Inferring User Actions from Provenance Logs

Xin Li, Chaitanya Joshi
Department of Statistics
University of Waikato
Hamilton, New Zealand
xl282@students.waikato.ac.nz, cjoshi@waikato.ac.nz

Alan Y. S. Tan, Ryan K. L. Ko
Cyber Security Lab, Department of Computer Science
University of Waikato
Hamilton, New Zealand
yst1@students.waikato.ac.nz, ryan@waikato.ac.nz

***Abstract*—Progger, a kernel-spaced cloud data provenance logger which provides fine-grained data activity records, was recently developed to empower cloud stakeholders to trace data life cycles within and across clouds. Progger logs have the potential to allow analysts to infer user actions and create a data-centric behaviour history in a cloud computing environment. However, the Progger logs are complex and noisy and therefore, currently this potential can not be met. This paper proposes a statistical approach to efficiently infer the user actions from the Progger logs.**

Inferring logs which capture activities at kernel-level granularity is not a straightforward endeavour. This paper overcomes this challenge through an approach which shows a high level of accuracy. The key aspects of this approach are identifying the data preprocessing steps and attribute selection. We then use four standard classification models and identify the model which provides the most accurate inference on user actions. To our best knowledge, this is the first work of its kind. We also discuss a number of possible extensions to this work. Possible future applications include the ability to predict an anomalous security activity before it occurs.

***Keywords*-Log Mining; User Actions; Progger; Provenance Mining; Data-centric Logger; Data Provenance; Data Security; Cloud Computing.**

I. Introduction

Data is arguably the most important asset in cloud computing. For the purposes of data security, leakage detection and provenance, a data-centric thinking has replaced the classical preventive, system-centric thinking [1]. In the last few years, there have been several data-centric logging tools. To our best knowledge, Progger [2] is the most advanced data-centric logging tool so far. Compared to the data-centric logging tools proposed earlier, namely Flogger [3], [4] and S2Logger [5], Progger has certain advantages. These advantages include (1) the ability to provide log tamper-evidence and prevention of fake/manual entries, (2) accurate and granular timestamp synchronisation across several machines, (3) log space requirements and growth, and (4) the efficient logging of root usage of the system [2].

While Progger logs provide fine-grained data activity audit, the logs are not deterministic. Progger logs generated by certain user action on different occasions are not identical, therefore an appropriate approach is required for inferring the user actions. On a large scale (e.g. above thousands of instances in clouds), it is generally difficult to infer meaningful actions quickly from the deluge of log data. Hence, it is still a challenge to infer the user actions from aggregated provenance logs. We aim to address these issues in this paper, and make the inference of user data actions near real-time, meeting present day security situation awareness needs.

Some attempts have previously been made to infer user actions using provenance logs [6], [7], [8], [9]. However, each of these was designed for a particular application (e.g. data curation, lab recording, scientific workflows) and does not necessarily capture the user actions at a fine-grained level. Also none of these systems was developed for applications to cloud computing, data security or cloud forensics.

In this paper, we set out to develop a statistical approach to infer user actions especially using Progger logs. Section II displays the structure of Progger log file, addresses the critical issues of Progger logs, defines user scenarios and explains the data simulation process. Section III describes the process we follow to find the algorithm for inferring user actions which yields the highest accuracy. Section IV reports the performances and summarizes the main findings as well. Finally, we discuss the limitation and future work in Section V.

