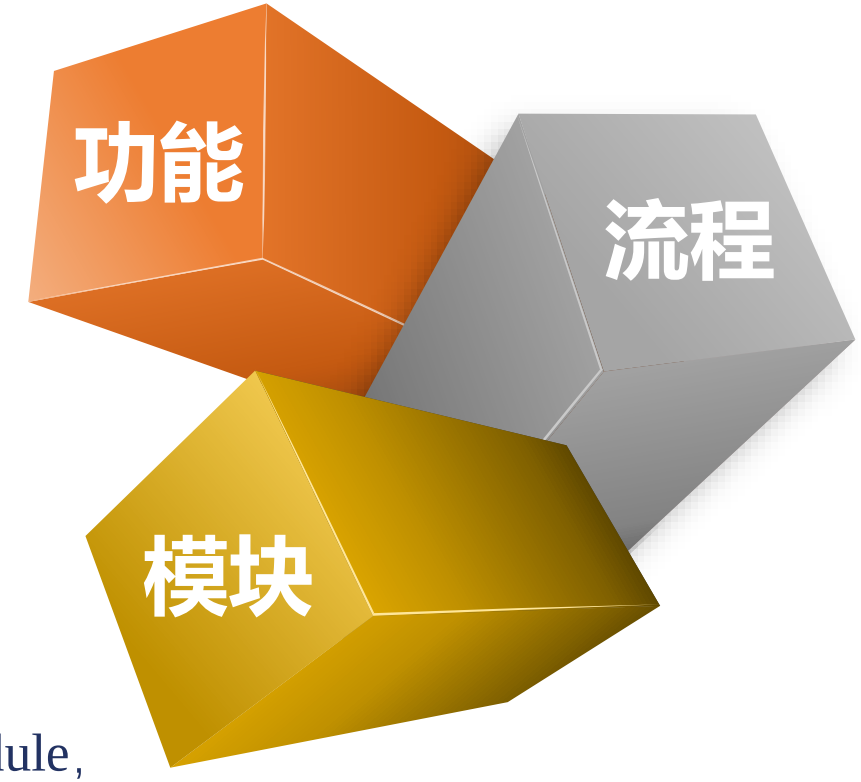
```
geo $country {
    default        ZZ;
    #include        conf/geo.conf;
    proxy          116.62.160.193;

    127.0.0.0/24    US;
    127.0.0.1/32    RU;
    10.1.0.0/16     RU;
    192.168.1.0/24 UK;
}
```

问题：以下命令执行时，变量country的值各为多少？（proxy为客户端地址）

- **curl -H 'X-Forwarded-For: 10.1.0.0,127.0.0.2' geo.taohui.tech**
- **curl -H 'X-Forwarded-For: 10.1.0.0,127.0.0.1' geo.taohui.tech**
- **curl -H 'X-Forwarded-For: 10.1.0.0,127.0.0.1,1.2.3.4' geo.taohui.tech**

基于MaxMind数据库从客户端地址获取变量：geoip模块

根据IP地址创建新变量



ngx_http_geoip_module，
默认未编译进nginx，
通过--with-http_geoip_module禁用

- 安装MaxMind里geoip的C开发库（https://dev.maxmind.com/geoip/legacy/downloadable/）
- 编译nginx时带上--with-http_geoip_module参数
- 下载MaxMind中的二进制地址库
- 使用geoip_country或者geoip_city指令配置好nginx.conf
- 运行（或者升级nginx）

geoip_country 指令提供的变量

Syntax: **geoip_country** *file*;
Default: —
Context: http

Syntax: **geoip_proxy** *address* | *CIDR*;
Default: —
Context: http

变量

- **$geoip_country_code**
  - 两个字母的国家代码，比如CN或者US
- **$geoip_country_code3**
  - 三个字母的国家代码，比如CHN或者USA
- **$geoip_country_name**
  - 国家名称，例如“China”, “United States”.

geoip_city 指令提供的变量

Syntax: **geoip_city** *file*;

Default: —

Context: http

geoip_city 指令提供的变量

- $geoip_latitude：纬度
- $geoip_longitude：经度
- $geoip_city_continent_code：属于全球哪个洲，例如EU或者AS
- 与geoip_country指令生成的变量重叠
  - $geoip_city_country_code：两个字母的国家代码，比如CN或者US
  - $geoip_city_country_code3：三个字母的国家代码，比如CHN或者USA
  - $geoip_city_country_name：国家名称，例如“China”, “United States”
- $geoip_region：洲或者省的编码，例如02
- $geoip_region_name：洲或者省的名称，例如Zhejiang或者Saint Petersburg
- $geoip_city：城市名
- $geoip_postal_code：邮编号
- $geoip_area_code：仅美国使用的电话区号，例如408
- $geoip_dma_code：仅美国使用的DMA编号，例如807

用nginx搭建waf防火墙的实践

- waf应用层攻击对抗详解
- 恶意http请求的甄别
- 恶意客户端的防御控制
- **降低waf防火墙的性能损耗**

问题

- 优化nginx waf，可以从哪些方面入手?
- 如何优化正则匹配速度?
- 如何提升CPU使用效率?
- 应用层/传输层网络协议如何优化?
- 如何提升内存分配速度?
- 如何降低写日志带来的性能损耗?
- 如何定位性能问题?

waf nginx性能优化

- **部署Nginx集群**
- **提升单机性能**
  - 硬件规格提升
  - waf本职工作的性能优化
    - 提升正则匹配速度
    - 提升CPU的使用效率
    - 优化网络协议栈
    - 提升内存分配速度
    - 减少access.log日志的磁盘IO
  - 7层LB性能优化
    - 缓存/CDN

PCRE库的优化

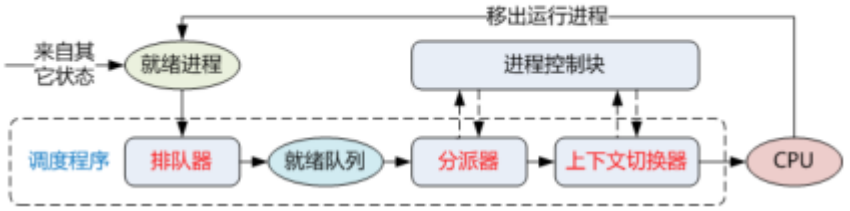
- PCRE：Perl-compatible Regular Expressions
  - jit即时编译技术：just in time
- 编译configure
  - 指定最新版源码：--with-pcre=
  - 指定编译选项：--with-pcre-opt=OPTIONS
  - 开启jit：--with-pcre-jit
- 启动jit

Syntax:	**pcre_jit** on | off;
Default:	pcre_jit off;
Context:	main

设置worker进程数量

- 分时操作系统：并行与串行
  - 把进程的运行时间分为一段段的时间片
  - OS调度系统依次选择每个进程，最多执行时间片指定的时长
- worker进程的工作方式
  - 非阻塞
  - 多路复用

■ ①排队器②分派器③上下文切换器

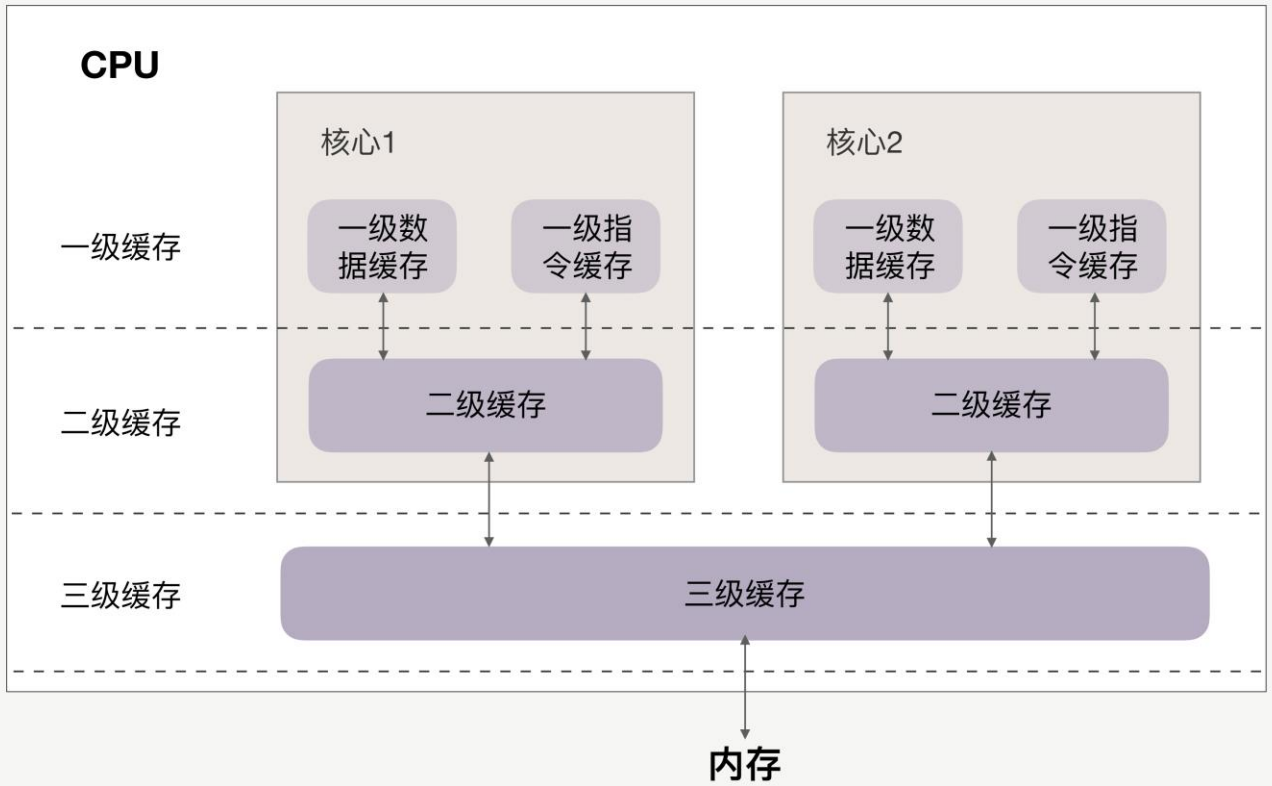


Syntax:	**worker_processes** number | auto;
Default:	worker_processes 1;
Context:	main

Syntax:	**master_process** on | off;
Default:	master_process on;
Context:	main

绑定worker到指定CPU

Syntax:	**worker_cpu_affinity** cpumask ...; **worker_cpu_affinity** auto [cpumask];
Default:	—
Context:	main

