Use of Bash History Novelty Detection for Identification of Similar Source Attack Generation

Jack Hance
Department of Computer Science
North Dakota State University
Fargo, ND, USA
jack.hance@ndsu.edu

Jeremy Straub
Institute for Cyber Security Education and Research
North Dakota State University
Fargo, ND, USA
jeremy.straub@ndsu.edu

***Abstract*—When a cyberattack occurs, tracking the attack back to an actual individual person can be problematic. Even identifying the workstation or device that it originated from does not necessarily identify the attacker, as the attacking device could, itself, be compromised. A system to determine whether activities that occur are from the same user or not facilitates forensic analysis as well as the detection of concurrent attacks from different devices by the same user. This paper proposes a system for identifying attackers based on behaviors expressed via their use of the Bash command line interface, the most common shell on Linux distributions. Prior systems were limited by issues such as requiring labeled user data which is difficult to acquire or not being specific enough to monitor individual persons. The approach proposed herein does not require labeled data and is specific enough to target individual users. The proposed system analyzes the level of variance between commands used and calculates an anomaly score for each given command. It uses these anomaly scores to compare Bash history sets together to identify if they were created by the same user.**

Keywords—anomaly detection, computer forensics, Bash history, cybersecurity

I. INTRODUCTION

Humanity has entered an information age where the systems used on a daily basis are becoming more vulnerable than ever. A 2018 study, conducted by the Center for Strategic and International Studies and McAfee [1], estimated the cost of cybercrime, on a global basis, at $445 billion to $608 billion.

The identification of an attacker, during and after a cyberattack, is a critical component of responding to and preventing future cybercrime and cyberterrorism, as well as properly attributing actions to state actors. Attacker identification, during an attack, can assist in defensive (and possibly retaliatory, for some entity types) action decision-making After a cyberattack has occurred, it is important to identify and possibly apprehend the culprit as soon as possible, so that stolen assets and exfiltrated information can possibly be recovered and the extent of the damage can be projected.

Problematically, even identifying the workstation or device that it originated from does not necessarily identify the attacker, as the attacking device could, itself, be compromised. This paper proposes a system that can recognize a user's Bash history via generating a classification score that allows the use patterns of multiple users to be compared. This allows multiple Bash histories to be compared to tell if they have been created by the same user. This is done by parsing the Bash history into a workable form, and then feeding it to a model that generates an anomaly score for the data. This score is generated for multiple data sets that can then be compared.

The proposed work facilitates the development of systems that are suitable for numerous applications and which focus on the identification of users across multiple devices. Prior work used supervised methods which required labeled datasets. Prior systems also focused on the identification of malicious activities instead of focusing, like the current system does, on the identification of multiple command histories being from the same user.

Prior approaches, which focused on malicious activity detection, required significant manual involvement as an expert was required to manually label the existing dataset to notate whether individual entries are malicious or not. These supervised approaches were also not adaptive, as threats that did not exist in the premade comparison set are not able to be identified in presented data. Even discounting these limitations, the systems were designed for network monitoring as opposed to finding or identifying a single individual person who is responsible for a given nefarious activity.

This work is based on the thesis that users tend to fall into a consistent pattern of switching between less common and more common commands at a relatively constant rate. This rate of switching between less common and more common commands can be expressed in the form of the number of commands that are considered anomalous in the entire data set. Since this number would be relatively constant between data sets if they were only considered in their own right, different Bash history sets are compared by training a model on an initial set, and then evaluating new sets by using the original model. Sets that have a similar level of anomalies to the original will have a similar amount of disparity in data when compared to the original. Data sets that do not have a similar number of anomalies, or do not match many patterns of the original data set, will show a very different level of disparity. The similarity to known patterns can be used as a virtual 'fingerprint' of an attacker.

II. Related Work

This section presents prior work in several areas that the current work builds upon. First, work on anomaly detection is reviewed. Next, prior work on computer forensics is discussed.

A. System Based Anomaly Detection

Anomaly detection on systems and system logs has been demonstrated in many different forms and for numerous applications. Bash history is a form of a system log making these techniques relevant to this work.

Supervised machine learning has been previously demonstrated for extracting anomaly information from Bash history [2]. It requires that a set of similar data is available and the requires an expert to manually label the data. This severely limits the utility of the approach, as data needs to have been previously collected and labeled. This approach benefits from being informed from the start as to what is and is not an anomaly, thus a self-training phase is not required. However, this approach is unable to detect new anomalies without human intervention and new training.

An alternate form of anomaly detection that has been used for network security is using system administrator defined anomalies and a system that detects them. One example is SAQL [3]. With SAQL, system events are converted into a stream of data which is then analyzed using predefined-by-administrator queries (written in the SAQL language). This approach is not able to detect new anomalies and it would not work for this application since it does not define anomalous events abstractly enough to create profiles for individual users.

