# Version Detection for Software Discovery in the Cloud

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

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

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

## Abstract

With the growth in server traffic and component diversity in cloud systems, administrators face the increasingly onerous task of monitoring system activity. Failure to keep track of the contents of virtual servers can limit overall efficiency and create security risks for users. Prior work in software discovery attempted to address this problem by identifying applications based on file system activity. While some of these methods have claimed to be extensible to detection of specific *versions* of an application, version detection has yet to be demonstrated. In this paper, we propose version detection algorithms that operate on top of Praxi, an existing open-source software discovery tool. These algorithms introduce a rule-based component to differentiate between versions, whose file system footprints can appear very similar. We find that our best method achieves up to 99.9% accuracy in version detection experiments compared to Praxi's original 94% accuracy, albeit at the cost of increased runtime. This work confirms the feasibility of version detection in software discovery and provides a starting point for implementing this feature in software discovery tools.

## 1 Introduction

Current trends in cloud computing, such as continuous integration and delivery (*CI/CD*) and a move towards open-source software, have expedited software release cycles and led to much more diversity in the software components found in the typical cloud system. With this diversification comes the problem of keeping track of the contents of the multitude of VMs and containers in the cloud — a task essential to ensuring the compliance, security, and efficiency of cloud applications. As a solution to this problem, software discovery methods attempt to continuously identify the software present on a given system. Version detection in software discovery — i.e., identifying the specific version of a given application installed — is vital for rapid location and patching of vulnerabilities. If a vulnerability is reported in a certain application, system administrators would ideally know not only if the application is installed, but also if the vulnerable *version* is installed in a particular VM or container, in order to avoid wasting time patching unaffected or irrelevant systems. Version detection allows for problematic software to be pinpointed more quickly, limiting the window of time during which a system is vulnerable.

Some prior efforts in software discovery have claimed to be extendable to version detection [1, 3], but none have actually demonstrated such a capability. We propose two version detection algorithms built upon Praxi, an open-source software discovery tool that uses machine learning to identify applications based on file system changes [1]. Our new methods use Praxi to identify applications and then use *rules* based on file paths to distinguish between versions of a given application. In our experiments, we find that our best algorithm is able to detect application versions with up to 99.9% accuracy, but takes 5.2x longer than the original Praxi method.

## 2 Background: Praxi

Praxi is a machine-learning based software discovery method that serves as the basis of comparison for our study and a building block for our novel algorithms. Praxi trains with *changesets*, which are recordings of all filesystem changes during a given period of time. To create examples of specific applications, changesets are recorded during application installations. Praxi uses Columbus [3], a tool for tokenizing a list of file paths and indexing them into a *frequency trie*[1], to reduce changesets down to a set of *tags*. The tags represent the most-frequent longest-common-prefixes among the filesystem tree tokens in each changeset. The top $k$ tags are selected ($k$ chosen heuristically) to form a *tagset* containing the tags occurring most frequently in all of the files created during a specific application's installation.

Once the training data has been reduced to tagsets, Praxi generates a trained classifier using the machine learning engine Vowpal Wabbit (VW) [2]. Tagsets are treated as "bags of words" and VW generates a single multi-class classifier for all single-label changesets in the corpus.

## 3 Proposed Version Detection Methods

### 3.1 A Robust Version Discovery Method

Our novel algorithms use Praxi to determine a changeset's application, and then employ rules to differentiate between specific versions of that application. Rules are file paths common to a specific software installation and are triggered when they are detected in a changeset. We keep two central properties in mind when devising rule creation methods: uniqueness and ubiquity. Each application version's rule should be *unique* to that individual version such that the rule does not apply to any other versions of the given application. Conversely, a rule for a version should be *ubiquitous* such that it always applies to every example of a given application version.

Both of our novel algorithms begin with a group of training changesets labeled with application names and versions. First, we train Praxi to recognize generally any version of a given application. Second, we create application-specific version rules. In the rule generation step, changesets are separated into groups by application name, and rules only need to differentiate between the individual versions of a single application. We implement two rule generation methods described in the following sections.

[1] A frequency trie is an extension of the information retrieval data structure, the trie. The structure is described in detail by Nadgowda et al. [3].

### 3.1.1 Basic Rules

Our basic rule creation algorithm assumes that a version of an application has at least one file that is unique to it and appears in every changeset example. We achieve this by taking the union of all changesets of a specific version, i.e., the set of all file paths that appear in every example of a given application version. The next step is finding any overlapping file paths between these unions and removing them. Any file path appearing in more than one of the unions cannot function as a rule, as it would cause the rule to violate the ubiquity principle mentioned above. Finally, we find the intersection of all changesets for a version, and all of the file paths in that intersection become the rules for that version.

