A Bro Primer

Presenter: Adam Pumphrey, Bricata

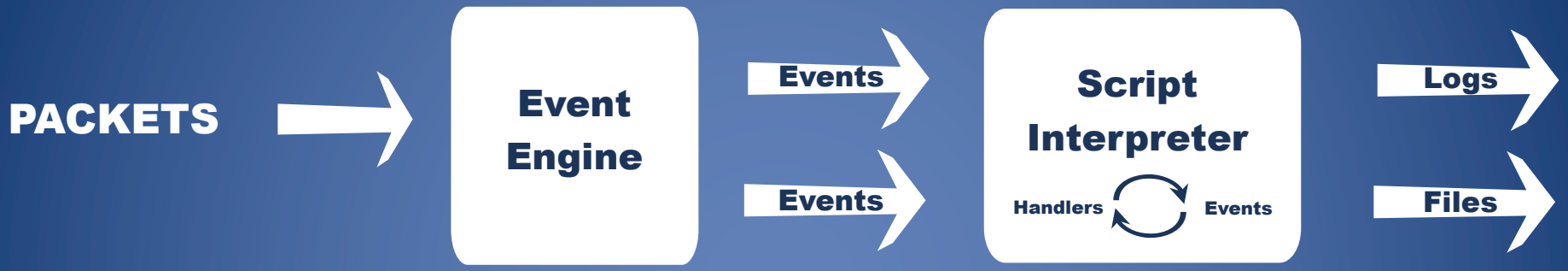
Intro

- Working in cybersecurity for about 17 years - most of which was with the civilian federal government
- Role and responsibilities have varied but mainly my work has been in network defense and cybersecurity
- First exposure to Bro was in 2009; engineers proposed it as a replacement for a network monitoring tool set that included argus, dsniff and httpry
- Went on to work on a variety of projects, stand up IR teams, build SOC's, design and deploy custom monitoring solutions… Bro has been part of the stack ever since

Purpose

- Many network and security operations personnel don't come from a programming background
- The potential value is apparent, Bro's logs can be used for monitoring, threat detection, incident response and forensics
- Learning the programming language can be a daunting task, but is necessary in order to realize Bro's full potential
- Several concepts that are central to how Bro works are also very relevant to learning the language
- Learning how to perform common, but frequently needed, tasks can be a great way to get started