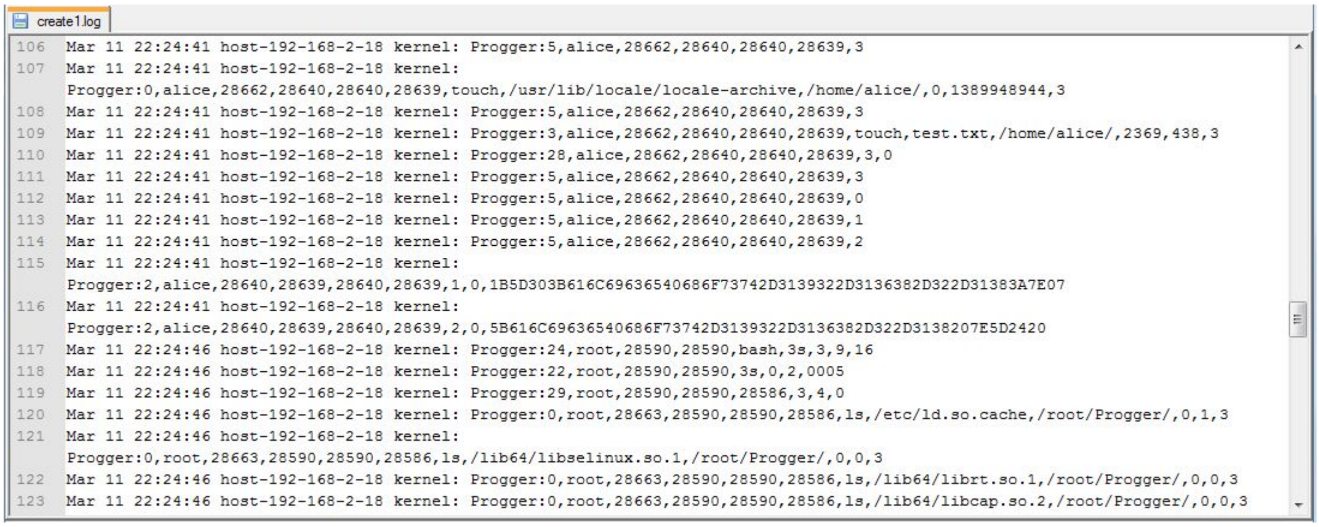
II. Progger Logs & User Scenarios

A. Introducing the Progger Log File

With reference to a real Progger log file shown in Figure 1, it can be seen that Progger logs start with timestamps, ID information and key words “kernel” and “Progger”. The numbers next to these keywords are the system call type numbers. Each system call type number corresponds to a system call type (Table I) [2]. Following the system call type number, there is some more information related to the system call such as user name, process ID and parent process ID.

B. Critical Issues Identified in Progger Logs

While manual inspection is possible, it is not feasible or scalable to infer user data actions from raw Progger logs. Automating the process is also a challenge because the logs

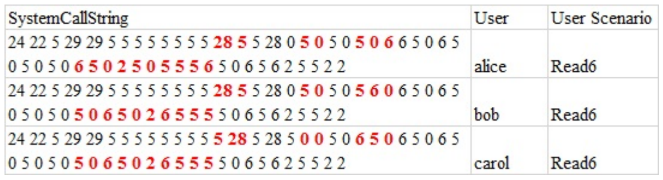

create1.log

```
Mar 11 22:24:41 host-192-168-2-18 kernel: Progger:5,alice,28662,28640,28640,28639,3
Mar 11 22:24:41 host-192-168-2-18 kernel:
Progger:0,alice,28662,28640,28640,28639,touch,/usr/lib/locale/locale-archive,/home/alice/,0,1389948944,3
Mar 11 22:24:41 host-192-168-2-18 kernel: Progger:5,alice,28662,28640,28640,28639,3
Mar 11 22:24:41 host-192-168-2-18 kernel: Progger:3,alice,28662,28640,28640,28639,touch,test.txt,/home/alice/,2369,438,3
Mar 11 22:24:41 host-192-168-2-18 kernel: Progger:28,alice,28662,28640,28640,28639,3,0
Mar 11 22:24:41 host-192-168-2-18 kernel: Progger:5,alice,28662,28640,28640,28639,3
Mar 11 22:24:41 host-192-168-2-18 kernel: Progger:5,alice,28662,28640,28640,28639,0
Mar 11 22:24:41 host-192-168-2-18 kernel: Progger:5,alice,28662,28640,28640,28639,1
Mar 11 22:24:41 host-192-168-2-18 kernel: Progger:5,alice,28662,28640,28640,28639,2
Mar 11 22:24:41 host-192-168-2-18 kernel:
Progger:2,alice,28640,28639,28640,28639,1,0,1B5D303B616C69636540686F73742D3139322D3136382D322D31383A7E07
Mar 11 22:24:41 host-192-168-2-18 kernel:
Progger:2,alice,28640,28639,28640,28639,2,0,5B616C69636540686F73742D3139322D3136382D322D3138207E5D2420
Mar 11 22:24:46 host-192-168-2-18 kernel: Progger:24,root,28590,28590,bash,3s,3,9,16
Mar 11 22:24:46 host-192-168-2-18 kernel: Progger:22,root,28590,28590,3s,0,2,0005
Mar 11 22:24:46 host-192-168-2-18 kernel: Progger:29,root,28590,28590,28586,3,4,0
Mar 11 22:24:46 host-192-168-2-18 kernel: Progger:0,root,28663,28590,28590,28586,ls,/etc/ld.so.cache,/root/Progger/,0,1,3
Mar 11 22:24:46 host-192-168-2-18 kernel:
Progger:0,root,28663,28590,28590,28586,ls,/lib64/libselinux.so.1,/root/Progger/,0,0,3
Mar 11 22:24:46 host-192-168-2-18 kernel: Progger:0,root,28663,28590,28590,28586,ls,/lib64/librt.so.1,/root/Progger/,0,0,3
Mar 11 22:24:46 host-192-168-2-18 kernel: Progger:0,root,28663,28590,28590,28586,ls,/lib64/libcap.so.2,/root/Progger/,0,0,3
```

