RACOON: Rapidly Generating User Command Data For Anomaly Detection From Customizable Templates

Ramkumar Chinchani Aarthie Muthukrishnan Madhusudhanan Chandrasekaran
Shambhu Upadhyaya
State University of New York at Buffalo
Buffalo, NY 14260
Email: {rc27, am52, mc79, shambhu}@cse.buffalo.edu

Abstract

One of the biggest obstacles faced by user command based anomaly detection techniques is the paucity of data. Gathering command data is a slow process often spanning months or years. In this paper, we propose an approach for data generation based on customizable templates, where each template represents a particular user profile. These templates can either be user-defined or created from known data sets. We have developed an automated tool called RACOON, which rapidly generates large amounts of user command data from a given template. We demonstrate that our technique can produce realistic data by showing that it passes several statistical similarity tests with real data. Our approach offers significant advantages over passive data collection in terms of being non-intrusive and enabling rapid generation of site-specific data. Finally, we report the benchmark results of some well-known algorithms against an original data set and a generated data set.

1 Motivation

Anomaly detection techniques have been applied at various levels such as network packets, system calls and user commands. Network level anomaly detection [20] attempts to detect network attacks based on packet patterns. Program behavior modeling approaches [4, 5] establish baseline behavior by observing the system call sequences and any malicious activity is captured through deviations from this baseline behavior. User profiling techniques study user behaviors through commands [11, 23, 15] that are issued by a user and these are used to make a distinction between legitimate users and masqueraders. An ample and compatible test data set is central to an in-depth evaluation of any anomaly detection algorithm. The nature of data used by network, program and user profiling based anomaly detection techniques is different, and consequently, so are the data availability and collection issues. We first compare and contrast some of these issues to provide an insight into the motivation of our work. Table 1 presents a summary of the issues and challenges faced.

IDS research has traditionally focused on network level intrusion detection and this is reflected in the number and variety of network level security tools available. One of the first evaluations of intrusion detection systems was performed using network audit data from DARPA/MIT Lincoln Labs [13, 12]. This data was collected over a few weeks by deploying a sniffer, and simulating normal and attack traffic over a testbed network. It was realized that although extensive as the data sets were, they were still limited, and they could not be deemed to cover all possible scenarios and network environments. On the other hand, evaluation of an IDS in an actual environment could cause problems in terms of being intrusive to the users on the network. For example, honeypots first require legal clearance and user cooperation before they can be deployed in a network. In view of these issues, MIT Lincoln Labs has developed a next generation IDS evaluation framework called LARIAT[21], wherein highly customizable test scenarios can be played with ease. Some of the recent research efforts have also focused on program execution based anomaly detection. Data availability and collection are not serious issues in this context. This is mainly because every program can be executed in isolation and a wealth of data can be generated simply by varying the inputs. Moreover, program behavior can be easily replicated as long as the versions and the inputs are identical.

User level intrusion detection deals with user command data, which is closely tied to user behaviors. The process of collecting or generating data suffers from similar issues as network level data if not worse. Obtaining a representative data set for each user takes months or some times years. Lane et al. reported [8] that their data collection pro-

	Network	Program	User
Nature of Data	Network packets	System call traces	User commands
Collection process	Sniffer deployment	Program execution	Command monitoring
Duration of collection	Days	Seconds or minutes	Months or years
Intrusive To Users	Possibly	No	Yes
Data generation tools	Yes	Not required	No

Table 1. A summary of data availability and collection issues of different categories of anomaly detection techniques

cess took nearly two years for only eight users. Not all of this data could be used because of errors and requirement of anonymization. As a result, the data set had to be sanitized. In general, the data collection process involves setting up a command monitoring tool on the user's computer and this could be looked upon as being intrusive to the user. Also, it doesn't guarantee good data because the user's behavior may alter if he is aware of the fact that his commands are being monitored. The closest research effort we have found that develops synthetic data based on user profiles is the effort by Lundin et al. [1] although their application domain was not intrusion detection per se.

To summarize, there are simulation tools available in the networking domain such as OPNET [18] and ns2 [17] and data generation tools such as LARIAT to aid researchers, and none are available for user command data. In view of these issues, we propose a user command data generation tool called RACOON to expedite the process of development and evaluation of user level intrusion detection. The two main aspects of our approach are:

- *Job-centric approach*

  RACOON works on the assumption that a user's computational behavior is a causal process indicated by the commands he uses to accomplish a job or task. Notable works such as that by Lane et al. [10] have used this assumption as the basis of most of their work; we just restate it here. There could be different classes of users such as programmers, scientists and system administrators, each with different job preferences and peculiarities in user command usage. In our approach, we use the notion of jobs as a second order description of user behavior; the first order being the commands itself. We first describe this model to capture user behavior and then show that data generated using this model is very similar to an actual user command data set.

- *Customizable templates*

  One of the important aspects of any data generation tool is the support for "tunability" of data. Data of varying quality not otherwise seen in the wild can be generated, and used to evaluate and expose the blind spots of an IDS. A template in RACOON represents a particular user profile. It contains the parameters used to replicate user behavior. Since these parameters are user-controlled, it is possible to customize each template to reflect a particular user profile. Subsequently, one can generate data which is pertinent to a particular computational environment and with required noise levels.

1.1 An Overview of RACOON

Figure 1 shows the overview of RACOON. There are two paths for generating user command data. In the first path, a user specified template is created and provided as input to the data generation module. Manual specification can be a onerous process and we have implemented a front-end to assist the user. In the second path, an available data set can be processed to create the template, which then follows the first path. Several user-controlled parameters such as data size allow the generation of data of desired size and quality. In order to evaluate our tool, we have generated data sets resembling Schonlau's data set (750,000 commands) [22] in a matter of a few seconds, and performed statistical similarity tests between the two data sets - generated data and Schonlau's data. Finally, we have also duplicated some well-known anomaly detection algorithms and benchmarked the two data sets.

2 Related Work

Our work finds great relevance in techniques which use user command data to perform anomaly detection. So first we mention some well-known works in this area and the progress that continues to be made towards better detection algorithms. Then, we describe some of the publicly available user command data sets, which the current anomaly detection systems have relied on. Finally, we discuss our work in the context of data generation and simulation tools available both in academic and commercial domains.