Bro Core

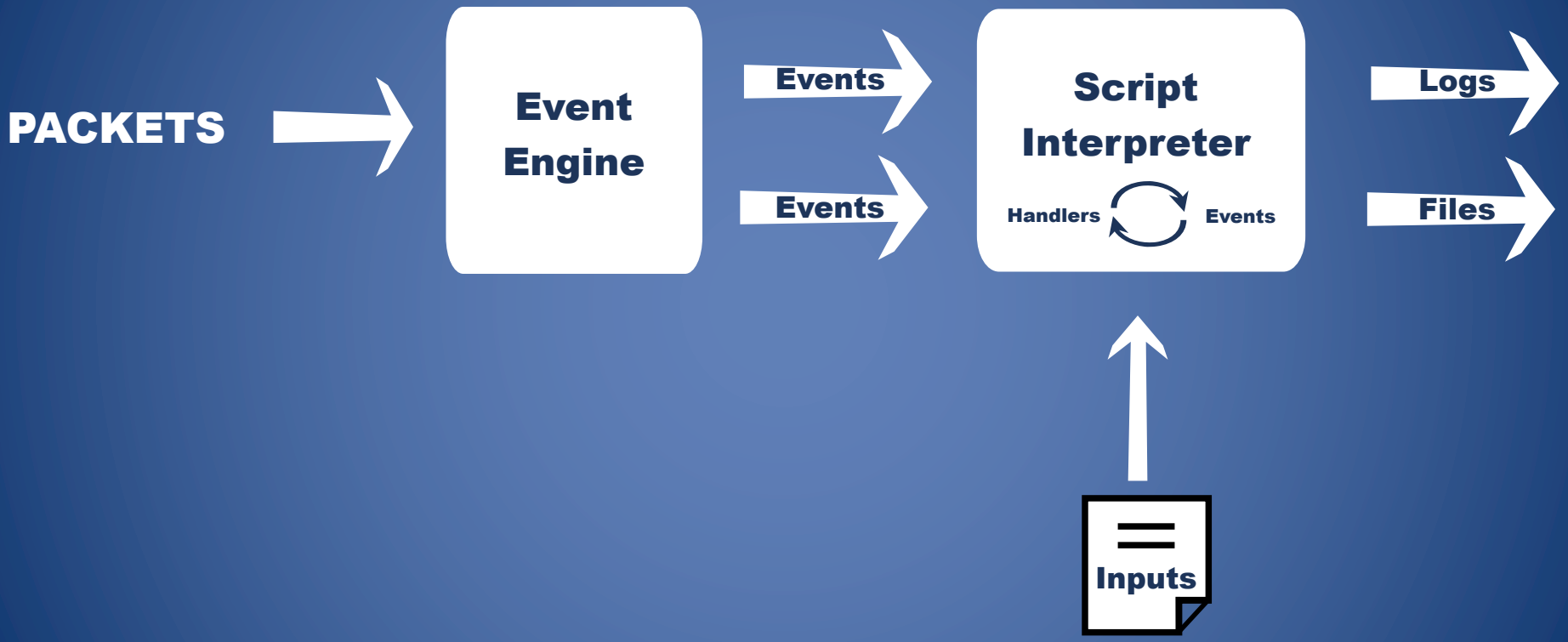


Things to know...

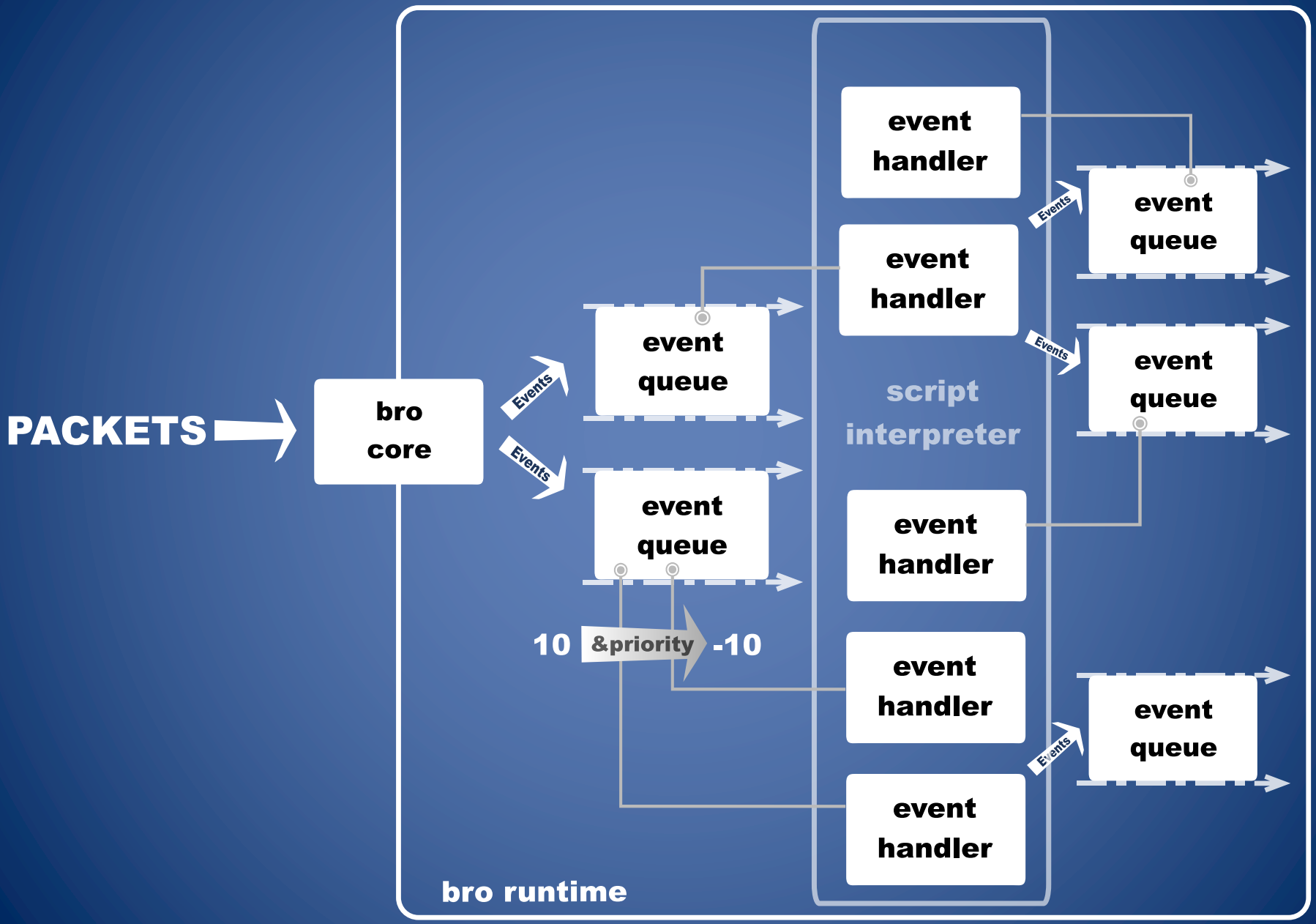
- You should be able to locate the local.bro policy file, typically here:
  - $BRO_HOME/share/bro/site/local.bro
- You should understand the purpose and use of Bro's *@load* directive
  - Very similar to ***include*** and ***import*** commands found in other languages
  - Specifies a script (absolute or relative paths) or a module directory
  - If a directory name is specified Bro will attempt to load the __load__.bro file the directory contains
- You have defined the Site::local_nets and Site::local_zones variables

