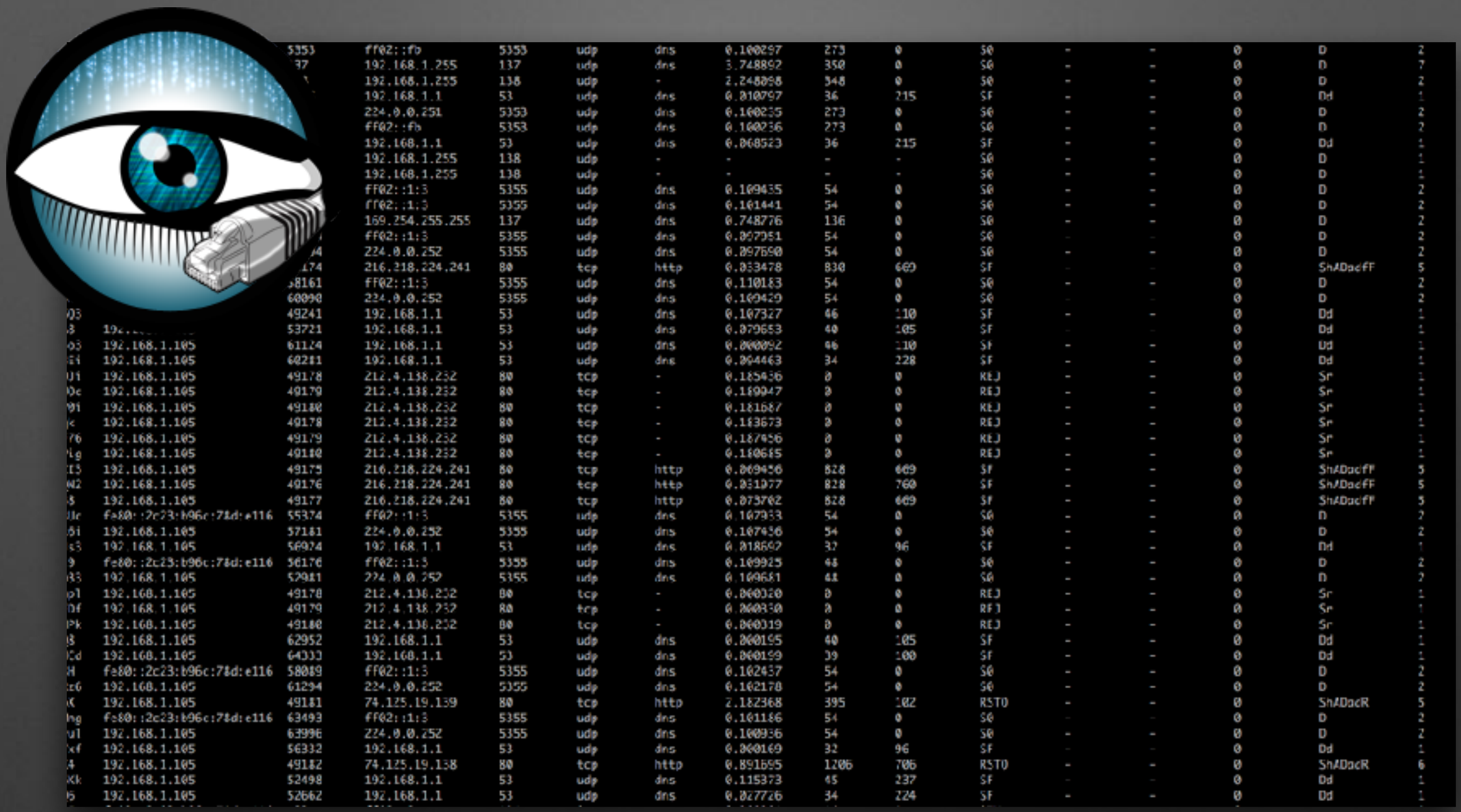


Put your thoughts on the wire with bro.org

Seth Hall
Corelight, Inc
Co-founder and Chief Evangelist

A bit about me

- BS in Geography from OSU
- Incident Responder at OSU
- Detection/Response architect at GE
- Core Bro developer at ICSI
- Co-founder and Chief Evangelist at Corelight

