算法

bm算法　　因为换行符所以用坏字符
多模式匹配

流式检测
1.缓冲池作为 内存操作
2.六个状态：
　"<"
　"<a"
　"href"
　"http://"
　">"
　NULL（输入为空，也会匹配）
3.如果没有分包，直接使用全局内存；分包则分配内存池。

AC自动机：http://www.docin.com/p-46845432.html

AC自动机的两个使用：
1.抽取href字段：用流式
2.复杂的检测：用零拷贝

strcrc:
1.小写
2.拼接

优化：
项目时间愿意，也不可能一开始就极限优化。
内存，拷贝，逻辑，算法。
抽样检测为依据：oprofile 定时看EIP，然后根据函数地址范围来判断。如果一个函数的次数很多，有两种情况：1.次数很多 2.运行很久

0.分包 缓冲池（控制内存占用和分配时间）；非分包全局变量。
1.非 分包 零拷贝
2.全局 变量 代替 malloc
3.改变 逻辑 顺序：1.先判断第一个字符　2.
4.算法优化：多维搜索，BM算法，AC自动机。

AF5.9 因为对html文件各种解析，导致性能下降，在针对WAF的SQL误判日志模块优化过程中，这里总结一下修改的一般思路：

A) 代码习惯层次
　　1) 举个例子，strcpy_n的size参数一般人都会传进去buf的最大值，但是如果要拷贝的数据都是很短，然后buf的最大值很大，每次就会拷贝很多无谓的东西。
　　2) 比如对字符串指定长度的查找，有些人可能会自己写一个遍历的查找操作，但是其实可以用memchr，效率更高，而且不引入没必要的多于代码（二分查找也经常这样）。
　　3) 比如需要用到时间，有些人会在ad_appd里面使用time这个系统调用，但是其实time是很耗性能的，有其他可替代的函数timer_get_ticks。

B) 算法层次
　a) 查找
　这个也简单，基本按图索骥，比如多个字符串查找就上strmstr（只是一个AC自动机），固定单个字符串的查找上BMHChr（一个只有坏字符的简化版BM算法），大片数据包的简单扣字符串可以学习linkextract.c的流式处理（运用多模式匹配的状态机）。

b) 内存操作

AF5.9性能达标就靠它了。基本就包括两个方面：

Ⅰ) 避免申请内存：

尽量使用全局变量作为buf；

如果是需要记录分包的信息，尽量使用内存池，既能控制内存使用量，又能减少申请内存的开销。

Ⅱ) 避免拷贝内存：

误判模块需要扣取标签里面的内容，因为可能分包，所以就每次都拷贝。后来做了一个优化，性能一下子就神奇的达标了。就是如果在这个数据包就已经能得到标签内完整内存，就直接返回packet的指针，实现了零拷贝；当然，如果遇到分包，还是要继续拷贝。

c) 改算法

举个求crc的例子，比如你需要求两个字符串拼接之后的crc，其实看了strcrc32的实现，发现里面也是一个循环求值的过程，所以可以修改一下算法，就可以不需要拷贝两个字符串，再去求crc。

C) 业务逻辑层次

a) 少走可能无谓的流程

比如有个逻辑如下：

```
ret_1 = fuck_1();
ret_2 = fuck_2();
if( ret_1 && ret_2) return 0;
return -1;
```

很明显，应该在ret_1失败就直接返回。

b) 调整判断流程

比如你有一个逻辑如下：

```
if (strcmp(a, "bsdfsdgdffsfsdfsdfsd" == 0) { ... }
else if(a[0] == 'a') { ... }
else { return -1 }
```

在两种情况可能都相同的情况下，很明显，先判断a[0]是否等于'a'，肯定会更好。