Malware detection based on domain generation algorithms

A Project Report
Presented to
Prof. Mark Stamp
Department of Computer Science
San José State University

In partial fulfillment
of the requirements for the Class
CS 266

By
Kushal Palesha & Tejas Saoji
December 2016

Contents

List of Figures

1 Introduction

In our heavily connected world, the threat from network based attacks is very real and rampant. Botnets make up a significant proportion of such attacks. Botnets are a network of computers that are infected with a malware (botware) and a command & control(C&C) server. The botware resides in the host computer and periodically tries to contact the C&C server to receive commands to carry out various kinds of attacks. To avoid detection, the C&C server does not register a static domain, but instead keeps changing it's address, and registers different domain names at different times. This is done using domain generation algorithms (DGA). For the botware to communicate with the C&C server, it also has to generate these domain names using the same DGA as the one used by the C&C server. In this report, we discuss the use of machine learning to detect these botware based on the domain names it generates.

2 Background

2.1 What is DGA?

A domain generation algorithm (DGA) "is an algorithm used to generate random looking domain names for any given time based on predefined variables" [1]. These algorithms are heavily used in botnet malware (botware) to commuincate with their command & control (C&C) servers. This allows the C&C server to avoid detection by registering itself on different domains at different times.

An example of one such algorithm from [2] for the cryptolocker botnet is shown in Fig. 1. Here, the year, month, and day act as a common seed value between the botware and

```
def generate_domain(year, month, day):
    domain = ""
    for i in range(16):
        year = ((year ^ 8 * year) >> 11) ^ ((year & 0xFFFFFFF0) << 17)
        month = ((month ^ 4 * month) >> 25) ^ 16 * (month & 0xFFFFFFF8)
        day = ((day ^ (day << 13)) >> 19) ^ ((day & 0xFFFFFFFE) << 12)
        domain += chr(((year ^ month ^ day) % 25) + 97)
    return domain
```

Fig. 1: DGA used by cryptolocker

the C&C , thus aiding communication between them. Similarly, other botnet malware use different seed values like time of day, top twitter trends, etc.

2.2 Approaches to detect malware based on DGA

- Anlayze a high volume of Non-existant domain (NXDOMAIN) responses to a single source.
  - A high volume of NXDOMAIN requests in a specific time window points to suspicious activity.
  - However, newer botware make communication attempts at less frequent intervals making it complicated to do network analysis of this kind.
- Reverse engineer the DGAs and create a blacklist of domain names.
  - This is not very feasible since the blacklist can become too large, making it diffcult to maintain them for security specialists.
- Machine learning to detect *algorithmically generated domains* (AGDs).
  - We can extract features from the domain names to train machine learning models and use them to detect AGDs.

– We explore decision tree and random forest classifiers in this report.

3 Our work

3.1 About the data

The data we have is from a GitHub repository [3], which contains a lot of information for someone to get started with AGD detection. It contains the reverse engineered algorithms for 8 botnet families (and 100,000 domain names generated using these algorithms) along with a huge set of benign domain names from openDNS and alexa. The 8 botnet families are:

1. Rovnix
2. Matsnu
3. Conficker
4. Cryptolocker
5. Zeus
6. Tinba
7. Ramdo
8. Pushdo

3.2 Feature selection

Feature selection involved figuring out what features to use for training and eventually scoring. For calculation of features, we only considered the *2nd-level string*. As defined in [4], the *2nd-level string* is "the substring between the last and second-last dots before the public suffix." For instance, in s1.s2.example.com and example.co.us, *example* is the *2nd-level string*. For each type of AGD, we could have used features that potentially had a good score due to specific traits of the corresponding DGAs. But we were on the lookout for features that we could use across all the botnet families.

```
1vz89zm5b2e981bgfhqdzbke3m.org
1jjcgb11mmtru8r7xsa1xqk8zh.biz
```

Fig. 2: Domains generated by the DGA in zeus

For example, Zeus generates domain names as shown in Fig. 2. Here we can think of featuers like length, entropy, n-grams and "randomness". Where,

- Entropy is calculated as:

$$-\sum_{i=0}^{n} p(i) * \log(p(i)) \tag{1}$$

  where,

$$p(i) = \text{probability that a character appears in the given domain name.}$$
$$n = \text{the number of unique characters in the domanin name.}$$

- n-gram score helps us identify how similar the domain name under consideration is to the benign domain name dataset.

- We define "randomness" as the count of switches made from an alphabet to a digit (and vice-versa) in the domain name.

```
theirtheandaloneinto.com
thathistoryformertrial.com
```

Fig. 3: Domains generated by the DGA in rovnix

Another example we would like to present is of the domain names generated by the rovnix DGA as shown in Fig. 3. We can clearly see here that the "randomness" score we came up with is rendered useless for these domain names.

An interesting detail about the rovnix DGA is that it randomly picks out words from the US Declaration of Independance to generate a domain name. We could imagine training a model on the Declaration of Independance to score how similar a domain name is to it. But we would only do this if our aim was just to detect rovnix.

