Demo Abstract: Praxi: Cloud Software Discovery That Learns From Practice

Anthony Byrne
abyrne19@bu.edu
Boston University
Boston, MA

Sadie L. Allen
sadiela@bu.edu
Boston University
Boston, MA

Shripad Nadgowda
nadgowda@us.ibm.com
IBM Research
Yorktown Heights, NY

Ayse K. Coskun
acoskun@bu.edu
Boston University
Boston, MA

Abstract

With today's rapidly-evolving cloud software landscape, users of cloud systems must constantly monitor software running on their *containers* and *virtual machines (VMs)* to ensure compliance, security, and efficiency. Traditional solutions to this problem rely on manually-created rules that identify software installations and modifications, but these require expert authors and are often unmaintainable. More recent automated techniques leverage knowledge of packaging practices to aid in discovery without requiring any pre-training, but these practice-based methods cannot provide precise-enough information to perform discovery by themselves. Other approaches use machine learning models to facilitate discovery of software present in a training corpus, but prior approaches have high runtime and storage requirements. This demonstration features *Praxi*, a new software discovery method that builds upon the strengths of prior approaches by combining the accuracy of learning-based methods with the efficiency of practice-based methods. We demonstrate Praxi's training and detection process in real time while allowing laptop-equipped participants to follow along using a provided remote virtual machine.

ACM Reference Format:
Anthony Byrne, Sadie L. Allen, Shripad Nadgowda, and Ayse K. Coskun. 2019. Demo Abstract: Praxi: Cloud Software Discovery That Learns From Practice. In *20th International Middleware Conference Demos and Posters (Middleware Demos and Posters '19), December 9–13, 2019, Davis, CA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3366627.3368111

1 Introduction

Modern cloud application development processes emphasize developer speed and agility as key determinants of business success. The shift from vendor-supplied proprietary software to open-source software has made the use of ready-to-use software components a common method of speeding up development. When put into practice, these forces often lead to small teams of developers making independent decisions on which components to use, causing software applications to end up with far more component diversity compared to that of software observed just a few years ago.

With rapid development and deployment cycles often moving too fast for proper documentation, it is all too easy to lose track of what software is installed on a system, leading to inadvertent hosting of non-compliant or buggy applications. Awareness of these issues is a must for proper protection of high-value cloud deployments, and with constant iteration, one-time scans are not enough – continuous awareness is necessary for security assurance in the cloud. Therefore, being able to continuously identify what software exists in a given system, i.e., *"software discovery,"* is becoming a core requirement for ensuring security and compliance.

Initial efforts for software discovery began with rule-based systems [2], where administrators often wrote rules detecting software that had been cataloged in knowledgebases like the National Vulnerability Database. A triggered rule would indicate a vulnerable system, which would then be quarantined and manually patched. This fragile, labor-intensive nature of rule-based methods became the motivation for more novel software detection systems, such as practice-based systems that exploited common patterns of software organization [4] and learning-based systems that could be automatically trained to identify software installations of concern [5]. These new methods had limitations of their own, of course, including a lack of machine-readability and slow machine learning model training times.

In this demonstration, we present a novel hybrid method of automated software discovery, *Praxi*, that combines aspects from both learning- and practice-based approaches. Praxi uses a recent practice-based method to generate informative tags without requiring any pre-training and then employs fast incremental learning methods to quickly update discovery models as new software packages become available. Praxi significantly reduces training time and the overhead associated with incremental training compared to existing learning-based approaches, while still providing highly accurate software discovery without requiring human involvement.

Middleware Demos and Posters '19, December 9–13, 2019, Davis, CA, USA
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7042-4/19/12.
https://doi.org/10.1145/3366627.3368111

2 Technical Approach

Praxi's approach to software discovery can be separated into three main phases: system change recording, feature reduction, and discovery by example.

2.1 System Change Recording

In much the same way as other modern software discovery approaches, Praxi begins by taking in as an input a set of filesystem changes observed during a software installation, known as a *changeset*. In our implementation, these changes are collected using the Linux kernel's *inotify* feature, which notifies a change recording daemon running in the background on the target system of the creation, modification, or deletion of any file or directory that has an inotify "watch" placed on it. After a certain amount of time or upon some user-configurable event, the recording daemon "closes" the changeset by sorting its changeset records by time of occurrence, removing any duplicate entries, and either saving the changeset to a file on-disk or uploading the changeset to a remote server for processing. The daemon then "opens" a new empty changeset for writing in preparation for the next software installation.

