Fall 2007 CS 261/294 Project Proposal
Identification of Sensitive Information

John Bethencourt Steve Hanna

Motivation and Goals

The goal of our project is to design and implement a system for identification of sensitive information within the persistent state of a user's system. The ability to determine whether a particular file or, say, Windows registry key contains personally identifying or otherwise confidential information is useful in a variety of contexts. Viewing a list of the files flagged as sensitive may assist the user in setting file system permissions, or this information may be used by systems performing detailed analysis of sensitive information flow or leakage.

In the past, rudimentary methods have been employed for this purpose. Tightlip, which attempts to track leaks of sensitive data, uses a set of ad hoc heuristics based on file names and content to determine which program inputs are sensitive (for example, a file that has the extension ".pst" or contains the string "Message-ID" is likely a saved email). However, such approaches are difficult to make comprehensive, as there will always be applications unknown to the developer of the heuristics.

Our approach is intended to provide a more complete accounting of sensitive information stored within a user's system, without resorting to a large set of manually written heuristics. We do, however, limit the scope of our task to tracking sensitive data stored and manipulated by benign software. Malware on the user's system which actively attempts to hide sensitive information from our system will easily be able to do so. Nevertheless, we believe a system which accurately tracks the location of sensitive information stored by legitimate software will be highly useful.

Technique

Classifying a piece of data already stored on a user's system as sensitive or non-sensitive is a difficult task. We base our approach on the following insight: it is easier to identify sensitive information as it *initially enters* the system than it is to classify stored data of unknown origin. If we assume that our system is present on the user's machine when it is freshly installed (and therefore free of sensitive information), then we do not need to classify any data already present on the machine and instead only need to keep track of new sensitive data which enters it.

The primary channels by which sensitive information is initially introduced to the system are the keyboard and network.[1] It may be generally assumed that any text typed in by the user is sensitive, and data received from the network may be classified with a simple port-based policy. For example, any data received as a result of an outbound connection on ports 80 (HTTP), 110 (POP), 21 (FTP), etc. might be considered sensitive.

When our system detects through monitoring of keystrokes and network traffic that an application has received such sensitive data, we must determine whether this data or something derived from it is later stored in the persistent state of the machine. In order to achieve minimal performance impact on the user's system, we will not perform detailed taint tracking as in, for example, Panorama. Instead, we take a rough, coarse grained approach based on text string matching.

Specifically, our system will assemble a list of strings typed in by the user and received over the network and remember which process received them. When that process (or one of its children) later writes to a file (or some other part of the machine's state), our system will be invoked and will try to determine if any of those strings have passed into the file, in which case it will be flagged as sensitive. We will be developing heuristics (e.g., the length of the greatest common substring) which will be used to make that determination while attempting to minimize false positives and false negatives. If another process later reads a file marked sensitive and writes to a second file, our system will check the second file for the strings which originally caused the first to be flagged, propagating the taint if necessary.

[1] Other, more subtle possibilities exist, but we will focus our attention on these.

Apart from very low performance overhead, an advantage of our architecture is that it does not require running the user's system in a virtual machine and may furthermore operate entirely in user space. It will be possible to have our system either target a specific application or operate system wide by injecting our hooking mechanism into all running applications. More specifically, as implemented on Windows, our system may operate by hooking the entry points into the **SSDT** (System Service Descriptor Table, the equivalent of the Linux syscall table) using a library such as Detours (*http://research.microsoft.com/sn/detours/*). Due to the tractable number of system calls, we can examine the inputs to all the functions that write to the file system or registry and determine if the buffers passed contain any previously recorded tainted information.

Challenges and Evaluation

Due to the compromises made in our design, a number of challenges exist in achieving high accuracy in tracking the sensitive information stored on the machine. While most sensitive information is textual, there are some exceptions, such as images downloaded by a web browser. More importantly, even in the case of textual data, various transformations may take place between its introduction into a process and being written to storage. Text typed into a word processor may be compressed, encrypted, or differently encoded (e.g., as UTF-16) in the document later saved. Text arriving over an SSL connection may be later stored in unencrypted form by the web browser. Another challenge is the possibility of information flowing between processes through IPC or some other mechanism. We plan to investigate ways to extend our basic design to address some of these situations, however, we feel that a system which only handles the simplest case will still provide useful and interesting information.

A couple approaches are possible for evaluating the accuracy our system when it has been implemented. We may run several applications under our system as they are installed, are operated, and communicate over the network. The resulting set of files flagged as sensitive may then be compared with a comprehensive, manual investigation of all files written by the software to determine whether any false positives or false negatives occurred. A somewhat more involved approach would be to also run the applications under Panorama in order to do a fine grained taint analysis, the results of which may be compared with our system's output. In addition to evaluating the accuracy of our system, we also intend to characterize its performance and compare it to the performance of a more accurate, fine grained taint tracking system such as Panorama.