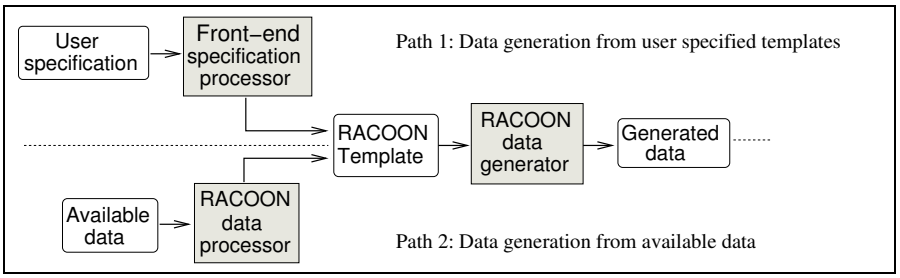


Figure 1. Overview of RACOON's data generation process

2.1 User Command Based Anomaly Detection

Lane et al. [9, 10] emphasized that a user's behavior is causal in nature and used a sequence matching algorithm to capture user actions. In a follow up work, they showed that a Hidden Markov Model [7] is a good approximation of user command behavior. In RACOON, we also use the fundamental Markov model to capture temporal properties. Schonlau et al. [23] compared several statistical techniques ("Uniqueness", "Bayes one-step Markov", "Hybrid multistep Markov", "Compression", "IPAM" and "Sequence-Match") to evaluate their effectiveness in masquerade detection. Maxion et al. [15] used the Naive Bayes Classifier on a data set containing truncated user commands and showed that their technique improved detection significantly with very low false positive rate. Later, they argued [14] that valuable information was lost when truncated command line data was used. They showed that by using enriched command data, the detection rates were further improved. A recent research effort [24] used data mining techniques to detect masqueraders. We have chosen some of these algorithms to benchmark the data that is generated through RACOON. Also, we would like to point out that there is only rudimentary support for generating command line arguments in RACOON. However, considering that Maxion et al.'s effort showed that enriched command lines provide more information, we are currently working on enhancing RACOON to support this feature in a more meaningful manner.

2.2 User Command Data Sets

There are three publicly available user command data sets - Greenberg data set (GDS) [6], Purdue's MILLENIUM data set (PMDS) [8] and Schonlau data set (SDS) [22]. Table 2 is a brief survey of research efforts and the data sets used for their empirical studies.

The Greenberg data set consists of user commands collected from 168 users. The command format is very rich, consisting of the command along with command line arguments, aliases if any, current working directory and whether the command execution was a success or failure. The users came from four user groups categorized as novice programmers, experienced programmers, computer scientists and non-programmers. Purdue MILLENIUM data was collected by Lane et al. over a period of up to two years for eight users. The data set consists of 15,000 to 100,000 commands. The Schonlau data set consists of command sets belonging to 50 users, where each command set contains 15,000 truncated command lines. The Schonlau data set has been the most popular data set among the research community. Therefore, for our evaluation purposes, we have also used this data set. We agree that this data set is flawed by nature as it contains several invalid or obsolete commands. However, choosing this data set allows us to compare our benchmark results with known results in the user level intrusion detection community.

Research Effort	Data Set Used
Lane et al.	PMDS
Schonlau et al.	SDS
Maxion et al.	SDS, GDS
Stolfo et al.	SDS

Table 2. Recent user command based anomaly detection research efforts and data sets used

2.3 Simulation and Data Generation Tools

With extensive research efforts being invested in the networking domain over the last four decades, there have been significant advances in technology and large scale communication networks is already a reality. On the downside, the sheer size of modern networks makes it impossible to test realistic deployments. Instead, several simulation testbeds such as OPNET and ns2 which serve as a substitute for the real-world scenarios. In the context of intrusion detection systems, the situation has been very similar. Following the

lessons learnt from the data generated during the 1998/1999 DARPA evaluation, it was soon realized that a testbed for intrusion detection systems was necessary to conduct large scale evaluations and experiments. LARIAT was one such tool that was developed, which allows highly customizable scenarios to be played over a test network. Some of the driving directions of RACOON have been inspired by tools like LARIAT. Debar et al. proposed and implemented an experimentation workbench [2] for intrusion detection systems. The workbench used a combination of techniques to simulate user interaction with various network services. One important research effort which is very similar to our work is Lundin et al. [1]. Their data generation methodology closely mirrors Path II of RACOON. The audit data which was used to seed the data generation process consisted of user actions and their side-effects. However, their application domain was fraud detection. In their future work section, they do conjecture that some of their techniques could be used for synthetic data generation for intrusion detection. RACOON could be considered as a positive proof-of-concept of that conjecture.

2.4 Known Critiques of Simulated Data

McHugh et al. [16] voiced some concerns over the DARPA offline intrusion detection evaluation experiments using data from MIT Lincoln Labs. Like all statistical data, there could be inherent biases which could skew the results obtained. Although noise was introduced in the data, it followed a known model at best. This may be a significant departure from the noise seen in the wild. In order to circumvent the privacy and sensitivity issues of real intrusion data, the evaluation was performed with synthetic data. This data set is very limited and its authenticity is questionable. Too much information about the evaluation process has been left out which prevents researchers from replicating the experiment. In the context of our data generation process, some of these concerns are very relevant and we try to address them. First, our tool is primarily intended to solve the problem of paucity of user command data sets. Using our tool, researchers can create a variety of data sets and perform evaluations. Privacy of user command data is definitely a serious issue and our tool alleviates that problem. Finally, we agree that RACOON like other simulation tools cannot generate real data, but rather acts as a good substitute.

3 Job-Centric Approach

User behavior is a causal process, and the key to our technique is identifying some key features which can be used to approximately reproduce that behavior. We describe these features and their role in the data generation process.

3.1 Preliminaries

Command. The basic unit of execution by a user is a *command*. Typical user behavior on a computer is a causal process, where a user attempts to achieve a certain goal by issuing relevant commands. Examples of a command are `vi` and `emacs`.

Meta-command. A higher level description of a command is called a *meta-command*. Commands producing the same effect are grouped together and such groupings form meta-commands. For example, the commands `vi` and `emacs` are both used for editing a file, and their corresponding meta-command is `editor`. The terms command and meta-command can be used interchangeably for all practical purposes, since a meta-command is merely a place-holder for an actual command. We use this notion of a meta-command during the data generation process.

Job. A user executes meta-commands to accomplish some system *job*. The goal that we speak of can be characterized as a job that a user wishes to accomplish. In general, no single meta-command accomplishes everything a user requires and a combination of them must be used. For example, a program development job requires the use of meta-commands such as `editor`, `compiler`, `linker` and perhaps a `debugger`.

Meta-command Sequence. A job is accomplished by a *meta-command sequence*. Meta-commands belonging to a job are typically executed in a sequence and the same sequence occurs whenever that particular job is being accomplished. Hence, there exists sets of meta-commands which demonstrate strong temporal and spatial properties. Referring to the previous example of the program development job, an executable binary cannot be produced without the use of an `editor`, `compiler` and a `linker` in some definite sequence.

Job Hopping. Intermittent switching between jobs is called *job hopping*. Realistically, a user may accomplish multiple jobs and he spends a finite time on each job before switching to another.

Session Scope. The set of jobs which a user may accomplish is called a *session scope*.

Figure 2 shows the hierarchical relationship between commands, meta-commands, jobs and session scope. To summarize, commands that perform the same kind of task are grouped together as a meta-command, meta-commands (or commands) can be combined to perform a job, and one or more jobs form a user's session scope.

