HoBIDS: A Data Mining Approach to Host Based Intrusion Detection

Department of Computer Science, Columbia University
CUCS Technical Report:

Shlomo Hershkop, Frank Apap, Eli Glanz, Tania D'alberti,
Eleazer Eskin, Sal Stolfo, Johnee Lee
{shlomo, fsa3, deg28, tfd4,
eeskin, sal, jl1025} @cs.columbia.edu

Nov 2001

Abstract

We introduce HoBID (Host Based Intrusion Detection), part of a larger Columbia University data mining based intrusion detection system (Columbia IDS). The HoBID approach is a practical realization of a host based IDS built on a data mining framework. This framework allows the system to detect both known and unknown intrusions in real time [3].

The HoBID system is being developed in conjunction with the Columbia HAUNT (Heuristic audit of Network Traffic) project, the network part of the IDS. The HoBID approach was to design a lightweight modular sensor system with minimal overhead and maximum fault tolerance.

In addition we discuss deployment of HoBID on both Microsoft Windows NT 4.0, Microsoft Windows 2000, Linux, and Solaris.

1 Introduction

Intrusion detection can be accomplished on two different levels; the host and network levels [5]. While the use of network intrusion detection systems is integral to the protection of data systems, no protection scheme is complete without the use of host-based tools.

The compromising of a host's operational integrity represents a very serious threat in today's computing world. Whether remotely via denial-of-service (DOS) attack, locally through malicious user interaction, rogue software or any other attacks, modern computers are under constant threat [4].

Any interruption of service or misuse of resources can cause significant operational and financial damage to today's computer users. The recent high-profile DOS attacks on the web services of Yahoo!, CNN and other large corporations only serve to underscore the need for protection against host intrusion.

While the vast majority of the current attacks occur via a network, usually over the public Internet, almost all attacks must originate from a host or group of hosts. Either malicious users initiate the attacks directly from their own systems, or more likely, commandeer resources from numerous compromised hosts. A smaller number of attacks, do not travel over a network but originate and remain on a host system (self host attacks).

Conventional tactics often call for the use of a network-based intrusion detection system (IDS) for the detection of malicious attacks. Besides the inability to capture self host attacks, network based intrusion detection have several other drawbacks. The first problem is that network data represents user actions, but not necessarily the consequences of those actions. The second problem, is the large volume of traffic present on modern networks, which would overwhelm all but the costliest of collection and analysis systems. The final, and perhaps gravest problem is that even if all traffic could be captured, all activity may not be detectable. Data traveling over encrypted channels, connection-less traffic, back-doors in legitimate protocol traffic, and incorrect or fragmented packets may evade even the most sophisticated of network-based tools. The system presented here aims to complete the IDS picture by operating on the host level.

Host-based intrusion detection allows one to observe all user activity on a host as well as the consequences those activities. In addition few pieces of data are ever completely inaccessible due to encryption or fragmentation. Finally as data collection and analysis is performed on a per-host basis, overwhelming data flows are rarely a concern. The Host Based Intrusion Detection (HoBID) system presented here combines a robust and lightweight sensor system with the capabilities of sophisticated data-mining techniques to build a powerful host-based IDS.

2 Current solutions

In general, computer security has been implemented as an after thought into computer systems [3]. Many different approaches to add security to current systems have been suggested and implemented on many different platforms and applications. Usually the solution involves defining the notion of a security violation, encoding these ideas for the computer to understand, and then waiting for a match to occur. The two main paradigms are misuse detection and anomaly detection [6].

In misuse detection, security violation rules are generally encoded as hand crafted signatures by a manual encoding of expert knowledge. The advantage to such a system is the low rate of false positive alerts, that is if an alert is generated we can assume that it was created because it matched (exactly) a past attack and that it is the same attack. the drawbacks is that assuming such a system would always work perfectly, new and therefore unknown security violations are obviously a problem. The system simply cannot generalize its known rules, to apply them to new types of violations.