The Bash history is a log of user commands, so log analysis techniques are also informative. Anomaly detection has been used with system logs previously, for example, to detect issues and security breaches [4].

DeepLog [5] is an example of unsupervised training for log analysis. It parses system logs and feeds them into a continuously training unsupervised model. Both the metadata from the logs and the content of the logs are analyzed and used to train a model that then looks for anomalous entries.

An event based system was implemented for the Android operating system [6]. The principal difference between event based and log based anomaly detection is that event based systems analyze system events in real time, rather than retrospectively (as with log analysis).

Natural language processing has been used, along with other data mining techniques, for log analysis in the past [7]. These techniques were shown to have significant promise with the predictive accuracy, in the use case described in the study, being about 90%. The analysis of logs using neural networks for anomaly detection was also demonstrated in [8]. In this study, a generalized approach to detection of anomalies in systems logs using state machines and system events was proposed.

Another approach, collective anomaly detection, considers multiple data sets concurrently and searches for anomalies across them. In [9], a multi-task Gaussian graphical model is used for this. It uses the concept of nodes being dependent on other nodes to extracting anomaly information from the data. This approach is used heavily in the collective anomaly detection field, due to showing promising results. Yet another method of collective anomaly detection is demonstrated in [10]. This study proposes a new framework for the purpose of detecting anomalies over several streams of data at once.

B. Forensics

Whether conducted in near-real time or after the fact, the technique proposed herein is a form of forensics and is closely related to prior work in behavior analysis. Behavior evidence analysis is a technique normally used in physical crimes, taking advantage of physical evidence, crime-scene characteristics, and anything else left or surrounding a crime scene. It also has found applications in cyber based cases. It has been used in cyberstalking cases [11], and could be used similarly in cyberattack investigations.

This is similar to prior use of neural networks and deep learning to detect code similarity, both in source code [12], and in binaries [13]. The use with source code, in particular, is quite similar to command log files.

While not as closely related, sentiment similarity detection in natural language processing [14] is also relevant. These techniques assess the similarity of sentences based on evidence of human sentence sentiment. Bash history does not hold the same amount of sentiment as natural language, but it does have similar structures such as the proximality of related words.

III. Overview of Key Concepts

This work requires knowledge of key characteristics of Bash and anomaly detection. This section provides a review of these concepts and other terminology used throughout this document.

A. Bash Overview

Command: Command refers to any text input given to a command line interface for interpretation. Shell style commands consists of a first word that is the name or reference to the binary to be executed. Subsequent words are delimited by spaces and are arguments for the command. Some systems, such as Bash, allow several commands to be concatenated on a single line.

Command Line History: A log of commands issued by a user via a command line interface. This can include information such as timestamps, but (as with Bash) it can also just include the commands themselves (including applicable arguments).

Bash History: Bash history is the command line history generated by the Bash command line shell. This format includes only the commands themselves, in an unaltered form. It lacks timestamps and any indication as to whether a command was successful or valid. An example of this format is shown in Listing 1.

Listing 1. Example of Bash history

```
cd ~
ls
cd Documents
ls
cat example.txt
vim example.txt
exit
```

B. Anomalies Overview

Broadly, an anomaly can be defined as anything that deviates from a data set at a high enough degree that it is can be considered to be out of place. There are several specific types of anomalies that are relevant to this work that are now discussed.

Point anomalies: A point anomaly is a singular point in a data set which deviates significantly from the set.

Collective anomalies: Collective anomalies are events that are considered unusual when multiple data sets are considered.

Contextual anomalies: Contextual anomalies are events that could be considered normal in one context, but are unusual in the analyzed context.

What constitutes an anomaly in Bash history will vary, due to the diversity of what history files can include. Anomalies could include commands that a user has not used before, commands of significantly longer length than normal and commands that don't make sense amongst concurrently issued commands. An example of a history file showing an anomaly is shown in Listing 2. Line four demonstrates a length anomaly, compared to the rest of the commands in the listing. It is also the only line that chains multiple command together at once. Of course, if the user did this regularly, it would not be an anomaly.

Listing 2. Snippet of Bash history containing an anomaly

```
cd Desktop
ls
touch example.txt
wget -O x 10.134.7.19 --http-user=admin
  --http-password=erqureevat;  ~/x;  rm
  ~/x
cd ~
ls
ls -la
```

C. Terminology

A few key terms are of particular importance. Their definitions are provided in this section.

Word: A single token from a command, strictly defined as any segment of text with preceding and trailing white space.

Document: A string of words, in this case usually referring to a command in its entirety.

Corpus: Collection of all documents in a body of text, in this case referring to a Bash history file in its entirety.