3.2 Simulating A User Profile

The hierarchical structure in Figure 2 only captures the semantics of a user's activity. The scope of a user's session combined with the statistical properties of jobs forms

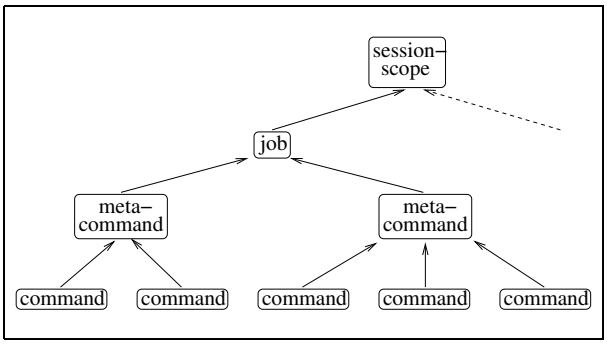


Figure 2. Hierarchical relationship between commands, meta-commands, jobs and session scope

.

the user's profile. We define the following statistical properties (shown in Figure 3), which along with the semantic structure forms a RACOON template.

- A meta-command is a set containing 2-tuples describing command instantiation. It is defined as $M = \{(c, p) \mid c \text{ is a command } \wedge\ p = \Pr(c\,|M)\}$. For each M, the sum of probabilities p is 1. For example, $\texttt{editor} = \{(\texttt{vi}, 0.5), (\texttt{emacs}, 0.5)\}$.

- A job is essentially a sequence of meta-commands. It is defined in terms of the transitions and sequence length, both being probabilistic. The meta-command sequence is modeled as a Markov chain and the sequence length is captured using a Gaussian distribution. More formally, a job is defined as $J = (S_M, P_M, Q_M, N(\mu, \sigma), n)$, where S_M is the set of meta-commands, P_M is the transition probability matrix, Q_M is the initial probability vector, the sequence length is a normally distributed random variable N with mean μ and standard deviation σ, and n is the total number of commands associated with the job.

- Finally, at the highest level, we have the session scope. A session scope can be viewed as a sequence of jobs in conformity with the phenomenon of job hopping. Job hopping is also modeled as a Markov chain. It is defined as $S = (S_J, P_J, Q_J)$, where S_J is the set of jobs, P_J is the transition probability matrix and Q_J is the initial probability vector. Note that the diagonal of the matrix P_J is empty because a job hopping from a job to itself collapses to the same job.

4 RACOON Template

Templates are central to RACOON's data generation process. They can either be specified manually or created from an available data set. Each path has its pros and cons. Site-specific data can be generated from manually created templates and this data would provide a more meaningful evaluation rather than using known data sets. However, template specification could be a very tedious process. On the other hand, template generation from known data sets is a completely automatic process and subsequently certain parameters of the template can be tweaked to generate different versions of the data set, but the data that is generated still reflects the environment where the original data set was collected. We now describe both paths.

4.1 Path 1: Manual Template Creation

A RACOON template represents the personality of a particular user. We have used XML for specification of semantic and statistical information. The manual template creation process uses a top-down approach. We start with the session scope definition. First, the jobs that a user wishes to accomplish are identified. Depending on how much a user focuses his actions on each job, the initial and job-to-job hopping probabilities are specified. Each job is characterized by a set of meta-commands and their sequences. Parameters of each job are specified in terms of meta-commands and their transition probabilities. A single meta-command represents a small task and a user may have a preference in terms of the commands he chooses to accomplish this task. This is captured as command instantiation probabilities. At this point, the session scope specification is complete for a particular user. Since, each parameter is user-controlled, it results in highly customized templates. However, manual template creation is a very cumbersome data-entry process, and we hide this behind a front-end.

4.2 Path 2: Template Creation From a Data Set

Alternatively, a RACOON template can be created automatically given a data set containing user commands. Perturbations can be introduced in these templates to create different user profiles. We now describe an algorithm called RACOON-TEMPLATE-GEN (see Table 3), which reads a stream of user command data and generates the template.

RACOON-TEMPLATE-GEN tries to identify jobs and their corresponding commands on the basis that commands belonging to a job show very strong cohesion to that job. It takes a command stream and calculates the initial and transition probabilities among commands assuming a Markov model, where each command is a state/node. High transition probabilities in excess of a pre-specified threshold

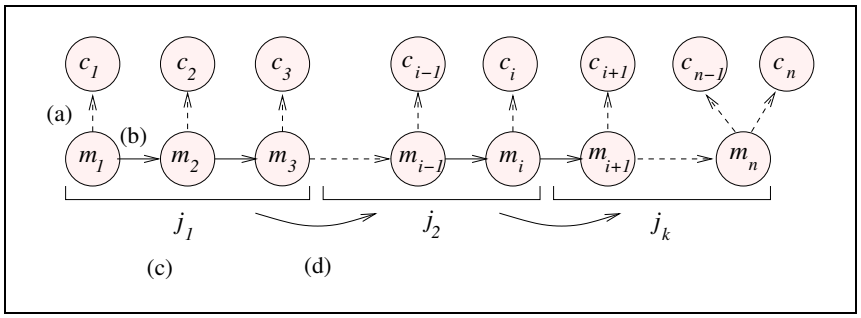


Figure 3. Statistical properties of interest - (a) command instantiation, (b) meta-command transitions, (c) meta-command sequence length, and (d) job transitions

RACOON-TEMPLATE-GEN$(D, size)$

```
 1  I[size] ← zero matrix
 2  T[size, size] ← zero matrix
 3  while data in D
 4  do For each command occurrence c_i, increment I[c_i]
 5     For each command transition c_i → c_{i+1}, increment T[c_i, c_{i+1}]
 6   Convert frequencies to probabilities in I and T
 7  A[size, size] ← zero matrix
 8  for each T[i, j]
 9  do if T[i, j] >= α
10        then A[i, j] ← A[j, i] ← 1
11  J ← φ
12  for each unvisited non-zero entry A[i, j]
13  do list ← DEPTH-FIRST-SEARCH(A[i, j])
14     For S_M, associate a unique meta-command to each command
15     Calculate P_M and Q_M from T and I respectively
16     Calculate μ and δ from D
17     Calculate n from D
18     Append list to J For session-scope S, calculate P_J and Q_J from D and J
19   Convert S to a RACOON template
```

Table 3. RACOON template generation algorithm from a known data set

α between any two commands indicates a relationship between these commands. Based on this principle, we construct an adjacency graph, where an edge is assumed in both directions between the two corresponding nodes. At the end of this process, the adjacency matrix represents a forest of trees and each tree corresponds to a job. By running a depth first search algorithm on each unvisited entry in the adjacency matrix, we are able to isolate these trees or equivalently, the jobs. Once the jobs have been identified, the remaining parameters which completely describe a job are calculated by revisiting the command stream. At the end of the algorithm, all the information is converted into XML. XML was chosen only for convenience and the template could very well have been represented in another form.

5 Data Generation

Data is generated from a RACOON template by walking down the tree and applying the relevant statistics on the way. Roughly speaking, data generation is the inverse procedure of the template creation algorithm. However, it is not as involved as the template creation algorithm. The main algorithm (RACOON-DATA-GEN) for the data gener-

ation process is provided in Table 4 and a helper routine (EXPAND-META-COMMAND) is shown in Table 5.

