Practical Log Analyzer to Detect SSH Compromise

Jaeho Lee (Jaeho.Lee@rice.edu)
Rabimba Karanjai (rk45@rice.edu)

April 22, 2016

Abstract

In this paper, we propose a general approach to analyzing log files and implement a framework to detect SSH compromise with log files. Our approach divides the log analysis process into the three steps: parsing log, managing abstract data and building back-end analyzers. In the first step, the log files are parsed based on the sequence of log in, log out events, and all session information are identified and extracted. In the second phase, the parsed results are abstracted to a higher level and saved in the database. We provide database APIs to facilitate access to the information. They are used in the last step in making back-end detection modules. We offer three back-end modules: the analysis reporter, the rule-based detector, and the behavioral-based detector. We paired two techniques to detect compromise effectively. The rule-based detector finds the compromise by checking violation of given rules. The behavioral-based detector finds the compromise by detecting an anomaly in user behavior patterns. They are complementary and allow for automatically detecting suspicious accesses only using SSH log files. We have implemented our approach and applied the framework to real SSH logs. From the experiment, our framework provides useful analysis report which helps us understand more about the attacks, and finds suspicious concurrent accesses from different locations.

1 Introduction

Since the explosion in the popularity of the internet, the number of attacks has been rapidly increasing, and the threat of attack on the network servers has been growing in the proportion to the advancement in the technology. Among the growing threats of hacking, Secure Shell (SSH) attacks are a major area of concern for security people because of its danger associated with a successful compromise[11]. Since

the intrusion detection model was proposed in 1987 [6], there have been a plethora of studies of the intrusion detection using data mining, machine learning, and rule-based detection [10]. However, most of the machine learning approaches assume specific environments so that they are often not practical in the general environment, and only a handful of those work have been applied in the real field [27]. Also, Intrusion Detection Systems (IDS) focus on real-time detection, so they require the installation in advance, and generally, rule-based detectors prevent only obvious attacks conservatively.
In our research, we paid attention to SSH log files. SSH log files exist in almost every SSH server and are accumulated all the time. However, there is less research to exploit these log files, and it is also difficult for an administrator to audit the log manually. Therefore, they usually end up occupying spaces uselessly. Our research goal is to exploit those SSH log files and provide the practical solution to detect SSH attack and compromise.

Hypothesis. We conduct our research with two hypothesis. The first one is that SSH log files in the real world have enough variance so that we can extract meaningful features and exploit them to make detectors. For example, the compromise by the brute-force attack can be easily identified by checking a successful login after a bunch of authentication failures. Also, with the pair of connection and disconnection time messages, we can pick up several features: average access time per user, access location pattern, weekly access pattern and failure frequency. We could build effective SSH compromise detector with that information.
Our second hypothesis is that it is highly likely that the attacker's behavior deviates from normal user's behavior, which means that the attacker's access pattern to SSH can be differentiated from the normal user's one. It is hard for intrusion prevention software like fail2ban to prevent the access of the malicious user who exploits a stolen credential of a normal user. However, if malicious users' access patterns are quite different from normal users' one, we can reveal their access using behavioral patterns from log files. For example, attackers may stay access during a very short time or connect from a strange location.

Based upon our hypothesis, we developed the practical log analyzer that detects compromise with rules and behavioral patterns. It can be applied in the general environment such as personal machines. This research is useful because SSH attacks are common these days [21] and many people open SSH port on their PC, but they hardly install the IDS or investigate their SSH logs manually to find anomalies. But if they have accumulated SSH log files, our framework could be applied and help them to analyze the SSH log. While implementing the framework, we followed the log analysis approach we propose, and we implemented the framework easy to extend by adding new rules. This approach will be applied to analyze other type of logs.

Challenges in our research. When we design a practical log analyzer that runs in a general environment, we have to find the solutions for following challenges.

- how to identify all SSH log messages and correlate them to be meaningful
- which features to extract from the log messages so as to correlation analysis
- how to effectively manage extracted log data
- how to implement rule-based detector and pattern-based detector
- how to experiment and verify our result

Contributions. In our research, our contributions are summarized as follows:

- *Proposing a general approach to analyzing logs.* We have proposed a general method to analyze log files.
- *Providing a practical SSH log analyzer.* We have developed the practical framework that analyzes SSH logs and helps detects anomaly by mixing rule-based and behavioral-based detection.
- *Applying to real SSH data.* We evaluated our framework with the real big field data, analyzed them and found some suspicious activities.

In the remainder of this paper, Section 2 provides an overview of our approach, Section 3 describes our implementation in detail, Section 4 evaluates our work, Section 5 discusses related work, Section 6 explains future work, and Section 7 draws some conclusion.

