Automatic Event Log Abstraction to Support Forensic Investigation

Hudan Studiawan, Ferdous Sohel, Christian Payne

College of Science, Health, Engineering and Education
Murdoch University, Perth, Australia

The Australasian Information Security Conference (AISC 2020)
Swinburne University of Technology, Melbourne, Victoria, Australia

CORE Student Travel Award

We acknowledge that we have received
a CORE Student Travel Award.

Outline

- Introduction
- Existing Methods
- The Proposed Method
    - Event Log Preprocessing
    - Grouping based on Word Count
    - Graph Model for Log Messages
    - Grouping with Automatic Graph Clustering
    - Extraction of Event Log Abstraction
- Experimental Results
- Conclusion and Future Work

Introduction

- Abstraction of event logs is the creation of a template that contains the most common words representing all members in a group of event log entries
- Abstraction helps the forensic investigators to obtain an overall view of the main events in a log file

Input log file: auth.log
Output abstractions:
#1 Mar * * nssal * removing removable location: *
#2 Mar 8 * nssal * Invalid user * from *
#3 Mar 8 * nssal * Failed password for * from * port * ssh2
...

Existing Methods

Existing log abstraction methods require user input parameters. It is time consuming due to the need to identify the best parameters.

- SLCT (Vaarandi, 2003): one mandatory parameter and 14 optionals
- LogCluster (Vaarandi and Pihelgas, 2015): one mandatory parameter 26 optionals
- IPLoM (Makanju et al., 2012): five mandatory parameters
- LogSig (Tang et al., 2011): one mandatory parameter
- Drain (He et al., 2017): three mandatory parameters
- Model training (Thaler et al., 2017)

The Proposed Method

Raw event logs

↓

Automatic log preprocessing

↓

Grouping based on word count

↓

Refine grouping with automatic graph clustering

↓

Get the event log abstraction per cluster

Event Log Preprocessing

- We parse the log files using the nerlogparser, a log parsing tool based on named entity recognition
- It supports fully automatic parsing because it provides a pre-trained model
- We then extract unique messages from the log entries

Input:
Jan 18 09:31:32 victoria dhclient: DHCPACK from 10.0.2.2

Process: automatic parsing with the nerlogparser tool

Output:
timestamp: Jan 18 09:31:32
hostname: victoria
service: dhclient
message: DHCPACK from 10.0.2.2

Grouping based on Word Count

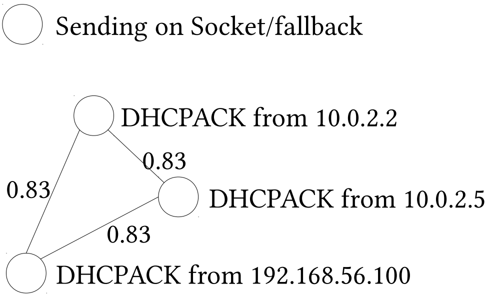
- We split the discovered unique messages based on space character then count the word length
- An abstraction is extracted from the always-occurring word in a group of log entries having the same length

```
Cluster #1:
Jan 18 09:31:32 victoria dhclient: DHCPACK from 10.0.2.2
Jan 18 10:56:40 victoria dhclient: DHCPACK from 10.0.2.2
Feb  6 13:31:12 victoria dhclient: DHCPACK from 10.0.2.5

Abstraction #1:
*   * *      victoria dhclient: DHCPACK from *

Cluster #2:
Feb  6 12:56:48 victoria init: Switching to runlevel: 0
Jan 18 17:13:49 victoria init: Switching to runlevel: 6
Feb  6 13:03:53 victoria init: Switching to runlevel: 6

Abstraction #2:
*   * *      victoria init: Switching to runlevel: *
```

Graph Model for Log Messages

The log entries have very diverse vocabularies, so we need to refine discovered groups based on the string similarity

- We use an automatic graph-based clustering
- Vertex: a unique message, edge: the weighted Hamming similarity



Grouping with Automatic Graph Clustering

Algorithm 1: The proposed automatic graph clustering.

```
Input: graph
Output: best_cluster
Procedure GetClusters(graph):
    best_cluster ← dict()
    max_modularity ← -1
    clusters ← GirvanNewman(graph)
    for cluster in clusters do
        modularity ← get_modularity(cluster, graph)
        if max_modularity < modularity then
            max_modularity ← modularity
            best_cluster ← cluster
        end
    end
End Procedure
```

Building Micro-clusters

Algorithm 2: Building micro-clusters automatically.

```
Input: graph
Output: final_clusters
Procedure AutomaticClustering(graph):
    clusters ← GetClusters(graph)
    final_clusters ← dict()
    for cluster in clusters do
        if len(cluster) ≤ micro-cluster size then
            final_clusters.update(cluster)
        else
            AutomaticClustering(cluster) // recursion
        end
    end
End Procedure
```

Extraction of Abstraction: Merging Abstractions

- We extract an abstraction from each micro-cluster
- Merging is needed because an abstraction from each micro-cluster has a possibility to be very similar with others
    - We find pair combinations (A_i, A_j) from all abstractions to be compared.
    - Two abstractions A_i and A_j will continue to be checked for merging if there is a weighted Hamming similarity between them.