Figure 1. Part of the Progger log showing user scenario Create1

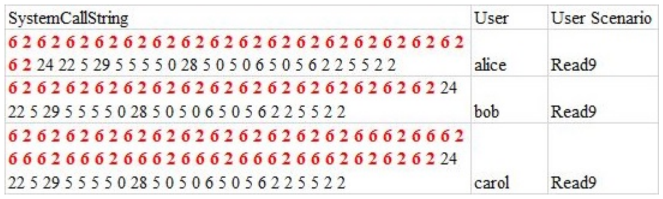
Table I
SYSTEM CALL LIST

System Call	Number	System Call	Number
OPEN	0	FCHOWN	16
UNLINK	1	LCHOWN	17
WRITE	2	FCHOWNAT	18
CREAT	3	CHMOD	19
MOVE	4	FCHMOD	20
CLOSE	5	FCHMODAT	21
READ	6	S_SENDMSG	22
S_CONNECT	7	S_ACCEPT	23
S_SENDTO	8	S_SOCKET	24
UNLINKAT	9	SENDFILE	25
MKDIR	10	S_RECVFROM	26
RMDIR	11	S_RECVMSG	27
SYMLINK	12	DUP2	28
LINK	13	PIPE	29
LINKAT	14	PIPE2	30
CHOWN	15	DUP	31



SystemCallString	User	User Scenario
24 22 5 29 29 5 5 5 5 5 5 5 5 **28 5** 5 28 0 **5 0** 5 0 **5 0 6** 6 5 0 6 5 0 5 0 5 0 **6 5 0 2 5 0 5 5 5 6** 5 0 6 5 6 2 5 5 2 2	alice	Read6
24 22 5 29 29 5 5 5 5 5 5 5 5 **28 5** 5 28 0 **5 0** 5 0 **5 6 0** 6 5 0 6 5 0 5 0 5 0 **5 0 6 5 0 2 6 5 5 5** 5 0 6 5 6 2 5 5 2 2	bob	Read6
24 22 5 29 29 5 5 5 5 5 5 5 5 **5 28** 5 28 5 **0 0** 5 0 **6 5 0** 6 5 0 6 5 0 5 0 5 0 **5 0 6 5 0 2 6 5 5 5** 5 0 6 5 6 2 5 5 2 2	carol	Read6

Figure 2. System Call Sequences for the Same User Scenario with Varying Orders



SystemCallString	User	User Scenario
6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 24 22 5 29 5 5 5 5 0 28 5 0 5 0 6 5 0 5 6 2 2 5 5 2 2	alice	Read9
6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 24 22 5 29 5 5 5 5 0 28 5 0 5 0 6 5 0 5 6 2 2 5 5 2 2	bob	Read9
6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 2 6 6 6 2 6 6 6 2 6 6 6 2 6 6 6 2 6 6 6 2 6 6 6 2 6 6 6 2 6 6 6 2 6 2 6 2 6 2 24 22 5 29 5 5 5 5 0 28 5 0 5 0 6 5 0 5 6 2 2 5 5 2 2	carol	Read9

Figure 3. System Call Sequences for the Same User Scenario with Varying Lengths

generated by certain user action on different occasions have: (1) varying orders, (2) varying lengths, (3) redundant data. These variations will significantly reduce the accuracy of the inferring process and make the inferring process very difficult.

1) Varying Orders: The orders of the logs generated on different occasions are inconsistent. This is because some processes are concurrent and the order of logs are according to the order in which they enter the kernel buffer. For instance, Figure 2 shows the system call sequences for the same user scenario *Read6* but for three different users. In fact, they all have the sane system calls but with different order. The bold numbers in Figure 2 are where they are different. Current defined user scenarios and their basic descriptions can be found in Section II-C along with their examples.

2) Varying Lengths: One simple command line generates logs of different lengths. The reason is that the Progger logs all actions of the user including the typing errors and text input actions. We are not concerned with this part of logs. For instance, in Figure 3, these three system call sequences represent the same user scenario *Read9* but the lengths are different.

3) Redundant Data: The background system processes are usually run as root. However, it is not the real user and does not contribute towards user actions inference or user behaviour prediction. Hence the log entries with user "root" need to be removed before inferring. Figure 1 gives a direct visualization of the Progger logs. As we can see, there are some logs for which the user is "root".

C. User Scenarios

Before we get started, we have to answer the question: what are user scenarios? We have defined user scenario as a narrative of foreseeable interaction of the user role and the Linux system in the cloud, but at this stage, we only focus on file activities. Therefore, we consider thirty user scenarios which have been classified into four basic categories. These are *create*, *update*, *read* and *delete*. Table II lists the categories and their basic description.

Table II
SCENARIO GROUP LIST

Group Name	Description
Create	Create a non-existing file
Update	Change the content of an existing file
Read	Display without changing anything
Delete	Remove the file

Each user scenario corresponds to a specific action that user takes. Table III shows the description of these scenarios. For simplicity, we use the acronym to represent the scenarios in this study.

It should be noted that some of the system calls are referred to using the same English verbs that are used to describe the user scenarios. For example, the system call *read* means a request to the operating system (kernel) to read from a file descriptor. Whereas, the user scenario category *read* means that the user reads the content of a file. One user scenario generates a series of system calls. For instance, the logs shown in Figure 1 all corresponds to one user scenario.

