

# Toward Realistic and Artifact-Free Insider-Threat Data

Kevin S. Killourhy  
ksk@cs.cmu.edu

Roy A. Maxion  
maxion@cs.cmu.edu

*Dependable Systems Laboratory  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA*

## Abstract

*Progress in insider-threat detection is currently limited by a lack of realistic, publicly available, real-world data. For reasons of privacy and confidentiality, no one wants to expose their sensitive data to the research community. Data can be sanitized to mitigate privacy and confidentiality concerns, but the mere act of sanitizing the data may introduce artifacts that compromise its utility for research purposes. If sanitization artifacts change the results of insider-threat experiments, then those results could lead to conclusions which are not true in the real world.*

*The goal of this work is to investigate the consequences of sanitization artifacts on insider-threat detection experiments. We assemble a suite of tools and present a methodology for collecting and sanitizing data. We use these tools and methods in an experimental evaluation of an insider-threat detection system. We compare the results of the evaluation using raw data to the results using each of three types of sanitized data, and we measure the effect of each sanitization strategy.*

*We establish that two of the three sanitization strategies actually alter the results of the experiment. Since these two sanitization strategies are commonly used in practice, we must be concerned about the consequences of sanitization artifacts on insider-threat research. On the other hand, we demonstrate that the third sanitization strategy addresses these concerns, indicating that realistic, artifact-free data sets can be created with appropriate tools and methods.*

## 1. Introduction

An insider is a person with legitimate access to an organization, and who acts *maliciously* against that organization. The insider threat is a significant and growing concern, especially in fields where espionage and fraud are profitable.

A survey of insider incidents in the banking and finance sector found that 30% resulted in losses in excess of \$500,000 each [10]. Examples of insider behavior include the unauthorized modification of company data for personal profit, the compromise of other employees' computer accounts, and the installation of "back doors" through which the insider regains access in the event he or she is terminated [4].

For almost two decades, researchers have been proposing systems to detect and prevent insider threat. These systems work by monitoring users, profiling their behavior, and identifying suspicious or anomalous activity. The earliest systems analyzed audit records and built profiles of the commands each user executes [1, 12]. The conjecture was that a legitimate user might be distinguished from an impostor by their distinct usage of commands (e.g., the user used `vi`, the impostor runs `emacs`); also, an insider straying from authorized activity might be detected when he used anomalous commands (e.g., to copy a file he does not normally access). As a result of that early work, many systems have been proposed for detecting insiders using Unix command-line data [5, 8, 11].

These insider-threat detection systems are best evaluated using natural, real-world data to measure, compare, and improve their performance. Researchers have instrumented computer systems to monitor the behavior of participating users [2, 5, 11]. They collect, *sanitize*, and share their data with the research community. Sanitization is the act of replacing sensitive data (such as passwords) with uninformative markers (such as the string `<XXXXXX>`). Sharing the data allows other researchers to evaluate and compare the performance of different insider-threat detectors.

However, since a goal of these evaluations is to estimate the "real-world" performance of a system, we must ask how effectively the data serve this goal. When an evaluation uses these data, how confident are we that the outcome of an evaluation carries over to the real world? If these evaluations are being used to determine which insider-threat detector is deployed, then we must be confident that the eval-

uation is accurate. Deploying an insider-threat system that under-performs is risky, increasing the already high cost of insider-threat.

Unfortunately, because of the way existing data sets were created, we must be wary of generalizing to the real world on the basis of evaluations that use these data sets. First, the data are not *realistic*, by which we mean that the data do not reflect what one would find in a real-world environment. Specifically, in the real world, an insider’s choice of commands would stem from his or her underlying malicious intentions. Existing data sets contain no actual insider behavior; it is experimentally induced, typically by designating some normal users as impostors, and using their commands as a substitute for insiders’ commands. Those commands might have been observed when the user was checking his mail, writing a program, or any other benign activity. We cannot assume that a system which is good at detecting when a user is checking his mail will perform equally well detecting insider activity.

