Research project - Report
Beacon detection in PCAP files

Leendert van Duijn[§]
Supervisor: Sjoerd Peerlkamp[*]

August 2014

[§]Student Msc. System and Network Engineering
[*]Shell Information Technology International B.V.

Abstract

Malware can be detected based on signatures and behavior. Using signatures leaves a window of opportunity for new variants of known agents to evade detection. Polymorphism elevates this risk by implementing automation in changing signatures.

Behavioral analysis can be implemented on several layers, from host level intrusion detection to global trending in network traffic statistics.

By looking at Beacon traffic, periodic traffic of arbitrary protocols, it is possible to detect infections based on passive network monitoring.

Beacons can be a simple notification to a control server to automatic updates and command polling. Complex Peer to Peer botnets have been shown to show patterns in network traffic behavior. These localized patterns allow for recognition of malicious control traffic.

This project concerns generic beacons. Though not all Malware will produce beacon traffic and although there are legitimate sources of Beacon traffic, the capability to detect unknown threats to a network or host can increase overall security. A reactive approach to early beacon traffic can reduce damages by Malware over time as the infections can be cleaned.

In this project beacons have been identified and traced back to Malware traffic. With limited false positives in legitimate traffic.

Contents

1 Introduction

Beacon detection in PCAP files is not about beacon frames as part of 802.11[1]. The question is how you can recognize the presence of compromised systems that are beaconing to command & control infrastructure if you have access to packet captures, while the actual beaconing can take place with differing frequencies.

The goal of this research was the detection of compromised systems in a, with regards to the compromised systems, non invasive manner. By searching for information in network traffic dumps, an entire network can benefit from a singular investigation. Increasing the time available to do such an investigation as opposed to in depth analysis, of individual hosts.

1.1 Related work

There has been ongoing research into protecting hosts and networks against intrusion and data leakage. Advanced Persistent Threats, Malware and Trojans deployed to a specific target, are hard to detect by traditional systems. As signatures will not be readily known. Vendors are unlikely to have encountered such a threat due to the limited deployment, new/advanced techniques and specialized nature.

There are effective methods to detect these, as shown in [1]. This system can detect an unknown APT, though it requires low level integration with each host to protect.

Systems exist to detect and identify Command and Control traffic. The Jackstraws system[2] does code analysis on software, linking behavior graphs to network connections. It can be used to categorize network traffic based on local software behavior, looking at which components and program flow cause network traffic. Using this it can identify Command and Control traffic even in the presence of friendly traffic.

The Jackstraws system requires locally gathered, behavioral information on all software accessing the network. Using the information from Jackstraws an active effort can be made to hinder any Botnet traffic, either by filtering or active interference.

Both the APT detection and Jackstraws system require deployment on each host to function. Which in larger networks may not be feasible.

There are systems which do not have such requirements to detect Command and Control traffic, in an automated fashion. Such as the ProVeX[3] system, which tries to detect Command & Control traffic based on deep packet inspection. It treats all packets as suspect, using reverse engineered decryption functions decrypts them into a classifier based on probabilistic vectorized signatures. This approach needs existing knowledge, specifically on the Malware the system can detect in order to implementation of decryption functions. This limits it to known Malware using re-implementable decryption with either constant or weak keys.

These requirements can be costly to fulfill, though deployment can be done to key nodes in a network, while maintaining full detection coverage. Its scope is limited against most APT, as they are unlikely to be compatible. Both

[1] IEEE Std. 802.11-2007,
http://www.mathworks.nl/help/comm/examples/ieee-802-11-wlan-beacon-frame.html

the decryption and detection methods are highly customized to a selection of conventional Malware.

The DMRP[4] system aims to detect presence of P2P Botnet control traffic. Looking at regional periodicity, effectively mining patterns specific to P2P traffic. It is implemented by looking at periodic behavior over shorter spans of time. It can reveal Peer to Peer traffic, as used for existing Botnet agents.

It does analysis on the subset of data it deems most likely to be malicious traffic using packet rate per second as a key variable. By detecting patterns in the packet rate, it can find the control traffic. It shows that it can separate it from traditional peer to peer traffic.

2 Research questions

This project will show that beacon detection on network traffic is possible and could be used to detect Malware activity. Doing so the following questions are answered:

- Can traffic dumps be used to detect beacons produced by Malware?
- Can partitioning traffic on common/shared features increase signal, and beacon, clarity?
- Is it possible to detect Malware in the presence of legitimate beacons?

3 Theory and approach

3.1 Definition, a Beacon

With a goal to detect Malware beacons, definition of a beacon is important.

Outgoing traffic can be classed as a beacon when

- Outgoing requests/packets showing similar features, in a repeating pattern over time.
- Response packets show similar features to the request or close relation to each other.
- The time between the requests is of a predictable nature, or a pattern holds over a longer period of time.

Since not all outgoing messages require/receive acknowledgment, during this project a beacon is defined as: *A repeated connection (attempt), showing a pattern over time.*

A number of identified variants have shown this behavior, among these are `Houdini FX` and `DaRK DDosser`.

Though not all Malware is required to show this behavior, a Malware agent could randomize its communications interval. Operate on seemingly legitimate channels, or combine multiple channels to hide by only using each channel for a limited time.

Malware with cloaking to prevent these traits from being observed fall outside of the scope of this project.

3.2 Traffic dumps

By looking at full network traffic dumps offline, it is possible to detect patterns generated by specific software and Malware. Patterns may reside in combinations of traffic, content and even meta data such as timing. Using this, exceptional traffic can be marked for analysis as anomaly detection.

A promising approach beyond simple packet or stream based anomaly detection is based on harnessing packet and traffic features to classify traffic. With separation of traffic based on suspected nature, the field of signal analysis can assist in finding anomalous connections.

Investigating activity over time, partitioned to isolate distinct signals, can help to identify temporal patterns which indicate beacons. These patterns may be detected by using auto correlation and frequency analysis. The information extracted via these methods should reveal to an analyst the likely beacons, without explicit knowledge about potential Malware agents.

3.3 Analyzing packets

Offline analysis requires access to packet data. During this project Shell made packet data available. The data source being a Sinkhole project, aimed at reducing the impact of potential Malware infections. In the form of PCAP files containing complete packets, containing traffic from a number of hosts, generated by several families of known Malware. In addition to several explicitly blocked web services. The set contained 20 files which approximately 2.5Million packets each, During this project the files were treated as separate data sets.

The Sinkhole data set contained primarily Malware related traffic. In order to validate findings and compare true positives with false positives some sample data sets were generated. One of these sets contained casual browsing, long living SSH connections with various activity and the watching of several online videos. An alternate set made on the gateway of a Wifi network. The network containing 2 devices, used for casual browsing, and an incidental port scan.

Due to the potentially sensitive data contained in the captures they are destroyed after the project concluded and will not be published.

To obtain the most effective results, and allow for further analysis of any detected beacons, the packet dump was chosen to contain full payload through the project. A reduction to only header data could save resources in both storage and processing time, at the cost of destroying the evidence required to do an investigation. Privacy is a consideration to make while doing traffic analysis, especially on complete packet dumps. However the contact with private data is limited to that within the final analysis steps, as any other content used for detection can be masked. Hiding possibly sensitive data such as source and destination.

3.4 Partitioning on classes

Class membership can be based on derived attributes of a packet. It is possible to declare a single packet member of several classes.

There are multiple approaches of generating classes or clusters of traffic. For this project 3 have been examined. For each a selection of implementations and extensions exist.

It is important to prevent introduction of non existent signals, patterns caused by traffic being shaped by the filter. Such patterns can lead to signals/communications being falsely labeled as Beacon traffic, reducing the trustworthiness of overall results.

In order to prevent this, variables used to conclude what streams are beaconing, should not be used in the stream selection. In cases where this information might be part of the classification process inspection should reveal whether results are false positives. To do so a set of known good, ie. no beacons, could be tested to see whether beacons are detected.

During this project the packet arrival time and packet offset/index are therefore unavailable for classification.

3.4.1 K means/mediods classifier

The unsupervised K means/mediods classifier can be used to determine clusters on numbers, vectors or packets. The classifier will produce a configurable number of classes, doing so by iterative adjusting classes in an attempt to minimize the distance of each item to its chosen class.

The results are non deterministic and there is initially no knowledge about what a class entails[2], though class membership of an arbitrary packet can be calculated efficiently.

With the limited control and high sensitivity to the number of desired classes, this classifier does not seem optimal for beacon detection. As the number of streams is not known in advance, and a trail and error approach to configuring the classification could be time consuming.

3.4.2 Unsupervised Tree building

This unsupervised clustering technique builds a guided number of clusters, by iterative joining of the most compatible clusters, building a decision tree. This hierarchical tree can be inspected to gain knowledge of the data set. This knowledge may not be useful in detecting beacons, depending of which fields are available to base tree building decisions on, and the specifics of the tree constructed.