D. Data Simulation

In this study, we use a training data set which is a log file generated by implementing each of the user scenario several times. This includes the logs generated for different users as well as the logs generated for the same user at different instances. Thus, the training data set is expected to capture the variability in the log data. We also use two testing data sets: (1) mono-scenario testing data set, (2) interleaving scenario testing data set. The mono-scenario refers to a situation where several scenarios are implemented in a sequential fashion, i.e. one after the other. On the other hand, the interleaving scenario refers to a more realistic situation where several scenarios are run concurrently (i.e., log entries from different scenarios may interleave each other). The interleaving scenario simulates multi-tasking on an operating system.

Figure 4 presents an example that generates an interleaving scenario data set. First, the user carol creates a new file *test.txt* using the *touch* command. At time 0002 carol opens the file *test.txt* using the *nano* editor, writes something to the file and closes the *nano* editor at time 0005. However, at time 0004, carol writes something to a file *result.txt* using the *echo* command. As a result (*nano* editor at time 0002 & 0005, *touch* at time 0003 and *echo* at time 0004) the logs generated interleave.

```
Date: 12th Dec
User: carol
Time Sequence of operations carried out
0001 create a flie : touch test.txt
0002 write “This is a test for Progger” with nano editor : nano test.txt
0003 create a file : touch result.txt
0004 write “This is a test for Progger” : echo “This is a test for Progger”
> result.txt
0005 quit nano editor
0006 read content of a file and append it to another existing file : cat
test.txt >> result.txt
0007 delete test.txt : rm test.txt
```

Figure 4. README File of Interleaving Scenario Data Set

Figure 5 presents an example that generates a mono-scenario data set. First, the user Alice creates a new file *test.txt* using the *vim* editor and exits the *vim* editor by saving the file. Then, Alice uses *cat* command to display the content of *test.txt*. After this, Alice copies the content of *test.txt* to a non-existing file *test2.txt*. Finally, Alice deletes the previous file *test.txt*.

```
Date: 30th Oct
User: Alice
Time Sequence of operations carried out
0028 vim test.txt
0029 cat test.txt
0030 cp test.txt test2.txt
0031 rm test.txt
```

Figure 5. README File of Mono-scenario Data Set

III. OUR APPROACH

An accurate and efficient statistical approach is necessary for inferring user actions from the Progger logs. The approach we develop involves data preprocessing, followed by using standard classification algorithms and model evaluation (Figure 6). Data preprocessing is an essential step in the inferring process because the quality of the input will strongly affect the final classification accuracy [10]. After data preprocessing, we use four classification methods for all possible combinations of the attributes considered and select the best algorithm based on the classification accuracy. We discuss this approach in detail below.

A. Data Preprocessing

The raw data presented in the log file is messy and complex (as illustrated by the log shown in Figure 1). Hence the contents of the log file need to be cleaned in this preprocessing step. The redundant data are removed and this eliminates the noise in the Progger log data. The logs are then converted into a format appropriate for the classification algorithm to be used. The data preprocessing step involves the following sub-steps:

Table III
USER SCENARIO LIST

No.	Acronym	Description	Command Example
1	Create1	Create a new empty file in user home directory	touch test.txt
2	Create2	Create a new file in user home directory with content	echo “Progger” > test.txt
3	Create3	Create a new file in user home directory with content using vim	vim test.txt
4	Create4	Create an empty file using vim editor	vim test.txt
5	Create5	Create an empty file using a C program, createfile	./createfile
6	Create6	Create an empty file using nano editor	nano test.txt
7	Create7	Launch editor inside program to create file	./cmdcreatefile
8	Create8	Redirect and append data into new file	echo “Progger” >> test.txt
9	Create9	Write to new file with nano editor	nano test.txt
10	Create10	Copying of files	cp test.txt result.txt
11	Create11	Copy file using redirect	cat test.txt > result.txt
12	Create12	Cut and paste file using redirect	mv test.txt > result.txt
13	Update1	Delete partial data from a file using nano editor	nano test.txt
14	Update2	Delete partial data from file using vim editor	vim test.txt
15	Update3	Delete content of file using pipe	> test.txt
16	Update4	Redirect (or append) data into an existing file	echo “Progger” > test.txt echo “Progger” >> test.txt
17	Update5	Redirect (or append) content of an existing file into an existing file	cat test.txt > result.txt cat test.txt >> result.txt
18	Read1	Read file by piping out the file from command line	cat test.txt
19	Read2	Read file using nano editor	nano test.txt
20	Read3	Read file using vim editor	vim test.txt
21	Read4	Open and read file using user program (C user program)	./readfile
22	Read5	Read file using tail command (built-in linux command)	tail test.txt
23	Read6	Read file with cat and pipe it to grep for filtering	cat test.txt | grep Progger
24	Read7	Redirect content of file to command/program	grep Progger < test.txt
25	Read8	Read content of file and display in default format using fmt command	fmt test.txt
26	Read9	Read and redirect content of file to fmt	fmt < test.txt
27	Read10	Read content from file and pipe to fmt	cat test.txt | fmt -w 5
28	Read11	Read data into vim editor’s buffer	vim test.txt and “yank” line
29	Read12	List file names in user home directory	ls
30	Delete1	Create a new empty file in user home directory	rm test.txt