2 Overview of Approach

We come up with a general process to analyze log files and build the analyzers. Figure 1 shows our approach. It is composed of three steps: parsing log, managing abstract data, and building back-end analyzers. The operation of those steps is similar to one with conventional compilers. As compilers parse source codes, the log analyzer parses a log file and generates meaningful data. While, instead of generating

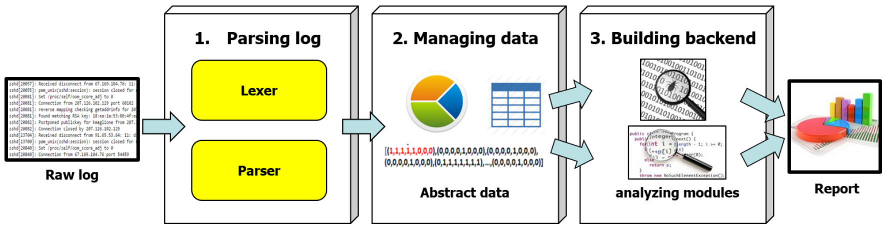


Figure 1: Overview of log analysis process

an abstract syntax tree (AST) or object codes resulting from compiling, parsing results of the analyzer should be abstract data that gives meaningful information. In the second step, it should provide the way to manage abstract date effectively, and also provide the easy way to retrieve the data and correlate them, so upon this infrastructure, backend analysis modules could be added easily. In the last step, analysis modules are implemented on the infrastructure. Different modules can be incrementally implemented according to the purpose.

We applied this process to our SSH log analysis. In the first step, we implemented a parser that accepts SSH log. In the second step, we use the relational database to manage the parsing result by identifying key features to tables and save them in the database management system (DBMS). In the last step, we implemented detection modules which conduct analysis and detection using the data in DBMS. In this chapter, we explain them in detail.

2.1 Parsing log

The log parser plays two roles in our framework. The first role is to parse raw log data and extract all session information per user. The second role is to analyze the session information more and get higher abstraction such as user information and attack sessions.

In the first role, it reads a log message line by line and check which session this message belongs to. The session means a set of messages from connection to disconnection. If the message is the connection message, it creates a new session, or if it is the disconnection event or error event, it closes the open session and saves it. In this way, it groups all log messages as session and stores them.

Secondly, it extracts higher abstraction from session information. By correlating a user's name, IP addresses and a user's public key, it extracts higher level information such as user profile, attack session, and suspicious IP addresses. Finally, the log parser populates the database with the result.

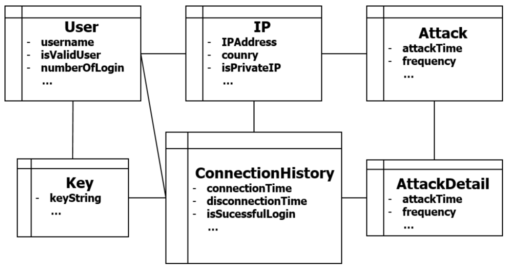


Figure 2: database table scheme in our framework

2.2 Managing abstract data

We abstract the parsing result and save them to tables in DBMS. It contains high-level information such as user information and attack streams. Also, we provide DB APIs that are used by the detectors in the later step. Figure 2 shows all relational tables we came up with. The User table contains all user information appeared in SSH login events. Therefore, it contains both valid users and invalid users that are used only in brute-force attacks. We identified the valid users by checking successful login numbers. The IP table contains all IP addresses that are used to connect the SSH server. For all those IP addresses, we gathered more information such as country, city and detail coordinates, which are used by detectors when checking users' moving pattern or concurrent accesses from different locations. The ConnectionHistory table includes all established session information such as when the session started and finished or which key was used. The Key table includes all public key strings used in the access. The Attack table and the AttackDetail table contain connection information that is presumed to be an attack. We defined consecutive access events as an attack if they try to access more than 30 times in a minute or with more than five different users in a minute. Suspicious IPs are derived from this table, and they are used in the detection.

2.3 Detection modules

The detection modules are the back-end engine and core module in our framework. At this moment, we provide three engines: two detector components (Behavioral-Based Detector, Rule-Based Detector) and one log analyzer (Analysis Reporter).

The rule-based detector works with defined rules. It looks through all DB tables while checking if there is any violation of the given rules. The rules are the set of predicates such as "successful login IP should not be in blacklist IP". It is designed to add new rule easily.

The behavioral-based detector looks through all history DB and profile a user information from defined behavioral features. After finishing to profile the behavior of every user, it looks through DB entries further and check if there is any exceptional case that deviates from the previous patterns. This behavioral-based detector is complementary to the rule-based detector in that it reduces false positives and finding a new anomaly. Detection results are separately reported from these two detectors.

Also, we provide the additional log analysis reporter that extracts useful information from the database and reports the analysis. For example, it gives information such as how many attacks happened during the term or what usernames are most frequently used in attack.

3 Implementation and Challenges

In our research, we implemented a prototype of our framework using Java and Python, and used MySQL as a DBMS. In this chapter, we will explain what challenges we faced and how we implemented the framework.

3.1 Parsing log

To implement the parser, we have to know token information (unique messages) and grammar information (how they are related). The first challenge is less documentation about SSH log. Our approach is to obtain that information by analyzing real SSH logs. We developed tools that identify unique log messages which are regarded as tokens and manually found the relationship among log messages which are considered as the grammar.

