

REQUEST DEMO (http://pages.endgame.com/request-demo-website.html)


OUR BLOG

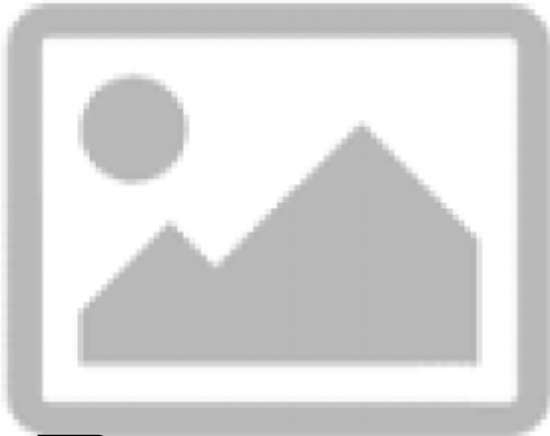
Endpoint Security, Simplified

ALL (/BLOG) TECHNICAL BLOG (/BLOG/TECHNICAL-BLOG)

EXECUTIVE BLOG (/BLOG/EXECUTIVE-BLOG)

Employing Latent Semantic Analysis To Detect Malicious Command Line Behavior

 Jonathan Woodbridge (/Our-Experts/Jonathan-Woodbridge) FEBRUARY 17, 2016



3SITE.HTML)

Detecting anomalous behavior remains one of security's most impactful data science challenges. Most approaches rely on signature-based techniques, which are reactionary in nature and fail to predict new patterns of malicious behavior and modern adversarial techniques. Instead, as a key component of research in Intrusion Detection, I'll focus on command line anomaly detection using a machine-learning based approach. A model based on command line history can potentially detect a range of anomalous behavior, including intruders using stolen credentials and insider threats. Command lines contain a wealth of information and serve as a valid proxy for user intent. Users have their own discrete preferences for commands, which can be modeled using a combination of unsupervised machine learning and natural language processing. I demonstrate the ability to model discrete commands, highlighting normal behavior, while also detecting outliers that may be indicative of an intrusion. This approach can help inform at scale anomaly detection without requiring extensive resources or domain expertise.

A Little Intro Material

Before diving into the model, it's helpful to quickly address previous research, the model's assumptions, and its key components. Some previous work focuses solely on the commands, while some use a command's arguments as well to create a richer dataset. I focus only on commands and leave the arguments for future work. In addition, this work focuses on server resources, as opposed to personal computers, where command line is often not the primary means of interacting with the machine. Since we are focusing on enterprise-scale security, I leave applications of this model for personal computers to future work. I also focus on UNIX/Linux/BSD machines due to current data availability.

Authors in previous work often rely on the uniqueness of their set of commands. For (an overly simple) example, developer A uses emacs while developer B uses vi, hence it is an anomaly if user A uses vi. These works come in many forms including sequence alignment (similar to bioinformatics), command frequency comparisons, and transition models (such as Hidden Markov Models). One common issue across many of these works is the explosion in the number of dimensions. To illustrate this, how many commands can you type from your command line? My OS X machine has about 2000 commands. Now add Linux, Windows and all the uncommon or custom commands. This can easily grow to the order of tens of thousands of commands (or dimensions)!

In addition to dimensionality challenges, data representation further contributes to the complexity of the data environment. There are many ways to represent a bunch of command sequences. The most simple is to keep them as strings. Strings can work for some algorithms, but can lack efficiency and generalization. For example, assumptions of Gaussian distributions don't really work for strings. In addition, plugging strings into complex models requiring mathematical operators like matrix multiplies (i.e. Neural Nets) are not going to work. Often, people use one-hot encoding (<https://en.wikipedia.org/wiki/One-hot>) in order to use more complicated models with nominal data, but this still suffers from the curse of dimensionality as the number of unique names

increases. In addition, one-hot encoding treats each unique categorical value as completely independent from other values. This, of course, is not an accurate assumption when classifying command lines.

Fortunately, dimensionality reduction algorithms can counteract the growing number of dimensions caused by one-hot encoding. Principle Component Analysis (PCA) (https://en.wikipedia.org/wiki/Principal_component_analysis) is one of the most common data reduction techniques, but one-hot encoding doesn't follow Gaussian distributions (for which PCA would optimally reduce the data). Another technique is binary encoding (https://en.wikipedia.org/wiki/Truncated_binary_encoding). This technique is generic, making it easy to use, but can suffer in performance as it doesn't take domain specific knowledge into account. Of course, binary encoding is typically used for compression, but it actually works fairly well in encoding categorical variables when each bit is treated as a feature.