### 3.1.2 Combination Rules

In the combination (combo) rules class, we consider additional file metadata when generating our mappings between file paths and rules, creating more sophisticated rules compared to those generated by our basic algorithm. Specifically, a file path can be used to differentiate between two versions and the absence of a file can be used as a rule. As in the basic rule generation, rules need only be created to distinguish between versions of a given application, as Praxi is used to classify changesets by application.

Three different types of rules are created for each version (assume rules are currently being generated for version <label>):

**<token>, 'unique to', <index>:** Indicates that the file path <token> appears in <index> different labels. So, if <index> is 1, this file path is unique to <label>.

**<token>, 'inside vs', <other_label>:** Indicates that <token> is found in <label> but not in <other_label>. Thus, the presence of a specific file can serve to differentiate between two version labels.

**<token>, 'outside vs', <other_label>:** Indicates that <token> is found in <other_label> but not in <label>. This can be used in the same way as the second type of rule.

The output of the rule generation script is a mapping from labels to sets of these rules. These mappings are used to make version classification decisions in the evaluation phase.

We also attempt to mitigate the issue of versions of the same application having similar file paths by appending file sizes to the ends of file paths listed in the changesets. We evaluate this mitigation on our basic and combo rule generation algorithms and compare the classification accuracy to that of our original methods.

## 3.2 Praxi for Version Detection

In order to provide a baseline for our experiments, we extend Praxi to support version detection by changing its input data and formatting. In Byrne et al.'s original experiments with Praxi, changesets for just one application version are recorded and then labeled with the name of that application [1]. In our experiments, we record changesets for several different versions of each application and label them with both the application name and version number, concatenated into a single string. Praxi's original algorithm can then be applied normally.

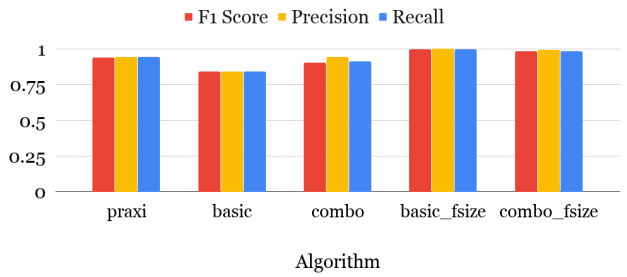


Figure 1: The accuracy results for version detection experiments

# 4 Results

## 4.1 Comparison of Accuracy and Overhead

In our experiments, in which we attempted to classify the application and version of 14,000 changesets of Ubuntu's top 100 most popular non-library packages, the basic rule-based algorithm with file size is able to detect specific versions of our applications with the best accuracy, achieving an F1 score of 0.999. Praxi itself achieved an F1 score of just 0.941 by comparison. See Figure 1 for comparison with other algorithms.

Although Praxi does not yield the best accuracy of the algorithms that we tested, it is the best in terms of running time, as the other methods that we tested ran 5.2 times slower. More experiments regarding how the running time overhead scales are necessary to understand the extent to which our rule-based methods are worse.

## 4.2 Case Study: OpenSSL

To show the value of version detection in software discovery, we demonstrate its functionality in a situation with a real, known vulnerability. Specifically, we tested whether two of our methods, Praxi and the basic rule-based algorithm with file size, could identify a version of OpenSSL with the Heartbleed bug. For these experiments, we recorded 30 changesets of two versions of OpenSSL–one with and one without the bug–and chose changesets from five other applications from the main experiments at random and ran the algorithms on this corpus to show they could accurately distinguish the OpenSSL application from others.

In this case study, Praxi achieves an F1 score of 0.936 while the basic rule based algorithm with file size has an F1 score of 0.999. This demonstrates that the algorithms can successfully identify a vulnerable application version.

# References

[1] A. Byrne, E. Ates, A. Turk, V. Pchelin, S. Duri, S. Nadgowda, C. Isci, and A. Coskun. Praxi: Cloud Software Discovery That Learns From Practice. *IEEE Transactions on Cloud Computing*, PP:1–1, 2 2020. doi:10.1109/TCC.2020.2975439.

[2] J. Langford, L. Li, and A. Strehl. Vowpal wabbit: Online learning project, 2007. URL: http://hunch.net/~vw/.

[3] S. Nadgowda, S. Duri, C. Isci, and V. Mann. Columbus: Filesystem tree introspection for software discovery. In *2017 IEEE International Conference on Cloud Engineering (IC2E)*, pages 67–74, Apr. 2017. doi:10.1109/IC2E.2017.14.