A session scope specification is provided as the input to the RACOON-DATA-GEN. A job is first chosen using the initial probability vector and a job chunk size is generated using a normal distribution. These are fed into the EXPAND-META-COMMAND routine, which does the actual command data generation. These two routines are almost identical and main differences are seen in the data structures they operate on and the terminating conditions. RACOON-DATA-GEN operates at the job level, while EXPAND-META-COMMAND works at the meta-command level. The data generation process terminates once the number of generated commands reaches the value specified during the input stage.

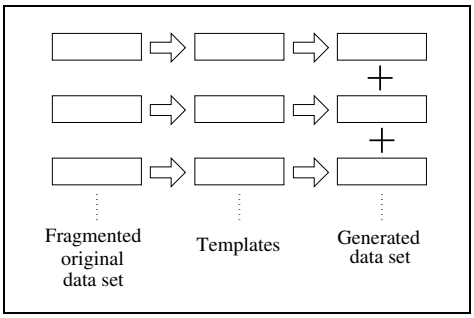


Figure 5. RACOON's data generation process modified to handle large input data sets

5.1 Data Set Discretization

RACOON's template generation process implicitly assumes time invariant behavior when computing the various statistics. For small original data sets from which templates are created, this is not a serious issue, and the generated data is similar to original data. However, user command data sets are seldom small, and an assumption of time invariance in the original data set will result in well-distributed and flattened statistics in the generated data set. In order to capture the localized statistical disturbances, we use a divide-and-conquer approach. Instead of creating one template from a large data set and then generating data from this single template, we break up the original data set into smaller segments and multiple templates are created from these fractional data sets. Eventually, piece-wise data generation increases the similarity between the original and generated data set. Figures 4 and 5 illustrate the different data generation strategies. For notational convenience, lets denote the fraction to be λ, where $0 < \lambda \leq 1$.

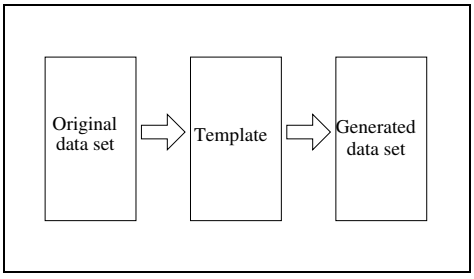


Figure 4. RACOON's original data generation process

We evaluated the aforementioned strategy against Schonlau's data set which is quite large with 15,000 commands for each user. We generated data with various fraction values of $\lambda = \{1, \frac{1}{3}, \frac{1}{15}, \frac{1}{30}\}$, which corresponds to data blocks of sizes 15,000, 5000, 1000 and 500 commands each respectively. We then performed several basic statistical similarity tests - 2 sample t-test (for equality of means), F-test (for equality variances), Levene test (equality of variances) and Mann-Whitney test (equality of medians), with a confidence interval of 99%. Each of these tests returns a p-value, where $0 \leq p \leq 1$. For strong similarity, the p-value should be at least 0.05, and greater the value, more similar the data sets. The effect of data discretization on a randomly chosen data set is shown in Figure 6.

As the value of $\lambda \rightarrow 0$, the individual block size $\rightarrow 1$. At this point, the data set is identically replicated. When $\lambda = 1$, the generated data set can be very different from the actual one depending on whether a user's command usage patterns are uniform over the entire data set or not. The choice of λ is important because it indirectly determines how much local noise should be retained in the generated data set. For our evaluation purposes, we have chosen a λ value of $\frac{1}{30}$. We chose this value because reducing it any further resulted in very small changes in the quality of data. So, we tried to choose a value that was as far away as possible from 0 and at the same time not compromise the quality of the generated data.

6 Evaluation

Ideally, we would like to evaluate both data generation paths of RACOON. However, since there is no mechanism through which open-ended data sets can be benchmarked, we evaluate only Path II using Schonlau's data set. For the evaluation process, we have used various metrics, and the results obtained provided good insight into the quality of data generated by RACOON.

```
RACOON-DATA-GEN(S, size)
 1  cmd_cnt ← 1
 2  Randomly choose job J_i using Q_J from S
 3  job_chunk_len ← N(μ, δ)
 4  EXPAND-META-COMMAND(J_i, job_chunk_len)
 5  cmd_cnt = cmd_cnt + job_chunk_len
 6  repeat
 7        Randomly choose job J_j using P_J and J_i
 8        job_chunk_len ← N(μ, δ)
 9        EXPAND-META-COMMAND(J_j, job_chunk_len)
10        cmd_cnt = cmd_cnt + job_chunk_len
11    until (cmd_cnt ≤ size)
```

Table 4. Data generation from a RACOON template

```
EXPAND-META-COMMAND(J, s_len)
 1  cmd_cnt ← 1
 2  Randomly choose meta-command M_i using Q_J
 3  Randomly choose command c from M_i
 4  Output c
 5  Increment cmd_cnt
 6  repeat
 7        Randomly choose meta-command M_j using P_J and M_i
 8        Randomly choose command c from M_j
 9        Output c
10        Increment cmd_cnt
11    until cmd_cnt ≤ s_len
```

Table 5. A helper routine to expand a meta-command

6.1 Statistical Similarity Measures

Commonly used statistical similarity measures [25, 3, 19] take two samples and verify whether they came from the same population. The underlying hypothesis testing framework requires that for strong similarity, both samples should lie in the 99% (or more) confidence interval, or in other words display a high degree of overlap. We show the results of four statistical measures, viz., 2 sample t-test, F-test, Levene's test and Mann Whitney test.

Figure 7 shows the p-values (indicated by the vertical bars) obtained for both actual and generated data sets for each test. We have used a very high confidence level of 99% for the statistical tests and a p-value of at least 0.05 indicates good evidence that both data sets are statistically similar. In a typical run on which the tests were conducted, 7 users failed the 2 sample t-test, 14 users failed the F-test, 13 users failed the Levene's test and only 2 users failed the Mann-Whitney test. In order to understand the reason for these failures, we visually inspected the histogram plots of failed users. Figure 8 shows a comparison of actual and generated data set of one such user. The two histograms are identical except for an outlier spike (circled) which effectively shifted the moments towards it and outside the 99% confidence interval. This spike can be traced to RACOON's data generation routine which on occasion repeatedly generates the same command, although it didn't occur with high frequency in the actual data set.