The tree building is computationally more expensive to attain. Though using a known tree to classify a packet is efficient. Using prior knowledge to guide these algorithms require careful consideration and experimentation, which can be time consuming. With these considerations it does not seem optimal for beacon detection.

3.4.3 Expert based classes

Another approach is the supervised creation off class distinctions.

This can be done to utilize all prior knowledge of the network, software and other context. All information related to the data set an operator has available.

Basing rules/partitions on known fields in packets such as protocol, port, source and destination, can allow very powerful partitions can be created. Interactive experimentation is possible, though not strictly required, by application

[2]manual analysis could be done on the chosen classes, though the simplistic nature of the simple K means method might deflate the results

of new rules, on any data set. The independence of this classification method allows for the design and tweaking of classes based on known good traffic, which can still be used directly on similar but unknown traffic.

Using this method, which does requires consideration of, and interaction with, fields to base classes on, it is possible to do in depth research into specific events/issues. It allows for tight control of what data is ignored during tests, with known bad traffic such as IP traffic directed at a sinkhole via DNS redirection.

Classification rules, based on known traffic could be beneficial in the process of finding anomalies. Even prior to pattern detection.

3.4.4 Chosen classifier

During this project an expert rule based system has been chosen for the following reasons,

Performance	Determining class membership is computationally cheap, allowing for large data sets being processed
Versatility	Using knowledge to neutralize noise based on out of band information. For example a roaming agent with short lived DHCP leases, using several distinct addresses. Combining these sources could improve detection for this host.
Simplicity	Initial results have shown that by using these simple classifiers Malware can already be separated from each other and benign traffic.
Extensible	By introducing more advanced features to incorporate in rules 'simple' rules can be extended to detect camouflaged Malware. Examples include, compression ratio of payload and adherence to custom models per packet.

3.5 Pattern detection

Beacons can behave in different ways. Short lived events occasionally pop up and call out, while consistent beacons produce constant connection attempts and data streams.

During this project the focus was on beacons with a relatively constant rate of beaconing, sending out packets on a constant interval. Detection of patterns could use either the number of packets per seconds, for protocols with smaller, temporally spaced, datagrams. Or bandwidth per seconds, for more active stream based beacons. Both have their benefits, though for this project the number of packets per time period was selected, as the known Malware interacted with the sinkhole in a limited fashion. Available payload, and thus realistic bandwidth consumption during operation, not fully representative of an active infection. This was caused by the sinkhole not implementing all handshakes required for Malware agents to initiate their true protocol.

Anomaly detection does not require analysis over time to obtain results, some indicators can be observed by the distribution of class size and membership.

- In a network on homogeneous hosts/agents probabilistic methods on classes size and ratio can reveal uncommon or foreign streams.
  This is anomaly detection on kinds of traffic seen by a sensor.
- Existence of certain 'known bad' classes can indicate Malware.
  This is signature based network intrusion detection.

- In depth analysis based on traffic within a class.
  This is related to stream reconstruction in that it uses similar traffic to 'reconstruct' what is happening on the line.

This project focuses on the latter: In depth analysis of traffic within a class, looking for patterns over time. Using classification to distinguish beacons from each other and alternate data streams.

3.5.1 Windowing

Starting a network capture does not mean that beacon traffic is already present, remains present, or even is ever present. Malware can limit their visible activity to reduce chance of capture. To increase chances to find beacons, it helps to do analysis over a proper time window and processing all available data. This can improve results in finding short lived beacons[3] and can vastly reduce computational cost by limiting the concurrently active resources.

By moving the window over all available data/time spans, analysis can be improved by revealing persistent patterns over time.

There are several methods in signal analysis which can be used to detect beacons, though few run on raw PCAP data. In order to use these methods the event based input, packets with time stamps, should be converted to activity over time. Building a complete time line of activity over small periods in time.

There are several parameters to consider when analyzing parts of the data set,

- What periodicity can be expected, what should stand out in the results?
- Over what period are beacon packets recorded, is there enough data to analyze?
- What are the effects of misalignment the analysis window with the first occurrence of a beacon?
- How many data points are available in what resolution?

Considering these facts, a configurable sliding window has been selected to determine periods of time. These periods of time are individually analyzed, and the results combined visually to reveal persistent results.

By selecting a width, appropriate to the specific context, it is possible to search for semi known beacons. Such as those by known Malware families. In order to find burst traffic a smaller stride can be selected, doing analysis on partially overlapping windows. This can be used to increase the chance of aligning the analysis windows with short lived beacons. This increases the chances that a smaller period of beacon traffic is clearly identified.

The question regarding the amount of data remains, a simple sliding window of a fixed width does not accommodate for gaps. This is considered for future expansion. Until such a time the sample quality should be manually estimated.

Scanning the class data over time can be done for several features, from statistics over packet rate and deviation from the overall statistics to signal analysis. During this project the packet rate per time period has been selected, as it has been shown to be feasible by the related work in [4], and later was confirmed by local experiments.

[3] Newly installed/active beacons, temporary beacons, or suddenly uninstalled/deactivated beacons

3.5.2 Frequency analysis

Detecting peaks and harmonics in a frequency spectrum could indicate patterns in a stream. This can be implemented by executing a Fast Fourier Transformation, turning a waveform into frequency magnitudes.

In order to get proper results some additional filters might need to be applied to correct the fact not all input will be periodic, this implementation together with normalization of the output is specific to the implementation and execution.

Using the scanning window to generate input a waterfall of frequency magnitudes can be produced, with beacons being periodic they should manifest as persistent frequency peaks over time. Visually, lines perpendicular to the flow of time.

1. Start at offset 0
2. At offset X in time measure the window data, obtain a histogram of packet counts
3. Run the Fourier implementation on this histogram, and fill out the first empty line in the output
4. Increase the offset by the $windowsize * stride$, moving region of time being analyzed
5. Repeat the process until all lines are filled and the data is exhausted

3.5.3 Auto correlation

Auto correlation peaks can indicate recurrences of a pattern, comparing a signal with itself shifted in time. The auto correlation can reveal matching patterns, regardless of their specific form.

Calculating the auto correlation for a window of data will reveal potential peaks for repeating signals such as beacons. With the sliding window a waterfall plot should produce output indicating beacons or other fixed patterns with lines perpendicular to the flow of time. With a large enough window size and pattern length harmonics will be visible as lines at multiples of the basic period. Visually these stand out even in the presence of noise as faint but consistently spaced lines.

3.6 When is it a beacon

To properly identify a beacon several occurrences must be recorded in the data set, with a larger number of observations increasing the strength of indicators, in both frequency and correlation.

As an example, in a network with homogeneous software deployment, active network protocols should exhibit similar protocols. This would lead to a stable distribution over classes during normal operation. To identify invasive beacons it is possible to eliminate the most typical classes, either by white listing on known good traffic, or by observations on class size[4]. Elimination based on

[4]larger traffic streams are more likely candidates for other detection methods, a beacon which consumes the entire network link would risk detection

white listing would be most effective in a fully homogeneous network with users with similar activities.

After a class of traffic has been identified specifically as good or bad this fact can be used to build context appropriate classification rules, when this information is unavailable general anomaly detection and investigation can be used to build such a set.

Lacking specific information about an infection exhaustive searching can be done, ideally prioritizing at risk hosts, protocols or artifacts.

It should be possible to combine beacon detection with a second stage of analysis, to provide complete reporting of potential beacon incidents. A score for beacons based on their actual content, white listing and popularity could be a powerful tool to identify anomalies caused by Malware and installation of automatically updating software.

4 Implementation

To examine the viability of beacon detection based on classified data set an experimental framework has been built. Designed to support offline analysis of PCAP data, combining and testing the methods described in this paper.

In order to allow rapid prototyping and future expansion of the proof of concept Python was selected as supporting programming language. A complete list of libraries used can be found in Appendix D.

4.1 GUI and libraries

The data set and beacons are analyzed over time, to visualize this a graphical interface was selected to directly interact with. From previous experience the Matplotlib Pyplot library was selected to generate graphs of numerical data, the Tkinter library to implement a control interface to the parameters used in classification, analysis and rendering of results.

To do signal and statistical analysis the Numpy and Scipy libraries have been chosen, both offering an interface to a variety of tools.

In order to keep the code base manageable an object oriented approach was chosen, though the proof of concept is that, a proof of concept. The user interface is quite basic and some features require specific program flow to work.

Publication details can be found in Appendix C.

4.2 Finding beacons, the approach

The method proposed by this paper to find beacons follows the information flow in Figure 1, for the experiments in this paper the following guideline was followed:

1. Obtain PCAP data
2. Import the data into the proof of concept
3. Make note of the size of the data set and time span
4. Select a classification rule
5. Identify anomalous/suspect classes
6. Inspect the size and time span for each selected class
7. Set the parameters for the waterfall plot

SliceStart	Skip this number of packets from the start of the input.
SlicePackets	Limit the number of packets to process to this number.
Resolution	The number of distinct time intervals to split the selected into.
WaterWidth	The width in distinct timesteps to use as an analysis window.
WaterStride	The ratio of `WaterWidth` the offset between consecutive windows.