A bit about me

- Always loved playing with programming languages.
- Discovered love for network traffic analysis.
- Intersection between these would be amazing!
- I did what any sensible person in my situation would do…

Doubled down and became an expert!
(in a niche language with very few users)

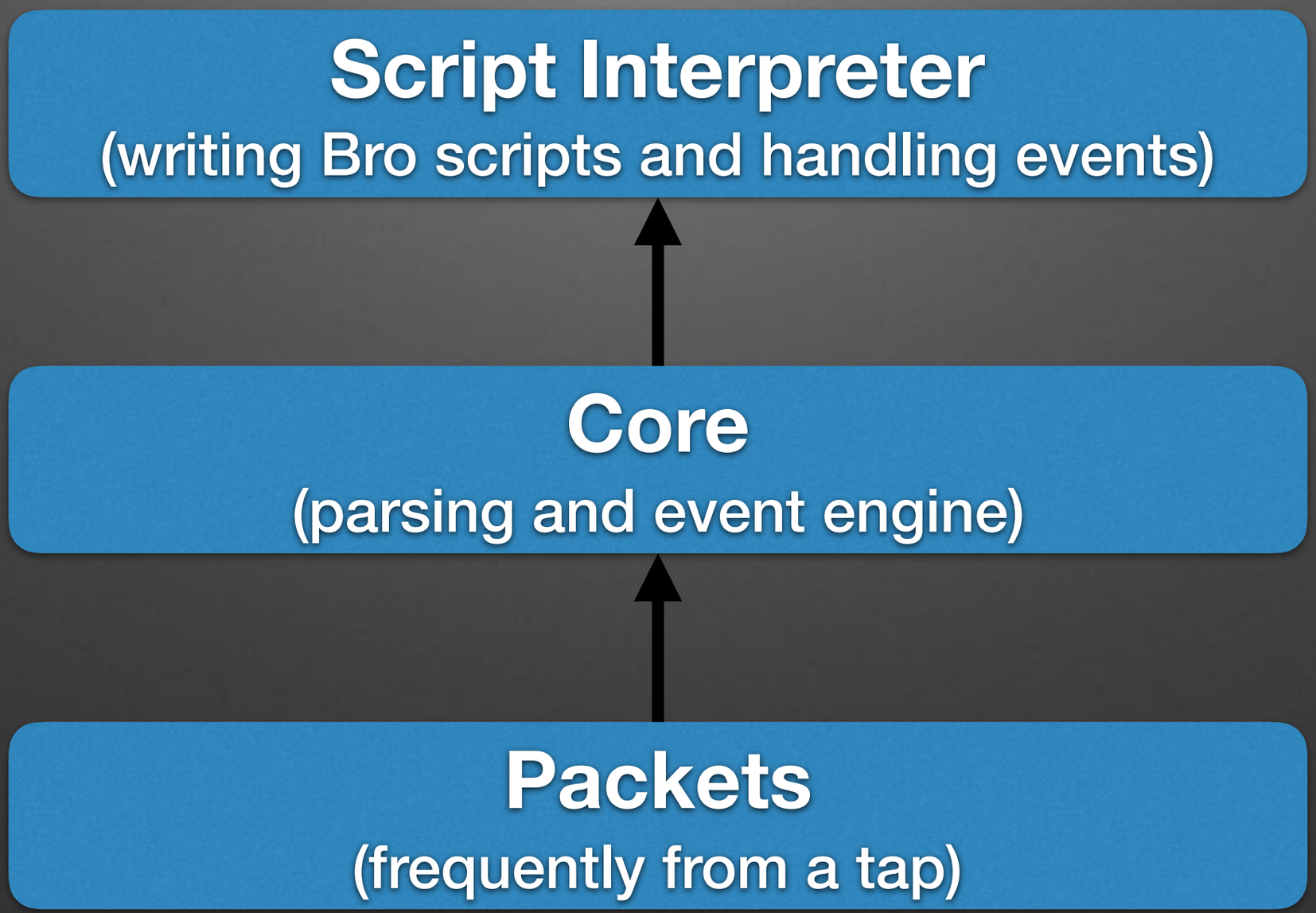
What is Bro?

```
event bro_init()
    {
    print "Hello World!";
    }
```

What is Bro?