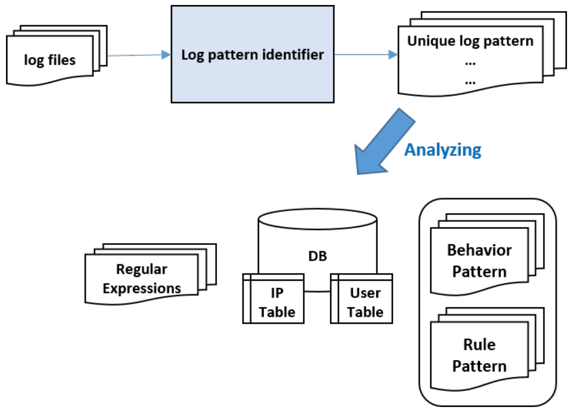


Figure 3: Structure of log parsing step

⟨ConF⟩	⊨	Connection from ⟨IP⟩ port ⟨PORT⟩
⟨ClosingTo⟩	⊨	Closing connection to ⟨IP⟩ port ⟨PORT⟩
⟨Invalid⟩	⊨	Invalid user ⟨USER⟩ from ⟨IP⟩
⟨RcvdD⟩	⊨	Received disconnect from ⟨IP⟩:⟨ PORT ⟩⟨MSG⟩
⟨FRSA⟩	⊨	Found matching RSA key: ⟨KEYSTRING⟩
⟨SOpen⟩	⊨	pam_unix(sshd:session): session opened for user ⟨USER⟩by⟨UID⟩
⟨SClose⟩	⊨	pam_unix(sshd:session): session closed for user ⟨USER⟩

Figure 4: Example of regular expression for log messages

Figure 3 shows the structure of this phase. Firstly, we made the log pattern identifier which gets the input of SSH log files and results unique log messages and their frequency. Figure 4 shows the example of the regular expressions that are used in tokenizing. After finding all unique SSH log messages, we manually analyzed how they relate each other. For example, we identified the sequence of messages for a successful login session and for a failed login session. Another challenge is that some log messages contain incomplete information to decide its

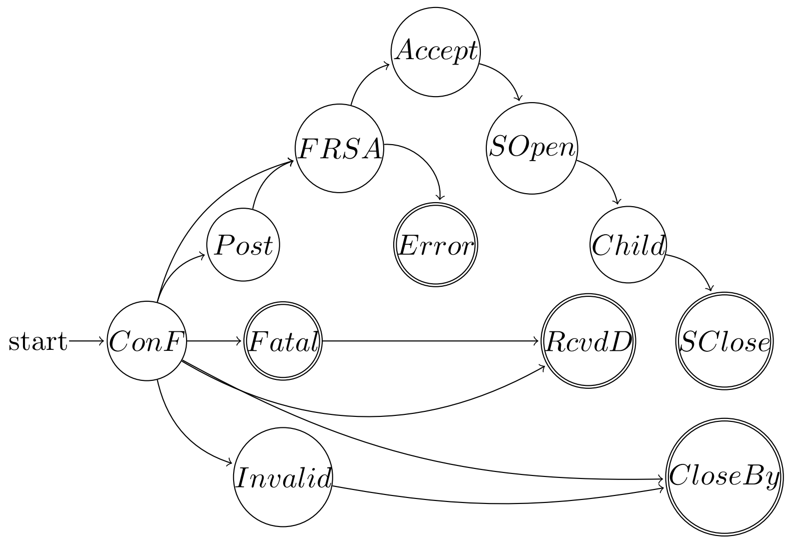


Figure 5: Example of the relation between log messages

stream. For example, some types of messages contain no IP address or username, so there is no clue, as the message, ”fatal: mm_request_send: write: Broken pipe”. Therefore, we used heuristic way to minimize ambiguity. Figure 5 shows one example of the relationship among log messages as the form of automata. We built the parser upon them using Python.

3.2 Managing abstract data

We built the log database system using MySQL on Ubuntu 14.04 and implemented DB-interaction APIs that provide an easy way to insert, select, and update entities in the database. The APIs are used in implementing rule-based detector, behavioral-based detector, and other back-end modules. We implemented Log2DB module that gets the session data from the parser and creates corresponding entities and inserts them to DB. For example, it creates user, key, IP, connection history entities from one session information, and insert them all into the related table in the database. We implemented APIs using Java. We made a wrapping class per DB table. List 1 shows the example of how to use the APIs. This simple example shows the checking code whether there has been the log-in from the ”root” user. Also, we implemented

Listing 1: Example code of checking the login with ”root”

```
boolean isThereRootLogin(DBManager db)
{
  // select 'root' user from User table
  User rootUser = User.loadbyUsername(db, "root");
  if (rootUser != null){
    // get all history where user name is 'root'
    Vector<ConnectionHistory> historyList =
        ConnectionHistory.loadByUser(db, rootUser)
        ;

    for (ConnectionHistory history : historyList){
      if(history.isSuccessfulLogin == true){
          // found!
        return true;
      }
    }
  }
  return false;
}
```

the web crawler using Python which gathers detailed IP information when given an IP address from the site, "http://www.iplocation.net", and insert detail information of IPs such as country, city and coordinates.