8. Plot the waterfalls, and visually look for suspect patterns.
9. Export the beacon hits to PCAP for inspection
10. Adjust plotting parameters to clear up undecided plots
11. Adjust parameters for classes to narrow down on potential beacons

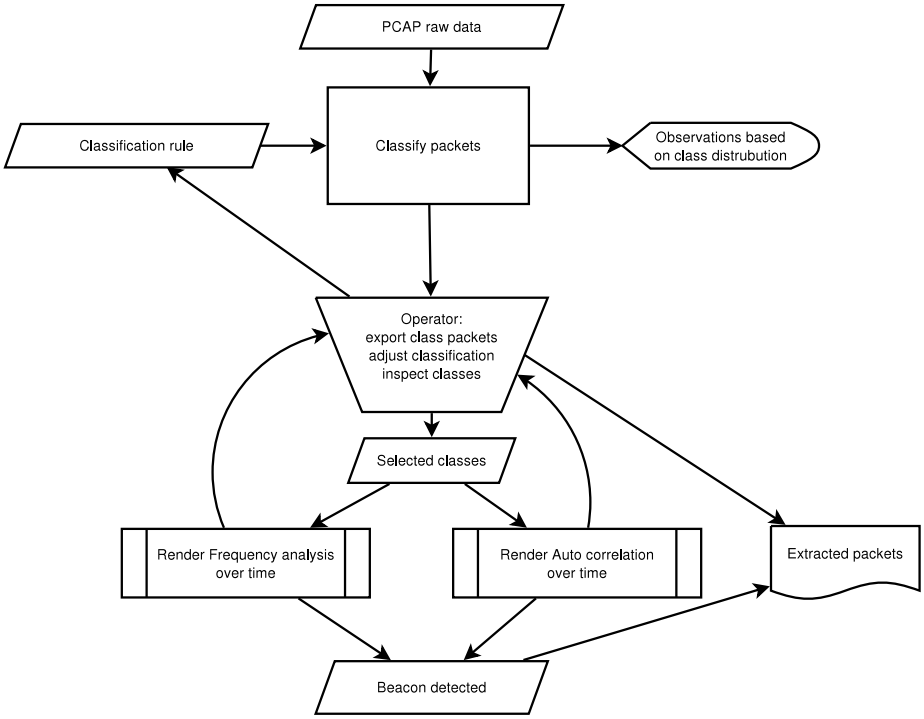


Figure 1: High level beacon detection

4.2.1 Patterns

The waterfall plot can contain indications of beacons and other artifacts, visually there are several distinct patterns. A selection of the most relevant patterns are described here.

Due to the fixed width time step in the class analysis gaps can occur, for incidental beaconing periods and host downtime this can prevent straightforward detection of these beacons. Collecting a sufficient amount of data and being aware of any downtime in the information stream can correct this.

In Figure 2 and 3 an example of data with gaps has been rendered onto Auto correlation and Frequency plots, both showing these gaps. Visually they manifest as horizontal lines of constant color, numerically as a window containing only zeros.

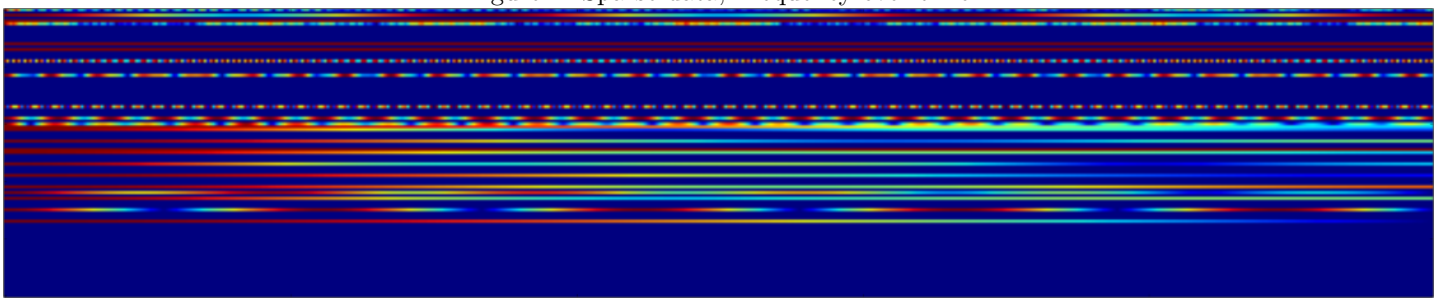
In Figure 2 the gap areas appear dark blue, and in figure 2 as white, rectangular, areas.

Short periods with high activity can mask other signals. In Figure 4 outgoing HTTP traffic over 30 minutes is shown. During this period several websites where browsed and kept open over longer periods. This data shows a noticeable feature, video data being transferred[5].

Though there appear to be other, minor, details visible they are out shadowed by two videos being watched. Resulting in 2 periods with a high fairly constant data rate. During these periods any minor activity has been masked

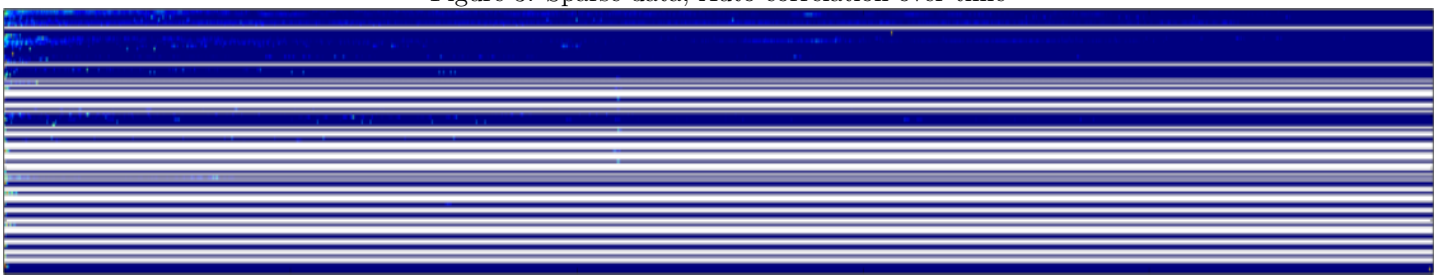
[5]This is based on the input data content, the visualization only indicates a burst of high bandwidth activity

Figure 2: Sparse data, Frequency over time



- FFT
- 2 Hours of DNS traffic
- Sample rate of 13.5Hz
- Window width 74 Seconds

Figure 3: Sparse data, Auto correlation over time



- Auto correlation
- 2 Hours of DNS traffic
- Sample rate of 13.5Hz
- Window width 74 Seconds

by the magnitude of the video streams. Though additional filtering, or alternate partitioning, could reveal masked patterns any noise in video traffic rates will adversely affect the results. By refining the classification rule it should be possible to separate most burst traffic from beacon traffic, in this specific case the destination would have clearly split video traffic from the potential beacon traffic.

Visually these bursts of data resemble the gradual red-green-blue transition, over the horizontal axis, the events taking place surrounding center of the traffic capture. The full color scale can be observed in Appendix A.

The patterns from beacon data can take many forms. Any repeating pattern can indicate repeating traffic. During this project, the known beacons produce a steady pattern which visually stands out. In Figure 5 the auto correlation over time can be seen to retain a consistent peak at steady offsets. These patterns have been found to be outgoing TCP connections which reoccur every 26 seconds.

