Finding Needles in Haystacks (the Size of Countries)

By Michael Baker, David Turnbull, Gerald Kaszuba

Abstract

The lament of security analysts is often a limitation in the amount of data they can process, and the ensuing loss of data fidelity as size increases. As data sets grow they become unwieldy, making it difficult to add context through correlating security event data with other relevant data sets.

Full packet capture provides a method for maintaining a forensic copy of all network conversations. However the reality up until now is that full packet capture and analysis has been bounded by the size of the data, the time to process it and the ability of applications and tools to encode key attack, deviations, misuse and anomaly data into visualisations.

When you can store all of your network data the issue then becomes how do you analyse it. How do you find the single conversation you are looking for in trillions of conversations?

Big Data has supplied both a method for parallel computation and at the same time the cost of storing all network data (full packet capture) is within reach of all organisations. At the same time threats are becoming more blended, complex and difficult to find. Big Data tools such as Apache Hadoop, PIG and NoSQL databases provide the ability to perform complex network traffic analysis at petabyte scale. These tools can be leveraged using the Amazon Cloud (Elastic Map Reduce) to process, query and persist packet capture data.

With these tools there is no time-cost trade off to analysing every single conversation on a network, enriching the data, intersecting data sets and sharing anonymized data sets.

Network Security Monitoring

Network Security Monitoring as a philosophy looks at the detection of threats as being just as important as the prevention. Often when attackers breach strong defenses there is little information to rely on to understand how they exploited the network, how long they were inside and what they stole or accessed.

Network Security Monitoring advocates defensible and inventoried networks and the use of network packet data (packet captures) to provide as much context in relation to attacks. Full packet capture involves storing all network communication for threat analysis, transaction analysis and to provide a complete record of events taking place on the network.

Full packet capture is often referred to as the “network flight recorder” or “network tivo”. It provides the ultimate context and the complete record of

events and many open source tools provide security analysts with the means to analyze packet captures. These include Snort, Bro, Squil, Flowgrep and Wireshark. These tools provide an atomic view of packet data and are generally used when the attack window is known. However they have difficulty analyzing days, weeks and months of network data that scales into the trillions of packets, conversations and terabytes of data.

Last year, 2011 was a watershed year in terms of the number of publicized attacks affecting major corporations and highly regarded security vendors themselves. It left the impression in our minds that anyone can be subverted and their intellectual property stolen, their internal communications publicized and their brand and image tarnished.

It can be seen that a focus on prevention and the lack of a detection and incident toolset led to breaches being undetected for long periods of time allowing deeper inroads to be made and breaches to be more severe. The idea of strong defenses keeping out the attackers is similar to most enterprise security postures. They are defending the vault, trying to identify all that come in and out of each location. The alternative is to look at security like a casino or the stock market where everyone can play or transact and security is built through transaction analysis, monitoring and surveillance.

Out of 2011 came the positive security story of Netwitness and the detection of the breach. A technology exhibiting the strengths of Network Security Monitoring was used to detect and understand the magnitude of the breach. This became a positive story for RSA compared to others that had breaches of similar severity.

When we look at how enterprises structure their defenses it's easy to see how they lose the fidelity of information. In our experience they analyze network information using a copy of a copy. Device logs that have been stored on a central or at best collected by a Security Information and Event Management system.

Lastly the security landscape is affected by scale. Terabytes of data, trillions of conversations lead to an inherent security analysis compromise. At some stage you must sacrifice fidelity and accuracy for aggregation to conserve resources on systems that weren't built to handle the explosion of information.

How do you provide the clues security analysts need to detect and investigate breaches in the era of Big Data?

Parallel Analysis and Storage

This is a brief history of MapReduce, Hadoop and Pig. In 2004 Google published their paper on MapReduce and in 2006 Bigtable - the Google File System. Hadoop was formed as an open source project to implement a number of the ideas from these papers. Hadoop provided the Hadoop File System (HDFS) and also an implementation of MapReduce that could be accessed using Java or other methods (like Hadoop Streaming).

Data could be stored across a number of commodity nodes and analysis could be distributed across these nodes to reduce the time taken to process the data.

Pig was started as an Apache project to create a data flow language as an alternative to the complexity of MapReduce. Pig's language is PigLatin a high level dataflow language that is much easier to use than programming MapRedice.