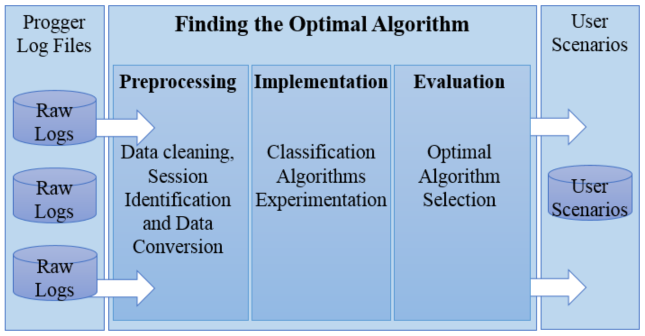

Progger Log Files
Raw Logs
Raw Logs
Raw Logs
Finding the Optimal Algorithm
Preprocessing
Data cleaning, Session Identification and Data Conversion
Implementation
Classification Algorithms Experimentation
Evaluation
Optimal Algorithm Selection
User Scenarios
User Scenarios

Figure 6. Steps of Finding the Optimal Algorithm

1) Data Cleaning: The data cleaning step solves the major data quality problems that were discussed in Section II-B. First, the root usage logs can be removed using the user name. Second, the logs generated because of the user editing the content are repetitive “6 -2(READ – WRITE)” in nature. This pattern can be detected and the noise generated can be minimised by merging the repetitive pattern.

2) Session Identification: The system call sequence is an on-going number sequence in practice. It embodies more than one user scenario. We have to find a way to split into subsequences and make sure each subsequence roughly matches a user scenario. Observing the training data set, it can be seen that the system call sequence of every scenario starts with a fixed pattern: “24-22-5-29 (S_SOCKET – S_SENDMSG – CLOSE – PIPE)”. In addition, this special pattern does not appear anywhere else. This pattern is therefore used to identify the start of a new user scenario. This way, the system call sequence is split into several subsequences, each corresponding to a specific user action.

3) Data Conversion: This is a conversion of the data in the log file into the format needed by the classification algorithms. The actual system call types are numerical values. In this step, the numerical values have been replaced with a descriptive name for readability and convenience.

We will be using the classification algorithms in WEKA [11]. As such the pooled system call sequences set is converted to an ARFF file to match the input format requirement of WEKA. An ARFF (Attribute-Relation File Format) file is a text file that describes a list of instances sharing a set of attributes. It was developed by the University of Waikato for use with the WEKA software [11].

B. Attributes Selection

After conversion, the ARFF file includes three variables: *SystemCallString*, *Length*, *UserScenario*. The first two are attributes and are used to infer the classes, the last one is used to evaluate the classification accuracy. The "System-CallString" represents the system call sequence. A system call sequence is an ordered system call type list and, as mentioned in Section II-A, the system call type is the focal point of the logs. It is the most important attribute for inferring the user actions. The "Length" is another important attribute. It represents the number of log entries that are related to one user scenario. The logs generated by some of the user scenarios are short (length < 70), whereas the others are much longer (length > 250). Therefore, the length data are simply grouped into two clusters. Thus, the length data is likely to be helpful in distinguishing these two clusters and therefore likely important in inferring the user actions. As explained in Section II B, the logs generated by the same user scenario at different instances have very similar (but not exactly the same) lengths. The similarity of log lengths explain why we consider it as an efficient attribute for classification.

After conversion, the data become a sequence of English words. If we simply input it into a classification algorithm, the information about the order will be lost. In Text Mining, often, a collection of co-occurring terms may describe the contents of a document better than any single term [12] [13]. For example, "cloud computing" might be such a phrase, which is a specific reference to a computing term, but has nothing to do with the common use of the term "cloud" as it might, for example, be used in descriptions of a visible mass of liquid droplets. Likewise, we try to treat the co-occurring system call as one attribute. We pool some system call together with different window lengths. The value of "window length" represents how many system calls have been pooled together. Figure 7 shows the original system call sequence of Create1. Figure 8 shows the reconstructed system call sequence (Window Length $= 3$) of Create1. We implement the classification methods with different window lengths to determine the value of the window length that yields the highest accuracy.

```
S-SOCKET S-SENDMSG CLOSE PIPE CLOSE CLOSE CLOSE CLOSE
OPEN CLOSE OPEN READ CLOSE OPEN READ CLOSE OPEN READ
CLOSE OPEN CLOSE CREATE DUP2 CLOSE CLOSE CLOSE CLOSE
WRITE WRITE READ WRITE
```

Figure 7. System Call Sequence before Pooling

```
S-SOCKETS-SENDMSGCLOSE              S-SENDMSGCLOSEPIPE
CLOSEPIPECLOSE     PIPECLOSECLOSE     CLOSECLOSECLOSE
CLOSECLOSECLOSE    CLOSECLOSEOPEN     CLOSEOPENCLOSE
OPENCLOSEOPEN      CLOSEOPENREAD      OPENREADCLOSE
READCLOSEOPEN      CLOSEOPENREAD      OPENREADCLOSE
READCLOSEOPEN      CLOSEOPENREAD      OPENREADCLOSE
READCLOSEOPEN      CLOSEOPENCLOSE     OPENCLOSECREATE
CLOSECREATEDUP2    CREATEDUP2CLOSE    DUP2CLOSECLOSE
CLOSECLOSECLOSE    CLOSECLOSECLOSE    CLOSECLOSEWRITE
CLOSEWRITEWRITE
```