In anomaly detection, users or programs on a computer system are profiled using statistical methods to create normal usage patterns. Then the system watches for some threshold to be passed before issuing an alert. The draw back is that you need to create clean training periods and anything not seen in training will be flagged as a violation. In addition there is a problem with the high rate of false positives which are generated when normal usage is not profiled correctly.

On host based platforms, for either single or multi user type operating systems, there exists no "one size fits all" solution for IDS. Some of the popular commercial solutions which have been put out in the last few years will be showcased as case examples. This is not a criticism on any one solution just a small very brief overview of the state of the art on host based IDS. The list is short because for a long time many people felt that one could only do IDS on a network based view.

BlackICE defender by NetWorkICE [http://www.networkice.com] is set up for a both home and corporate users. It claims to do some type of protocol analysis as its IDS a form of anomaly detection. When it registers an attempted intrusion, it automatically blocks traffic from that source, as well as logs all traffic from that source. They have a distributed architecture of host audit monitors which build user and traffic profiles. Although the system seems very flashy some drawbacks do appear. By blocking traffic from IPs which register as anomalous, a system can be compromised. for example if an attacker would fake the address of the IP which provides virus definition updates or software updates, all inbound updates would be stopped allowing the attacker to use the holes or a virus to compromise the system.

Another system is NIDES by SRI international (http://www.sdl.sri.com/projects/nides/). Its an IDS system composed of a rule based part and anomaly detection part. The two parts are run over audit logs and detect intrusions when either a rule is triggered or a pattern for individual user os broken. as mentioned a drawback with rule based is that new attacks will not be flagged, and the problem with statistical patterns is the need for clean training and for normal usages. In addition as new program become available for users, and as users become more aware of what they can do on a system, in a good sense, their patterns of usage will change and be flagged as an alert.

Dragon Squire, a host based signature system by Enterasys (http://www.enterasys.com/ids/squire/) monitors system logs and SNMP traps for patterns which may indicate variety of security significant events. Its touted as a solution to Network based IDSs because it can analyze encrypted data to the host. In addition to doing host based IDS, it also does log based ids and allows the user to encode signatures.

Snort an open source [http://www.snort.org], lightweight, cross platform packet capture program with rule capacity. Because of its open source and flexible rule system, it has gained some following in recent times. It can be used as both a host based IDS or fire-wall and network based system. New rules are constantly being released and updated sometimes as fast as every couple of hours. Although it is a very flexible product, it uses user defined signatures to match alerts, and can only be used on incoming/outgoing network activity. It will not work on local malicious user activities.

An alternative solution to ID is to use snapshots of files at a known clean stages and to compare changes done to the system at some later point. Tripwire by Tripwire Inc (http://www.tripwire.com) is an integrity checker. At setup, a clean system snapshot is taken of all the files on the system using md5 hash. This represents a snapshot of the 'clean' system. At a later time using scripts md5 of certain executable programs can be compared and if there is a difference an alert can be issued. The main drawback of the system is both a need for clean snapshots and re snapping a newer snapshot every time an executable is patched. With software holes being fixed all the time, this can be a time consuming and overlooked process. A malicious user can take advantage of a system upgrade, exchange a file with a Trojan version, and slip into the Tripwire database as a legitimate file.

Veracity detects changes in file systems by creating and manipulating snapshots of directory trees. A snapshot consists of an ordinary text file that records the structure of the directory tree, the names of the files in the tree, and the cryptographic digests of the files in the tree. This snapshot can later be used to generate a list of changes in the tree since the snapshot was made. Because Veracity doesn't record the actual contents of files in its snapshots (unless requested), snapshot files are usually about 1/200th of the size of the target tree. This makes them highly tractable "representatives" of the tree.

These are some of the newer commercial solutions, many others have been proposed in the research world [][][].

A new paradigm in doing intrusion detection is to use data mining techniques to accomplish the goal [2]. Our system builds upon data mining research at Columbia University [3] to develop a more flexible and robust tool for Intrusion Detection (ID) systems. By using data mining it is hoped that the knowledge of known attacks can be generalized to allow detection of new, unseen attacks. Data mining is particularly apropos for Host Based ID as individual hosts provide for relatively controlled, predictable environments providing a rich set of data for our evaluation tools to analyze. We named our system HOBID or Host Based Intrusion Detection providing some of the coverage needed in the complete IDS picture.

