CS2-AAAB

The Big Picture: Using Desktop Imagery for Detection of Insider Threats

a Major Qualifying Project Report
submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the
Degree of Bachelor of Science by

David Muchene

Klevis Luli

December 18, 2012

Professor Craig A. Shue

Abstract

The insider threat is one of the most difficult problems in information security. These threats are caused by trusted people misusing systems in which they have privileged access. Prior research has addressed insider threat detection by using machine learning techniques to profile user behavior. In many of these papers, the user behavior profiled is represented as low level system events (e.g. system calls). System calls do not provide sufficient contextual information about the user's environment or intentions, and as such using them as a representation of user behavior causes insider detection systems to have high misclassification rates.

In order to acquire a better capture of user's intentions, we explored using video of user's sessions as the representation of their behavior. Further, we developed a system that uses these recordings to detect moments during which users performed actions which warrant more scrutiny. Analysis of the recordings is accomplished using Optical Character recognition (OCR) and basic text classification. Moreover, we use scene detection algorithms to shorten the recordings into key frames and consequently reduce the time needed for analysis. The system outputs its results as a web interface which contains a timeline highlighting all of the moments that were identified as sensitive. Our results show that using high level information such as desktop images is a viable alternative to using system calls for insider threat detection.

Contents

List of Figures

List of Tables

1 Introduction

Unhappy former employees, contractors, and other authorized users, present a significant risk to an enterprises network due to the knowledge they possess about the organization's systems and security measures. Insider attacks can cause more damage than outsider attacks because they are more extensive and they can go undetected for a longer time. In addition, such incidents are unreported because companies and organizations want to avoid bad publicity. Recent real life stories examples like Wikileaks and EMC's RSA security breach are both examples of insider threat that had serious consequences for goverments and industries [21, 16]. Detecting and mitigating the threats insiders pose is a significant problem in computer security. In fact, the problem is significant enough that in their report, "A Road Map for Cybersecurity Research", the Department of Homeland Security placed it among the 11 hard problems in the computer security field [17]. In the report, they define an insider threat as one that is posed by people who abuse granted privileges, and specify the scope of the problem to include individuals who masquerade as others, individuals who abuse their own privileges, and individuals who are fooled by attackers into abusing their own privileges. These malicious insiders are typically concerned with exfiltration of data, but they can also compromise integrity, availability and the accountability of systems [17].

According to the DHS report, the insider threat today is most commonly addressed through awareness training, background checks, good labor practices, identity management, and user authentication. Limited audits, network modeling, application-level profiling and monitoring are also measures employed in combatting insider threat. Current research and literature that deals with insider threat detection can be grouped into three categories: host-based user profiling, network-based sensors, and integrated approaches [24]. The host-based user profiling approach entails profiling users by the commands they issue, monitoring system calls, or monitoring database and file access patterns. Profiling is accomplished by modeling statistical features such as frequency and duration of events. Several papers have used machine learning techniques to perform this statistical modeling [24]. Network-based sensors do not rely on profiling, but rather on detecting whether a user is violating network policy. For instance in their paper, Maloof and Stephens developed a system, ELICIT, which determines the likelihood a user is an insider threat by detecting whether they accessed information they did not need for their job [4]. Other papers have also looked into using traps such as honey tokens (information a user is not authorized to access) to detect malicious insiders [24].

One of the problems with current insider threat research is the lack of attack data to be used with profiling or to study a general solution. The reason for this is that it is quite difficult to collect data from normal users in different environments, and it is even more difficult to collect data from various malicious insiders [24]. Additionally, techniques that rely on behavior profiling suffer because observed behavior is not a good approximation of user intent when there is not sufficient information about the user and the environment. This leads systems to either have false alarms or be less stringent in an attempt to decrease the false alarms [2].

1.1 Proposal

In this project, we propose and develop a system that uses high level information to solve the problem of lacking contextual information about the environment. The system allows administrators to audit users sessions and aids in determining whether the user is a malicious insider. Our system model assumes users in a certain network would use Virtual Network Computing (VNC) to login to an enterprise network, and have their own personalized profiles to carry their daily work routines. VNC is a graphical desktop sharing system that uses the Remote Frame Buffer (RFB) protocol to remotely control another computer, which we will later discuss in detail. We propose the usage of VNC under the assumption that more companies are using it to provide remote access to their employees because it provides a more secure alternative compared to providing users with a Virtual Private Network (VPN) connection. The reason for this is because instead of having to give network access to a remote machine that is not trusted, they are only pushing a stream of frames and everything is executed in a computer inside the network. Once a user logs in to his VNC environment, the system begins recording all of his/her actions for analysis. Manually auditing every user's recording (which might be 8 hours or longer) would be quite difficult, especially with a large number of users. For this reason, the script performs an analysis on the video and points to the most sensitive parts. The analysis consists of two phases:

- First, we create a caption of the recording, which contains the words seen by the users, including the window names, achieved through Optical Character Recognition technology (OCR). Performing OCR on every frame of a potentially long video would take a long time, so for optimization we implement an algorithm that prunes redundant frames.
- After the caption is created, it is processed to detect key words, phrases, or symbols. Additional information such as system calls and running processes will be added to each caption at this point, to provide more context.

The guiding idea for this project is that knowing user activity at particular times, we get a better idea of their intent, and therefore implement an approach that is more complete than the other ones.

The rest of the paper is organized as follows: we summarize prior insider threat research, describe the various components of our system in detail, perform various tests on the framework and describe the results, and finally we conclude with a summary of the lessons learned and future improvements.

2 Background

In this section, we summarize prior research that addresses the insider threat problem, provide an overview of machine learning techniques used to detect these threats, and introduce technologies we considered using for this project.

2.1 Overview of Prior Insider Threat Detection and Mitigation Work

The insider threat is a reocurring problem in software security that happens when trusted users, who may have an understanding of the system, misuse it in ways that allow them to access or tamper with data for their gain. Researchers at the CERT Insider Threat Center at Carnegie Mellon University have been analyzing insider threat cases since 1999 in an effort to draw conclusions about the specifics of these attacks and attackers [15, 7]. They have shown that while a majority of attacks are perpetrated by outsiders, a significant portion of threats originate from insiders. Their study published in 2005 found that 29% of reported attacks were suspected to have come from insiders, but the researchers did also note that only 70% of the respondents were able to distinguish between outsider and insider attacks [15]. In their most recent study, they looked at 80 insider fraud cases in the financial sector (67 insider fraud cases and 13 external fraud cases) in order to develop insights and risk indicators of insider threat [7]. They found that most insiders were not very sophisticated technically, and most incidents were detected through routine audits, customer complaints, or coworker suspicions. The insider threats could be especially expensive to companies in which intellectual property represents a significant portion of their value. Rantala found that even though intellectual property theft constituted less than a percent of all cyber crimes, it was the cause of more than 50% of all monetary loss [23].

