**Security-Informatics:**

Analysis of Malware Connections to Command and Control Servers

|  |  |
| --- | --- |
| **Submitted by:** | **Ralph “Ray” Zupancic –** [**rrzupanc@iu.edu**](mailto:rrzupanc@iu.edu) |
| **Course :** | [FA15-BL-INFO-I590-34717](https://iu.instructure.com/courses/1491582) |

**TABLE OF CONTENTS**

[Introduction 1](#_Toc437268934)

[Coding platforms 1](#_Toc437268935)

[Data mining and Malware Detection 1](#_Toc437268936)

[Command and control 1](#_Toc437268937)

[Packet Captures 3](#_Toc437268938)

[Generating Packet Captures 4](#_Toc437268939)

[Data Collection Mediums 5](#_Toc437268940)

[PCAP Repositories 5](#_Toc437268941)

[Systems for malware Forensics 5](#_Toc437268942)

[False Positives 5](#_Toc437268943)

[System Modular Architecture 6](#_Toc437268944)

[Geographic Module 7](#_Toc437268945)

[Parsing PCAPs 7](#_Toc437268946)

[Visualization 10](#_Toc437268947)

[Bibliography 13](#_Toc437268948)

[Appendix 1 Definitions and Acronyms 1](#_Toc437268949)

[Appendix 2 Code Submissions 1](#_Toc437268950)

**TABLE OF TABLES**

[Table 1: Sensor Data Types 5](#_Toc437268906)

[Table 2: Software Modules for a Forensic Beaconing Detection Application 6](#_Toc437268907)

[Table 3 Appendix 1: Acronyms from this document 1](#_Toc437268908)

**TABLE OF FIGURES**

[Figure 1: Code example - Beacon to C&C server code simulation 2](#_Toc437268914)

[Figure 2: Beacon to command server simulation 2](#_Toc437268915)

[Figure 3: OSI 7-layer network model 3](#_Toc437268916)

[Figure 4: Format of a PCAP file (Gilman) 4](#_Toc437268917)

[Figure 5: Simulated beacon – with packet contents shown in ASCII dump at bottom pane 4](#_Toc437268918)

[Figure 6: Code example – IP src and dst parser based on dpkt 8](#_Toc437268919)

[Figure 7: dpkt parsing results for IP src and dst 8](#_Toc437268920)

[Figure 8: Code example – Python shell demonstration of the GeoLiteCity database 9](#_Toc437268921)

[Figure 9: Code example – geocoder routine to return lat and long 9](#_Toc437268922)

[Figure 10: output of latitudes, longitudes 10](#_Toc437268923)

[Figure 11: Code example – creating lists of lat longs with the count of occurrence using Python Collections Counter class (complete code in Appendix) 11](#_Toc437268924)

[Figure 12: Use the code above to generate data points to build an HTML file with javascript pointing to Google maps – see complete HTML file in Appendix 11](#_Toc437268925)

[Figure 13: Code example - Beacon Listener TCP Service 1](#_Toc437268926)

[Figure 14: Code example – function to get latitude and longitudes (geocodes) 1](#_Toc437268927)

[Figure 15: Code example – function to generate Google Map datapoints 2](#_Toc437268928)

[Figure 16: Code example – function to output HTML file footer for Google Maps 3](#_Toc437268929)

[Figure 17: Code example – function to output HTML file header for Google Maps 3](#_Toc437268930)

[Figure 18: Code example – Main driver routine for packet parser 4](#_Toc437268931)

[Figure 19: HTML File for Google Maps 4](#_Toc437268932)

# Introduction

This paper will explore a facet of network forensics using data analysis techniques where, in principle, the techniques can be applied to arbitrarily large data sets or “Big Data”. The particular problem of interest involves detection of malware that installs itself and beacons or connects to a Command and Control (C&C) server for instructions and/or to download appropriated data. The goal is to develop principles and concepts used to architect a data-mining system for network forensics, and to demonstrate the architectural requirements of the system along with selected code components.