Figure 8. System Call Sequence after Pooling (Window Length = 3)

C. Classification Algorithms

In this step, we use different classification algorithms and compare their performance to find the best performing algorithm for classifying Progger logs. This application requires a classification algorithm that is accurate as well as highly efficient. Efficiency is very important since in a real life deployment, this approach will be implemented on the huge amount of streaming log data generated by a cloud computing environment. For this reason, we use simple and time tested classification methods. We employ four commonly used classification methods, namely, Naive Bayes Classifier, Multinomial Naive Bayes Classifier, Nearest-neighbour Classifier (IB1) and Decision Tree (J48). Each method is implemented both with/without *length* attribute, and also using different window lengths.

In this paper, these algorithms are implemented in WEKA [11]. WEKA is a data mining software that has been developed at the University of Waikato (New Zealand). The WEKA package names for each of the classification algorithms is shown in Table IV [11].

The attribute *SystemCallString* is a string attribute. However, NaïveBayes, NaïveBayesMultinomial and IB1 can not deal with string attributes directly. Therefore, we need a filter to convert string attributes into numeric vectors first. *StringToWordVector* produces numeric attributes that represent the frequency of words in the value of a string attribute [14]. The new attribute set is determined from the training data set. By default each word becomes an attribute whose initial value is 1 or 0, reflecting that word's presence in the string.

Table IV
CLASSIFIER LIST

Classifier Name	WEKA Package Name
Naïve Bayes Classifier	NaiveBayes
Multinomial Naïve Bayes Classifier	NaiveBayesMultinomial
Nearest-neighbour Classifier	IB1
Decision Tree	J48

D. Evaluation

We compare the output of the candidate statistical approaches to find the best one. The performance measure is the classification accuracy. The approaches are evaluated, first, using sixfold cross-validation on the training data set, and then by measuring the classification accuracy on the mono-scenario testing data set and the interleaving scenario testing data set. We discuss the results of this evaluation in Section IV.

IV. Performance

A. Statistical Approach

Table V shows the sixfold cross-validation accuracy achieved for each algorithm. Each algorithm was implemented for all possible combinations of the two attributes, *Length* and *window length*. It can be seen that the classification accuracy achieved in most cases is in the range of about 80% to 90%. The Nearest-neighbour (IB1), the Naive Bayes and the Decision Tree (J48) seem to perform well but the accuracy achieved using the Multinomial Naive Bayes algorithm is significantly less. It is clear that including the length information does not necessarily improve the accuracy significantly. In general, the length feature seems to be only helpful when the window length is small (less than four) or equal to the length. For Multinomial Naive Bayes, the length lowers the classification accuracy except when the window length equals to one.

In a cross validation, both the testing data and the training data come from the same data set. As a result, the cross validation may often overestimate the accuracy [14]. To get a more accurate estimate of the accuracy that may be achieved in practice, we need to implement the algorithms on the two testing datasets. The cross validation results do however help us identify the optimal combinations of attributes that are likely to provide the highest possible accuracy for each of the algorithms. The four algorithms were then implemented using these optimal combinations of attributes on the two testing data sets, namely, the mono-scenario testing data set and interleaving scenario testing data set. The results are summarized in Table VI.

Nearest-neighbour (IB1) with length attribute, window length $= 6$ and without length attribute, window length $= 5$ seems to outperform the others on the classification accuracy achieved on both the mono-scenario data set as well as the interleaving scenario data set. Note that in some cases, for e.g., IB1 with window length $= 7$, the accuracy on the test data sets is significantly lower than the cross validation accuracy. This is possibly because the test datasets were small in size and did not incorporate a wide range of user scenarios.

B. Summary

We have built a novel approach to infer the user actions using the Progger logs. The first challenge is how to define and remove the "noise" in the system call sequence. As described in Section III-A, data preprocessing is an important step in this study. As a result of this step, the variation due to noise is nearly eliminated.

Another challenge is to identify the key attributes. In theory, having more features should result in more discriminating power. However, in practice, adding irrelevant or distracting attributes to a data set often confuses classification systems [14]. It can be seen that the length attribute does not seem to have any sizable effect on the classification accuracy. Thus, a case could be made to remove it. Eliminating an unnecessary attribute may also likely improve the efficiency of the algorithm.

Finally, this work has identified a classification approach that is both accurate as well as efficient for the limited set of Linux user scenarios considered.

