Zeek's Logging Framework

Streams, filters, writers, and related features

Christian Kreibich
christian@corelight.com
@ckreibich

Part 1

Background

Zeek

Packets

Events

→

Zeek

$$\begin{tikzcd} \text{Packets} & & \\ {} \arrow[r] & \text{Zeek} \arrow[r] & ? \\ \text{Events} & & \end{tikzcd}$$

$$\begin{tikzcd}\text{Packets} \\ \text{Events} \arrow[r] & \text{Zeek} \arrow[r] & \text{Logs!}\end{tikzcd}$$

Part 2

Logging basics

Set the stage

```
# Create ID for our new stream:
redef enum Log::ID += { LOG };

# Define (or redef, in order to extend) the record type to log.
# This defines all log columns and data types:
type Info: record {
    ts: time        &log;
    id: conn_id     &log;
    service: string &log &optional;
    missed_bytes: count &log &default=0;
};

event bro_init() {
    # Create the stream. This adds a default filter.
    Log::create_stream(Foo::LOG, [$columns=Info,
                                  $path="foo"]);
}
```

Write a log entry

```
# Identify suitable event and trigger write:
event connection_established(c: connection) {
    local rec: Foo::Info = [$ts=network_time(), $id=c$id];
    Log::write(Foo::LOG, rec);
}
```

Enjoy output file foo.log!

```
#separator \x09
#set_separator  ,
#empty_field (empty)
#unset_field -
#path   foo
#open   2019-04-07-00-27-05
#fields ts      id.orig_h       id.orig_p       id.resp_h       id.resp_p       service missed_bytes
#types  time addr port addr port string  count
1052146262.950001       203.241.248.20  3051     80.4.124.41     80    -    0
#close  2019-04-07-00-27-05
```

Or, with `LogAscii::use_json=T…`

```
{"ts":1052146262.950001,
 "id.orig_h":"203.241.248.20",
 "id.orig_p":3051,
 "id.resp_h":"80.4.124.41",
 "id.resp_p":80,
 "missed_bytes":0}
```

Key components

- **Streams** identify data flows
    - **What** to log
- **Filters** control their manifestations
    - **How / whether** to log
- **Writers** output the data
    - **Where** to log

Key component: streams

```
# Type defining the content of a logging stream.
type Stream: record {
    # A record type defining the log's columns.
    columns: any;

    # Event that will be raised once for each log entry.
    ev: any &optional;

    # A file path inherited by any filters added to the stream
    path: string &optional;
};
```

Key component: filters

```
# A filter type describes how to customize logging streams.
type Filter: record {
    # Descriptive name to reference this filter.
   name: string;

   # The logging writer implementation to use.
   writer: Writer &default=default_writer;

   # Indicates whether a log entry should be recorded.
   pred: function(rec: any): bool &optional;

   # Output path for recording entries.
   path: string &optional;

   # Subsets of column names to record.
   include: set[string] &optional;
   exclude: set[string] &optional;

   # Many others -- take a look in scripts/base/frameworks/logging/main.bro!
```

Filter predicate example

```
function http_only(rec: Conn::Info) : bool {
    # Record only connections with successfully analyzed HTTP traffic
    return rec?$service && rec$service == "http";
}

event bro_init() {
    local filter: Log::Filter = [$name="http-only", $path="conn-http",
                                 $pred=http_only];
    Log::add_filter(Conn::LOG, filter);
}
```

Key components: writers

- In-core components, built as Zeek plugins
- Support a wide range of output destinations

- ASCII
- SQLite
- None
- Postgres
- Kafka
- ZeroMQ
- MongoDB
- RITA

Key components: writers

- In-core components, built as Zeek plugins
- Support a wide range of output destinations

- ASCII
- SQLite
- None

Shipped with Zeek

- Postgres
- Kafka
- ZeroMQ
- MongoDB
- RITA

Key components: writers

- In-core components, built as Zeek plugins
- Support a wide range of output destinations

