15.1.2. Eve JSON Format

Example:

```
{
  "timestamp": "2017-04-07T22:24:37.251547+0100",
  "flow_id": 586497171462735,
  "pcap_cnt": 53381,
  "event_type": "alert",
  "src_ip": "192.168.2.14",
  "src_port": 50096,
  "dest_ip": "209.53.113.5",
  "dest_port": 80,
  "proto": "TCP",
  "metadata": {
    "flowbits": [
      "http.dottedquadhost"
    ]
  },
  "tx_id": 4,
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 2018358,
    "rev": 10,
    "signature": "ET HUNTING GENERIC SUSPICIOUS POST to Dotted Quad with Fake Browser 1",
    "category": "Potentially Bad Traffic",
    "severity": 2
  },
  "app_proto": "http"
}
```

15.1.2.1. Common Section

All the JSON log types share a common structure:

```
{"timestamp":"2009-11-24T21:27:09.534255","flow_id":ID_NUMBER, "event_type":"TYPE",
...tuple... ,"TYPE":{ ... type specific content ... }}
```

15.1.2.1.1. Field: flow_id

Correlates the network protocol, flow logs EVE data and any evidence that Suricata has logged to an `alert` event and that alert's metadata, as well as to `fileinfo` /file transaction and anomaly logs, if available. The same correlation and logs are produced regardless if there is an alert, for any session/flow.

The ability to correlate EVE logs belonging to a specific session/flow was introduced in 2014 (see commit f1185d051c21).

Further below, you can see several examples of events logged by Suricata: an alert for an `HTTP` rule, `fileinfo`, http, anomaly, and flow events, all easily correlated using the `flow_id` EVE field:

```
$ jq 'select(.flow_id==1676750115612680)' eve.json
```

Event type: `alert`:

```
{
  "timestamp": "2023-09-18T06:13:41.532140+0000",
  "flow_id": 1676750115612680,
  "pcap_cnt": 130,
  "event_type": "alert",
  "src_ip": "142.11.240.191",
  "src_port": 35361,
  "dest_ip": "192.168.100.237",
  "dest_port": 49175,
  "proto": "TCP",
  "pkt_src": "wire/pcap",
  "ether": {
    "src_mac": "52:54:00:36:3e:ff",
    "dest_mac": "12:a9:86:6c:77:de"
  },
  "tx_id": 1,
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 2045001,
    "rev": 1,
    "signature": "ET ATTACK_RESPONSE Win32/LeftHook Stealer Browser Extension Config
Inbound",
    "category": "A Network Trojan was detected",
    "severity": 1,
    "metadata": {
      "affected_product": [
        "Windows_XP_Vista_7_8_10_Server_32_64_Bit"
      ],
      "attack_target": [
        "Client_Endpoint"
      ],
      "created_at": [
        "2023_04_17"
      ],
      "deployment": [
        "Perimeter"
      ],
      "former_category": [
        "ATTACK_RESPONSE"
      ],
      "signature_severity": [
        "Major"
      ],
      "updated_at": [
        "2023_04_18"
      ]
    }
  },
  "http": {
    "hostname": "142.11.240.191",
    "http_port": 35361,
    "url": "/",
    "http_content_type": "text/xml",
    "http_method": "POST",
    "protocol": "HTTP/1.1",
    "status": 200,
    "length": 5362
  },
```

```
  "files": [
    {
      "filename": "/",
      "gaps": false,
      "state": "CLOSED",
      "stored": false,
      "size": 5362,
      "tx_id": 1
    }
  ],
  "app_proto": "http",
  "direction": "to_client",
  "flow": {
    "pkts_toserver": 13,
    "pkts_toclient": 12,
    "bytes_toserver": 1616,
    "bytes_toclient": 8044,
    "start": "2023-09-18T06:13:33.324862+0000",
    "src_ip": "192.168.100.237",
    "dest_ip": "142.11.240.191",
    "src_port": 49175,
    "dest_port": 35361
  }
}
```


Event type: `fileinfo`:

```
{
  "timestamp": "2023-09-18T06:13:33.903924+0000",
  "flow_id": 1676750115612680,
  "pcap_cnt": 70,
  "event_type": "fileinfo",
  "src_ip": "192.168.100.237",
  "src_port": 49175,
  "dest_ip": "142.11.240.191",
  "dest_port": 35361,
  "proto": "TCP",
  "pkt_src": "wire/pcap",
  "ether": {
    "src_mac": "12:a9:86:6c:77:de",
    "dest_mac": "52:54:00:36:3e:ff"
  },
  "http": {
    "hostname": "142.11.240.191",
    "http_port": 35361,
    "url": "/",
    "http_content_type": "text/xml",
    "http_method": "POST",
    "protocol": "HTTP/1.1",
    "status": 200,
    "length": 212
  },
  "app_proto": "http",
  "fileinfo": {
    "filename": "/",
    "gaps": false,
    "state": "CLOSED",
    "stored": false,
    "size": 137,
    "tx_id": 0
  }
}
```


Event type: `HTTP`:

```
{
  "timestamp": "2023-09-18T06:13:33.903924+0000",
  "flow_id": 1676750115612680,
  "pcap_cnt": 70,
  "event_type": "http",
  "src_ip": "192.168.100.237",
  "src_port": 49175,
  "dest_ip": "142.11.240.191",
  "dest_port": 35361,
  "proto": "TCP",
  "pkt_src": "wire/pcap",
  "ether": {
    "src_mac": "12:a9:86:6c:77:de",
    "dest_mac": "52:54:00:36:3e:ff"
  },
  "tx_id": 0,
  "http": {
    "hostname": "142.11.240.191",
    "http_port": 35361,
    "url": "/",
    "http_content_type": "text/xml",
    "http_method": "POST",
    "protocol": "HTTP/1.1",
    "status": 200,
    "length": 212,
    "request_headers": [
      {
        "name": "Content-Type",
        "value": "text/xml; charset=utf-8"
      },
      {
        "name": "SOAPAction",
        "value": "\"http://tempuri.org/Endpoint/CheckConnect\""
      },
      {
        "name": "Host",
        "value": "142.11.240.191:35361"
      },
      {
        "name": "Content-Length",
        "value": "137"
      },
      {
        "name": "Expect",
        "value": "100-continue"
      },
      {
        "name": "Accept-Encoding",
        "value": "gzip, deflate"
      },
      {
        "name": "Connection",
        "value": "Keep-Alive"
      }
    ],
    "response_headers": [
      {
        "name": "Content-Length",
        "value": "212"
```

```
      },
      {
        "name": "Content-Type",
        "value": "text/xml; charset=utf-8"
      },
      {
        "name": "Server",
        "value": "Microsoft-HTTPAPI/2.0"
      },
      {
        "name": "Date",
        "value": "Mon, 18 Sep 2023 06:13:33 GMT"
      }
    ]
  }
}
```


Event type: `anomaly`:


```
{
  "timestamp": "2023-09-18T06:13:58.882971+0000",
  "flow_id": 1676750115612680,
  "pcap_cnt": 2878,
  "event_type": "anomaly",
  "src_ip": "192.168.100.237",
  "src_port": 49175,
  "dest_ip": "142.11.240.191",
  "dest_port": 35361,
  "proto": "TCP",
  "pkt_src": "wire/pcap",
  "ether": {
    "src_mac": "12:a9:86:6c:77:de",
    "dest_mac": "52:54:00:36:3e:ff"
  },
  "tx_id": 3,
  "anomaly": {
    "app_proto": "http",
    "type": "applayer",
    "event": "UNABLE_TO_MATCH_RESPONSE_TO_REQUEST",
    "layer": "proto_parser"
  }
}
```


Event type: `flow`:

```
{
  "timestamp": "2023-09-18T06:13:21.216460+0000",
  "flow_id": 1676750115612680,
  "event_type": "flow",
  "src_ip": "192.168.100.237",
  "src_port": 49175,
  "dest_ip": "142.11.240.191",
  "dest_port": 35361,
  "proto": "TCP",
  "app_proto": "http",
  "flow": {
    "pkts_toserver": 3869,
    "pkts_toclient": 1523,
    "bytes_toserver": 3536402,
    "bytes_toclient": 94102,
    "start": "2023-09-18T06:13:33.324862+0000",
    "end": "2023-09-18T06:14:13.752399+0000",
    "age": 40,
    "state": "closed",
    "reason": "shutdown",
    "alerted": true,
    "exception_policy": [
      {
        "target": "stream_midstream",
        "policy": "ignore"
      }
    ]
  },
  "ether": {
    "dest_macs": [
      "52:54:00:36:3e:ff"
    ],
    "src_macs": [
      "12:a9:86:6c:77:de"
    ]
  },
  "tcp": {
    "tcp_flags": "1e",
    "tcp_flags_ts": "1e",
    "tcp_flags_tc": "1a",
    "syn": true,
    "rst": true,
    "psh": true,
    "ack": true,
    "state": "closed",
    "ts_max_regions": 1,
    "tc_max_regions": 1
  }
}
```


Note

It is possible to have even more detailed alert records, by enabling for instance body, or alert metadata (alert output).

Examples come from pcap found at https://app.any.run/tasks/ce7ca983-9e4b-4251-a7c3-fefa3da02ebe/.

15.1.2.1.2. Event types

The common part has a field "event_type" to indicate the log type.

```
"event_type":"TYPE"
```

When an application layer protocol event is detected, the common section will have an `app_proto` field.

```
"app_proto": "http"
```

15.1.2.1.3. PCAP fields

If Suricata is processing a pcap file, additional fields are added:

```
"pcap_cnt": 123
```

`pcap_cnt` contains the packet number in the pcap. This can be used to look up a packet in Wireshark for example.

```
"pcap_filename":"/path/to/file.pcap"
```

`pcap_filename` contains the file name and location of the pcap that generated the event.

Note

the pcap fields are only available on "real" packets, and are omitted from internal "pseudo" packets such as flow timeout packets.

15.1.2.2. Event type: Alert

This field contains data about a signature that matched, such as `signature_id` (`sid` in the rule) and the `signature` (`msg` in the rule).

It can also contain information about Source and Target of the attack in the `alert.source` and `alert.target` field if target keyword is used in the signature.

In firewall mode, the `alert.engine` field identifies which rule engine generated the alert: `fw` for firewall rules and `td` for threat detect rules. This field is omitted outside of firewall mode.

This event will also have the `pcap_cnt` field, when running in pcap mode, to indicate which packet triggered the signature.

```
"alert": {
  "action": "allowed",
  "gid": 1,
  "signature_id": 2024056,
  "rev": 4,
  "signature": "ET MALWARE Win32/CryptFile2 / Revenge Ransomware Checkin M3",
  "category": "Malware Command and Control Activity Detected",
  "severity": 1,
  "metadata": {
    "affected_product": [
      "Windows_XP_Vista_7_8_10_Server_32_64_Bit"
    ],
    "attack_target": [
      "Client_Endpoint"
    ],
    "created_at": [
      "2017_03_15"
    ],
    "deployment": [
      "Perimeter"
    ],
    "former_category": [
      "MALWARE"
    ],
    "malware_family": [
      "CryptFile2"
    ],
    "performance_impact": [
      "Moderate"
    ],
    "signature_severity": [
      "Major"
    ],
    "updated_at": [
      "2020_08_04"
    ]
  }
},
```

15.1.2.2.1. Action field

Possible values: "allowed" and "blocked".

Example:

```
"action":"allowed"
```

Action is set to "allowed" unless a rule used the "drop" action and Suricata is in IPS mode, in which case it is set to "blocked". It is also set to "blocked" when the rule used the "reject" action. It is important to note that this does not necessarily indicate the final verdict for a given packet or flow, since one packet may match on several rules.

15.1.2.2.2. Verdict

An object containing info on the final action that will be applied to a given packet, based on all the signatures triggered by it and other possible events (e.g., a flow drop). For that reason, it is possible for an alert with an action `allowed` to have a verdict `drop`, in IPS mode, for instance, if that packet was dropped due to a different alert.

- Action: `alert`, `pass`, `drop` (this latter only occurs in IPS mode)
- Reject-target: `to_server`, `to_client`, `both` (only occurs for 'reject' rules)
- Reject: an array of strings with possible reject types: `tcp-reset`, `icmp-prohib` (only occurs for 'reject' rules)

Example:

```
"verdict": {
   "action": "drop",
   "reject_target": "to_client",
   "reject": "[icmp-prohib]"
 }
```

15.1.2.2.3. Pcap Field

If pcap log capture is active in *multi* mode, a *capture_file* key will be added to the event with value being the full path of the pcap file where the corresponding packets have been extracted.