V. Future Work and Conclusion

This paper discusses a statistical approach that is able to infer, with high accuracy, the user actions using the Progger log data. This is the first such work to the best of our knowledge. Although, this study only considered a limited set of user scenarios, it has showed that by cleaning the Progger logs appropriately and then looking up for specific patterns in the system call sequence, satisfying accuracy can be achieved. The nearest-neighbour classifier (IB1) seems to be the best choice for this data. It also showed that the length information may be dispensable and the optimal window length was five for this data. However, this is a pilot study. Further work is needed to develop an algorithm which can produce efficient and accurate inference on user actions in a real life environment. This is described below.

A. Future Work

1) Real Time Analysis for Data Streams: As described in Section II-C, we only focus on the file activities and therefore only consider thirty user scenarios. In fact, user scenarios are non-exhaustive. The real scenarios on the cloud are much more complex and variable. The main contribution of the present work is to show how a statistical approach could be developed to accurately and efficiently identify the user scenarios. The next stage is to incorporate more user scenarios, such as network communication and interprocess communication scenarios and implement this approach to develop a classification algorithm that can work on streaming cloud data. It may happen that a real life data may provide very little training data for certain rare scenarios. How the training should be handled for such scenarios is a challenge. Further, the classification models will need to be implemented on data stream mining platforms such as Massive Online Analysis (MOA) [15]. MOA is designed to deal with the challenging problem of scaling up the implementation of state of the art algorithms to real world dataset sizes.

2) Improving the Accuracy: There are two possible ways to achieve higher accuracy for a given set of user scenarios. The first one is to identify additional features and second by using additional classification algorithms such as support

Table V
ACCURACY OF CLASSIFIERS ON SIXFOLD CROSS-VALIDATION

Classifier	Length	Window Length								
		1	2	3	4	5	6	7	8	Whole
Naive Bayes	Yes	70.83%	88.02%	90.10%	92.19%	92.71%	92.71%	93.23%	93.23%	86.46%
	No	41.67%	82.29%	89.06%	92.19%	92.71%	92.71%	93.23%	93.23%	80.21%
Multinomial Naive Bayes	Yes	47.40%	59.38%	53.65%	57.29%	63.54%	49.48%	47.40%	43.23%	7.29%
	No	37.50%	65.63%	73.44%	84.90%	86.46%	86.46%	89.06%	90.10%	79.69%
Nearest-neighbour (IB1)	Yes	78.65%	89.58%	92.19%	92.19%	92.19%	92.71%	92.71%	91.15%	84.90%
	No	33.33%	77.08%	86.46%	90.63%	93.23%	92.71%	92.71%	91.15%	78.13%
Decision Tree (J48)	Yes	78.65%	90.63%	91.67%	93.23%	92.19%	93.23%	92.71%	91.67%	83.85%
	No	40.63%	82.81%	88.02%	91.67%	92.19%	93.23%	92.71%	91.67%	71.35%

Table VI
ACCURACY OF CLASSIFIERS ON SUPPLIED TESTING DATA SETS

Classifer	Length	Window Length	Sixfold Cross-validation	Mono-scenario	Interleaving Scenario
Naive Bayes	Yes	8	93.23%	50.00%	66.67%
	No	8	93.23%	50.00%	83.33%
Multinomial Naive Bayes	Yes	5	63.54%	50.00%	66.67%
	No	8	90.10%	66.67%	83.33%
Nearest-neighbour (IB1)	Yes	6	92.71%	83.33%	83.33%
	Yes	7	92.71%	58.33%	55.56%
	No	5	93.23%	83.33%	83.33%
Decision Tree (J48)	Yes	4	93.23%	58.33%	83.33%
	Yes	6	93.23%	66.67%	83.33%
	No	6	93.23%	66.67%	83.33%

vector machine and neural networks. This need to be explored. The proposed approach has achieved considerable accuracy, however, the accuracy will need to be even higher for successful data provenance and security applications.

3) Data Leakage Detection: A data distributor has given sensitive data to a set of supposedly trusted agents. However, data leakage frequently happens along around the world. For example, in 2013, 1143 leaks of confidential information were recorded and reported in the media and registered by InfoWatch Analytical Center [16].

Our approach presents an efficient way of data leakage detection. Using the log sequence, we can infer the user actions in the cloud. The user includes the legal user and malicious user such as a hacker. Since the logs are at the system level, it is impossible for the hacker to hide his/her track. Another strength of our approach is that our output gives a fine-grained data activity audit so that the cloud user can judge whether the data leakage is happening.

4) Cloud Data Provenance: With the proliferation of database views and curated databases, the issue of data provenance – where a piece of data came from and the process by which it arrived in the database – is becoming increasingly important, especially in scientific databases where understanding provenance is crucial to the accuracy of data.

As we discussed before, our approach can infer data provenance when the data of interest has been created, updated, read or deleted by any command which include database queries. It will be interesting to see result of applying our approach in this area.

5) User Behavior Prediction: Data provenance is only about the process of tracing and recording the origins of data and its movement between databases. It is not sufficient to successfully keep away from security hazards in cloud computing. An intelligent cloud service needs a mechanism for predicting user actions and giving correct early warnings [17]. User behavior prediction is a problem in which we attempt to predict the next set of user actions based on the knowledge of the previous actions. The aggregation and comparison of behavioral patterns in cloud represent a tremendous opportunity for understanding past behaviors and predicting future behaviors. Once this approach is extended to able to infer the user actions in a streaming cloud data, it can be used to document the user behavior history which can then be used to predict the future actions by the user.