Figure 4: Burst data, Auto correlation over time

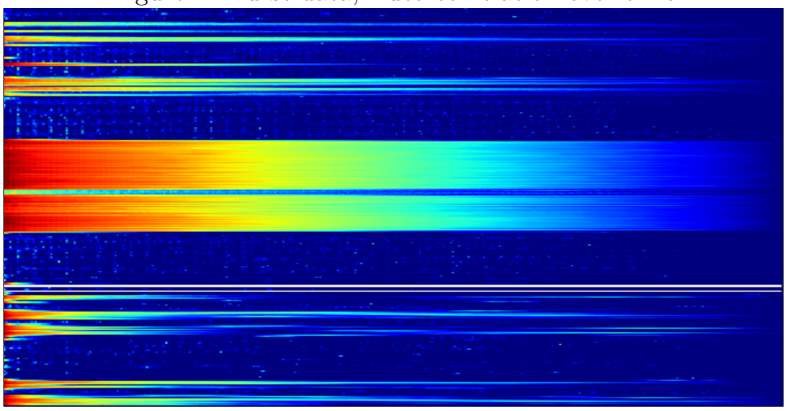
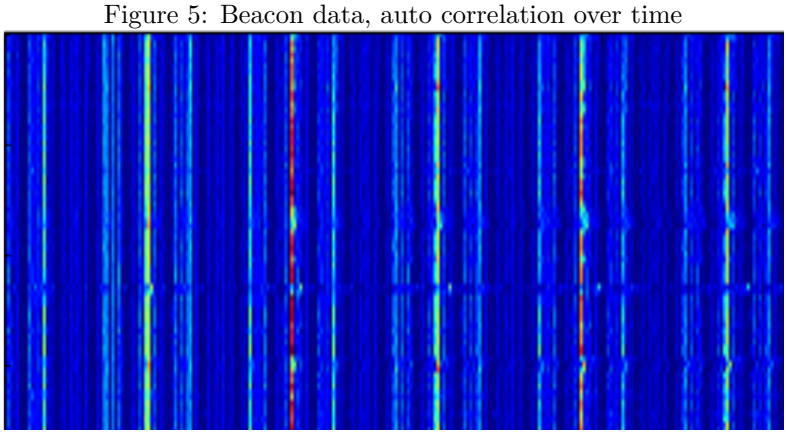


Figure 5: Beacon data, auto correlation over time



The data in question comes from the Sinkhole data, based on a source classification from a top 10 host in number of packets. Visually this beacon shows several, relatively flawless vertical lines. Due to the longer period this beacon was active the auto correlation shows multiple distinct lines, for the singular beacon. As multiple periods fit within one measurement window.

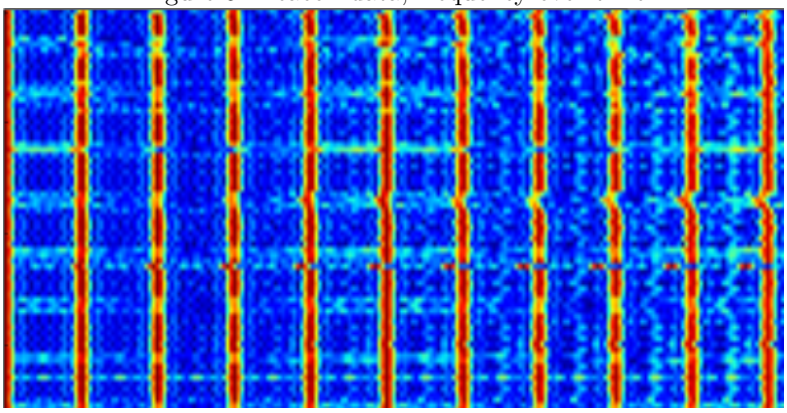
The frequency analysis found in Figure 6 shows the presence of the beacon by strong peaks, in the frequency pattern at the base frequency and several harmonics.

4.3 Fields for classification

With classification applied on individual packets, no attempts at stream reconstruction are done. In the proof of concept a selection of header fields and derived meta data have been implemented.

The classification has been implemented, internally keeping a dictionary/mapping between a list of supported fields and python functions returning a string value. By joining a set of these with underscores as separators a class identifier is constructed. Packets with matching identifiers defined as being within the same class.

Figure 6: Beacon data, frequency over time



The computational cost of this method is linear in relation to the number of packets to classify and depends on the fields in use. Each rule is unconditionally evaluated for each packet, and should not rely on global state. It should be possible to scale this over multiple processes, each handling a subset of packets to classify. Ideally this approach would be embarrassingly parallel, however the current implementation in python is single threaded, single process.

The implemented rules rely on the Python `dpkt` library to parse protocols, and the `zlib` library to estimate entropy.

Implemented rules are listed in Appendix B.

Using these a powerful filter can be created, looking for beacon data of a single source these fields are likely candidates:

- Source and destination, A client calling to a specific C&C server.
- Protocol, a single piece of software typically uses a limited number of protocols.
- Ports in use, the destination port for typical services/protocols are defined, though by no means all Malware needs to fit this behavior.
- Entropy, for packets which are large enough the rate of compression can identify compressed or even encrypted data, regardless the use of port.

During this project one of the most effective classifier for the Sinkhole data has been one of the simplest, the source IP address effectively splitting streams and Malware agents. This can be attributed to the Sinkhole setup, where primarily malicious streams are redirected with limited false positives.

The classifier settings are specified by the user, though a list of classifiers could be compiled for non-interactive analysis. Time constraints and interactivity during analysis prevented this extension.

4.4 Example rules

The user configures the classifier by supplying a line of text, field names separated by underscores. Some fields have parameters which become part of the field name.

- `src`

- `prot_dst`
- `prot_dst_tcpdport`
- `tcpdport_entropy.zlib:100,50`

4.4.1 A rule: TOS

During testing of the classification the IP Type of Service field was inspected, a data set selected from the Sinkhole collection, a total of 2.5M packets.

This experiment revealed an anomaly, as the expected classes/output would fit one of two models

- No use of this field, all packets the same (default) value.
- Active use of this field, differentiating streams in relation to protocol or network segment.

This data set contained known bad traffic, with a small percentage of packets in support of operating the sinkhole server and software. Of the traffic within this dump 'support' traffic was a candidate for a preferential Type of Service.

The resulting classes shows the majority of packets had TOS set to 0 (2.4M+ packets, 99.307%). The smaller classes contained traffic from various sources on using a selection of protocols.

Raw value TOS	Occurrence	Comments
72	160	DNS queries for specific domains
64	12K	Traffic from one localized network 2 TCP ports
104	4K	Multiple hosts over 5 TCP ports
192	315	ICMP port unreachable
200	155	ICMP port unreachable
Unknown	250	All local ARP traffic
0	2.449M	All leftover, IP traffic

Inspection of the leftover class reveals traces of the other classes. The reason for the TOS anomaly remains unknown. Using classes for anomaly detection does show results, prior to beacon detection.

This experiment indicates that there are several moments during the analysis where anomalies can occur. This specific rule yields suspect data, but cannot be used as a one on one indicator. With the resulting data only a subset of similar traffic[6] from the input data set.

5 Observations and results

During this project several capture files were inspected.

5.1 Experimental data, about the Sinkhole

The Sinkhole data sets contain full length packet dumps over a period of several weeks, split so each individual set contains some 2.5 Million of data. Traffic was

[6] source, destination, port and payload

directed to this machine based on DNS records for several known bad domains and prohibited services.

Due to the nature of this data set the destination of beacon traffic has been hidden, the original destination lost and could not trivially be reconstructed.

With redirection as opposed to quarantining the expected bad ratio nears the 95% range, estimated by the sinkhole operator. The remainder consisting of legitimate traffic origination from the server, such as software updates and DNS queries used for keeping logs.

5.2 Sinkholed Malware, based on source

In order to test classification, both in code execution and a general tool the following experimental question was looked at,

1. Does a trivial classification, such as IP source address, allow for detection of known bad traffic, on a data set with a high bad ratio?

To test this theory a data file was selected randomly, and investigated.By setting the classification rule to `src` all classes would consist of traffic from a specific source IP address. The data file in question containing a total of 2542513 packets over a 394 classes.

With one class reserved for Non IP traffic this leaves nearly four hundred classes to inspect. As such a selection of the top 10 classes in number of packets was made. This selection ranging from 18K to 1.3M packets.

Class ID	Number of packets
A	18089
B	19585
C	20134
D	20285
E	20375
F	21714
G	23431
H	39525
I	156098
J	1311544

With these class sizes two stand out, inspection with Wireshark reveals predominant traffic, and which roles they played during the operation of the sinkhole.

I	156098	Local DNS resolver used for looking up PTR records
J	1311544	The sinkhole IP, all outgoing traffic

With these two accounted for the remaining 8 classes are suspect hosts, as such a number of packets per host to end in the sinkhole indicates a significant chance of infection.

Class ID	Number of packets	Period of activity
A	18089	23.6 Hours, with a large gap of inactivity
B	19585	25.2 Hours
C	20134	25.2 Hours
D	20285	25.2 Hours
E	20375	25.2 Hours
F	21714	25.2 Hours
G	23431	25.2 Hours
H	39525	23.6 Hours

With these time ranges and packet counts a sufficiently large sample exists to detect beacons. In order to get a resolution of approximately 0.5 seconds and detection of beacons running at least two and a half minutes the following settings are used, which should detect beacons with a frequency starting at 1 second and show strong indications for any beacon frequencies of $K * 0.5$seconds.

Resolution	200000
Slicepackets	All
Sliceoffset	None
Waterwidth	300
Waterstride	1.0

This yields several auto correlation graphs showing persisting peaks with several harmonics.

In Figure 7 the classes F, H and A can be seen. From analysis in Wireshark it appears that the class H could be an infection with `DaRK DDoSseR`, based on information from `http://blog.trendmicro.com/trendlabs-security-intelligence/dark-ddosser-leads-to-gh0st-rat/`. Regardless the specific variant/version of Malware, this class was indeed unwanted beacon traffic and was identified as such by the analysis.