The resulting sections will detail a component architecture with code development and demonstration. Of course, a complete implementation of such a system would take many thousands of lines of code as well as purchased subscriptions and interfaces to other products. However, the hope is that the architecture and direction can be made clear in this brief presentation.

# Coding platforms

The platform and code examples presented here are developed in the Python language: Python is proving to be one of the most, if not the most, popular and efficient languages for both data security analysis and Informatics in general. Black hat and white hat hackers use Python to code up attacks and forensic tools, and data scientists like both the compactness as well as the rich visualization tools available with Python for data analysis. More and more security researchers are publishing works based on python: noted forensics researcher and author Chet Hosmer (2014) writes

The development environment is free, the language is platform independent, the code is as easy to read and write as English, and support is vast and worldwide. In my opinion it should revolutionize the development of new cyber-crime, forensic and incident response-based solutions.

# Data mining and Malware Detection

## Command and control

Beaconing consists of systematically and periodically contacting a host system for purposes such as keep-alive or scheduled transmission. The working definition of malware beaconing can be stated as the following: the systematic and periodic transmission of data between a client computer and a Command and Control (C&C) server for the purposes of updating client software and/or uploading client network data to the host.

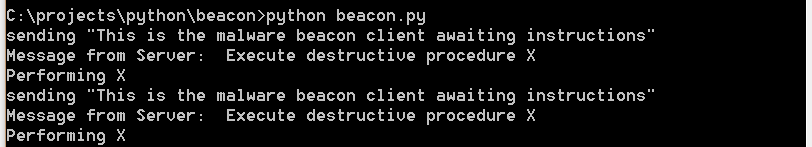
Installed malware can serve a variety of purposes and intents. The creator may aim to launch distributed denial of service (DDOS) attacks on a target—in this case the malware must contact the C&C to be updated with the target of choice. The malware may be designed to listen for keyboard input and collect banking or financial passwords—in this case the collected data must be uploaded to the C&C. Almost any modern malware will exhibit this feature of “phoning home”: this is a departure from older malware that would simply destroy files or flash banners. Malware can represent a significant investment of time and effort, and the creator generally wants to recoup that effort monetarily.

It is straight-forward to code up an example of a client beaconing to a C&C for simulation purposes: one need only setup a client-server system in Python where the client initiates the beacon on some schedule. As outlined, the client may both upload files from the local network and/or host, or it may simply fetch new or updated instructions from the C&C server. The code example below simulates a beacon client server while performing no action other than echoing messages back and forth, but this effectively simulates malware for purposes of data mining network forensics datasets for detection.

Figure : Code example - Beacon to C&C server code simulation



Figure : Beacon to command server simulation



It is worth noting the compactness and ease with which Python can setup listeners, connect to ports, scan networks, etc. The upshot is that Python is not only a great platform for performing network forensics, it is also a very handy tool for hackers to exploit a network if left available on hosts.

There are no professional features in this code snippet such as error handling for timed-out connection, encryption routines, or even modules that actually carry out instructions. Thus this example code is harmless, but an administrator would want to know if this code was set to periodically reach out across a network. The question to be addressed is: how might such activity on a network be detected and trigger an alert.

## Packet Captures

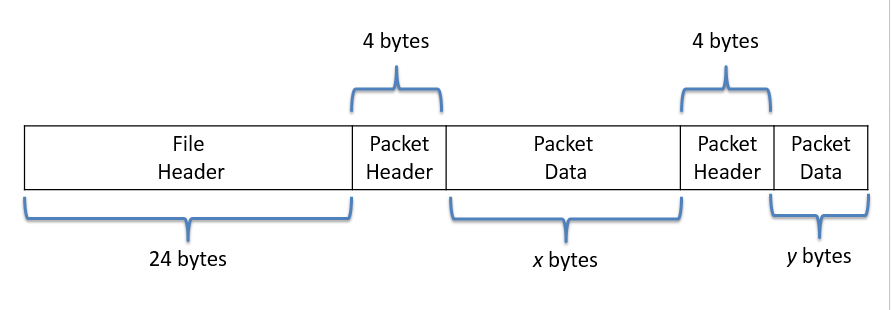
Integral to the detection of malware is a system for data mining packet capture results. A packet capture consists of a recording of data as it transits a network wire (e.g. Ethernet). A typical capture will break down data in a hierarchy similar to the OSI 7-layer model (depicted below) where different layers of the packet can be viewed individually and/or grouped into a flow according to the layer one is interested in. Packet captures can quickly grow to be very large data collections that are aggregated from several points on an Enterprise network: for example, perimeter routers along with certain key sensor points in an internal network can be captured and fed to a central set of programs or scripts that look for patterns and signatures. This can lead to Terabytes of data being captured in a very short amount of time. This is where” Big Data” analysis techniques can come in to play for performing network forensics.