Pig translates PigLatin into MapReduce jobs that operate across a Hadoop cluster accessing data stored on an HDFS file system.

Packetpig

Over the last year we have been working to launch Packetloop, a Big Data Security Analytics platform that provides unique insights into network packet captures. This involved authoring numerous MapReduce jobs that operate across packet captures and then store results in a column-oriented database. A lot of what we learnt we wanted to give back the security community.

We created Packetpig an open source network security monitoring platform that would allow anyone to analyze terabytes of network packet captures either on their own commodity hardware or on Amazon's Elastic Map Reduce (EMR) and S3 storage.

Packetpig is made up of Pig Loaders, Pig User Defined Functions (UDF's), existing open source tools and libraries that allow anyone to analyze large data sets.

Packetpig also provides methods to visualize the data that is created by running Pig queries across packet captures. We use the R statistical programming language to provide statistical analysis and also to plot output. We also provided a number of examples of how these visualizations could be transitioned to the web using the D3 JavaScript library and the Google WebGL Globe.

The key to Packetpig is allowing people who aren't subject matter experts in MapReduce or distribute processing to create queries against a small amount of packet capture data on their laptop and then transition that to a cluster to run across a large number of commodity nodes. All jobs are capable of being run on a laptop against packet captures and then pushed to a Hadoop cluster. This is how we work at Packetloop for prototyping ideas and analyzing data sets that are provided to us by customers.

It also allows anyone to contribute dataflow queries, visualizations (in R or D3 or anything else), custom loaders or wrap existing open source tools using the work we have done as an example.

The project is hosted at Github https://github.com/packetloop/packetpig we encourage you to download it play with it, fork it and contribute to it. If you want to contact us regarding the project please email us on packetpig@packetloop.com or follow the project on Twitter @packetpig.

We would also like to credit the work of others that we leveraged to build Packetpig.

- The Apache Projects Hadoop and Pig.
- Libpcap
- The Kraken PCAP project.
- Libnids and pynids which we use for conversation and file dissection.
- Libmagic for file type identification.
- Snort and p0f which we wrap to provide IDS and Passive OS detection.
- Maxming Geo Location.
- R, D3 and Google for the Google Globe.

What can Packetpig do?

Once data network packet data has been captured and stored it can be analyzed using Packetpig using any of the core features of the project;

- Packets – access to any field located in the IP header including TCP/UDP fields. This includes
    - IP Version
    - IP Header Length
    - IP TOS
    - IP Total Length
    - IP ID
    - IP Flags
    - IP TTL
    - IP Proto
    - IP Checksum
    - IP Source
    - IP Destination
    - TCP Source Port
    - TCP Destination Port
    - TCP Sequence ID
    - TCP ACK ID
    - TCP Offset
    - TCP NS, CWR, ECE, URG, ACK, PSH, RST, SYN, FIN
    - TCP Window
    - TCP Length.
    - UCP Source Port
    - UDP Destination Port
    - UDP Length
    - UDP Checksum
- Protocol Analysis
    - Using the Packetpig Packet Loader protocols can be grouped and filtered based on source port or destination port information as well as via the IP Protocol field.
    - Packetpig also provides the ability to perform ngram analysis across the data portion of the packet. This can be used to find distributions of ASCII characters in the data payload.
    - DNS and HTTP can be fully dissected with full access to all fields.
- Conversations and Flows

    - Conversation Loader provides the ability to identify flows. Currently we use a generic source, source port, destination, destination port every 60 seconds to identify flows.
    - The conversation loader can be used to identify conversations and then the inter-packet delay or time between packets.
    - The conversation loader also allows Packetpig to track conversations and extract mime types (see File Dissection).
- Threat Analysis
    - The Snort Loader allows Snort to be run distributed across the Hadoop Cluster. The loader provides access to the following Snort Data;.
    - Timestamp, Signature ID, Priority, Message, Protocol, Source IP, Source Port, Destination IP Destination Port.
- Geo-Location
    - Using the Maxmind java library Packetpig provides the ability to return.
    - Given an IP address return a Latitude and Longitude.
    - Given an IP address return an Autonomous System Number / Organization Name.
    - Give an IP address return the Country of origin.
- OS Fingerprinting
    - The fingerprint loader wraps p0f Version 3 (by Michal Zalewski). Allowing operating system, link/mtu, application names and uptime to be analyzed for and party in a conversation.
- File Dissection
    - Outputs the file name, mime type, extension.
    - A MD5, SHA1 and SHA256 hash is produced for each file.
    - Files can be dumped to HDFS to be accessed or further analyzed.

