MySQL数据库审计采集技术调研之Packetbeat，eBPF

2022-01-10 3594 版权

简介： 在数据安全的重要性日益突出的今天，数据库审计是数据安全的重要组成部分。在数据库审计的过程中往往需要关注数据库操作行为和性能数据：需要知道数据库来源IP，数据库服务器IP/端口，数据库登录用户，操作数据库名称，表名称，SQL语句，执行时间和返回数据条数等。这些数据的获取在数据库内核实现无疑是最佳的方式，但是并非所有的数据库都支持审计插件。几乎所有的数据库都是C/S模式，客户端与数据库通过网络协议沟通，数据库的协议大部分也是公开的，所以一种更通用的数据库审计数据采集的方案，应该是通过“外围”的抓包方案，具体是通过分析网络协议来采集数据库的行为数据。

数据库审计抓包方案

在数据安全的重要性日益突出的今天，数据库审计是数据安全的重要组成部分。在数据库审计的过程中往往需要关注数据库操作行为和性能数据：需要知道数据库来源IP，数据库服务器IP/端口，数据库登录用户，操作数据库名称，表名称，SQL语句，执行时间和返回数据条数等。这些数据的获取在数据库内核实现无疑是最佳的方式，但是并非所有的数据库都支持审计插件。几乎所有的数据库都是C/S模式，客户端与数据库通过网络协议沟通，数据库的协议大部分也是公开的，所以一种更通用的数据库审计数据采集的方案，应该是通过“外围”的抓包方案，具体是通过分析网络协议来采集数据库的行为数据。
MySQL通信协议已经包含了客户端与服务端的所有的交互细节，在MySQL的各种driver实现中，都需要对MySQL协议进行解析和封装，客户端的各种操作都需要通过标准协议进行传输，所以通过对MySQL协议的解析出各类操作，然后将这些操作封装成特定的结构，然后传入日志数据等分析平台，可以实现数据库的审计。因此，对MySQL通信协议进行抓包，解析是数据库安全审计重要前提。

接下来就在MySQL数据库审计角度介绍几种常见的抓包技术。首先会简单介绍下MySQL通信协议，然后介绍一些数据库协议抓包相关的技术如Packetbeat及在云原生流行的eBPF技术在数据库审计中的应用。

MySQL通信协议

通过数据库协议来分析数据库的行为，需要了解MySQL客户端与服务端的通信协议，MySQL协议主要包括两个方向：Client到Server端和Server到Client端，通过官方文档和资料可以了解MySQL协议的大致结构。客户端与服务端的交互主要包括认证阶段和命令执行阶段。

客户端->服务端

客户端向服务端发送常见命令，如常见的增删改查等命令，预处理语句等都是通过命令协议，命令协议主要是包含命令类型和对应的参数。格式如下：

Type	Name	Description
int<1>	执行命令	执行的操作，比如切换数据库，查询表等操作
string	参数	命令相应的参数

云存储
技术博文 +关注
热门文章 最新文章
【推荐几款实用的网盘资源搜... 125
SMB协议基础篇 83
让 Agent 拥有长期记忆：基于 ... 74
如何使用基站查询API帮你解析... 61
【qemu虚拟化】将img镜像文... 60
【提取翻译竖排文字日文图片... 44
VIN车辆识别码查询车五项 API ... 44

其中执行命令列表如下，通过解析不同的命令，可以解析出用户执行的查询语句，查询的结果内容等，这些命令对于数据库审计都有重要的作用。

类型值	命令	功能
0x00	COM_SLEEP	（内部线程状态）
0x01	COM_QUIT	关闭连接
0x02	COM_INIT_DB	切换数据库
0x03	COM_QUERY	SQL查询请求
0x04	COM_FIELD_LIST	获取数据表字段信息
0x05	COM_CREATE_DB	创建数据库
0x06	COM_DROP_DB	删除数据库
0x07	COM_REFRESH	清除缓存
0x08	COM_SHUTDOWN	停止服务器
0x09	COM_STATISTICS	获取服务器统计信息
0x0A	COM_PROCESS_INFO	获取当前连接的列表
0x0B	COM_CONNECT	（内部线程状态）
0x0C	COM_PROCESS_KILL	中断某个连接
0x0D	COM_DEBUG	保存服务器调试信息
0x0E	COM_PING	测试连通性
0x0F	COM_TIME	（内部线程状态）
0x10	COM_DELAYED_INSERT	（内部线程状态）
0x11	COM_CHANGE_USER	重新登陆（不断连接）
0x12	COM_BINLOG_DUMP	获取二进制日志信息
0x13	COM_TABLE_DUMP	获取数据表结构信息
0x14	COM_CONNECT_OUT	（内部线程状态）
0x15	COM_REGISTER_SLAVE	从服务器向主服务器进行注册
0x16	COM_STMT_PREPARE	预处理SQL语句
0x17	COM_STMT_EXECUTE	执行预处理语句
0x18	COM_STMT_SEND_LONG_DATA	发送BLOB类型的数据
0x19	COM_STMT_CLOSE	销毁预处理语句
0x1A	COM_STMT_RESET	清除预处理语句参数缓存
0x1B	COM_SET_OPTION	设置语句选项
0x1C	COM_STMT_FETCH	获取预处理语句的执行结果