Figure : OSI 7-layer network model



The most common format used by network administrators for storing collections of network data captures is libpcap (Wireshark Wiki, 2015). The file format used by libpcap is often simply referred to as .PCAP. Strictly speaking, PCAP is a generic abbreviation, however, any mention of a PCAP in this paper refers simply to a capture file in libpcap format. Tools such as Wireshark can save captures to PCAP files for distribution to other administrators or investigators: PCAPs are of great interest to administrators and forensic investigators attempting to detect or re-trace the steps used in a malware attack. PCAPs are binary files and a program is necessary to examine one: the size of packet captures makes it unwieldly to work with text formatted files.

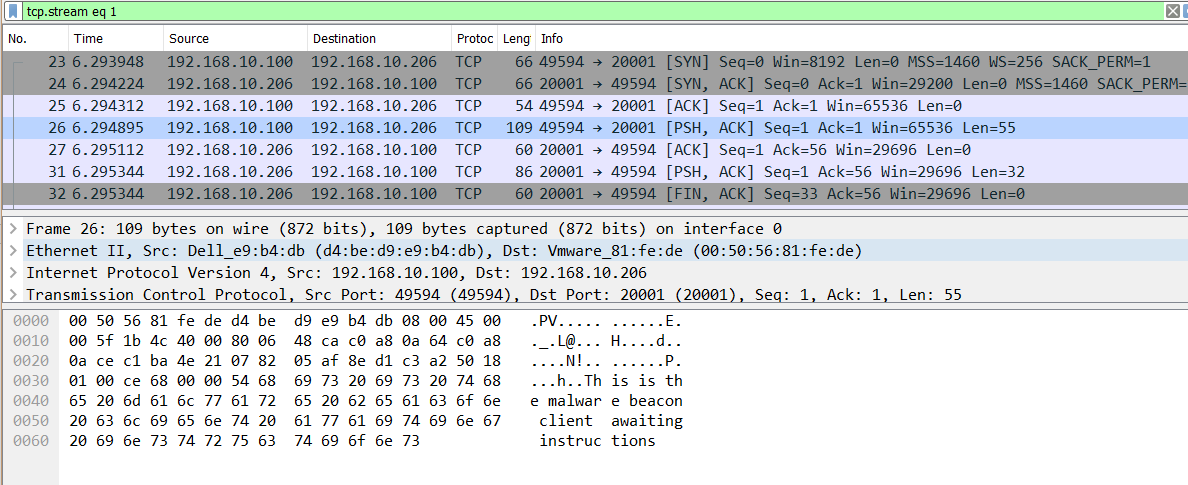
Figure : Format of a PCAP file (Gilman)



## Generating Packet Captures

The most granular tools for generating packet captures are wire sniffing tools such as WireShark (<https://www.wireshark.org/>) and tcpdump. Tcpdump is an open source command line utility that is often pre-installed on common Unix distributions and Wireshark is open source GUI-based software that can be downloaded to a variety of platforms such as Linux, Mac and Windows. An example of Wireshark examining the packets associated with the simulated beaconing code is presented below.

Figure : Simulated beacon – with packet contents shown in ASCII dump at bottom pane



Packet capture utilities function by placing the network interface card (NIC) hosting them into so-called promiscuous mode: while in this mode the NICs accept all packets that pass through the NIC, as opposed to the default (non-promiscuous) behavior of discarding all but those addressed to the actual host system. In addition to the NIC mode, it is usually necessary to configure network switches to forward traffic across the switch to a so-called SPAN (Switched Port Analyzer) port. There are a wide array of network forensic and detection tools that use packet sniffing to monitor the network wires. The open source Snort Network Intrusion Detection System (NIDS) is of special interest because it is widely deployed and can read PCAPs and apply user-defined rules and signatures to data mine PCAPs for beaconing and other malicious activity.