Second, when the data are sanitized, *artifacts* are introduced into the data set. Artifacts are modifications of the data (due to the sanitization process) that alter the outcome of experiments using the data. For instance, if sanitization replaces all user names with `<XXXXXX>`, then commands which were once distinct become indistinguishable. Suppose an evaluation tested whether a system could detect when user `dave` began snooping around user `mary`’s home directory. The benign command `cd dave` and the suspicious command `cd mary` are easily distinguished in the raw, unsanitized data, but both appear as `cd <XXXXXX>` in the sanitized data. A detector which would detect `dave`’s snooping in the real world might miss it in the sanitized data because of artifacts. On the basis of this failed evaluation, a detector which would actually work well in practice might never be deployed.

## 2. Problem and approach

Experiments using existing insider-threat data may not generalize to the real world because: (1) benign commands typed by normal users are injected as a substitute for commands typed by malicious insiders; and (2) unintended sanitization artifacts may be introduced when sensitive data are replaced with uninformative markers. As a result, benign user activity could be flagged as insider activity when it is not, or actual insider activity could go unnoticed.

To address these issues, we have developed a suite of tools and a methodology for using them. To maximize the realism of injected insider behavior, we developed a library of carefully scripted and vetted insider activities. While the realism of insider injections is a concern, the primary focus of this work is on the effect of sanitization artifacts. To ensure that sanitization does not introduce arti-

facts, we developed a sanitizing engine which allows users to review their data, mark sensitive data, and export sanitized data sets. This Sanitizer incorporates three different sanitization strategies: Redact-Only, Token-Only, and Word-Token. They differ in how they cover up sensitive data (e.g., Redact-Only uses a “black box” like `<XXXXXX>`, Token-Only uses a distinct token like `<TOKEN-12>`, and Word-Token breaks the sensitive data into words and then uses one token for each word). Redact-Only and Token-Only are similar to strategies used to sanitize existing data sets. The Word-Token strategy was designed to be artifact-free.

The present work is framed much like earlier work in insider detection. In particular, we compare two types of insider monitoring, called Enriched (comprising the entire command line typed by a user) and Truncated (comprising only the name of the command executed). The purpose of conducting this experiment is to compare the results obtained using raw, unsanitized data to the results using each of the three sanitization strategies. We intend to establish whether artifacts arise in the two types of monitored data (Truncated and Enriched) as a consequence of sanitization, and what the effects of those artifacts are on the ability to detect insider activity.

## 3. Related work

Three existing insider-threat data sets are commonly used by researchers. Unfortunately, each contains unrealistic insider injections and sanitization artifacts. Greenberg [2] collected a corpus of Unix command-line data, and Maxon [7] assembled it into an insider-threat data set. However, benign commands from normal users were used as a substitute for insider commands. Further, to protect participating users’ privacy, usernames were sanitized by replacing each letter of the username with an “x” character (e.g., `dave` and `mary` each contain four characters, and both would appear as `xxxx`). Lane and Brodley [5] also collected users’ commands, but again, benign commands were used as a substitute for insider commands. The commands were also sanitized by replacing every sequence of one or more file names with a number indicating how many names were in the sequence. Schonlau et al. [11] collected the names of programs executed (rather than the full command line), but again, benign commands were used in place of insider commands. Further, one could argue that collecting only the names of programs constitutes a form of sanitization (especially since the authors state that full command lines were not collected because of “privacy concerns”). While these data sets have helped researchers develop and refine insider-detection methods, we remain concerned about the effect of unrealistic insider injections and sanitization artifacts on those researchers’ experiments.

The problem of data-set artifacts has been demonstrated by Mahoney and Chan [6] in the domain of intrusion detection. They found that an existing data set used to evaluate intrusion-detection systems contained evidence of the artificial procedure used to synthesize the data. They demonstrated that a detector could be built which would perform well in the evaluation (by detecting these artifacts), and yet it would have little chance of working well in practice. Their findings highlight the need for data to be realistic and artifact-free.

