第三届 eBPF开发者大会

www.ebpftravel.com

基于eBPF的Nginx可观测性实践

——从请求追踪到性能瓶颈定位

分享人：格尔软件安全网关团队 - 贺红杰

中国·西安

问题背景与挑战

Nginx作为广泛使用的高性能Web服务器，随着应用复杂性的不断增加，为了保障系统的稳定可靠运行，实现对其内部状态的实时监控和对故障的快速定位越发重要。

运维痛点

- 监控指标不足
  - 基础指标缺失：原生指标覆盖不足，存在大量指标缺失（如SSL握手耗时、SSL Session尺寸）
  - 隐形指标缺失：一些性能相关问题需要观测函数级别的处理耗时，原生并不支持
  - 指标关联断层：内核态处理（如TCP建立）与Nginx用户态处理（Accept）难以联动分析
- 排障效率低下
  - 请求链路黑盒化： 难以追踪单个请求在Nginx内部的完整生命周期（特别是从内核TCP建连到Nginx请求响应处理）
  - 偶发故障难复现： 只在特定环境下偶发的问题排查定位十分困难（如ssl session偶发性创建失败）

传统方案局限性

- 应对监控指标不足
  - 定制扩展指标：常规性指标缺失，可以通过扩展源码或使用第三方模块来实现支持
  - 隐形指标缺失：临时修改代码来添加指标，费时费力
  - 指标关联断层： 
- 排障效率低下
  - 请求链路黑盒化：依赖于运行日志观察请求链路，由于数据量巨大，从中追踪单个请求让人痛不欲生 
  - 偶发故障难复现： 

eBPF方案优势

零侵入：动态Hook Nginx内部函数，无需修改源码，能够很好地解决监控指标缺失问题
广覆盖：通过同时追踪内核态 + 用户态连接请求处理全过程，能够应对指标关联断层问题
低开销：观测过程不影响正常业务，对偶发性故障可以做到7x24小时观测
高安全：观测操作不会导致内核崩溃进而引发业务故障，便于在生产环境使用

eBPF方案挑战

如何解析Nginx自定义结构?

Nginx内部自定义了许多结构（如ngx_http_request_t），要怎样方便地观测到特定结构的特定字段（如r->uri）？

如何观测Nginx全局变量?

Nginx全局变量并不总是作为函数入参与返回值，要怎样观测全局变量（如ngx_accept_mutex_held）？

如何观测Nginx局部变量?

Nginx局部变量存储在栈/寄存器中，且生命周期仅限于函数上下文，要怎样观测局部变量（如ssl_session）？

……

解析Nginx自定义结构

往常方法是通过mock方式简化结构定义：结构成员间深层嵌套时特别麻烦

```
struct ngx_http_request_s {
    uint32_t          signature;
    void              *connection;
    ngx_str_t         uri;
    ...
};
```

内核追踪可以直接引用相应头文件，用户态对Nginx的追踪能不能做到类似效果？🤔

```
/lib/modules/6.12.13-amd64/source/include/linux/types.h:20:26: error: typedef redefinition with different types
('__kernel_fd_set' vs 'struct fd_set')
/usr/include/x86_64-linux-gnu/sys/select.h:70:5: note: previous definition is here
/lib/modules/6.12.13-amd64/source/include/linux/types.h:21:25: error: typedef redefinition with different types
('__kernel_dev_t' (aka 'unsigned int') vs '__dev_t' (aka 'uns
igned long'))
...
```

解析Nginx自定义结构

要做到能够引用Nginx头文件，必须首先解决与内核头文件存在的冲突，对Nginx原始头文件进行一些调整，比如

- 需要引入limits.h，以完成对各类数据类型的范围限制定义，避免查到内核态定义造成冲突
- 移除一些对追踪而言不必要的头文件引用，如：#include <netinet/in.h>

...

```
#!/usr/bin/env bpftrace

#include <ngx_config.h>
#include <ngx_core.h>
#include <ngx_http.h>

//追踪探测函数入口实例演示：追踪解析请求
uprobe:/opt/nginx/sbin/nginx:ngx_http_process_request
{
    $r = (struct ngx_http_request_s *) arg0;

     printf("%s(%d): %s\n", comm, pid, str($r->uri.data, $r->uri.len+1));
}

# sudo bpftrace -I nginx_headers ngx_http_process_request.bt
nginx(2745892): /demo
```