Since most attacks follow the same pattern (bait, redirect, exploit, additional malicious software delivery, check-in) [18], [19], [20], our results above indicated that it may be worthwhile to tie these steps together with security alarms and timestamps. This can then be used to predict and therefore prevent future attracts.

B. Conclusions

In the last few years, the emergence of cloud and cloud computing related technologies has changed the way people store and share their data. In order to make a completely secure environment for the data, a great number of cloud security applications and products have been developed.

However, many studies point out that security is still the top challenge for the cloud systems. The cloud users would like to know what has happened to their data. A professional cloud data service should be able to answer this question.

This paper introduces a novel statistical approach for inferring what has happen on cloud data. This problem appears highly under-studied before. The logs data are multivariate and contain noise. However, we have shown that by looking up specific pattern in the system call sequence, considerable accuracy can be achieved. This approach would be a suitable and powerful tool to enhance the trust between the cloud users and service providers.

The cloud computing environment may never be completely free of security problems, however, further studies to identify potential problems and solution in the areas of data leakage detection, cloud data provenance and user behaviour prediction should result in a much more safe and reliable environment.

References

[1] R. K. L. Ko, M. Kirchberg, and B. S. Lee, “From system-centric to data-centric logging-accountability, trust & security in cloud computing,” in *Defense Science Research Conference and Expo (DSR), 2011*. IEEE, 2011, pp. 1–4.

[2] R. K. L. Ko and M. A. Will, “Progger: An efficient, tamper-evident kernel-space logger for cloud data provenance tracking,” in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 881–889.

[3] R. K. L. Ko, P. Jagadpramana, and B. S. Lee, “Flogger: A file-centric logger for monitoring file access and transfers within cloud computing environments,” in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. IEEE, 2011, pp. 765–771.

[4] R. K. L. Ko, P. Jagadpramana, M. Mowbray, S. Pearson, M. Kirchberg, Q. Liang, and B. S. Lee, “Trustcloud: A framework for accountability and trust in cloud computing,” in *Services (SERVICES), 2011 IEEE World Congress on*. IEEE, 2011, pp. 584–588.

[5] C. H. Suen, R. K. L. Ko, Y. S. Tan, P. Jagadpramana, and B. S. Lee, “S2logger: End-to-end data tracking mechanism for cloud data provenance,” in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*. IEEE, 2013, pp. 594–602.

[6] D. W. Archer, L. M. Delcambre, and D. Maier, “A framework for fine-grained data integration and curation, with provenance, in a dataspace.” in *Workshop on the Theory and Practice of Provenance*, 2009.

[7] D. Gotz and M. X. Zhou, “Characterizing users’ visual analytic activity for insight provenance,” *Information Visualization*, vol. 8, no. 1, pp. 42–55, 2009.

[8] P. J. Guo and M. Seltzer, “Burrito: Wrapping your lab notebook in computational infrastructure.” in *TaPP*, 2012.

[9] P. Chen, B. Plale, and M. S. Aktas, “Temporal representation for scientific data provenance,” in *E-Science (e-Science), 2012 IEEE 8th International Conference on*. IEEE, 2012, pp. 1–8.

[10] C. I. Ezeife and Y. Lu, “Mining web log sequential patterns with position coded pre-order linked wap-tree,” *Data Mining and Knowledge Discovery*, vol. 10, no. 1, pp. 5–38, 2005.

[11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: an update,” *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[12] H. Ahonen, O. Heinonen, M. Klemettinen, and A. I. Verkamo, “Applying data mining techniques for descriptive phrase extraction in digital document collections,” in *Research and Technology Advances in Digital Libraries, 1998. ADL 98. Proceedings. IEEE International Forum on*. IEEE, 1998, pp. 2–11.

[13] I. H. Witten, “Text mining,” *Practical handbook of Internet computing*, pp. 14–1, 2005.

[14] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[15] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl, “Moa: Massive online analysis, a framework for stream classification and clustering.” 2010.

[16] I. A. Labs, “The global data leakage report for the 2013,” InfoWatch Analytical Labs, Tech. Rep., 2014.

[17] K. Radinsky, K. Svore, S. Dumais, J. Teevan, A. Bocharov, and E. Horvitz, “Modeling and predicting behavioral dynamics on the web,” in *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012, pp. 599–608.

[18] E. Adar, D. S. Weld, B. N. Bershad, and S. S. Gribble, “Why we search: visualizing and predicting user behavior,” in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 161–170.

[19] M. A. Awad and I. Khalil, “Prediction of user’s web-browsing behavior: Application of markov model,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 42, no. 4, pp. 1131–1142, 2012.

[20] J. J. Lee, R. McCartney, and E. Santos Jr, “Learning and predicting user behavior for particular resource use.” in *FLAIRS Conference*, 2001, pp. 177–181.