服务端->客户端

服务端向客户端发送内容主要包括用户查询返回的数据，用户增删改产生的数据更新条数信息等，服务端向客户端发送的数据包有两个限制：

- 每个数据包大小不能超过2^24字节(16MB);
- 个数据包前都需要加上数据包头信息；

的基本格式：

Type	Name	Description
int<3>	payload_length	具体数据包的内容长度，排除头部4个字节
int<1>	sequence_id	包的序列id，数据长度大于16MB时需要用，从0开始，依次增加，新的命令执行会重载为0
string	payload	除去头部后的具体数据内容

服务端返回的payload主要分为OK_Packet，ERR_Packet，EOF_Packet，Result Set Packet，每种Packet可以根据返回的Payload中Header来区分，其中常见的Update语句返回的影响行数，Ping数据库的返回结果都是OK_Packet；执行Select，Show等命令时服务端会返回Result Set Packet，其中会记录返回列数，返回的数据的值。

网络抓包的特点

从应用最早的单体架构到现在的云原生部署架构，数据库的部署方式也在发生着变化，从单机部署到读写分离，在到各种云支持的云数据库RDS，应用在使用数据库时的便利性在不断提升，数据库管理的职责也在从业务方逐渐转移到云厂商，业务方可以将更多的精力关注在业务创新上。
随着数据库部署方式的演变，使用网络协议抓包的方式对数据库行为进行收集成为一种数据库审计技术中的采集的主流。通过协议抓包具有如下特点。

部署方式灵活

MySQL通信协议是在客户端和服务端之间是对称出现的，即同一个网络包的内容在客户端和服务端的网卡都会出现一次，这样就使得通过网络抓包来进行数据库审计可以在部署模式上会更加灵活，一般有以下几种方式：

- 流量端口镜像：通过在数据库流量经过的交换机进行端口镜像，通过采集镜像的流量收集MySQL的操作信息，这种方式对数据库及业务方影响最小，但需要有额外的部署。
- 代理部署：在客户端与服务端之间的MySQL代理中部署审计程序，审计程序可以看到所有经过代理的流量，从而进行审计数据的采集，这种方式对代理性能要求比较高。
- 服务端部署：在MySQL所在的服务器进行网络流量抓包，对于数据库服务会有一些性能的影响，取决于抓包程序使用的技术。
- 业务端（客户端）部署：在业务服务器上进行网络流量抓包，缺点是需要在每个业务机器上部署，在不能直接部署在服务器端或者端口镜像时，如RDS等，可以在业务端部署审计抓包程序。

对数据库性能影响小

此外对于数据库的影响会更少，由于抓包程序的运行与数据库服务进程是隔离的，往往对数据库的性能是没有影响或者影响非常小。
接下来介绍几种比较常见的针对MySQL协议的抓包技术，以及其中涉及的原理。

Packetbeat方案

Packetbeat是Beats系列的一个重要补充，主要用来采集服务器间的网络流量，支持多种常见的网络协议的解析，主要支持的网络协议如下：

- ICMP (v4 and v6)
- DHCP (v4)
- DNS
- HTTP
- AMQP 0.9.1
- Cassandra
- Mysql
- PostgreSQL
- Redis
- Thrift-RPC
- MongoDB
- Memcache
- NFS
- TLS
- SIP/SDP (beta)

从支持的网络协议可以看出，对于数据库类的支持还是比较完善的，包括多种数据库：Cassandra，MySQL，PostgreSQL，MongoDB和内存数据库Memcache，Redis，只需要进行简单的配置，即可完成对于协议的解析，并且可以将解析后的内容发送到Kafka，ElasticSearch，Logstash等；将解析后的协议数据存储后，可以对数据库的各类操作进行日志查询，识别风险SQL，分析操作系统，生成合规报表等。

Packetbeat支持两种部署方式

- 通过镜像端口或者Tap设备来获取流量，部署在特定的服务器上
- 直接部署在应用服务器上（这里应用服务器在MySQL抓包上可以是Client业务服务器，也可以是Server服务器）

直接部署在在镜像端口的采集设备是上对应用服务器不会产生影响，但是需要专门的网络支持，通常在云环境下不支持这种配置；直接部署在应用服务器上，会消耗一定的应用服务器的资源。