3 Design

The HoBID system is composed of three main modules; sensors, collectors, and detectors. At the lowest level the Basic Auditing Modules (BAM) collect audit information. The Communication component is responsible for making sure the collected data reaches its destination. The last piece is the Learning Evaluation Engine (LEE) which reviews the collected data and tries to match models against the real time data.

3.1 BAM

In nature there is a well-known rule that for every action there is some sort of a reaction. This very much true in an operating system environment. Every action done within the system can be monitored and can potentially contain some information which can come into practical use. Collecting the information and understanding what is being collected can best be done on a low system level. Once the information exits the collection point, it might not make sense in a system-wide view of events. Although the amount of

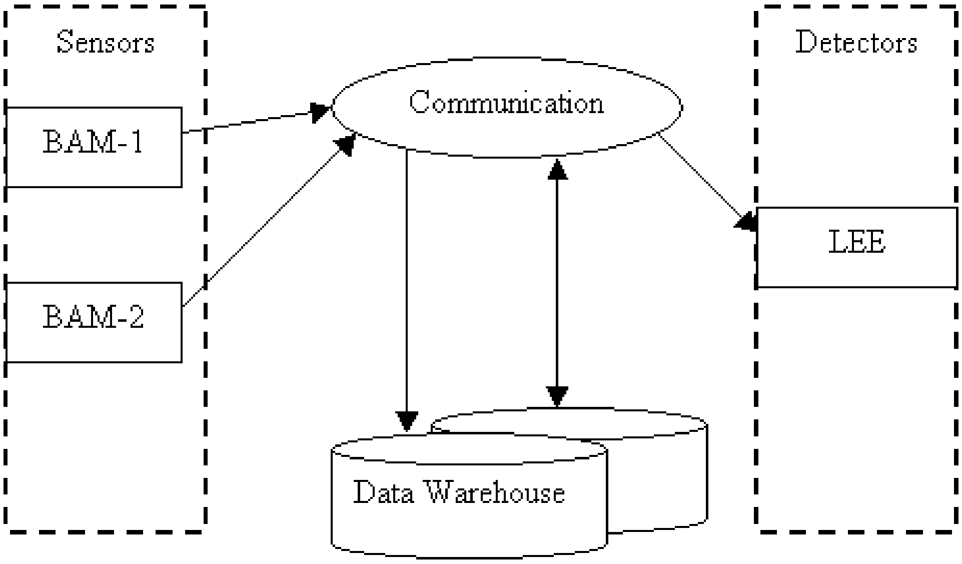

Figure 1: The Architecture of Host Based IDS

data which potentially could be collected could easily get out of hand very quickly, some planning can avoid this situation. Repeated non important data can be dropped, and any sensor which generates high volume audit, can be changed to summarize its data.

The BAM defines the low level sensor(s) present on each host system. A BAM is a lightweight program, which sits in the background of an operating system and collects audit information. Once collected, this information is minimally processed then packaged out and sent to the Communication component.

Many operating systems offer some kind of build in logging features. In some cases the information only make sense at the source of the log. By having the BAM's annotate the information at the source, higher-level processes can make sense of this information at some later point.

In a classical Server-Client view, the BAM would be a client program. Multiple BAMs reflecting a multiple view of the running system must all work side by side during normal runs. They all must be able to collect and communicate their information in a seemingly seamless way. In the same way, the BAMs must be able to be independent of each other and of the main system. BAM's should be able to turn on and off without affecting other BAMs. In addition the system performance can not be affected by any amount of running BAMs.

Using a subset of XML encoding, all information is annotated before being sent higher up in the system. In this way, all source information can be processed and evaluated in a uniform manner.