So how can we reduce the number of dimensions while utilizing domain knowledge to squeeze the best performance out of our classifiers? One answer, that I present here, is Latent Semantic Analysis (https://en.wikipedia.org/wiki/Latent_semantic_analysis) or LSA (also known as Latent Semantic Indexing or LSI). LSA is a technique that takes in a bunch of documents (many thousands or more) and assigns "topics" to each document through singular value decomposition (SVD). LSA is a mature technique that is heavily used (meaning lots of open source!) in many other domains. To generate the topics, I use man pages and other documentation for each command.

The assumption (or hypothesis) is that we can represent commands as a distribution of some limited and overlapping set of topics that proxy user intent, and can be used for detecting anomalous behavior. For an overlapping example, mv (or move) can be mimicked using a cp (copy) and an rm (delete). Or, from our previous example, emacs and vi do basically the same thing and probably overlap quite a bit.

LSA on Command Lines

To test the hypothesis, I need to evaluate how well LSA organizes commands into topics using the text from man pages. I use around 3100 commands (and their respective man pages) to train my LSA model. Next, I take the top 50 most used commands and show how well they cluster with other commands using cosine similarity (https://en.wikipedia.org/wiki/Cosine_similarity). I could visualize even more commands, but the intent is to show a coherent and understandable clustering of commands (so you don't have to run *man* a hundred times to understand the graphic). Similarly, only edges with weights greater than .8 are kept for visualization purposes (where cosine similarity is bounded in [0,1] with 1 as the most similar).



If you look closely you can see clusters of like commands. This was done totally unsupervised. No domain experts. That's pretty cool.

That's a great first step, but how can we use this to classify command lines? The idea is to average intent over small windows of commands (such as three, ten or fifty commands) and to use this as a feature vector. For example, if the user types *cd*, *ls*, *cat*, we find the LSA representation of each command from their corresponding man pages. Assuming we model commands with 200 topics, we take each of the three 200 point feature vectors and do a simple mean to get one 200-point feature vector for those three commands. I tried a few other simple ways of combining features vectors, such as concatenating, but found mean works the best. Of course, there are probably better more advance techniques, but this is left to future work. We can generate a large training and testing set by applying a sliding window over a user's command sequence. For fun, I use the one-class SVM from sklearn (<http://scikit-learn.org/stable/modules/svm.html#svm-outlier-detection>) and employ data from the command line histories of eleven colleagues. I create a total of eleven models trained on each respective user. These are one-class models, so no positive (i.e., anomalous) examples are in any of the training. I run ten folds using this setup and average the results. For each fold, I train on 50% of the data and keep 50% of all commands from each user for testing. I admit this setup is not completely representative of a real world deployment as the numbers of anomalous command sequences far outweigh the numbers of normal. I also do the most basic preprocessing such as stemming and removal of stop words using NLTK (<http://www.nltk.org/>) and stop_words (<http://pypi.python.org/pypi/stop-words>) (both can be installed through pip) on the man pages before running LSA to create topics.

For a baseline, I run the same experiment using one-hot, binary, and PCA encoded feature vectors for each command. I take the mean of these feature vectors over windows as I did before.

I run the experiment on windows of three, ten, and fifty and display the corresponding receiver operating characteristic (ROC). The ROC curve describes how well the eleven user models identified the held out commands. One caveat is that not all commands are represented in the man pages. For simplicity and reproducibility, I currently ignore those commands and leave that to future work.



The first image is not so great. Here we are displaying the ROC for a window size of 3. Except for PCA, everything is about the same. LSA is marginally better than one-hot and binary encoding. However, with such a small window size, you're best off using PCA.



As we increase the window size, the results get a little more interesting. One-hot and LSA encoding get the largest boost in performance, while PCA degrades. As I stated earlier, PCA is a bad choice for reducing categorical variables, so this drop-off is not overly surprising. The other interesting point is that larger windows make a better classifier. This is also not very surprising as the users in this study are very similar in their usage patterns. Larger windows incorporate more context allowing for a more informative feature vector.



The results get even better for LSA with a window size of fifty. Of course, we could enrich our features with command line arguments and probably get even better results, but we are already doing really well with just the commands.

Final Thoughts

LSA works very well in clustering the command line arguments, serving as a useful proxy for user intent and more importantly detecting anomalous behavior. This was completely unsupervised making the model an easy fit for a real world deployment where labels often don't exist. One assumption of this post is the training data is not polluted (i.e., does not contain command line sequences from other users). Also, this data comes from the command lines of software developers or researchers who are very similar in their usage patterns. This means a command line pattern may be common across several users leading to false negatives in this experimental setup. Hence, we may see much **better** results when we feed a malicious user's command lines to a normal user's model. In addition, we could possibly create a more robust model by using the command histories of multiple normal users (instead of building a model from a single user). I will leave the answers to these questions to another post!

