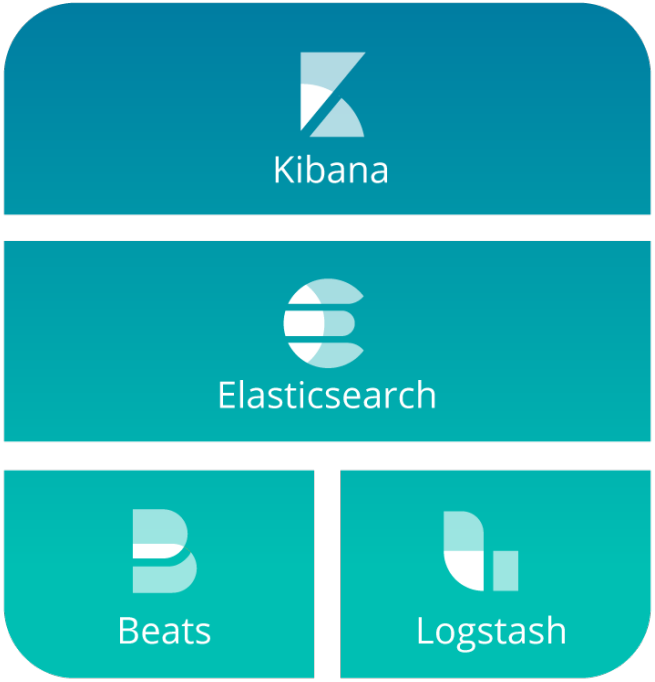
From Raw Logs to Real Insights

Getting started with log analytics using the Elastic Stack

Robert Cowart
Solutions Architect

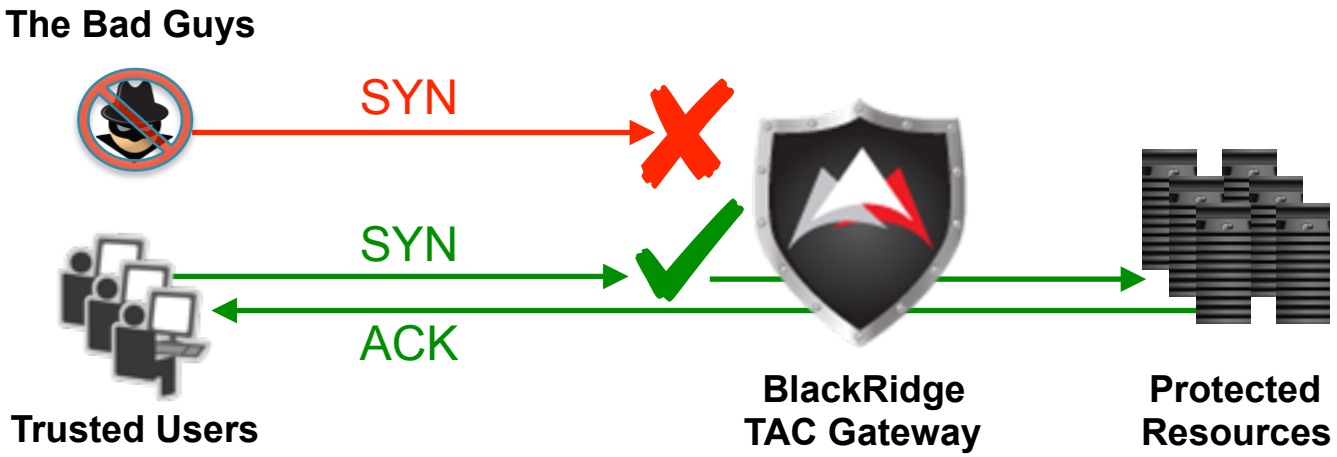
Elastic Stack

100% open source
No enterprise edition



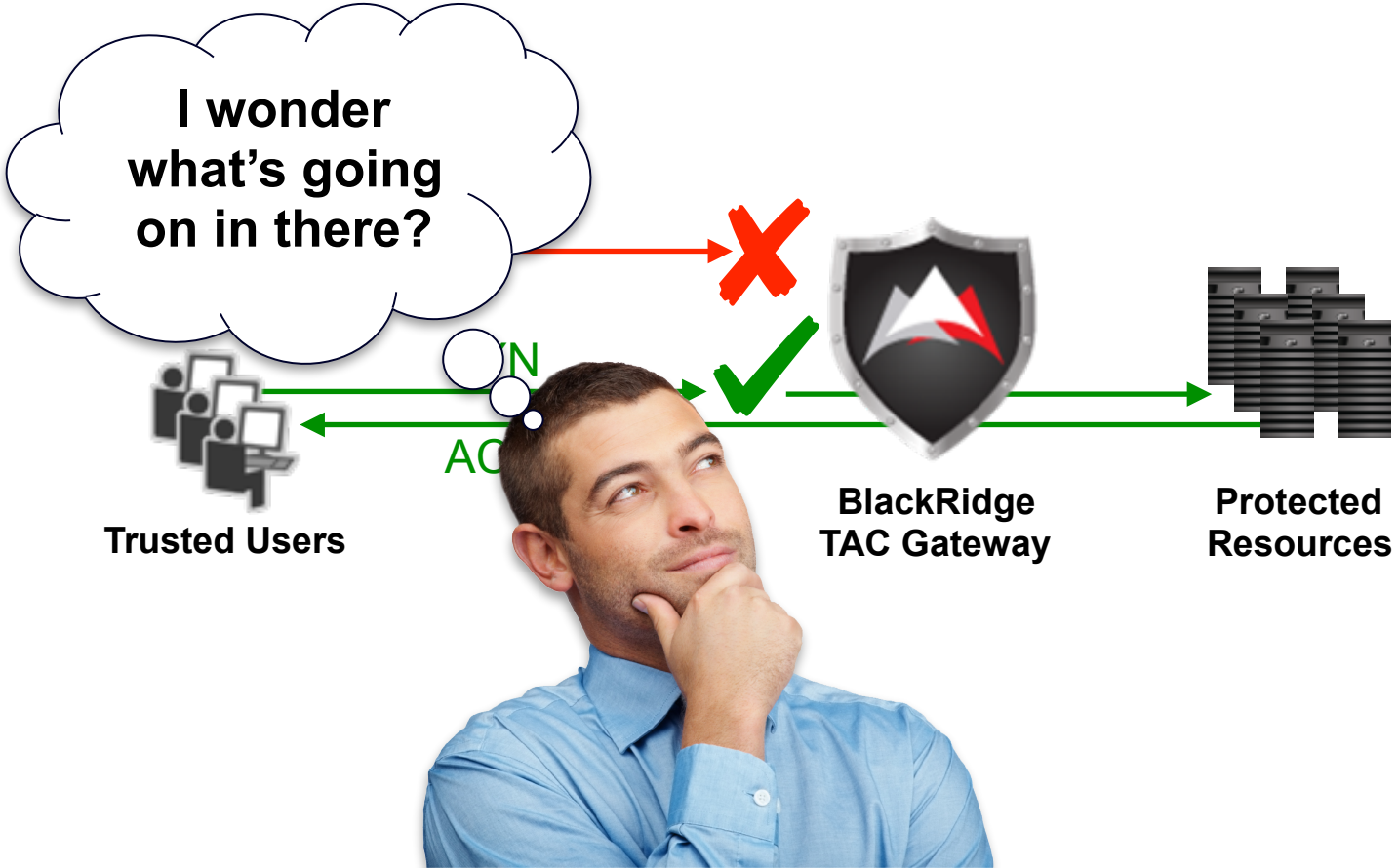
elastic

Our sample data… from the BlackRidge TAC Gateway



elastic

Our sample data… from the BlackRidge TAC Gateway



elastic

A look at BlackRidge Message Format (BMF)

<5>Oct 18 12:57:30 BRDC-2 kernel: [BlackRidge|Gateway|3.0.0.4619] class="Attribution" category="Unknown Identity" ctx="bump0" src="125.33.12.234" srcPort="25654" dest="5.149.112.53" destPort="23" identity="honeypot2Id" gwAction="DISCARD" gwMode="Monitor"

What's included in BlackRidge Message Format (BMF)?