## Data Collection Mediums

In order to analyze network data there must be some design around the capturing of data. Typically, in an Enterprise network the Information Assurance team will have designed a mesh of traffic detectors that analyzes passing traffic and feeds results back into some type of central console. Packet captures are not the only type of data collected in this central repository; other examples are provided in the Sensor Data Types Table.

Table : Sensor Data Types

|  |  |
| --- | --- |
| **Source** | **Type** |
| firewalls | connections logs e.g. HTTP logs, mail logs, etc. |
| applications | application defined logs |
| network flow | Cisco NetFlow or competing network flow standard |
| host logs | Windows Event logs, Syslog |
| network device logs | switches, routers, load balancers |

## PCAP Repositories

There exist several repositories of malware-related PCAPs for researchers and students to study. This means that it is possible to simply download a series of interesting PCAP files and write code against them to test hypotheses and develop detection systems. One such repository is a blog site called Contagio maintained by Mila Parkour. The PCAP examples in this paper (except for the trivial beacon capture) use PCAPs from this Contagio.

# Systems for malware Forensics

## False Positives

The bane of any network administrator using an array of sensors to detect malicious or unauthorized users is the false positive. Sophisticated NIDS implementations are often so laden with false positive that they quickly become unusable as alerts pile up and administrators begin to ignore them. Tuning for false positives without introducing false negatives—the opposite condition of a false positive where activity that should trigger an alert fails to—is the full-time pursuit of many Security Operations (SecOps) specialists.

This is true with the detection of beaconing signals associated with malware: many normal and innocuous system processes will look like malware beaconing. For example, if a user has a mail client that periodically goes and out checks for new messages on a server, this is a process that is superficially indistinguishable from malware beaconing. Given this, a system for malware beaconing detection must address the question of false positives.

Fortunately, there are many tell-tale signs that distinguish malicious activity that can be gleaned from a dataset. These can be arranged as a feature matrix that might comprise architectural components in a detection system.

## System Modular Architecture

Laying out the features of a detection analysis system is simply a matter of observing that specific signs or tendencies that malware exhibits that can be mined in a sensor packet dump. The following is a proposed module architecture for building such a system

Table : Software Modules for a Forensic Beaconing Detection Application

|  |  |
| --- | --- |
| **Feature** | **Functionality** |
| Packet Parser | The core module is simply a packet parser: the packet parser will need to be extensible and flexible to work with all other components and provide access to the data stream to be analyzed. In this case, the data stream will be PCAPs. |
| Geographic Module | Communications with domains in certain geographic locations (e.g. North Korea, Eastern Europe and China) – a software module can be designed to evaluate geographic origin of traffic. There may be certain domains that are configured as automatic alert triggers. |
| Geographic Visualization Module | A very simple way to assist administrators in evaluating suspicious source/destination trends is to simply draw them a picture of where connections are stemming from. This module simply takes the geographic output and draws world map with points of interest being network connection points. |
| Newborn Communications Module | Communications with domains that are less than 60 days old—most malicious software communicates with systems that are relatively new, as old domains become published on blacklists by administrators and services that specialize is producing lists of malicious domains. |
| DNS Module | Because malware C&C servers are shut down routinely, DNS errors are commonly generated by malware. Counting DNS failures on a network per host is a useful barometer. |
| Reputation Module | a software module can be provided that connects to a reputation database to calculate a reputation-based score on the communication |
| Connection Periodicity Module | Malware beaconing generally, but not always, happens in a sustained period manner (e.g. every hour, twice a day, etc). Detecting unknown processes that beacon out periodically is useful in identifying at risk hosts. |

### Geographic Module

To build a geographic module it is necessary to implement code that does performs three high-level functions:

1. parse PCAPS
2. identify IP source and destination addresses in PCAPs
3. Map the IPs to latitude and longitude
4. Creating visualizations

#### Parsing PCAPs

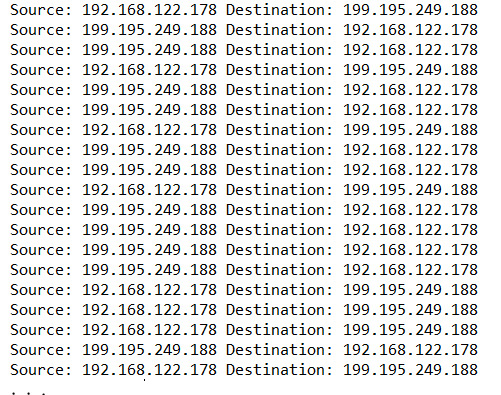
Recall that one of the reasons for choosing Python is the language’s vast support and a world-wide network of contributors. Consequently, one of the benefits of performing analysis in Python is that there is usually a library available to suit a programming purpose, or at least use as starting point. In terms of parsing PCAPs, several existing libraries are available. Dpkt is a relatively simple library available from GitHub (2015) for free download and use according to the BSD license.

Using the dpkt library it is possible to ingest a PCAP and return any information on a per packet basis. One simple example is finding the source and destination IPs of the various packets along with the transport protocol in use. This is simple and common enough that virtually the exact code needed exists in online tutorials, the code below is adapted from Jon Oberheide’s page.

Figure : Code example – IP src and dst parser based on dpkt



Figure : dpkt parsing results for IP src and dst



The next exercise is to associate the dpkt results with a library that provides geographic information relative to IP addressing. To perform this it is necessary to gain access to an IP address to geographic location database. There are many available, but many are expensive. O’Conner (2013) carries out a similar exercise using a freely available database by MaxMind Inc. called GeoLiteCity along with a Python library developed by Jennifer Ennis called pygeoip. The Python Shell example below shows how powerful and simple a combination these tools are—the example uses the first non-private IP address from the PCAP used above.

Figure : Code example – Python shell demonstration of the GeoLiteCity database

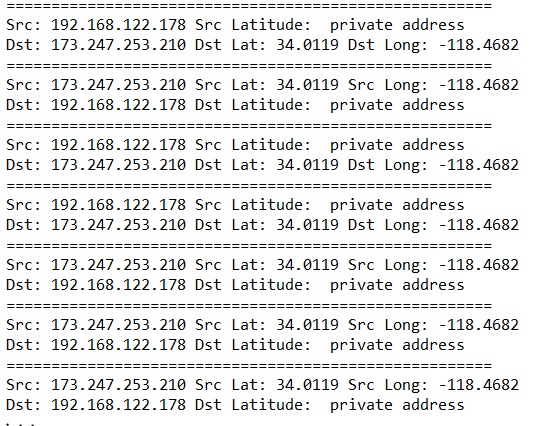


It is simple to formulate a procedure around this functionality and combine it with the parsing code to create a simple geocode routine. This allows the lat and long of each packet to be captured.

Figure : Code example – geocoder routine to return lat and long



Figure : output of latitudes, longitudes



#### Visualization

It is only useful to know latitude, longitude information if it can be tied to a country or region of interest. It is straightforward to alter the code to show the country and/or city associated with the connections. However, one of the greatest problems faced by administrators and SecOps personnel is data presented in too bulk a manner to be actionable. Therefore the preferable presentation of these data points is through a map visualization so that the viewer can determine at a glance if there is anything noteworthy enough to look closer at the bulk data.

There are many possible presentation effects that can be added to enhance at-a-glance inspection for administrators:

1. Hover over packet count
2. Hover over country name
3. Hover over port information (what port was used
4. Heatmap of country with colors reflecting areas of greatest communication

All of the above would likely be useful in a production tool. The main objective is to reduce the data to a manageable amount, which means it is not useful to produce a point per packet, but rather to make the unique connection points visible with the ability to “hover over” to gain other information. Below is a code snippet that produces a unique connection point count and draws a map using Google visualization JavaScript available from the Google Developers Visualization geomap page. The Python Collections functionality is very useful for boiling the data down to aggregated conversations versus a mass of individual packets.