15.1.2.3. Event type: Anomaly

Events with type "anomaly" report unexpected conditions such as truncated packets, packets with invalid values, events that render the packet invalid for further processing or unexpected behaviors.

Networks which experience high occurrences of anomalies may experience packet processing degradation when anomaly logging is enabled.

15.1.2.3.1. Fields

- "type": Either "decode", "stream" or "applayer". In rare cases, type will be "unknown". When this occurs, an additional field named "code" will be present. Events with typ
detected by the application layer parsers.
- "event" The name of the anomalous event. Events of type "decode" are prefixed with "decoder"; events of type "stream" are prefixed with "stream".

- "code" If "type" is "unknown", than "code" contains the unrecognized event code. Otherwise, this field is not present.

The following field is included when "type" has the value "applayer":

- "layer" Indicates the handling layer that detected the event. This will be "proto_parser" (protocol parser), "proto_detect" (protocol detection) or "parser."

When `packethdr` is enabled, the first 32 bytes of the packet are included as a byte64-encoded blob in the main part of record. This applies to events of "type" "packet" or "stream" only.

15.1.2.3.2. Examples

```
"anomaly": {
  "type": "decode",
  "event": "decoder.icmpv4.unknown_type"
}

"anomaly": {
  "type": "decode",
  "event": "decoder.udp.pkt_too_small"
}

"anomaly": {
  "type": "decode",
  "event": "decoder.ipv4.wrong_ip_version"
}

"anomaly": {
  "type": "stream",
  "event": "stream.pkt_invalid_timestamp"
}

{
  "timestamp": "1969-12-31T16:04:21.000000-0800",
  "pcap_cnt": 9262,
  "event_type": "anomaly",
  "src_ip": "208.21.2.184",
  "src_port": 0,
  "dest_ip": "10.1.1.99",
  "dest_port": 0,
  "proto": "UDP",
  "packet": "////////AQEBAQEBCABFAAA8xZ5AAP8R1+DQFQK4CgE=",
  "packet_info": {
    "linktype": 1
  },
  "anomaly": {
    "type": "decode",
    "event": "decoder.udp.pkt_too_small"
  }
}

{
  "timestamp": "2016-01-11T05:10:54.612110-0800",
  "flow_id": 412547343494194,
  "pcap_cnt": 1391293,
  "event_type": "anomaly",
  "src_ip": "192.168.122.149",
  "src_port": 49324,
  "dest_ip": "69.195.71.174",
  "dest_port": 443,
  "proto": "TCP",
  "app_proto": "tls",
  "anomaly": {
    "type": "applayer",
    "event": "APPLAYER_DETECT_PROTOCOL_ONLY_ONE_DIRECTION",
    "layer": "proto_detect"
  }
}
```

```
{
  "timestamp": "2016-01-11T05:10:52.828802-0800",
  "flow_id": 201217772575257,
  "pcap_cnt": 1391281,
  "event_type": "anomaly",
  "src_ip": "192.168.122.149",
  "src_port": 49323,
  "dest_ip": "69.195.71.174",
  "dest_port": 443,
  "proto": "TCP",
  "tx_id": 0,
  "app_proto": "tls",
  "anomaly": {
    "type": "applayer",
    "event": "INVALID_RECORD_TYPE",
    "layer": "proto_parser"
  }
}
```


15.1.2.4. Event type: fileinfo

Note that the checksum values for `md5`, `sha1`, and `sha256` are available when

- The command line option `disable-hashing` was not used
- There are no gaps (areas missing)

15.1.2.4.1. Fields

- "end: The offset of the last byte captured
- "file_id": Integer value representing the id of a file that has been stored
- "filename": Name of the file as observed in network traffic
- "gaps": Boolean value indicating if there were gaps in the file
- "magic": [optional, requires libmagic] The magic value for the file
- "md5": Iff closed, md5 sum
- "sha1": Iff closed, sha1 sum
- "sha256": The sha256 value for the file, if available
- "sid": One or more signature ids that triggered a *filestore*
- "size": The observed size of the file, in bytes
- "start": The offset of the first byte captured
- "state": The state of the file when the record is written
- "stored": Boolean value indicating whether the file has been stored
- "storing": Boolean value indicating whether the file is in the process of being
  when not yet stored

- "tx_id": The transaction id in effect

15.1.2.4.1.1. Offset values

This example shows the offset values from a `fileinfo` event -- note the `http` content range *start* and *end* value are replicated in the `fileinfo` fields:

```
http.content_range.raw: bytes 500-1000/146515
http.content_range.start: 500
http.content_range.end: 1000
http.content_range.size: 146515
fileinfo.start: 500
fileinfo.end: 1000
```

15.1.2.5. Event type: HTTP

15.1.2.5.1. Fields

- "hostname": The hostname this HTTP event is attributed to
- "url": URL at the hostname that was accessed
- "http_user_agent": The user-agent of the software that was used
- "http_content_type": The type of data returned (ex: application/x-gzip)
- "cookie"

In addition to these fields, if the extended logging is enabled in the suricata.yaml file the following fields are (can) also included:

- "length": The content size of the HTTP body
- "status": HTTP status code
- "protocol": Protocol / Version of HTTP (ex: HTTP/1.1)
- "http_method": The HTTP method (ex: GET, POST, HEAD)
- "http_refer": The referer for this action

In addition to the extended logging fields one can also choose to enable/add from more than 50 additional custom logging HTTP fields enabled in the suricata.yaml file. The additional fields can be enabled as following:

```
- eve-log:
    enabled: yes
    type: file #file|syslog|unix_dgram|unix_stream
    filename: eve.json
    # the following are valid when type: syslog above
    #identity: "suricata"
    #facility: local5
    #level: Info ## possible levels: Emergency, Alert, Critical,
                 ## Error, Warning, Notice, Info, Debug
    types:
      - alert
      - http:
          extended: yes     # enable this for extended logging information
          # custom allows additional http fields to be included in eve-log
          # the example below adds three additional fields when uncommented
          #custom: [Accept-Encoding, Accept-Language, Authorization]
          custom: [accept, accept-charset, accept-encoding, accept-language,
          accept-datetime, authorization, cache-control, cookie, from,
          max-forwards, origin, pragma, proxy-authorization, range, te, via,
          x-requested-with, dnt, x-forwarded-proto, accept-range, age,
          allow, connection, content-encoding, content-language,
          content-length, content-location, content-md5, content-range,
          content-type, date, etags, expires, last-modified, link, location,
          proxy-authenticate, referer, refresh, retry-after, server,
          set-cookie, trailer, transfer-encoding, upgrade, vary, warning,
          www-authenticate, x-flash-version, x-authenticated-user]
```

The benefits here of using the extended logging is to see if this action for example was a POST or perhaps if a download of an executable actually returned any bytes.

It is also possible to dump every header for HTTP requests/responses or both via the keyword `dump-all-headers`.

15.1.2.5.2. Examples

Event with non-extended logging:

```
"http": {
    "hostname": "www.digip.org",
    "url" :"\/jansson\/releases\/jansson-2.6.tar.gz",
    "http_user_agent": "<User-Agent>",
    "http_content_type": "application\/x-gzip"
}
```

In case the hostname shows a port number, such as in case there is a header "H
www.test.org:1337":

```
"http": {
    "http_port": 1337,
    "hostname": "www.test.org",
    "url" :"\/this\/is\/test.tar.gz",
    "http_user_agent": "<User-Agent>",
    "http_content_type": "application\/x-gzip"
}
```

Event with extended logging:

```
"http": {
    "hostname": "direkte.vg.no",
    "url":".....",
    "http_user_agent": "<User-Agent>",
    "http_content_type": "application\/json",
    "http_refer": "http:\/\/www.vg.no\/",
    "http_method": "GET",
    "protocol": "HTTP\/1.1",
    "status":"200",
    "length":310
}
```

Event with `dump-all-headers` set to "both":

```
"http": {
    "hostname": "test.co.uk",
    "url":"\/test\/file.json",
    "http_user_agent": "<User-Agent>",
    "http_content_type": "application\/json",
    "http_refer": "http:\/\/www.test.com\/",
    "http_method": "GET",
    "protocol": "HTTP\/1.1",
    "status":"200",
    "length":310,
    "request_headers": [
        {
            "name": "User-Agent",
            "value": "Wget/1.13.4 (linux-gnu)"
        },
        {
            "name": "Accept",
            "value": "*/*"
        },
    ],
    "response_headers": [
        {
            "name": "Date",
            "value": "Wed, 25 Mar 2015 15:40:41 GMT"
        },
    ]
}
```

15.1.2.6. Event type: DNS

DNS has 2 logging style that can be used together or independently:

- "detailed": "rrname", "rrtype", "rdata" and "ttl" fields are logged for each answer
- "grouped": answers logged are aggregated by their type (A, AAAA, NS, ...)

If no format is chosen, "detailed" will be used by default.

It will be still possible to use the old DNS logging format, you can control it with "version" option in dns configuration section.

Suricata 8.0.0 introduces version 3 of the DNS logging format. This update unifies the DNS logging style used by `dns` events as well as the `dns` object in `alert` records. See DNS Logging Changes for 8.0 for more details on the changes to logging format.

Note

Suricata 7 style DNS logging can be retained by setting the `version` field to 2, however this will be removed in Suricata 9.

15.1.2.6.1. Fields

Outline of fields seen in the different kinds of DNS events:

- "type": Indicating DNS message type, can be "request" or "response".
- "id": Identifier field
- "version": Indicating DNS logging version in use
- "flags": Indicating DNS answer flag, in hexadecimal (ex: 8180 , please note 0x is not output)
- "qr": Indicating in case of DNS answer flag, Query/Response flag (ex: true if set)
- "aa": Indicating in case of DNS answer flag, Authoritative Answer flag (ex: true if set)
- "tc": Indicating in case of DNS answer flag, Truncation flag (ex: true if set)
- "rd": Indicating in case of DNS answer flag, Recursion Desired flag (ex: true if set)
- "ra": Indicating in case of DNS answer flag, Recursion Available flag (ex: true if set)
- "z": Indicating in case of DNS answer flag, Reserved bit (ex: true if set)
- "rcode": (ex: NOERROR)
- "ttl": Time-To-Live for this resource record
- "queries": A list of query objects
- "answers": A list of answer objects
- "authorities": A list of authority objects
- "additionals": A list of additional objects

More complex DNS record types may log additional fields for resource data:

- "soa": Section containing fields for the SOA (start of authority) record type
  - "mname": Primary name server for this zone
  - "rname": Authority's mailbox
  - "serial": Serial version number
  - "refresh": Refresh interval (seconds)
  - "retry": Retry interval (seconds)
  - "expire": Upper time limit until zone is no longer authoritative (seconds)
  - "minimum": Minimum ttl for records in this zone (seconds)
- "sshfp": section containing fields for the SSHFP (ssh fingerprint) record type
  - "fingerprint": Hex format of the fingerprint (ex: `12:34:56:78:9a:bc:de:...`)
  - "algo": Algorithm number (ex: 1 for RSA, 2 for DSS)
  - "type": Fingerprint type (ex: 1 for SHA-1)
- "srv": section containing fields for the SRV (location of services) record type
  - "target": Domain name of the target host (ex: `foo.bar.baz`)
  - "priority": Target priority (ex: 20)

- "weight": Weight for target selection (ex: 1)
- "port": Port on this target host of this service (ex: 5060)

One can control which RR types are logged by using the "types" field in the suricata.yaml file. If this field is not specified, all RR types are logged. More than 50 values can be specified with this field as shown below:

Configuration:

```
- eve-log:
    enabled: yes
    type: file #file|syslog|unix_dgram|unix_stream
    filename: eve.json
    # the following are valid when type: syslog above
    #identity: "suricata"
    #facility: local5
    #level: Info ## possible levels: Emergency, Alert, Critical,
                 ## Error, Warning, Notice, Info, Debug
    types:
      - alert
      - dns:

        # Logging format. In 8.0 version 3 is the default. Can be
        # set to 2 to keep compatibility with Suricata 7.0.
        # version: 3

        # Control logging of requests and responses:
        # - requests: enable logging of DNS queries
        # - responses: enable logging of DNS answers
        # By default both requests and responses are logged.
        requests: yes
        responses: yes
        # DNS record types to log, based on the query type.
        # Default: all.
        #types: [a, aaaa, cname, mx, ns, ptr, txt]
        types: [a, ns, md, mf, cname, soa, mb, mg, mr, null,
        wks, ptr, hinfo, minfo, mx, txt, rp, afsdb, x25, isdn,
        rt, nsap, nsapptr, sig, key, px, gpos, aaaa, loc, nxt,
        srv, atma, naptr, kx, cert, a6, dname, opt, apl, ds,
        sshfp, ipseckey, rrsig, nsec, dnskey, dhcid, nsec3,
        nsec3param, tlsa, hip, cds, cdnskey, spf, tkey,
        tsig, maila, any, uri]
```