6.2 Information Theoretical Measures

Lee et al. [26] showed that information theoretical measures such as entropy and relative entropy can be used for anomaly detection. The main idea is that two similar data sets will have the the same amount of randomness. We have separately computed the entropy values of both actual and generated data sets for all 50 users in a typical run and this

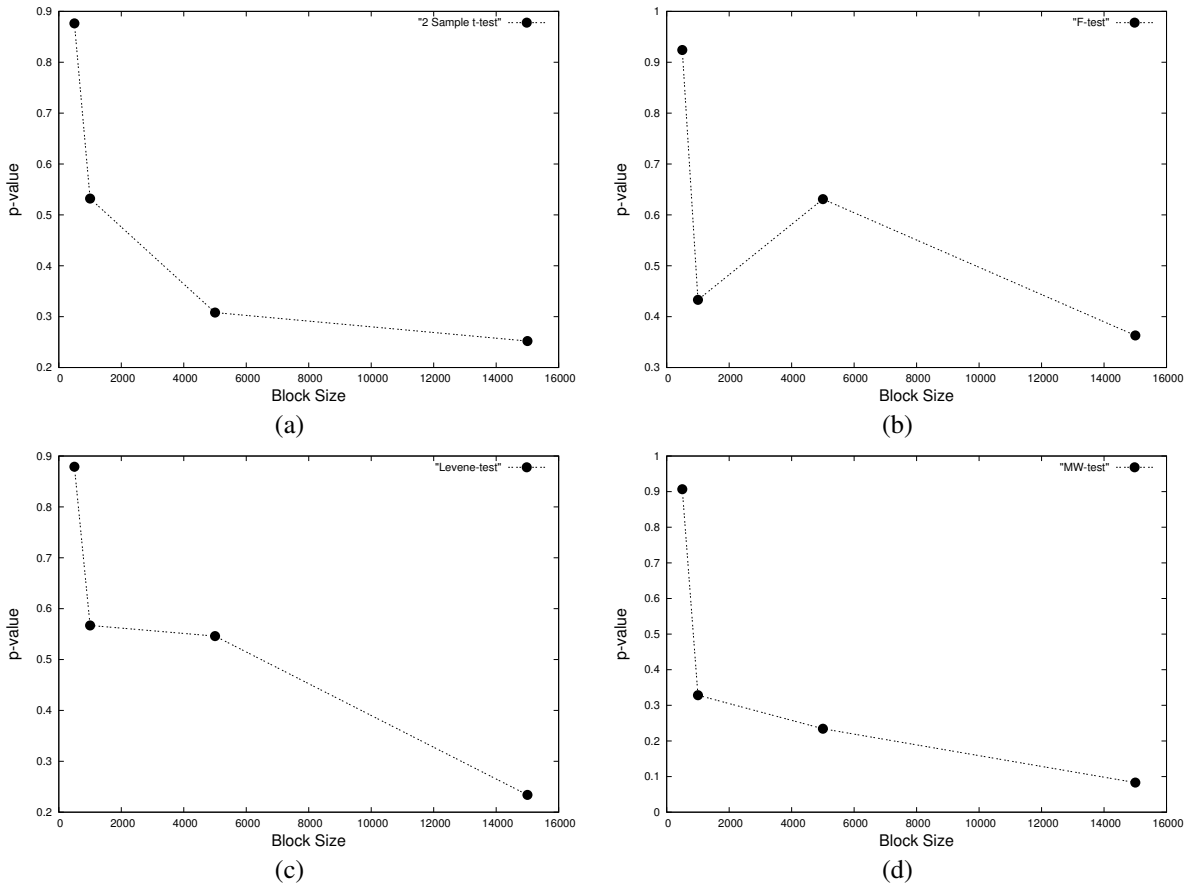


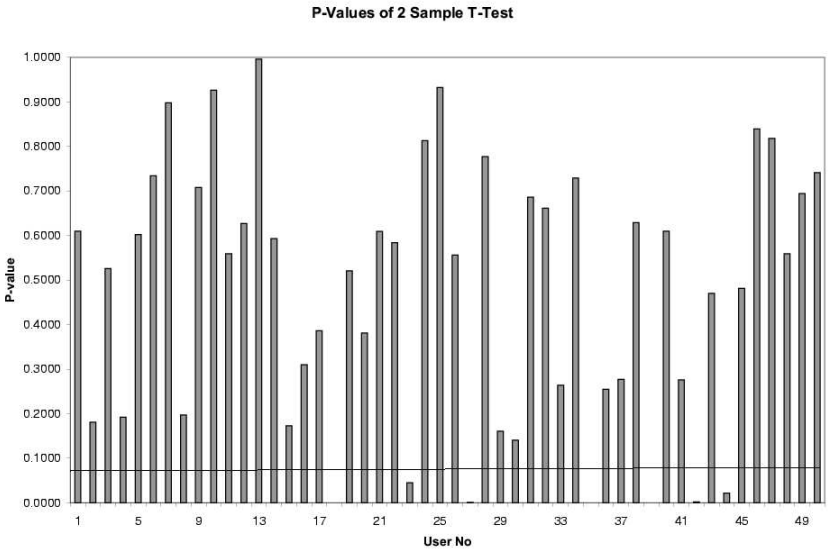
Figure 6. Effect of data set discretization on similarity measures - (a) 2 sample t-test, (b) F-test, (c) Levene test, and (d) Mann-Whitney test

is shown in Figure 9 (adjacent vertical bars). It can be seen that both the data sets have very similar entropy values, differing only by 1.5 to 2.0 for almost all users. Furthermore, the pair-wise relative entropy values of actual and generated data sets (shown in Table 6) are very close to zero indicating similarity from a information-theoretic point of view.

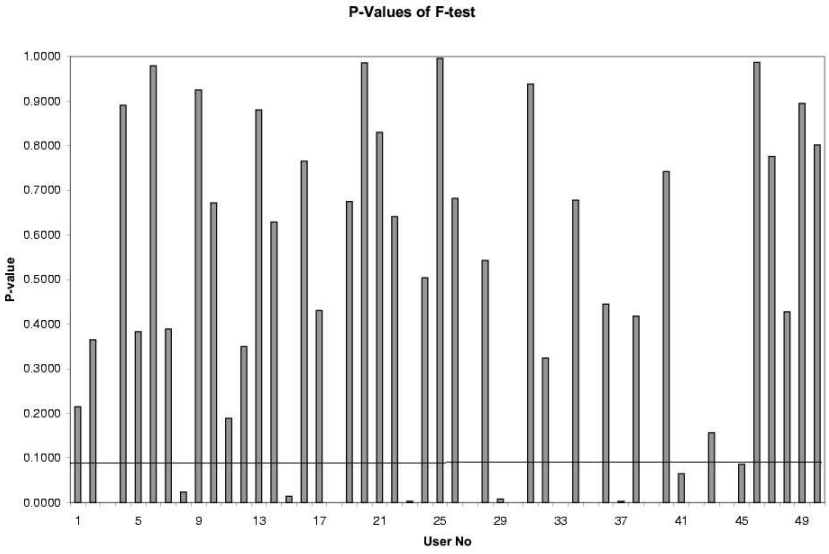
6.3 Anomaly Detection Algorithms

In our experiments, both the statistical and information theoretical measures were generally in favor of strong similarity. However, both these classes of tests are dependent on command frequencies and the first few moments. Therefore, for a true evaluation, we also compared the generated data against actual data using algorithms which have been shown to perform good masquerade detection; in particular, Uniqueness [23] by Schonlau et al. and Naive Bayes Classifier [15] by Maxion et al. Both these algorithms reportedly perform very well on Schonlau's data set with high detection rates or low false positive rates or both. We tried to replicate the experimental design the best we could based on the papers, and ran our implementations on the actual and generated data sets.