Events

Event Interactions



Event Lifecycle



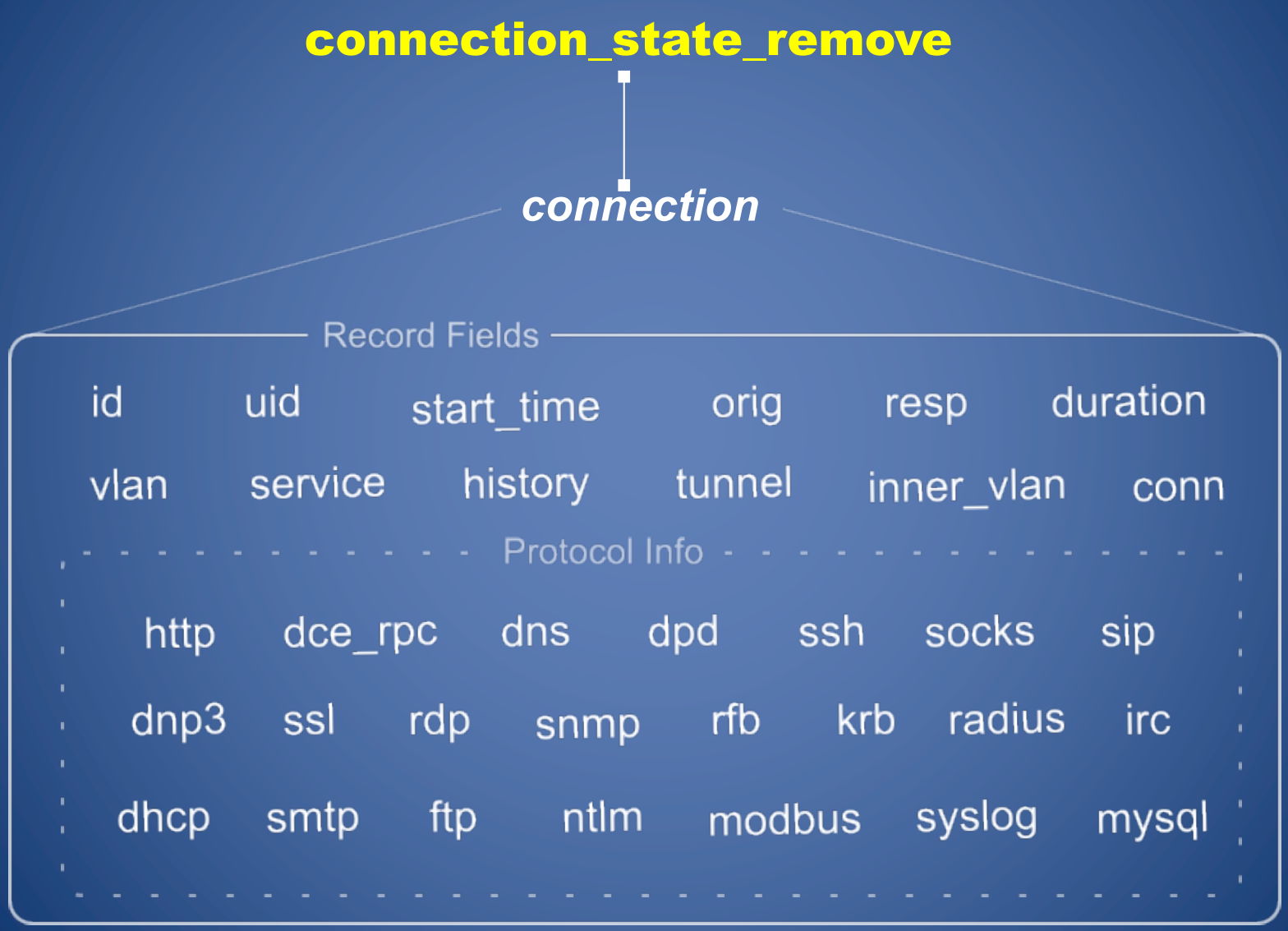
Bro Process Events

bro_init	Generated once, when Bro initializes. Script operations that only need to be executed once in the lifetime of a Bro process should occur in this event handler.
bro_done	This Bro process is terminating. Any operations that preserve or cleanup what is in memory should go here.

Connection State Events

new_connection	A new connection has been observed. Generated for every new packet that is not part of a tracked connection. Bro's flow-based definition of connection includes TCP, UDP and ICMP flows.
connection_established	The SYN-ACK packet from a responder in a new connection has been seen. This does not indicate the 3-way handshake has completed. Bro tracks the state either way.
connection_state_remove	A connection is being removed from memory. By this point, all protocol analyzers have attached their data to the connection record and it is about to be written to the conn log stream.
udp_session_done	A UDP transaction has completed. Intended to make UDP handling more like TCP, supported protocols are: DNS, NTP, NetBIOS, Syslog, AYIYA, Teredo, and GTPv1