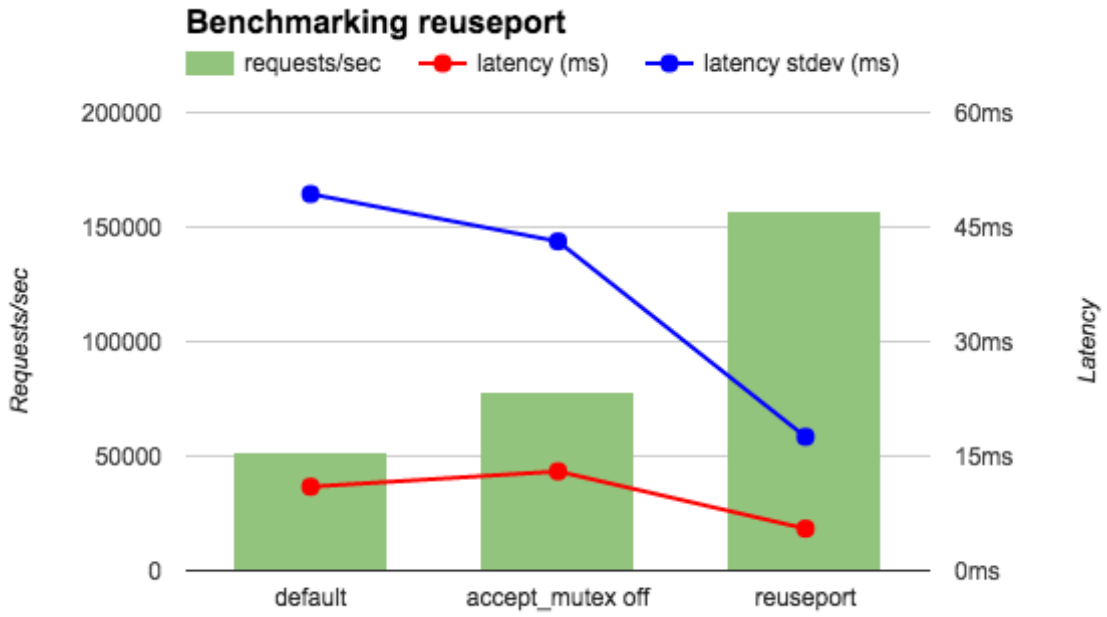


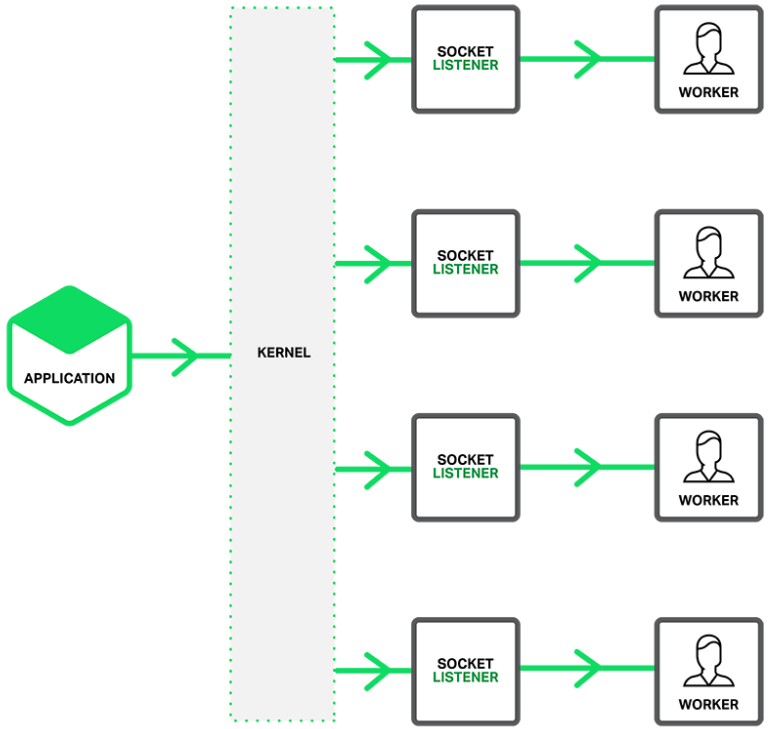
主动切换 VS 被动切换

- Pidstat
  - cswch/s： 主动切换
  - nvcswch/s:时间片用尽

Syntax:	**worker_priority** number;
Default:	worker_priority 0;
Context:	main

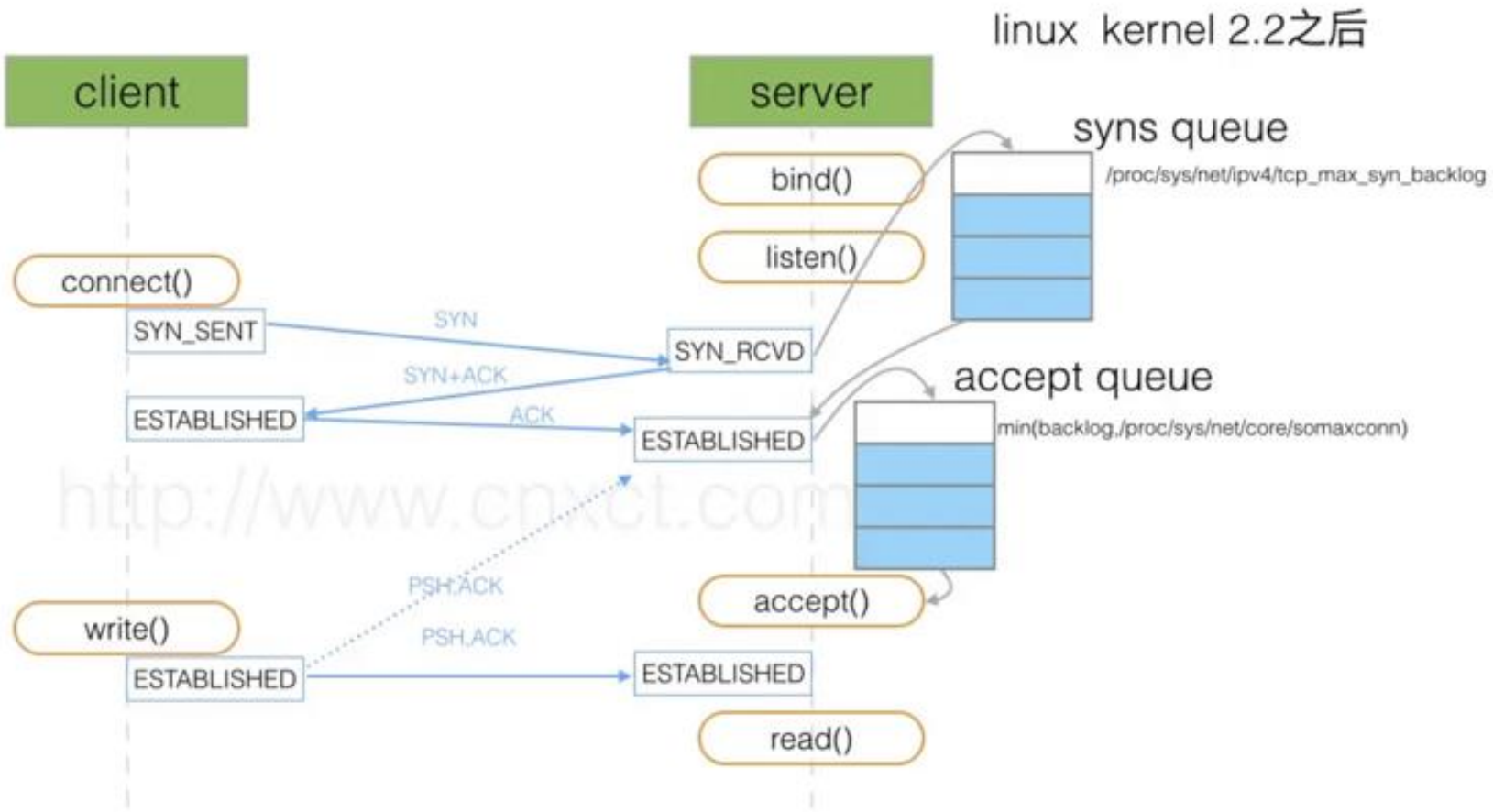
worker进程间负载均衡





延迟处理新连接

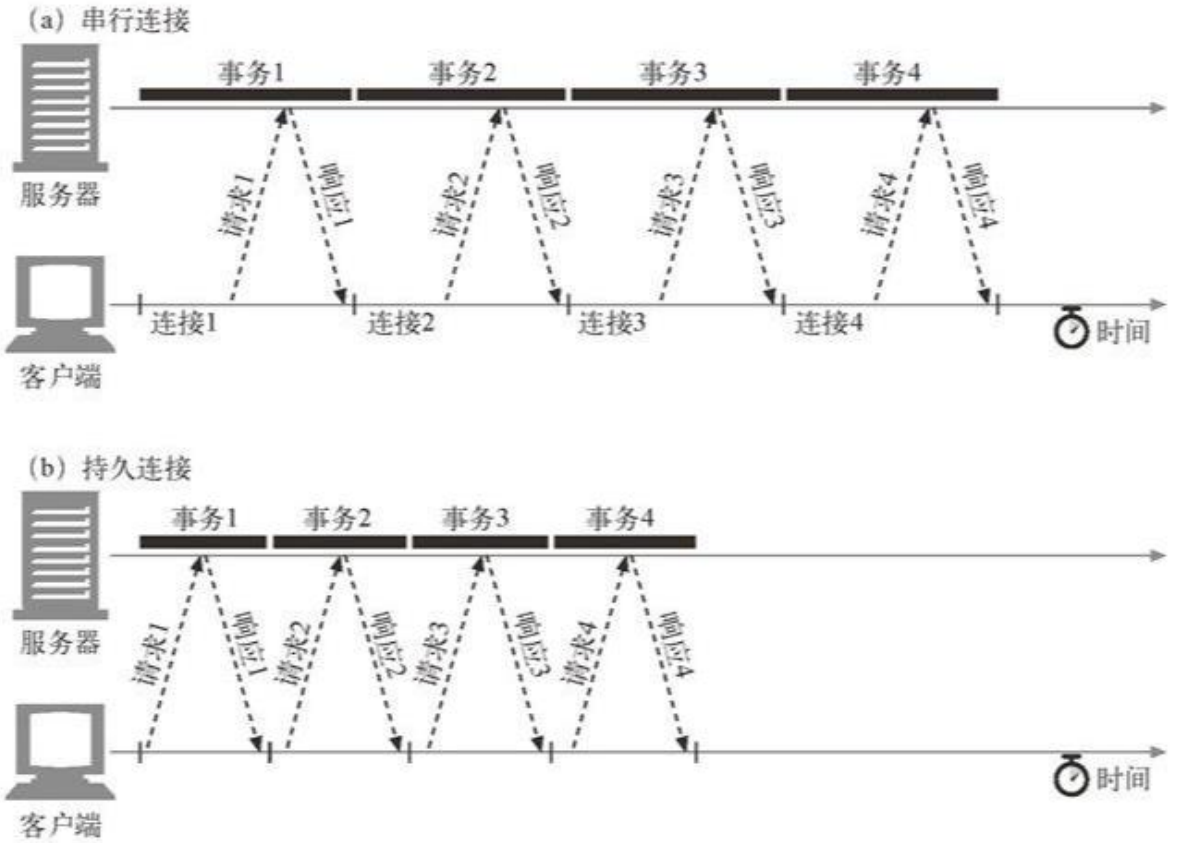
Syntax:	**listen** address[:port] [deferred];
Default:	listen *:80 | *:8000;
Context:	server