Given the prevalence of this problem and the fact that it is impossible to design a system without trusted people, a considerable amount of research has been aimed at mitigating the insider threat problem rather than preventing it. This research generally approaches the problem by profiling user behavior over an extended period of time in order to identify abnormalities that could indicate the presence of a threat. Profiling is accomplished by modeling various statistical features such as frequency of events (e.g. logons, file deletion), duration of events, co-occurrence of multiple events combined through logical operators, and the sequence or transition of events [24]. These features are modeled using a number of statistical methods and machine learning techniques such as Markov Chains [14], naive Bayes classifier [18, 31], and SVMs [30, 28, 25].

2.1.1 Brief Introduction to Machine Learning and its Applications in Insider Threat Detection

A significant amount of research has approached the insider threat problem from a machine learning perspective. Machine learning is a scientific field concerned with building computer systems that can automatically improve with experience [20]. The field has grown considerably since its inception from being studied by a few scientists and engineers to being a broad discipline that has produced significant results in various fields such as speech recognition, computer vision and robot control. There are various categories of machine learning algorithms including supervised learning, unsupervised learning, and reinforcement learning. Supervised learning algorithms are concerned with learning some initially unknown function using a given set of labeled examples called the training data. For instance, if a supervised learning model was trained using a dataset that contained some inputs X and corresponding outputs Y, the model would infer the function that maps X to Y [20]. Supervised learning can also be used for classification, in which case the labels would represent the class a particular data point belongs. For example if we were trying to classify a group of cars as either sports cars or not we might use training data in which the input was a set of attributes (e.g. weight, speed, acceleration) and the output is a simple yes/no tag [6]. Unsupervised leaning algorithms attempt to classify data when no information is given about the membership of data points to pre-defined classes [12]. Much of the prior insider threat research that has used machine learning techniques to profile user behavior employs supervised learning techniques such as Support Vector Machines (SVM), Markov chains, and Bayes classifiers, which we describe below.

2.1.2 Naïve Bayes Classification to Classify User Behavior

Naïve Bayes classifiers are probabilistic classifiers that have a fast learning time and are robust to noise [18]. They are based on Bayes theorem and use the Naïve Bayes assumption which means that they make the assumptions that all the attributes of the data are independent of each other. Training naïve Bayes classifiers involves calculating prior probabilities. The naïve assumption simplifies training and classification especially when the number of attributes is large, because the probabilities can be calculated separately and multiplied together. While the independence assumption is rarely accurate in real world cases, Naïve Bayes classifiers have been shown to perform really well. This has led to their use in various domains such as text classification and gene classification [8]. Maxion and Townsend used Naïve Bayes classification to improve previous user behavior profiling due to the method's proven success in other classification domains [19]. The model assumes that the user generates a sequence of commands, with a fixed probability that each command is independent of the commands preceding it. The output of the naive Bayes classifier was the probability that a sequence of commands originated from a given user versus the probability that the sequence of commands did not originate from the same user. The larger the ratio between the two the greater the evidence the user was behaving normally. The authors also provided a detailed error analysis of the classifier.

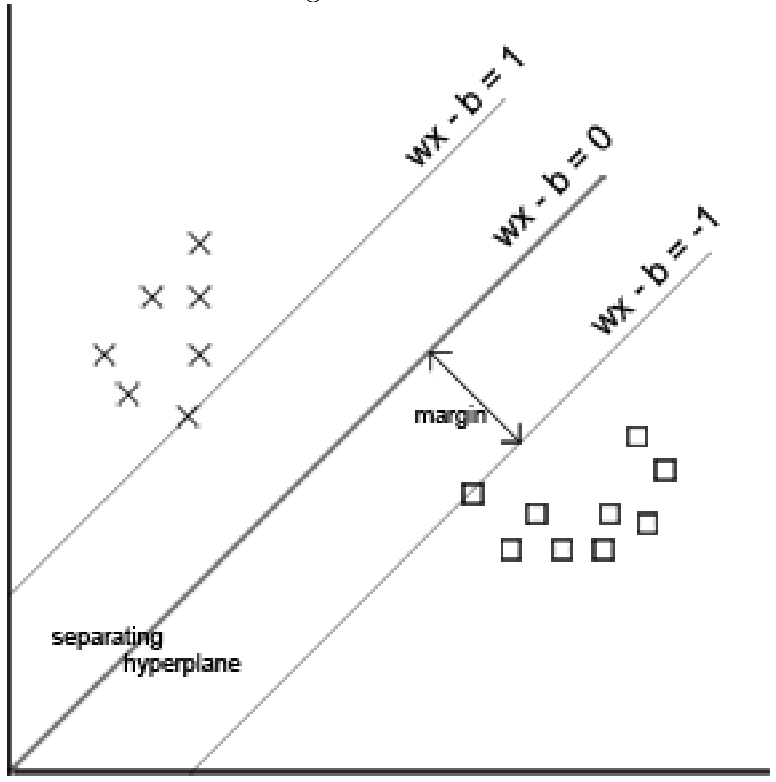
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

2.1.3 SVM for Detecting Anomales Associated with Attacks

SVMs were introduced in 1995 by Vladimir Vapnik and Corinna Cortes as a way of classifying data into two groups [5]. Within a short period of time, SVMs were being used in numerous domains such as handwritten digit recognition, object recognition, speaker identification, and face detection [3]. The basic idea behind SVMs is to separate a given set of data into two classes by maximizing the margin between the two classes' closest points. The SVM problem is introduced by first presenting a simple case where the data points are linearly separable (i.e. if they are 2D, they can be separated by a line and if they are higher dimensional they can be separated by a hyper-plane). In this case, the problem becomes finding the function which maps a given input to one side of the linear separator (depending on the class the input belongs)1. The problem is then extended to a non-simple case where the data is not linearly separable, in which case the data is first mapped to a higher dimension and linear classification is performed. There has been plenty of research performed extending and improving SVM. Scholkopf et al. for instance extended the methodology to handle one class classification (i.e. does a dataset belong so a particular class). Further, the methodology has also been extended to handle multiclass classification.