Figure : Code example – creating lists of lat longs with the count of occurrence using Python Collections Counter class (complete code in Appendix)



Figure : Use the code above to generate data points to build an HTML file with javascript pointing to Google maps – see complete HTML file in Appendix

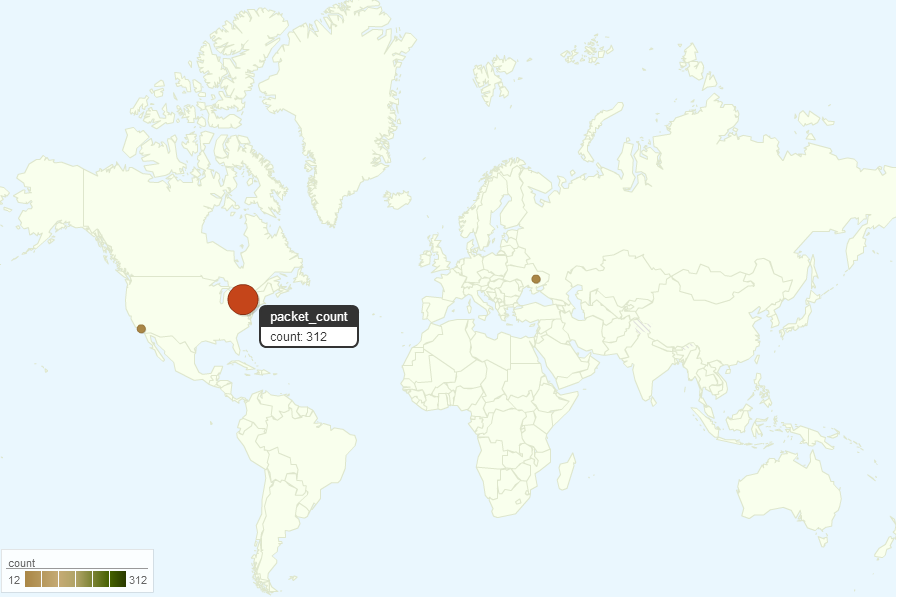
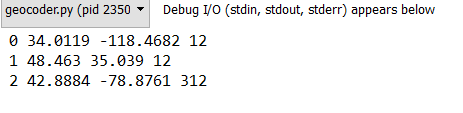


Figure : Code output – unique latitude longitudes and packet count



## Strategies for Scaling Visualization to Large Data Sets

The example above analyzed a PCAP with 324 network layer packets—not a particularly large number in the context of even a home router much less an Enterprise network. The capture had a particular conversation captured rather than a wide swath of general traffic. It is a fair question to ask how the visualizer would work if there were 324 billion packets rather than 324. There are several products that can be used to extend the visualizer to a large data set:

1. Splunk
2. Hadoop MapReduce

#### Splunk

The Splunk Enterprise platform (<http://www.splunk.com>) is especially useful for intermediate to large-sized datasets because the product scales to the Enterprise and includes some built in tools for visualization and PCAP analysis—meaning the Google maps step would be provided with no programming. Another plus with Splunk is that it allows streaming of packets in a PCAP format from a capture source and hence it would be simple to setup real-time dashboard visualization of geographic connection endpoints.

Splunk implementations include a series of indexing systems that may or may not be large enough to practically handle truly huge datasets—typically a Splunk architect sets up the system such that it has IOPS in the range of 1000-2000 (Splunk), which would not typically be suitable for an aggregated packet capture across a large Enterprise Network. Another practical consideration is that Splunk is licensed by bandwidth—it would not be practical for most organizations to send all Enterprise traffic through Splunk in terms of cost (author’s anecdote, Splunk’s largest customer spends $200M a year on licensing and they are not sending anything as voluminous as systematic packet captures to the product—that cost would go up exponentially if they were).

Most likely the more practical use of Splunk Enterprise would be to use another platform to summarize data and Splunk to ingest the summary information and present it as part of a comprehensive reporting and alerting platform. Conveniently, Splunk has a product called Hunk which can read data off of the Hadoop File System (HDFS), and that brings up the next option: Splunk as a post-processor for Hadoop MapReduce.