Packetbeat采集原理

Packetbeat是由go实现的抓包，底层引用了github.com/tsg/gopacket来实现抓包逻辑，gopacket是一个使用go实现的对pcap，pfring，afpacket的封装库。使用一套通用的框架实现对各类网络协议的解析。

采集模式

libpcap

libpcap是Unix/Linux平台下的网络数据包捕获函数库，提供了一套可移植的用户层网络抓包API接口；主要包含两部分：网络分接头(Network Tap)和数据过滤器(Packet Filter)。网络分接头从网络设备驱动程序中收集数据拷贝，过滤器决定是否接收该数据包。Libpcap利用BSD Packet Filter(BPF)算法对网卡接收到的链路层数据包进行过滤。
libpcap的网络包拷贝路径包含4次：

- 1. DMA：网卡寄存器 -> 内核网卡缓冲区ring buffer
- 2. 内核网卡缓冲区 -> 内核数据结构sk_buff
- 3. 符合BPF规则的网络包 -> BPF内核缓冲区
- 4. 使用PF_PACKET从内核缓冲区 -> 用户缓冲区

libpcap具有广泛的应用，其中tcpdump和wireshark的底层实现都用到了libpcap。

af_packet

packetbeat支持在Linux上配置更高效的抓包功能，通过开启af_packet选项。主要使用到了PACKET_MMAP，通过mmap减少了一次内存拷贝，可以很大的提高抓包效率。

采集数据

packetbeat并没有对mysql的协议进行完全抓取，只是选取比较重要的客户端和命令进行抓取，主要包括增删改查以及SQL语句的预处理和执行4类命令，对于数据库的使用方来说已经涵盖了大部分的场景，通过对这些SQL内容的分析，可以实现一定程度的审计需求。

```
// Packet types
const (
mysqlCmdQuery      = 3
mysqlCmdStmtPrepare = 22
mysqlCmdStmtExecute = 23
mysqlCmdStmtClose   = 25
)
```

通过packetbeat采集到的数据可以output输出到不同的分析平台进行分析，这里简单介绍一种简便的将数据采集到SLS平台的方法，Logtail可以通过Lumberjack协议将Beats系列软件采集的数据上传到SLS。

1. 安装Logtail并配置lumberjack插件，Logtail会在BindAddress上开始监听

```
{
 "inputs": [
  {
   "detail": {
    "BindAddress": "0.0.0.0:5044"
   },
   "type": "service_lumberjack"
  }
 ]
}
```

2. 在Packetbeat配置输出方式到logstash

```
output.logstash:
 hosts: ["127.0.0.1:5044"]
```

启动Packetbeat即可将packetbeat采集的数据通过Logtail上传到SLS，通过SLS控制台可以进行查询分析，构建仪表盘，对于数据库行为中的异常和性能进行监控告警。

下图是通过Packbeat采集到SLS的数据片段，可以看出包含的数据字段包括客户端信息，服务端信息，执行命令类型，执行命令语句，网络流量，网络延迟等。

小结：Packetbeat方案的主要特点是支持数据库比较广泛，是一种比较经济的数据库抓包选项，通过简单配置即可完成对数据库审计数据的采集。

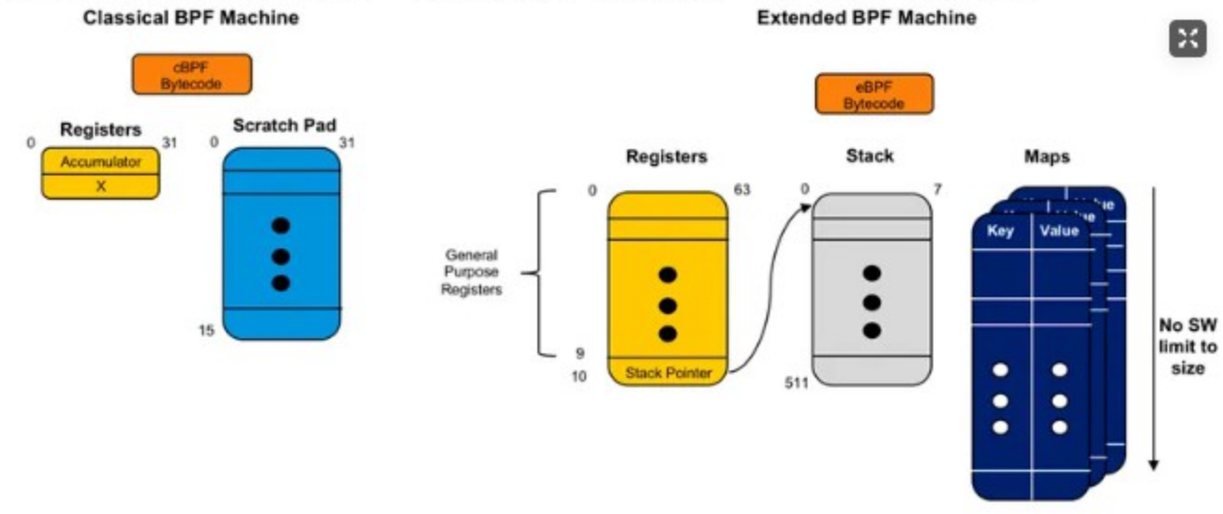
eBPF方案

eBPF (extended BPF)，是 Linux 内核对 BPF 的扩展。为了与传统的 BPF 更好地区别，传统的 BPF 现在被命名为 cBPF (classical BPF)。传统的BPF仅作为过滤网络包使用，eBPF扩展后增加了内核追踪，应用性能监控，流量控制等领域的功能。