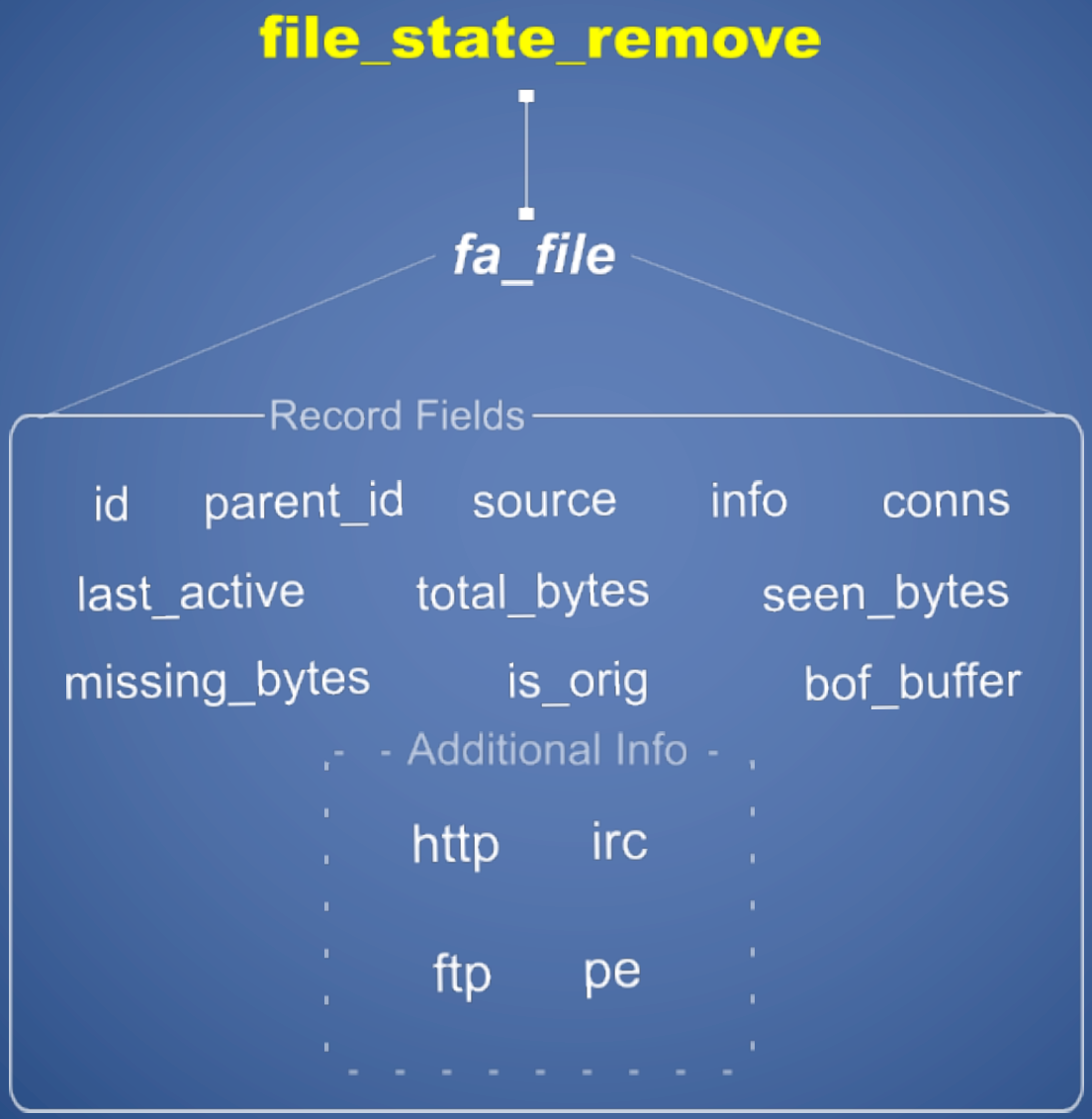
connection_state_remove and ***connection*** records



File Analysis Events

file_new	Generated once for each new file Bro identifies and begins to analyze. Contains information about the connection, but nothing about the file.
file_sniff	Generated once, for each analyzed file. Contains the inferred metadata, including mime_type, based on analysis of the first chunk of the file.
file_state_remove	File analysis for a file is ending. At this point the fa_file record contains all of the information gathered by the file analyzers that ran.

file_state_remove and fa_file record



Application Layer Protocol Events

smtp	arp	ssl	imap	xmpp	smb
udp	modbus		sip	gnutella	irc
dce-rpc	tcp		bittorrent	gtpv1	rpc
smnp	dhcp		ssh	syslog	krb
ntlm	ntp		teredo	dnp3	ident
socks	mysql		rfb	finger	rdp
dns	ncp		ftp	http	icmp
netbios			radius		pop3

Find out more about events…

- List of protocol-independent events Bro generates
  - https://www.bro.org/sphinx/scripts/base/bif/event.bif.bro.html
- Review events generated by the various plugins
  - https://www.bro.org/sphinx/scripts/base/bif/plugins
- Documentation of Files framework and the log_files event
  - https://www.bro.org/sphinx/scripts/base/frameworks/files/main.bro.html#events
  - file_sniff and other file analysis events are generated by Bro core, see the top URL for more info