#### Hadoop MapReduce

A larger scale and more cost efficient approach to geographic visualization of aggregated Enterprise traffic is to use Hadoop. On the surface, this would seem challenging in that the PCAPs are binary—requiring that one write a routine in Java to handle the file type in Hadoop Java API to be ingested in the mapper, as well as integrate the Geo translation into the mapper. The fact the GeoCityLite database is binary is not that much of a problem given that it can simply be loaded and invoked from Distributed Cache.

Upon reflection, however, it is possible to reduce the complexity by turning to Python and Pig. In this case, Pig would be invoked to call a python script, which has the Pig routine to abstract the mapping and reducing embedded within it. Here are the steps:

1. Load the GeoCityLite.dat into HDFS for use in Hadoop Distributed Cache
2. Translate the PCAP to the src and dst IPs needed using program called Tshark
3. Use Python to implement distributed cache and to invoke the GeoCityLite.dat to transform the lists of IPs into a file containing a simple list of Lat/Longs
4. Also from Python, call an embedded Pig routine that is not much more than a simple word count routine
5. Output the counts
6. Ingest into either Google Visualizer geomap (as in the previous example), KML (Google Earth, or Splunk Visualizer

The key concept above is using Pig: Pig is a high-level, SQL-like language that abstracts and simplifies MapReduce in comparison to using the Java MapReduce API directly. Another key is avoiding binary files: Hadoop typically uses text files as input. It is technically possible to process a binary file with MapReduce, but simpler to convert the PCAP to a text file using a program such as Tshark (Tshark). The figures depict selected steps, with the output of the Pig geo-packet at the end.

Figure : Placing the GeoLiteCity database in HFS so that it can placed in Distributed Cache

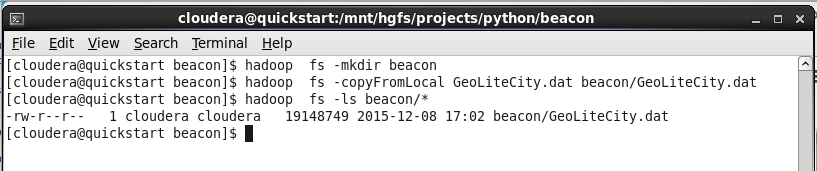
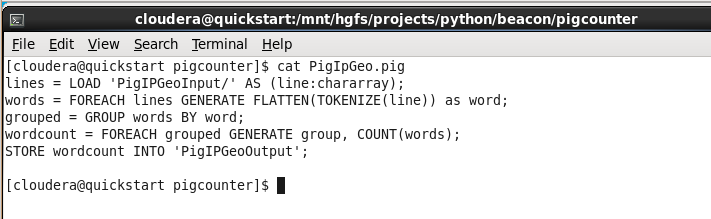
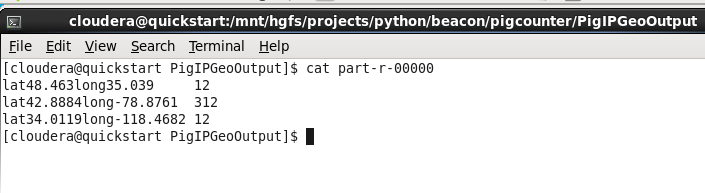


Figure : Embedded Pig – note that this is essentially the code for Word Count in the Apache Hadoop documentation pages



The Pig routine produces the same result as the earlier Python-based geographic end-point counter. The code can be embedded into a Python script that performs other actions such as invoking the IP to lat/long translation and producing the input stream.

Figure : Show the output from the Pig counter—note this is exactly the same as from the Counter class in Python in the non-MapReduce version



# Summary

Advanced Persistent Threats (APTs) consisting of highly disguised malware that beacons to C&C servers are among the most dangerous cyber-security threats in existence. Building and deploying systems that can detect the sometimes faint signals is a challenging part of network forensics. As obvious as packet geographic visualization seems, many if not most network administrators do not employ it. In larger networks with abundant resources, it would be possible to setup a stream to a datamining cluster that produced batch results that could output a simple morning report for administrators to review regarding the previous day’s traffic.