3.2 Communication

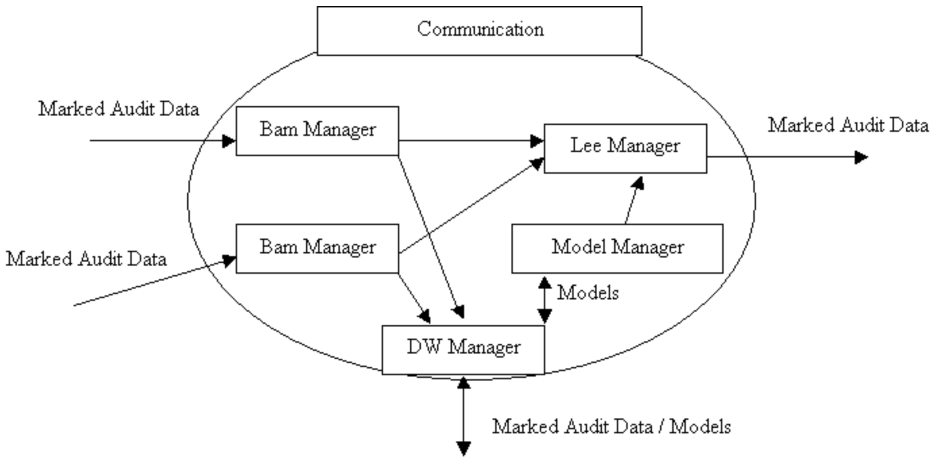

Figure 2: Communication

The next part for our system is the collectors level. The Communication component is named after its main function, a communication hub. In addition to integrating multiple BAM's into the collection system, the component is responsible in ensuring all log data is sent off-line storage and to local evaluation. Off-line storage is achieved through the Data Warehousing (DW) part of the project. The annotated audit data collected through the BAM engines, are passed to the DW, if the DW is available. At the same time all data is collected from multiple BAM's and packaged out to the real time Learning and Evaluation Engine (LEE).

The other major component within Communication is the Model Manager (MM). Model Manager is responsible for off-line storage of the attack models, and timely updates of these local models from the DW. Under the current architecture all models are stored in the DW. MM retrieves and updates the off-line model library from the DW. Each of the models have a version number associated with them. the MM is responsible for fetching updated models and distributing them to the detector(s).

At start up, the Communication component [Figure2] listens for new BAM's on a specified port. To maximize flexibility many options are loaded at run time from a predefined configuration file. For each in coming BAM the Communication component threads out a small programs which interact with the BAM's. These include a program top interface to LEE and another for passing audit information to the DW.

3.3 Lee

The Learning and Evaluation Engine forms the detector of the system. This component is responsible for real time evaluation of the models over current audit data. Because all data have a time-stamp generated when they were recorded, we can partition the information either by amount of records or at some time T. This would depend on both load of information to be processed and the nature of the models being compared. WE have our collection of models M which we can run over our current window of data W.

When a model mX matches some subset of data in the window, denoted as wx, we issue an alert. Offline alerts are issued to a local file and real-time alerts are issued to the DW. An alert contains the model, version of model, set of audit which set off the alert, host name, and other book keeping information. All this information is XML encoded for both the DW and any other component which would want to manipulate the information.

In order to address efficiency issues, the incoming XML encoded records are sorted and evaluated against the current models. As new records come in the old ones are dropped with some information being retained within the current models. Although in general the models are worked against the current window of records, they are general enough to include models based on past records. So that some of the models can encode levels of detail which are not necessarily all within one window of records.

3.4 Data Warehouse

Although the Data Warehouse (DW) is not part of the HoBID system we make reference to it [Figure1]. This is a separate project (http://www.cs.columbia.edu/ids/dw) which implements a XML based DW. It stores real-time raw audit data, models, labeled data, and anything of interest to the Intrusion Detection system.

4 Deployment

The system was designed for maximum flexibility in mind. Therefore many of the options are both configurable and also loaded up at run time from a text configuration file. In this way we could test many different ideas without having to recompile the core code.

The configuration file is a text based file. Lines starting with the Hash (#) symbol are ignored as comments. all other lines consist of a variable name followed by a space or equal sign (=) then the value it takes on. sample variables are file locations, ports to listen on, ports to connect to, location of the DW.

