CS422 Project: Searching for Markovian Malware C&C on Twitter

Maxime Augier and Gowthami Ramasamy

May 4, 2014

1 Project steps

Data collection We used the archive.org twitter stream and uploaded it to the lab cluster hdfs. The original file is a tar archive containing bzipped fragments of text files. The chunks were small, in the order of 1.5MB, so we repacked them into bigger chunk. After repacking gzip compression was used. Although the gzip format is inferior as it does not allow to seek within the file, bzip2 compression is significantly slower. One could also have concatenated the bzip fragments by fiddling with the headers but we preferred the safe approach.

For this operation we used the python code under the "extraction" directory. It offers a generator-based API to process the tweets. There is also a script to upload data to a MongoDB instance.

Early assessment Recognizing base64 turned out to be harder than expected. Regex matching yielded a significant number of false positives.

We did, however, identify one odd pattern: sequences of nucleotids (ATGC letters) in long strings. We found a couple of users tweeting these sequences as "Human DNA pieces". While we initially thought this may be a covert communication channel, the sequences we tested turned out to be genuine human DNA, present in the publicly-available BLAST nucleotid database.

Language classification Before trying to apply any language model, we will try to tag our messages with one of the readily available language recognition frameworks. Later on, our work will cover two separate corpuses, the full one and the one consisting of English-only languages.

Model building MapReduce is especially well fitted to compute the probability distributions over a Markov model. We will compute our models both on letters (n-grams) and tokenized words (for the english corpus only). We also need to test several tokenization models.As a part of n-gram model and tokenized words, a MapReduce program has been developed, the logic of the model follows

Mapper [n-gram]:
Input: The twitter stream, without any pre-processing
Functionality: 1. The twitter stream is parsed and the language field - 'Lang' and Tweet text 'Text' gets extracted.
2. The language fields is used according to the model that we are build [full one or English only]
3. The tweet text further parsed into n-grams, without elimination any bytes.
Output: Each n-gram will be passed as a key to the reducer. The value is one. n-gram, one
Mapper [tokenized words]:
Input: The twitter stream, without any pre-processing
Functionality: 1. The twitter stream is parsed and the language field - 'Lang' and tweet text 'Text' gets extracted.
2. The language fields is used according to the model that we are build [full one or English only]
3. The tweet text further parsed into words, tweet texts always contains special characters,plural forms which should be eliminated. So regex pattern is used to extract only alpha-numeric characters
Output: Each word will be passed as a key to the reducer. The value is one. word, one
Reducer [same for n-gram and tokenized words]:
Input: Key, Value
Functionality: Count the Key
Output: Key Count

Another option we want to consider (but less likely to give results) is encoding messages in deliberate grammar or orthography mistakes, trough permutation of simple characters. Decoding would be performed by a standard spell checker like aspell.

Channel building We propose to use an inverted form of Huffman coding to encode arbitrary bits into human-looking chains. The algorith wil work as follows:

First, prune the model to keep it down to a reasonable size, by excluding uncommon words.

Then, for every state in the chain, build a Huffman tree over the distribution for the next symbol.

To encode, begin with an empty starting state in the Markov chain. Perform a Huffman decoding operation on the tree of the current state, which will consume an arbitrary number of bits and produce a symbol. Output the symbol, compute the new state, and repeat until no bits are left, or the chain picks a terminating symbol, or the maximum output size is exceeded.

To decode, begin with the same empty state; for each token encoutered, perform Huffmann compression, repeat until the message is complete.

Entropy estimation Once the encoding channel is working, we shall make it run with random input bits, and measure on average how many bits fit in each tweet, for all our models (n-grams, english n-grams, english words). Combined with compression, we will estimate how many messages would be required for typical C&C operations, checking in how many bits we can fit (for instance) a DDoS attack target or a typical shady url.

Detection The difficult part will be extracting suspicious messages conforming too well to our model.

One first technique will be to use different Markov models of different orders, and see if for given accounts, their distribution of tweets follows a low-order model anormally better than a high-order one.

We will also use the n-gram table to compute likelihood for a set of hashtags, and try to extract randomly-generated hashtags from it. We will apply the same n-gram analysis to messages contents, and compare with the first naive regex-based classifier.

2 Resource usage

As much as we can get.

Maxime can provide storage space and compute time on LACAL clusters for low-priority jobs, can be used in case of backups. We'll have to see if it's not too costly to replicate from the common Twitter data source.

3 Milestones

01.04.2014 Have a sample set of data ready in case the global collection effort does not work (MA), write a MapReduce implementation to build n-gram datasets (GR). Write MapReduce simple matchers (MA).

08.04.2014 Results from early assessment and detection

22.04.2014 Proof of concept for stealthy channel, MapReduce chain trimmer (GR), compressor/decompressor sample code (MA)

06.05.2014 Integrate final data sources and start looking for actual bots

13.05.2014 Final report