The tools we develop in this study are similar to others in the literature. The HoneyNet Project offers a data-collection tool called Sebek [3] which is similar to the data collector we develop. Data anonymization algorithms such as those proposed by Sweeney [13], and tools like that developed by Pang and Paxson [9] for Internet traffic are also useful for data sanitization. One might argue that anonymization and sanitization are the same, but the literature is not clear on whether anonymization includes the removal of confidential or sensitive data (as stated by Pang and Paxson) or just the removal of identifying data (as stated by Sweeney). In any event, our tools were not designed to supplant these existing tools but to provide similar capabilities. We created our own suite of tools because they had to be interoperable, not because existing tools provided lesser functionality. As a consequence, our findings should be relevant to users of these similar tools as well.

## 4. Overview of methodology

In order to examine the consequences of sanitization artifacts on insider-threat experiments, we replicated a typical experiment from the literature. In our replication, the original experiment was conducted first using raw, unsanitized data and then repeated using data treated with each of the three sanitization strategies. We compared the results from these experiments to reveal whether they were altered by sanitization artifacts.

The experiment chosen for this exercise was conducted by Maxion [7] who was studying the effect of two different types of data (called Truncated and Enriched) on the effectiveness of a naive-Bayes insider-threat detector. He compared the performance of a detector given only the (Truncated) program names typed by a Unix user to the performance of the same detector given the full (Enriched) command line. Performance was measured in terms of the cost of error of the detector (calculated as the sum of the miss and false-alarm rates). Maxion found that the cost of using Enriched data was 9% lower than the cost of using Truncated data. However, his experiment used the Greenberg data set [2], which contains sanitization artifacts as discussed above. As a consequence, we cannot be sure that the 9% difference in cost predicted by the experiment would

also be seen in a real-world deployment.

In order to see the effects of sanitization on this experiment, we built a data-collection program called Monolog, and deployed it on the workstations of system administrators and operations staff members within the university. The data collector recorded each user's commands during his or her natural daily activities.

We assembled a "library" of realistic insider attacks, scripted the attacks, and launched the scripts against participating users' accounts. Whereas other researchers injected one user's commands into another user's data as a simulated attack, our scripts perform a real-world attack against the user's account (and then recover from the attack). The scripts were designed to impersonate an attacker who gains unauthorized access to an account (e.g., by using the victim's workstation while he or she is away).

In another departure from other researchers' methods, users in the present study sanitized their own data. We felt that the users themselves were in the best position to identify sensitive information, and that their input would help researchers understand what sort of data users are reluctant to share. An application called the Sanitizer was built to allow users to view and to search their own data, to mark records as sensitive, and to export sanitized copies of the data. Data could be exported using any of the three sanitization strategies (Redact-Only, Token-Only, and Word-Token), which are described in detail in Section 5.3. By having users sanitize their own data, researchers avoided having to employ unnecessarily broad, draconian sanitization techniques.

To replicate Maxion's experiment, we converted the users' sanitized data into Truncated and Enriched evaluation data sets, and the performance of the naive-Bayes detector was tested on each one. The miss rate, false alarm rate, and cost of error were calculated as in the earlier experiment. To compare the results to those using raw, unsanitized data, we deployed the naive-Bayes detector on users' workstations, and we walked the users through the process of running it on their own raw data and reporting the results. In this way, we were able to compare the results of the experiment on raw data to the results with each of the three types of sanitization, yet the users' sensitive raw data were never shared with the researchers.

## 5. Apparatus

We wrote three software programs to enable this investigation: (1) a reliable data-collection package called Monolog; (2) an insider-script library to automate the injection of realistic insider attacks; and (3) a Sanitizer application to enable the review and sanitization of collected data.