- 23 year old network traffic analysis software written in C++
- Heavily used in academic research and by operational security teams.
- Domain specific programming language for event analysis through time.
  - With a built in source of events from network traffic!

How does it work?



Domain Specific Features

- Event driven architecture (more on this in a second)
- Network oriented data types (IP addresses, subnets, ports)
- Built in state expiration
  - global my_data: set[string] &write_expire=1hr;
- Built in network protocol parsing, supported protocols
  - IP/TCP/UDP/HTTP/SMTP/SMB/SSL/FTP/DCE-RPC/DHCP/Modbus/Radius/RDP/RFB/SIP/NTLM/Syslog/DNS/SSH

Another Domain Specific Feature - Error handling

- Runtime errors don't take down Bro (like accessing an unset field in a record).
  - Generate a "Reporter" event and unwind the current stack.
  - This can sometimes cause memory leaks but people tend to have Bro doing lots of tasks and they'd rather have the memory leak than have Bro shutdown.

Event driven architecture

```
event connection_established(c: connection)
    {
    print fmt("%s established a connection to %s:%d", c$id$orig_h, c$id$resp_h, c$id$resp_p);
    }

event connection_state_remove(c: connection)
    {
    print fmt("%s ended a connection to %s:%d", c$id$orig_h, c$id$resp_h, c$id$resp_p);
    }
```

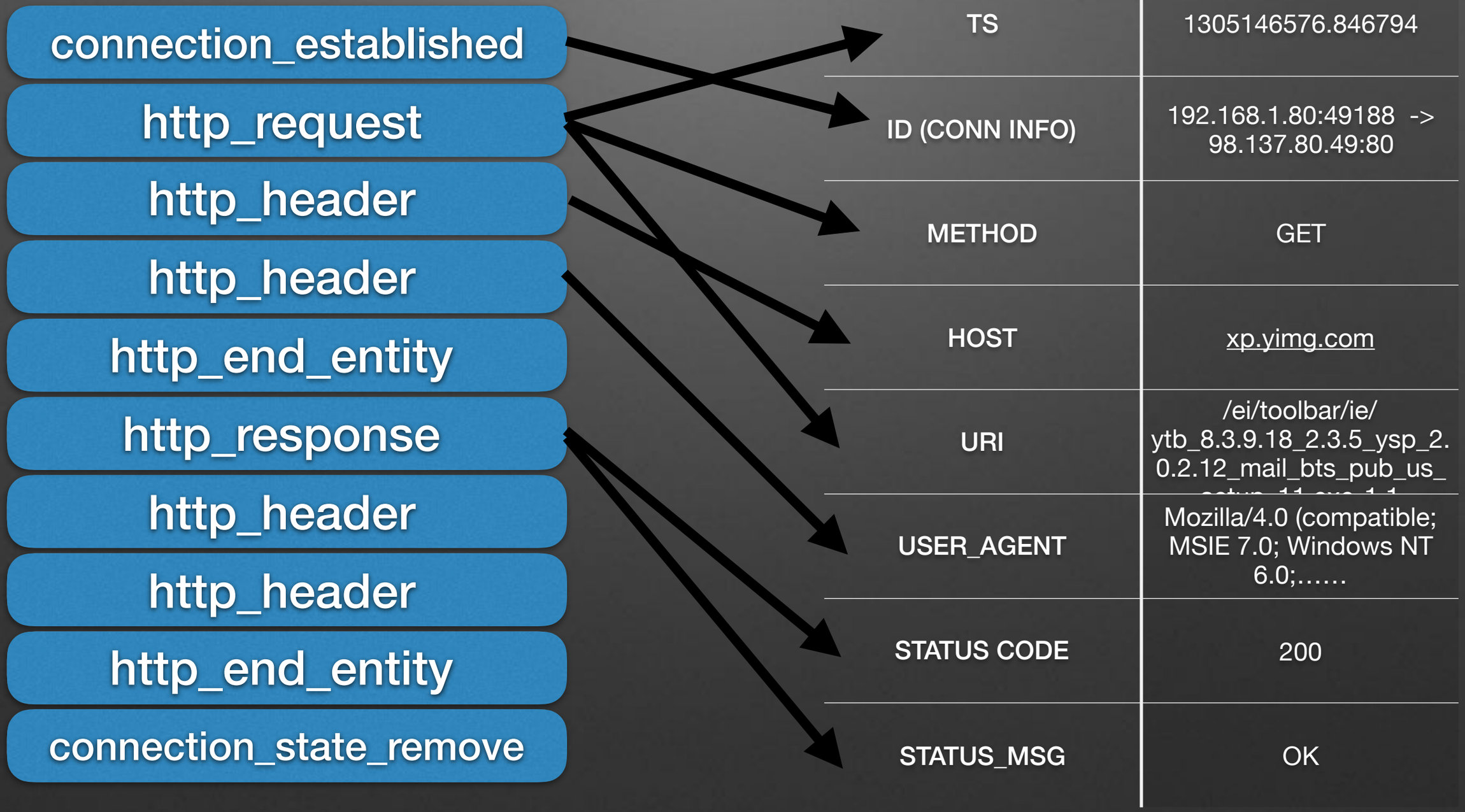
```
192.168.1.80 established a connection to 98.137.80.32:80
192.168.1.80 established a connection to 98.138.6.52:80
192.168.1.80 established a connection to 205.177.95.54:80
192.168.1.80 established a connection to 64.4.52.169:80
192.168.1.80 ended a connection to 192.168.1.1:53
192.168.1.80 established a connection to 66.196.80.71:80
192.168.1.80 established a connection to 216.34.207.62:443
```