4.1 Windows BAM

4.1.1 BAMNT

The windows BAM was initially built on top of the windows logging system. Microsoft has included within Windows NT the ability to log many events on the system if certain registry keys are set to it. Once auditing is set, windows will monitor new processes being opened, closed, etc. The LOGNTBAM is a program which can listen to either of the three Event logs. These are the System, Security, and Event logs. The BAM runs through the log writing out records to the Communication component. Then it listens on the system and as new events are written it catches them and writes them out.

The information collected by each of the LogNTBams includes not only information on successful or failures to log on to the system, but such rich information as new processes, spawned processes, and many other monitored events.

In addition it is configured to process saved .EVT files for incorporating into our DW.

4.1.2 REGBAM

Another set of Windows BAM's are those which monitor the Windows Registry. The Registry BAM (REGBAM) monitors all traffic in the Windows Registry, the data is parsed into useful fields and stored in the DW. REGBAM was used as a data source for an anomaly detection heuristic used for detecting malicious activity [8]. Each Windows sensor can function as a stand alone data source or as part of the larger HoBID system.

4.1.3 Netstat BAM

Netstat is a built in command which provides information on open ports. Within the BAM the command is run at every time T, and a summery is created whenever a port is opened or closed. Just dumping the output of the Netstat command would quickly overload the system, as plain text output would reach gigabyte proportions in a few days on a busy system. Instead whenever the command is run, we notice which ports are new and when the these new ports are not seen again we can output the information on the port. This includes service, port number, duration of observed activity, and time-stamp.

4.2 BAM UNIX

4.2.1 proc Sensors

All Unix hosts provide a /proc virtual file-system. This file-system makes available a simple interface to internal kernel data structures in user-space. [1] Software meeting the definitions of sensors given above was built to collect the data stored in /proc. We constructed three sensors, each analyzing a different part of the /proc file-system to fulfill a specific need; a process sensor to collect statistics on all running processes, a network sensor to report on all open sockets, and a resource usage sensor to monitor the impact of the IDS on host performance. These sensors operate by polling their respective areas of /proc for changes at specified intervals. Each of these sensors is able to do limited data filtering to reduce network and storage overhead.

4.2.2 The Process Sensor

/proc exposes dozens of pieces of information about each running process. Organized by the numeric process ID, this information allows one to monitor the execution of a process from creation to death. The BAM process sensor built is able to track approximately 40 different parameters, including command lines, execution environments, user ID's and open files. To reduce the data overhead, only processes that exhibit a change in key parameters are reported to the rest of the system. For example, the following is a single record showing the status of a web server operating on a particular host. Notice the XML tags that identify the meaning of each piece of data, as well as its data type.

```
<rec><time l>985750214</time><cmdline s>httpd</cmdline>
<pid i>14698</pid><ppid i>1</ppid>
<starttime l>47018241</starttime>
<state c>S</state> <priority i>0</priority>
<uid i>0</uid><gid i>0</gid>
<tty i>0</tty><wd s>/</wd><ctime l>0</ctime>
<vsize u>41172992</vsize><sigcatch i>18027</sigcatch>
<fd s>/dev/null</fd><fd s>/dev/null</fd>
<fd s>/var/log/httpd/error_log</fd>
<fd s>/var/run/mm.14697.sem</fd>
<fds>/var/log/httpd/ssl_scache.sem</fd>
<fd s>/tmp/session_mm.sem</fd>
<fd s>pipe:[52082]</fd><fd s>pipe:[52082]</fd>
<fd s>/var/log/httpd/error_log</fd>
<fd s>/var/log/httpd/ssl-error_log</fd>
<fd s>socket:[52094]</fd><fd s>socket:[52095]</fd>
<fds>/var/log/httpd/ssl_engine_log</fd>
<fd s>/var/log/httpd/ssl_mutex.14697</fd><fd s>/dev/null</fd>
<fd s>/var/log/httpd/access_log</fd>
<fd s>/var/log/httpd/ssl-access_log</fd>
<fd s>/var/log/httpd/ssl_request_log</fd>
<fd s>/etc/httpd/httpd.lock.14698 (deleted)</fd></rec>
```

Data such as this allows for easy tracking of activities such as the changing of user ID's, the excessive use of memory or processor time, or the accessing of files.