Building Blocks

Pig Loaders are written in Java and provide access to specific information in Packet Captures. The Loader can be written exclusively in Java or wrap Python scripts (e.g. lib/tcp.py) or Binary files (e.g. p0f, Snort). Loaders can be extended and new loaders created as part of Packetpig.

Packetpig is made up of the following loaders and User Defined Functions;

- PacketLoader() for low-level access to packet data and IP headers.
- ConversationLoader() access conversation information. The Source/Destination IP, Source/Destination Ports, how the conversation was ended, timestamps of each packet sent in the conversation and also the delay between packets.
- FingerprintLoader() – the wrapper for p0f.
- PacketNGramLoader() – allows ASCII Ngrams to be generated from packet data.
- SnortLoader() – the wrapper for Snort.
- DNSConversationLoader() – Deep packet inspection for DNS.
- HTTPConversationLoader() – Deep packet inspection for HTTP.

- Geoip
    - Given an IP address returns Lat/Long, ASN and Country information.

The Basics

The Packetpig project contains a detailed README.md for explanation of how to use the different Loaders, UDF's and example PigLatin queries. This whitepaper is likely to be out of data relatively quickly so refer to the Packetpig project on Github for the latest information.

The first step to analyzing packet captures is choosing a Loader or multiple Loaders to get the data that you want to analyze. A loader populates a variable in Pig with data according to a schema.

For example if you want full access to IP and TCP/UDP header information use the PacketLoader().

```
packets = load '$pcap' using
com.blackfoundry.pig.loaders.pcap.packet.PacketLoader() AS (
    ts,

    ip_version:int,
    ip_header_length:int,
    ip_tos:int,
    ip_total_length:int,
    ip_id:int,
    ip_flags:int,
    ip_frag_offset:int,
    ip_ttl:int,
    ip_proto:int,
    ip_checksum:int,
    ip_src:chararray,
    ip_dst:chararray,

    tcp_sport:int,
    tcp_dport:int,
    tcp_seq_id:long,
    tcp_ack_id:long,
    tcp_offset:int,
    tcp_ns:int,
    tcp_cwr:int,
    tcp_ece:int,
    tcp_urg:int,
    tcp_ack:int,
    tcp_psh:int,
    tcp_rst:int,
    tcp_syn:int,
    tcp_fin:int,
    tcp_window:int,
    tcp_len:int,

    udp_sport:int,
    udp_dport:int,
    udp_len:int,
    udp_checksum:chararray
);
```

Figure 1 – Extract of pig/examples/basic_packets.pig

After executing this statement in Pig the variable 'packets' is populated with integer, long and chararray's based on the schema defined. If you DUMP the packets variable you are presented with the packet data;

```
(1322645014,4,20,0,1500,42804,2,0,64,6,43874,192.168.0.19,97.90.192.1
11,50977,32982,1231665539,-
2129045419,8,0,0,0,0,1,0,0,0,0,10797,1448,0,0,0,0)
(1322645014,4,20,0,284,42805,2,0,64,6,45089,192.168.0.19,97.90.192.11
1,50977,32982,1231666987,-
2129045419,8,0,0,0,0,1,1,0,0,0,10797,232,0,0,0,0)
(1322645014,4,20,0,48,17448,0,0,117,17,4348,124.149.179.72,192.168.0.
19,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,30710,51413,28,59379)
```

Figure 2 - Dump of pig/examples/basic_packets.pig

Please note '-' in the table above are new lines, they won't appear in the console output.

Once you have access to the data that you want to analyze you can group it into keys, iterate over it using FOREACH statements, count, average, sum and find distinct values in it. These operators are the building blocks of the complex jobs you will want to run across all packet captures.

As a quick example let's group on destination port and find the number of packets that are related to this port.

Firstly group all packets from the PacketLoader() using the TCP destination port (tcp_dport)

```
packets = group packets by tcp_dport;
DUMP packets;
```

Figure 3 - Extract from pig/examples/basic_grouping.pig

This grouping reorders the entire data set with tcp_dport as the key and the packet data as the records.