Token: The numerical representation of a word.

IV. Methodology

Bash history has similarities to natural language, as it conveys intention and action and related words are used proximal to each other. It differs from natural language where the flow of words is pseudo-random as multiple orders can have the same result. Placing flags in commands, for example, does not require a specific order unless an argument is being named for subsequent use. Given this focus is placed on techniques involving the specific use of words and the habits of users, instead of natural language techniques.

Novelty detection techniques can be used to profile users based on their "normal" behavior. This is based on the frequency of use of commands and arguments to those commands. This profile can then be used to evaluate different snippets of Bash history to analyze if they were created by the same user.

The Python libraries SciKit Learn [15], Matplotlib [16], and NumPy [17] were used heavily throughout this project. The figures in this paper were generated by Matplotlib.

A. Pipelining Bash History

The Isolation Forest Model does not accept strings as input, so the Bash history needs to be converted into numerical form before it can be analyzed. The commands are tokenized and vectorized to convert words to a corresponding numerical representation. The method used for this is a combination of count vectorization and term frequency inverse document frequency (TF-IDF) vectorization [18]. This combined approach was chosen as it is non-lossy. The two methods simply translate the command strings to a constant numerical form, instead of adding or removing information.

Count vectorizaiton both converts the words into numerical tokens and takes occurrence into account. It operates by creating a matrix of terms and assigning each word to its corresponding column inside the matrix. Words of higher frequency have lower index values and lower frequency words have higher index values. Words that did not exist in the corpus of the initial training set will be ignored and assigned an index of zero when being evaluated. With this initial vectorization of the commands, commands that are the same will be given identical vectors.

TF-IDF vectorization is then used in order to limit noise caused by the presence of a large number of words in longer commands. TF-IDF item weighing is used to scale the frequency of words based on the size of the document they are found in. The longer a command is, the more instances of a word it may contain, thus skewing the frequency of that single word. TF-IDF has two parts: term frequency and inverse document frequency.

Term frequency is the number of occurrences of a single word inside of a document. Each word has its column value scaled by the frequency of itself within the entire document. Terms that appear more frequently are considered more important and thus placed higher within the ranking of words. This ranking may cause words within commands that are common but less important to rise up higher (e.g., --help, which may be used frequently but is unimportant); this effect is reversed in inverse document frequency conversion.

The inverse document frequency step of TF-IDF scales words inversely to their occurrences within the overall text. Thus, less common words become more prominent and more common words become less prominent. This is performed on a per word basis. The number of documents that contain the specified word is determined. Then, the inverse fraction of the frequency across the entire document is calculated and this inverse frequency is scaled logarithmically. This is useful for this application, as the words that are of most interest are the ones that the user tends to use the least frequently. These are the words that are potentially used in anomalous commands.

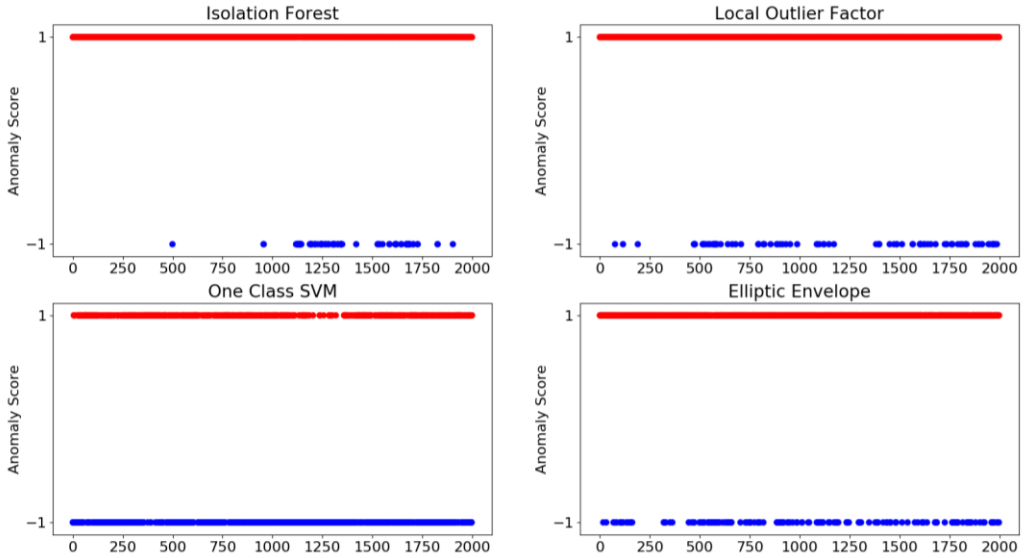

Fig. 1. Out of the 2000 entries given, Isolation Forest marked 100 anomalous. Local Outlier Factor marked 97 anomalous, One Class SVM marked 1022 anomalous, and Elliptic Envelope marked 200 anomalous.