Slight change to that last one

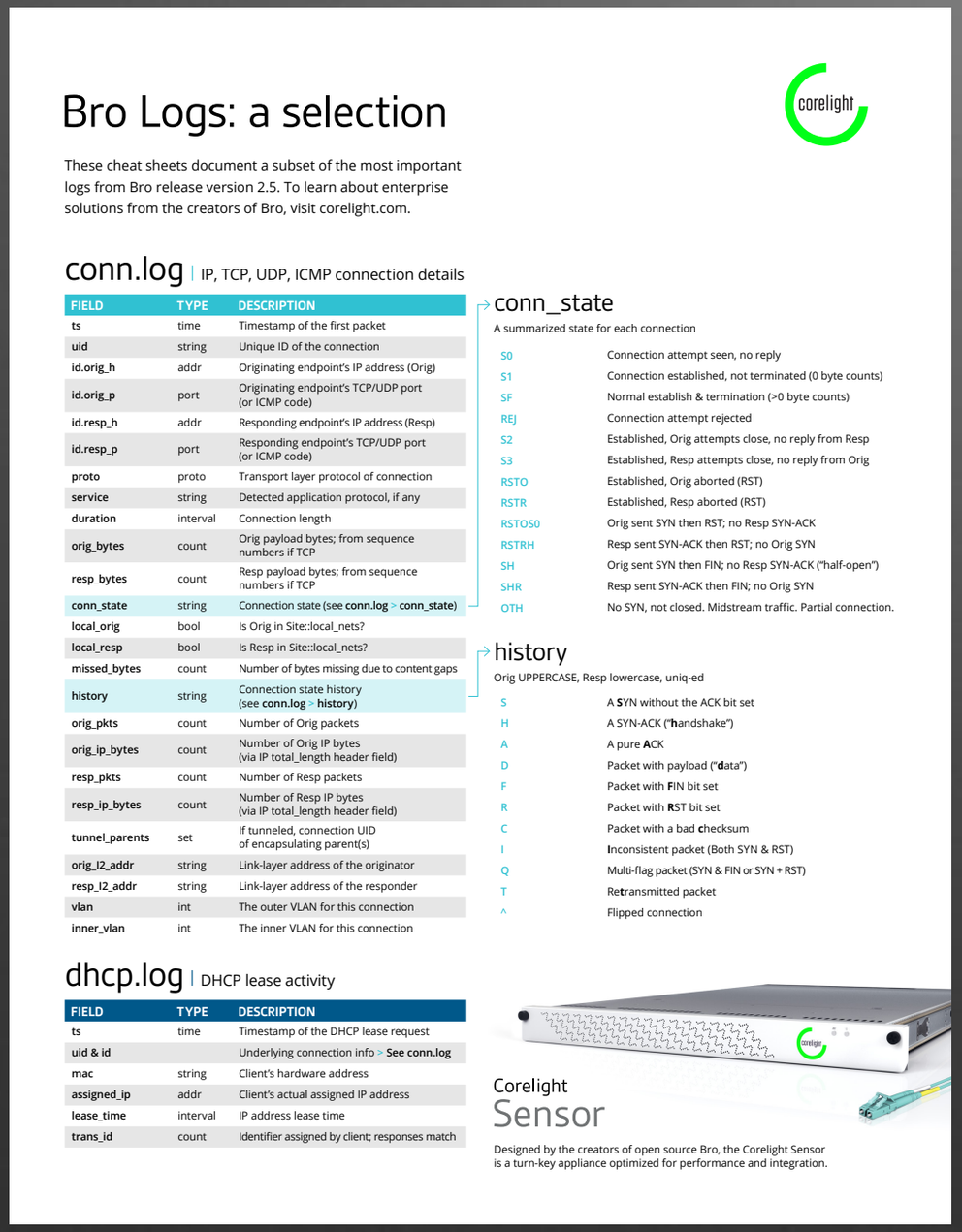
```
event connection_established(c: connection)
    {
    if ( c$id$resp_p == 80/tcp )
        print fmt("%s established a connection to %s:%d", c$id$orig_h, c$id$resp_h, c$id$resp_p);
    }

event http_request(c: connection, method: string, original_URI: string,
                   unescaped_URI: string, version: string) &priority=5
    {
    print fmt("%s requested %s from %s", c$id$orig_h, original_URI, c$id$resp_h);
    }
```