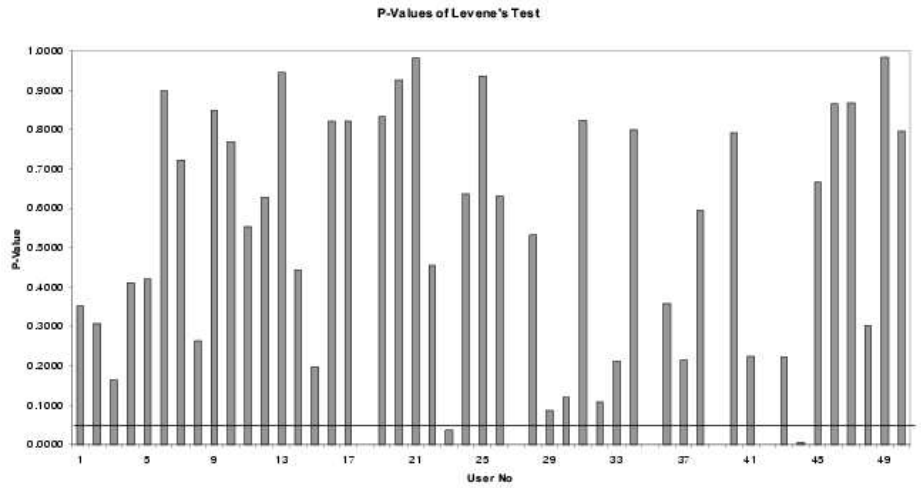
Figure 10 shows the true and false positive rates obtained when a user is masqueraded with the remaining 49 users. We have chosen a user who is a representative of both the best case and worst case scenario for the RACOON generated data set. In general, the true positive and false positive rates for the actual and generated data set follow each other closely. However, there are cases where we can see drastic gaps. This is mainly because RACOON's data generation parameters do not dynamically adapt to the actual data; some users are extremely noisy, while others are not. Since RACOON uses the same procedure for all users, these cases stand out. This shows that RACOON is not perfect and we are looking at ways to enable RACOON to handle diverse cases differently. Note that in the figures we have used connected lines because we found it a convenient way to show


P-Values of 2 Sample T-Test
P-value
1.0000
0.9000
0.8000
0.7000
0.6000
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
1
5
9
13
17
21
25
29
33
37
41
45
49
User No

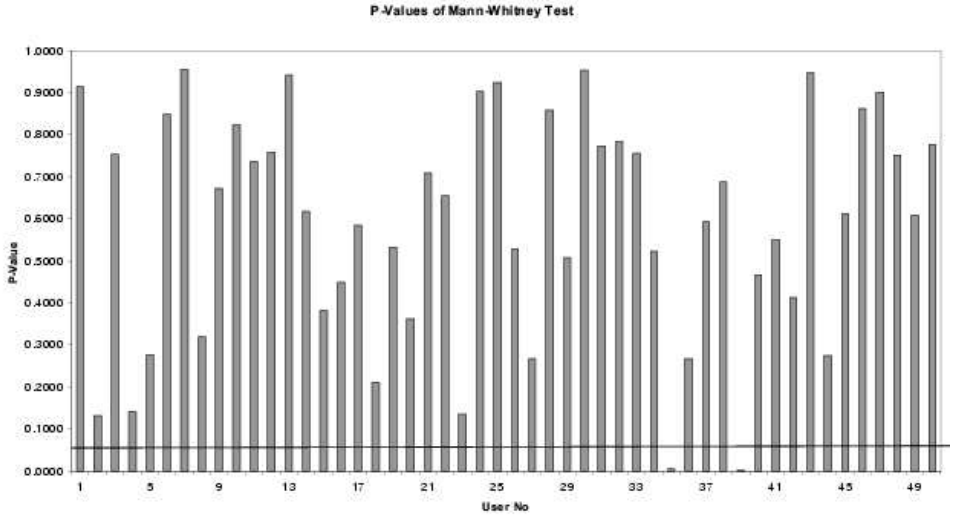
(a)


P-Values of F-test
P-value
1.0000
0.9000
0.8000
0.7000
0.6000
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
1
5
9
13
17
21
25
29
33
37
41
45
49
User No

(b)



(c)



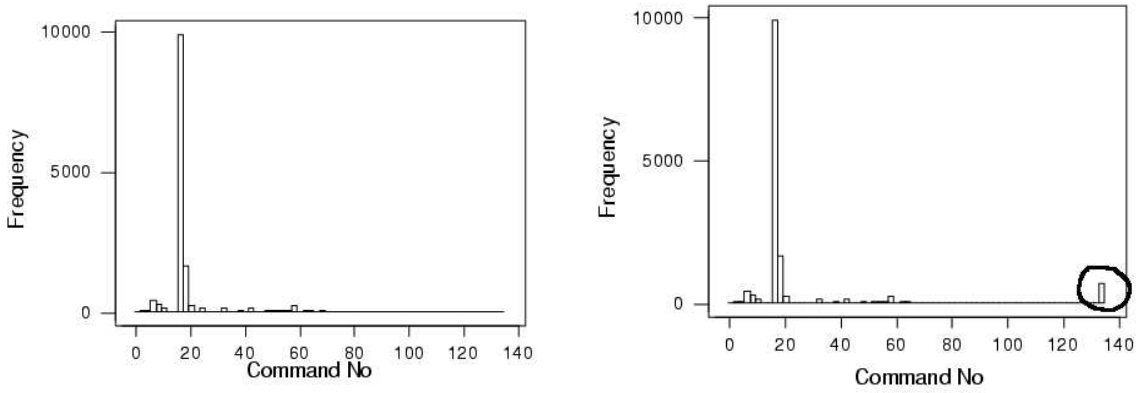
(d)

Figure 7. Statistical similarity tests: Plot of p-value for each pair of users from actual and generated data set (a) 2 sample t-test, (b) F-test, (c) Levene's test, and (d) Mann Whitney test

the gap between the results obtained in the two data sets, and they are not meant to indicate any correlation behavior. Also, we have not used ROC curves because we were separately comparing false and true positive rates with the values reported in the papers which describe the algorithms used for evaluation.