2.2 Feature Reduction

After a changeset has been closed, it is analyzed using a practice-based method called Columbus [4]. Columbus uses a frequency trie to discover a set of tags made up of the most frequent longest-common-prefix string tokens found in a changeset, relying on the organizational conventions in place among today's software developers and package maintainers. Because Columbus places only the tags that occur more than once in the resulting tagset, noise arising from irrelevant stimuli (such as log file rotations, caching, etc.) during the recording period is filtered out.

Beyond noise reduction, the practice-based analysis step also offers other practical benefits. By condensing changesets (which vary in size from a few kilobytes to tens of megabytes, depending on number of changes captured) down to tagsets (which are typically less than a kilobyte), Praxi requires much less storage space than other methods. Perhaps most importantly, the simple data structure of tagsets (basic space-separated-value strings) also makes them easy to use with text-based machine learning tools such as Vowpal Wabbit (VW) [3].

2.3 Discovery By Example

After a tagset has been produced, it is used as a feature to train a machine learning model using sparse gradient descent on a hinge loss function. The tagset is treated as if it was a "bag of words," much in the same way that human-readable sentences are processed by natural language classification models. Vowpal Wabbit, the tool we use to train our models, vastly outperforms the other machine learning engines we tested (such as scikit-learn) and unlike many traditional learning engines, VW supports incremental or "online" learning, where trained machine learning models can be updated with new data without requiring a full retraining.

Finally, once a trained model has been created, we present it with unlabeled tagsets (presumably created from live installations of unknown software) for classification.

3 Related Work

As mentioned, there are three main approaches to the problem of software discovery. Rule-based approaches are the usual choices for discovering and identifying system changes in the cloud [2]. Rule-based approaches, however, are often not performant at scale and require too much human-intervention to be practically maintained in today's large cloud deployment. Practice-based approaches like Columbus [4] have the advantage of not requiring any training or corpus, but since these approaches operate on natural language principles, their output is not consistent-enough to be machine-readable. Similarly to Praxi, learning-based approaches like DeltaSherlock leverage "discovery by example" to perform high-accuracy software discovery with the aid of machine learning [1, 5]. Unlike Praxi, however, pre-existing learning-based methods require full regeneration and retraining of models (a slow and storage-intensive process) whenever new or updated software needs to be added to the corpus. In addition to its incremental training abilities, Praxi significantly outperforms DeltaSherlock in terms of runtime, and requires significantly less storage.

Acknowledgments

This work has been partially funded by an IBM Research award.

References

[1] Hao Chen, Sastry S. Duri, Vasanth Bala, Nilton T. Bila, Canturk Isci, and Ayse K. Coskun. 2014. Detecting and identifying system changes in the cloud via discovery by example. In *2014 IEEE International Conference on Big Data (Big Data)*. 90–99. https://doi.org/10.1109/BigData.2014.7004217

[2] Minkyong Kim, Han Chen, Jonathan Munson, and Hui Lei. 2012. Management-Based License Discovery for the Cloud. In *Service-Oriented Computing*, Chengfei Liu, Heiko Ludwig, Farouk Toumani, and Qi Yu (Eds.). 499–506. https://doi.org/10.1007/978-3-642-34321-6_33

[3] John Langford, Lihong Li, and Alex Strehl. 2007. Vowpal Wabbit: Online Learning project. http://hunch.net/~vw/

[4] Shripad Nadgowda, Sastry Duri, Canturk Isci, and Vijay Mann. 2017. Columbus: Filesystem Tree Introspection for Software Discovery. In *2017 IEEE International Conference on Cloud Engineering (IC2E)*. 67–74. https://doi.org/10.1109/IC2E.2017.14

[5] Ata Turk, Hao Chen, Anthony Byrne, John Knollmeyer, Sastry S. Duri, Canturk Isci, and Ayse Kivilcim Coskun. 2016. DeltaSherlock: Identifying changes in the cloud. In *2016 IEEE International Conference on Big Data (Big Data)*. 763–772. https://doi.org/10.1109/BigData.2016.7840669