The two techniques create two different models which create dictionaries of words using the first Bash history file provided. Additional Bash history files fed through them are then profiled using these original dictionaries and non-similar commands are parsed out. This limits the Bash histories that are compared to the commands used in the first file, removing the noise of commands that are not used when profiling the second.

B. Novelty Detection Algorithm

Several Novelty Detection algorithms were tested on different sets of Bash History, including Local Outlier Factor [19], Isolation Forest [20], One Class SVM [21], and Elliptic Envelope [22]. The algorithm that was found to perform best, for this application, was Isolation Forest. A comparison of these algorithms is presented in Fig. 1.

Isolation Forest works by using isolation trees that separate infrequently and frequently used commands. As Bash history has a tendency to have many different types of commands executed frequently, there tend to be many small to medium sized clusters of data that would be considered non-anomalous. Isolation Forest handles these chaotic patterns effectively.

It scores commands that the user uses frequently with high values, while scoring commands that are infrequently used with low values. This separation of entries into their scores is shown in Figure 2. This scoring of command frequency creates a profile of the user and what commands they tend to use often and what they use less often. This allows a comparison of the disparity between often used and infrequently used commands between different Bash history files.

The Isolation Forest model is trained using the first data set, so the definition of anomalies is derived only from this set and not skewed by commands that are only in the second. This model is used to generate anomaly scores for each entry in both sets. These two sets of anomaly scores are then analyzed to ascertain the level of similarity between the sets.

C. Analysis of Novelty Scores

A user's Bash history tends to include a relatively consistent amount of noise, since users tend to use a primary set of commands frequently and a secondary set of commands infrequently. How often a user switches between these two is a characteristic of the user and can be quantitatively calculated. This amount of switching between these two modes of command usage is characterized by the variance of the anomaly scores of the given Bash history file.

The vectorizer pipeline and an Isolation Forest model are trained using one set of Bash history. These two trained models are then used to vectorize the second set of Bash history and generate anomaly scores to remove noise and make their usage patterns closer. The anomaly score of each entry in these sets is then calculated; these values tend to lie between -0.3 and -0.5. The variance of each respective set is then calculated and directly compared to create a variance difference D, which is calculated using the equation:

$$D = |Var(h_1^S) - Var(h_2^S)|$$

where h_x^S is found by creating a set of all anomaly score values for the original Bash history set.

Bash histories that have a similar pattern of switching between the frequent and infrequent commands will have similar variances. A comparison will yield a D value closer to 0. Alternately, sets which have a different pattern of switching between the modes will have a high value of D. Current work suggests that a value of D below 2 may be indicative that the Bash histories are from the same person.

D. Requirements for User Identification

Fundamentally, the identification of two particular Bash histories as belonging to a given individual user is a question of statistical significance. The greater the level of difference between the two Bash history files, the more quickly that it can be shown that they are from different users. Files that are similar (or perhaps even have a user trying to impersonate another user) will require additional data to detect their smaller differences in nuance.

While showing that two files have statistically significant differences discounts them from potentially being produced by the same user engaging in a similar process, the lack of statistically significant differences does not guarantee that they are from the same user. Evidentiary standards will need to be

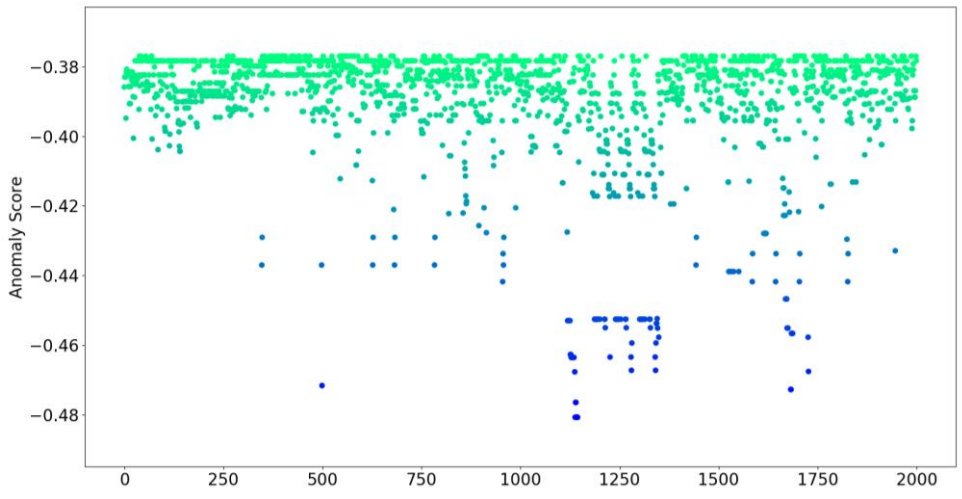

Fig. 2. Isolation Forest score distribution on a set of 2,000 entries. The lower a point is on the graph, the more anomalous the corresponding command is considered.

developed for this technology to determine what level of similarity is acceptable for the use of the analysis for different purposes (such as investigatory purposes, civil suits and prosecutions). Further development and testing of this approach is required to evaluate its suitability for various uses and to identify these thresholds.