SVMs have many applications including insider threat detection since they can be used to classify whether user's behavior is normal given a dataset containing examples of normal user behavior. In their work, Wang et al. compared the performance of one class SVMs with that of naive Bayes classifiers in modeling user commands [30].

Figure 1: SVM



2.1.4 Markov Chains for Classifying Users as Insider threats

A Markov chain is a process described in the following way: given a set of states, the process begins in one of the states and moves along from one state to another. The probability that a process moves from state i to state j is called the transition probability [10]. The states are said to be conditionally independent which means the transition probabilities are only dependent on the current state. Transition probabilities are stored in a matrix called the transition matrix T, where T(i,j) contains the probability of a transition from state i to j [11]. These probabilities are used to calculate the probability of a particular sequence. Training Markov models involves estimating the transition probabilities using an method referred to as Maximum Likelihood estimation. Significant work has examined Markov models and they have been applied in many different areas including detection of new genes and prediction of stock returns.Ju and Vardi used high order Markov chains to profile Unix command sequences and determine a user's signature behavior. Their guiding rational is that even though user command sequences are random, they generally follow a probabilistic pattern which can be captured by a rich Markov model. They then compared a user's sequence of commands with his estimated signature behavior using statistical hypothesis testing [14].

2.1.5 Alternatives to Machine Learning for Insider threat Detection

Unfortunately, these behavioral profiling approaches to detecting insider threat suffer from several limitations. First, the profiled features or behaviors only approximate the user's intent due to a lack of information about the context of the user's overall environment. Additionally, the difficulty of generating realistic insider threat datasets complicates profiling [2]. Due to these challenges, Bowen et al. suggest that insider threat detection systems must be multifaceted such that they leverage multiple complementary techniques to detect threats. In the system they propose, they augment their use of long-term behavior profiling with decoy documents (honey-tokens) that trigger alerts when they are misused. The idea of using decoy documents and honeypots was also used by Spitzner [27]. His method was based on the assumption that the insider knows what information he is after, so he implanted honey-tokens with fake value in the network. These honey-tokens would then direct the insider to advanced honey pots that can be used to distinguish malicious intent from benign intent by inspecting the insider's interaction with the honey pot. However, one of the deficiencies of this approach is that it relies on the expectation that the attacker will interact with the honey pot or honey-token, which may never be the case if they know of their existence.

Another alternative to determining insider threat by detecting deviation in user behavior is deciding what constitutes suspicious behavior (i.e. printing away from your office, searching files not needed the job), and creating detectors for such behavior. Deanna et al. for instance studied the differences in behavior between malicious and benign users, and used the data to create rules. Their system used Bayesian networks to calculate the likelihood that a user was a threat given the set of rules that user broke [4].

2.2 Virtual Network Computing

Virtual Network Computing (VNC) is a graphical desktop sharing system that uses the Remote Frame Buffer (RFB) protocol to remotely control another computer. The system consists of a client, a server, and the RFB communication protocol. RFB is a simple protocol that transmits the keyboard and mouse event messages from client to server, and relays the graphical screen updates back from server to client. The VNC server runs on the machine that is sharing its screen and resources, and the client (or viewer) is the program that watches, controls, and interacts with the server.

VNC by default uses TCP port 5900:x, where x is the display number (usually :0 for a physical display). Several implementations also start a basic HTTP server on port 5800:x to provide a VNC viewer as a Java applet, allowing easy connection through any Java-enabled web browser. Because it works at the framebuffer level, RFB is applicable to all windowing systems and applications. A VNC viewer on one operating system may connect to a VNC server on the same or any other operating system. There are clients and servers for many GUI-based operating systems and for Java. Multiple clients may connect to a VNC server at the same time. The display that is served by VNC is not necessarily the same display seen by a user on the server. Because Unix/Linux computers support multiple simultaneous X11 sessions, it is also possible to run multiple VNC sessions from the same computer. However, on Microsoft Windows, the VNC session served is always the current user.

VNC is often used for remote technical support, accessing files on one's work computer from one's home computer, or to give access to centralized resources from simple, inexpensive devices. Using VNC, simple devices get access to more powerful server machines that are connected to the network and provide applications, data, and storage for a users preferences and personal customizations. It allows an entire desktop environment to be accessed from any Internet-connected machine using a simple VNC viewer, and it saves state and configuration.

2.3 Recording VNC sessions

We considered several tools for recording VNC sessions, and we have included an overview of their features.

2.3.1 RFB Proxy

RFBProxy is a program for recording, playback, and video conversion of VNC screen sessions, for later replay to a VNC client or export as a series of PPM frames. The tool acts as a proxy between the VNC viewer and the VNC server, and relays messages between the two. The tool offers several recording modes.

Running with the simple record option set, will wait for a VNC client to connect to localhost:10, then connect to localhost:0 and relay all VNC traffic between them, recording the screen updates in FBS 1.0 format to session.fbs and exiting when the client disconnects.

```
# rfbproxy -r session.fbs
```

Running it with the `s` option tells it to connect to localhost:0 as a shared session with the client and record the screen updates in FBS. The user also has the option of specifying a different server and display through the server=x.x.x.x:0 parameter.

```
# rfbproxy -rs session.fbs
```

By default, the `type` parameter sets the recording mode and is by default set to capture screen updates. However, when `type=events` it will record keyboard/mouse events into a RFM (remote framebuffer macro) file. This RFM file can later be played back with tools such as rfbplaymacro to another VNC server display.

The next step is exporting the recorded screen session as a series of PPM frames (-x option), to allow for conversion to video using Linux tools such as mjpegtools, ffmpeg, and/or ffmpeg2theora.