Example of Merging Abstractions

Example 1:
Abstraction #1: Invalid user * from *
Abstraction #2: Invalid user admin from *

Example 2:
Abstraction #1: Invalid user * from 200.27.148.45
Abstraction #2: Invalid user * from *

Extraction of Abstraction: Final Abstractions

- In all previous steps, we consider only the message field in a log entry.
- In the final step, we consider all other fields such as timestamp, host name, and service name.

```
Cluster #1:
Jan 18 09:31:32 victoria dhclient: DHCPACK from 10.0.2.2
Jan 18 10:56:40 victoria dhclient: DHCPACK from 10.0.2.2
Feb  6 13:31:12 victoria dhclient: DHCPACK from 10.0.2.5

Abstraction #1:
*    * *        victoria dhclient: DHCPACK from *

Cluster #2:
Feb  6 12:56:48 victoria init: Switching to runlevel: 0
Jan 18 17:13:49 victoria init: Switching to runlevel: 6
Feb  6 13:03:53 victoria init: Switching to runlevel: 6

Abstraction #2:
*    * *        victoria init: Switching to runlevel: *
```

Experimental Results: Datasets

Murdoch University

Table 1: List of Public Forensic Datasets

Dataset	# Files	# Lines	# Abs
Digital Corpora (Casper) [7]	15	11,086	3,422
DFRWS 2009 (Jhuisi) [2]	25	11,737	3,488
DFRWS 2009 (Nssal) [2]	40	107,093	5,573
DFRWS 2016 (DF16) [3]	1	3,304	102
Honeynet Challenge 7 (Honey) [1]	12	8,712	2,039

- For all datasets except DFRWS 2016, we recovered the directory `/var/log/` from the forensic disk images
- We retrieved some common log files such as authentication logs, kernel logs, and system logs

Parameter Settings

Table 2: Parameter Settings for Experiments

Method	Parameter settings
Proposed method	none
IPLoM [17]	file support threshold = 0
	partition support threshold = 0
	upper bound = 0.9
	lower bound = 0.25
	cluster goodness threshold = 0.175
LogSig [24]	number of cluster = ground truth cluster
Drain [12]	tree depth = 4
	similarity threshold = 0.4
	maximum child = 100
LogMine [10]	levels of pattern hierarchy = 2
	maximum distance = 0.001
	distance weight = 1
Spell [4]	message type threshold = 0.5

Comparison of Performance

Table 3: F-measure Value Comparison (in %) of The Proposed Method and Five Other Methods

Method	Datasets				
	Casper	Jhuisi	Nssal	DF16	Honey
IPLoM [17]	93.72	87.58	88.15	10.90	89.31
LogSig [24]	77.16	80.33	79.71	12.00	85.50
Drain [12]	90.01	87.49	89.05	17.50	94.12
LogMine [10]	72.06	74.17	66.17	14.60	77.49
Spell [4]	82.00	82.02	79.00	10.80	83.70
Proposed method	**94.94**	**96.32**	**92.11**	**97.10**	**96.26**

- IPLoM shows a good performance because the bijective relationship in a group of log entries can accurately capture the most frequently occurring words
- LogSig' clustering is performed based on a local search algorithm and can lead to local optima. Therefore, it cannot cluster log messages precisely

Comparison of Performance

- Drain performs well because it considers the first few words in a log entry as contributing most significantly to its abstraction. These words are used to construct a fixed-depth tree.
- LogMine performs over-clustering for all datasets because the clustering process is conducted incrementally. If a log entry similarity with an existing cluster representation is less than the given threshold, it will be grouped with that particular cluster.
- Spell employs the longest common subsequence (LCS) technique to obtain the abstractions. LCS cannot capture any potential abstraction that has separate substrings.

Over-clustering vs Under-clustering

- The most important procedure in discovering event log abstractions is the clustering step
- If the clustering is performed well, then good abstractions will be produced
- We need to get the best cluster composition from event logs

Conclusion and Future Work

- This paper proposes an automatic method of event log abstraction
- Being automatic, there is no need for a forensic investigator to supply any parameters
- This is a significant improvement as the existing approaches either need many user inputs or need a model training
- Future work will focus on integrating the automatic abstraction with event reconstruction and anomaly detection

References

- Risto Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In Proceedings of the IEEE Workshop on IP Operations and Management. 119126.
- Risto Vaarandi and Mauno Pihelgas. 2015. LogCluster - a data clustering and pattern mining algorithm for event logs. In Proceedings of the 11th International Conference on Network and Service Management. 17.
- Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. 2012. A lightweight algorithm for message type extraction in system application logs. IEEE Transactions on Knowledge and Data Engineering 24, 11 (2012), 19211936.
- Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating system events from raw textual logs. In Proceedings of the 20th ACM International Conference on Information and Knowledge Management. 785794.
- Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In Proceedings of the IEEE International Conference on Web Services. 3340.
- Stefan Thaler, Vlado Menkonvski, and Milan Petković. 2017. Towards a neural language model for signature extraction from forensic logs. In Proceedings of the 5th International Symposium on Digital Forensic and Security. 16.

Thank you

AISC 2020