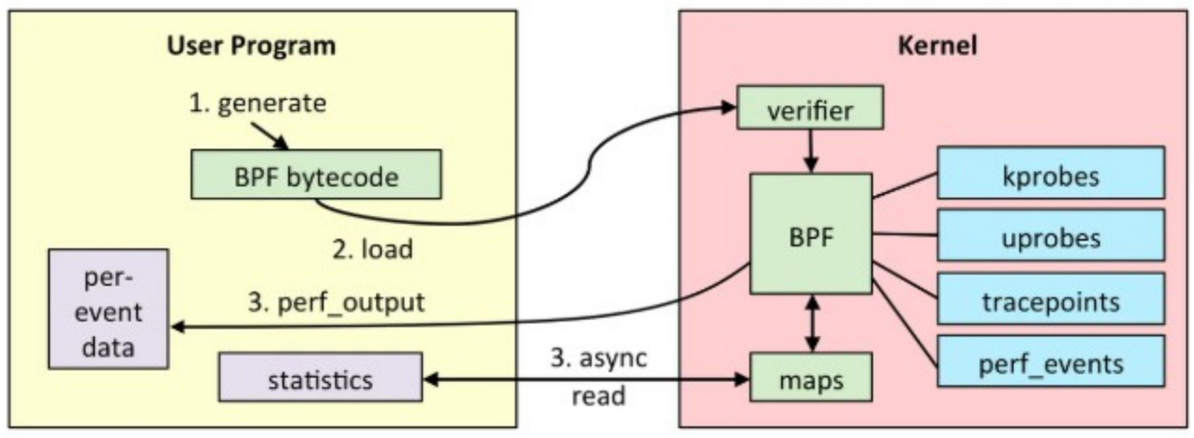
eBPF是一种允许在内核空间执行沙盒程序，而不需要修改内核内核代码的机制，eBPF的程序主要包含两大类，一类是tracing，主要用来追踪内核函数或者用户函数执行的传入传出参数、执行时间等；一类是网络相关，如XDP可以在网络包打到内核的前期进行干预，性能相比传统的包处理机制会高很多。

虚拟机

BPF和eBPF本质上都是一种小型虚拟机，可以运行在内核中。eBPF扩展了更多了寄存器，指令集，存储（Map）并支持更多内核事件处理支持，包括数据包，内核函数，用户函数，跟踪点等。



eBPF程序执行流程



eBPF程序分为用户空间和内核空间

- 用户空间负责加载BPF字节码到内核
- 在内核中BPF执行特定的内核事件，在执行的过程中可以将执行结果通过maps或者perf_output发送到用户空间。

核心流程包括

- 通过LLVM编译BPF代码为字节码
- 将字节码加载进内核，内核通过verifier校验BPF程序的安全性，防止内核崩溃，确认安全后将其加载到相应的内核模块执行，相关的程序主要包括kprobes, uprobes, tracepoints和perf_events。
- BPF程序执行过程中的返回值可以通过maps和perf_output传回指用户空间。

bcc

bcc 是一个为了方便的创建高效内核跟踪和操作程序的工具包，包括一些开箱即用的工具和示例。 它基于 eBPF 开发，使用python和Lua使得eBPF程序写起来更加简便，用户可以只关心BPF程序的编写，bcc会负责编译，解析ELF，加载BPF代码，创建map等。

bcc hello world

一个简单实例hello world，跟踪kprobes的例子，只要sys_clone执行时，控制台就会打印hello world。

```
#!/usr/bin/python
# Copyright (c) PLUMgrid, Inc.
# Licensed under the Apache License, Version 2.0 (the "License")

# run in project examples directory with:
# sudo ./hello_world.py"
# see trace_fields.py for a longer example

from bcc import BPF

# This may not work for 4.17 on x64, you need replace kprobe__sys_clone with kprobe____x64_sys_clone
BPF(text='int kprobe__sys_clone(void *ctx) { bpf_trace_printk("Hello, World!\\n"); return 0; }').trace_print()
```