Convert the VNC session in session.fbs to MPEG-2 using mjpegtools

```
# rfbproxy −x session.fbs | ppmtoy4m −S 420_jpeg | mpeg2enc −o video.mpg
```

Convert the VNC session in session.fbs to MPEG-4 using ffmpeg (ppmtoy4m is from mjpegtools).

```
# rfbproxy −x session.fbs | ppmtoy4m −S 420_jpeg | ffmpeg −f
yuv4mpegpipe −i −−vcodec mpeg4 video.avi
```

2.3.2 Vncrec

Vncrec is used to both record and play back a vncrec session. Vncrec adds three more parameters to Vncviewer's options for recording and playback of vncrec sessions:

- -Record filename
- -Play filename
- -Movie filename

Record will record the VNC session and name it whatever the provided filename. A vncrec recording session can be ended by pressing F8 and selecting exit, closing the vncviewer window, or sending Ctrl-c in the terminal where vncrec was executed.

The play option plays a recorded VNC session. The movie option will extract each frame of the recorded session out to an .xpm file. The following commands starts recording a VNC session:

```
# vncrec −record filename servername
```

To play the recorded session through vncrec:

```
# vncrec −play session.vnc
```

Sessions recorded with vncrec can also be exported to video. This requires help from tools such as transcode. The basic syntax for transcoding a vncrec session into an avi file is:

```
# transcode −i session.vnc −g 800x600 −y xvid −o session.avi
```

The -i switch designates the input file and -g declares the video stream frame size, which in this case is 800x600. The -y switch declares the video export module, in this case xvid. The output file, designated by -o, is session.avi.

2.3.3 VNC2Swf

Vnc2swf is a tool that is available in Python and C for recording of VNC sessions to ShockWave Flash (swf) or Flash Video (flv) format. It is distributed under the GNU GPL and can be downloaded from `http://www.unixuser.org/~euske/vnc2swf`. It acts as a VNC client and communicates directly with a VNC server.

It runs in two different modes: GUI (Graphical User Interface) mode and CLI (Command Line Interface) mode. In the GUI mode, recording can be started by clicking the “Start” button, and saved by choosing “Save as...” from the “File” menu after the recording has been stopped. In the CLI mode, a user needs to specify the output filename from command line to start recording immediately, and hit Control-C to stop recording. If the server requires authentication the user will also be prompted for a password. A user can choose three different methods to encode movie image: “flv”, “swf5”, “swf7”, “mpeg” (PyMedia required), or “vnc”.

Recording a VNC screen session:

```
# vnc2swf.py −o session.flv localhost:1
```

2.4 Video Processing

In this section, we provide an overview of the tools that we used to work with the recorded VNC sessions. We needed a good video processing library and a framework for performing Optical Character Recognition on the video frames, so we considered OpenCV and several OCR tools.

2.4.1 OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision. It is released under a BSD license, so it is free for both academic and commercial use. It has C++, C, Python and soon Java interfaces running on Windows, Linux, Android and Mac. The library has more than 2500 optimized algorithms and is used in areas such as facial recognition systems, human computer interaction, mobile robotics, motion understanding, object identification, and segmentation and recognition.

2.4.2 OCR and Tesseract

Optical Character Recognition (OCR) is the translation of images containing handwritten, typewritten, or printed text into machine-encoded text documents. It was first implemented and patented in the form of a machine by a German named Gustav Tauschek in 1929, followed by an American patent in 1935. Applications of OCR include the postal service, language translation, and digital libraries, and mobile applications.

Tesseract is an open source OCR Engine that was originally developed by Hewlett Packard from 1985 to 1996. The code has now been released under the Apache 2.0 license as open source and its development is currently being sponsored by Google, which has benefited extensively from using it in their Google books project. Tesseract is considered one of the most accurate free OCR engines available [10].

The Tesseract OCR algorithm follows a step-by-step pipeline. First, a grayscale or color image is provided as input. To allow for good results, the image should be flat, (as supplied from a flatbed scanner or a near parallel image capture), as opposed to one that contains perspective distortions. Next, the algorithm performs adaptive thresholding using Otsu's method. Otsu's method is used to automatically perform histogram shape-based image thresholding, or reduce a graylevel image to a binary image. It assumes that the image to be threshold contains two classes of pixels (foreground and background), and then calculates the optimal threshold that separates the two pixel classes such that the variance between the two is minimal.

Following the initial setup Tesseract performs a connected component analysis, during which it searches through the image, identifies foreground pixels, and stores their outlines as blobs (potential characters). Blobs are organized into text lines, by analyzing the image space adjacent to potential characters for fixed pitch or proportional text. Text lines are broken into words differently according to the kind of character width. Fixed pitch text is chopped immediately by character cells, whereas proportional text is broken into words using definite spaces and fuzzy spaces. This is followed by a two-pass recognition process. During the first pass, the algorithm attempts to recognize each word in turn, and each satisfactory word is passed to an adaptive classifier as training data. This allows the adaptive classifier to continuously improve its text recognition ability as it goes down the page. The second pass is run over the page to make sure that the adaptive classifier makes a better contribution near the top of the page after it might have learned something useful from the training data. The final phase of the algorithm resolves fuzzy spaces, and checks alternative hypotheses for the x-height to locate smallcap text [26].

3 Methodology

In this section we start with an overview of the environment and basic infrastructure needed for the project. We describe the methods and technologies used to analyse the video, some of the challenges we faced in the process, and describe the interface we used to present the results.

3.1 Environment Setup

For this project we needed a server and a client machine that had a reliable network connection to each other, and all the necessary applications and libraries to support the VNC infrastructure and the insider threat analysis phase. We decided to achieve this through two Linux virtual machines (one server and one client) running on a KVM virtualization server. The virtual machines had the following specifications:

	Server	Client
OS	Ubuntu 12.04	Ubuntu 12.04
Memory	2GB RAM	1GB RAM
Capacity	30GB	10GB
CPU	2 cores	1 core

Table 1: Server and client specifications

3.1.1 VNC Infrastructure

First, we needed to install and configure the VNC server on the designated Ubuntu server virtual machine. We chose to use vnc4server which is available through the Ubuntu package manager. For a better virtual desktop environment we also chose to install gnome-session on the server and configure it to be the default windowing system for VNC.