- 192.168.1.80 established a connection to 74.125.161.101:80
- 192.168.1.80 requested /edgedl/toolbar/components/GoogleToolbar_64_D7A51B83F435BE9A.dll.lz from 74.125.161.101
- 192.168.1.80 requested /search?sourceid=navclient&ie=UTF-8&q=msn+toolbar from 74.125.225.18
- 192.168.1.80 established a connection to 74.125.95.101:80
- 192.168.1.80 requested /toolbar/ie8/accelerators/intl/en/manifest.txt from 74.125.95.101

If we can do that, could we put together a log?!



Apply the same approach to other protocols



Bro Logs: a selection

These cheat sheets document a subset of the most important logs from Bro release version 2.5. To learn about enterprise solutions from the creators of Bro, visit corelight.com.

conn.log | IP, TCP, UDP, ICMP connection details

FIELD	TYPE	DESCRIPTION
ts	time	Timestamp of the first packet
uid	string	Unique ID of the connection
id.orig_h	addr	Originating endpoint's IP address (Orig)
id.orig_p	port	Originating endpoint's TCP/UDP port (or ICMP code)
id.resp_h	addr	Responding endpoint's IP address (Resp)
id.resp_p	port	Responding endpoint's TCP/UDP port (or ICMP code)
proto	proto	Transport layer protocol of connection
service	string	Detected application protocol, if any
duration	interval	Connection length
orig_bytes	count	Orig payload bytes; from sequence numbers if TCP
resp_bytes	count	Resp payload bytes; from sequence numbers if TCP
conn_state	string	Connection state (see **conn.log** > **conn_state**)
local_orig	bool	Is Orig in Site::local_nets?
local_resp	bool	Is Resp in Site::local_nets?
missed_bytes	count	Number of bytes missing due to content gaps
history	string	Connection state history (see **conn.log** > **history**)
orig_pkts	count	Number of Orig packets
orig_ip_bytes	count	Number of Orig IP bytes (via IP total_length header field)
resp_pkts	count	Number of Resp packets
resp_ip_bytes	count	Number of Resp IP bytes (via IP total_length header field)
tunnel_parents	set	If tunneled, connection UID of encapsulating parent(s)
orig_l2_addr	string	Link-layer address of the originator
resp_l2_addr	string	Link-layer address of the responder
vlan	int	The outer VLAN for this connection
inner_vlan	int	The inner VLAN for this connection