实现数据库性能查询的示例

跟踪 MySQL/PostgreSQL 的查询时间高于阈值，查询阈值高于2毫秒的语句，参考

```
$ dbslower mysql -p `pidof mysqld` -m 2

Tracing database queries for pids 3350 slower than 2 ms...
TIME(s)      PID       MS QUERY
1.765087     3350      2.996 UPDATE sbtest1 SET k=k+1 WHERE id=963
3.187147     3350      2.069 UPDATE sbtest1 SET k=k+1 WHERE id=628
5.945987     3350      2.171 UPDATE sbtest1 SET k=k+1 WHERE id=325
7.771761     3350      3.853 UPDATE sbtest1 SET k=k+1 WHERE id=595
```

关键源码分析：https://github.com/iovisor/bcc/blob/master/tools/dbslower.py
通过BPF.get_user_functions_and_addresses通过符号表获取符合dispatch_command正则的用户态函数，

```
...
regex = "\\w+dispatch_command\\w+"
    symbols = BPF.get_user_functions_and_addresses(args.path, regex)

    if len(symbols) == 0:
        print("Can't find function 'dispatch_command' in %s" % (args.path))
        exit(1)

    (mysql_func_name, addr) = symbols[0]
    ...
```

通过attach_uprobe和attatch_uretprobe跟踪函数执行的过程，并通过query_start和query_end计算sql语句执行的延迟，并通过perf_output传递到用户空间。

```
...
# Uprobes mode
  bpf = BPF(text=program)
  bpf.attach_uprobe(name=args.path, sym=mysql_func_name,
                fn_name="query_start")
  bpf.attach_uretprobe(name=args.path, sym=mysql_func_name,
                  fn_name="query_end")
  ...
```

上面通过eBPF对uprobes的追踪来实现对数据库查询的解析和检测。

Pixie

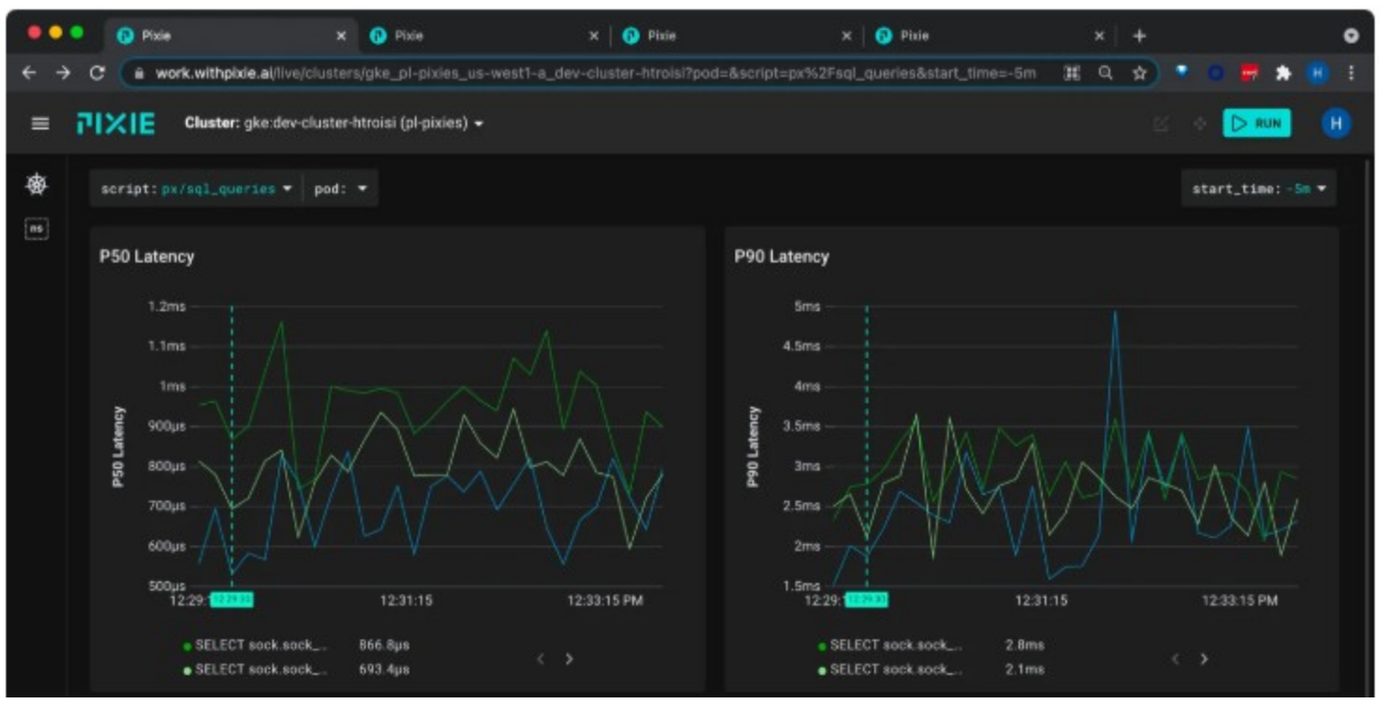
Pixie是New Relic开源的K8S观测与诊断平台，基于eBPF，无需修改应用就可以通过PxL脚本对系统和应用进行诊断，观测和调试。Pixie特色有三：