V. Data and Analysis

Fig.3 shows a comparison of two matching and two non-matching sets of Bash history. The top graph within the figure shows that the anomalies within the two data sets are relatively similar, while the bottom graph shows that the two data sets have a very different level of variance in the data.

The top data sets consist of two different sets of Bash histories generated by the same user across two different computers. The bottom data sets consists of one of the original generated sets from the top graph along with one artificially generated set of commands resembling the workflow of a C++ developer. A weighted graph system was used to produce pseudo natural, but still artificial, history file. A comparison of these two non-matching Bash histories can be seen in Listing 3.

The limited data sets compared demonstrate the potential promise of this technique. The evaluation of the technique with a larger set of data from multiple users collected across an extended period of time is a key area of future work.

VI. Application Cases

Identification of users from Bash history has a large number of applications in the security and forensics fields. The data could be used to identify if multiple concurrent intrusions or intrusions that occurred over time may be from the same source. This could help in assessment of the threat that is posed by the detected attacks, help predict the potential for future attacks and help determine what type of response is most appropriate.

The tool may also be helpful for forensics evidence processing. If samples of user histories are available or can be collected, it may be possible to identify who committed the attack or confirm that a suspect is the perpetrator using the technique.

One example of how this could be effective is the case of an insider attack. Insider threats and their detection are a subject of ongoing study [23], [24]. The proposed technique could be used to demonstrate that a suspect is the culprit or to help clear an innocent user whose credentials or workstation have been used.

For example, if there has been a breach of information from an employee's computer, it is possible that someone else has used their computer without their knowledge. A method of reducing the suspect pool size would be to compare the bash history of each employee to that of what was left from the attack. This can be done without alerting individual employees, as only their Bash history would be needed for analysis. This would be a low-cost method of determining who might have been involved (and clearing those who are likely not involved).

Another potential use case, in organizational policy enforcement, would be to gather Bash history from employee computers and compare this against verified history files generated by users in the past. This could detect if someone other than the employee is using the computer (possibly in real time), if the histories are significantly different. This may serve as an indication that a laptop has been lost or stolen or that users are improperly sharing credentials.

VII. Weaknesses of the Approach

The proposed approach has several weaknesses that could impair its performance, generally, or in specific circumstances. Several identified weaknesses are now discussed.

A. *Possibility of False Positives*

Due to the method of feature extraction used on the history logs, there is a chance of false positive associations. The proposed method does not examine the content of the history. This could cause users' activity to appear similar if users have a similar level of proficiency or similar training and perform similar activities.

B. *Requirement of Similar Activities*

If the activities in two sets of Bash history from the same individual are different enough, then the two histories could be erroneously classified as being from different sources. What is considered anomalous is defined based on one of the sets. Thus, if the actions that are being performed are different enough, for example comparing sets from a user's home and work computer, they may be classified as being from different users.

This may not always occur, as the analysis of a user switching between less common and more common commands

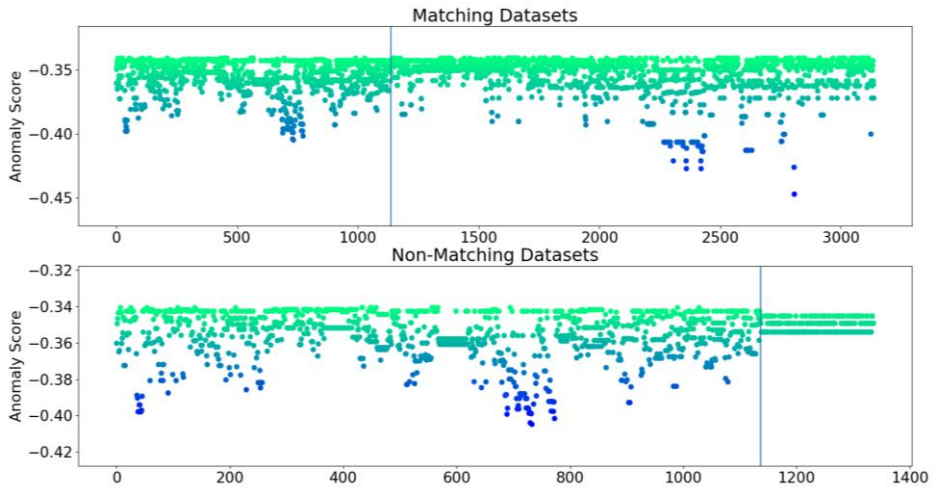

Fig. 3. Comparison of matching and non-matching Bash history data sets.

still may be present between different types of tasks. If the switching pattern used is similar, then the two sets could be identified as being from the same user.