15.1.2.6.2. Examples

Example of a DNS query for the IPv4 address of "twitter.com" (resource record type 'A'):

```
"dns": {
    "version": 3,
    "type": "request",
    "id": 16000,
    "queries": [
      {
        "rrname": "twitter.com",
        "rrtype": "A"
      }
    ]
}
```

Example of a DNS answer with "detailed" format:

```
"dns": {
    "version": 3,
    "type": "answer",
    "id": 45444,
    "flags": "8180",
    "qr": true,
    "rd": true,
    "ra": true,
    "rcode": "NOERROR",
    "queries": [
      {
        "rrname": "www.suricata.io",
        "rrtype": "A"
      }
    ],
    "answers": [
      {
        "rrname": "www.suricata.io",
        "rrtype": "CNAME",
        "ttl": 3324,
        "rdata": "suricata.io"
      },
      {
        "rrname": "suricata.io",
        "rrtype": "A",
        "ttl": 10,
        "rdata": "192.0.78.24"
      },
      {
        "rrname": "suricata.io",
        "rrtype": "A",
        "ttl": 10,
        "rdata": "192.0.78.25"
      }
    ]
}
```


Example of a DNS answer with "grouped" format:

```
"dns": {
    "version": 3,
    "type": "answer",
    "id": 18523,
    "flags": "8180",
    "qr": true,
    "rd": true,
    "ra": true,
    "rcode": "NOERROR",
    "grouped": {
      "A": [
        "192.0.78.24",
        "192.0.78.25"
      ],
      "CNAME": [
        "suricata.io"
      ]
    }
}
```

15.1.2.7. Event type: LLMNR

LLMNR (Link-Local Multicast Name Resolution, RFC 4795) is a protocol that allows hosts on the same local link to perform name resolution. It uses DNS message format, but operates on multicast (224.0.0.252 for IPv4, ff02::1:3 for IPv6) over UDP and TCP port 5355.

Suricata logs both requests and responses as separate events. For UDP, queries are sent to multicast and responses come back as unicast from the responder's IP, resulting in separate flows.

15.1.2.7.1. Fields

Outline of fields seen in LLMNR events:

- "type": Indicating LLMNR message type, can be "request" or "response"
- "id": On-wire LLMNR transaction identifier
- "queries": A list of query objects, each containing "rrname" and "rrtype"
- "answers": A list of answer objects, each containing "rrname", "rrtype" with "rdata"
- "authorities": A list of authority objects
- "additionals": A list of additional objects

15.1.2.7.2. Examples

Example of an LLMNR request for the A record of "hs2011":

```
"llmnr": {
    "type": "request",
    "tx_id": 0,
    "id": 49985,
    "opcode": 0,
    "queries": [
      {
        "rrname": "hs2011",
        "rrtype": "a"
      }
    ]
}
```

Example of an LLMNR response with an A record answer:

```
"llmnr": {
    "type": "response",
    "tx_id": 0,
    "id": 49985,
    "opcode": 0,
    "queries": [
      {
        "rrname": "hs2011",
        "rrtype": "a"
      }
    ],
    "answers": [
      {
        "rrname": "HS2011",
        "a": "192.168.10.101"
      }
    ]
}
```

Example of an LLMNR NXDOMAIN response:

```
"llmnr": {
    "type": "response",
    "tx_id": 0,
    "id": 12345,
    "opcode": 0,
    "rcode": "NXDOMAIN",
    "queries": [
      {
        "rrname": "unknown-host",
        "rrtype": "a"
      }
    ]
}
```

Example of an LLMNR response with NS authority and additional records:

```
"llmnr": {
    "type": "response",
    "tx_id": 0,
    "id": 54321,
    "opcode": 0,
    "queries": [
      {
        "rrname": "server.local",
        "rrtype": "a"
      }
    ],
    "answers": [
      {
        "rrname": "server.local",
        "a": "192.168.1.100"
      }
    ],
    "authorities": [
      {
        "rrname": "local",
        "ns": "ns.local"
      }
    ],
    "additionals": [
      {
        "rrname": "ns.local",
        "a": "192.168.1.200"
      }
    ]
}
```

15.1.2.8. Event type: FTP

15.1.2.8.1. Fields

- "command": The FTP command.
- "command_data": The data accompanying the command.
- "reply": The command reply, which may contain multiple lines, in array format.
- "completion_code": The 3-digit completion code. The first digit indicates whether the response is good, bad or incomplete. This is also in array format and may contain multiple completion codes matching multiple reply lines.
- "dynamic_port": The dynamic port established for subsequent data transfers, when applicable, with a "PORT" or "EPRT" command.
- "mode": The type of FTP connection. Most connections are "passive" but ma
- "reply_received": Indicates whether a response was matched to the comman typical cases, a command may lack a response.

15.1.2.8.2. Examples

Example of regular FTP logging:

```
"ftp": {
  "command": "RETR",
  "command_data": "100KB.zip",
  "reply": [
    "Opening BINARY mode data connection for 100KB.zip (102400 bytes).",
    "Transfer complete."
  ],
  "completion_code": [
    "150",
    "226"
  ],
```

Example showing all fields:

```
"ftp": {
  "command": "EPRT",
  "command_data": "|2|2a01:e34:ee97:b130:8c3e:45ea:5ac6:e301|41813|",
  "reply": [
    "EPRT command successful. Consider using EPSV."
  ],
  "completion_code": [
    "200"
  ],
  "dynamic_port": 41813,
  "mode": "active",
  "reply_received": "yes"
}
```

15.1.2.9. Event type: FTP_DATA

15.1.2.9.1. Fields

- "command": The FTP command associated with the event.
- "filename": The name of the involved file.

15.1.2.9.2. Examples

Example of FTP_DATA logging:

```
"ftp_data": {
  "filename": "temp.txt",
  "command": "RETR"
}
```

15.1.2.10. Event type: TLS

15.1.2.10.1. Fields

- "subject": The subject field from the TLS certificate
- "issuer": The issuer field from the TLS certificate
- "session_resumed": This field has the value of "true" if the TLS session was resumed via a session id. If this field appears, "subject" and "issuer" do not appear, since a TLS certificate is not seen.

If extended logging is enabled the following fields are also included:

- "serial": The serial number of the TLS certificate
- "fingerprint": The (SHA1) fingerprint of the TLS certificate
- "sni": The Server Name Indication (SNI) extension sent by the client
- "version": The SSL/TLS version used
- "notbefore": The NotBefore field from the TLS certificate
- "notafter": The NotAfter field from the TLS certificate
- "ja3": The JA3 fingerprint consisting of both a JA3 hash and a JA3 string
- "ja3s": The JA3S fingerprint consisting of both a JA3 hash and a JA3 string
- "ja4": The JA4 client fingerprint for TLS
- "client_alpns": array of strings with ALPN values
- "server_alpns": array of strings with ALPN values

JA3 and JA4 must be enabled in the Suricata config file (set 'app-layer.protocols.tls.ja3-fingerprints'/'app-layer.protocols.tls.ja4-fingerprints' to 'yes').

In addition to this, custom logging also allows the following fields:

- "certificate": The TLS certificate base64 encoded
- "chain": The entire TLS certificate chain base64 encoded
- "client_handshake": structure containing "version", "ciphers" ([u16]), "exts" ([u16]), "sig_algs" ([u16]), for client hello supported cipher suites, extensions, and signature algorithms respectively, in the order that they're mentioned (ie. unsorted)
- "server_handshake": structure containing "version", "chosen cipher", "exts" ([u16]), for server hello in the order that they're mentioned (ie. unsorted)

15.1.2.10.2. Examples

Example of regular TLS logging:

```
"tls": {
    "subject": "C=US, ST=California, L=Mountain View, O=Google Inc, CN=*.google.com",
    "issuerdn": "C=US, O=Google Inc, CN=Google Internet Authority G2"
}
```

Example of regular TLS logging for resumed sessions:

```
"tls": {
    "session_resumed": true
}
```

Example of extended TLS logging:

```
"tls": {
    "subject": "C=US, ST=California, L=Mountain View, O=Google Inc, CN=*.google.com",
    "issuerdn": "C=US, O=Google Inc, CN=Google Internet Authority G2",
    "serial": "0C:00:99:B7:D7:54:C9:F6:77:26:31:7E:BA:EA:7C:1C",
    "fingerprint": "8f:51:12:06:a0:cc:4e:cd:e8:a3:8b:38:f8:87:59:e5:af:95:ca:cd",
    "sni": "calendar.google.com",
    "version": "TLS 1.2",
    "notbefore": "2017-01-04T10:48:43",
    "notafter": "2017-03-29T10:18:00"
}
```

Example of certificate logging using TLS custom logging (subject, sni, certificate):

```
"tls": {
    "subject": "C=US, ST=California, L=Mountain View, O=Google Inc, CN=*.googleapis.com
    "sni": "www.googleapis.com",
    "certificate": "MIIE3TCCA8WgAwIBAgIIQPsvobRZN0gwDQYJKoZIhvcNAQELBQAwSTELMA [...]"
 }
```

15.1.2.11. Event type: TFTP

15.1.2.11.1. Fields

- "packet": The operation code, can be "read" or "write" or "error"

- "file": The filename transported with the tftp protocol
- "mode": The mode field, can be "octet" or "mail" or "netascii" (or any combination of upper and lower case)

Example of TFTP logging:

```
"tftp": {
    "packet": "write",
    "file": "rfc1350.txt",
    "mode": "octet"
 }
```

15.1.2.12. Event type: KRB5

15.1.2.12.1. KRB5 Fields

- "cname" (string): The client PrincipalName
- "encryption" (string): Encryption used (only in AS-REP and TGS-REP)
- "error_code" (string): Error code, if request has failed
- "failed_request" (string): The request type for which the response had an error_code
- "msg_type" (string): The message type: AS-REQ, AS-REP, etc...
- "realm" (string): The server Realm
- "sname" (string): The server PrincipalName
- "ticket_encryption" (string): Encryption used for ticket
- "ticket_weak_encryption" (boolean): Whether the encryption used for ticket is a weak cipher
- "weak_encryption" (boolean): Whether the encryption used in AS-REP or TGS-REP is a weak cipher

Examples of KRB5 logging:

Pipe open:

```
"krb5": {
  "msg_type": "KRB_TGS_REP",
  "cname": "robin",
  "realm": "CYLERA.LAB",
  "sname": "ldap/dc01",
  "encryption": "aes256-cts-hmac-sha1-96",
  "weak_encryption": false,
  "ticket_encryption": "aes256-cts-hmac-sha1-96",
  "ticket_weak_encryption": false
}
```

15.1.2.13. Event type: SMB

15.1.2.13.1. SMB Fields

- "id" (integer): internal transaction id
- "dialect" (string): the negotiated protocol dialect, or "unknown" if missing
- "command" (string): command name. E.g. SMB2_COMMAND_CREATE or SMB1_COMMAND_WRITE_ANDX
- "status" (string): status string. Can be both NT_STATUS or DOS_ERR and other variants
- "status_code" (string): status code as hex string
- "session_id" (integer): SMB2+ session_id. SMB1 user id.
- "tree_id" (integer): Tree ID
- "filename" (string): filename for CREATE and other commands.
- "disposition" (string): requested disposition. E.g. FILE_OPEN, FILE_CREATE and FILE_OVERWRITE. See https://msdn.microsoft.com/en-us/library/ee442175.aspx#Appendix_A_Target_119
- "access" (string): indication of how the file was opened. "normal" or "delete on close" (field is subject to change)
- "created", "accessed", "modified", "changed" (integer): timestamps in seconds since unix epoch
- "size" (integer): size of the requested file
- "fuid" (string): SMB2+ file GUID. SMB1 FID as hex.
- "share" (string): share name.
- "share_type" (string): FILE, PIPE, PRINT or unknown.
- "client_dialects" (array of strings): list of SMB dialects the client speaks.
- "client_guid" (string): client GUID
- "server_guid" (string): server GUID
- "request.native_os" (string): SMB1 native OS string
- "request.native_lm" (string): SMB1 native Lan Manager string
- "response.native_os" (string): SMB1 native OS string
- "response.native_lm" (string): SMB1 native Lan Manager string