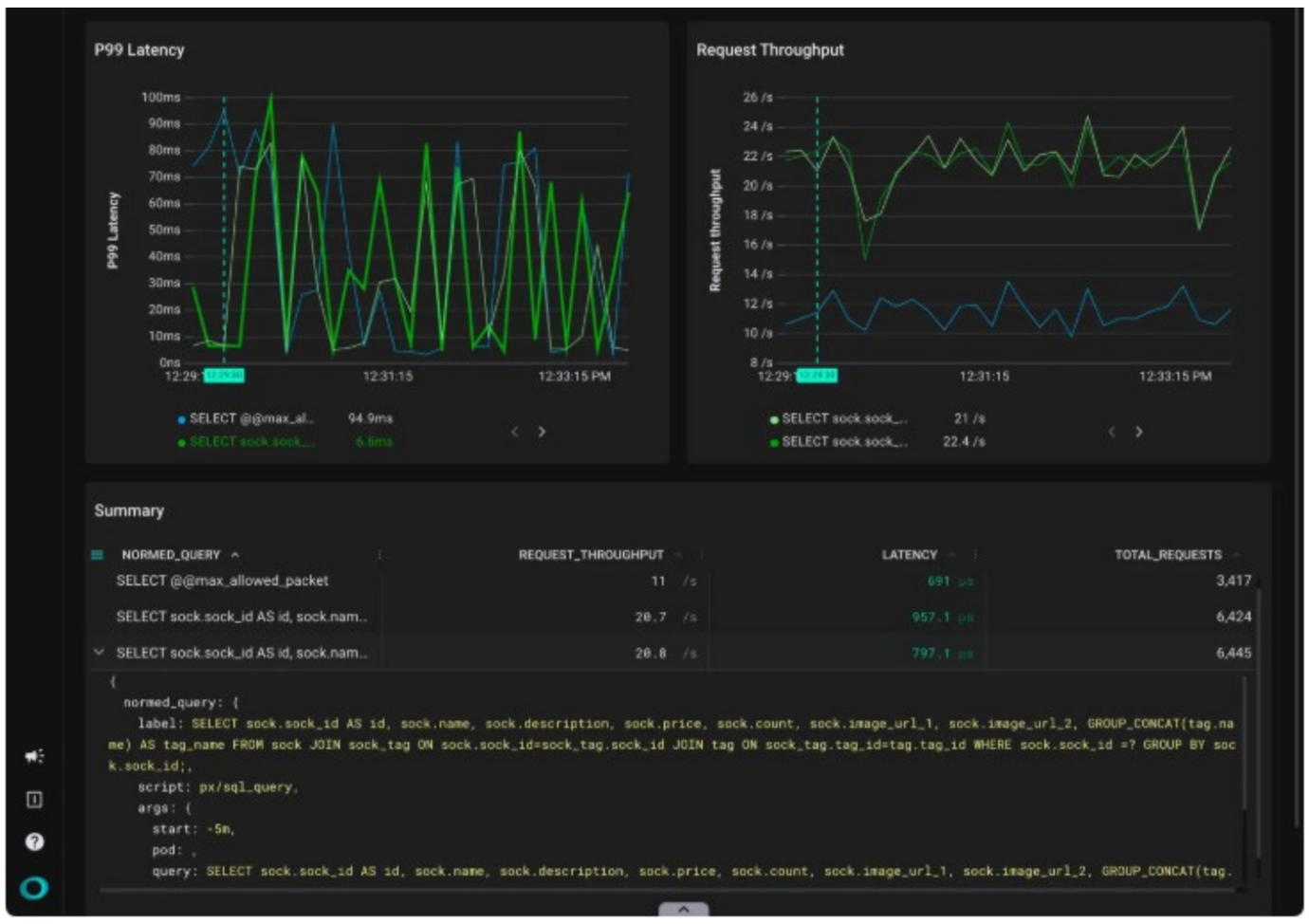
- 自动测量（Auto-telemetry）：借助eBPF自动收集测量数据，包含不限于请求体，资源，网络数据，应用调优等。
- 集群内边缘计算（In-cluster edge compute）：Pixie收集，存储，查询观测数据都在本地集群，不需要发送到外部平台。
- 可编程（Scriptability）：PxL的pythonic查询语言，可以用在UI，CLI，API中。

下面通过一个例子来查看Pixie的功能。

Database query profiling

通过px/sql_queries可以查看sql执行的整体延迟和吞吐量，同时在页面底部还提供了Normalized Query的展示，说明Pixie已经拿到了Sql的执行语句和查询延迟，接下来通过源码来看下Pixie的数据库检测与bcc的dbslower基于uprobe监测有什么不同。