7 Conclusion and Future Work

In this paper, we have presented the technical details of our user command data generation tool called RACOON. This tool was mainly developed to answer the need for diverse and substantial data sets for user level anomaly detection systems; an area where large scale data collection is difficult and often impractical. RACOON is a specialized tool to simplify this process. We summarize the expected



Actual data Generated data

Figure 8. Comparison of histograms of user commands for a failed user

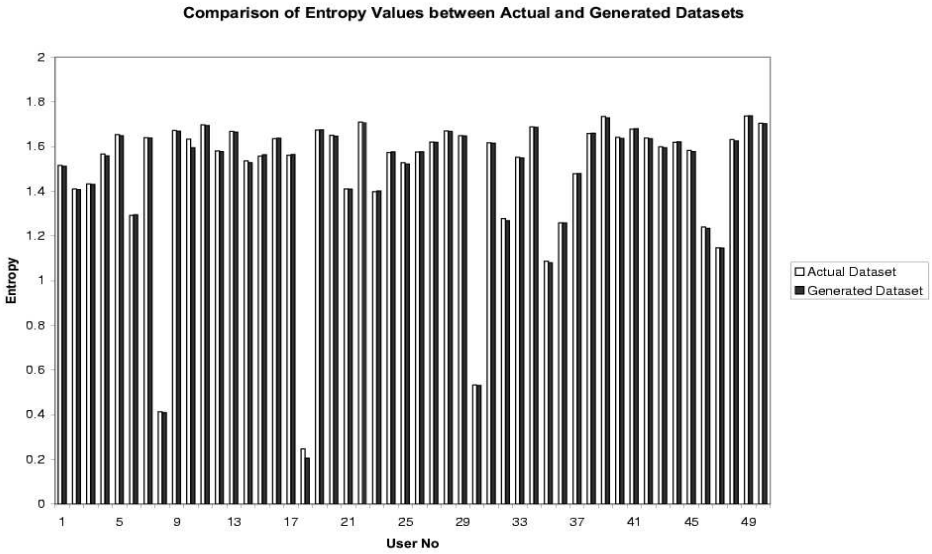


Figure 9. Entropy values for all 50 users of actual and generated data set

impact as follows.

- *Rapid Data Availability*

  Passive data collection requires long-term cooperation from multiple users. Collecting a large enough data set will likely take months or even years. RACOON makes it possible to generate user command data of desired quality very rapidly.

- *Non-intrusive Alternative*

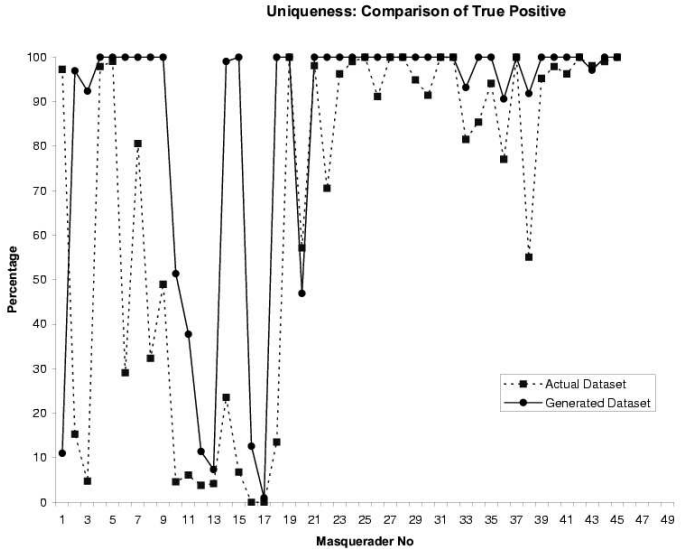
  RACOON provides a non-intrusive alternative to user command data collection, and all technical and non-technical hurdles can be avoided.

- *Accelerated Development and Evaluation Cycle*

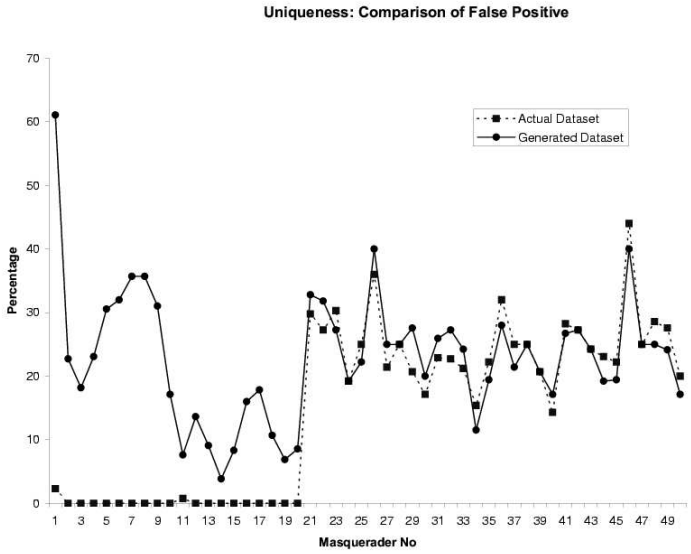
  Data availability is a major obstacle for any IDS development. So far the only option regarding user command data has been patient data collection or being constrained by available data sets. We have shown that not only can RACOON generate reasonably good data sets, but also create these data sets very rapidly.

- *Environment-specific Data*

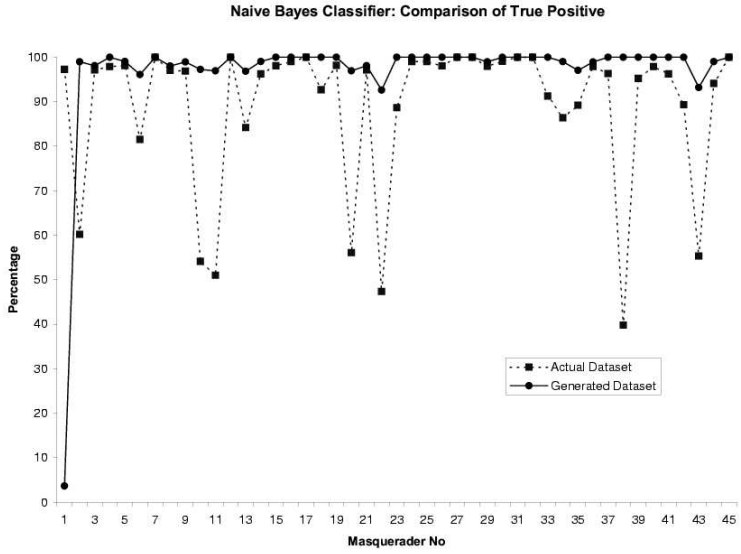
  User behavior is largely governed by the computational resources available to a user. Therefore, user command data is specific to a particular environment. Hit and miss rates of an anomaly detection algorithm


Uniqueness: Comparison of True Positive
Percentage
Masquerader No
Actual Dataset
Generated Dataset

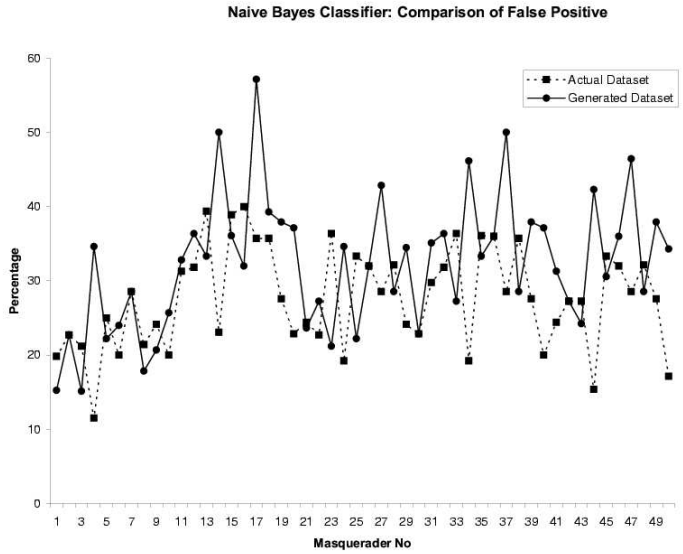
(a)