Now let's count the number of packets we have seen to each destination TCP Port.

```
packets = FOREACH packets GENERATE group, COUNT(packets);
```

I have kept the variable as 'packets' because I am still manipulating the same dataflow. What this line states is that for each line in the dataset grouped by tcp_port regenerate the group (key is tcp_port) and count the number of packets referenced by the key. The output is shown below;

```
(0,331)
(80,116)
(32982,48)
(33104,1)
(33105,1)
(33106,5)
(33301,15)
(33483,1)
(33522,1)
(33523,1)
```

```
(34268,1)
(34299,43)
(39283,1)
(39284,1)
(39285,1)
(41789,13)
(41790,2)
(41791,3)
(41819,2)
(42513,1)
(42514,22)
(42582,3)
(42583,1)
(44746,1)
(50977,24)
(54222,6)
(54223,1)
(55944,1)
(56524,1)
(56543,8)
(58121,1)
(59113,1)
(59214,4)
(60640,1)
(60641,1)
(63964,1)
```
Figure 4 - Output of pig/examples/basic_grouping.pig

In the output above the first value is the key which is the destination TCP port and the second value is the number of packets found with the destination port set to key value. Values are automatically separated by a comma.

Often you will have a lot of data so you will want to order and limit. You can order the output above easily using the following statement;

```
ordered = ORDER packets by $1 desc;
```
Figure 5 - Output of pig/examples/basic_grouping.pig

The $1 is a little troubling at first but look at output in Figure 4 - Output of pig/examples/basic_grouping.pig. The group is the key and can be referenced as $0 and the value is the count of the packets and can be referenced in subsequent statements as $1. To avoid this confusion you can create a variable to reference the count by. As in the example below using 'count_packets';

```
packets = FOREACH packets GENERATE group, COUNT(packets) as
count_packets;

ordered = ORDER packets by count_packets desc;
```
Figure 6 - Referencing COUNT(packets) as a the variable 'count_packets'

A limit can then be applied to the ordered list;

```
packets = FOREACH packets GENERATE group, COUNT(packets) as
count_packets;

ordered = ORDER packets by count_packets desc;

limit_packets = LIMIT ordered 10;
```

```
DUMP limit_packets;
```

The output of dumping the limit statement is shown below;

```
(0,331)
(80,116)
(32982,48)
(34299,43)
(50977,24)
(42514,22)
(33301,15)
(41789,13)
(56543,8)
(54222,6)
```

Joining data from different datasets is performed frequently in Packetpig. Source and Destination IP addresses are often used as keys to perform joins between datasets. For example you may have a directory for yesterdays packet captures and a directory for todays packet captures. What you are trying to work out is how many source IP addresses were accessing your network yesterday and also today.

You can use the PacketLoader() to load yesterday's traffic (passing the directory as a parameter) and load today's traffic (passing the directory as a parameter). You can then group by ip_src and Join the two datasets together.

A full example is provided in pig/examples/basic_joining.pig.

Binning Time

Timestamps are output by all Loaders in Packetpig. The timestamp can be used to sort events into time bins. This is important for plotting time series data in applications that have difficulty handling lots of data points.

Data from Loaders can be binned in user defined periods (default is 60 seconds). An example of binning is pig/examples/binning.pig. The output is shown below;

```
(1312696740,773,690,2831)
(1312696800,107447,6140,124087)
(1312696860,7519,690,12057)
(1312696920,72544,5505,87069)
(1312696980,79735,950,87161)
(1312697040,161784,10319,189979)
(1312697100,3065,854,5931)
(1312697160,172780,5302,187080)
(1312697220,917747,2068,945607)
(1312697280,58431,5148,70295)
(1312697340,77068,969,84777)
(1312697400,102706,6203,121893)
(1312697460,1025,1008,3911)
(1312697520,111122,4984,127438)
```

Figure 7 - Output from pig/examples/binning.pig

The binning.pig script outputs a binned timestamp, the SUM of TCP packets lengths for that period, the SUM of UDP packets for that period and the total IP lengths for that binned period.

Threat Analysis

The SnortLoader() uses the pcap functionality of Snort (snort –r) to inspect packet captures across Hadoop nodes using Pig.

The SnortLoader() reads security events related to the packet capture and returns them using the following schema;

- Timestamp
- Signature ID
- Priority
- Message
- Protocol
- Src IP
- Destination IP
- Source Port
- Destination Port

An example of the SnortLoader() is provided in pig/examples/basic_snort.pig.