3.3 Rule-based detector

The rule-based detector is implemented using DB APIs, and it executes all checking routine for defined rules and shows their results. In our implementation, each rule is represented as a Java class that implements Rule Java interface. The rule-based detector can be easily extended in its functionality by creating new Rule class. List 2 shows Rule interface in Java. Only two methods are needed to be implemented and run method is the core method that gets DBManager as an argument which is the interface of DB APIs and used to access DB query. Currently, we provide the two rules: one is to check concurrent access from the different locations, and another is to check successful login from suspicious IP addresses.

3.4 Behavioral-based detector

For behavioral-based detector, we defined five features per user: average working time per day, login location, login times, failure numbers and login method. The login method just notes if they log in using a key or a password. The behavioral-based detector extracts these five features from the database and checks anomaly. The detector extracts the defined features which can be applied to different machine learn-

Listing 2: Rule interface

```java
interface Rule {
  String getRuleName();
  RuleResult[] run(DBManager db)
  throws SQLException;
}

enum RuleSeverity {
  NORMAL, WARN, CRITICAL
}
class RuleResult {
  RuleSeverity severity;
  String message="";
}
```

ing technique. Currently, we are still trying and experimenting which method are best in our detector.

One of the biggest challenges is to come up with evaluation methodology of our behavioral-based detection. It is very hard to get labeled SSH logs that show which accesses are malicious. To deal with this challenge, we came up with the idea to evaluate techniques, though we have not finished the module of evaluation. This part is our future work. Our idea is to mutate original log files, add some malicious logs randomly and check the result of our detector whether it can detect them. For this, we need to implement additional modules, Log Mutator, which mutates the original logs and generates a mutated log file with an expected result. Then, the anomaly detector will run with the mutated log file. The verification will be done by comparing the detection report from the mutated log and the expected result generated by Log Mutator. If the verification result is not successful, we will look through the result, and enhance the detector algorithms. This iteration will be repeated until our system will pass this verification step. Due to the nature fo the problem we are leaning towards unsupervised methods to train our method.

4 Experiments

4.1 Evaluation data

We proceed to an evaluation of our framework using real SSH logs. The initial results were obtained from two SSH logs from production machines spanning three months.

4.2 Analysis reporter

We run the analysis reporter about two server log data. The analysis reporter gives statistics about users, access IP addresses, and attack events by correlating DB tables. Table 2 shows the general statistics about the SSH logs. It can be seen from the table that there has been a lot of authentication failures. On average, more than 1,000 failures happen every day in the both servers. Especially, three-quarters of log-in tries were failed in Server 1. Also, the table shows that many

Name	Size	Total Messages	Period
Server1.log	169 Mb	1,325,683	3 months
Server2.log	433 Mb	3,531,592	3 months

Table 1: information of SSH log files used in evaluation

Type	Server1.log	Server2.log
Total login attempts	134,640	350,168
Login success	37,212 (27 %)	199,496 (56%)
Total user number	3,213	91,810
Valid users	79 (2.5%)	274 (0.3%)
Total IP number	1,818	2,996
login IPs	236 (13%)	1,104 (37 %)

Table 2: General SSH statistics

different usernames have been used in log-in attempts. Totally, 91,810 different user names are identified from Server 2, and only 0.3 % of them are presumed to be valid users.

More detail information about attacks is shown in Table 3. There has been 494 and 621 attack sessions in Server 1 and Server 2, respectively. On average, one attack session consists of 193 and 611 connection tries in Server 1 and Server 2. Noticeably, the longest attack session in Server 2 happened for 10 hours and throughout the term, it tried 50,000 connections. It is also interesting that 6 % of running time on Server 2 has been under the attack.

Finally, Figure 4 and Figure 5 respectively show the top 10 user names and countries that are involved in the attacks. Table 4 shows privileged accounts (root, admin) are most targeted by the attack and possible accounts (test, guest) are also the big targets. It can be seen that platform dependent users names (ubnt, oracle, www-data) are also attractive targets. The two server's top 10 user lists are similar except root user. The majority of attacks tried with root user on Server 1, but there are a few of tries with root user in Server 2. We cannot figure

Type	Server1.log	Server2.log
Total time	92d 20h	92d 20h
Time on attack	1d 23h (2 %)	5d 17h (6%)
Total attack sessions	494	621
Total attack connections	95,464 (51%)	379,597 (59%)
Avg. tries per attack	193	611
Max tries per attack	10,848	50,202
Avg. duration per attack	5m	13m
Max duration per attack	5h 47m	10h 7m