HTTP长连接

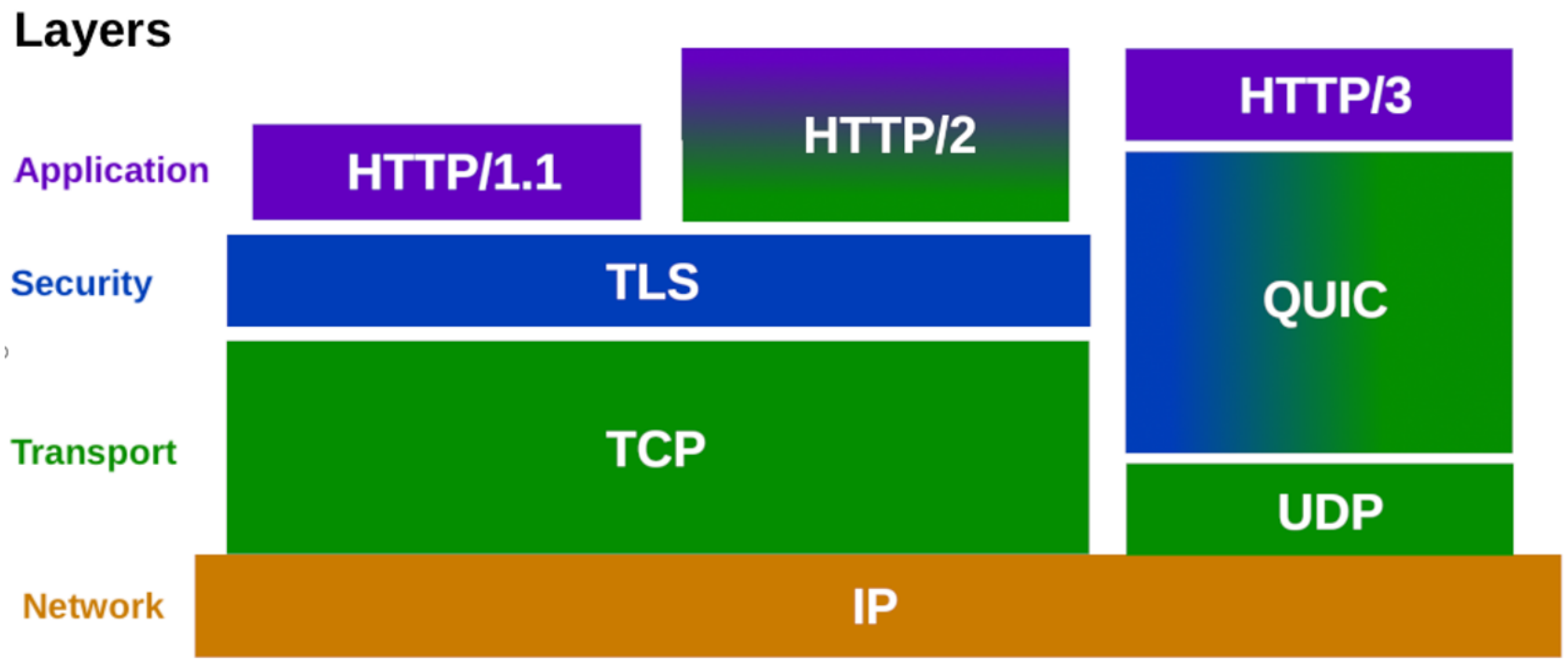
Syntax:	**keepalive_requests** number;
Default:	keepalive_requests 100;
Context:	http, server, location

Syntax:	**keepalive_requests** number;
Default:	keepalive_requests 100;
Context:	upstream



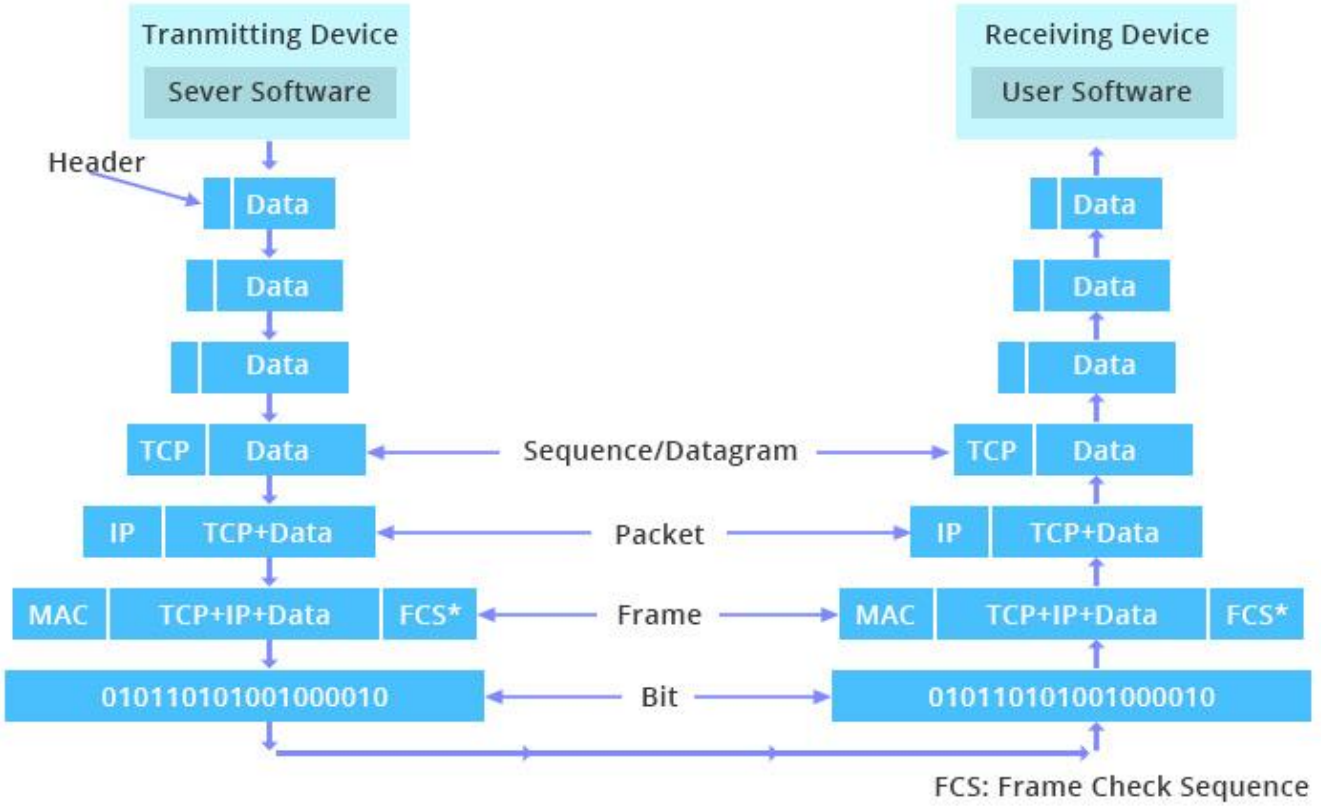
升级http协议

- 向前兼容http/1.x协议
- 多路复用
- 降低建链成本
- 更高效的信息压缩比



应用层提升有效信息量

Syntax:	**postpone_output** size;
Default:	postpone_output 1460;
Context:	http, server, location



gzip压缩

功能：
通过实时压缩http包体，提升网络传输效率
模块：
ngx_http_gzip_module，通过--without-http_gzip_module禁用模块

Syntax:	**gzip** on | off;
Default:	gzip off;
Context:	http, server, location, if in location

压缩哪些请求的响应?

Syntax:	**gzip_types** mime-type ...;
Default:	gzip_types text/html;
Context:	http, server, location

Syntax:	**gzip_min_length** length;
Default:	gzip_min_length 20;
Context:	http, server, location

Syntax:	**gzip_disable** regex ...;
Default:	—
Context:	http, server, location

Syntax:	**gzip_http_version** 1.0 | 1.1;
Default:	gzip_http_version 1.1;
Context:	http, server, location

是否压缩上游的响应

Syntax:	**gzip_proxied** off | expired | no-cache | no-store | private | no_last_modified | no_etag | auth | any ...;
Default:	gzip_proxied off;
Context:	http, server, location

- off
  - 不压缩来自上游的响应
- expired
  - 如果上游响应中含有Expires头部，且其值中的时间与系统时间比较后确定不会缓存，则压缩响应
- no-cache
  - 如果上游响应中含有“Cache-Control”头部，且其值含有”no-cache”值，则压缩响应
- no-store
  - 如果上游响应中含有“Cache-Control”头部，且其值含有”no-store”值，则压缩响应
- private
  - 如果上游响应中含有“Cache-Control”头部，且其值含有”private”值，则压缩响应
- no_last_modified
  - 如果上游响应中没有 “Last-Modified” 头部，则压缩响应
- no_etag
  - 如果上游响应中没有 “ETag” 头部，则压缩响应
- auth
  - 如果客户端请求中含有 “Authorization” 头部，则压缩响应
- any
  - 压缩所有来自上游的响应

其他压缩参数

Syntax:	**gzip_comp_level** level;
Default:	gzip_comp_level 1;
Context:	http, server, location

Syntax:	**gzip_buffers** number size;
Default:	gzip_buffers 32 4k|16 8k;
Context:	http, server, location

Syntax:	**gzip_vary** on | off;
Default:	gzip_vary off;
Context:	http, server, location

扩大连接数

- 操作系统全局
  - fs.file-max
    - 操作系统可使用的最大句柄数
  - 使用fs.file-nr可以查看当前已分配、正使用、上限
    - fs.file-nr = 21632 0 40000500
- 限制用户
  - /etc/security/limits.conf
    - nofile
- 限制进程