Traffic Analysis

Traffic analysis involves being able to identify sessions or flows between source and destination IP addresses and then extracting features out of those flows.

Some common features for analyzing traffic and protocols include;

- Packet size.
- Inter-packet delay.
- Ngram analysis of packet payloads.

Features of conversations such as these are often used to different one type of traffic from another (e.g. DNS and HTTP) but furthermore to infer or guess at what data is being conveyed in encrypted traffic or whether encrypted traffic is being tunneled across a network. In this way Network Security Monitoring is able to make assumptions based upon traffic analysis.

To support this analysis Packetpig is able to join conversations it finds with the raw packets that make up the conversation. Currently we use an approximation (which works well) of the source, source port destination, destination port 4-tuple every 60 seconds.

An example of tracking packets in conversations is pig/examples/conversations.pig which tracks all conversations and then outputs the total bytes transferred in the conversation.

An example of tracking inter-packet delay for each conversation is provided in pig/examples/conversation_packet_intervals.pig. It outputs every conversation, the number of packets and the delay between each packet.

For DNS and HTTP traffic Packetpig is capable of performing deep packet inspection and access all fields within the packet data.

Passive Operating System Detection

The FingerprintLoader() wraps Michal Zalewski's p0f utility to perform distributed passive operating system detection across a Hadoop cluster. It returns identical data to that of p0f using the following schema;

- Timestamp
- Source IP
- Source Port
- Destination IP
- Destination Port
- Operating System
- Application
- Distribution
- Language
- Params
- Raw_freq
- Raw_mtu
- Raw_sig
- Uptime

The FingerPrintLoader() can be used in conjunction with other Loaders such as the HttpConversationLoader() to provide some interesting information about the source IP addresses. For example you could get all fingerprints and all the user-agents from web requests and compare them for every source IP address. It is then possible to output the Source IP address, Operating System (from p0f) and the user-agent (from HTTPConversationLoader() to see whether the source user agent does not match their operating system and whether their user-agent is changing from request to request.

An example of this exact query can be found in pig/examples/p0f_http.pig

File Extraction and Hashing

PacketPig provides the ability to extract files and output the following information regarding files via the FileConversationLoader();

- Timestamp
- Source IP Address
- Source Port
- Destination IP Address
- Destination Port
- File type – information from libmagic, same as file command in Linux.
- Mime Type
- File Extension
- MD5 Hash
- SHA1 Hash
- SHA256 Hash

```
(1322689152,10.1.0.90,54837,199.181.254.21,80,GIF image data version
89a 182 x
```

```
130,image/gif,.gif,48f2294cba16cf7eaa5cc51201b91689,70d91d6f7f20a1755
cc1a46ee4b248e24db1c1dd,28ce7d3bb11d87eb2461d06bb237cffa1d221b1dc6c08
d46e0545c0e216a03af)
```

Figure 8 - Example output from pig/examples/extracted_files.pig

The FileConversationLoader() takes a number of parameters.

- Path – e.g. 'tmp' the directory in HDFS to dump all files to.
- Mime e.g. image/gif to only dump or report on GIF images.
- Hash e.g. MD5/SHA1/SHA256 to only report on specific hashes.

If the FileConversationLoader() is not passed the 'path' parameter then it will not dump the files. This is beneficial if you only want to perform a search and not actually dump the files.

The FileConversationLoader() can easily be used to track all files transferred to or from a particular IP address or transferred on a particular source or destination port. If combined with the Geoip UDF it is possible to search for files that are transferred to or from specific Countries or ISP (ASN's).

Future

- Sentiment analysis across files and also SMTP messages to find positive or negative terms and information.
- Build search indexes using Lucene that could be queried with something like Solr.
- Probability loader to determine whether a specific traffic flow is x% similar to all other flows of it's type.
- Expose features to Machine Learning algorithms.
- Extraction/hashing of files with zip/gzip files.

Works Cited

Bejtlich, Robert (2004, July 22). The Tao of Network Security Monitoring (Beyond Intrusion Detection).

Google Inc, Dean Jeffrey, Ghemawat Sanjay (2004, December) MapReduce: Simplified Data Processing on Large Clusters from http://research.google.com/archive/mapreduce.html

Zalewski, Michael (2012) p0f from http://lcamtuf.coredump.cx/p0f3/