We updated the package list to the latest 12.04 releases, and continued with the VNC and Gnome installs.

```
# sudo apt-get update
# sudo apt-get install gnome-core gnome-session-fallback
# sudo apt-get install vnc4server
```

After successful installation, we ran the VNC server with the default configuration in order to configure a password for the default display, and edited the server config file to customize it for our needs.

```
# vncserver
# vncserver -kill :1
# nano .vnc/xstartup
```

config file after customization:

```
#!/bin/sh
unset SESSION_MANAGER
gnome-session session=gnome-classic &
[ -x /etc/vnc/xstartup ] && exec /etc/vnc/xstartup
[ -r $HOME/.Xresources ] && xrdb $HOME/.Xresources
```

```
xsetroot −solid grey
vncconfig −iconic &
```

In order for the changes to take effect, the VNC server was restarted with the desired screen size depth. The following command starts the server with a 1024x768 resolution and a 24 bit color mode.

```
# vncserver −geometry 1024x768 −depth 24
```

Finally, in order for the VNC server to automatically start at system startup, we added aforementioned command to the command script for system startup.

```
# sudo nano /etc/rc.local
add '/usr/bin/vncserver −geometry 1024768 −depth 24' to the end of the file
```

3.1.2 VNC Session Recording

Following the installation and configuration of the VNC server, we needed a method of recording sessions in a format that could be easily played back. We tested all of the VNC session recording tools mentioned in the background chapter for this task.

To test RFBProxy, we began by recording a sample VNC session and saving it to the FBS format. We replayed this session using a VNC client, following which we converted from the FBS format to a playable video. The conversion was accomplished by using RFBproxy to push PPM frames, which were then converted to tools such as as mjpegtools, ffmpeg, and/or ffmpeg2theora. We chose not to use RFBproxy because it took a long time to convert the PPM frames, and it failed to produce a working video. We also tried vncrec without success because it failed to connect to the VNC server and record the session.

Ultimately we chose to use vnc2swf because it was successful in recording a session and it had various advantages over the alternative tools. One of these advantages was that it provided the option of recording the session in different formats (flv included), so no time was spent converting to video. Additionally, vnc2swf also allowed us to choose the number of frames per second the video was recorded. This was especially helpful since it made it possible to reduce the number of frames that had to be analyzed, thereby improving overall performance.

3.2 Session Video Analysis

This section describes in detail how we analysed the video to detect scenes changes and remove redundant frames and performing OCR on the frames selected from each scene.

3.2.1 Pre-processing the video

After recording the VNC session, we used OpenCV to open the flv file and split it into frames for further analysis. The framed are stored in memory as IplImage, a format which was inherited from the Intel Image Processing Library. The IpllImage stores image attributes such as the number of channels, origin, width, height, and depth.

In order to remove redundant frames, we implemented a scene detection algorithm. In the algorithm, the difference between subsequent frames is stored in a buffer and compared to the mean of the most recent differences. If the difference is greater than several deviations above the mean, we flag the frame as indicating a scene change. Since there is no way to take the difference between IplImages, we had to convert from the IplImage format to a matrix of pixels which are represented as RGB tuples. The scene detection algorithm is similar to the Frame Difference Analyzer tool, developed by Alex Arsenovic and based on Bollinger Bands [1].

Subsequently, we performed several tests to determine the number of frames that should be analyzed per scene. We concluded that selecting 2% of each detected scene would give us an appropriate coverage of the most important frames in the video.

3.2.2 Extracting text using OCR

For the task of extracting text from the user session video, we considered several OCR tools such as OCRopus, GOCR, and Tesseract. We tested the tools by using each one to OCR a set of screenshots, and qualitatively determining how well each performed. In order to test OCRopus, we ran several scripts that were provided with the application. These scripts performed noise and border removal, grayscale normalization, column finding, text line finding, language modeling, and outputted the results to HTML format. OCRopus was able to recognize certain documents well (such as a high scale scanned newspaper image), but the script provided for recognizing lattices continually failed while recognizing the screenshots. In order to test GOCR, we used the application gscan2pdf which allows the user to choose an engine with which to perform OCR. GOCR was able to recognize some characters, but the output was not up to the standards needed for our application. Ultimately, after performing several tests and tweaking the variables that we thought would contribute to the OCR results, we found that Tesseract was the most promising library and as such we decided to use it as our engine.

In order to improve Tesseract's performance, we tried several adjustments including changing the operating system fonts, training it with Ubuntu Fonts, and adjusting the desktop's contrast. Changing the font and other desktop settings had little effect on the results, and often times the performance would be worse with the changed fonts. Training Tesseract to recognize the Ubuntu fonts also failed to improve the results considerably. Additionally, these fixes seemed like temporary solutions since the system would not be very secure if changing fonts degraded performance. Another possible approach would have been to use edge detection as a means of identifying different windows in a frame and thus improving the significance of the OCR results.

Following suggestions from online communities of Tesseract users and the Tesseract project page, we decided to add a pre-processing phase to our process, in which each frame is scaled by a factor of 2, and pixels below a certain threshold are removed. The transformations were accomplished using OpenCVs resize and threshold functions. As for the parameters (i.e threshold, and scaling factor), we made an initial estimate, and tuned them according to quality of the output. The OCR results seemed to improve whenever we scaled the images up but it also took a much longer time to complete the recognition. We found that scaling the images by more than a factor of four did not improve the performance significantly enough to warrant the increase in recognition time. Threshholding the image improved the OCR results slightly and did not cause the recognition time to increase significantly.

In addition to performing well, Tesseract had a good advantage over the other tools because there were Python libraries that allowed us to interface easily with OpenCV. After the parameters were tuned the scene detection and OCR portion were combined and tested.

3.3 Dictionary matching

We considered several tools to aid in analyzing the text extracted from the frames in the previous phase. The tools included Yahoo's Term Extraction web service and the Python library Natural Language Processing Toolkit (NLTK). Yahoo's Term extraction service allows a user to grab key phrases from a given text by submitting the text via a POST request to the web services URL [13]. The idea was that these keywords would serve as the subject of each frame, and by comparing them to dictionaries containing finance words we could determine whether a particular frames subject was finance and as such deem it sensitive. Unfortunately, the web service has a rate limitation of 5000 requests per day for non-commercial users. NLTK is a suite of open source python modules which can be used to perform natural language processing tasks such as clustering and classifying documents [22]. The classification package provides methods for assigning labels to text which could be used to classify a set of documents by subject. The classification is done by first extracting features, which are identifying aspects of a text. A classifier (naive Bayes, SVM) is then trained and used to label new documents. For our purposes we were planning to gather a number of different financial documents from various sources on the web and use them to train a naive Bayes model. This model would then be used to classify the extracted texts as either finance related and therefore sensitive or not.