Syntax:	**worker_rlimit_nofile** number;
Default:	—
Context:	main

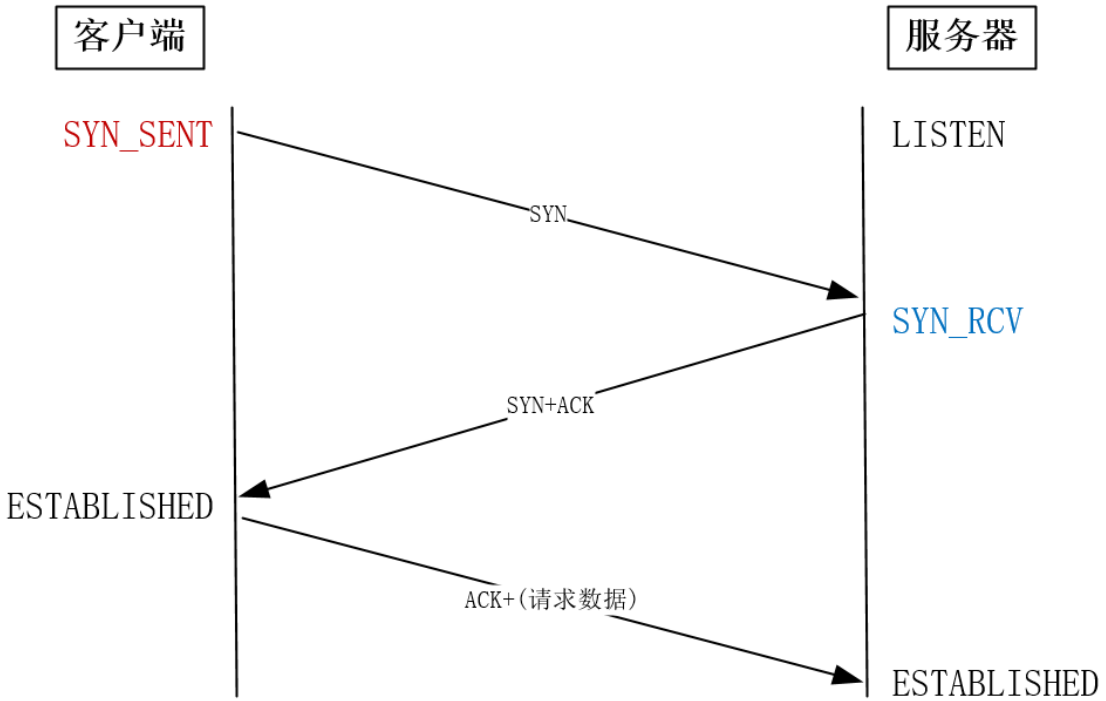
Syntax:	**worker_connections** number;
Default:	worker_connections 512;
Context:	events

客户端三次握手超时

Syntax:	**proxy_connect_timeout** time;
Default:	proxy_connect_timeout 60s;
Context:	http, server, location

Syntax:	**proxy_connect_timeout** time;
Default:	proxy_connect_timeout 60s;
Context:	stream, server

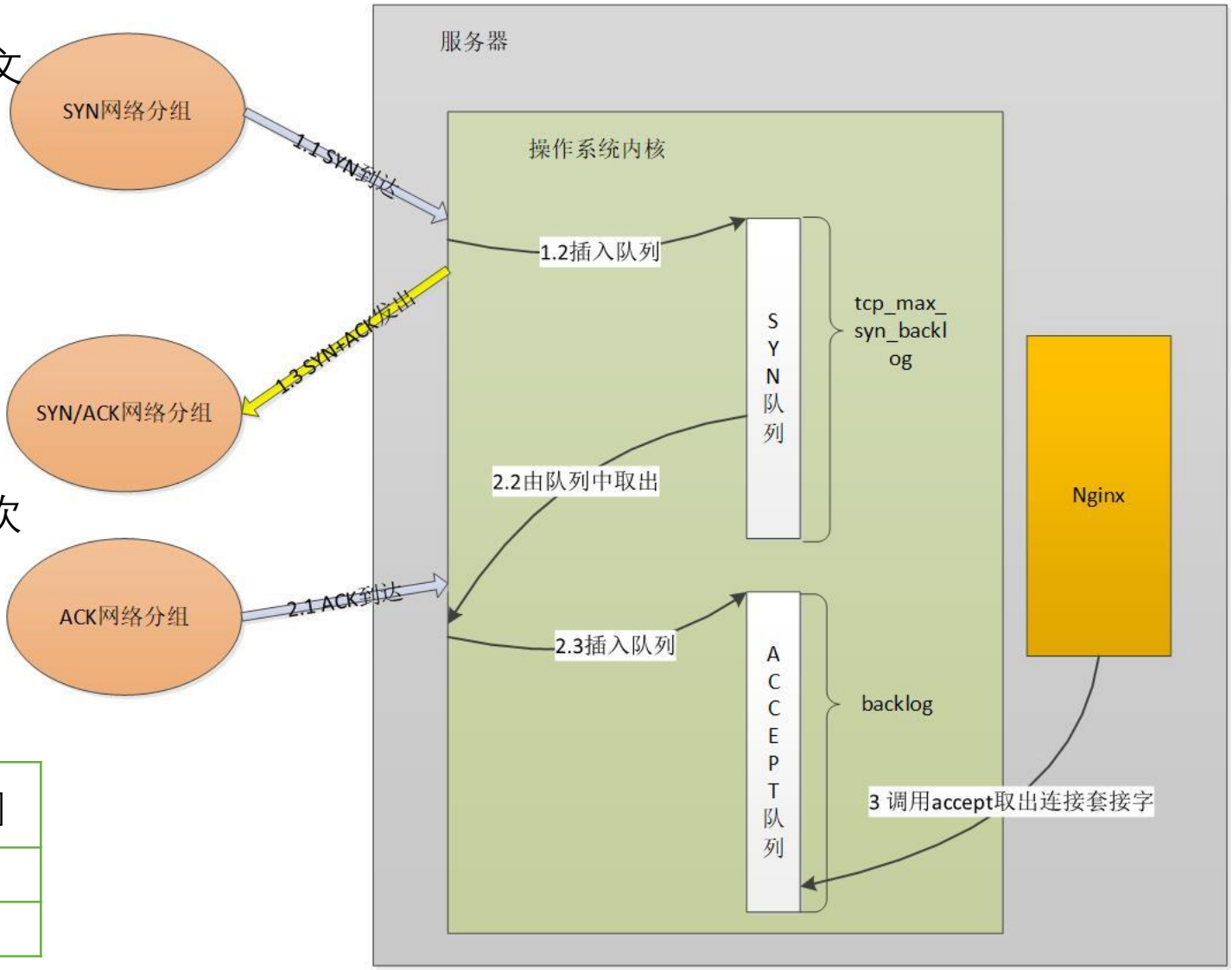
- SYN_SENT
  - net.ipv4.tcp_syn_retries = 6
    - 主动建立连接时，发SYN的重试次数
  - net.ipv4.ip_local_port_range = 1024　60999
    - 建立连接时的本地端口可用范围



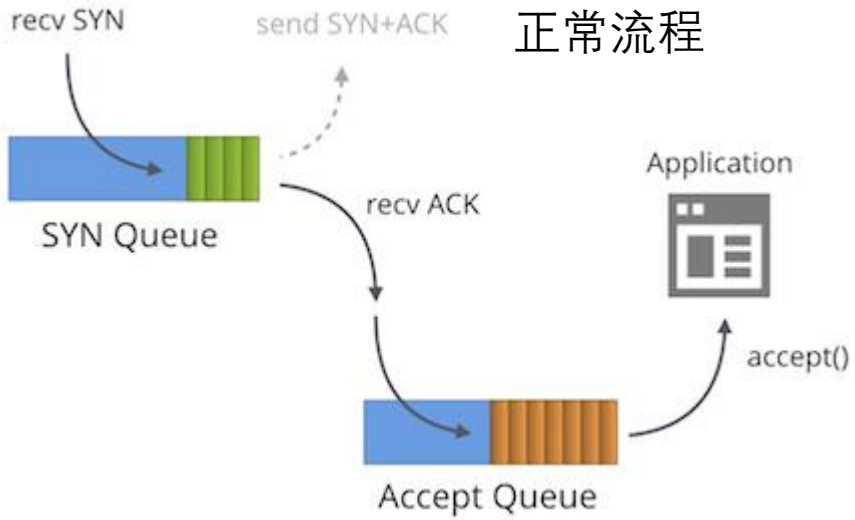
服务器端处理三次握手

- net.core.netdev_max_backlog
  - 接收自网卡、但未被内核协议栈处理的报文队列长度
- net.ipv4.tcp_abort_on_overflow
  - 超出处理能力时，对新来的SYN直接回包RST，丢弃连接
- SYN_RCVD状态
  - net.ipv4.tcp_max_syn_backlog
    - SYN_RCVD状态连接的最大个数
  - net.ipv4.tcp_synack_retries
    - 被动建立连接时，发SYN/ACK的重试次数
- ACCEPT队列已完成握手
  - net.core.somaxconn
    - 系统级最大backlog队列长度

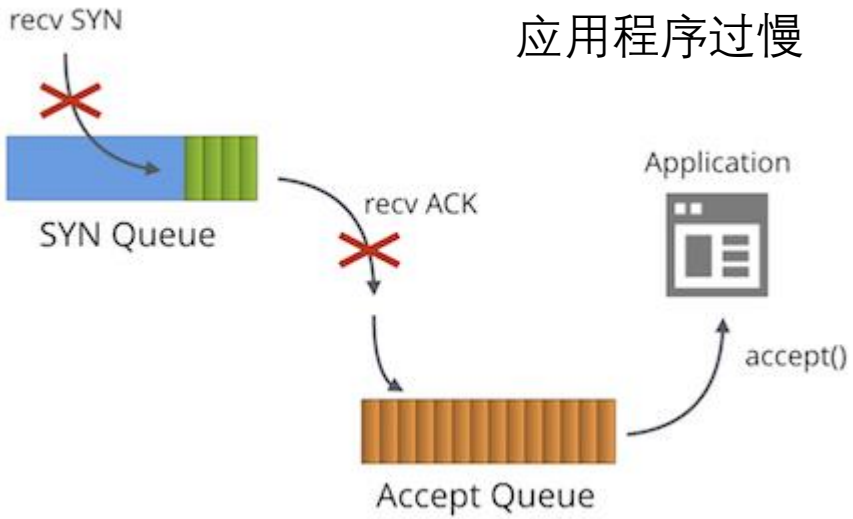
Syntax:	**listen** address[:port] [backlog=number]
Default:	listen *:80 | *:8000;
Context:	server



