**Abstract**

A Practical Application of Machine Learning-Based Classification Techniques to Proactively Identify Insider Threats

Joshua D. Bowen

Northern Arizona University

Prescott Valley Campus

Insider incidents are on the rise. Just like the high-profile security breaches such as Snowden, thousands of insider-perpetrated security breaches occur in United States businesses every day. While current commercial software can monitor, log, and prevent access to designated files and directories, it remains difficult to predict and prevent unauthorized insider usage. Due to the gaps in research in this area, the focus of this study is to more accurately predict insider threats within a terminal environment.

Linux was chosen specifically because of its ubiquity on commercial servers around the globe. Amazon’s Machine Learning (AML) service has been selected to analyze the data, reduce the necessary computing power, and to minimize human factors considerations in the design of the machine learning architecture. AML uses multinomial logistic regression for multi-class classification and uses the stochastic gradient descent optimization technique.

The method adapted is to utilize the AML software. It will be trained on a dataset comprised of normal user situations, crafted mistakes, and malicious activity. After providing the training dataset, the software will be instructed to make predictions against similar datasets to verify accuracy. It will then be tested against a human actor that will simulate multiple different roles, and test predictions in a high-fidelity simulation.

If it can be demonstrated that ML software can accurately identify and predict insider threats, this research could be a foundation for future cyber security software architectures. Other opportunities for research in this area would include ML applications in intruder and malware detection.

**Data** **Gathering**

There are three main sources of data used for the machine learning model. The first is an un-edited history file from the test system. Second is a list of commands assembled during this research. The third is the UNIX User Dataset from Purdue University. Simple programs were developed to remove unnecessary lines and markup from the two gathered datasets.

**Methods**

The method utilizes the Amazon Machine Learning software. It will be trained on a dataset comprised of normal user situations, crafted mistakes, and simulated malicious activity. The software will use the training dataset to make predictions against similar datasets to verify accuracy. It will then be tested against a human actor to test predictions in a high-fidelity simulation. A testing dataset of 100 commands will be given to the model for batch prediction and will be evaluated for statistical accuracy. The python timeit module will be used to test for latency between the user entering their command and when it executed. Four categories will be analyzed; code only, execution only, code + execution, and code + execution + AML. To have a baseline for the network speed, the pings against the AML endpoint will be analyzed.

**Results**

The batch predictions have shown that the model is 100% accurate at predicting normal use and has a 30% rate of false negatives for malicious and mistake commands.

I HAVE SOME TROUBLE WITH THE PREVIOUS PARAGRAPH. I’D PREFER TO SEE FALSE POSITIVE AND FALSE NEGATIVE RATES FOR DIFFERENT CATEGORIES. I GUESS IT’S JUST THE WAY YOU WORDED IT THAT BOTHERS ME.

The batch predictions show that the model is 100% accurate at predicting commands in the normal use category. The model has a 30% rate of false negatives for both malicious and mistake commands. It did not predict false positives in any category.

The results of the high-fidelity simulation indicate that the software is partially effective at capturing all input. The participants were able to escape it and, after that, became untraceable to the system. This indicates that further work is needed in this area.

In the latency tests, results show a small discrepancy of an additional ~0.25ms between the combined times of code only and execution only, compared with the code + execution time. There is a major difference between the network speed, execution, and the time it takes to interact with the AML of over ~440ms.

Overall the process has indicated that a prototype like this can capture near 80% of the commands and suggests that further work on this model can produce far more reliable results.

This paragraph confuses me a little. You indicate that the prototype captures 80% of the commands. Are these any and all commands, are they simply commands that could be considered malicious, and so forth. Assuming that it relates to any and all commands, then this would contribute to lower than desired performance for the prototype. Is that correct? If so, then your final statement makes more sense. By improving the reliability of command capture you should be able to improve the reliability of the system in terms of the correct detection of malicious/intruder events. Do you have anything that suggests what type of improvement might be possible?

The process has indicated that a prototype like this can capture all commands however it cannot execute them all properly due to the method of capture. This problem affects about 20% of commands. This limitation suggests further work on the prototype can produce far more reliable results.

**Conclusions**

The results of the test indicate the model isn't sufficiently accurate in detecting malicious and mistake activity even when given special weighting. This is likely due to the lack of data in those categories.

Future research in the area needs more open data. One solution is to ask organizations to publish the cleaned logs after an attack.