Table 3: SSH attack statistics

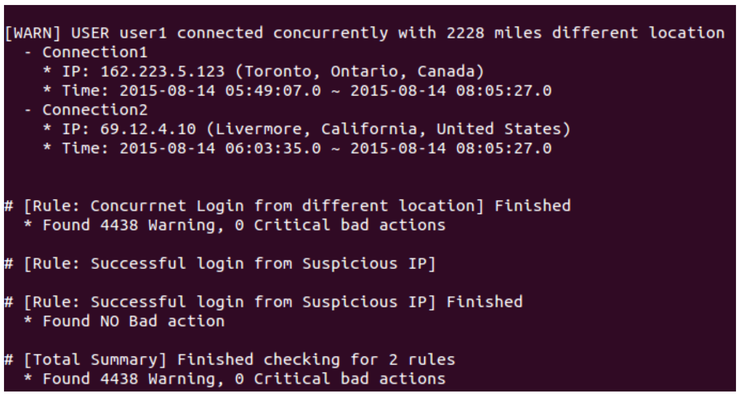
Server1.log		Server2.log	
User name	Attempts number	User name	Attempts number
root	53,564 (39.8 %)	admin	4205 (1.2 %)
admin	2,711 (2 %)	test	4198 (1.19 %)
test	1,476 (2 %)	guest	2297 (0.65 %)
ubnt	1,162 (0.86 %)	zabbix	1697 (0.48 %)
guest	1,096 (0.81%)	ubnt	1333 (0.38 %)
nagios	611 (0.45%)	b56girard@gmail.com	1262 (0.36 %)
pi	599 (0.44%)	user	906 (0.25 %)
user	556 (0.41%)	www-data	773 (0.22 %)
oracle	511 (0.37%)	oracle	717 (0.2 %)
zabbix	466 (0.34%)	zxin10	692 (0.19 %)

Table 4: Top 10 user names on the attack

Server1.log		Server2.log	
Country	Attempts number	Country	Attempts number
China	423 (23.5 %)	U.S.	800 (26.8 %)
U.S.	366 (20 %)	China	548 (18.4 %)
Brazil	123 (6.8 %)	France	162 (5.4 %)
Russia	90 (5 %)	Canada	151(5 %)
India	85 (4.7%)	Germany	141 (4.7 %)
Canada	56 (3.1%)	Brazil	135 (4.5 %)
South Korea	43 (2.4%)	Russia	106 (3.55 %)
Germany	40 (2.2%)	India	92 (3 %)
Italy	39 (2.16%)	U.K.	78 (2.61 %)
Seychelles	38 (2.11%)	Hong Kong	67 (2.2 %)

Table 5: Top 10 country of the attack origin

out this reason, but it is plausible that Server 2 has some feature which prevents access from the root user. The most top 2 countries where attacks originated from are China and U.S. on the both servers. Their percentages heavily outnumber ones of other eight countries. Seven countries among the top 10 are listed in the both servers. There is an unusual country in Server 1: Seychelles which is an island nation located in the Indian Ocean.



```
[WARN] USER user1 connected concurrently with 2228 miles different location
  - Connection1
    * IP: 162.223.5.123 (Toronto, Ontario, Canada)
    * Time: 2015-08-14 05:49:07.0 ~ 2015-08-14 08:05:27.0
  - Connection2
    * IP: 69.12.4.10 (Livermore, California, United States)
    * Time: 2015-08-14 06:03:35.0 ~ 2015-08-14 08:05:27.0


# [Rule: Concurrnet Login from different location] Finished
  * Found 4438 Warning, 0 Critical bad actions

# [Rule: Successful login from Suspicious IP]

# [Rule: Successful login from Suspicious IP] Finished
  * Found NO Bad action

# [Total Summary] Finished checking for 2 rules
  * Found 4438 Warning, 0 Critical bad actions
```

Figure 6: Example of the result of rule-based detector

4.3 Rule-based detector

We run the rule-based detector for the two log files. Currently, we provide two rules. Figure 6 shows the result of running the rule-based detector. There is no critical behavior, but several warnings. There is no rule violation that a successful log-in happened from suspicious IPs. However, for some users, there were concurrent accesses from the different locations. Since there is some inaccuracy in measuring exact location from IP, we set 500 miles of distance to the threshold for different locations. Looking though these warnings manually, we noticed that this account has logged into the server from two IP addresses at the same time repeatedly. Also, it happened thousand times during the specific period. This is possible if two people share the account or the user of this account was doing some experiment. Otherwise, the account must be compromised.

4.4 Behavioral-based detector

4.4.1 Extracting features

Figure 7 shows the graph which is drawn with the features extracted from the behavioral-based detector. Those features are from the same user who gets warning messages from the rule-based detector in Figure 6. It can be observed from the graph that anomaly is concentrated during the specific term. This period is the same as the period where the rule-based detector gives the alarms. This cannot be verified without manual intervention and audit of the user whose usage showed this behavior . However, this result shows that there has some event during this period that leads to change their working pattern, location. Especially, it is suspicious because during the time there was a huge