Though classes A and F do not contain payload traffic to analyze, only a single type of TCP connection being setup, and closed down due to the limited interaction the sinkhole offers, still indicated long term beacon on a non standard TCP ports. 82 and 81 respectively.

This experiment shows that Malware traffic can be identified by auto correlation, even if no Malware payload has been transferred.

5.3 Sinkhole, benign traffic

To see differences between an appropriate classifier, and a highly generic one the packets are again classified but this time using the `proto` classification rule. Effectively splitting separation ARP,TCP,UDP and ICMP traffic.

This experiment should show that the larger class, even when only a smaller period of time is analyzed shows poor results due to the variety of overlapping signals.

The analysis was done on the first 50K for UDP and the first 150K for TCP[7]. The resolution set to 0.1 second, allowing for fast beacons to be found and a window width to look for beacons active for 60 seconds or more. With this window width should allow for detection even in presence of noise, where a

[7]These numbers where chosen to get more than an hour of data each and a minimum of 50K packets

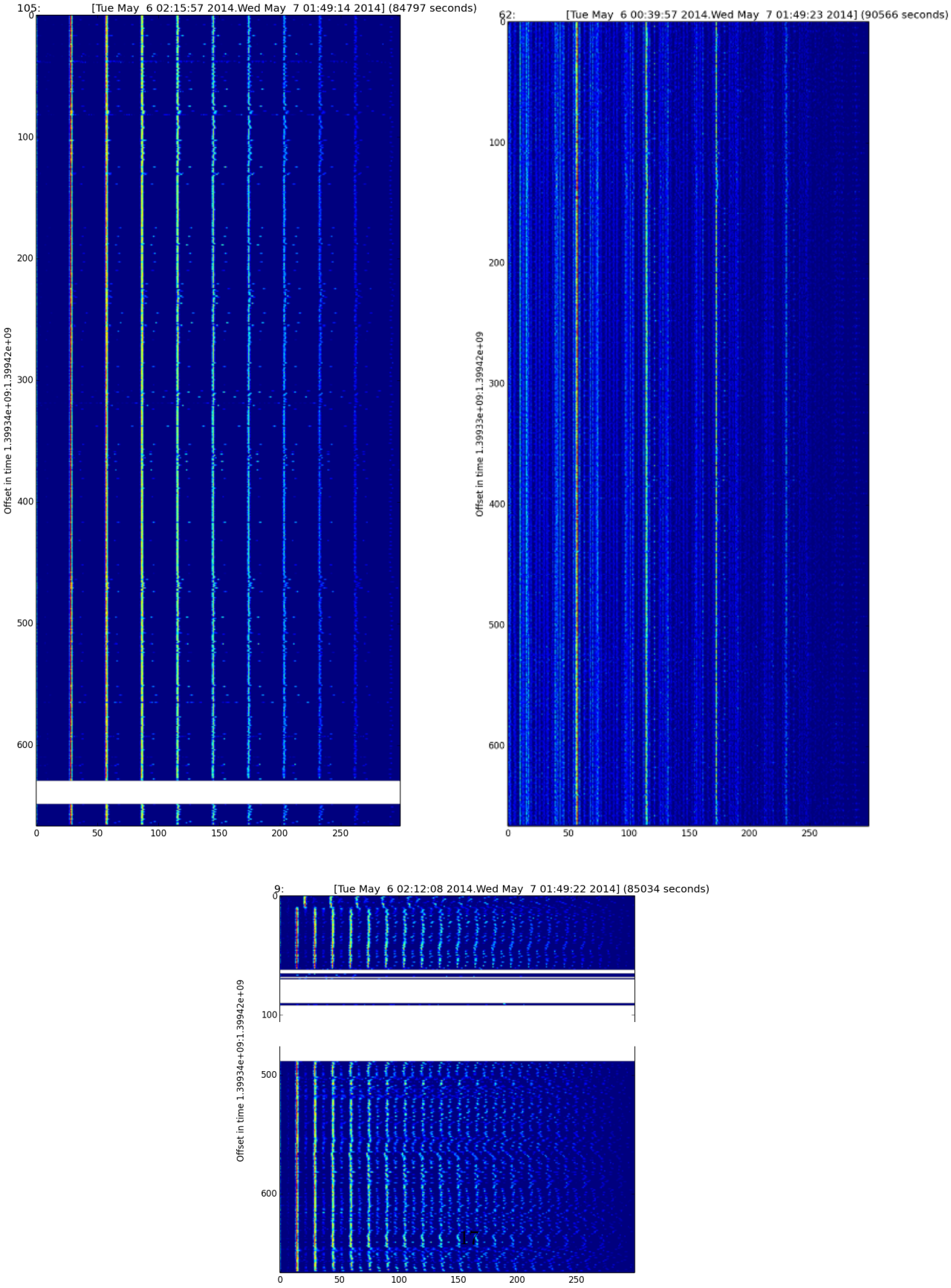


Figure 7: Auto correlation over time suspected beacons

smaller window would not see enough repetitions of a beacon to get significant readings.

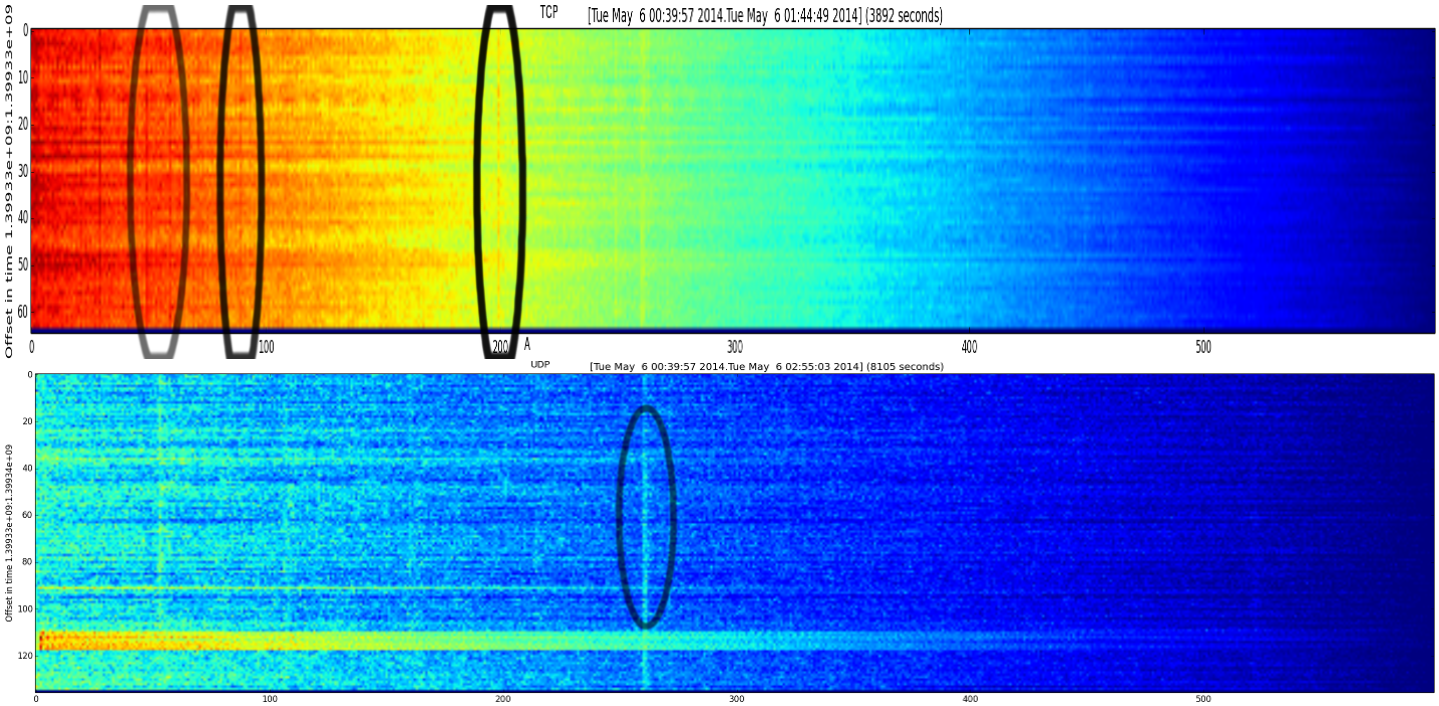


Figure 8: Auto correlation per per protocol

In Figure 8 the auto correlation was rendered, even with a mix of unrelated traffic several faint lines can be seen, for TCP the most easily seen would be at the circled area with the A mark, indicating a pattern with an offset of 20 seconds. Other lines can also be found, two are also circled for clarity.

For UDP very limited overlap exists with the TCP features. With new features at the cirled mark, around 26 seconds.

Though these features do still exist, their clarity is faint compared to earlier 'pure beacon' results. Manual analysis on these larger classes would be also severely more time consuming as the signal to noise ratio is relatively poor, and the indication that a beacon does exists cannot directly be used to find said beacon packets in the input.