One can restrict which transactions are logged by using the "types" field in the suricata.yaml file. If this field is not specified, all transactions types are logged. 9 values can be specified with this field as shown below:

Configuration:

```
- eve-log:
    enabled: yes
    type: file
    filename: eve.json
    types:
      - smb:
          types: [file, tree_connect, negotiate, dcerpc, create,
            session_setup, ioctl, rename, set_file_path_info, generic]
```

Examples of SMB logging:

Pipe open:

```
"smb": {
  "id": 1,
  "dialect": "unknown",
  "command": "SMB2_COMMAND_CREATE",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 4398046511201,
  "tree_id": 1,
  "filename": "atsvc",
  "disposition": "FILE_OPEN",
  "access": "normal",
  "created": 0,
  "accessed": 0,
  "modified": 0,
  "changed": 0,
  "size": 0,
  "fuid": "0000004d-0000-0000-0005-0000ffffffff"
}
```

File/pipe close:

```
"smb": {
  "id": 15,
  "dialect": "2.10",
  "command": "SMB2_COMMAND_CLOSE",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 4398046511121,
  "tree_id": 1,
}
```

Tree connect (share open):

```
"smb": {
  "id": 3,
  "dialect": "2.10",
  "command": "SMB2_COMMAND_TREE_CONNECT",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 4398046511121,
  "tree_id": 1,
  "share": "\\\\admin-pc\\c$",
  "share_type": "FILE"
}
```


Dialect negotiation from SMB1 to SMB2 dialect 2.10:


```
"smb": {
  "id": 1,
  "dialect": "2.??",
  "command": "SMB1_COMMAND_NEGOTIATE_PROTOCOL",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 0,
  "tree_id": 0,
  "client_dialects": [
    "PC NETWORK PROGRAM 1.0",
    "LANMAN1.0",
    "Windows for Workgroups 3.1a",
    "LM1.2X002",
    "LANMAN2.1",
    "NT LM 0.12",
    "SMB 2.002",
    "SMB 2.???"
  ],
  "server_guid": "aec6e793-2b11-4019-2d95-55453a0ad2f1"
}
"smb": {
  "id": 2,
  "dialect": "2.10",
  "command": "SMB2_COMMAND_NEGOTIATE_PROTOCOL",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 0,
  "tree_id": 0,
  "client_dialects": [
    "2.02",
    "2.10"
  ],
  "client_guid": "601985d2-aad9-11e7-8494-00088bb57f27",
  "server_guid": "aec6e793-2b11-4019-2d95-55453a0ad2f1"
}
```


SMB1 partial SMB1_COMMAND_SESSION_SETUP_ANDX:

```
"request": {
  "native_os": "Unix",
  "native_lm": "Samba 3.9.0-SVN-build-11572"
},
"response": {
  "native_os": "Windows (TM) Code Name \"Longhorn\" Ultimate 5231",
  "native_lm": "Windows (TM) Code Name \"Longhorn\" Ultimate 6.0"
}
```

15.1.2.13.2. DCERPC fields

- "request" (string): command. E.g. REQUEST, BIND.
- "response" (string): reply. E.g. RESPONSE, BINDACK or FAULT.
- "opnum" (integer): the opnum
- "call_id" (integer): the call id
- "frag_cnt" (integer): the number of fragments for the stub data
- "stub_data_size": total stub data size
- "interfaces" (array): list of interfaces
- "interfaces.uuid" (string): string representation of the UUID
- "interfaces.version" (string): interface version
- "interfaces.ack_result" (integer): ack result
- "interfaces.ack_reason" (integer): ack reason
- "interfaces.service" (string): service name for the UUID
- "interfaces.procedure" (string): procedure name for the UUID and opnum

DCERPC REQUEST/RESPONSE:

```
"smb": {
  "id": 4,
  "dialect": "unknown",
  "command": "SMB2_COMMAND_IOCTL",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 4398046511201,
  "tree_id": 0,
  "dcerpc": {
    "request": "REQUEST",
    "response": "RESPONSE",
    "opnum": 0,
    "req": {
      "frag_cnt": 1,
      "stub_data_size": 136
    },
    "res": {
      "frag_cnt": 1,
      "stub_data_size": 8
    },
    "call_id": 2
  }
}
```


DCERPC BIND/BINDACK:

```
"smb": {
  "id": 53,
  "dialect": "2.10",
  "command": "SMB2_COMMAND_WRITE",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 35184439197745,
  "tree_id": 1,
  "dcerpc": {
    "request": "BIND",
    "response": "BINDACK",
    "interfaces": [
      {
        "uuid": "12345778-1234-abcd-ef00-0123456789ac",
        "service": "samr",
        "version": "1.0",
        "ack_result": 2,
        "ack_reason": 0
      },
      {
        "uuid": "12345778-1234-abcd-ef00-0123456789ac",
        "service": "samr",
        "version": "1.0",
        "ack_result": 0,
        "ack_reason": 0
      },
      {
        "uuid": "12345778-1234-abcd-ef00-0123456789ac",
        "service": "samr",
        "version": "1.0",
        "ack_result": 3,
        "ack_reason": 0
      }
    ],
    "call_id": 2
  }
```


15.1.2.13.3. NTLMSSP fields

- "domain" (string): the Windows domain.
- "user" (string): the user.
- "host" (string): the host.
- "version" (string): the client version.

Example:

```
"ntlmssp": {
  "domain": "VNET3",
  "user": "administrator",
  "host": "BLU",
  "version": "60.230 build 13699 rev 188"
}
```

More complete example:

```
"smb": {
  "id": 3,
  "dialect": "NT LM 0.12",
  "command": "SMB1_COMMAND_SESSION_SETUP_ANDX",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 2048,
  "tree_id": 0,
  "ntlmssp": {
    "domain": "VNET3",
    "user": "administrator",
    "host": "BLU",
    "version": "60.230 build 13699 rev 188"
  },
  "request": {
    "native_os": "Unix",
    "native_lm": "Samba 3.9.0-SVN-build-11572"
  },
  "response": {
    "native_os": "Windows (TM) Code Name \"Longhorn\" Ultimate 5231",
    "native_lm": "Windows (TM) Code Name \"Longhorn\" Ultimate 6.0"
  }
}
```

15.1.2.13.4. Kerberos fields

- "kerberos.realm" (string): the Kerberos Realm.
- "kerberos.snames (array of strings): snames.

Example:

```
"smb": {
  "dialect": "2.10",
  "command": "SMB2_COMMAND_SESSION_SETUP",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 35184439197745,
  "tree_id": 0,
  "kerberos": {
    "realm": "CONTOSO.LOCAL",
    "snames": [
      "cifs",
      "DC1.contoso.local"
    ]
  }
}
```

15.1.2.14. Event type: BITTORRENT-DHT

15.1.2.14.1. Common fields:

- "transaction_id" (hex): the unique id of the transaction, generated by node making the request (a.k.a the querying node). Same transaction_id is echoed back by responding nodes.
- "client_version" (hex): identifies the type and version of the bittorrent-dht client. Some implementations may be missing this field.

15.1.2.14.2. Extra fields:

Packets should also contain one of either the fields:

error

- **"error": details of an error which occurred while processing the request**
    - "error.num" (num): the error code
    - "error.msg" (string): the error message

request_type and request

- "request_type" (string): the type of the request (a.k.a. the query). Included if this packet was a request
- **"request": a request (a.k.a. a query) sent by the bittorrent-dht client**
    - "request.id" (hex): the node ID of the node which sent the request (20 network byte order)
    - "request.target" (hex): the target node ID. Used by the find_node request_type

- "request.info_hash" (hex): info hash of target torrent (20 bytes). Used by the get_peers and announce_peer request_types
- "request.token" (hex): token key received from previous get_peers request. Used by the announce_peer request type
- "request.implied_port" (num): 0 or 1, if 1 ignore provided port and use source port of UDP packet. Used by the announce_peer request_type
- "request.port" (num): port on which peer will download torrent. Used by the announce_peer request_type

response

- **"response": a response to the client's request**
  - "response.id" (hex): the node ID of the node which sent the response (20 bytes in network byte order)
  - "response.nodes" (array): find_node/get_peers - a list of info objects for target node or K(8) closest good nodes in routing table
  - "response.nodes6" (array): find_node/get_peers - a list of info objects for target node or K(8) closest good nodes in routing table (ipv6)
  - "response.values" (array): list of compact peer info strings. Used by the get_peers request_type
  - "response.token" (hex): token key required for sender's future announce_peer query

node object

- "id" (hex): node ID
- "ip" (string): IPv4 or IPv6 address of node
- "port" (integer): node port

peer object (values array)

- "ip" (string): IPv6 or IPv6 address of node
- "port" (integer): node port

15.1.2.14.3. Examples:

Ping and response:

```
"bittorrent_dht": {
  "transaction_id": "0c17",
  "client_version": "4c540126",
  "request_type": "ping",
  "request": {
    "id": "41aff1580119f074e2f537f231f12adf684f0d1f"
  }
}

"bittorrent_dht": {
  "transaction_id": "0c17",
  "client_version": "5554b50c",
  "response": {
    "id": "42aeb304a0845b3b9ee089327b48967b8e87b2e2"
  }
}
```

Find_node and response:

```
"bittorrent_dht": {
  "transaction_id": "420f0000",
  "client_version": "5554b50c",
  "request_type": "find_node",
  "request": {
    "id": "37579bad1bad166af4329508096fae8c553c6cf4",
    "target": "37579bad1bad166af4329508096fae8c553c6cf4"
  }
}
```

Get_peers and response with values param:

```
"bittorrent_dht": {
  "transaction_id": "05e4",
  "client_version": "4c540126",
  "request_type": "get_peers",
  "request": {
    "id": "41aff1580119f074e2f537f231f12adf684f0d1f",
    "info_hash": "19a6fcfcba6cc2c6d371eb754074d095adb5d291"
  }
}
"bittorrent_dht": {
  "transaction_id": "05e4",
  "client_version": "555462d6",
  "response": {
    "id": "19a6f98be177e32e7b5bd77276d529f03e3ba8a9",
    "values": [
      {
        "ip": "45.238.190.2",
        "port": 6881
      },
      {
        "ip": "185.70.52.245",
        "port": 51215
      },
      {
        "ip": "45.21.238.247",
        "port": 55909
      },
      {
        "ip": "62.28.248.195",
        "port": 6881
      }
    ],
    "token": "c17094641ca8844d711120baecb2b5cf25435614"
  }
}
```


Get_peers and response with nodes param:

```
 "bittorrent_dht": {
  "transaction_id": "44e6",
  "client_version": "4c540126",
  "request_type": "get_peers",
  "request": {
    "id": "41aff1580119f074e2f537f231f12adf684f0d1f",
    "info_hash": "19a6fcfcba6cc2c6d371eb754074d095adb5d291"
  }
}

"bittorrent_dht": {
  "transaction_id": "44e6",
  "response": {
    "id": "19a7c8f4f6d14d9f87a67671720633e551f30cb7",
    "values": [
      {
        "ip": "45.22.252.153",
        "port": 36798
      },
      {
        "ip": "94.41.206.37",
        "port": 30850
      },
      {
        "ip": "84.228.120.50",
        "port": 6881
      },
      {
        "ip": "178.81.206.84",
        "port": 12373
      },
      {
        "ip": "110.188.93.186",
        "port": 22223
      }
    ],
    "token": "c897ee539e02a54595b4d7cfb6319ad48e71b282"
  }
}
```


Announce_peer and response:

```
"bittorrent_dht": {
  "transaction_id": "aa",
  "request_type": "announce_peer",
  "request": {
    "id": "abcdefghij0123456789",
    "info_hash": "mnopqrstuvwxyz123456",
    "token": "aoeusnth",
    "port": 6881
  }
}
"bittorrent_dht": {
  "transaction_id": "aa",
  "response": {
    "id": "mnopqrstuvwxyz123456"
  }
}
```

Announce_peer with implied_port param and response:

```
"bittorrent_dht": {
  "transaction_id": "7fe9",
  "client_version": "4c540126",
  "request_type": "announce_peer",
  "request": {
    "id": "51bc83f53417a62a40e8a48170cad369a13fef3c",
    "info_hash": "19a6fcfcba6cc2c6d371eb754074d095adb5d291",
    "token": "cacbef35",
    "implied_port": 1,
    "port": 54892
  }
}

"bittorrent_dht": {
  "transaction_id": "7fe9",
  "client_version": "4c54012f",
  "response": {
    "id": "19a66dece45e0288ab75d141e0255738a1ce8508"
  }
}
```

Sample error responses:

```
"bittorrent_dht": {
  "transaction_id": "aa",
  "error": {
    "num": 201,
    "msg": "A Generic Error Ocurred"
  }
}
"bittorrent_dht": {
  "transaction_id": "aa",
  "error": {
    "num": 203,
    "msg": "Malformed Packet"
  }
}
```

15.1.2.15. Event type: SSH

15.1.2.15.1. Fields

- "proto_version": The protocol version transported with the ssh protocol (1.x, 2.x)
- "software_version": The software version used by end user
- "hassh.hash": MD5 of hassh algorithms of client or server
- "hassh.string": hassh algorithms of client or server

Hassh must be enabled in the Suricata config file (set 'app-layer.protocols.ssh.hassh' to 'yes').