Log Streams and Filters

Bro Log Format

- Bro's built-in ASCII writer provides two primary output formats:
  - Tab-delimited (Bro's proprietary log format)
  - JSON
- The default is tab-delimited, but you can enable JSON with Bro Script

```
# Enable JSON Logging
redef LogAscii::use_json = T;

# Specify the timestamp format, epoch is default
redef LogAscii::json_timestamps = "JSON::TS_ISO8601";
```

Bro Logs – Native Format

```
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path conn
#open 2017-08-14-20-02-32
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p…
#types time string addr port addr port enum string…
1502741076.550466 C1CxFS3pdJ8kwbmnMl 172.16.253.131 1046…
```

Bro Logs – JSON Format


```
{
        "ts":"2017-08-14T20:04:36.550466Z",
        "uid":"C1CxFS3pdJ8kwbmnM1",
        "id.orig_h":"172.16.253.131",
        "id.orig_p":1046,
        "id.resp_h":"64.90.61.19",
        "id.resp_p":80,
        "proto":"tcp",
        "service":"http",
        "duration":0.853405,
        "orig_bytes":316,
        …
}
```

Enable JSON for specific streams – Copy the default filter

```
event bro_init()
    {

    # Load the default filter of the Conn log
    local filter = Log::get_filter(Conn::LOG,"default");

    # Specify a new filter name
    filter$name = cat(filter$name, "-json");
    …
```


Enable JSON for specific streams – Configure JSON options

```
…
# Specify a new stream path
filter$path = cat(filter$path, "-json");

# Add the config options to the filter
filter$config = table(
    ["use_json"] = "T",
    ["json_timestamps"] = "JSON::TS_ISO8601");

# Apply the modified default filter
Log::add_filter(Conn::LOG, filter);

}
```

Disable a Log Stream

```
# Disable the communication log
event bro_init()
    {
    Log::disable_stream(Communication::LOG);
    }
```

Include only certain fields in a stream

```
# Handle the bro_init event
event bro_init()
    {
    # Retrieve the default filter for the HTTP log stream
    local f = Log::get_filter(HTTP::LOG, "default");

    # Define a new value for the "include" field
    f$include = set("ts" ,
                    "id.orig_h",
                    "host",
                    "uri");

    # Add the modified default filter back to the HTTP log
    Log::add_filter(HTTP::LOG, f);
    }
```

Remove certain fields from a stream

```
# Handle the bro_init event
event bro_init()
    {
    # Remove the default filter
    Log::remove_default_filter(SMTP::LOG);

    # Add a new filter, use the exclude field
    Log::add_filter(SMTP::LOG,
                    [$name = "no_smtp_recips",
                     $exclude = set("rcptto")]
                    );
    }
```

Log only select events

```
# Log only select events
event bro_init()
        {
        # Remove the default filter
        Log::remove_default_filter(SMTP::LOG);

        # Provide an argument to the $pred option
        Log::add_filter(SMTP::LOG,
                        [$name = "incoming_email",
                         $pred(rec: SMTP::Info) = {
                return ! Site::is_local_addr(rec$id$orig_h);
                                                }
                        ]);

        }
```

Route events based on their content – Define the function

```
# Function to return desired log file name
   function sort_mail(id: Log::ID, path: string,
                      rec: SMTP::Info): string
      {
      if (Site::is_local_addr(rec$id$orig_h) &&
            ( ! Site::is_local_addr(rec$id$resp_h)))
            return "outgoing_email";
      else if ( ! Site::is_local_addr(rec$id$orig_h) &&
                  Site::is_local_addr(rec$id$resp_h) )
            return "incoming_email";
      }
```