观测Nginx全局变量

由于全局变量不总作为函数入参与返回值存在，并且其地址会随ASLR（地址空间随机化）发生变化，对全局变量的追踪需要采用base+offset的方式

```
#!/usr/bin/env bpftrace

#include <ngx_config.h>
#include <ngx_core.h>
#include <ngx_event.h>
#include <ngx_event_openssl.h>

//追踪探测全局变量实例演示：探测ngx_accept_mutex锁在nginx worker间的持有情况
uprobe:/opt/nginx/sbin/nginx:ngx_trylock_accept_mutex
{
    //head -1 /proc/$(pidof nginx)/maps | awk -F'-' '{print $1}'
    $base = (uint64)0x562bc6c45000;
    $ngx_accept_mutex_held_offset = (uint32)uaddr("ngx_accept_mutex_held");
    $ngx_accept_mutex_held = *(uint64*)($base + $ngx_accept_mutex_held_offset);

     printf("%s(%d): %d\n",  comm, pid, $ngx_accept_mutex_held);
}

# sudo bpftrace -I nginx_headers ngx_accept_mutex.bt
nginx(2745893): 0
nginx(2745892): 1
```

```
ngx_int_t
ngx_trylock_accept_mutex(ngx_cycle_t *cycle)
{
  if (ngx_shmtx_trylock(&ngx_accept_mutex)) {

    if (ngx_accept_mutex_held && ngx_accept_events == 0) {
      return NGX_OK;
    }

    if (ngx_enable_accept_events(cycle) == NGX_ERROR) {
      ngx_shmtx_unlock(&ngx_accept_mutex);
      return NGX_ERROR;
    }

    ngx_accept_events = 0;
    ngx_accept_mutex_held = 1;

    return NGX_OK;
  }

  if (ngx_accept_mutex_held) {
    if (ngx_disable_accept_events(cycle, 0) == NGX_ERROR) {
      return NGX_ERROR;
    }

    ngx_accept_mutex_held = 0;
  }

  return NGX_OK;
}
```

观测Nginx局部变量

由于局部变量不作为函数入参与返回值存在，且生命周期仅限于函数上下文，但其值存储在栈或寄存器中，因此可以通过栈与寄存器来实现对局部变量的追踪

```
#!/usr/bin/env bpftrace

#include <ngx_config.h>
#include <ngx_core.h>
#include <ngx_event.h>
#include <ngx_event_openssl.h>

//追踪探测局部变量实例演示: 统计ssl session尺寸
uprobe:/opt/nginx/sbin/nginx:ngx_ssl_new_session+712
{
    $sess_id = *(struct ngx_ssl_sess_id_s **)(reg("bp")-32);

    $id_len = $sess_id->len;
    $session_len = (uint8)$sess_id->node.data;
    $meta_len = sizeof(struct ngx_ssl_sess_id_s);

    $total_len = $id_len + $session_len + $meta_len;
    printf("session size: %d = %d + %d + %d \n", $total_len, $id_len, $session_len, $meta_len);}
}
```

```
static int
ngx_ssl_new_session(ngx_ssl_conn_t *ssl_conn, ngx_ssl_session_t *sess)
{
  int               len;
  ngx_ssl_sess_id_t     *sess_id;

  ...

  ngx_memcpy(cached_sess, buf, len);
  ngx_memcpy(id, session_id, session_id_length);

  hash = ngx_crc32_short(session_id, session_id_length);
  ...
  sess_id->node.key = hash;
  sess_id->node.data = (u_char) session_id_length;
  sess_id->id = id;
  sess_id->len = len;
  sess_id->session = cached_sess;
  sess_id->expire = ngx_time() + SSL_CTX_get_timeout(ssl_ctx);

  ...
  ngx_rbtree_insert(&cache->session_rbtree, &sess_id->node);
  ...
  return 0;
```

```
# sudo bpftrace -I nginx_headers ngx_ssl_sessoin_size.bt
session size: 318 = 158 + 32 + 128
```

试用交流

试用地址：https://github.com/honghaier250/Nginx-With-eBPF