Syslog header:

<5>Oct 18 12:57:30 BRDC-2 kernel:

BMF Header:

[BlackRidge|Gateway|3.0.0.4619]

BMF Payload:

class="Attribution" category="Unknown Identity" ctx="bump0" src="125.33.12.234" srcPort="25654" dest="5.149.112.53" destPort="23" identity="honeypot2Id" gwAction="DISCARD" gwMode="Monitor"

Ingesting data into the Elastic Stack

pipeline

parse, format and enrich the data

index template

specify how the data should be indexed

index pattern & dashboards

view and analyze the data

Building the Pipeline

The structure of a Logstash pipeline

```
input {

}

filter {

}

output {

}
```

From where we get the data.

How we parse, format and enrich the data.

Where we send the data.

Listen for syslog but read from a file for development

```
input {
  udp {
    type => "syslog"
    port => "${ESLOG_SYSLOG_PORT:514}"
  }

  tcp {
    type => "syslog"
    port => "${ESLOG_SYSLOG_PORT:514}"
  }

  file {
    path => "${ESLOG_BASE}/logs/dev.syslog"
    sincedb_path => "/dev/null"
    start_position => "beginning"
    ignore_older => 0
  }
}
```

Environment variables can make pipelines more portable.

We will use sample data from a file as we develop the pipeline.

These settings tell logstash to reread the file each time it is run.

This is for development, NOT production.

Output to stdout

```
output {
  stdout {
    codec => rubydebug {
      metadata => true
    }
  }
}
```

For now we only care about seeing the results of our pipeline configuration, so we will send the data to stdout.

The rubydebug codec formats the output for better readability

By default @metadata fields are not output, but we can specify that we want to see them.

Run Logstash (01_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/01_blackridge.logstash.conf
```

RESULT:

```
{
  "path" => "/Users/rob/src/eslog_tutorial/logs/dev.syslog",
  "@timestamp" => 2017-10-23T06:02:29.528Z,
  "@metadata" => {
    "path" => "/Users/rob/src/eslog_tutorial/logs/dev.syslog",
    "host" => "ws5.local"
  },
  "@version" => "1",
  "host" => "ws5.local",
  “message" => "<5>Oct 18 12:57:30 BRDC-2 kernel: [BlackRidge|Gateway|3.0.0.4619] class=\"Attribution\"
category=\"Unknown Identity\" ctx=\"bump0\" src=\"125.33.12.234\" srcPort=\"25654\" dest=\"5.149.112.53\"
destPort=\"23\" identity=\"honeypot2Id\" gwAction=\"DISCARD\" gwMode=\"Monitor\""
}
```

It would be better if this was the the timestamp from the received message.

This is the host that forwarded the message, not necessarily from where it originated.

message is the raw data that was received.

Does this look like a syslog message?

```
filter {
  grok {
    match => {
      "message" => "(?:<%{INT:syslog_pri}>\s*)?(?:%{SYSLOGTIMESTAMP:[@metadata][syslog_timestamp]}|%
{TIMESTAMP_ISO8601:[@metadata][syslog_timestamp]})\s+(?:%{SYSLOGFACILITY}\s+)?%{IPORHOST:logging_host}\s+%
{PROG:logging_process}(?:\[%{INT:logging_pid}\])?:\s+%{GREEDYDATA:logged_message}"
    }
  }
  if "_grokparsefailure" in [tags] {
    drop { }
  }
}
```

Let's see if this looks like a syslog message

This will be the BMF formatted message

This is from where the message originated

We will drop messages that don't look like properly formatted syslog. You may want to handle them differently depending on your needs.

TO BE CONTINUED

Run Logstash (02_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/02_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"logging_host" => "BRDC-2",
"logging_process" => "kernel",
"@metadata" => {
  "syslog_timestamp" => "Oct 18 12:57:30"
},
"logged_message" => "[BlackRidge|Gateway|3.0.0.4619] class=\"Attribution\" category=\"Unknown Identity\"
ctx=\"bump0\" src=\"125.33.12.234\" srcPort=\"25654\" dest=\"5.149.112.53\" destPort=\"23\"
identity=\"honeypot2Id\" gwAction=\"DISCARD\" gwMode=\"Monitor\"",
“syslog_pri" => "5"
```

We can use this to set *@timestamp*

syslog_pri can be further decoded

The BMF formatted message, which we will process further.