During analysis analysis an evaluation should be made as for a smaller, more uniform, data set it would be more trivial to confirm a beacon as the cause of analysis features. The larger data sets might show such signals due to other, non beacon causes. One example would be that (in)activity on an investigated network could be caused by load on external networks. Careful consideration of classification rules should allow the focus to be aimed at locally generated events/beacons.

This experiment supports shows that by increasing the number of distinct classes the clarity of patterns can increase, by demonstrating the reverse by generalizing the classification for a data set with identified beacons.

5.4 Finding beacons in legitimate traffic

A network dump was made over several hours from a network segment containing a laptop and a phone. The dump included a variety of protocols used while browsing the web and running software in the background.

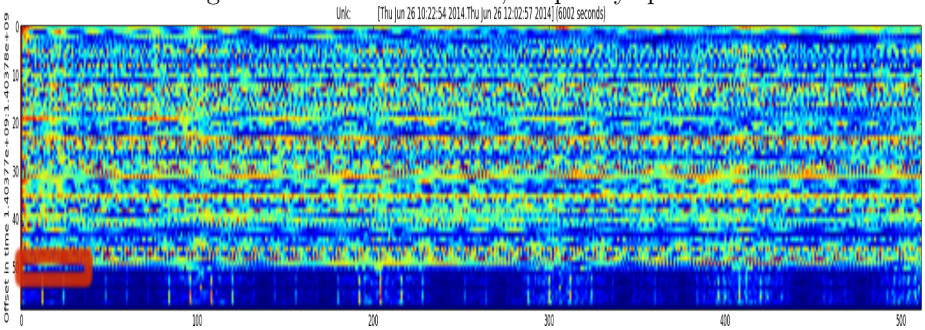
To find beacons in this traffic could lead to one of multiple conclusions,

- Malware as active at the time
- Legitimate traffic strongly resembles beacon traffic
- Beacon patterns are introduced by filtering and processing

Doing classification based on IP endpoint pair reveals one possible beacon class in the top 10 of Number of packets, this class specifically mapping to non IPv4 traffic. In Figure 9 the frequency spectrum over time can be seen with a resolution of 0.1 second. During selection of beacon classes a width of 30 seconds for correlation was selected, one of 102.4 seconds for the FFT.

These parameters match those tried on the other selected classes, though visually none contained beacon traffic.

Figure 9: Non IP over time, frequency spectrum



In the frequency spectrum in Figure 9 a clear change occurs around the red mark and downwards, this moment corresponding to roughly 5160 seconds into the stream. Wireshark shows this event coincides with losing one of the clients[8], causing ARP broadcasts to locate said client. Without other traffic to mask it only a short period of ARP traffic, and ever present DropBox Lan Sync Discovery packets remaining.

With Figure 10 confirming this beacon the experiment shows that beacons can be detected from network dumps, though not all beacons are Malware.

5.5 Artificial data and friendly beacons

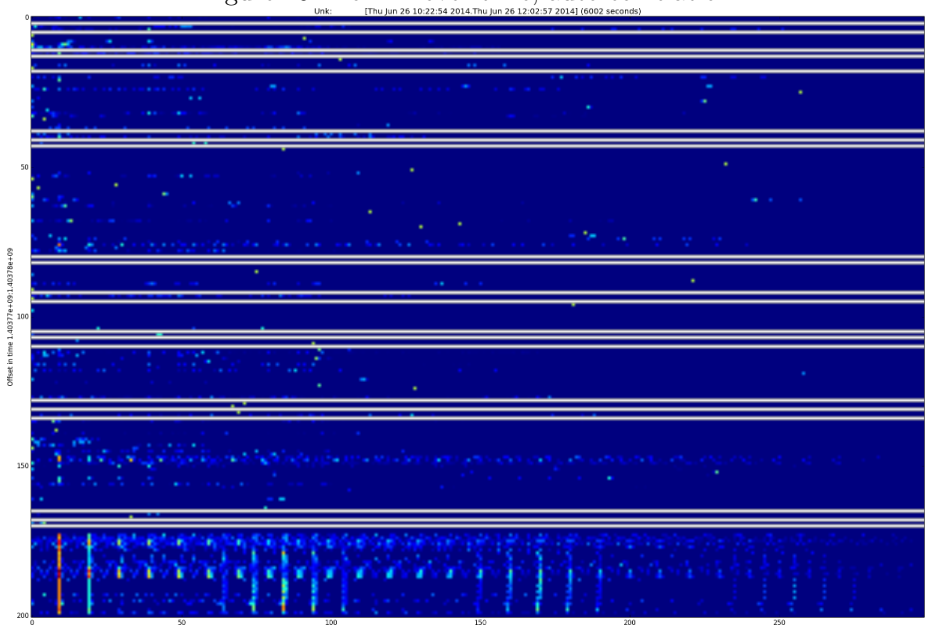
In order to examine false positives a data set was constructed with casual browsing, watching a couple of videos and accessing several machines over SSH. There was no known Malware on the machine at the time.

The following software was active,

SSH running `top` on a remote machine, a process monitor

[8] the other not far behind in shutting down

Figure 10: Non IP over time, auto correlation



SSH	running `ncmpc` on a remote machine, a command line client for a music player
Chromium	browsing several sites including *Facebook.com*, *Twitter.com*, *Archlinux.com* and *Youtube.com*
Chromium	used to watch a couple of random videos during a break from active browsing
Pacman	the Arch linux package manager doing a system upgrade

With expected beacons over SSH and minor beacons over HTTP/S the classification was selected to use only TCP destination port. The window parameters were set to achieve a 0.1 second resolution and for beacons of at least 30 seconds of activity[9] In Figure 11 the auto correlation and frequency graphs can be seen for this data set, it shows long lived beacons via persistent vertical peaks. During the session user interaction with the filesystem on the remote server listed a several million files, explaining the noise between marked by the blue sideline.

The frequency analysis and correlation data support the observation that even during this period of added activity beacons still persisted. Though preexisting knowledge of the data set might influenced the observation, which could have originated from expecting SSH beacon results. The clear contrast visible in the frequency spectrum of the analysis, leaves limited room for conclusions other than consistent, periodic traffic.

In Figure 12 one of the larger classes shows at most a very faint beacon, visually a faint line just below the 60 mark on the horizontal axis, indicated by the red circle in the frequency plot. In the auto correlation plot it shows a horizontal gradient, indicating a stable signal[10]. In this case closer inspec-

[9] This period should allow at least SSH beacons to stand out

[10] All bins in the histogram containing a larger number of packets, with relatively little