C. Requirement of Attacker Leaving Bash History Unaltered

To be effective, this approach requires unaltered log files. Bash history is a relatively easy log file to access and change or delete on many systems, so the risk of it being altered or destroyed during or after an attack is high. An attacker may delete the history simply to attempt to hide what he or she did, without even being aware of the possibility of its analysis for identification. This issue is part of the broader area of anti-forensics [25]. This type of an approach would require several layers of protection against file manipulation or deletion. While a missing history file is problematic for any forensic investigation, the proposed technique cannot be used if files are not available.

D. Requirement of Large Amount of History for Comparison

In order for there to be significant comparison between multiple sets of Bash history, each history log must be of substantial length. It was noted during this work that sets of history with under 200 entries were unreliable for comparison. Due to the nature of Bash history, it is common for large amounts to be naturally generated quickly so, in many cases, there will still be sufficient Bash history for comparison.

VIII. Areas of Potential Future Work

This paper presented initial work on the topic of user identification from history analysis. Given this, there are a number of areas of prospective future work, which are discussed in this area.

A. Use of Supervised Learning on Bash History

A method of analysis that would be worthwhile to pursue in future work is the use of supervised learning on Bash history rather than the unsupervised approach used in this research. Supervised learning requires effort to label large amounts of data for analysis. Thus, it may not be suitable for all uses but it could have promising results. Supervised learning could be used for detecting anomalies or it could be used for labeling if two sets of Bash history were created by the same source or not.

B. Different Methods of Pipelining Bash History for Analysis

The method of pipelining used herein maintained words position and content. The words are simply converted to numbers with slight calculation to scale them across the entire document. There are other methods for performing pipelining that could be considered.

One established method of converting words to vectors for analysis is Word2Vec [26], [27]. This approach holds the context of words after vectorization, which maintains word proximity knowledge. The current approach does not consider word proximity, losing the command's context.

The Word2Vec approach would facilitate consideration of the order of flags in commands. This would allow the system to differentiate between typical and atypical behavior at a more granular level. Users may have different common orderings of command flags. Using the Word2Vec approach could differentiate between entries with different flag patterns.

Data regarding the use of command characteristics and the use of special functions could also be analyzed, independent

Listing 3. Comparison of two unlike Bash histories in text form.

SET A (Human Generated)

```
./qr
objdump -d qr
base64 qr
objdump -help
objdump qr
hexdump qr
hexdump -help
hexdump -h
xxd qr
hexdump qr | xxd -r temp
hexdump qr | xxd -r
hexdump qr | xxd -r > temp
```

SET B (Pseudo Human Generated)

```
g++ matrixproxy.cpp -o a.out
touch vgui_bitmapimage.cpp
touch initializer.cpp
g++ matrixproxy.cpp vgui_bitmapimage.cpp
    initializer.cpp -o a.out
./a.out
g++ matrixproxy.cpp vgui_bitmapimage.cpp
    initializer.cpp -o a.out
g++ matrixproxy.cpp vgui_bitmapimage.cpp
    initializer.cpp -o a.out
g++ matrixproxy.cpp vgui_bitmapimage.cpp
    initializer.cpp -o a.out
```

from just being considered as part of the Bash history. Examples of potentially interesting functionality and characteristics include command length, the use of privilege escalation, the number of arguments used, use of command chaining, whether commands are run in the background, writing results to a file and accessing the internet. Changes in the use of these commands may be identifiable anomalies for analysis.

Using multiple input sources could enhance the accuracy of identifying usage patterns and, thus, users. This could lower the possibility of data sets being mistakenly flagged as similar by providing more ways for distinctions to be identified.

C. Different Modes of Extraction of Bash History

There are several methods used for storing command history on Linux systems. The method considered in this work is the .bash_history file, which is typically located in a user's home directory. While this is a convenient way of collecting Bash history, it may not be the method that provides the most information. Many Linux distributions also include a history command that provides the command history held in memory.

Bash history is not written to the user's history file until shutdown, on many systems. The history command provides all history currently available on the system for the current user. The use of this command instead of collecting from the user's home directory would potentially provide a larger set and more current history data to be analyzed. Notably, the common forensics response technique of unplugging a suspect computer (to avoid processes changing stored data) may result in the loss of recent history data if action is not taken to preserve it.

D. Supplemental Information Added to Bash History During Runtime

Bash history, in its unaltered state, provides a large amount of information. However, it lacks several types of data that would be beneficial for post-attack analysis. Logging additional data on systems could aid future analysis.