Decode *syslog_pri*

```
else {
  syslog_pri { }
}
```

It looks like we got a syslog message so let's decode *syslog_pri*.

Since we stored the syslog priority value in a field named *syslog_pri*, we don't have to specify any options.

TO BE CONTINUED

Run Logstash (03_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/03_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"syslog_facility_code" => 0,
"syslog_facility" => "kernel",
"syslog_severity_code" => 5,
"syslog_severity" => "notice",
```

You may want to use the color formatter within the Kibana index pattern definition to display different severity values in different colors.

Set @timestamp to the time from the raw message

```
date {
  locale => "en"
  match => [ "[@metadata][syslog_timestamp]", "MMM  d HH:mm:ss", "MMM dd HH:mm:ss", "MMM dd YYYY HH:mm:ss",
"MMM  d YYYY HH:mm:ss", "ISO8601" ]
  timezone => "${ESLOG_SYSLOG_TZ:UTC}"
}
```

TO BE CONTINUED

Ideally we would always get a timestamp in UTC, but just in case we need to change it, let's make it configurable via an environment variable.

Syslog timestamps can come in a lot of different formats. Specify the common ones here to make the pipeline more adaptable.

Run Logstash (04_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/04_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"@timestamp" => 2017-10-18T12:57:30.000Z
```

@timestamp is now set to the timestamp from the raw syslog message.

Everything up to this point was handling of the syslog-specific aspects of the message. You may want to reuse this as the foundation for all of your syslog sources.

Does this look like a BlackRidge BMF message?

```
grok {
  patterns_dir => [ "${ESLOG_GROK_PATTERNS_DIR:/etc/logstash/patterns}" ]
  match => { "logged_message" => "%{BMF_MSG}" }
  add_tag => [ "blackridge" ]
}
if "_grokparsefailure" in [tags] {
  drop { }
}
```

This time we are using an external grok patterns file instead of an inline pattern.

BMF_MSG is defined in an external patterns file.

```
BMF_MSG ^\[[BL][lE][aE][cF]%{GREEDYDATA}\]\s+%{GREEDYDATA:[@metadata][bmf_payload]}$
```

TO BE CONTINUED

Run Logstash (05_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/05_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"@metadata" => {
  "bmf_payload" => "class=\"Attribution\" category=\"Unknown Identity\" ctx=\"bump0\" src=\"125.33.12.234\"
srcPort=\"25654\" dest=\"5.149.112.53\" destPort=\"23\" identity=\"honeypot2Id\" gwAction=\"DISCARD\"
gwMode=\”Monitor\""
},
"tags" => [
  [0] "blackridge"
],
```

The *@metadata.bmf_payload* field now contains the key-value portion of the BMF message.

Tags are useful for many purposes, including controlling the how the message is processed.

Transform the key-value pairs into fields

```
else {
  kv {
    source => "[@metadata][bmf_payload]"
    trim_key => "\""
    trim_value => "\""
  }
}
```

The *kv* filter adds a field for each key-value pair.

We can use the *trim_key* and *trim_value* options to remove the quotes from the message

TO BE CONTINUED

Run Logstash (06_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/06_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"src" => "125.33.12.234",
"srcPort" => “25654”,
"dest" => "5.149.112.53",
"destPort" => “23",
"class" => "Attribution",
"category" => "Unknown Identity",
"ctx" => "bump0",
"gwAction" => "DISCARD",
"gwMode" => "Monitor",
"identity" => "honeypot2Id",
```

Wouldn’t it be interesting to know where this IP is located?

Do users know which service is being accessed? We can make this more user-friendly.

The *gwAction* field is another good candidate for the color formatter in Kibana.