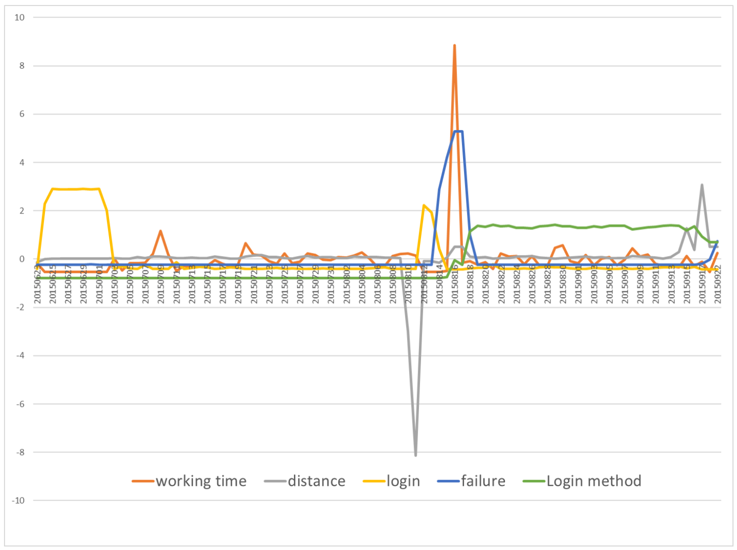


Figure 7: Feature extraction from behavioural-based detector

increase in authentication failures.

4.4.2 Principal component analysis

In the context of detecting anomalies in log files, PCA can help to filter frequently repeating patterns in the data which makes it easier to detect anomalies. We try to see the variance of the data we have by trying Principal Component Analysis on it. The features we use for this are date, log in time, user, IP address logged from. The log in time is always increasing and will be mostly bulked messages into similar groups if most people try to log in at the same time. We expected to get the variance from the data to detect outliers in our data. We ran PCA on the whole data set with 284 unique users. The result we got pointed us the outliers in our data point which are probable anomalies in the dataset. [Figure 8.a]

However, this does not give us much insight into what is wrong. We can select the outlier and point it out to the anomalous log, but it doesn't tell us why it is an anomaly. Also, we realized that in the log data we have, the user distribution is not even. Some users have a lot of activity and some very few. So a user specific activity profiling may give us a much better outlier. Which we tried on a specific user with the highest number of activity and got the following activity[Figure 8.b]

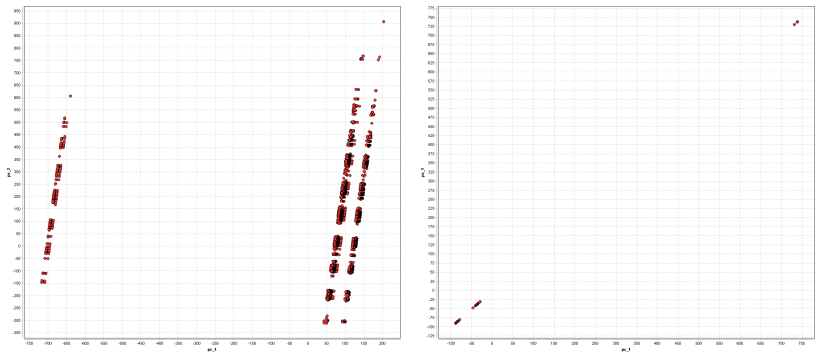


Figure 8: (a) and (b)

4.4.3 K-Means clustering

The algorithm tries to find groups of data instances with similar size and low variance different types of graphs from it to see what other data we can get. For our case [2] we take the feature sets of Start Time, End Time, User, Logged in Duration as our features and try to map out outliers. In this case the X axis denotes each user and the Y axis denotes the duration of his activity. We can see that from this graph there are two points in the log which have negative duration. Which is not possible. That helped us find erroneous log entries and entries which our parser had parsed incorrectly. we also got different types of graphs from it to see what other data we can get. This encouraged us to experiment more with this approach with another set of vectors with user,time duration and IP logged in from.

5 Related Work

There are three categories which are related to our work: anomaly detection, SSH attack and defense, and effective log analysis.

Anomaly detection. Firstly, anomaly detection has been an active research field for a long time. Since first proposed in [6], a variety of research have been conducted in finding anomaly detection, and there have also been many survey papers [2, 3, 10, 14, 1]. The most vibrant research trend in anomaly detection is to apply machine learning technique to identify anomalies. There are many proposed techniques which could identify attacks: Artificial Neural Network [4] and RIPPER, k-nearest neighbor (KNN) [29], Support Vector Machine, Bayesian Algorithm (ISABA) and fuzzy logic [25] [8]. In the work [28], they propose an autonomic way for tuning the SSH parameters of anomaly-based intrusion detection systems. The research [26] developed two mechanisms to achieve a fast intrusion detection system by using output weight optimization hidden weight optimization (OWO-HWO) training algorithm, and a feature relevance analysis. In the work [7], they proposed using a sliding window approach in combination with a signature based approach in order to detect anomalies by comparing previously clustered data. In the work [15], they proposed feature-based anomaly detection using histogram-based traffic information.