Simple flags such as conversations to countries with high-levels of cyber-attack launches and conversations to new domains could allow detection of some APT activity. Though it may seem very complicated to build a Hadoop-based system for ingesting large datasets of network activity and analyzing it; in some cases the problem ends up no more complex that a simple Word Count routine studied as an introduction to MapReduce coding.

# Bibliography

Apache Hadoop. MapReduce Tutorial. Retrieved from <https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html> December 9th 2015

Dpkt GitHub page, Retrieved from < <https://github.com/kbandla/dpkt>> December 2, 2015.

Dpkt Read The Docs Page. Retrieved from <https://dpkt.readthedocs.org/en/latest/examples.html#examples-in-dpkt-examples> December 3rd 2015.

Gilman, R. “Intro to PCAP Powerpoint”. Retrieved from <http://www.opensecuritytraining.info/Pcap_files/intro-to-pcap-public-release.pptx> November 29th, 2015.

Google Developers. Visualization Geomap. Retrieved from < <https://developers.google.com/chart/interactive/docs/gallery/geomap?hl=en#regionsexample>> December 4th 2015.

Hosmer, C. (2014). *Python Forensics*. Waltham, MA: Elsevier,

Oberheide, J. “dpkt Tutorial”. Retrieved from < <https://jon.oberheide.org/blog/2008/10/15/dpkt-tutorial-2-parsing-a-pcap-file/>> on December 1st 2015

MaxMind GeoLiteCity IP to Geographic Database. Retrieved from <http://dev.maxmind.com/> December 4th 2015.

O’Connor, T.J. (2013). *Violent Java*. Waltham, MA: Elsevier.

Parkour, M. Contagio Blog. Retrieved from <,<http://contagiodump.blogspot.com/>> December 1st 2015.

Pig (programming tool). (2015, August 2). In *Wikipedia, The Free Encyclopedia*. Retrieved from <<https://en.wikipedia.org/w/index.php?title=Pig_(programming_tool)&oldid=674267119>> December 6th 2015.

Splunk Documentation. “Capacity Planning Manual”. Retrieved from <http://docs.splunk.com/Documentation/Splunk/6.2.0/Capacity/Referencehardware> on December 6th 2015.

Tshark. Documentation Page. Retrieved from <https://www.wireshark.org/docs/man-pages/tshark.html> on December 7th 2015.

Wireshark Wiki. “Libpcap File Format”. Retrieved from <<https://wiki.wireshark.org/Development/LibpcapFileFormat>> November 30th 2015.

1. Definitions and Acronyms

Table Appendix 1: Acronyms from this document

| **Acronym** | **Title** |
| --- | --- |
| **APT** | Advanced Persistent Threats |
| **AV** | Anti-virus |
| **C&C** | Command and Control |
| **DDOS** | Distributed Denial of Service |
| **dst** | Destination |
| **HDFS** | Hadoop Distributed File System |
| **IP** | Internet Protocol |
| **IPSec** | IP Security |
| **IT** | Information Technology |
| **LAN** | Local Area Network |
| **lat** | latitude |
| **Long** | longitude |
| **MAC** | Media Access Control |
| **NIC** | Network Interface Card |
| **NIDS** | Network Intrusion Detection System |
| **OS** | Operating System |
| **PCAP** | Packet Capture |
| **SAN** | Storage Area Network |
| **SECOPS** | Security Operations |
| **SPAN** | Switched Port Analyzer |
| **src** | Source |
| **TCP** | Transmission Control Protocol |
| **WAN** | Wide-Area Network |

1. Code Submissions

Figure : Code example - Beacon Listener TCP Service



Figure : Code example – function to get latitude and longitudes (geocodes)



Figure : Code example – function to generate Google Map datapoints



Figure : Code example – function to output HTML file footer for Google Maps



Figure : Code example – function to output HTML file header for Google Maps



Figure : Code example – Main driver routine for packet parser



Figure : HTML File for Google Maps