Uniqueness: Comparison of False Positive
Percentage
Masquerader No
Actual Dataset
Generated Dataset

(b)



(c)



(d)

Figure 10. True positive and false positive rates for both actual and generated data sets: (a) Uniqueness true positive rate, (b) Uniqueness false positive rate, (c) Naive Bayes Classifier true positive rate, (d) Naive Bayes Classifier false positive rate

User pair	Relative entropy	User pair	Relative entropy
1-1	0.004885	26-26	0.00812
2-2	0.005053	27-27	0.013434
3-3	0.006049	28-28	0.003726
4-4	0.010531	29-29	0.011378
5-5	0.005026	30-30	0.005978
6-6	0.003863	31-31	0.005897
7-7	0.006649	32-32	0.003849
8-8	0.001991	33-33	0.011544
9-9	0.007206	34-34	0.00393
10-10	0.031917	35-35	0.105776
11-11	0.002871	36-36	0.010511
12-12	0.009603	37-37	0.005099
13-13	0.008937	38-38	0.005677
14-14	0.003358	39-39	-0.039891
15-15	0.011122	40-40	0.007047
16-16	0.011062	41-41	0.006167
17-17	0.003946	42-42	0.017824
18-18	0.075082	43-43	0.006492
19-19	0.004035	44-44	0.005466
20-20	0.008113	45-45	0.009323
21-21	0.001495	46-46	-0.000117
22-22	0.007645	47-47	0.002272
23-23	0.008207	48-48	0.011631
24-24	0.003783	49-49	0.005915
25-25	0.003662	50-50	0.004892

Table 6. Relative entropy measures between actual and generated data sets

on one data set does not translate literally to another. Moreover, the amount of noise also varies. RACOON allows an analyst to generate a wide variety of data sets using differently constructed user profile templates and perform a comprehensive or site-specific evaluation.

Any synthetic data generation methodology is controversial and looked upon with suspicion. However, with the severe shortage of data in the field of intrusion detection, there seems to be no other alternative. We emphasize again that the most desirable aspect of such tools is the data generation process is highly tunable. RACOON's templates, either manually created or obtained from a known data set, are the fundamental basis of its "tunability". RACOON is still a tool in its infancy and there are many areas of improvement. First, we want to integrate the support for enriched command lines to truly simulate a shell command. Next, RACOON presently provides only offline user command data and it would be desirable to develop an API to interact with the core engine to produce data in an on-demand online manner.

We are planning on a public release of RACOON soon to aid the research community in its efforts towards better user command based anomaly detection techniques.

References

[1] E. L. Barse, H. Kvarnström, and E. Jonsson. Synthesizing test data for fraud detection systems. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003)*, 2003.

[2] H. Debar, M. Dacier, A. Wespi, and S. Lampart. An experimentation workbench for intrusion detection systems. Technical Report RZ2998, IBM Research Division, Zurich Research Laboratory, Zurich, Switzerland, 1998.

[3] M. H. DeGroot and M. J. Schervish. *Probability and Statistics (3rd Edition)*. Pearson Addison Wesley, 2001.

[4] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A Sense of Self for Unix Processes. In *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[5] A. Ghosh, A. Schwartzbard, and M. Schatz. Learning program behavior profiles for intrusion detection. In *1st USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, California, April 1999.

[6] S. Greenberg. Using Unix: Collected Traces of 168 Users. Technical Report 88/333/45, Department of Computer Science, University of Calgary, Calgary, Canada, 1988.

[7] T. Lane. Hidden markov models for human/computer interface modeling. In *Proceedings of the IJCAI-99 Workshop on Learning About Users*, pages 35–44, 1999.

[8] T. Lane. Purdue UNIX User Data. http://www.cs.unm.edu/ terran/research/data/Purdue_UNIX_user_data.tar.gz, 1999.

[9] T. Lane and C. E. Brodley. An application of machine learning to anomaly detection. In *Proceedings of the Twentieth National Information Systems Security Conference*, volume 1, pages 366–380, Gaithersburg, MD, 1997. The National Institute of Standards and Technology and the National Computer Security Center, National Institute of Standards and Technology.

[10] T. Lane and C. E. Brodley. Sequence matching and learning in anomaly detection for computer security. In *Proceedings of AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 43–49, 1997.

[11] T. Lane and C. E. Brodley. An empirical study of two approaches to sequence learning for anomaly detection. *Machine Learning*, 1999. Under review.

[12] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. The 1999 DARPA Off-line Intrusion Detection Evaluation. *Computer Networks: The International Journal of Computer and Telecommunications Networking (Special issue on recent advances in intrusion detection systems)*, 34(4):579 – 595, October 2000.

[13] R. P. Lippmann, I. Graf, D. Wyschogrod, S. E. Webster, D. J. Weber, and S. Gorton. The 1998 DARPA/AFRL Off-Line Intrusion Detection Evaluation. In *First International Workshop on Recent Advances in Intrusion Detection (RAID)*, Louvain-la-Neuve, Belgium, 1998.

[14] R. A. Maxion. Masquerade detection using enriched command lines. In *International Conference on Dependable Systems and Networks (DSN-03)*, San Franciso, CA, USA, June 2003.

[15] R. A. Maxion and T. N. Townsend. Maquerade detection using truncated command lines. In *International Conference on Dependable Systems and Networks (DSN'02)*, pages 219–228, June 2002.

[16] 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 of Information and System Security (TISSEC)*, 3(4), November 2000.

[17] ns2. *The Network Simulator*. http://www.isi.edu/nsnam/ns/ns-documentation.html.

[18] OPNET. *OPNET's Technology Guide for "Optimizing and Deploying Networked Applications*. http://www.opnet.com.

[19] A. Papoulis and S. U. Pillai. *Probability, Random Variables and Stochastic Processes*. 2001.

[20] P. A. Porras and P. G. Neumann. EMERALD: event monitoring enabling responses to anomalous live disturbances. In *1997 National Information Systems Security Conference*, oct 1997.

[21] L. M. Rossey, R. K. Cunningham, D. J. Fried, J. C. Rebek, R. P. Lippmann, J. W. Haines, and M. A. Zissman. Lariat: Lincoln adaptable real-time information assurance testbed,. In *2002 IEEE Aerospace Conference*, March 2002.

[22] M. Schonlau. Masquerading User Data. http://www.schonlau.net/intrusion.html, 1998.

[23] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi. Computer intrusion:detecting masquerades. *Statistical Science*, 16(1):58–74, 2000.

[24] K. Wang and S. J. Stolfo. One class training for masquerade detection. In *rd IEEE Conf Data Mining Workshop on Data Mining for Computer Security*, 2003.

[25] N. A. Weiss. *Introductory Statistics 5/e*. Pearson Addison Wesley.

[26] D. X. Wenke Lee. Information-theoretic measures for anomaly detection. In *IEEE Symposium on Security and Privacy*, 2001.