- ASCII
- SQLite
- None

Shipped with Zeek

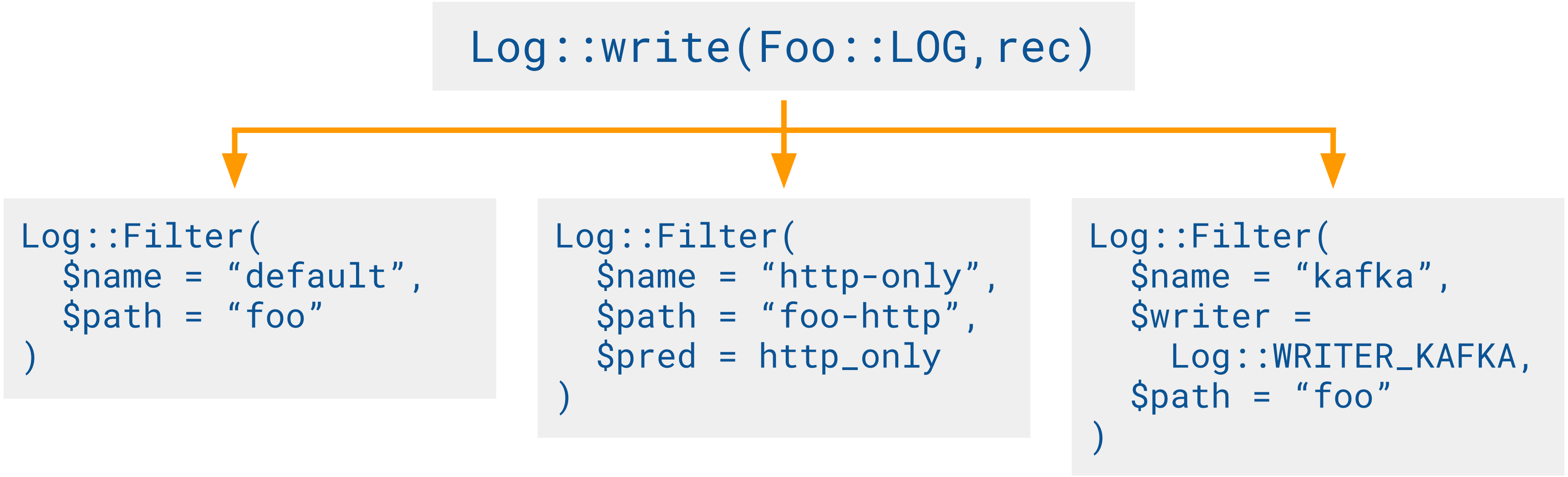
- Postgres
- Kafka
- ZeroMQ
- MongoDB
- RITA

Available via bro-pkg

A log write

```
Log::write(Foo::LOG,rec)
```

A log write



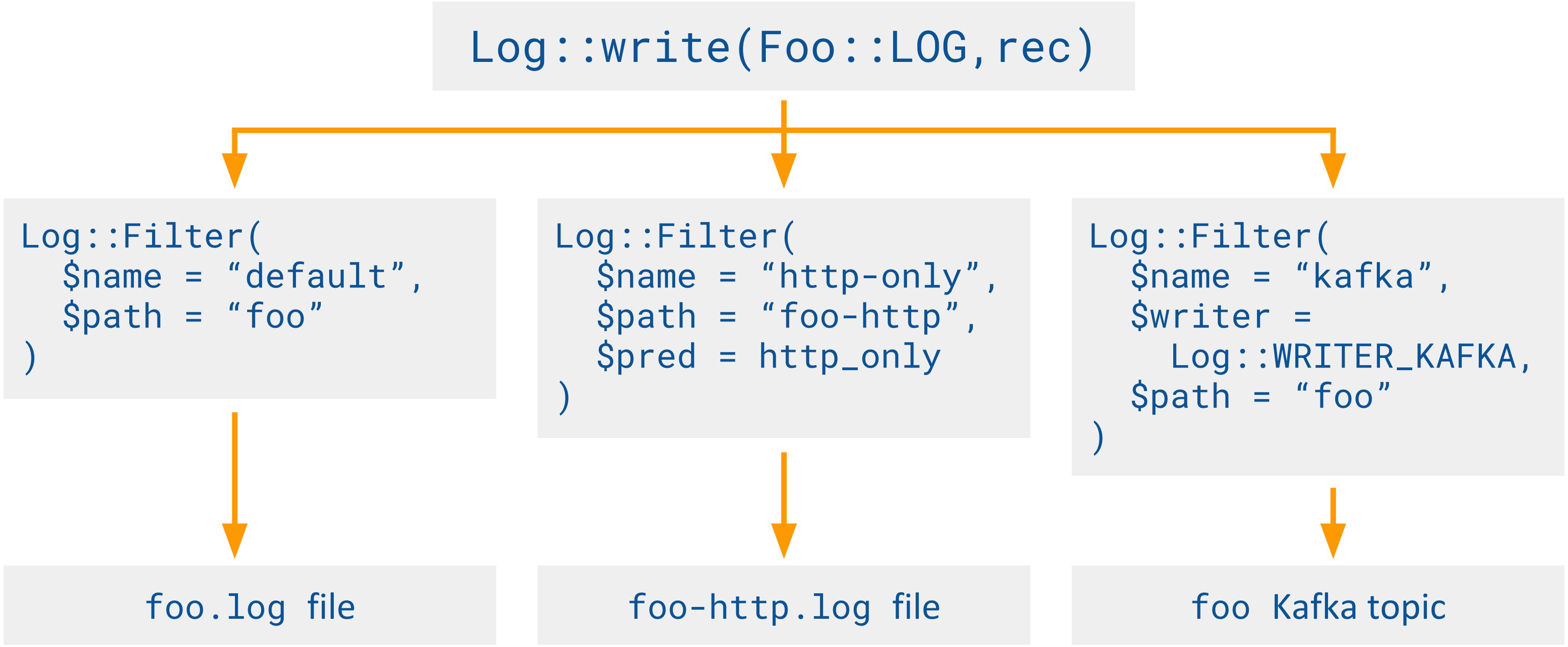
```
Log::write(Foo::LOG,rec)
```

```
Log::Filter(
  $name = “default”,
  $path = “foo”
)
```

```
Log::Filter(
  $name = “http-only”,
  $path = “foo-http”,
  $pred = http_only
)
```

```
Log::Filter(
  $name = “kafka”,
  $writer =
    Log::WRITER_KAFKA,
  $path = “foo”
)
```