Example of SSH logging:

```
"ssh": {
  "client": {
      "proto_version": "2.0",
      "software_version": "OpenSSH_6.7",
      "hassh": {
          "hash": "ec7378c1a92f5a8dde7e8b7a1ddf33d1",
          "string": "curve25519-sha256,diffie-hellman-group14-sha256,diffie-hellman-
group14-sha1,ext-info-c",
      }
   },
  "server": {
      "proto_version": "2.0",
      "software_version": "OpenSSH_6.7",
      "hassh": {
          "hash": "ec7378c1a92f5a8dde7e8b7a1ddf33d1",
          "string": "curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256",
      }
   }
}
```

15.1.2.16. Event type: Flow

15.1.2.16.1. Fields

- "pkts_toserver": total number of packets to server, include bypassed packets
- "pkts_toclient": total number of packets to client
- "bytes_toserver": total bytes count to server
- "bytes_toclient": total bytes count to client
- "bypassed.pkts_toserver": number of bypassed packets to server
- "bypassed.pkts_toclient": number of bypassed packets to client
- "bypassed.bytes_toserver": bypassed bytes count to server
- "bypassed.bytes_toclient": bypassed bytes count to client
- "start": date of start of the flow
- "end": date of end of flow (last seen packet)
- "age": duration of the flow
- "bypass": if the flow has been bypassed, it is set to "local" (internal bypass) or "capture"
- "state": display state of the flow (include "new", "established", "closed", "bypassed")
- "reason": mechanism that did trigger the end of the flow (include "timeout", "forced" and "shutdown")
- "alerted": "true" or "false" depending if an alert has been seen on flow
- "action": "pass" or "drop" depending if flow was PASS'ed or DROP'ed (no present if none)
- "tx_cnt": number of transactions seen in the flow (only present if flow has an application layer)
- "exception_policy": array consisting of exception policies that have been triggered by the flow:
    - "target": if an exception policy was triggered, what setting exceptions led to this (cf. Exception Policy - Specific Settings).
    - "policy": if an exception policy was triggered, what policy was applied (to the flow or to any packet(s) from it).

Example

```
"flow": {
  "pkts_toserver": 23,
  "pkts_toclient": 21,
  "bytes_toserver": 4884,
  "bytes_toclient": 7392,
  "bypassed": {
    "pkts_toserver": 10,
    "pkts_toclient": 8,
    "bytes_toserver": 1305,
    "bytes_toclient": 984
  },
  "start": "2019-05-28T23:32:29.025256+0200",
  "end": "2019-05-28T23:35:28.071281+0200",
  "age": 179,
  "bypass": "capture",
  "state": "bypassed",
  "reason": "timeout",
  "alerted": false,
  "action": "pass",
  "exception_policy": [
    {
      "target": "stream_midstream",
      "policy": "pass_flow"
    }
  ]
}
```


15.1.2.17. Event type: RDP

Initial negotiations between RDP client and server are stored as transactions and logged.

Each RDP record contains a per-flow incrementing "tx_id" field.

The "event_type" field indicates an RDP event subtype. Possible values:

- "initial_request"
- "initial_response"
- "connect_request"
- "connect_response"
- "tls_handshake"

15.1.2.17.1. RDP type: Initial Request

The optional "cookie" field is a string identifier the RDP client has chosen to provide.

The optional "flags" field is a list of client directives. Possible values:

- "restricted_admin_mode_required"
- "redirected_authentication_mode_required"
- "correlation_info_present"

15.1.2.17.2. RDP type: Initial Response

In the event of a standard initial response:

The "protocol" field is the selected protocol. Possible values:

- "rdp"
- "ssl"
- "hybrid"
- "rds_tls"
- "hybrid_ex"

The optional "flags" field is a list of support server modes. Possible values:

- "extended_client_data"
- "dynvc_gfx"
- "restricted_admin"
- "redirected_authentication"

Alternatively, in the event of an error-indicating initial response:

There will be no "protocol" or "flags" fields.

The "error_code" field will contain the numeric code provided by the RDP server.

The "reason" field will contain a text summary of this code. Possible values:

- "ssl required by server" (error code 0x1)
- "ssl not allowed by server" (error code 0x2)
- "ssl cert not on server" (error code 0x3)
- "inconsistent flags" (error code 0x4)
- "hybrid required by server" (error code 0x5)
- "ssl with user auth required by server" (error code 0x6)

15.1.2.17.3. RDP type: Connect Request

The optional "channel" field is a list of requested data channel names.

Common channels:

- "rdpdr" (device redirection)
- "cliprdr" (shared clipboard)
- "rdpsnd" (sound)

The optional "client" field is a sub-object that may contain the following:

- "version": RDP protocol version. Possible values are "v4", "v5", "v10.0", "v10.1", "v10.2", "v10.3", "v10.4", "v10.5", "v10.6", "v10.7", "unknown".
- "desktop_width": Numeric desktop width value.
- "desktop_height": Numeric desktop height value.
- "color_depth": Numeric color depth. Possible values are 4, 8, 15, 16, 24.
- "keyboard_layout": Locale identifier name, e.g., "en-US".
- "build": OS and SP level, e.g., "Windows XP", "Windows 7 SP1".
- "client_name": Client computer name.
- "keyboard_type": Possible values are "xt", "ico", "at", "enhanced", "1050", "9140", "jp".
- "keyboard_subtype": Numeric code for keyboard.
- "function_keys": Number of function keys on client keyboard.
- "ime": Input method editor (IME) file name.
- "product_id": Product id string.
- "serial_number": Numeric value.
- "capabilities": List of any of the following: "support_errinfo_pdf", "want_32bpp_session", "support_statusinfo_pdu", "strong_asymmetric_keys", "valid_connection_type", "support_monitor_layout_pdu", "support_netchar_autodetect", "support_dynvc_gfx_protocol", "support_dynamic_time_zone", "support_heartbeat_pdu".
- "id": Client product id string.
- "connection_hint": Possible values are "modem", "low_broadband", "satellite", "high_broadband", "wan", "lan", "autodetect".
- "physical_width": Numeric physical width of display.
- "physical_height": Numeric physical height of display.
- "desktop_orientation": Numeric angle of orientation.
- "scale_factor": Numeric scale factor of desktop.
- "device_scale_factor": Numeric scale factor of display.

15.1.2.17.4. RDP type: Connect Response

With this event, the initial RDP negotiation is complete in terms of tracking and logging.

15.1.2.17.5. RDP type: TLS Handshake

With this event, the initial RDP negotiation is complete in terms of tracking and logging.

The session will use TLS encryption.

The "x509_serials" field is a list of observed certificate serial numbers, e.g., "16ed2aa0495f259d4f5d99edada570d1".

15.1.2.17.6. Examples

RDP logging:

```
"rdp": {
  "tx_id": 0,
  "event_type": "initial_request",
  "cookie": "A70067"
}

"rdp": {
  "tx_id": 1,
  "event_type": "initial_response"
}

"rdp": {
  "tx_id": 2,
  "event_type": "connect_request",
  "client": {
    "version": "v5",
    "desktop_width": 1152,
    "desktop_height": 864,
    "color_depth": 15,
    "keyboard_layout": "en-US",
    "build": "Windows XP",
    "client_name": "ISD2-KM84178",
    "keyboard_type": "enhanced",
    "function_keys": 12,
    "product_id": 1,
    "capabilities": [
      "support_errinfo_pdf"
    ],
    "id": "55274-OEM-0011903-00107"
  },
  "channels": [
    "rdpdr",
    "cliprdr",
    "rdpsnd"
  ]
}

"rdp": {
  "tx_id": 3,
  "event_type": "connect_response"
}
```


RDP logging, with transition to TLS:

```
"rdp": {
  "tx_id": 0,
  "event_type": "initial_request",
  "cookie": "AWAKECODI"
}

"rdp": {
  "tx_id": 1,
  "event_type": "initial_response",
  "server_supports": [
    "extended_client_data"
  ],
  "protocol": "hybrid"
}

"rdp": {
  "tx_id": 2,
  "event_type": "tls_handshake",
  "x509_serials": [
    "16ed2aa0495f259d4f5d99edada570d1"
  ]
}
```

15.1.2.18. Event type: RFB

15.1.2.18.1. Fields

- "server_protocol_version.major", "server_protocol_version.minor": The RFB protocol version offered by the server.
- "client_protocol_version.major", "client_protocol_version.minor": The RFB protocol version agreed by the client.
- "authentication.security_type": Security type agreed upon in the logged transaction, e.g. `2` is VNC auth.
- "authentication.vnc.challenge", "authentication.vnc.response": Only available when security type 2 is used. Contains the challenge and response byte buffers exchanged by the server and client as hex strings.
- "authentication.security_result": Result of the authentication process (`OK`, `FAIL` or `TOOMANY`).
- "screen_shared": Boolean value describing whether the client requested screen sharing.
- "framebuffer": Contains metadata about the initial screen setup process. Only available when the handshake completed this far.
- "framebuffer.width", "framebuffer.height": Screen size as offered by the server.
- "framebuffer.name": Desktop name as advertised by the server.
- "framebuffer.pixel_format": Pixel representation information, such as color d
  RFC6143 (https://tools.ietf.org/html/rfc6143) for details.

15.1.2.18.2. Examples

Example of RFB logging, with full VNC style authentication parameters:

```
"rfb": {
  "server_protocol_version": {
    "major": "003",
    "minor": "007"
  },
  "client_protocol_version": {
    "major": "003",
    "minor": "007"
  },
  "authentication": {
    "security_type": 2,
    "vnc": {
      "challenge": "0805b790b58e967f2b350a0c99de3881",
      "response": "aecb26faeaaa62179636a5934bac1078"
    },
    "security_result": "OK"
  },
  "screen_shared": false,
  "framebuffer": {
    "width": 1280,
    "height": 800,
    "name": "foobar@localhost.localdomain",
    "pixel_format": {
      "bits_per_pixel": 32,
      "depth": 24,
      "big_endian": false,
      "true_color": true,
      "red_max": 255,
      "green_max": 255,
      "blue_max": 255,
      "red_shift": 16,
      "green_shift": 8,
      "blue_shift": 0
    }
  }
```

15.1.2.19. Event type: MQTT

EVE-JSON output for MQTT consists of one object per MQTT transaction, with some common and various type-specific fields.

15.1.2.19.1. Transactions

A single MQTT communication can consist of multiple messages that need to be between broker and client. For example, some actions at higher QoS levels (> 0) usually involve a combination of requests and acknowledgement messages that are linked by a common

identifier:

- `CONNECT` followed by `CONNACK`
- `PUBLISH` followed by `PUBACK` (QoS 1) or `PUBREC` / `PUBREL` / `PUBCOMP` (QoS 2)
- `SUBSCRIBE` followed by `SUBACK`
- `UNSUBSCRIBE` followed by `UNSUBACK`

The MQTT parser merges individual messages into one EVE output item if they belong to one transaction. In such cases, the source and destination information (IP/port) reflect the direction of the initial request, but contain messages from both sides.