tcp_syncookies

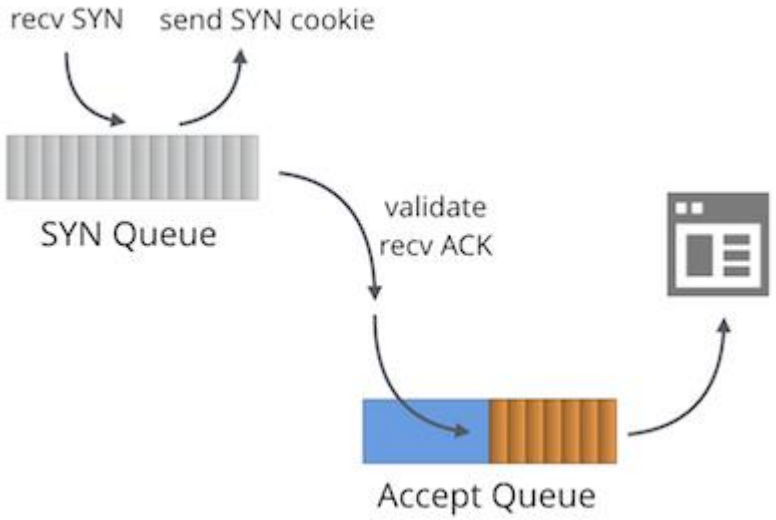


正常流程



应用程序过慢

SYN队列满，启用cookie



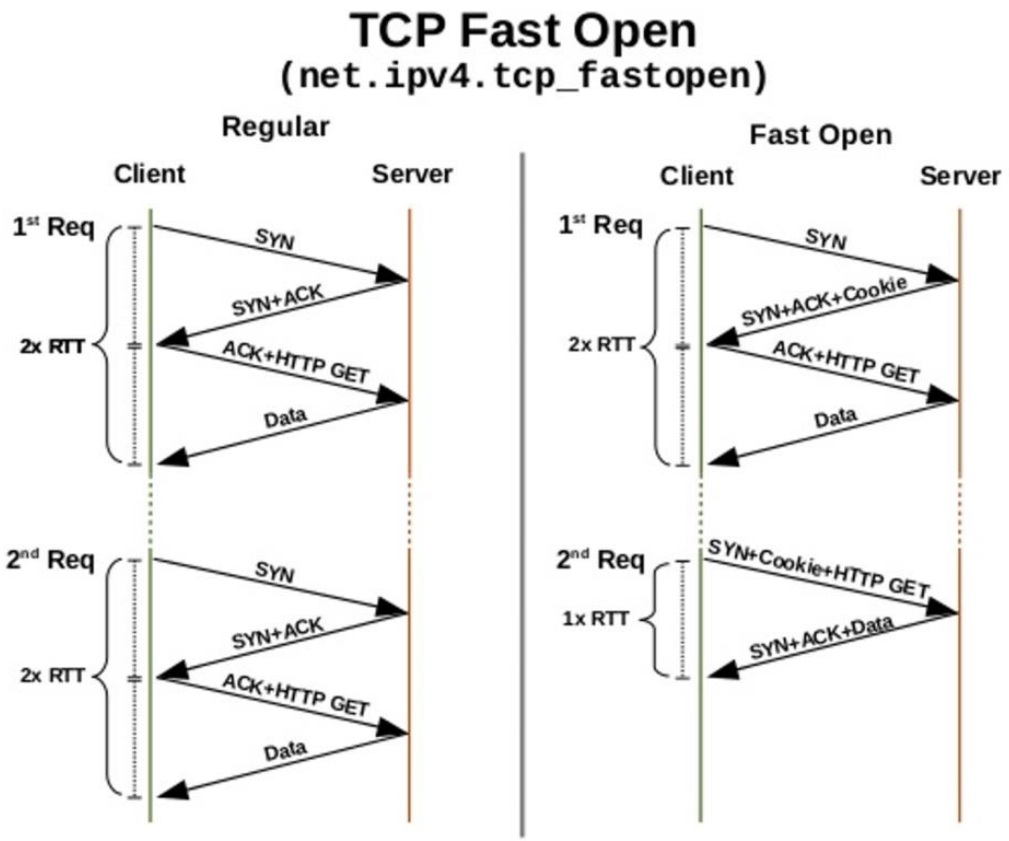
- net.ipv4.tcp_syncookies = 1
  - 当SYN队列满后，新的SYN不进入队列，计算出cookie再以SYN+ACK中的序列号返回客户端，正常客户端发报文时，服务器根据报文中携带的cookie重新恢复连接
    - 由于cookie占用序列号空间，导致此时所有TCP可选功能失效，例如扩充窗口、时间戳等

TCP Fast Open

- net.ipv4.tcp_fastopen：系统开启TFO功能
  - 0：关闭
  - 1：作为客户端时可以使用TFO
  - 2：作为服务器时可以使用TFO
  - 3：无论作为客户端还是服务器，都可以使用TFO

Syntax: **listen** *address*[:*port*] [fastopen=*number*];
Default: listen *:80 | *:8000;
Context: server

- fastopen=number
  - 为防止带数据的SYN攻击，限制最大长度，指定TFO连接队列的最大长度

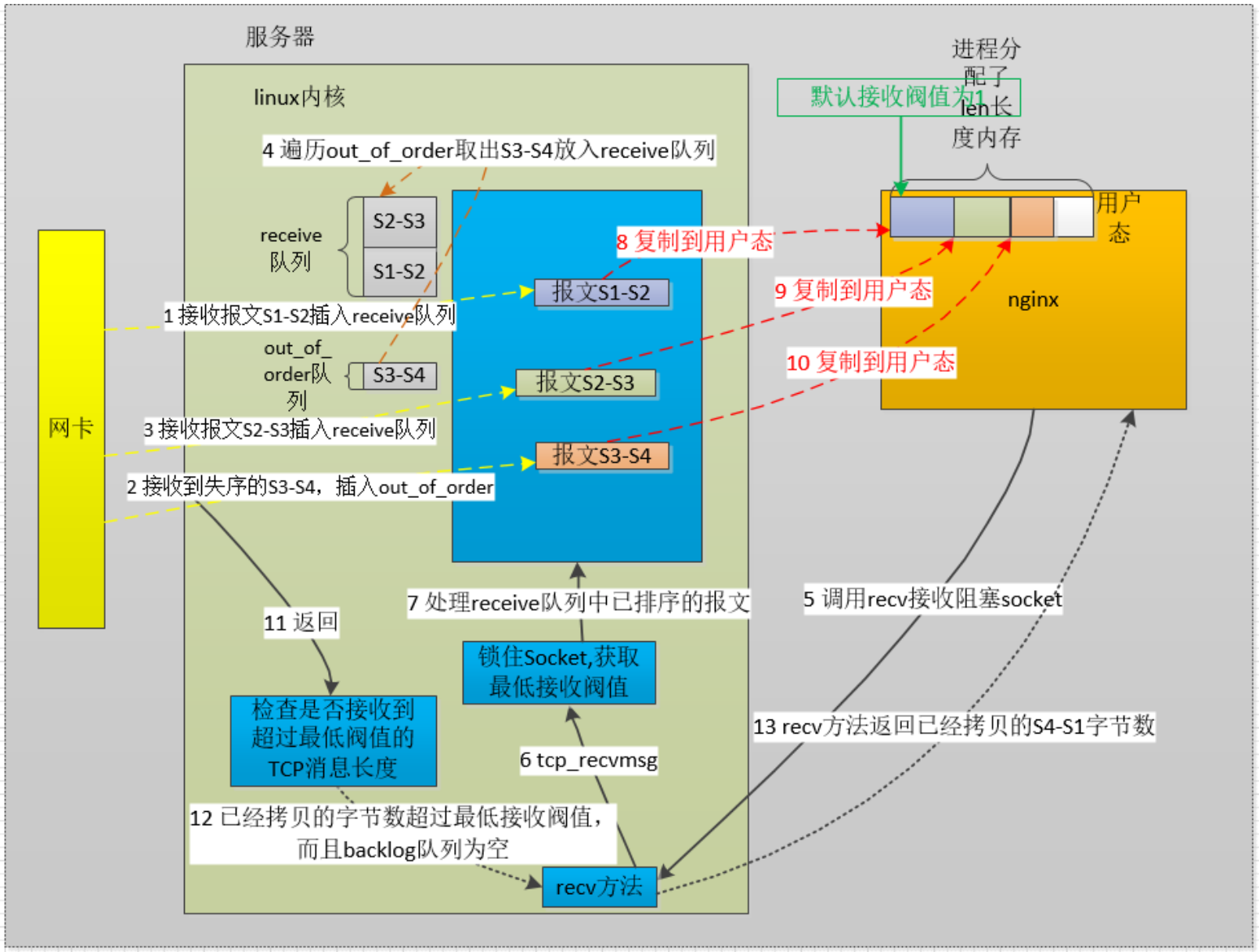


TCP消息的接收超时

Syntax:	**client_body_timeout** time;
Default:	client_body_timeout 60s;
Context:	http, server, location

Syntax:	**proxy_read_timeout** time;
Default:	proxy_read_timeout 60s;
Context:	http, server, location

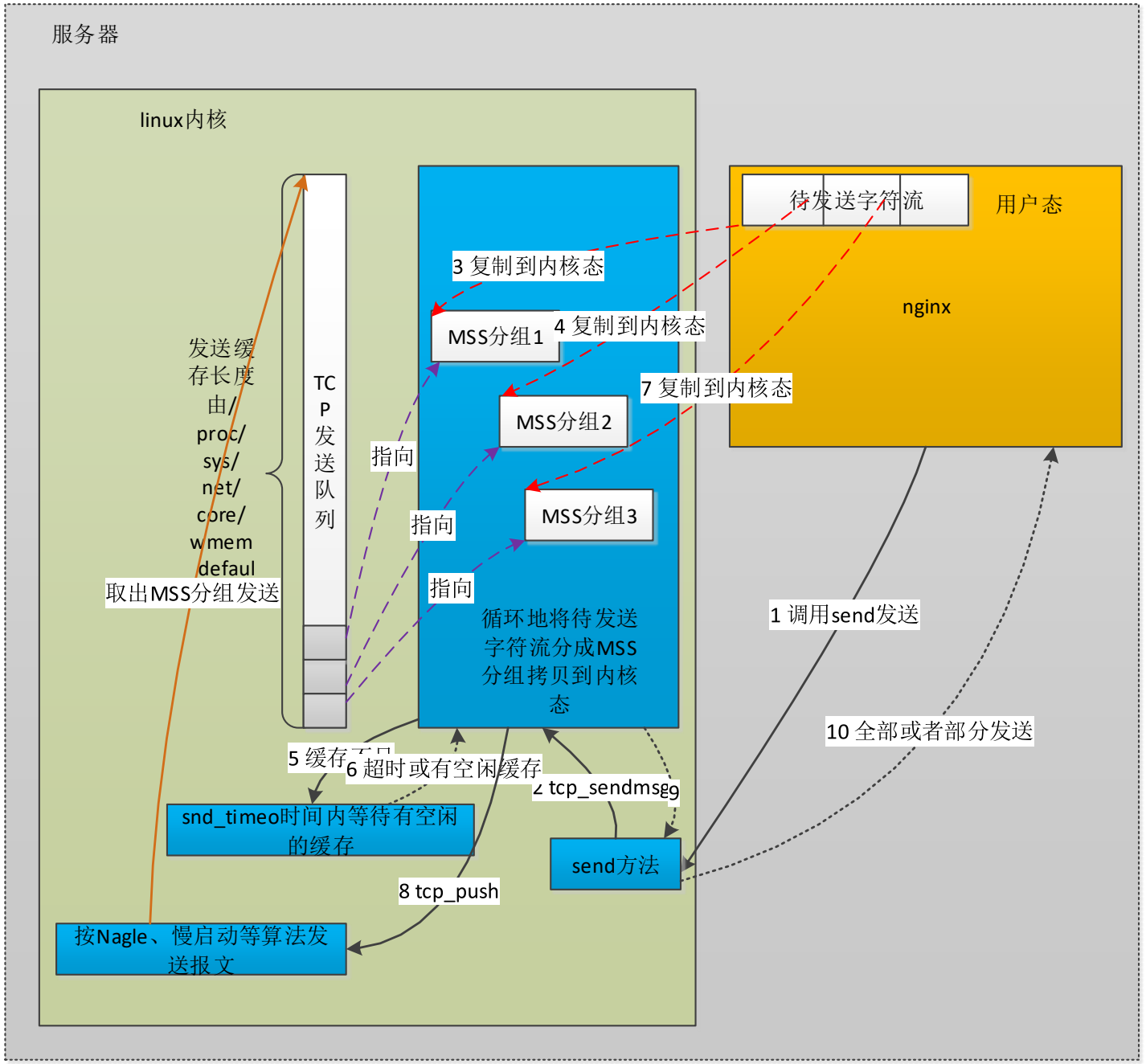
Syntax:	**proxy_timeout** timeout;
Default:	proxy_timeout 10m;
Context:	stream, server