Route events based on their content – Apply the filter

```
event bro_init()
    {
    # Remove the default filter
    Log::remove_default_filter(SMTP::LOG);

    # Use sort_mail for the path function
    Log::add_filter(SMTP::LOG,
                    [$name = "email_sorter",
                     $path_func = sort_mail
                    ]);
    }
```

Add a field to a log stream – Modify the Info record

```
# Redef the Conn::Info record
export {
    # Add the new pcr field
    redef record Conn::Info += {
        pcr: double &log &optional;
    };
}
```

Add a field to a log stream – Populate the field

```
# Handle the appropriate event
event connection_state_remove(c: connection) &priority=3
    {
    # Verify required fields exists
    if ( ! c$conn?$orig_bytes || ! c$conn?$resp_bytes ) {
        return;
    }
    # Test for specific field value conditions
    else if (c$conn$orig_bytes == 0 && c$conn$resp_bytes == 0 ) {
        c$conn$pcr = 0.0;
    }
    # Calculate the new value and store in the pcr field
    else {
        local n = (c$conn$orig_bytes + 0.0) - (c$conn$resp_bytes + 0.0);
        local d = (c$conn$orig_bytes + 0.0) + (c$conn$resp_bytes + 0.0);
        local x = ( n / d );
        c$conn$pcr = x;
    }
}
```

Adding a new log stream

- Sample Use Case – Separately log flow and SSL information about potential file downloads occurring over HTTPS

- 3 parts:

1. Set up the environment
2. Create the log stream
3. Write events to the stream

Add a New Log Stream

Set up the script environment

```
# Load the required scripts
@load ./calculate_pcr
@load base/utils/site

# Define a module namespace
module https_transfer;
```


Define export block for module declarations

```
# Export the required objects
export {
    # Redef the Log::ID enum to include our module's stream
    redef enum Log::ID += {
        LOG
    };

    # Define an Info record for the log
    type Info: record {
        ts:              time              &log;
        uid:             string            &log;
        orig_h:          addr              &log;
        resp_h:          addr              &log;
        pcr:             double            &log;
        bytes_received:  count             &log;
        server_name:     string            &log;
        subject:         string            &log;
        issuer:          string            &log;
    };

    global log_https_transfer: event(rec: https_transfer::Info);
}
```

Create the log stream

```
# Create the stream inside the bro_init event handler
event bro_init()
    {
    Log::create_stream(https_transfer::LOG,
                       [
                        $columns=https_transfer::Info,
                        $ev=log_https_transfer,
                        $path="https_transfers"
                       ]);
    }
```

Test traffic conditions with the IF statement

```
# Handle connection_state_remove event to work with the pcr value
event connection_state_remove(c: connection) &priority=0
    {
    # Check for SSL between an internal client and external server
    if ( c?$ssl && (Site::is_local_addr(c$id$orig_h) && !
                        Site::is_local_addr(c$id$resp_h)) )
        {
        # Check if the pcr value signifies likely file download
        if ( c$conn$pcr <= -0.99 )
            {
       …
```

Write the Info record to the log stream

```
        …
        # Create and populate the Info record
        local rec = https_transfer::Info(
          $ts = c$start_time,
          $uid = c$uid,
          $orig_h = c$id$orig_h,
          $resp_h = c$id$resp_h,
          $pcr = c$conn$pcr,
          $bytes_received = c$conn$resp_bytes,
          $server_name = c$ssl?$server_name ?
                                c$ssl$server_name : "",
          $subject = c$ssl?$subject ? c$ssl$subject : "",
          $issuer = c$ssl?$issuer ? c$ssl$issuer : ""
        );

        # Write the Info record to the log stream
        Log::write(https_transfer::LOG, rec);
        }
    }
}
```

Additional Resources

- Script Reference, supplement to the documentation pulled from source code comments:
  - https://www.bro.org/sphinx/script-reference/index.html
    - Operators
    - Types
    - Attributes
    - Declarations and Statements
    - Directives
- FloCon 2014 PCR Presentation by Cater Bullard and John Gerth
  - https://qosient.com/argus/presentations/Argus.FloCon.2014.PCR.Presentation.pdf