Example for a PUBLISH at QoS 2:


```
{
  "timestamp": "2020-05-19T18:00:39.016985+0200",
  "flow_id": 1454127794305760,
  "pcap_cnt": 65,
  "event_type": "mqtt",
  "src_ip": "0000:0000:0000:0000:0000:0000:0000:0001",
  "src_port": 60105,
  "dest_ip": "0000:0000:0000:0000:0000:0000:0000:0001",
  "dest_port": 1883,
  "proto": "TCP",
  "mqtt": {
    "publish": {
      "qos": 2,
      "retain": false,
      "dup": false,
      "topic": "house/bulbs/bulb1",
      "message_id": 3,
      "message": "OFF"
    },
    "pubrec": {
      "qos": 0,
      "retain": false,
      "dup": false,
      "message_id": 3
    },
    "pubrel": {
      "qos": 1,
      "retain": false,
      "dup": false,
      "message_id": 3
    },
    "pubcomp": {
      "qos": 0,
      "retain": false,
      "dup": false,
      "message_id": 3
    }
  }
}
```

Note that some message types (aka control packet types), such as `PINGREQ` and `PINGRESP`, have no type-specific data, nor do they have information that facilitate grouping into transactions. These will be logged as single items and only contain the common fields listed below.

15.1.2.19.2. Common fields

Common fields from the MQTT fixed header:

- "*.qos": Quality of service level for the message, integer between 0 and 2.
- "*.retain": Boolean value of the MQTT 'retain' flag.
- "*.dup": Boolean value of the MQTT 'dup' (duplicate) flag.

15.1.2.19.3. MQTT CONNECT fields

- "connect.protocol_string": Protocol string as defined in the spec, e.g. `MQTT` (MQTT 3.1.1 and later) or `MQIsdp` (MQTT 3.1).
- "connect.protocol_version": Protocol version as defined in the specification:
    - protocol version `3`: MQTT 3.1
    - protocol version `4`: MQTT 3.1.1
    - protocol version `5`: MQTT 5.0
- "connect.flags.username", "connect.flags.password": Set to *true* if credentials are submitted with the connect request.
- "connect.flags.will": Set to *true* if a will is set.
- "connect.flags.will_retain": Set to *true* if the will is to be retained on the broker.
- "connect.will.clean_session": Set to *true* if the connection is to made with a clean session.
- "connect.client_id": Client ID string submitted my the connecting client.
- "connect.username", "connect.password": User/password authentication credentials submitted with the connect request. Passwords are only logged when the corresponding configuration setting is enabled (`mqtt.passwords: yes`).
- "connect.will.topic": Topic to publish the will message to.
- "connect.will.message": Message to be published on connection loss.
- "connect.will.properties": (Optional, MQTT 5.0) Will properties set on this request. See 3.1.3.2 in the spec for more information on will properties.
- "connect.properties": (Optional, MQTT 5.0) CONNECT properties set on this request. See 3.1.2.11 in the spec for more information on CONNECT properties.

Example of MQTT CONNECT logging:

```
"connect": {
  "qos": 0,
  "retain": false,
  "dup": false,
  "protocol_string": "MQTT",
  "protocol_version": 5,
  "flags": {
    "username": true,
    "password": true,
    "will_retain": false,
    "will": true,
    "clean_session": true
  },
  "client_id": "client",
  "username": "user",
  "password": "pass",
  "will": {
    "topic": "willtopic",
    "message": "willmessage",
    "properties": {
      "content_type": "mywilltype",
      "correlation_data": "3c32aa4313b3e",
      "message_expiry_interval": 133,
      "payload_format_indicator": 144,
      "response_topic": "response_topic1",
      "userprop": "uservalue",
      "will_delay_interval": 200
    }
  },
  "properties": {
    "maximum_packet_size": 11111,
    "receive_maximum": 222,
    "session_expiry_interval": 555,
    "topic_alias_maximum": 666,
    "userprop1": "userval1",
    "userprop2": "userval2"
  }
}
```


15.1.2.19.4. MQTT CONNACK fields

- "connack.session_present": Set to *true* if a session is continued on connection.
- "connack.return_code": Return code/reason code for this reply. See 3.2.2.2 in the spec for more information on these codes.
- "connect.properties": (Optional, MQTT 5.0) CONNACK properties set on this request. See 3.2.2.3 in the spec for more information on CONNACK properties.

Example of MQTT CONNACK logging:

```
"connack": {
  "qos": 0,
  "retain": false,
  "dup": false,
  "session_present": false,
  "return_code": 0,
  "properties": {
    "topic_alias_maximum": 10
  }
}
```

15.1.2.19.5. MQTT PUBLISH fields

- "publish.topic": Topic this message is published to.
- "publish.message_id": (Only present if QOS level > 0) Message ID for this publication.
- "publish.message": Message to be published.
- "publish.properties": (Optional, MQTT 5.0) PUBLISH properties set on this request. See 3.3.2.3 in the spec for more information on PUBLISH properties.

Example of MQTT PUBLISH logging:

```
"publish": {
  "qos": 1,
  "retain": false,
  "dup": false,
  "topic": "topic",
  "message_id": 1,
  "message": "baa baa sheep",
  "properties": {
    "content_type": "mytype",
    "correlation_data": "3c32aa4313b3e",
    "message_expiry_interval": 77,
    "payload_format_indicator": 88,
    "response_topic": "response_topic1",
    "topic_alias": 5,
    "userprop": "userval"
  }
}
```

15.1.2.19.6. MQTT PUBACK/PUBREL/PUBREC/PUBCOMP fields

- "[puback|pubrel|pubrec|pubcomp].message_id": Original message ID this message refers to.
- "[puback|pubrel|pubrec|pubcomp].reason_code": Return code/reason code for this reply. See the spec for more information on these codes.
- "[puback|pubrel|pubrec|pubcomp].properties": (Optional, MQTT 5.0) Properties set on this request. See the spec for more information on these properties.

Example of MQTT PUBACK/PUBREL/PUBREC/PUBCOMP logging:

```
"puback": {
  "qos": 0,
  "retain": false,
  "dup": false,
  "message_id": 1,
  "reason_code": 16
}
```

15.1.2.19.7. MQTT SUBSCRIBE fields

- "subscribe.message_id": (Only present if QOS level > 0) Message ID for this subscription.
- "subscribe.topics": Array of pairs describing the subscribed topics:
  - "subscribe.topics[].topic": Topic to subscribe to.
  - "subscribe.topics[].qos": QOS level to apply for when subscribing.
- "subscribe.properties": (Optional, MQTT 5.0) SUBSCRIBE properties set on this request. See 3.8.2.1 in the spec for more information on SUBSCRIBE properties.

Example of MQTT SUBSCRIBE logging:

```
"subscribe": {
  "qos": 1,
  "retain": false,
  "dup": false,
  "message_id": 1,
  "topics": [
    {
      "topic": "topicX",
      "qos": 0
    },
    {
      "topic": "topicY",
      "qos": 0
    }
  ]
}
```

15.1.2.19.8. MQTT SUBACK fields

- "suback.message_id": Original message ID this message refers to.
- "suback.qos_granted": Array of QOS levels granted for the subscribed topics, in the order of the original request.
- "suback.properties": (Optional, MQTT 5.0) SUBACK properties set on this request. See 3.9.2.1 in the spec for more information on SUBACK properties.

Example of MQTT SUBACK logging:

```
"suback": {
  "qos": 0,
  "retain": false,
  "dup": false,
  "message_id": 1,
  "qos_granted": [
    0,
    0
  ]
}
```

15.1.2.19.9. MQTT UNSUBSCRIBE fields

- "unsubscribe.message_id": (Only present if QOS level > 0) Message ID for this unsubscribe action.
- "unsubscribe.topics": Array of topics to be unsubscribed from.
- "unsubscribe.properties": (Optional, MQTT 5.0) UNSUBSCRIBE properties set on this request. See 3.10.2.1 in the spec for more information on UNSUBSCRIBE properties.

Example of MQTT UNSUBSCRIBE logging:

```
"unsubscribe": {
  "qos": 1,
  "retain": false,
  "dup": false,
  "message_id": 1,
  "topics": [
    "topicX",
    "topicY"
  ]
}
```

15.1.2.19.10. MQTT UNSUBACK fields

- "unsuback.message_id": Original message ID this message refers to.

Example of MQTT UNSUBACK logging:

```
"unsuback": {
  "qos": 0,
  "retain": false,
  "dup": false,
  "message_id": 1
}
```

15.1.2.19.11. MQTT AUTH fields (MQTT 5.0)

- "auth.reason_code": Return code/reason code for this message. See 3.15.2.1 in the spec for more information on these codes.
- "auth.properties": (Optional, MQTT 5.0) Properties set on this request. See 3.15.2.2 in the spec for more information on these properties.

Example of MQTT AUTH logging:

```
"auth": {
  "qos": 0,
  "retain": false,
  "dup": false,
  "reason_code": 16
}
```

15.1.2.19.12. MQTT DISCONNECT fields

- "auth.reason_code": (Optional) Return code/reason code for this message. See 3.14.2.1 in the spec for more information on these codes.
- "auth.properties": (Optional, MQTT 5.0) Properties set on this request. See 3.14.2.2 in the spec for more information on DISCONNECT properties.

Example of MQTT DISCONNECT logging:

```
"disconnect": {
  "qos": 0,
  "retain": false,
  "dup": false,
  "reason_code": 4,
  "properties": {
    "session_expiry_interval": 122,
  }
}
```

15.1.2.19.13. Truncated MQTT data

Messages exceeding the maximum message length limit (config setting `app-layer.protocols.mqtt.max-msg-length`) will not be parsed entirely to reduce the danger of denial of service issues. In such cases, only reduced metadata will be included in the EVE-JSON output. Furthermore, since no message ID is parsed, such messages can not be placed into transactions, hence, they will always appear as a single transaction.

These truncated events will -- besides basic communication metadata -- only contain the following fields:

- "truncated": Set to *true* if the entry is truncated.
- "skipped_length": Size of the original message.

Example of a truncated MQTT PUBLISH message (with 10000 being the maximum length):


```
{
  "timestamp": "2020-06-23T16:25:48.729785+0200",
  "flow_id": 1872904524326406,
  "pcap_cnt": 107,
  "event_type": "mqtt",
  "src_ip": "0000:0000:0000:0000:0000:0000:0000:0001",
  "src_port": 53335,
  "dest_ip": "0000:0000:0000:0000:0000:0000:0000:0001",
  "dest_port": 1883,
  "proto": "TCP",
  "mqtt": {
    "publish": {
      "qos": 0,
      "retain": false,
      "dup": false,
      "truncated": true,
      "skipped_length": 100011
    }
```


15.1.2.20. Event type: HTTP2

15.1.2.20.1. Fields

There are the two fields "request" and "response" which can each contain the same set of fields : * "settings": a list of settings with "name" and "value" * "headers": a list of headers with either "name" and "value", or "table_size_update", or "error" if any * "error_code": the error code from GOAWAY or RST_STREAM, which can be "NO_ERROR" * "priority": the stream

15.1.2.20.2. Examples

Example of HTTP2 logging, of a settings frame:

```
"http2": {
  "request": {
    "settings": [
      {
        "settings_id": "SETTINGSMAXCONCURRENTSTREAMS",
        "settings_value": 100
      },
      {
        "settings_id": "SETTINGSINITIALWINDOWSIZE",
        "settings_value": 65535
      }
    ]
  },
  "response": {}
}
```

Example of HTTP2 logging, of a request and response:

```
"http2": {
  "request": {
    "headers": [
      {
        "name": ":authority",
        "value": "localhost:3000"
      },
      {
        "name": ":method",
        "value": "GET"
      },
      {
        "name": ":path",
        "value": "/doc/manual/html/index.html"
      },
      {
        "name": ":scheme",
        "value": "http"
      },
      {
        "name": "accept",
        "value": "*/*"
      },
      {
        "name": "accept-encoding",
        "value": "gzip, deflate"
      },
      {
        "name": "user-agent",
        "value": "nghttp2/0.5.2-DEV"
      }
    ]
  },
  "response": {
    "headers": [
      {
        "name": ":status",
        "value": "200"
      },
      {
        "name": "server",
        "value": "nghttpd nghttp2/0.5.2-DEV"
      },
      {
        "name": "content-length",
        "value": "22617"
      },
      {
        "name": "cache-control",
        "value": "max-age=3600"
      },
      {
        "name": "date",
        "value": "Sat, 02 Aug 2014 10:50:25 GMT"
      },
      {
        "name": "last-modified",
        "value": "Sat, 02 Aug 2014 07:58:59 GMT"
      }
```

```
    ]
  }
}
```

15.1.2.21. Event type: PGSQL

PGSQL eve-logs reflect the bidirectional nature of the protocol transactions. Each PGSQL event lists at most one "Request" message field and one or more "Response" messages.