TCP消息的发送超时

Syntax:	**send_timeout** time;
Default:	send_timeout 60s;
Context:	http, server, location

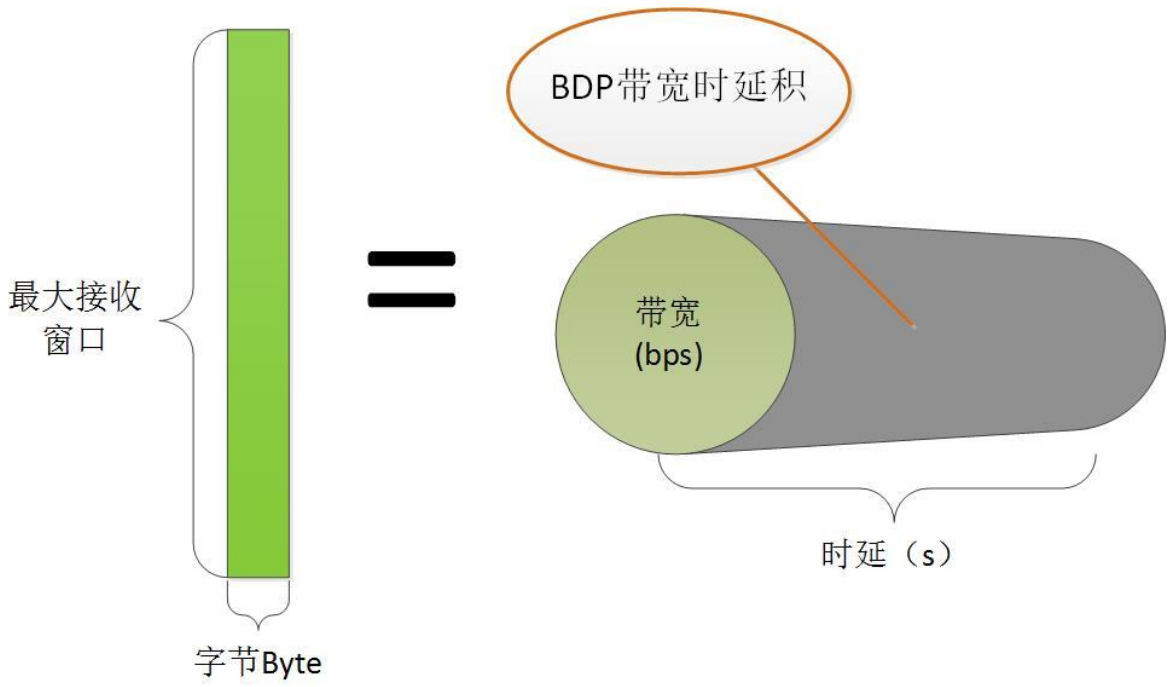
Syntax:	**proxy_send_timeout** time;
Default:	proxy_send_timeout 60s;
Context:	http, server, location



TCP缓冲区

- net.ipv4.tcp_rmem = 4096　　87380　6291456
  - 读缓存最小值、默认值、最大值，单位字节，覆盖net.core.rmem_max
- net.ipv4.tcp_wmem = 4096　　16384　4194304
  - 写缓存最小值、默认值、最大值，单位字节，覆盖net.core.wmem_max
- net.ipv4.tcp_mem = 1541646　　2055528 3083292
  - 系统无内存压力、启动压力模式阀值、最大值，单位为页的数量
- net.ipv4.tcp_moderate_rcvbuf = 1
  - 开启自动调整缓存模式

Syntax:	**listen** address[:port] [rcvbuf=size] [sndbuf=size];
Default:	listen *:80 | *:8000;
Context:	server

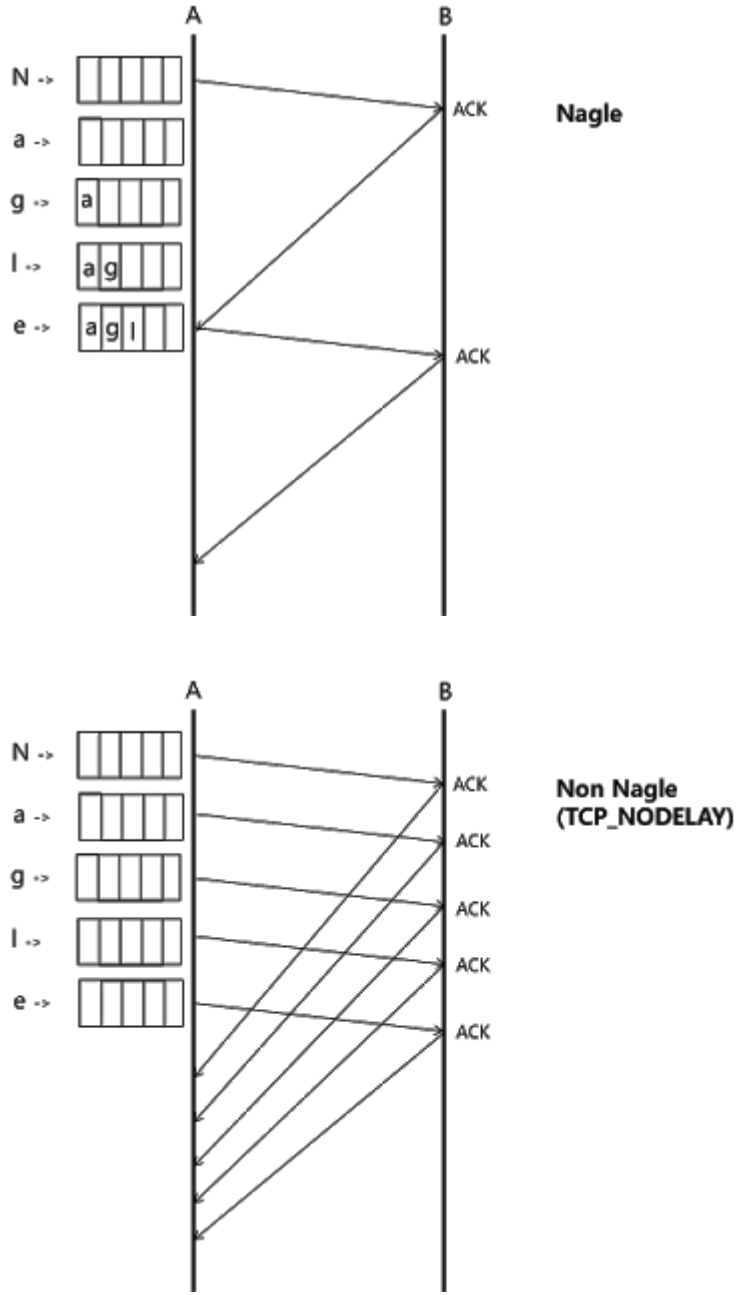


提升有效信息比：Nagle 与 Cork

- Nagle算法
  - 避免一个连接上同时存在大量小报文
    - 最多只存在一个小报文
    - 合并多个小报文一起发送
  - 提高带宽利用率，
- 吞吐量优先：启用Nagle算法，tcp_nodelay off
- 低时延优先：禁用Nagle算法，tcp_nodelay on

Syntax:	**tcp_nodelay** on | off;
Default:	tcp_nodelay on;
Context:	http, server, location

Syntax:	**tcp_nopush** on | off;
Default:	tcp_nopush off;
Context:	http, server, location



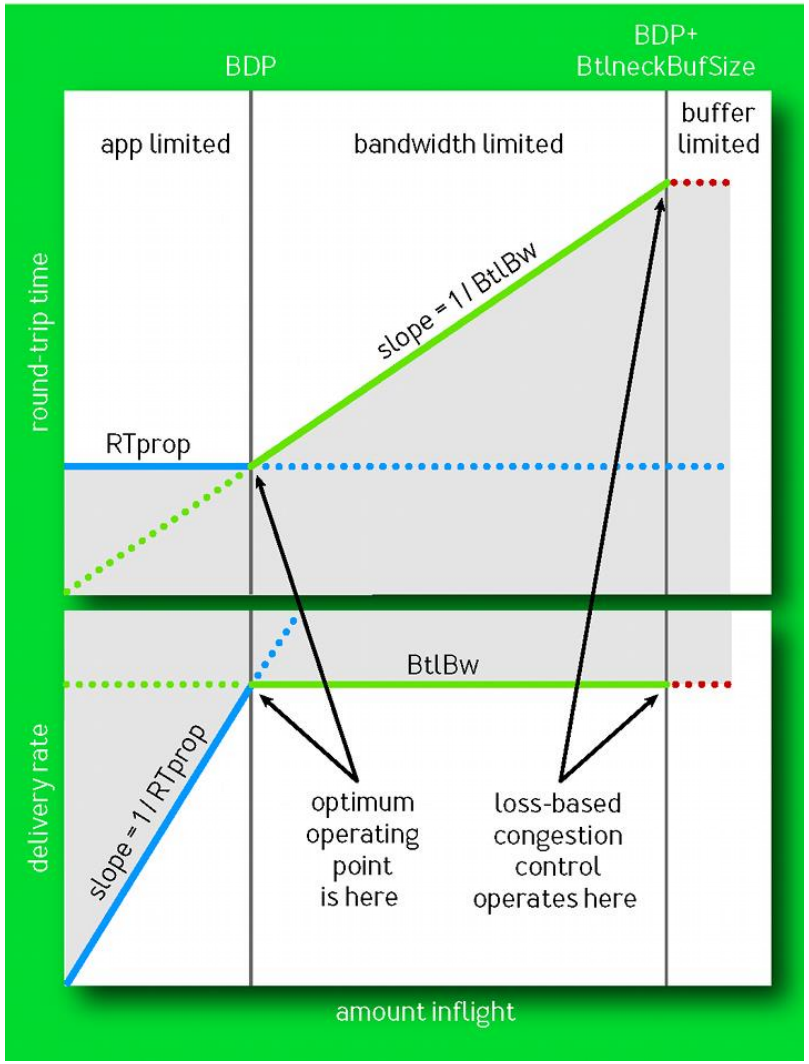
丢包驱动的拥塞控制

TCP Slow Start

测量驱动的拥塞控制
--1979 Leonard Kleinrock

- 基于丢包的拥塞控制算法
  - 高时延，大量丢包
  - 随着内存便宜，时延更高
  - net.ipv4.tcp_congestion_control=bbr
- 左边纵线（对整体网络有效）
  - 最大带宽下
  - 最小时延
  - 最低丢包率
- RTprop 与 BtlBw 独立变化
  - 同时只有一个可以被准确测量