Was the access attempt internal or external?

```
if [src] {
  cidr {
    address => [ "%{src}" ]
    network => [ "10.0.0.0/8", "172.16.0.0/12", "192.168.0.0/16", "fc00::/7", "127.0.0.0/8", "::
1/128","169.254.0.0/16", "fe80::/10","224.0.0.0/4", “ff00::/8", "255.255.255.255/32" ]
    add_field => { "src_locality" => "private" }
  }
}
```

TO BE CONTINUED

The *cidr* filter determines if an IP address is within a specified subnet. Let’s check all private IP blocks.

Run Logstash (07_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/07_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"src_locality" => "private"
```

If the *src* is in fact a private IP then you would see the above output. However in the sample event we are working with the *src* is a public IP address.

From where did the access attempt come?

```
if ![src_locality] {
  mutate {
    add_field => { "src_locality" => "public" }
  }
  geoip {
    source => "src"
    database => "${ESLOG_GEOIP_DBS_DIR:/etc/logstash/geoipdbs}/GeoLite2-City.mmdb"
    target => "geoip"
  }
  geoip {
    source => "src"
    database => "${ESLOG_GEOIP_DBS_DIR:/etc/logstash/geoipdbs}/GeoLite2-ASN.mmdb"
    target => "geoip"
  }
}
```

Since the *src_locality* isn't private it must be public.

Lookup the IP in the City DB for its geo-location.

Lookup the IP in the ASN DB to determine the public network space (the autonomous system) from which it originated.

TO BE CONTINUED

The *geoip* filter includes the City and ASN DBs, but if you want to update them more regularly or use the commercial City and ISP DBs it is useful to maintain them in their own directory.

Run Logstash (08_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/08_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"geoip" => {
  “ip" => "125.33.12.234",
  "longitude" => 116.3883,
  "latitude" => 39.9289,
  "continent_code" => "AS",
  "city_name" => "Beijing",
  "country_name" => "China",
  "country_code2" => "CN",
  "country_code3" => "CN",
  "region_code" => "11",
  "region_name" => "Beijing",
  "timezone" => “Asia/Shanghai",
  "location" => {
    "lon" => 116.3883,
    "lat" => 39.9289
  },
  "as_org" => "China Unicom Beijing Province Network",
  "asn" => 4808
}
```

The geoip.country_code2 field will allow us to visualize the data on Kibana’s vector map visualization.

The *geoip.location* object will allow us to visualize the data on a tilemap.

Let's make the *destPort* more user-friendly!

```
if [src] {
  ...
}

if [destPort] {
  translate {
    dictionary_path => "${ESLOG_DICTIONARY_PATH:/etc/logstash/dictionaries}/iana_service_names_tcp.yml"
    field => "destPort"
    destination => "service"
    fallback => "__UNKNOWN"
  }
  if [service] == "__UNKNOWN" {
    mutate {
      replace => { "service" => "%{[destPort]}" }
    }
  } else {
    mutate {
      replace => { "service" => "%{[service]} (%{[destPort]})" }
    }
  }
}
```

If the *destPort* exists let's create a more user-friendly *service* field.

The translate filter can lookup values in an external dictionary file. Here we use one made from IANA's registry of common services.

If we don't find a service name in the dictionary then let's set the *service* to the *destPort*.

If we do find a service name in the dictionary then let's construct the *service* to from the service name and the *destPort*.

Run Logstash (09_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/09_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"service" => "telnet (23)"
```

The resulting *service* field provides a much more user-friendly representation than the *destPort*.

Time to send our data to Elasticsearch

```
output {
  #stdout {
  #  codec => rubydebug {
  #    metadata => true
  #  }
  #}

  if "blackridge" in [tags] {
    elasticsearch {
      hosts => [ "${ESLOG_ELASTICSEARCH_HOSTS:127.0.0.1:9200}" ]
      user => "${ESLOG_ELASTICSEARCH_USER}"
      password => "${ESLOG_ELASTICSEARCH_PASSWORD}"
      index => "blackridge-%{+YYYY.MM.dd}"
    }
  }
}
```

Comment out the *stdout* output.

Only send to Elasticsearch if it was identified as a BlackRidge BMF message.

Specify the Elasticsearch server to connect to, including any needed credentials.

Specify the index name. Here we are using daily indexes.