A log write



```
Log::write(Foo::LOG,rec)
```

```
Log::Filter(
  $name = “default”,
  $path = “foo”
)
```

```
Log::Filter(
  $name = “http-only”,
  $path = “foo-http”,
  $pred = http_only
)
```

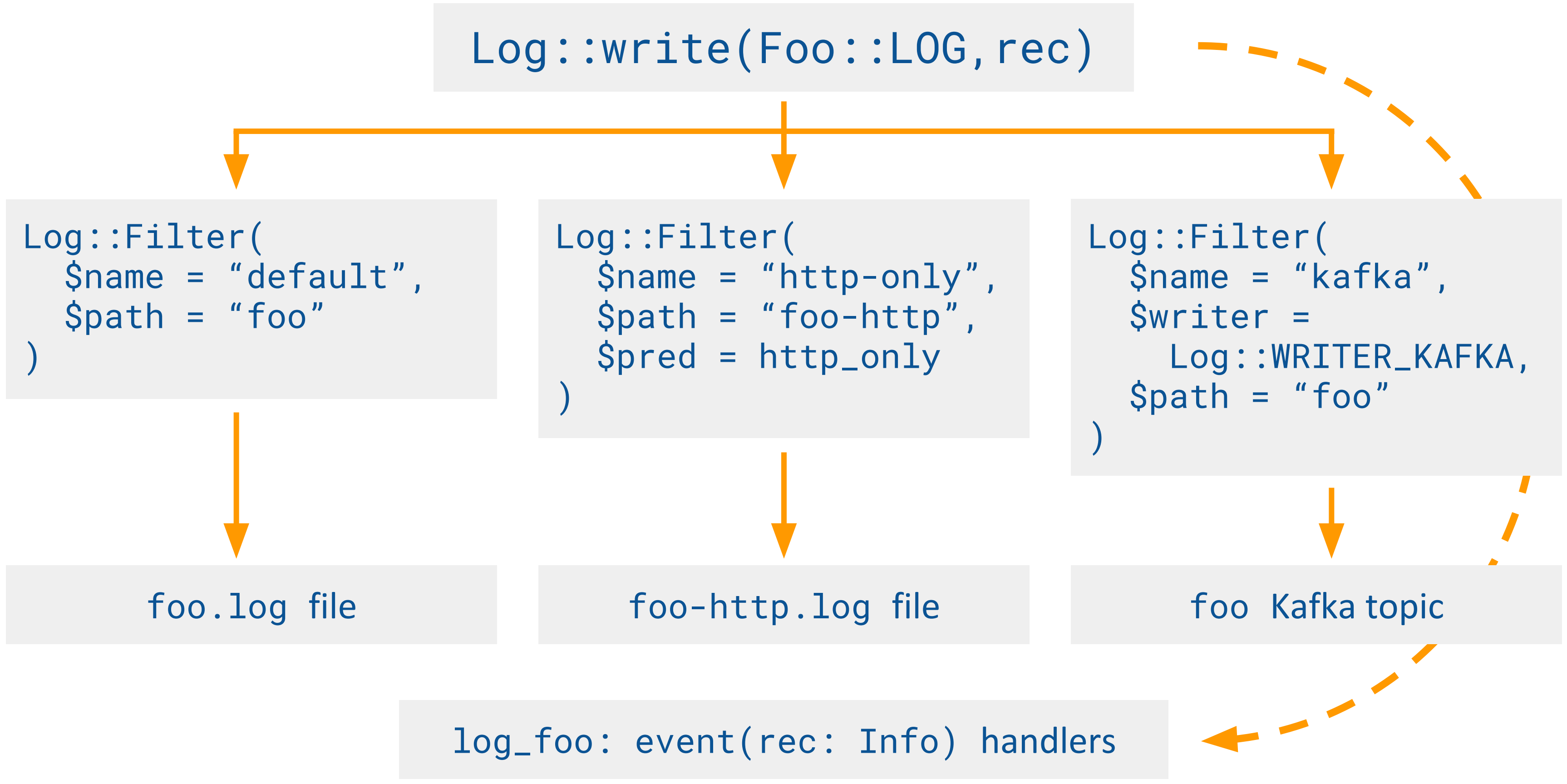
```
Log::Filter(
  $name = “kafka”,
  $writer =
    Log::WRITER_KAFKA,
  $path = “foo”
)
```