原理分析

Stirling是Pixie的一个数据收集器，Stirling通过eBPF技术来收集节点上应用性能数据，内核事件，系统库和应用本身的数据。主要收集的数据包括：

- 应用的CPU，内存和网络使用率等。
- 应用的网络消息事件，包括不同的协议，如HTTP，MySQL，Posgres等。

Stirling将不同的采集源称为source connectors，大部分的采集源使用eBPF技术来收集数据，同时使用bcc工具来编写采集源。
这里我们关心的mysql的数据源采集在socket_tracer中。

Socket tracker deploys eBPF probes onto network IO syscalls (read/write, send/recv etc.), captures data, and reassemble & parse them back into application-level protocol messages.
可以看出pixie的socket tracer主要通过将探针放到网络IO系统调用上，来对网络数据进行抓包，组装并转成应用层消息，其中http2/gRPC使用uprobes，其他所有协议使用kprobes方式。

通过socket_trace_connector.h文件，可以看出Pixe使用跟踪kprobe的方式来进行mysql的数据的跟踪。通过跟踪内核的读写函数，剩下的协议解析与packeat等产品大同小异。区别在于两者采集方式的不同。

```
...
static constexpr auto kProbeSpecs = MakeArray<bpf_tools::KProbeSpec>({
    {"connect", bpf_tools::BPFProbeAttachType::kEntry, "syscall__probe_entry_connect"},
    {"connect", bpf_tools::BPFProbeAttachType::kReturn, "syscall__probe_ret_connect"},
    {"accept", bpf_tools::BPFProbeAttachType::kEntry, "syscall__probe_entry_accept"},
    {"accept", bpf_tools::BPFProbeAttachType::kReturn, "syscall__probe_ret_accept"},
    {"accept4", bpf_tools::BPFProbeAttachType::kEntry, "syscall__probe_entry_accept4"},
    {"accept4", bpf_tools::BPFProbeAttachType::kReturn, "syscall__probe_ret_accept4"},
    {"open", bpf_tools::BPFProbeAttachType::kReturn, "syscall__probe_ret_open"},
    {"creat", bpf_tools::BPFProbeAttachType::kReturn, "syscall__probe_ret_open"},
    {"openat", bpf_tools::BPFProbeAttachType::kReturn, "syscall__probe_ret_open"},
    {"write", bpf_tools::BPFProbeAttachType::kEntry, "syscall__probe_entry_write"},
    {"write", bpf_tools::BPFProbeAttachType::kReturn, "syscall__probe_ret_write"},
    {"writev", bpf_tools::BPFProbeAttachType::kEntry, "syscall__probe_entry_writev"},
    {"writev", bpf_tools::BPFProbeAttachType::kReturn, "syscall__probe_ret_writev"},
    {"send", bpf_tools::BPFProbeAttachType::kEntry, "syscall__probe_entry_send"},
    {"send", bpf_tools::BPFProbeAttachType::kReturn, "syscall__probe_ret_send"},
  ...
```

在protocols/mysql/types.h中可以看到对于mysql的网络协议的定义，包括了客户端往服务端发送的命令协议和服务端回复客户端的包协议。

eBPF支持的程序类型对于网络包的处理，除了BPF_PROG_TYPE_KPROBE类型之外，还有BPF_PROG_TYPE_SOCKET_FILTER，BPF_PROG_TYPE_XDP等类型，在流量流动的过程中经过的地方，都可以通过eBPF程序进行追踪采集。

在Pixie的实现中，对于数据库MySQL的网络抓包，是通过eBPF的kprobes技术来实现，与网络抓包相比，数据已经通过内核网络协议栈，与不经过内核协议栈的其他抓包或者eBPF类型程序相比会有一定的性能损失。Pixie的实现为数据库审计数据采集提供了另一种思路。

总结

通过对Packetbeat，eBPF的调研，可以发现在数据库审计的抓包技术，不仅包括传统的旁路网络抓包，还包括内核抓包的方式，传统的旁路镜像抓包方式具有对技术成熟，生态丰富的优点，并且可以做到对数据库服务器或者应用服务器性能不影响。eBPF作为一种新的内核跟踪技术，正在快速发展，从内核可观测到云原生可观测，都有着不错的开源项目，在网络抓包领域也具有相应的应用。

数据库审计抓包方案对于数据库审计系统至关重要，通过对Packetbeat及eBPF方案的调研，可以看出抓包方案也在不断的进化，数据库审计抓包技术在业务技术架构不断演进的过程中也需要不断的发展。其他高性能网络抓包方案还有PF_RING，DPDK等，本文主要关注数据库采集相关，留待后续调研。