Figure 11: Artificial data, SSH beacon

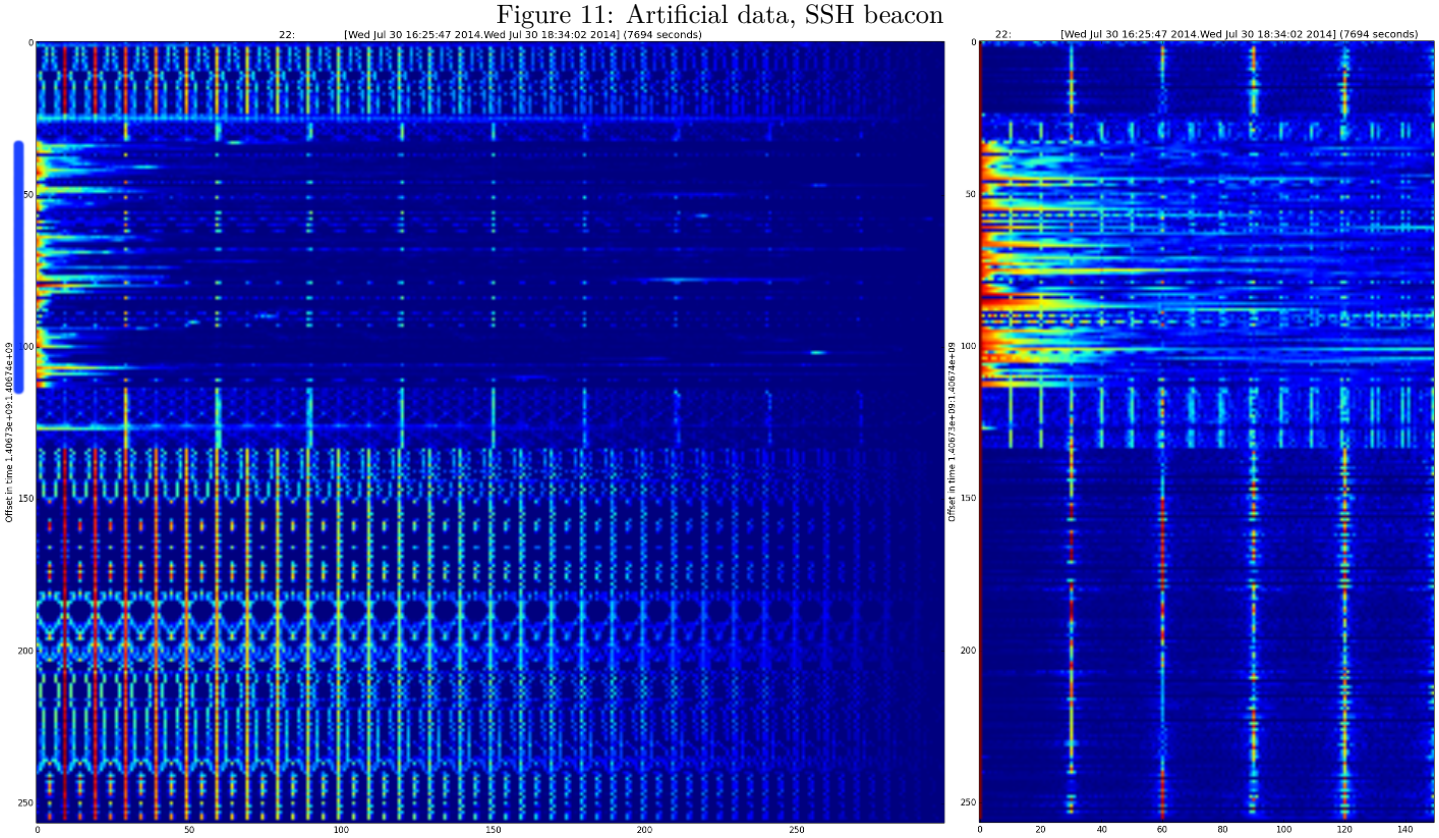
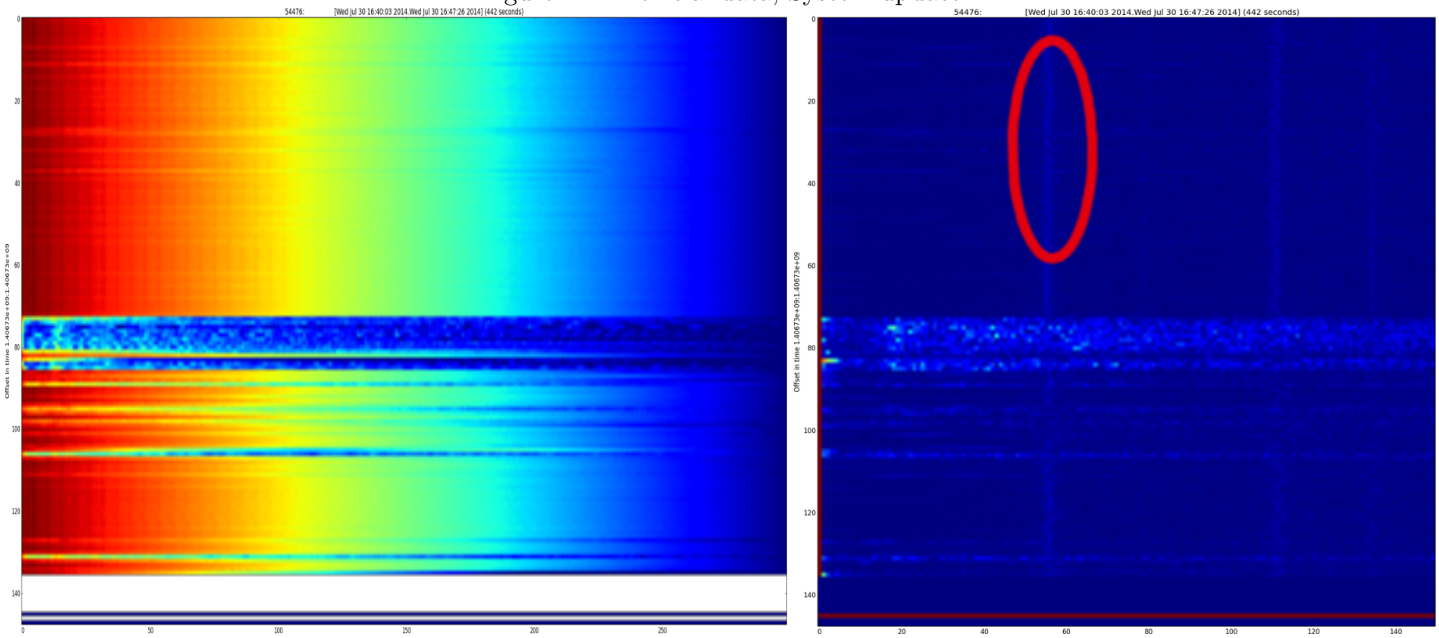


Figure 12: Artificial data, System update



tion revealed that the traffic originated from a mirror hosting files, downloaded during a system upgrade.

Indications of beacon like traffic could originate, as observed during the generation of the data, from the network bottleneck saturating the down link connection. Constant bandwidth, packets size and acknowledgments can form a pattern with a fixed interval. This being the key indicator or a Beacon during this project.

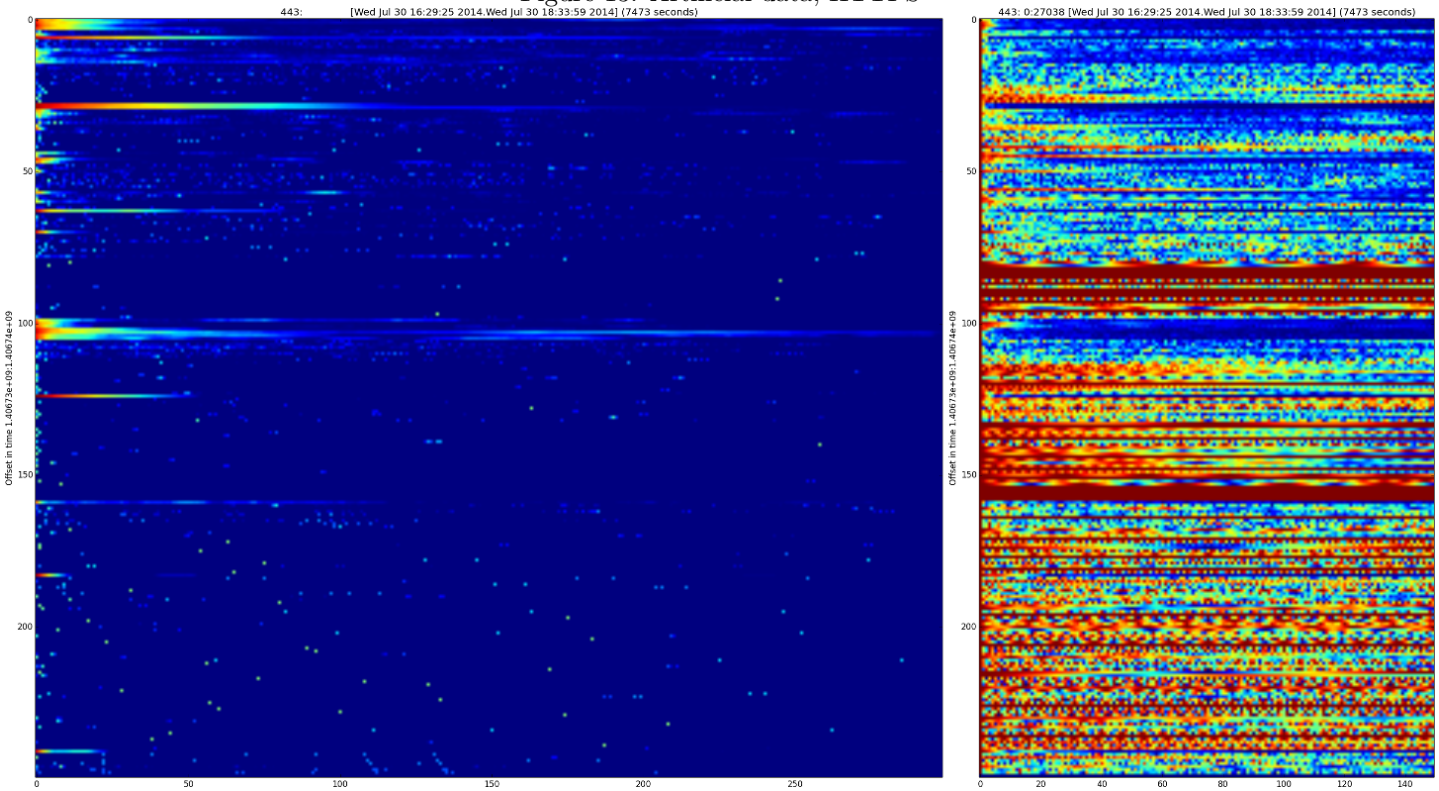
In Figure 13 the HTTPS traffic can be seen, which was expected to contain some beacons in the presence of user activity masking them.