Ultimately, we chose to use a simple algorithm that utilizes the frequency of certain words to determine whether a particular frame is sensitive. In order for this to work, we built a dictionary by scraping Yahoo's finance glossary using a simple python script.

The algorithm works as follows:

```
D = Load Dictionary from File
```

```
 For each Word in the Text
        If Word in D
            increment Unigram Count
For every two Words in the Text
        if the two words in D
            increment the Bigram count
For every three Words in the Text
        if the three words in D
                increment the Trigram count
Metric = (Unigram Count)/(total words)
                + (Bigram Count)/ (Total Words  1) + (Trigram Count) /(Total Words −2)
```

We ran the algorithm on text extracted from a test video in order to determine what threshold to use in determining whether a text was finance related.

Due to the noisy output produced by the OCR, the number of dictionary hits was quite low even when analyzing frames whose subject was finance. Further, the dictionary built was not as comprehensive as we would have wished and contained various finance colloquialisms such as “axe to grind” which complicated the analysis. Future work should focus on improving the dictionaries or considering a different approach to the classification.

3.4 Process information

We decided that we needed to augment the video analysis with system level information so as to provide a full picture of the system when a sensitive frame is recording. For this task, we wrote a script that called “ps -aux” every X seconds and stored the result in a local sqlite3 table. The reason we chose to use the sqlite3 as opposed to saving the output in one file was that it provided us with a way to retrieve specific sections of the output without going through the whole file (i.e. SQL). Moreover, we plan to improve the application handle the analysis of many users' sessions. This will require parallelization which would be made easier by using a database albeit one that with more features than sqlite3. The table was organized such that it contained three fields, "id", "time", and "running". In the application, when a frame has been identified as being sensitive, processes that were running at the time the frame was recorded are queried from the table and used to build the JSON string needed for the user interface.

3.5 Interface

A very important aspect of our project was the presentation of the results obtained from the analysis of the session videos. We wanted to achieve this through an interface that would highlight the most sensitive parts of a user session, along with the closed caption (obtained through OCR), and the information about system processes active at that time. We decided to do this using a web interface built with HTML5, CSS and several JavaScript libraries. Figure 2 shows a mockup of the interface.

The timeline is where all the sensitive portions of the user session will be highlighted, in the order that they happen. Upon selection of one of these frames by clicking on them, the upper area of the interface is populated with information specific to the selected frame. This information consists of the actual frame shown on the left side of the interface and expanded when clicked on, the closed caption for this frame, and the processes that are running on the system at that selected time.

The implementation of the interface turned out to be easy with the help of already available JavaScript libraries (based on jQuery) that could be adapted to satisfy our requirements. TimelineJS [29] is a JavaScript library provided under the MIT license that can create intuitive media-rich web timelines. The library can pull data from Google spreadsheets or detailed JSON documents. We made minor changes to the look and functionality of the interface to support our requirements, and decided to provide the data through a JSON document because it would be easier to integrate with the rest of our scripts. Figure 3 shows the format and the JSON representation of an object that describes a frame from the user session.

The frame object is named "date" and has a field for the date and time the frame belongs to (startDate), a "headline" field for a frame title, a "text" field for the closed caption, a "process" field for the process information and an asset object which describes the media type (a picture in this case). Figure 4 shows the final look of the interface. We used the Lightbox2 [9] library to create

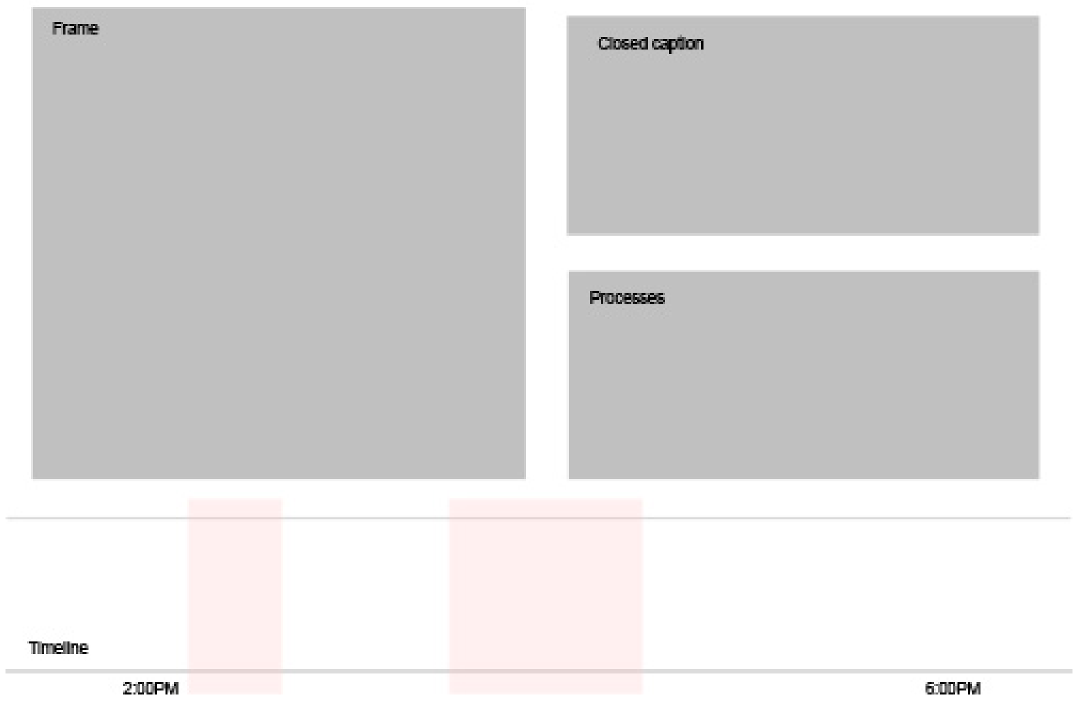
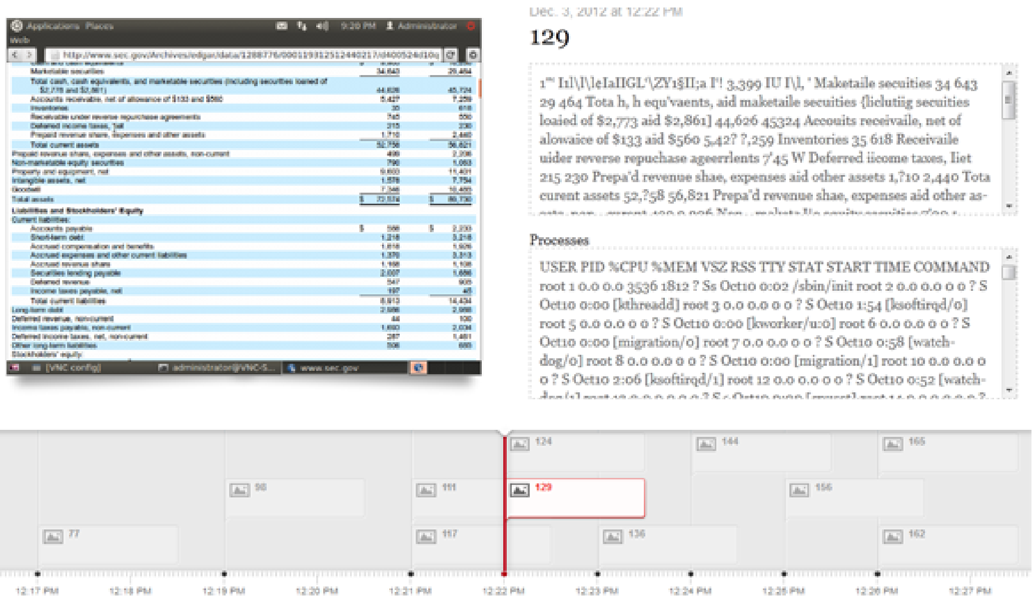