Bash history provides no indication of the time between commands. One feature that would facilitate analysis is the inclusion of timestamps in bash history. While this is a feature that can be used in bash history [28], it is not implemented by default on many popular Linux distributions.

The inclusion of timestamps would allow for the analysis of usual patterns in the context of time. For example, the time between commands submitted may be of interest. Changes in command timing could indicate different user levels of proficiency, typing speeds or even automated input.

Bash allows programs to return error codes at the end of their execution. These error codes hold information about the status of the program that ran or the command itself. By default in Bash, an error code is returned when an invalid command is submitted. This data would be interesting for analysis. A change in the level of erroneous commands issued could indicate a different level of operating fluency or a lack of knowledge about the system's configuration. Both could potentially be indicators that a different user was operating the system.

Bash also allows the level of privilege of the current user on the system to be altered. Commands can be individually performed at a heightened level of privilege, or a user may move to a higher level of privilege and issue several commands in sequence at that level. The use of privilege change commands, if they are not normally used, could indicate that a malicious third-party is executing commands on a victim's computer.

E. Other Anomaly Detection Methods

There are many methods of anomaly detection, including those based on statistical analysis and machine learning. Each approach has different strengths and weaknesses and different accuracy when applied to a given application. Thus, evaluating the performance of other techniques could be beneficial.

One potential approach, autoencoders, use a method of abstracting data into a simplified form. They then use this minimized abstract form to try to recreate the original data. This can be used for anomaly detection by measuring the difference between what was expected and what was produced by the encoding and decoding process. This method has been successfully demonstrated in [29]. It has also been demonstrated to be effective for natural language analysis applications, such as grammar analysis and generation [30].

Another potentially useful form of machine learning, for this application, is cluster analysis. Cluster analysis has been previously used for anomaly detection. In [31], it was demonstrated for use in detecting outliers in the use of graphical interfaces. This may be effective for log analysis as it may reveal detailed data about what a user does within their Bash history.

Many anomaly detection techniques label in a binary fashion without providing insight as to why a certain item was flagged as anomalous while others were not. Cluster analysis provides more meaning for individual commands' anomaly scores. The principal drawback of this technique is that the actual labeling of anomalous scores cannot be done as easily automatically. Instead of evaluating commands by their spread anomaly scores, it may be beneficial to compare them by the number of clusters present and the clusters' shapes.

IX. Conclusion

This paper has presented a technique for user identification based on log file analysis. Specifically, the analysis of Bash logs has been studied. The technique has been described in this paper and the results of initial analysis have been presented. The paper has also discussed several prospective weaknesses with the proposed system, under some circumstances, and has presented a number of possible areas for future work.

The goal and focus of this work was to find a method of natural text analysis on human generated commands issued on the Bash command shell, that would yield significant and specific enough information on users that it would allow for direct comparison of the data between different data sets. This would aid in forensic research as it has a low monetary and time cost in the initial phases of post-attack forensics. It does not require user involvement, past potentially providing system logs. It can also be applied to a broader set of applications such as verification of individuals using the correct workstations in an office and the detection of laptop theft. The method of analysis presented is a starting point for a large field of analysis that can

be performed based on user pattern detection in Bash history and other similar log files.

This work is part of a larger field of analyzing human generated text. This is growing application area for machine learning and statistical analysis. The work presented herein can, thus, potentially benefit from advances in this broad field.

With the growing number of cyberattacks and the considerable threat that they pose, computer forensics techniques are critical for analyzing attacks to facilitate recovery and to prepare for future attacks. User analysis can aid computer forensics efforts by generating meaningful information about user habits that would normally be not available using only traditional analysis techniques.

ACKNOWLEDGMENT

This research was supported by the United States National Science Foundation (NSF award # 1757659). Some facilities and equipment were provided by the NDSU Institute for Cyber Security Education and Research and the NDSU Department of Computer Science. Thanks are given to Ben Bernard for his involvement.

REFERENCES

[1] J. Lewis, "Economic Impact of Cybercrime—No Slowing Down," Feb. 2018.

[2] M. Ahmed, A. Naser Mahmood, and J. Hu, "A survey of network anomaly detection techniques," Journal of Network and Computer Applications, vol. 60. Academic Press, pp. 19–31, 01-Jan-2016.

[3] P. Gao et al., "Saql: A Stream-based Query System for Real-Time Abnormal System Behavior Detection," in Proceedings of the 27th Usenix Security Symposium, 2018.

[4] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience Report: System Log Analysis for Anomaly Detection," in Proceedings - International Symposium on Software Reliability Engineering, ISSRE, 2016, pp. 207–218.