From visual analysis it seems unlikely persistent beacons have been captured, only specks on the correlation plot, and no discernible patterns in the frequency spectrum. Do to limited data no conclusions can be drawn on generic HTTPS or typical web traffic, however this does show that not all encrypted HTTP traffic will be indicated as beacon using these methods.

In order to conclude about generic web traffic, largely interactive web applications, social media and advertising generated traffic more data would have to be collected and investigated. During the project an inquiry was made however no such data could be available within the allotted time.

deviation

Figure 13: Artificial data, HTTPS



5.6 About parameters

There are several parameters, all influence detection results and output in some fashion, optimum values are context specific but some observations can be made in general.

- Depending on the classification rule the input traffic may require additional filtering, this can be done offline using tcpdump/wireshark.
  When the intention is to look out outgoing TCP connection attempts filtering out any other traffic can reduce processing time.
- The resolution for the Time stamp to time line conversion greatly affects the quality of the analysis.
  Should a network link be saturated any traffic could resemble beacon traffic, sending it out as the highest possible frequency. This rate is likely to be a relative constant and as such could superimpose beacon like behavior on arbitrary persistent streams.
- The waterfall plotting uses a fixed window interval, and can be misalligned analysis on data being processed which may yield sub optimal results for short lived/burst beacons.
  The stride can be configured to do analysis on partially overlapping data to find sharper results. This in exchange for partially redundant output data.

5.7 Experimental observations

In conclusion the following observations have been made during experimentation:

1. Both auto correlation and frequency analysis reveal the presence of beacon data
2. Beacons can be detected on activity of some Malware agents
3. Legitimate traffic can show beacon behavior, though all observed occurrences could be explained, and anticipated.

6 Conclusions

- Can traffic dumps be used to detect beacons produced by Malware?
  Traffic dumps can be used to detect beacons. Visualization of correlation and frequency show distinctive patterns in presence of known Malware beacons.
- Can partitioning traffic on common/shared features increase signal, and beacon, clarity?
  By partitioning traffic it is possible to reveal faint signals in a larger data set. Using context relevant knowledge can aid in detecting specific beacons and reducing the search space by separating known good traffic.
- Is it possible to detect Malware in the presence of legitimate beacons?
  While both types can show identical patterns in the detection stage the diversity of beacons can be applied to judge the results, from the data sets available for this project the beacon traffic shows consistent behavior over time, distinguishable from each other and typical user traffic.

By applying machine learning and data mining techniques it should be possible to automate the process of judging the nature of a detected beacon, this currently remains further research to prove and implement.

Though the proof of concept does not do fully autonomous beacon detection it does aid in investigating traffic, by auditing a PCAP sample it is possible to find beacons in a non invasive manner. Tough a packet dump has to be made on a key point in a network no configuration changes are required on devices in the network, allowing for audits on live network segments to find Malware while not relying on traditional signatures.

6.1 Practical consideration

Initial results show that encryption does not prevent a beacon signature from being detected, though confirmation of a stream is affected by encryption, encoding and volume of data.

The methods in this paper rely on interactive exploration of suspect beacons, and requires more automation to be deployed to larger, heterogeneous networks.

In an enterprise environment the current approach does seems promising, as a tool to audit small network segments for potential intrusions.

Though there are methods of evading detection by the approach presented in this paper, the existence of threats not deploying these techniques should be taken into consideration. The method does not rely on classic signatures, replacing this requirement with the generic fingerprint of a beacon. Apart from detecting beacons it might be an additional tool to gain insight into the workings of a network, by identification of typical behavior, common beacons, and traffic distribution.

7 Future work and improvements

There are several improvements and extensions that could improve beacon detection results and usability of the proof of concept.

Introducing automated detection of 'interesting' results, as opposed to visual inspection, would allow a much higher level of automation, running a configured set of classifiers and triggers to generate a periodic report on likely beacons.

Automate more of the parameters, this could increase both results and efficiency during an audit.

Closer inspection and experimental implementation of unsupervised classifiers could increase automation, being able to introduce machine learning brings beacon detection closer to real time, enterprise deployment.

Handling of sparse data, traffic from hosts with limited uptime is currently stretched. Removing these gaps or splitting into separate classes might help to improve results, reducing the required amount of traffic to positively identify beacons.

Due to time and data constraints only a limited collection of data sets were available, an evaluation with larger dumps of every day data would give insight into the potential for active monitoring and effectiveness of the classification method.

References

[1] Yu Liang, Guojun Peng, Huanguo Zhang, and Ying Wang. An unknown trojan detection method based on software network behavior. *Wuhan University Journal of Natural Sciences*, 18(5):369–376, 2013.

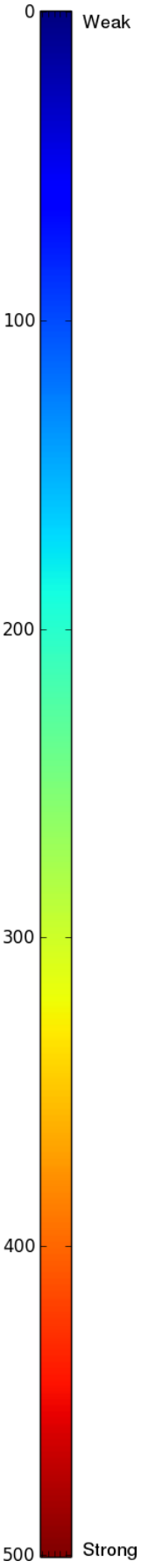
[2] Gregoire Jacob, Ralf Hund, Christopher Kruegel, and Thorsten Holz. Jackstraws: Picking command and control connections from bot traffic. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 29–29, Berkeley, CA, USA, 2011. USENIX Association.

[3] Christian Rossow and Christian J. Dietrich. ProVeX: Detecting Botnets with Encrypted Command and Control Channels. In *Proceedings of the 10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, July 2013.

[4] Yong Qiao, Yuexiang Yang, Jie He, Chuan Tang, and Yingzhi Zeng. Detecting p2p bots by mining the regional periodicity. *Journal of Zhejiang University - Science C*, 14(9):682–700, 2013.

A Graphical scale

The images in this paper generated by pyplot use the, default, color scale as shown on the right of this page. In this rendering the data ranges from 0 to 500, the rendering internally normalizes data to utilize the entire color range regardless of the input data. This normalization is done in regards to the entire image.



Color scale pyplot

B Classification rule table

src	The IP source address.
dst	The IP destination address.
srcdst	The IP source and destination, sorted to match both directions of a stream.
len	Length op the IP payload.
datalen	With two arguments, Boolean 'the packet size falls within this range'.
prot	The protocol in the IP payload.
prot=	With one argument, Boolean 'The protocol in the IP payload being a matches . . . '.
ttl	The IP Time To Live field.
tos	The IP Type of Service/Differentiated Services field.
icmp	The packet is ICMP traffic.
icmp.type	The ICMP type.
icmp.code	The ICMP code.
tcpsport	TCP source port.
tcpdport	TCP destination port.
tcplowport	The minimum of the TCP source and destination port.
tcphighport	The maximum of the TCP source and destination port.
udpsport	UDP source port.
udpdport	UDP destination port.
udplowport	The minimum of the UDP source and destination port.
udphighport	The maximum of the UDP source and destination port.
entropy.zlib	The rate of compression using ZLIB.

C Getting the code

The code will be published on `https://github.com/LeendertD/Beacondetection.git`, in order to see it in action PCAP data has to be collected, some applications which generate traffic to see beacons in a controlled environment:

- SSH working on a server, listing files, initiating system updates
- SSH running top for a longer time
- Running command line clients for the MPD deamon, over SSH
- Browsing the web
- Watching parody videos on the internet
- Connecting multiple mobile devices running various apps, news readers, weather information, email retrieval etc.

D Libraries

The following libraries are used by the proof of concept.

- dpkt
  - Author : Dug Song <dugsong@monkey.org>
  - Copyright : Copyright (c) 2004 Dug Song
  - License : BSD
  - Url : `http://dpkt.googlecode.com/`
  - Version : 1.8
- numpy
  - The NumPy homepage : <http://www.scipy.org>
  - Version : 1.8.1
- matplotlib
  - Version : 1.3.1
- pyplot
  - Url : `http://matplotlib.org`
  - Version : 1.3.1
- scipy.signal
  - Url : `http://docs.scipy.org`
  - Version : 0.14.0
- Tkinter
  - Url : `http://docs.python.org/library/Tkinter`
  - Version : 81008
- pypcap
  - Author : Dug Song <dugsong@monkey.org>
  - Copyright : Copyright (c) 2004 Dug Song
  - License : BSD license
  - Url : `http://monkey.org/~dugsong/pypcap/`
  - Version : 1.1
- zlib
  - Url : `http://docs.python.org/library/zlib`
  - Version : 1.0
- time
  `http://docs.python.org/library/time`