Figure 2: Mockup of the interface for the session viewer

```
{
    "timeline":
    {
        "headline":"VNC Session",
        "type":"default",
         "text":"VNC session example",
         "startDate":"2011, 12, 27, 15, 20",
        "date": [

            ...

              {
                  "startDate":"2011, 12, 27, 17, 23",
                  "headline":"Desktop",
                  "text":"Closed caption text",
              "process":"Process information",
                  "asset":
                  {
                      "media":"frame.jpg"
                  },


           ...

               ]
}
```

Figure 3: JSON format


Dec. 3, 2012 at 12:22 PM

129

1™ I1l\I\leIaIIGL'\ZY1§II;a I'! 3,399 IU I\l, ' Maketaile secuities 34 643 29 464 Tota h, h equ'vaents, aid maketaile secuities {liclutiig secuities loaied of $2,773 aid $2,861] 44,626 45324 Accouits receivaile, net of alowaice of $133 aid $560 5,42? ?,259 Inventories 35 618 Receivaile uider reverse repuchase ageerrlents 7'45 W Deferred iicome taxes, Iiet 215 230 Prepa'd revenue shae, expenses aid other assets 1,?10 2,440 Tota curent assets 52,?58 56,821 Prepa'd revenue shae, expenses aid other as-

Processes

```
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND root 1 0.0 0.0 3536 1812 ? Ss Oct10 0:02 /sbin/init root 2 0.0 0.0 0 0 ? S Oct10 0:00 [kthreadd] root 3 0.0 0.0 0 0 ? S Oct10 1:54 [ksoftirqd/0] root 5 0.0 0.0 0 0 ? S Oct10 0:00 [kworker/u:0] root 6 0.0 0.0 0 0 ? S Oct10 0:00 [migration/0] root 7 0.0 0.0 0 0 ? S Oct10 0:58 [watch-dog/0] root 8 0.0 0.0 0 0 ? S Oct10 0:00 [migration/1] root 10 0.0 0.0 0 0 ? S Oct10 2:06 [ksoftirqd/1] root 12 0.0 0.0 0 0 ? S Oct10 0:52 [watch-
```

Figure 4: Final interface

the lightbox effect that expands the frame image on click.

4 Results

There were no similar products against which we could compare our application so in order to gauge the performance we measured time take to complete the various stages and the accuracy of frames as sensitive or otherwise. The overall time was measured with respect to session length and the results have been summarized in table 2. We found that ratio between time it took to analyze a session to the actual length of the session was quite high. For instance, it took 20 minutes for the application to analyze a 4 minute long session for a ration of 5 to 1. The time to analyze a session grew relatively linearly with respect to the length of the video. It should be observed however, that if a really long video contained few scene changes then the time to analyze such a video would not be very long since OCR was the clear bottleneck in the system. In fact, for the 16 minute session scene detection reduced the number of frames that had to be analyzed by over 90% and since performing OCR on a single frame could take up to 48 seconds, this resulted in a huge performance improvement.

The performance of the OCR was related to the resolution of the recorded session, and the constants used during the preprocessing phase of the frame. We found that that scaling the images by some factor improved the detection rate, but also increased the processing time. For our tests, we used a scale factor of four which resulted in a detection rate of about 50%.

To test the number of false positives and negatives the application generated we used a 16 minute long session during which various normal activities such as exploring file-system and setting up meetings with colleagues, and various sensitive activities such as viewing and sending earnings reports were performed. In the 16 minute long video, 10 frames were identified as sensitive, all of which were earnings reports. It took about 2 hours to analyze all of the frames where most of the time was spent performing OCR on the frames. After cleaning up the dictionary, there were very few false positives which was an expected result since it contained very specific terms.

The system was able to identify the earnings report as sensitive though this was not surprising since the report contained so many words from the dictionary. However when frames were not overwhelmingly about finance, the application was not able to flag them as sensitive.

Length (minutes)	Time for full analysis (minutes)	Avg OCR time (seconds)
4	19.57	11.33
6	40.06	24.32
16	106.51	16.07

Table 2: Video analyis results

5 Discussion

One of the main results of this project was showing that the approach of using high level analysis (i.e. OCR) of recorded sessions is a viable alternative to using system calls in order to mitigate insider threat problem. However, many improvements still need to be made if this application is to be used in a real work environment. These improvements mainly have to do with improving the way frames are classified as sensitive and the way many users with long sessions can be handled efficiently.