The PGSQL parser merges individual messages into one EVE output item if they belong to the same transaction. In such cases, the source and destination information (IP/port) reflect the direction of the initial request, but contain messages from both sides.

Example of `pgsql` event for a SimpleQuery transaction complete with request with a `SELECT` statement and its response:

```
{
  "timestamp": "2021-11-24T16:56:24.403417+0000",
  "flow_id": 1960113262002448,
  "pcap_cnt": 780,
  "event_type": "pgsql",
  "src_ip": "172.18.0.1",
  "src_port": 54408,
  "dest_ip": "172.18.0.2",
  "dest_port": 5432,
  "proto": "TCP",
  "pgsql": {
    "tx_id": 4,
    "request": {
      "simple_query": "select * from rule limit 5000;"
    },
    "response": {
      "field_count": 7,
      "data_rows": 5000,
      "data_size": 3035751,
      "command_completed": "SELECT 5000"
    }
  }
}
```

While on the wire PGSQL messages follow basically two types (startup messages and regular messages), those may have different subfields and/or meanings, based on the message type. Messages are logged based on their type and relevant fields.

We list a few possible message types and what they mean in Suricata. For more details on message types and formats as well as what each message and field mean for PGSQL, check PostgreSQL's official documentation.

15.1.2.21.1. Fields

- "tx_id": internal transaction id.
- "request": each PGSQL transaction may have up to one request message. The possible messages will be described in another section.
- "response": even when there are several "Response" messages, there is one `response` field that summarizes all responses for that transaction. The possible messages will be described in another section.

15.1.2.21.2. Request Messages

Requests are sent by the frontend (client), which would be the source of a pgsql flow. Some of the possible request messages are:

- "startup_message": message sent to start a new PostgreSQL connection
- "password": if password output for PGSQL is enabled in suricata.yaml, carries the password sent during Authentication phase
- "password_redacted": set to true in case there is a password message, but its logging is disabled
- "simple_query": issued SQL command during simple query subprotocol. PostgreSQL identifies specific sets of commands that change the set of expected messages to be exchanged as subprotocols.
- `"message": "cancel_request"`: sent after a query, when the frontend attempts to cancel said query. This message is sent over a different port, thus bring shown as a different flow. It has no direct answer from the backend, but if successful will lead to an `ErrorResponse` in the transaction where the query was sent.
- "message": requests which do not have meaningful payloads are logged like this, where the field value is the message type
- "copy_data_in": object. Part of the CopyIn subprotocol, consolidated data resulting from a `Copy From Stdin` query
- "copy_done": string. Similar to `command_completed` but sent after the frontend finishes sending a batch of `CopyData` messages

There are several different authentication messages possible, based on selected authentication method. (e.g. the SASL authentication will have a set of authentication message when `md5` authentication is chosen).

15.1.2.21.3. Response Messages

Responses are sent by the backend (server), which would be the destination of a pgsql flow. Some of the possible request messages are:

- "authentication_sasl_final": final SCRAM `server-final-message`, as explained at https://www.postgresql.org/docs/14/sasl-authentication.html#SASL-SCRAM-SHA-256
- "message": Backend responses which do not have meaningful payloads are logged like this, where the field value is the message type
- "error_response"
- "notice_response"
- "notification_response"
- "authentication_md5_password": a string with the `md5` salt value
- "parameter_status": logged as an array
- "backend_key_data"
- "data_rows": integer. When one or many `DataRow` messages are parsed, the total returned rows
- "data_size": in bytes. When one or many `DataRow` messages are parsed, the total size in bytes of the data returned
- "command_completed": string. Informs the command just completed by the backend
- "copy_in_response": object. Indicates the beginning of a CopyIn mode, shows how many columns will be copied from STDIN (`columns` field)
- "copy_out_response": object. Indicates the beginning of a CopyTo mode, shows how many columns will be copied to STDOUT (`columns` field)
- "copy_data_out": object. Consolidated data on the CopyData sent by the backend in a CopyOut transaction
- "copy_done": string. Similar to `command_completed` but sent after the backend finishes sending a batch of `CopyData` messages
- "ssl_accepted": bool. With this event, the initial PGSQL SSL Handshake negotiation is complete in terms of tracking and logging. The session will be upgraded to use TLS encryption

15.1.2.21.4. Examples

The two `pgsql` events in this example represent a rejected `SSL handshake` and a following connection request where the authentication method indicated by the backend was `md5`:

```
{
  "timestamp": "2021-11-24T16:56:19.435242+0000",
  "flow_id": 1960113262002448,
  "pcap_cnt": 21,
  "event_type": "pgsql",
  "src_ip": "172.18.0.1",
  "src_port": 54408,
  "dest_ip": "172.18.0.2",
  "dest_port": 5432,
  "proto": "TCP",
  "pgsql": {
    "tx_id": 1,
    "request": {
      "message": "SSL Request"
    },
    "response": {
      "accepted": false
    }
  }
}
{
  "timestamp": "2021-11-24T16:56:19.436228+0000",
  "flow_id": 1960113262002448,
  "pcap_cnt": 25,
  "event_type": "pgsql",
  "src_ip": "172.18.0.1",
  "src_port": 54408,
  "dest_ip": "172.18.0.2",
  "dest_port": 5432,
  "proto": "TCP",
  "pgsql": {
    "tx_id": 2,
    "request": {
      "protocol_version": "3.0",
      "startup_parameters": {
        "user": "rules",
        "database": "rules",
        "optional_parameters": [
          {
            "application_name": "psql"
          },
          {
            "client_encoding": "UTF8"
          }
        ]
      }
    },
    "response": {
      "authentication_md5_password": "Z\\xdc\\xfdf"
    }
  }
}
```


`AuthenticationOk`: a response indicating that the connection was successfully e

```
{
  "pgsql": {
    "tx_id": 3,
    "response": {
      "message": "authentication_ok",
      "parameter_status": [
        {
          "application_name": "psql"
        },
        {
          "client_encoding": "UTF8"
        },
        {
          "date_style": "ISO, MDY"
        },
        {
          "integer_datetimes": "on"
        },
        {
          "interval_style": "postgres"
        },
        {
          "is_superuser": "on"
        },
        {
          "server_encoding": "UTF8"
        },
        {
          "server_version": "13.6 (Debian 13.6-1.pgdg110+1)"
        },
        {
          "session_authorization": "rules"
        },
        {
          "standard_conforming_strings": "on"
        },
        {
          "time_zone": "Etc/UTC"
        }
      ],
      "process_id": 28954,
      "secret_key": 889887985
    }
  }
}
```


Note

In Suricata, the `AuthenticationOk` message is also where the backend's `process_id` and `secret_key` are logged. These must be sent by the frontend when it issues a `CancelRequest` message (seen below).

A `CancelRequest` message:

```
{
   "timestamp": "2023-12-07T15:46:56.971150+0000",
   "flow_id": 775771889500133,
   "event_type": "pgsql",
   "src_ip": "100.88.2.140",
   "src_port": 39706,
   "dest_ip": "100.96.199.113",
   "dest_port": 5432,
   "proto": "TCP",
   "pkt_src": "stream (flow timeout)",
   "pgsql": {
     "tx_id": 1,
     "request": {
       "message": "cancel_request",
       "process_id": 28954,
       "secret_key": 889887985
     }
   }
}
```


Note

As the `CancelRequest` message is sent over a new connection, the way to correlate it with the proper frontend/flow from which it originates is by querying on `process_id` and `secret_key` seen in the `AuthenticationOk` event.

References:

- PostgreSQL protocol - Canceling Requests in Progress
- PostgreSQL message format - BackendKeyData

15.1.2.21.5. Field Reference

15.1.2.21.5.1. Top Level (object)

Name	Type	Description
request	object	
response	object	
tx_id	integer	

15.1.2.21.5.2. response (object)

Name	Type	Description
authentication_md5_password	string	

Name	Type	Description
authentication_sasl_final	string	
code	string	
command_completed	string	
copy_data_out	object	CopyData message from CopyOut mode
copy_in_response	object	Backend/server response accepting CopyIn mode
copy_out_response	object	Backend/server response accepting CopyOut mode
data_rows	integer	
data_size	integer	
field_count	integer	
file	string	
line	string	
message	string	
parameter_status	array of objects	
process_id	integer	
routine	string	
secret_key	integer	
severity_localizable	string	
severity_non_localizable	string	
ssl_accepted	boolean	

15.1.2.21.5.3. response.parameter_status (array of objects)

Name	Type	Description
application_name	string	
client_encoding	string	
date_style	string	
integer_datetimes	string	
interval_style	string	

Name	Type	Description
is_superuser	string	
server_encoding	string	
server_version	string	
session_authorization	string	
standard_conforming_strings	string	
time_zone	string	

15.1.2.21.5.4. response.copy_out_response (object)

Name	Type	Description
columns	integer	Number of columns that will be copied in the CopyData message

15.1.2.21.5.5. response.copy_in_response (object)

Name	Type	Description
columns	integer	Number of columns that will be copied in the CopyData message

15.1.2.21.5.6. response.copy_data_out (object)

Name	Type	Description
data_size	integer	Accumulated data size of all CopyData messages sent
row_count	integer	Number of rows sent in CopyData messages

15.1.2.21.5.7. request (object)

Name	Type	Description
copy_data_in	object	CopyData message from CopyIn mode
message	string	
password	string	

Name	Type	Description
password_redacted	boolean	Indicates if a password message was received but not logged due to Suricata settings
process_id	integer	
protocol_version	string	
sasl_authentication_mechanism	string	
sasl_param	string	
sasl_response	string	
secret_key	integer	
simple_query	string	
startup_parameters	object	

15.1.2.21.5.8. request.startup_parameters (object)

Name	Type	Description
optional_parameters	array of objects	
user	string	

15.1.2.21.5.9. request.startup_parameters.optional_parameters (array of objects)

Name	Type	Description
application_name	string	
client_encoding	string	
database	string	
datestyle	string	
extra_float_digits	string	
options	string	
replication	string	

15.1.2.21.5.10. request.copy_data_in (object)

Name	Type	Description
data_size	integer	Accumulated data size of all CopyData messages sent
msg_count	integer	How many CopyData messages were sent (does not necessarily match number of rows from the query)

15.1.2.22. Event type: IKE

The parser implementations for IKEv1 and IKEv2 have a slightly different feature set. They can be distinguished using the "version_major" field (which equals either 1 or 2). The unique properties are contained within a separate "ikev1" and "ikev2" sub-object.

15.1.2.22.1. Fields

- "init_spi", "resp_spi": The Security Parameter Index (SPI) of the initiator and responder.
- "version_major": Major version of the ISAKMP header.
- "version_minor": Minor version of the ISAKMP header.
- "payload": List of payload types in the current packet.
- "exchange_type": Type of the exchange, as numeric values.
- "exchange_type_verbose": Type of the exchange, in human-readable form. Needs `extended: yes` set in the `ike` EVE output option.
- "alg_enc", "alg_hash", "alg_auth", "alg_dh", "alg_esn": Properties of the chosen security association by the server.
- "ikev1.encrypted_payloads": Set to `true` if the payloads in the packet are encrypted.
- "ikev1.doi": Value of the domain of interpretation (DOI).
- "ikev1.server.key_exchange_payload", "ikev1.client.key_exchange_payload": Public key exchange payloads of the server and client.
- "ikev1.server.key_exchange_payload_length", "ikev1.client.key_exchange_payload_length": Length of the public key exchange payload.
- "ikev1.server.nonce_payload", "ikev1.client.nonce_payload": Nonce payload of the server and client.
- "ikev1.server.nonce_payload_length", "ikev1.client.nonce_payload_length": Length of the nonce payload.
- "ikev1.client.client_proposals": List of the security associations proposed to the server.
- "ikev1.vendor_ids": List of the vendor IDs observed in the communication.
- "server_proposals": List of server proposals with parameters, if there are more than one. This is a non-standard case; this field is only present if such a situation was observed in the inspected traffic.