SSH attack. SSH attacks have become a major security concern because of its danger associated with a successful compromise[11]. Therefore, there have been many studies related to SSH attacks and defenses. In the works [21, 22, 24], they showed the threat of SSH attacks and how they are common nowadays. Also, there has been many efforts to enhance SSH security. The one of major research is to detect SSH attacks in advance. There are two types of studies. In [11], they propose effective detection of SSH attacks by installing modified SSH client called SSHCure. Also, in [13], they provide to find compromise detection by analyzing network flow. The both of them are to aim to find SSH attack in real time.

Analysis based on log files. There are also papers which provide the way of detecting anomaly based on log files or helping log analysis easier. Firstly, Adrian Frei and Marc Rennhard [9] presented an approach for detecting and visualizing anomalies in a log file using word count to create a single predictor attribute which they use to create a histogram for each log line every hour. In [20], the authors talk about some of the most common application of log analysis, problems in using statistical methods in log analysis and some methods people use to analyze logs. In the work [30], they propose a method which tackles the issues of clustering of extremely large log files and this work can be combined with HMAT by [9] to build a new approach to anomaly detection. Wei Xu [31] proposes a method in their paper which first creates composite features by analyzing console logs and then use these features in machine learning to detect operational anomalies in the log. Finally, in work [16, 23], the authors provide the effective way of log file analysis and the technique of correlating events in the log. Also, using these techniques, they suggest the effective way of detecting system problems without massive manual labor.

Although there is much-related work, we could not find the research that has the same goal as ours. Also, we could notice two interesting points. Firstly, most of the studies are done based on machine learning techniques, but there are two limitations using machine learning in the area. Firstly, even though there are a lot of anomaly research using machine learning, they have rarely been used in the practical field [27]. The authors of the work [27] claim that machine learning techniques seem to effective in anomaly detection, but it is very hard to apply to the real fields. This is because many kinds of research work in its limited environment and have very high costs of classification errors. Therefore, there is a certain semantic gap between research results and their operational interpretation. In addition to this problem, many studies use KDD Cup 1999 Data [12] in the experiment to evaluate machine learning schemes. In practice, this dataset is decade old and has many criticisms [17, 18] for current research. Because of those reasons, many applications of anomaly detection using machine learning are not applicable in the real field.

6 Future Work

6.1 Improving behavioral-based detector

Until now, we have defined features, extracted them from the raw log files and get some visual data to show patterns. However, still we haven't found which method is best in detecting an anomaly. This work should be done together with the evaluation module. After completing evaluation module, we could do more experiments with different machine learning algorithms and find the best solution. Also, we need to come up with more features from the SSH log which enhance our detector. Therefore, we plan to apply clustering or mining technique to our framework such as Time series clustering and Frequent itemset mining.

6.2 Generalizing parser

SSH servers in different platform give different types of log messages. Therefore, we need to generalize our parser. Also, we found our parser sometimes relates messages from different streams into the one stream. This is due to the ambiguity from incompleteness in SSH log files. Since there are a few cases, it still causes to reduce accuracy in our detection. To solve this, we need to find the better heuristic way by checking every incorrect case and customizing routines for them.

6.3 Experimenting with bigger data

We experimented with the big size of SSH log files, but the period of them is short (3 months). We need to experiment our analyzer with

SSH logs with longer term, which could give us more insight.

6.4 Extracting Features

Apart from 5 features we use, we need to come up with more feature. We can use the feature creation method for our future work. For example, the log messages will be divided into two categories. Message identifiers and state variables. During normal execution, the relative frequency of each value of a state variable in a time window usually stays the same. For example, the ratio (of appearance in a defined time interval) between the connection from and disconnection time is stable during normal behavior, and it should change in any case of the anomaly. We formalize this relationship regarding State ratio Vector (Ys) and exploit them in our framework.

7 Conclusion

We propose a general approach to analyzing log files and present a practical SSH log analyzer that gives useful analysis report from SSH log events and detects compromise. To implement the detector, we have analyzed SSH log files and identifies log patterns that are necessary to extract log streams. From this information, we have implemented the parser, database modules, and detection/report engines. Log data are parsed and put into the database to make further analysis effective. We implemented three back-end modules: Log reporter, Rule-based detector, and Behavioral-based detector. Using our framework, we have analyzed big SSH logs and evaluated our framework. From the experiment, we show that our framework provides useful analysis report which helps us understand the SSH attacks more. Also, our detectors have yielded some warning messages after analyzing. Fortunately, there was no critical level of messages, but some warning messages are still suspicious because the period of the violation coincides with the time when the behavioral pattern indicates anomaly such as the sharp increase in authentication failures. Our solution is practical since it is applicable only using SSH log files. Soon, if we enhance the behavioral-based detector, it can help people analyze SSH logs automatically.

References

[1] F. Bezerra and J. Wainer, "Algorithms for anomaly detection of traces in logs of process aware information systems," *Information Systems*, vol. 38, pp. 33–44, 2013.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Outlier detection: A survey," *ACM Computing Surveys*, 2007.