Moving forward, one possible improvement we are considering is changing how frames are flagged as sensitive by using different classification algorithms such as SVMs and Naive Bayes, which have been shown to be successful in various classification domains. Additionally, we plan on improving the classification by using computer vision techniques such as edge detection and object detection. Using edge detection would entail applying one of OpenCV's edge detection filters to get the location of window borders and running OCR to grab text from individual window. Similarly object detection would involve leveraging OpenCV to detect whether certain symbols (i.e. application icons) are present in the frame. In addition to these techniques, classification could be improved by better incorporating the system level information such as which files are accessed and what system calls are invoked. Currently, the application keeps track of the processes that were running when a frame was recorded, but the information is not used to flag the frame.

For this application to be deployed in a real enterprise environment, it needs to be able to handle many user sessions, which could be hours long, efficiently. This could be done by distributing the OCR and classification work load across several computers. Additionally the application would need a configuration method which would allow a system administrator to tune various aspect such as OCR settings and number of frames analyzed per scene. Finally, the system could also benefit from having an easy way of switching domains (i.e. from finance to medical or government work).

Overall, the application we presented in this paper provides a good basis upon which interesting approaches can be developed. Various aspects such as the classification of frames and the general scalability need to be improved but our results suggest that the approach is promising.

6 Conclusion

In this project, we explored a new approach to identifying insider threat in virtual desktop environments by analyzing recorded user sessions. The insider threat problem has been a prevalent one and has been continuously listed as one of the hardest challenges in cyber security. Prior research has approached the problem by profiling user behavior using machine learning techniques, or using trap based techniques. In many of these publications, user behavior is represented by a sequence of system calls or other low level system events.

However, low level system events do not always accurately characterize user behavior and they only approximate the user's intent because they may be a side effect of applications. Additionally, it is difficult to generate realistic insider threat datasets.

In order to address the shortcomings of these established approaches, we collected not just background information, but a video capture of events and actions taken by the user. Using scene detection algorithms we compartmentalized the capture, partitioning it into a limited number of frames from each scene. We extracted text from the selected frames and used it to classify the user actions as sensitive or normal. The sensitive portions were then presented in a web interface in the form of a timeline where each entry in the timeline contained the frame, the text extracted from the frame, and the processes running at the time the frame was recorded.

Our work gave promising results and showed that using recorded user sessions could be a possible alternative to the previous approaches. Further work is necessary to improve the quality of the frame classification and performance of the entire process. Better results can be achieved by leveraging computer vision methods such as edge detection, using machine learning techniques to classify the importance of the text extracted from the frames, and implementing some of the other suggestions discussed in Section 5. Finally, we believe that the techniques explored in this project make a good contribution to the ongoing research that is trying to address the insider threat problem.

References

[1] Alex Arsenovic. Frame difference analyzer. This is an electronic document. Date of publication: [Date unavailable]. Date retrieved: November 28, 2012. Date last modified: [Date unavailable].

[2] B.M. Bowen, M. Ben Salem, S. Hershkop, A.D. Keromytis, and S.J. Stolfo. Designing host and network sensors to mitigate the insider threat. *Security & Privacy, IEEE*, 7(6):22–29, 2009.

[3] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[4] D. Caputo, M. Maloof, and G. Stephens. Detecting insider theft of trade secrets. *Security & Privacy, IEEE*, 7(6):14–21, 2009.

[5] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[6] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods.* Cambridge university press, 2000.

[7] A. Cummings, T. Lewellen, D. McIntire, A.P. Moore, and R. Trzeciak. Insider threat study: Illicit cyber activity involving fraud in the us financial services sector. 2012.

[8] L. De Ferrari and S. Aitken. Mining housekeeping genes with a naive bayes classifier. *BMC genomics*, 7(1):277, 2006.

[9] Lokesh Dhakar. Lightbox2, 2012.

[10] Google. Tesseract-ocr.

[11] C.M. Grinstead and J.L. Snell. *Introduction to probability.* Amer Mathematical Society, 1997.

[12] N. Grira, M. Crucianu, and N. Boujemaa. Unsupervised and semi-supervised clustering: a brief survey. *A Review of Machine Learning Techniques for Processing Multimedia Content, Report of the MUSCLE European Network of Excellence (FP6)*, 2004.

[13] Yahoo! inc. Term extraction documentation for yahoo! search, 2012.

[14] W.H. Ju and Y. Vardi. A hybrid high-order markov chain model for computer intrusion detection. *Journal of Computational and Graphical Statistics*, 10(2):277–295, 2001.

[15] M. Keeney and United States. Secret Service. *Insider threat study: Computer system sabotage in critical infrastructure sectors.* US Secret Service and CERT Coordination Center, 2005.

[16] Rachael King. Emc's rsa security breach may cost bank customers $100 million.

[17] D. Maughan et al. A roadmap for cybersecurity research. *US Department of Homeland SecurityNovember*, 2009, 2009.

[18] R.A. Maxion and T.N. Townsend. Masquerade detection using truncated command lines. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on Dependable Systems and Networks*, pages 219–228. IEEE, 2002.

[19] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48, 1998.

[20] T.M. Mitchell. *The discipline of machine learning.* Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

[21] Sunshine Press. Wikileaks, 2012.

[22] NLTK Project. Natural language toolkit, 2012.

[23] R.R. Rantala. Cybercrime against businesses, 2005. 15(14):9, 2008.

[24] M.B. Salem, S. Hershkop, and S.J. Stolfo. A survey of insider attack detection research. *Insider Attack and Cyber Security*, pages 69–90, 2008.

[25] J. Seo and S. Cha. Masquerade detection based on svm and sequence-based user commands profile. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 398–400. ACM, 2007.

[26] R. Smith. An overview of the tesseract OCR engine. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 629–633. IEEE, 2007.

[27] L. Spitzner. Honeypots: Catching the insider threat. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 170–179. IEEE, 2003.

[28] B.K. Szymanski and Y. Zhang. Recursive data mining for masquerade detection and author identification. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pages 424–431. IEEE, 2004.

[29] VeriteCo. Timelinejs, 2012.

[30] K. Wang and S.J. Stolfo. One-class training for masquerade detection. In *Workshop on Data Mining for Computer Security, Melbourne, Florida*, pages 10–19, 2003.

[31] K. Yung. Using self-consistent naive-bayes to detect masquerades. *Advances in Knowledge Discovery and Data Mining*, pages 329–340, 2004.