3.2.1 Features we used in our model

1. length
2. entropy
3. tri-gram score
4. quad-gram score

Calculation of entropy is explained in (1).

We built a tri-gram and quad-gram model using 100,000 benign domains from the alexa dataset. For a domain name, it's n-grams (n = 3 & 4) are generated and the score is the number of matches against the tri-gram/quad-gram model we have.

3.3 Decision Tree and Random Forest

Decision trees seemed to be an intuitive solution for the problem at hand [5]. Also, they are very simple to understand and implement. Python's sklearn library was used to create decision tree and random forest models.

3.3.1 Models for each botnet

A decision tree and a random forest model was trained for every botnet. The training set consisted of 80% of the botnet's AGD dataset and 80% of the alexa legit domain names dataset. Both the datasets of size 100,000. The rest 20% from both the datasets was used for testing.

For 7 out of the 8 models, the feature that provided the most information gain in the first split was length. But, in case of conficker, tri-gram provided the most information gain. This makes sense since the domains generated by conficker are variable in length and are similar in length to benign domains as opposed to other botnets that generate either longer domain names or domain names of same length.

3.3.2 Combined Model

- A decision tree and a random forest model was trained for a combined AGD dataset from every botnet.
- The size of the dataset set was fixed at 100,000. The size of a chunk of domain names picked up from every botnet is given by: $\frac{100{,}000}{8(\text{Number of botnets})}$
- The combined model was trained on the entire combined AGD dataset. The model was then tested on each of the botnet's AGD dataset combined with 100,000 alexa benign domain names dataset.
- For the combined model, the feature that provided the most information gain was tri-gram.

Model	Decision Tree	Random Forest
Rovnix	0.967475	0.983175
Matsnu	0.992075	0.996075
Pushdo	0.94515	0.953975
Ramdo	0.99745	0.998075
Conficker	0.8781	0.90505
Cryptolocker	0.9740	0.9807
Tinba	0.9848	0.9884
Zeus	0.9999	0.99985

Fig. 4: Per model accuracy

3.4 Scores

- The per-model accuracy for all the models obtained from the training & testing explained in the section 3.3.1 are given in the Fig. 4
- The accuracy for the combined model trained and tested in section 3.3.2 are given in the Fig. 5
- The models obtained in the 3.3.1 were also tested on the openDNS-randomm and openDNS-top benign datasets to get the false-positives. The testing was conducted on 1000 samples from each dataset. The per-model accuracy for all the models are given in the Fig. 6

Test data	Combined models	
	Decision Tree	Random Forest
Rovnix + Alexa	0.9238	0.9594
Matsnu + Alexa	0.9806	0.9960
Pushdo + Alexa	0.8842	0.8928
Ramdo + Alexa	0.9918	0.9939
Conficker + Alexa	0.8782	0.8689
Cryptolocker + Alexa	0.9715	0.9746
Tinba + Alexa	0.9816	0.9832
Zeus + Alexa	1.0	0.9980

Fig. 5: Accuracy for the combined model for all botnet families

Model	Test data			
	OpenDNS - random		OpenDNS - top	
	Decision	Random Forest	Decision Tree	Random Forest
Rovnix	0.954	0.950	0.998	0.998
Matsnu	0.992	0.991	1.0	1.0
Pushdo	0.937	0.94	0.944	0.942
Ramdo	0.997	0.998	0.999	1.0
Conficker	0.883	0.912	0.838	0.863
Cryptolocker	0.975	0.978	0.996	0.997
Tinba	0.987	0.989	0.995	0.997
Zeus	1.0	1.0	1.0	1.0
Combined	0.882	0.903	0.913	0.932

Fig. 6: Per model accuracy for benign domains

4 Conclusion

The individual models performed really well. However, our aim here was to create a general model for all the 8 datasets. Hence, we trained a decision tree on a combined dataset. The combined model also gave us good accuracy scores. The number of false-positives for the individual models as well as the combined model were also less.

References

[1] A. V. A. Chiu, "Threat spotlight: Dyre/dyreza: An analysis to discover the dga," 2015 (accessed November, 2016). [Online]. Available: http://blogs.cisco.com/security/talos/threat-spotlight-dyre

[2] Wikipedia, "Domain generation algorithm — wikipedia, the free encyclopedia," 2016 (accessed November, 2016). [Online]. Available: https://en.wikipedia.org/w/index.php?title=Domain_generation_algorithm&oldid=743520067

[3] A. Abakumov, "Dga," 2015 (accessed Novemebr, 2016). [Online]. Available: https://github.com/andrewaeva/DGA

[4] M. Mowbray and J. Hagen, "Finding domain-generation algorithms by looking at length distribution," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, Nov 2014, pp. 395–400.

[5] M. Stamp, *Machine Learning with applications in Information Security*, 2016 (unpublished), ch. Many Mini Topics, pp. 164–168.