[5] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in Proceedings of the ACM Conference on Computer and Communications Security, 2017, pp. 1285–1298.

[6] G. Safi, A. Shahbazian, W. G. J. Halfond, and N. Medvidovic, "Detecting event anomalies in event-based systems," in Proceedings of the 2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2015, pp. 25–37.

[7] C. Bertero, M. Roy, C. Sauvanaud, and G. Tredan, "Experience Report: Log Mining Using Natural Language Processing and Application to Anomaly Detection," in Proceedings - International Symposium on Software Reliability Engineering, ISSRE, 2017, vol. 2017-October, pp. 351–360.

[8] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," in Proceedings of the 1st Workshop on Machine Learning for Computing Systems, 2018, vol. 18, pp. 1–8.

[9] T. Ide, D. T. Phan, and J. Kalagnanam, "Multi-task multi-modal models for collective anomaly detection," in Proceedings - IEEE International Conference on Data Mining, ICDM, 2017, vol. 2017-November, pp. 177–186.

[10] Y. Jiang, C. Zeng, J. Xu, and T. Li, "Real time contextual collective anomaly detection over multiple data streams," in Proceedings of the ODD'14 Conference, 2014.

[11] N. Al Mutawa, J. Bryce, V. N. L. Franqueira, and A. Marrington, "Forensic investigation of cyberstalking cases using behavioural evidence analysis," in DFRWS 2016 EU - Proceedings of the 3rd Annual DFRWS Europe, 2016, vol. 16, pp. S96–S103.

[12] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, 03-Sep-2016. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7582748?casa_token=mur8dqJfq7MAAAAA:mOvQDPxGy3fmqKQxVRv43dJJGShxaKs2NM1ZuixcEZCMwusHFsfYwsxxBApyA92afX86PMVgaQ. [Accessed: 27-Jul-2020].

[13] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 363–376.

[14] H. He and J. Lin, "Pairwise Word Interaction Modeling with Deep Neural Networks for Semantic Similarity Measurement," in Proceedings of NAACL-HLT, 2016, pp. 937–948.

[15] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[16] J. D. Hunter, "Matplotlib: A 2D graphics environment," Comput. Sci. Eng., vol. 9, no. 3, pp. 99–104, May 2007.

[17] T. E. Oliphant, Guide to NumPy, 1st ed. CreateSpace Independent Publishing Platform, 2015.

[18] L. Havrlant and V. Kreinovich, "A simple probabilistic explanation of term frequency-inverse document frequency (tf-idf) heuristic (and variations motivated by this explanation)," Int. J. Gen. Syst., vol. 46, no. 1, pp. 27–36, Jan. 2017.

[19] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in Proceedings of the 2000 ACM SIGMOD international conference on Management of data, 2000, pp. 93–104.

[20] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation forest," in Proceedings of the IEEE International Conference on Data Mining, 2008.

[21] C. C. Chang and C. J. Lin, "LIBSVM: A Library for support vector machines," ACM Trans. Intell. Syst. Technol., vol. 2, no. 3, pp. 1–27, 2011.

[22] P. J. Rousseeuw and K. Van Driessen, "A fast algorithm for the minimum covariance determinant estimator," Technometrics, vol. 41, no. 3, pp. 212–223, 1999.

[23] A. Sanzgiri and D. Dasgupta, "Classification of insider threat detection techniques," in Proceedings of the 11th Annual Cyber and Information Security Research Conference, 2016, pp. 1–4.

[24] B. Bose, B. Avasarala, S. Tirthapura, Y. Y. Chung, and D. Steiner, "Detecting Insider Threats Using RADISH: A System for Real-Time Anomaly Detection in Heterogeneous Data Streams," IEEE Syst. J., vol. 11, no. 2, pp. 471–482, Jun. 2017.

[25] M. Gül and E. Kugu, "A survey on anti-forensics techniques," in IDAP 2017 - International Artificial Intelligence and Data Processing Symposium, 2017.

[26] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," arXiv, Sep. 2013.

[27] Y. Goldberg and O. Levy, "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method," arXiv.org, Feb. 2014.

[28] V. Gite, "Bash History Display Date And Time For Each Command," nixCraft Website, 2018. [Online]. Available: https://www.cyberciti.biz/faq/unix-linux-bash-history-display-date-time/. [Accessed: 27-Jul-2020].

[29] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," in Proceedings of the AAAI Conference on Artificial Intelligence, 2019, vol. 2495, no. 01.

[30] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar Variational Autoencoder," arXiv.Org, vol. 4, Mar. 2017.

[31] V. Avdiienko, K. Kuznetsov, I. Rommelfanger, A. Rau, A. Gorla, and A. Zeller, "Detecting behavior anomalies in graphical user interfaces," in Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017, 2017, pp. 201–203.