3SITE.HTML)

REQUEST DEMO (http://pages.endgame.com/request-demo-website.html) BLOG (/blog)
(/#facebook) (/#twitter) (/#linkedin) 800.550.1250 (tel:703-650-1250)

RELATED POSTS

(/blog/technical-blog/endpoint-malware-detection-hunt-real-world-considerations)



(/blog/technical-blog/endpoint-malware-detection-hunt-real-world-considerations)

AUGUST 14, 2016

Endpoint Malware Detection for the Hunt: Real-world Considerations
(/blog/technical-blog/endpoint-malware-detection-hunt-real-world-considerations)

VIEW DETAILS (/BLOG/TECHNICAL-BLOG/ENDPOINT-MALWARE-DETECTION-H

3SITE.HTML)



(/blog/technical-blog/stopping-certified-malware)

NOVEMBER 09, 2017

Stopping Certified Malware (/blog/technical-blog/stopping-certified-malware)

[VIEW DETAILS \(/BLOG/TECHNICAL-BLOG/STOPPING-CERTIFIED-MALWARE\)](#)

(/blog/technical-blog/introducing-ember-open-source-classifier-and-dataset)



ember
an open source malware classifier and dataset

(/blog/technical-blog/introducing-ember-open-source-classifier-and-dataset)

APRIL 16, 2018

Introducing Ember: An Open Source Classifier and Dataset (/blog/technical-blog/introducing-ember-open-source-classifier-and-dataset)


REQUEST DEMO (http://pages.endgame.com/request-demo-website.html)

BLOG (/blog)

703.650.1250 (tel:703-650-1250)

VIEW DETAILS (/BLOG/TECHNICAL-BLOG/INTRODUCING-EMBER-OPEN-SOUR

REQUEST DEMO (http://pages.endgame.com/request-demo-website.html)



(/blog/technical-blog/using-deep-learning-detect-dgas)

NOVEMBER 18, 2016

Using Deep Learning to Detect DGAs (/blog/technical-blog/using-deep-learning-detect-dgas)

VIEW DETAILS (/BLOG/TECHNICAL-BLOG/USING-DEEP-LEARNING-DETECT-DC

ALL (/BLOG) TECHNICAL BLOG (/BLOG/TECHNICAL-BLOG)

EXECUTIVE BLOG (/BLOG/EXECUTIVE-BLOG)

3SITE.HTML)

REQUEST DEMO (HTTP://PAGES.ENDGAME.COM/REQUEST-DEMO-WEBSITE.HTML)

Get in touch.

703.650.1250 (tel:703-650-1250)

CONTACT US
703-650-1250

SIGN UP FOR OUR NEWSLETTER

Email Address

Submit

Explore

3SITE.HTML)

[Why Endgame \(/why-endgame\)](#)

[REQUEST DEMO \(http://pages.endgame.com/request-demo-website.html\)](#) [BLOG \(/blog\)](#)

[Platform \(/platform\)](#) [703.650.1250 \(tel:703-650-1250\)](#)

[Company \(/company\)](#)

[Resources \(/resource\)](#)

[News \(/news\)](#)

Cyber Security Dictionary

[Threat Hunting \(/resource/solution-brief/pdf/automated-threat-hunting\)](#)

[Security Service \(/endgame-services\)](#)

[Ransomware \(/blog/technical-blog/wcry-wanacry-ransomware-technical-analysis\)](#)

[Machine Learning \(/blog/machine-learning-you-gotta-tame-beast-you-let-it-out-its-cage\)](#)

[Security News \(/news\)](#)

[Endpoint Protection \(/why-endgame\)](#)

[Cyber Hunting \(/blog/technical-blog/how-hunt-file-path-less-traveled\)](#)

[Cybersecurity Financial Services \(/resource/solution-brief/endgame-financial-services-booklet\)](#)

[Automated Hunt \(/resource/white-paper/sans-white-paper-automating-hunt-hidden-threats\)](#)

[Cyber Attack \(/blog/hackers-guide-not-having-your-passwords-stolen\)](#)

Connect

[3101 Wilson Blvd](#)

[Arlington, VA 22201](#)

[703-650-1250 \(tel:703-650-1250\)](#)

[Contact Us \(https://www.endgame.com/contact\)](#)

[Request a Demo \(http://pages.endgame.com/request-demo-website.html\)](#)

© Endgame 2018

[in](#) [t](#) [f](#)

[\(https://www.endgame.com/\)](#) [\(https://www.endgame.com/\)](#) [\(https://www.endgame.com/\)](#)

(/)