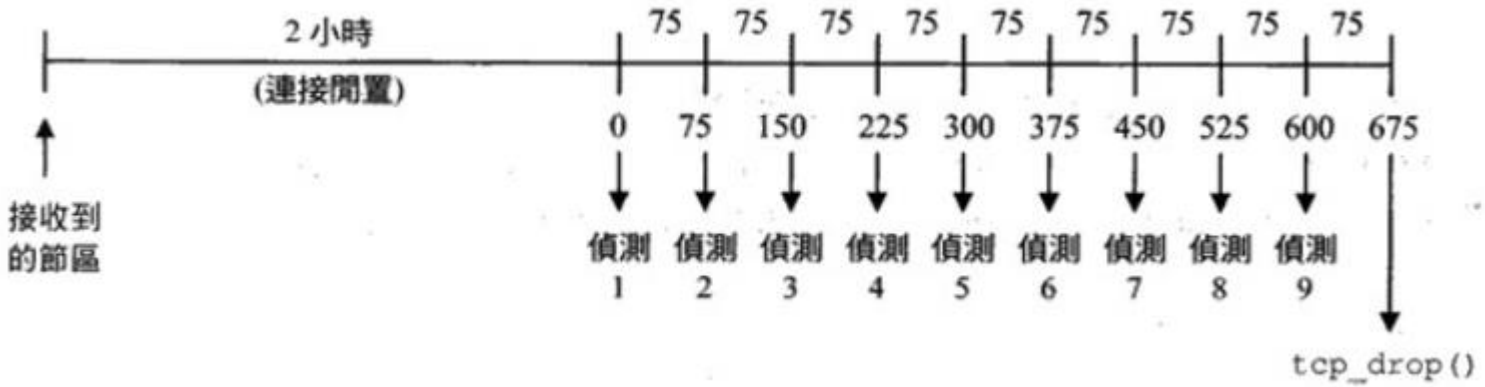
FIGURE 1: DELIVERY RATE AND ROUND-TRIP TIME VS. INFLIGHT



TCP的Keep-Alive功能

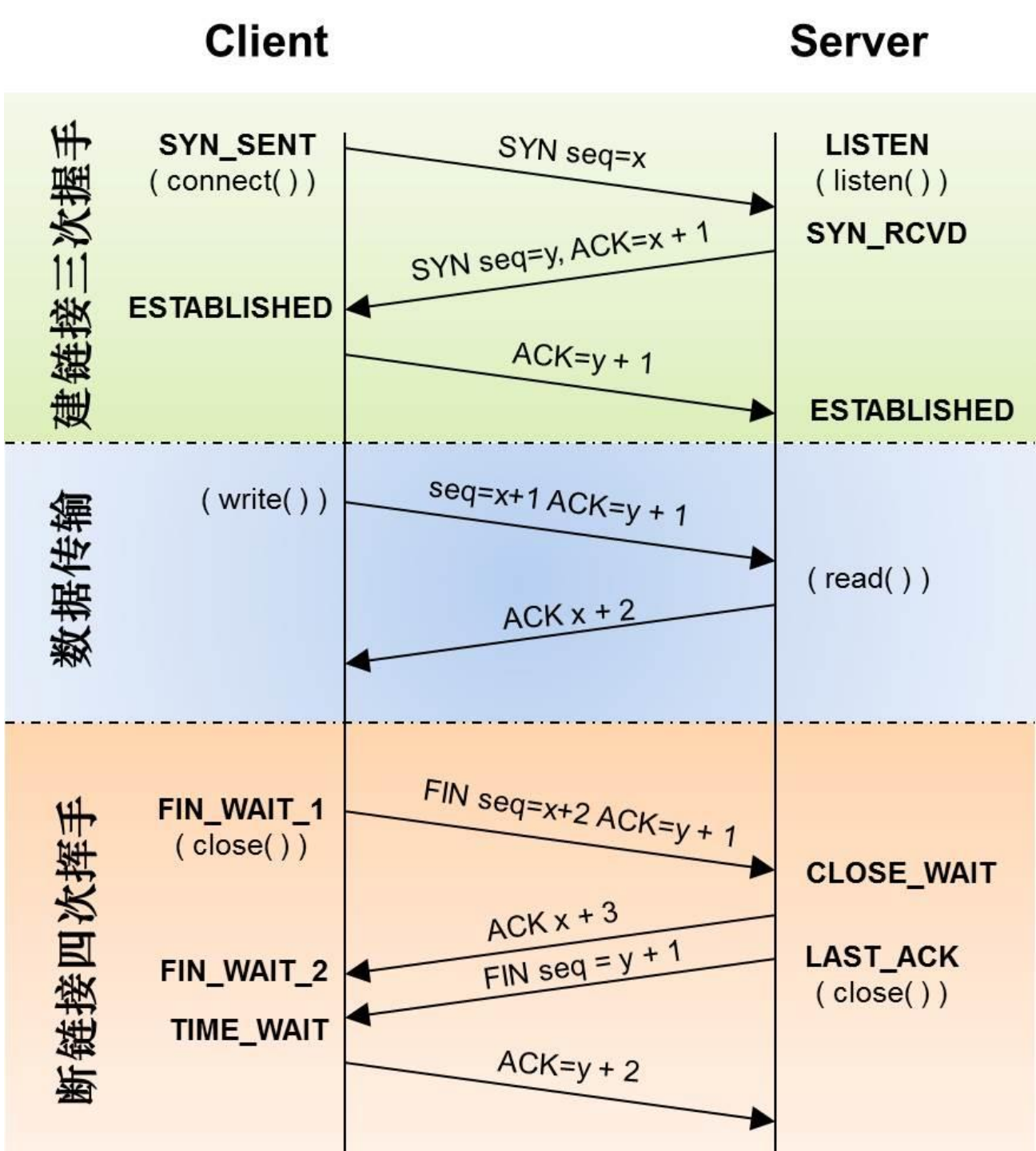
Nginx的Tcp keepalive
so_keepalive=30m::10
keepidle, keepintvl, keepcnt

Linux的tcp keepalive
发送心跳周期
Linux: net.ipv4.tcp_keepalive_time = 7200
探测包发送间隔
net.ipv4.tcp_keepalive_intvl = 75
探测包重试次数
net.ipv4.tcp_keepalive_probes = 9



TCP连接的关闭

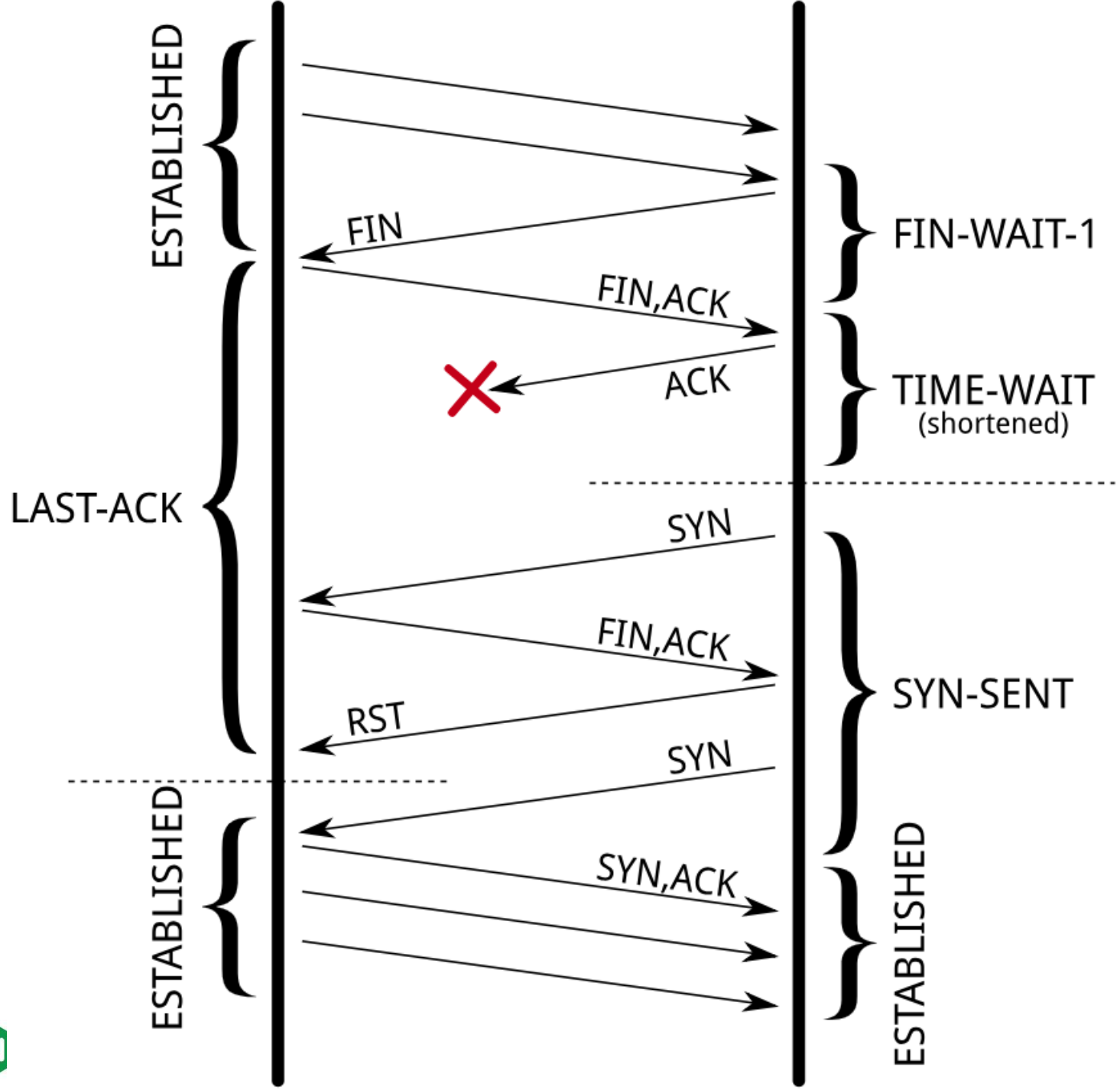
- 被动端
  - CLOSE_WAIT状态
    - 应用进程没有及时响应对端关闭连接
  - LAST_ACK状态
    - 等待接收主动关闭端操作系统发来的针对FIN的ACK报文
- 主动端
  - fin_wait1状态
    - net.ipv4.tcp_orphan_retries = 0
      - 发送FIN报文的重试次数，0相当于8
  - fin_wait2状态
    - net.ipv4.tcp_fin_timeout = 60
      - 保持在FIN_WAIT_2状态的时间
  - time_wait状态