15.1.2.22.2. Examples

Example of IKE logging:

```
"ike": {
  "version_major": 1,
  "version_minor": 0,
  "init_spi": "8511617bfea2f172",
  "resp_spi": "c0fc6bae013de0f5",
  "message_id": 0,
  "exchange_type": 2,
  "exchange_type_verbose": "Identity Protection",
  "sa_life_type": "LifeTypeSeconds",
  "sa_life_type_raw": 1,
  "sa_life_duration": "Unknown",
  "sa_life_duration_raw": 900,
  "alg_enc": "EncAesCbc",
  "alg_enc_raw": 7,
  "alg_hash": "HashSha2_256",
  "alg_hash_raw": 4,
  "alg_auth": "AuthPreSharedKey",
  "alg_auth_raw": 1,
  "alg_dh": "GroupModp2048Bit",
  "alg_dh_raw": 14,
  "sa_key_length": "Unknown",
  "sa_key_length_raw": 256,
  "alg_esn": "NoESN",
  "payload": [
    "VendorID",
    "Transform",
    "Proposal",
    "SecurityAssociation"
  ],
  "ikev1": {
    "doi": 1,
    "encrypted_payloads": false,
    "client": {
      "key_exchange_payload": "0bf7907681a656aabed38fb1ba8918b10d707a8e635a...",
      "key_exchange_payload_length": 256,
      "nonce_payload": "1427d158fc1ed6bbbc1bd81e6b74960809c87d18af5f0abef14d5274ac232904",
      "nonce_payload_length": 32,
      "proposals": [
        {
          "sa_life_type": "LifeTypeSeconds",
          "sa_life_type_raw": 1,
          "sa_life_duration": "Unknown",
          "sa_life_duration_raw": 900,
          "alg_enc": "EncAesCbc",
          "alg_enc_raw": 7,
          "alg_hash": "HashSha2_256",
          "alg_hash_raw": 4,
          "alg_auth": "AuthPreSharedKey",
          "alg_auth_raw": 1,
          "alg_dh": "GroupModp2048Bit",
          "alg_dh_raw": 14,
          "sa_key_length": "Unknown",
          "sa_key_length_raw": 256
        }
      ]
    },
    "server": {
      "key_exchange_payload": "1e43be52b088ec840ff81865074b6d459b5ca7813b46...",
      "key_exchange_payload_length": 256,
```

```
      "nonce_payload": "04d78293ead007bc1a0f0c6c821a3515286a935af12ca50e08905b15d6c8fcd4",
      "nonce_payload_length": 32
    },
    "vendor_ids": [
      "4048b7d56ebce88525e7de7f00d6c2d3",
      "4a131c81070358455c5728f20e95452f",
      "afcad71368a1f1c96b8696fc77570100",
      "7d9419a65310ca6f2c179d9215529d56",
      "cd60464335df21f87cfdb2fc68b6a448",
      "90cb80913ebb696e086381b5ec427b1f"
    ]
  },
}
```

15.1.2.23. Event type: Modbus

15.1.2.23.1. Common fields

- "id": The unique transaction number given by Suricata

15.1.2.23.2. Request/Response fields

- "transaction_id": The transaction id found in the packet
- "protocol_id": The modbus version
- "unit_id": ID of the remote server to interact with
- "function_raw": Raw value of the function code byte
- "function_code": Associated name of the raw function value
- "access_type": Type of access requested by the function
- "category": The function code's category
- "error_flags": Errors found in the data while parsing

15.1.2.23.3. Exception fields

- "raw": Raw value of the exception code byte
- "code": Associated name of the raw exception value

15.1.2.23.4. Diagnostic fields

- "raw": Raw value of the subfunction code bytes
- "code": Associated name of the raw subfunction value
- "data": Bytes following the subfunction code

15.1.2.23.5. MEI fields

- "raw": Raw value of the mei function code bytes
- "code": Associated name of the raw mei function value
- "data": Bytes following the mei function code

15.1.2.23.6. Read Request fields

- "address": Starting address to read from
- "quantity": Amount to read

15.1.2.23.7. Read Response fields

- "data": Data that was read

15.1.2.23.8. Multiple Write Request fields

- "address": Starting address to write to
- "quantity": Amount to write
- "data": Data to write

15.1.2.23.9. Mask Write fields

- "address": Starting address of content modification
- "and_mask": And mask to modify content with
- "or_mask": Or mask to modify content with

15.1.2.23.10. Other Write fields

- "address": Starting address to write to
- "data": Data to write

15.1.2.23.11. Generic Data fields

- "data": Data following the function code

15.1.2.23.12. Example

Example of Modbus logging of a request and response:

```
"modbus": {
  "id": 1,
  "request": {
    "transaction_id": 0,
    "protocol_id": 0,
    "unit_id": 0,
    "function_raw": 1,
    "function_code": "RdCoils",
    "access_type": "READ | COILS",
    "category": "PUBLIC_ASSIGNED",
    "error_flags": "NONE",
  },
  "response": {
    "transaction_id": 0,
    "protocol_id": 0,
    "unit_id": 0,
    "function_raw": 1,
    "function_code": "RdCoils",
    "access_type": "READ | COILS",
    "category": "PUBLIC_ASSIGNED",
    "error_flags": "DATA_VALUE",
  },
}
```


15.1.2.24. Event type: QUIC

15.1.2.24.1. Fields

- "version": Version of the QUIC packet if contained in the packet, 0 if not
- "cyu": List of found CYUs in the packet
- "cyu[].hash": CYU hash
- "cyu[].string": CYU string
- "ja3": The JA3 fingerprint consisting of both a JA3 hash and a JA3 string
- "ja3s": The JA3S fingerprint consisting of both a JA3 hash and a JA3 string
- "ja4": The JA4 client fingerprint for QUIC

15.1.2.24.2. Examples

Example of QUIC logging with CYU, JA3 and JA4 hashes (note that the JA4 hash is only an example to illustrate the format and does not correlate with the others):

```
"quic": {
  "version": 1362113590,
  "cyu": [
      {
          "hash": "7b3ceb1adc974ad360cfa634e8d0a730",
          "string": "46,PAD-SNI-STK-SNO-VER-CCS-NONC-AEAD-UAID-SCID-TCID-PDMD-SMHL-ICSL-
NONP-PUBS-MIDS-SCLS-KEXS-XLCT-CSCT-COPT-CCRT-IRTT-CFCW-SFCW"
      }
  ],
  "ja3": {
      "hash": "324f8c50e267adba4b5dd06c964faf67",
      "string": "771,4865-4866-4867,51-43-13-27-17513-16-45-0-10-57,29-23-24,"
  },
  "ja4": "q13d0310h3_55b375c5d22e_cd85d2d88918"
}
```


15.1.2.24.3. Output Reference

15.1.2.24.3.1. Top Level (object)

Name	Type	Description
cyu	array of objects	JA3-like fingerprint for versions of QUIC before standardization
extensions	array of objects	list of extensions in hello
ja3	object	JA3 from client, as in TLS
ja3s	object	JA3 from server, as in TLS
ja4	string	
sni	string	Server Name Indication
ua	string	User Agent for versions of QUIC before standardization
version	string	Quic protocol version

15.1.2.24.3.2. ja3s (object)

Name	Type	Description
hash	string	JA3s hex representation
string	string	JA3s string representation

15.1.2.24.3.3. ja3 (object)

Name	Type	Description
hash	string	JA3 hex representation
string	string	JA3 string representation

15.1.2.24.3.4. extensions (array of objects)

Name	Type	Description
name	string	Human-friendly name of the extension
type	integer	Integer identifier of the extension
values	array of strings	Extension values

15.1.2.24.3.5. cyu (array of objects)

Name	Type	Description
hash	string	CYU hash hex representation
string	string	CYU hash string representation

15.1.2.25. Event type: DHCP

The default DHCP logging level only logs enough information to map a MAC address to an IP address. Enable extended mode to log all DHCP message types in full detail.

When a DHCP message carries the Option Overload entry (option 52, RFC 2132), the BOOTP `sname` and `file` header fields are used as extra option storage. Suricata parses any options found in those continuation areas alongside the standard options block, so values carried in either area show up in the same EVE fields below.

15.1.2.25.1. Fields

- "type": message type (e.g. request, reply)
- "id": DHCP transaction id
- "client_mac": client MAC address
- "assigned_ip": IP address given by DHCP server
- "client_ip": client IP address
- "dhcp_type": DHCP message type
- "client_id": DHCP client identifier
- "hostname": DHCP client host name
- "params": DHCP parameter request list

- "requested_ip": DHCP client requesting specific IP address
- "relay_ip": BOOTP relay agent IP address
- "next_server_ip": BOOTP next IP address to use for booting process
- "subnet_mask": subnet mask to use with client IP address
- "routers": IP address(es) to be used as default gateways on DHCP client
- "lease_time": Duration of IP address assignment to client
- "renewal_time": Time in seconds since client began IP address request or renewal process
- "rebinding_time": Time in seconds before the client begins to renew its IP address lease
- "dns_servers": IP address(es) of servers the client will use for DNS queries

15.1.2.25.2. Examples

Example of DHCP log entry (default logging level):

```
"dhcp": {
  "type":"reply",
  "id":755466399,
  "client_mac":"54:ee:75:51:e0:66",
  "assigned_ip":"100.78.202.125",
  "dhcp_type":"ack",
  "renewal_time":21600,
  "client_id":"54:ee:75:51:e0:66"
}
```

Example of DHCP log entry (extended logging enabled):

```
"dhcp": {
  "type":"reply",
  "id":2787908432,
  "client_mac":"54:ee:75:51:e0:66",
  "assigned_ip":"192.168.1.120",
  "client_ip":"0.0.0.0",
  "relay_ip":"192.168.1.1",
  "next_server_ip":"0.0.0.0",
  "dhcp_type":"offer",
  "subnet_mask":"255.255.255.0",
  "routers":["192.168.1.100"],
  "hostname":"test",
  "lease_time":86400,
  "renewal_time":21600,
  "rebinding_time":43200,
  "client_id":"54:ee:75:51:e0:66",
  "dns_servers":["192.168.1.50","192.168.1.49"]
}
```

15.1.2.26. Event type: ARP

15.1.2.26.1. Fields

- "hw_type": network link protocol type
- "proto_type": internetwork protocol for which the request is intended
- "opcode": operation that the sender is performing (e.g. request, response)
- "src_mac": source MAC address
- "src_ip": source IP address
- "dest_mac": destination MAC address
- "dest_ip": destination IP address

15.1.2.26.2. Examples

Example of ARP logging: request and response

```
"arp": {
  "hw_type": "ethernet",
  "proto_type": "ipv4",
  "opcode": "request",
  "src_mac": "00:1a:6b:6c:0c:cc",
  "src_ip": "10.10.10.2",
  "dest_mac": "00:00:00:00:00:00",
  "dest_ip": "10.10.10.1"
}
```

```
"arp": {
  "hw_type": "ethernet",
  "proto_type": "ipv4",
  "opcode": "reply",
  "src_mac": "00:1a:6b:6c:0c:cc",
  "src_ip": "10.10.10.2",
  "dest_mac": "00:1d:09:f0:92:ab",
  "dest_ip": "10.10.10.1"
}
```

15.1.2.27. Event type: POP3

15.1.2.27.1. Fields

- "request" (optional): a request sent by the pop3 client

    - "request.command" (string): a pop3 command, for example "USER" or "STAT", if unknown but valid *UnknownCommand* event will be set
    - "request.args" (array of strings): pop3 command arguments, if incorrect number for command *IncorrectArgumentCount* event will be set

- **"response" (optional): a response sent by the pop3 server**

    - "response.success" (boolean): whether the response is successful, ie. +OK
    - "response.status" (string): the response status, one of "OK" or "ERR"
    - "response.header" (string): the content of the first line of the response
    - "response.data" (array of strings): the response data, which may contain multiple lines

Example of POP3 logging:

```
"pop3": {
    "request": {
        "command": "USER",
        "args": ["user@example.com"],
    },
    "response": {
        "success": true,
        "status": "OK",
        "header": "+OK password required for \"user@example.com\"",
        "data": []
    }
 }
```

15.1.2.28. Event type: Netflow

15.1.2.28.1. Fields

- "age": duration of the flow (measured from timestamp of last packet and first packet)
- "bytes": total number of bytes to client
- "end": date of the end of the flow
- "max_ttl": maximum observed Time-To-Live (TTL) value
- "min_ttl": minimum observed TTL value
- "pkts": total number of packets to client
- "start": date of start of the flow
- "tx_cnt": number of transactions seen in the flow (only present if flow has an application layer)

Example

```
"netflow": {
   "pkts": 1,
   "bytes": 160,
   "start": "2013-02-26T17:02:42.907340-0500",
   "end": "2013-02-26T17:02:42.907340-0500",
   "age": 0,
   "min_ttl": 1,
   "max_ttl": 1
 }
```