conn_state

A summarized state for each connection

S0	Connection attempt seen, no reply
S1	Connection established, not terminated (0 byte counts)
SF	Normal establish & termination (>0 byte counts)
REJ	Connection attempt rejected
S2	Established, Orig attempts close, no reply from Resp
S3	Established, Resp attempts close, no reply from Orig
RSTO	Established, Orig aborted (RST)
RSTR	Established, Resp aborted (RST)
RSTOS0	Orig sent SYN then RST; no Resp SYN-ACK
RSTRH	Resp sent SYN-ACK then RST; no Orig SYN
SH	Orig sent SYN then FIN; no Resp SYN-ACK ("half-open")
SHR	Resp sent SYN-ACK then FIN; no Orig SYN
OTH	No SYN, not closed. Midstream traffic. Partial connection.

history

Orig UPPERCASE, Resp lowercase, uniq-ed

S	A **S**YN without the ACK bit set
H	A SYN-ACK ("**h**andshake")
A	A pure **A**CK
D	Packet with payload ("**d**ata")
F	Packet with **F**IN bit set
R	Packet with **R**ST bit set
C	Packet with a bad **c**hecksum
I	**I**nconsistent packet (Both SYN & RST)
Q	Multi-flag packet (SYN & FIN or SYN + RST)
T	Re**t**ransmitted packet
^	Flipped connection

dhcp.log | DHCP lease activity

FIELD	TYPE	DESCRIPTION
ts	time	Timestamp of the DHCP lease request
uid & id		Underlying connection info > **See conn.log**
mac	string	Client's hardware address
assigned_ip	addr	Client's actual assigned IP address
lease_time	interval	IP address lease time
trans_id	count	Identifier assigned by client; responses match

Corelight
Sensor

Designed by the creators of open source Bro, the Corelight Sensor is a turn-key appliance optimized for performance and integration.

dns.log | DNS query/response details

FIELD	TYPE	DESCRIPTION
ts	time	Timestamp of the DNS request
uid & id		Underlying connection info > See conn.log
proto	proto	Protocol of DNS transaction—TCP or UDP
trans_id	count	16 bit identifier assigned by DNS client; responses match
rtt	interval	Round trip time for the query and response
query	string	Domain name subject of the query
qclass	count	Value specifying the query class
qclass_name	string	Descriptive name of the query class (e.g.,C_INTERNET)
qtype	count	Value specifying the query type
qtype_name	string	Descriptive name of the query type (e.g.,A, AAAA, PTR)
rcode	count	Response code value in the DNS response
rcode_name	string	Descriptive name of response code (e.g.,NXDOMAIN, NODATA)
AA	bool	Authoritative answer: T = server is authoritative for the query
TC	bool	Truncation: T = the message was truncated
RD	bool	Recursion desired: T = recursive lookup of query requested
RA	bool	Recursion available: T = server supports recursive queries
Z	count	Reserved field, should be zero in all queries and responses
answers	vector	List of resource descriptions in answer to the query
TTLs	vector	Caching intervals of the answers
rejected	bool	Whether DNS query was rejected by server
auth*	set	Authoritative response for the query
addl*	set	Additional responses for the query

*If policy/protocols/dns/auth-addl.bro is loaded

files.log | File analysis results

FIELD	TYPE	DESCRIPTION
ts	time	Timestamp when file was first seen
fuid	string	Unique identifier for a single file
tx_hosts	set	Host(s) that sourced the data
rx_hosts	set	Host(s) that received the data
conn_uids	set	Connection UID(s) over which file transferred
source	string	An identification of the source of the file data
depth	count	Depth of file related to source (e.g.,HTTP request depth)
analyzers	set	Set of analyzers attached during file analysis
mime_type	string	File type, as determined by Bro's signatures
filename	string	Filename, if available from source analysis
duration	interval	The duration that the file was analyzed for
local_orig	bool	Did the data originate locally?
is_orig	bool	Was the file sent by the originator?
seen_bytes	count	Number of bytes provided to file analysis engine
total_bytes	count	Total number of bytes that should comprise the file
missing_bytes	count	Number of bytes in file stream missed
overflow_bytes	count	Out of sequence bytes in the stream due to overflow
timedout	bool	If the file analysis timed out at least once
parent_fuid	string	Container file ID this was extracted from
md5/sha1	string	MD5/SHA1 hash of the file
extracted	string	Local filename of extracted file if enabled
entropy	double	Information density of the file contents