TIME_WAIT优化：tcp_tw_reuse

- MSL(Maximum Segment Lifetime)
  - 报文最大生存时间
- 维持2MSL时长的TIME-WAIT状态
  - 保证至少一次报文的往返时间内端口是不可复用

- net.ipv4.tcp_tw_reuse = 1
  - net.ipv4.tcp_timestamps = 1
- net.ipv4.tcp_tw_recycle = 0
- net.ipv4.tcp_max_tw_buckets =



lingering关闭连接

Syntax:	**lingering_close** off | on | always;
Default:	lingering_close on;
Context:	http, server, location

- off：关闭功能
- on：由Nginx判断，当用户请求未接收完（根据chunk或者Content-Length头部等）时启用功能，否则及时关闭连接
- always：无条件启用功能

Syntax:	**lingering_time** time;
Default:	lingering_time 30s;
Context:	http, server, location

当功能启用时，最长的读取用户请求内容的时长，达到后立刻关闭连接。

Syntax:	**lingering_timeout** time;
Default:	lingering_timeout 5s;
Context:	http, server, location

当功能启用时，检测客户端是否仍然请求内容到达，若超时后仍没有数据到达，则立刻关闭连接

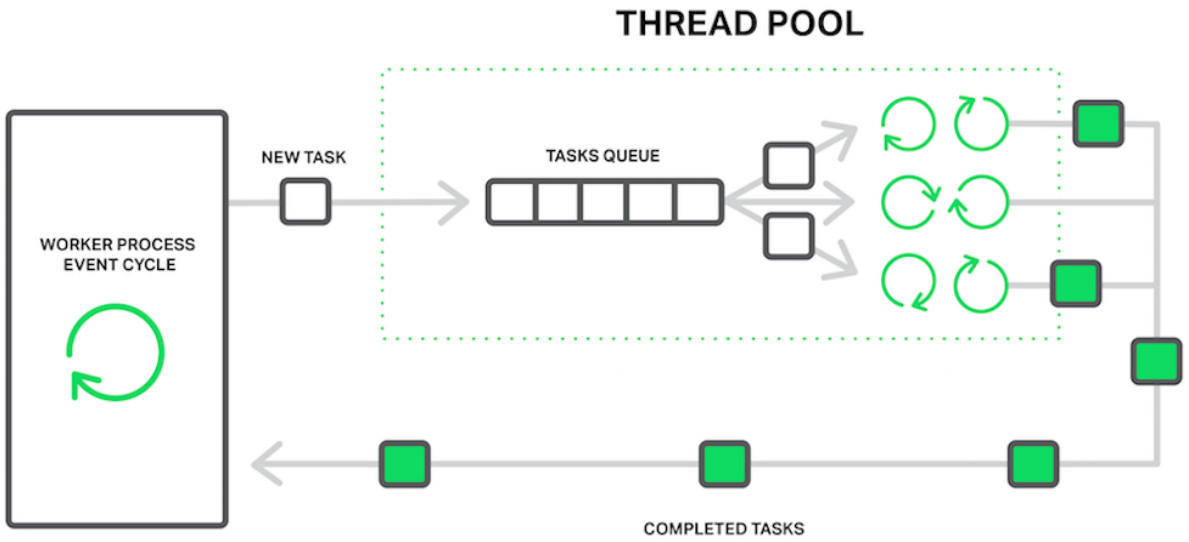
以RST代替正常的四次握手关闭连接

当其他读、写超时指令生效引发连接关闭时，通过发送RST立刻释放端口、内存等资源来关闭连接

Syntax:	**reset_timedout_connection** on | off;
Default:	reset_timedout_connection off;
Context:	http, server, location

减少磁盘IO

- 优化读取
  - Sendfile零拷贝
  - 内存盘、SSD盘
- 减少写入
  - AIO
  - 增大error_log级别
  - 关闭access_log
  - 压缩access_log
  - 是否启用proxy buffering?
  - syslog替代本地IO
- 线程池thread pool

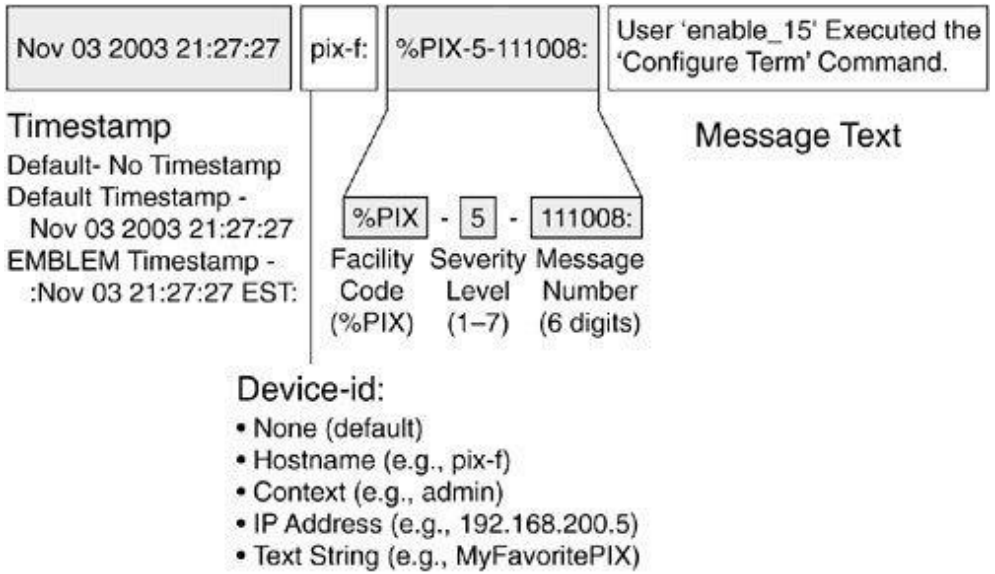


access日志写入优化

Syntax:	**access_log** path [format [**buffer=size] [gzip[=level]] [flush=time**] [if=condition]];
Default:	access_log logs/access.log combined;
Context:	http, server, location, if in location, limit_except

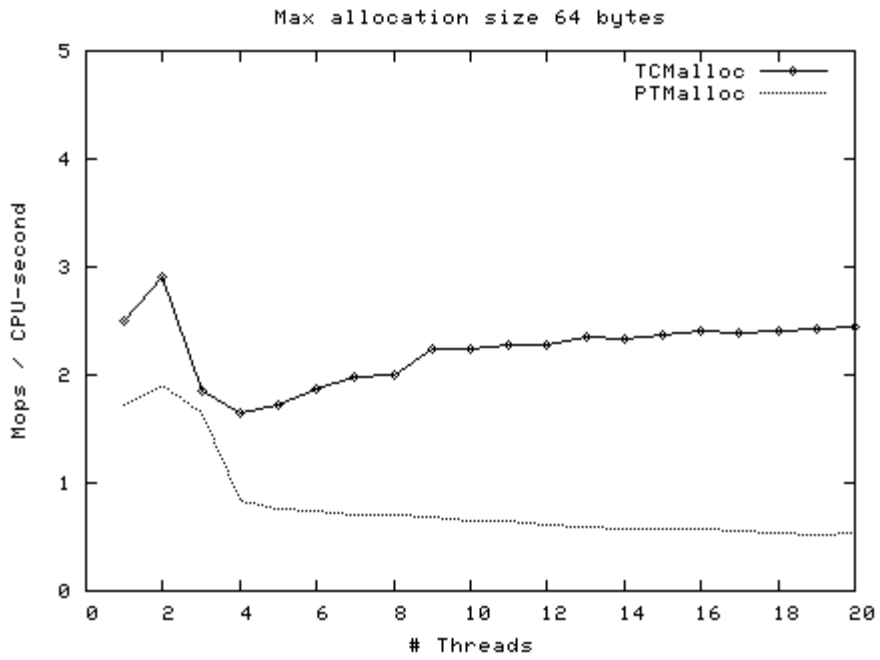
syslog网络协议：

日志格式：



tcmalloc

- 更快的内存分配器
  - 并发能力强于glibc
    - 并发线程数越多，性能越好
  - 减少内存碎片
  - 擅长管理小块内存
    - 256KB、1MB



使用gperftools定位nginx性能问题

- gperftools库：
  - https://github.com/gperftools/gperftools/releases
- 结果展示
  - 文本展示
    - pprof --text
  - 图形展示：
    - pprof --pdf
    - 依赖graphviz
- 依赖：
  - https://github.com/libunwind/libunwind/releases
- 模块：
  - ngx_google_perftoos_module，通过--with-google_perftools_module启用

Syntax: **google_perftools_profiles** *file*;

Default: —

Context: main

flame graph

- perf
  - perf record -F 99 -p 进程PID -g --call-graph dwarf
- **转换ASCII格式**
  - perf script > out.perf
- https://github.com/brendangregg/FlameGraph.git
  - 转换数据格式
    - FlameGraph/stackcollapse-perf.pl out.perf > out.folded
  - 生成矢量图
    - FlameGraph/flamegraph.pl out.folded > out.svg

stub_status模块的监控项

Active connections: 1
server accepts handled requests
 2511 2511 4764
Reading: 0 Writing: 1 Waiting: 0

- Active connections
  - 当前客户端与Nginx间的TCP连接数，等于下面Reading、Writing、Waiting数量之和
- accepts
  - 自Nginx启动起，与客户端建立过的连接总数
- handled
  - 自Nginx启动起，处理过的客户端连接总数。如果没有超出worker_connections配置，该值与accepts相同
- requests
  - 自Nginx启动起，处理过的客户端请求总数。由于存在HTTP Keep-Alive请求，故requests值会大于handled值
- Reading
  - 正在读取HTTP请求头部的连接总数
- Writing
  - 正在向客户端发送响应的连接总数
- Waiting
  - 当前空闲的HTTP Keep-Alive连接总数

谢谢