[3] V. Chandola, A. Banerjee, and V.Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[4] W.-H. Chen, S.-H. Hsu, and H.-P. Shen, "Application of svm and ann for intrusion detection," *Computers & Operations Research*, vol. 32, no. 10, pp. 2617–2634, 2005.

[5] Cisco, "Observations of login activity in an ssh honeypot," Cisco Security Intelligence Operations, Tech. Rep., 2009.

[6] D. E. Denning, "An intrusion-detection model," *Software Engineering, IEEE Transactions on*, no. 2, pp. 222–232, 1987.

[7] M. Esmaeili and A. Almadan, "Article: Stream data mining and anomaly detection," *International Journal of Computer Applications*, vol. 34, no. 9, pp. 39–42, Nov 2011.

[8] D. M. Farid, M. Z. Rahman, and C. M. Rahman, "Article: Adaptive intrusion detection based on boosting and nave bayesian classifier," *International Journal of Computer Applications*, vol. 24, no. 3, pp. 12–19, Jun 2011.

[9] A. Frei and M. Rennhard, "Histogram matrix: Log file visualization for anomaly detection," in *ARES 2008 - 3rd International Conference on Availability, Security, and Reliability, Proceedings*, 2008, pp. 610–617.

[10] P. Gogoi, D. Bhattacharyya, B. Borah, and J. K. Kalita, "A survey of outlier detection methods in network anomaly identification," *The Computer Journal*, 2011.

[11] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, and A. Pras, "Sshcure: A flow-based ssh intrusion detection system," in *Dependable Networks and Services.* Springer, 2012, pp. 86–97.

[12] S. Hettich and S. Bay, "The uci kdd archive [http://kdd. ics. uci. edu]. irvine, ca: University of california," *Department of Information and Computer Science*, p. 152, 1999.

[13] R. Hofstede, L. Hendriks, A. Sperotto, and A. Pras, "Ssh compromise detection using netflow/ipfix," *ACM SIGCOMM computer communication review*, vol. 44, no. 5, pp. 20–26, 2014.

[14] V. Jyothsna, V. V. R. Prasad, and K. M. Prasad, "A review of anomaly based intrusion detection systems," *International Journal of Computer Applications*, vol. 28, pp. 26–35, 2011.

[15] A. Kind, M. P. Stoecklin, and X. Dimitropoulos, "Histogram-based traffic anomaly detection," *IEEE Transactions on Network Service Management*, vol. 6, pp. 110–121, 2009.

[16] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection." in *USENIX Annual Technical Conference*, 2010.

[17] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection," in *Recent Advances in Intrusion Detection.* Springer, 2003, pp. 220–237.

[18] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM transactions on Information and system Security*, vol. 3, no. 4, pp. 262–294, 2000.

[19] Y.-X. Meng, "The practice on using machine learning for network anomaly intrusion detection," in *2011 International Conference on Machine Learning and Cybernetics*, vol. 2, July 2011, pp. 576–581.

[20] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Communications of the ACM*, vol. 55, no. 2, 2012.

[21] J. P. Owens Jr, "A study of passwords and methods used in brute-force ssh attacks," Ph.D. dissertation, Clarkson University, 2008.

[22] D. Ramsbrock, R. Berthier, and M. Cukier, "Profiling attacker behavior following ssh compromises," in *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on.* IEEE, 2007, pp. 119–124.

[23] J. P. Rouillard, "Real-time log file analysis using the simple event correlator (sec)." in *LISA*, vol. 4, 2004, pp. 133–150.

[24] C. Seifert, "Analyzing malicious ssh login attempts," 2006.

[25] R. Shanmugavadivu and N. Nagarajan, "Network intrusion detection system using fuzzy logic," *Indian Journal of Computer Science and Engineering (IJCSE)*, vol. 2, no. 1, pp. 101–111, 2011.

[26] M. Sheikhan and A. A. Sha'bani, "Fast neural intrusion detection system based on hidden weight optimization algorithm and feature selection," *World Applied Sciences Journal, Special Issue of Computer & IT*, vol. 7, pp. 45–53, 2009.

[27] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," *2010 IEEE Symposium on Security and Privacy*, pp. 305–316, 2010.

[28] A. Sperotto, M. Mandjes, R. Sadre, P.-T. De Boer, and A. Pras, "Autonomic parameter tuning of anomaly-based idss: an ssh case study," *Network and Service Management, IEEE Transactions on*, vol. 9, no. 2, pp. 128–141, 2012.

[29] M. Su, "Real-time anomaly detection systems for denial-of-service attacks by weighted k-nearest-neighbor classifiers," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 3492–3498, 2011.

[30] N. Taerat, J. Brandt, A. Gentile, M. Wong, and C. Leangsuksun, "Baler: Deterministic, lossless log message clustering tool," *Computer Science - Research and Development*, vol. 42, pp. 285–295, 2011.

[31] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, vol. 10, p. 117, 2009.

[32] M. J. Zaki, "Spade: An efficient algorithm for mining frequent sequences," *Machine Learning*, vol. 42, pp. 31–60, 2001.