4.2.3 The Network Connection Sensor

This sensor collects information regarding the network connections made to and from the local host. For example, the following piece of data might be collected:

```
<rec><time l>985751417</time><localhost l>XXXXX</localhost>
<localport i>0050</localport><remotehost l>XXXXX</remotehost>
<remoteport i>0000</remoteport><uid i>0</uid>
<inode i>52095</inode></rec>
```

Data such as this can easily be used to detect back-door connections or port scanning.

4.2.4 The System Utilization Sensor

This sensor uses /proc to gather information regarding the utilization of a hosts processor and memory usage. This was particularly useful in determining the impact of running the BAM IDS on various host systems. Sample data might look like the following:


```
<rec><time l>988260835</time>
<cpu p>1.39</cpu>
<mem p>96.24</mem>
<swap p>3.84</swap></rec>}
```


4.2.5 Other Sensors

The BAM sensor architecture provides for the simple integration of numerous sensors, including those that are event-driven. In addition to the above sensors, a file modification sensor was built to record additions to files on the host as they occur. This sensor is particularly suited to the monitoring of system and application log files. By using this sensor together with applications with their own logging facilities one can easily convert the output of third-party software to data suitable for use by the mining components.

5 Future Direction

There are still many things which we are working on as this report is being written. The system described here only represents half the picture. To complete the IDS picture, there must be both a network component and a way to tie the two together. The HAUNT project is the answer [htpp://www.cs.columbia.edu/ids/haunt].

Among new BAMs, a working example is a SnortBAM. This BAM wraps the snort program and captures snort output to our XML encoding. a proof of concept was done in Java , but we are implementing it within the Snort code. We are also streamlining and stress testing the current set of working BAMs. In addition to providing new BAMs for newer Operating Systems as they appear.

The next major step is to test all this in a real world setting. Small tests have confirmed that these proof of concepts work [8] [2]. WE are deploying

MIT's Lariet test bed system in our testing environment which would facilitate such test. It will be able to mimic a real world setting of a very large working computer network.

6 Acknowledgments

This has been an ongoing project here at Columbia University. Many people have been very useful to the project and we would like to take the opportunity to thank them for participating on the project. They include among others Anthony Rodriguez, Christy Lauridsen, Xue Rui, Johnie Lee, Ganesh N Sastry, and others who we are sure we left out inadvertently.

7 References

1. Bauer, Bodo and Terrehon Bowden. The /proc Filesystem.
   http://www.linuxhq.com/kernel/v2.2/doc/proc.txt.html.
   January 1999.

2. Wenke Lee, Salvatore J. Stolfo, Philip K. Chan,
   Eleazar Eskin, Wei Fan, Matthew Miller, Shlomo Hershkop and Junxin Zhang. "Real Time Data Mining-based Intrusion Detection." Proceedings of DISCEX II. June 2001.

3. Wenke Lee, Salvatore J. Stolfo, Kui W. Mok
   "A Data Mining Framework for Adaptive Intrusion Detection" June 1998.

4. CERT/cc statistics
   http://www.cert.org/stats/cert_stats.html
   1999

5. Biswanath Mukherjee, L. Todd Heberlein, Karl N Levitt
   "Network Intrusion Detection"
   http://seclab.cs.ucdavis.edu/papers/mhl94.pdf
   1994

6. Sandeep Kumar, Eugene H Spaford
   "A Pattern Matching Model for Misuse Intrusion Detection"

7. Eleazar Eskin, Matthew Miller, Zhi-Da Zhong, George Yi, Wei-Ang Lee, Sal Stolfo
"Adaptive Model Generation for Intrusion Detection Systems"
Workshop on Intrusion Detection and Prevention,
7th ACM Conference on Computer Security, Athens, GR:
November, 2000

8. Frank Apap, Andrew Honig, Shlomo Hershkop, Eleazar Eskin, Sal Stolfo
"Detecting Malicious Software by Monitoring Anomalous Windows Registry Access"
submitted to Proceedings: 2002 IEEE Symposium on Security and Privacy