参考

- MySQL协议，https://dev.mysql.com/doc/internals/en/overview.html
- MySQL网络协议解析，https://scala.cool/2017/11/mysql-protocol/
- Packetbeat Overview，https://www.elastic.co/guide/en/beats/packetbeat/current/packetbeat-overview.html
- Packet_MMAP，https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt
- eBPF技术简介，https://cloudnative.to/blog/bpf-intro/
- bcc dbslower，https://github.com/iovisor/bcc/blob/master/tools/dbslower.py
- bcc观测MySQL延迟，https://opensource.actionsky.com/20200324-mysql/
- Pixie Database Query Profiling，https://docs.pixielabs.ai/tutorials/pixie-101/database-query-profiling/
- Pixie github，https://github.com/pixie-io/pixie
- SLS Logtail采集Beats和Logstash数据源，https://help.aliyun.com/document_detail/74394.html

文章标签： 数据库审计 | 云解析DNS | 云数据库 RDS MySQL 版 | 日志服务 | Cloud Native | 应用服务中间件 | 数据采集 | SQL | 关系型数据库 | NoSQL | 数据安全/隐私保护 | 网络协议 | MySQL | 数据库

关 键 词： 云数据库 RDS MySQL 版技术 | 数据库技术 | 云数据库 RDS MySQL 版采集 | 数据库审计 | 数据库采集

评论

登录后可评论

相关文章

阿里云瑶池数据库_ | 4月前 | Cloud Native · 关系型数据库 · 分布式数据库

登顶TPC-C | 云原生数据库PolarDB技术揭秘：弹性并行查询（ePQ）篇

「PolarDB登顶TPC-C技术揭秘」系列硬核文章

154 13 13

阿里云瑶池数据库_ | 4月前 | 存储 · 关系型数据库 · 分布式数据库

登顶TPC-C | 云原生数据库PolarDB技术揭秘：高可用-无感切换篇

「PolarDB登顶TPC-C技术揭秘」系列硬核文章

152 0 0

阿里云瑶池数据库_ | 4月前 | 存储 · 关系型数据库 · 分布式数据库

登顶TPC-C | 云原生数据库PolarDB技术揭秘：成本优化-软硬协同篇

详细介绍 PolarDB 软硬协同的 I/O 链路技术

264 3 3

数据库知识分享者小北 | 5月前 | Cloud Native · 关系型数据库 · 分布式数据库

登顶TPC-C | 云原生数据库PolarDB技术揭秘：Limitless集群和分布式扩展篇

阿里云PolarDB云原生数据库在TPC-C基准测试中以20.55亿tpmC的成绩刷新世界纪录，展现卓越性能与性价比。其轻量版满足国产化需求，兼具...

418 6 6

数据库知识分享者小北 | 5月前 | 存储 · 关系型数据库 · 分布式数据库

登顶TPC-C | 云原生数据库PolarDB技术揭秘：单机性能优化篇

阿里云PolarDB云原生数据库在TPC-C基准测试中，以20.55亿tpmC的成绩打破性能与性价比世界纪录。此外，国产轻量版PolarDB已上线，提供...

599 33 33

蓝易云 | 30天前 | 缓存 · 关系型数据库 · MySQL

在MySQL中处理高并发和负载峰值的关键技术与策略

采用上述策略和技术时，每个环节都要进行细致的规划和测试，确保数据库系统既能满足高并发的要求，又要保持足够的灵活性来应对各种突发的...

83 15 15

数据库知识学习者 | 2月前 | 存储 · 人工智能 · 关系型数据库

诚邀您参加《智启云存：AI时代数据库RDS存储新突破》线上闭门技术沙龙！

诚邀您参加6月11日（周三）14:00在线上举行的《智启云存：AI时代数据库RDS存储新突破》闭门活动。免费报名并有机会获得精美礼品，快来报...

105 1 1

阿里云瑶池数据库_ | 3月前 | 人工智能 · 关系型数据库 · 分布式数据库

媒体声音｜从亚太到欧美，阿里云瑶池数据库凭何成为中企出海的技术底气？

在中企出海的时代浪潮中，瑶池数据库正凭借其技术创新、场景化解决方案、智能化能力、全球化布局，成为企业跨越挑战、构建全球竞争力的关...

146 8 8

阿里云瑶池数据库_ | 3月前 | SQL · AliSQL · 关系型数据库

致敬MySQL 30周年｜阿里云自研内核AliSQL的技术演进

30而立

308 8 8

NineData | 3月前 | 安全 · Apache · 数据库

【倒计时3天】NineData x Apache Doris x 阿里云联合举办数据库技术Meetup，5月24日深圳见！

5月24日，NineData联合Apache Doris与阿里云在深圳举办数据库技术Meetup。活动聚焦「数据实时分析」与「数据同步迁移」两大领域，邀请...

80 1 1