ftp.log | FTP request/reply details

FIELD	TYPE	DESCRIPTION
ts	time	Timestamp of the FTP command
uid & id		Underlying connection info > See conn.log
user	string	Username for the FTP session
password	string	Password for the FTP session
command	string	Command issued by the client
arg	string	Any command arguments
mime_type	string	File type if there's a file transfer
file_size	count	Size of transferred file
reply_code	count	Reply code from server in response to the command
reply_msg	string	Reply message from server in response to the command
data_channel	record	Information about the data channel (orig, resp, inclusive)
fuid	string	File unique ID

http.log | HTTP request/reply details

FIELD	TYPE	DESCRIPTION
ts	time	Timestamp of the HTTP request
uid & id		Underlying connection info > See conn.log
trans_depth	count	Pipelined depth into the connection
method	string	HTTP Request verb: GET, POST, HEAD, etc
host	string	Value of the "Host" header
uri	string	URI used in the request
referrer	string	Value of the "Referer" header
user_agent	string	Value of the "User-Agent" header
request_body_len	count	Uncompressed content size of Orig data
response_body_len	count	Uncompressed content size of Resp data
status_code	count	Status code returned by the server
status_msg	string	Status message returned by the server
info_code	count	Last seen 1xx info reply code by server
info_msg	string	Last seen 1xx info reply message by server
tags	set	Indicators of various attributes discovered
username	string	Username if basic-auth is performed
password	string	Password if basic-auth is performed
proxied	set	Headers indicative of a proxied request
orig_fuids	vector	File unique IDs from Orig
orig_filenames	vector	File names from Orig
orig_mime_types	vector	File types from Orig
resp_fuids	vector	File unique IDs from Resp
resp_filenames	vector	File names from Resp
resp_mime_types	vector	File types from Resp

CORELIGHT, INC. | INFO@CORELIGHT.COM

https://github.com/corelight/bro-cheatsheets

Why any of this?

- Know what's happening on your network.
  - Everyones environment gets the same logs!
- Detect intrusions

Find Stuff!

SSH Bruteforcing

- Guess if a connection is successful
  - our SSH analyzer will do this for you and generates an event with the heuristic
- Watch for too many failed connections in a short period.

SSH Geofencing

- Watch for probable successful SSH connections
- Do a geoip lookup for the non-local address
- If it’s in a particular set of countries, let me know!

SSL fingerprinting

- Salesforce has a script that generates fingerprints based on the various settings sent while negotiating an SSL connection.
- There are some fingerprints they include to detect various pieces of software.

Conn Burst

- Detect connections that are moving a lot of data quickly.
- It may indicate connections that can be ignored.
- Yet another tiny signal.

Intel (intelligence) framework

- This is a built in part of Bro.
- Load IOCs (indicators of compromise) into Bro
- Scripts will feed data into the intel framework and check it.
- You get a log that says what was found, when it was found, where it was found, and any meta data about the intelligence item (feed it was from, url for more information, etc)

Credit Card Exposure

- Watch for credit card candidates in HTTP and SMTP bodies.
- Take the candidates and validate them with the Luhn checksum
- Do a local lookup in a table of IINs (issuer identification numbers)
- Log all of the information including an excerpt around the CC and redact the number by default.

SSN Exposure

- Grab candidate SSNs out of HTTP and SMTP
- Look for either a defined set of state prefixes or a particular value in a set.
- Log it with some context.

Bro Package Manager

- Everyone has a common platform.
- Analysis and logging scripts can be shared between institutions.
- All of the previous scripts are either built into Bro or available through the Bro Package Manager.

Thanks!

We’re hiring at Corelight