`foo.log` file

`foo-http.log` file

`foo` Kafka topic

A log write



```
Log::Filter(
  $name = “default”,
  $path = “foo”
)
```

```
Log::Filter(
  $name = “http-only”,
  $path = “foo-http”,
  $pred = http_only
)
```

```
Log::Filter(
  $name = “kafka”,
  $writer =
    Log::WRITER_KAFKA,
  $path = “foo”
)
```

Part 3

Recent(ish) additions

Log extensions

- Runtime mechanism to add columns to logs
- Can operate globally or per-filter
- More control than with the `redef` approach
- See `field-extension-*.bro` btests in Zeek tree
- Good packaged example: https://github.com/corelight/json-streaming-logs

Log extensions: in filter

```
type Filter: record {
    # ...

    # Function to collect a log extension value.  If not specified,
    # no log extension will be provided for the log.
    # The return value from the function *must* be a record.
    ext_func: function(path: string): any &default=default_ext_func;

    # ...
};
```

Log extensions: global application

```
# Some metadata we want to add to each log entry:
type Extension: record {
    write_ts: time &log;
    stream: string &log;
    system_name: string &log;
};

# The extension callback, producing the values:
function add_extension(path: string): Extension {
    return Extension($write_ts = network_time(),
                     $stream = path,
                     $system_name = peer_description);
}

# Register the callback as a global default:
redef Log::default_ext_func = add_extension;
```

Log extensions: resulting log

```
#separator \x09
#set_separator  ,
#empty_field   (empty)
#unset_field   -
#path   conn
#open   2016-08-10-17-45-11
#fields _write_ts      _stream _system_name   ts      uid    ...
#types  time           string  string         time    string ...
1300475173.475401       conn    bro    1300475169.780331 ...
1300475173.475401       conn    bro    1300475168.892913 ...
```

Plugin hooks

- Zeek plugins can hook into the logging framework
- Useful e.g. for fine-grained, stateful control
- HOOK_LOG_INIT for writer instantiation
- HOOK_LOG_WRITE prior to log writes

Part 4

Quirks

Filter predicates don’t chain well

- After filter creation there’s no API to manage them
- Assigning a new one means clobbering the old one

```
event bro_init() {
    local filter: Log::Filter =
        [$name="http-only", $path="conn-http",
         $pred=http_only];
    Log::add_filter(Conn::LOG, filter);
}
```

Filter predicates have no context

- Closures would be a nice solution to this one

```
type Filter: record {
    # ...

    # Indicates whether a log entry should be recorded.
    pred: function(rec: any): bool &optional;

    # ...
};
```

Beware of tweaking default filters

- Easy to create subtle last-tweak-wins scenarios
- Be mindful of existing modifications

Extension functions don’t see log data

- So they can’t make decisions based on them
- Example: tweak any log entry that has an IP address, but not others

```
type Filter: record {
    # ...

    ext_func: function(path: string): any
        &default=default_ext_func;

    # ...
};
```

Summary

- Logging works on **records**, managed as **streams**, controlled by **filters**, and directed by **writers**
- Mostly settled framework, with inconveniences around predicates and extensions
- This talk skipped several features, e.g. rotation, postprocessors