This study also indicated that further improvements are needed to the program to accurately capture information from a live terminal. While the program created a fake prompt, it was severely limited in what it could capture and that skewed the data from the simulation. Further work on this prototype could decrease latency, allow all commands to be executed properly, and prevent the user from escaping command capture.

There are a few factors that come into account with the latency of the terminal capture system. One is that most of the code in the `evaluate()` function was commented out and the `get\_prediction()` and `get\_input()` functions were not called at all.

While the high-latency is understandable, this is within range of unacceptable latency and would be highly noticeable to all users. In the future, using a white-list of commands that don’t need evaluation would reduce time. Another option would be to allow command execution before getting results from AML, eliminating the issue altogether.

Future research is needed in order to determine how to best approach the problem of practical proactive

intruder detection. As other researchers have indicated, larger datasets comprising of more points such as time-stamps and keyboard information would help better track user behavior in an attempt to establish recognizable patterns by neural networks. This and other research indicate that more data is needed.

**Funding**

The work reported in this study was initiated and self-funded by the author.

**Acknowledgements**

Faculty Advisor - James A. Subach, Ph. D

Academic Credit Advisor - Nancy L. Jensen

Code Help - Ian Harvey on Stack Overflow

Data Sources - Purdue University

Poster Template - Felix Breuer

The NAU Cyber Security Team

The Embry-Riddle [Ethical] Hackers Club

Amazon Web Services

Font Awesome

GitHub

**Source Code & Materials are available at:** <http://github.com/Skraelingjar/cyberml>

**References**

1. Brdiczka O, Juan Liu, Price B, et al. Proactive insider threat detection through graph learning and psychological context. spw. 2012:142-149. http://ieeexplore.ieee.org.libproxy.nau.edu/document/6227698. doi: 10.1109/SPW.2012.29.

2. Rashid T, Agrafiotis I, Nurse J. A new take on detecting insider threats. Proceedings of the 8th ACM CCS International Workshop on managing insider security threats. Oct 28, 2016:47-56. http://dl.acm.org.libproxy.nau.edu/citation.cfm?id=2995964. doi: 10.1145/2995959.2995964.

3. Parveen P, Weger ZR, Thuraisingham B, Hamlen K, Khan L. Supervised learning for insider threat detection using stream mining. ICTAI. 2011:1032-1039. http://ieeexplore.ieee.org.libproxy.nau.edu/document/6103467. doi: 10.1109/ICTAI.2011.176.

4. Eldardiry H, Bart E, Juan Liu, Hanley J, Price B, Brdiczka O. Multi-domain information fusion for insider threat detection. spw. 2013:45-51. http://ieeexplore.ieee.org.libproxy.nau.edu/document/6565228. doi: 10.1109/SPW.2013.14.

5. Probst CW, Bishop M, Gollmann D, Hunker JA. Insider threats in cyber security. Vol 49. Boston: Springer Science+Business Media; 2010. http://lib.myilibrary.com.libproxy.nau.edu?ID=790801.

6. Maybury M, Chase P, Cheikes B, et al. Analysis and detection of malicious insiders. . 2005. http://www.dtic.mil/docs/citations/ADA456356.

7. Greengard S. Cybersecurity gets smart. Communications of the ACM. 2016;59(5):29-31. http://dl.acm.org.libproxy.nau.edu/citation.cfm?id=2898969. doi: 10.1145/2898969.

8. Rash W. 'Deep learning' technology sees through security software blind spots. eWeek. 2015:1.

9. Preimesberger C. Eight tactics for circumventing insider security threats. eWeek. 2015:1.

10. Hackers offering money to company insiders in return for confidential data. The Hacker News.

11. Sinno Jialin Pan, Qiang Yang. A survey on transfer learning. TKDE. 2010;22(10):1345-1359. http://ieeexplore.ieee.org.libproxy.nau.edu/document/5288526. doi: 10.1109/TKDE.2009.191.

12. Rick Dana Barlow. Grand theft data. Health Management Technology. 2014;35(2):8. http://search.proquest.com.libproxy.nau.edu/docview/1542396962.

13. M. Schonlau, W. DuMouchel, et. al. Computer Intrusion: Detecting masquerades. Statistical Science 2001;16(1):58-74.

14. Maxion RA, Townsend TN. Masquerade detection augmented with error analysis. TR. 2004;53(1):124-147. http://ieeexplore.ieee.org/document/1282170. doi: 10.1109/TR.2004.824828.