RZ 2998 (# 93044) 03/09/98
Computer Science/Mathematics 15 pages

Research Report

An Experimentation Workbench for Intrusion Detection Systems

H. Debar, M. Dacier, A. Wespi and S. Lampart

IBM Research Division
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

IBM Research Division
Almaden · Austin · Beijing · Haifa · T.J. Watson · Tokyo · Zurich

An Experimentation Workbench for Intrusion Detection Systems

H. Debar, M. Dacier, A. Wespi and S. Lampart[1]

IBM Research Division, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract

To compare existing intrusion detection solutions, an experimentation workbench has been developed at the IBM Zurich Research Laboratory. We focus here on its design principles and implementation. This paper is intended to make other teams working in the intrusion detection arena aware of the existence of such a workbench. Thus we hope to foster interaction with them to perform further experiments in the near future.

[1]Stefan Lampart has participated to the work reported here while writing his Master Thesis [9] at the IBM Zurich Research Laboratory.

1 Introduction

Since the seminal work by D. Denning in 1981 [4], many intrusion detection prototypes have emerged. M. Sobirey maintains a, yet non exhaustive, list of 53 of them [16]. Many tools and techniques have been investigated but, unfortunately, no testbed exists to compare their respective advantages and drawbacks. Several people have examined the problem ([1],[15]) but no generic framework has, to our knowledge, been proposed so far.

At the Zurich Research Laboratory, we are experimenting with several techniques to implement intrusion detection systems. We have designed and implemented a workbench to support a comparative evaluation of these techniques. This workbench enables us to compare the respective efficiency of our prototypes in terms of, for example, false alarm rates. The design of this workbench is such that it can be used as a generic intrusion detection testbed. To date, we have only used it to test our own tools.

2 Background

Intrusion detection systems ([5], [16], [17]) analyze audits provided by an environment (machine, network,....) to detect attacks or misbehavior occurring in this environment.

The term **audit** denotes the information gathered by an operating system or a network device characterizing its operation. Examples of audits include but are not limited to accounting, syslog, and C2 audit files. These audits are automatically produced by the environment, reflect its operation, and describe the actions taken by its users and by other devices, either directly or indirectly.

The term **user** denotes a person who uses the environment for every-day tasks. By extension, it is also used for virtual entities that *use* the environment to deliver a given service (e.g. an *ftp server* can be regarded as a user of the machine on which it is running).

The term **operator** denotes the workbench user.

Existing IDSs are frequently classified into *misuse* and *anomaly* detection methods (see, e.g., [8]) depending on the kind of prohibited activities they are trying to detect. We prefer a different classification based on the paradigm used to detect suspicious activities. Thus we consider two families of techniques, which we call **knowledge-based**[2] and **behavior-based**:

- **Knowledge-based** intrusion detection techniques exploit the knowledge accumulated about specific attacks and system vulnerabilities. The intrusion detection system contains information about these vulnerabilities and looks for evidence of attempts to exploit these vulnerabilities, successfully or unsuccessfully. When an attempt is detected, an alarm is triggered; otherwise the actions of the user are deemed appropriate from a security point of view.

  In other words, any action not explicitly recognized as an attack is considered acceptable.

  Advantages of the knowledge-based approaches are that the ideal false alarm rate is very low, and the contextual analysis proposed by the intrusion detection system is detailed, making it easier for the security officer using this IDS to take preventive or corrective action.

  Drawbacks include the difficulty of gathering the required information on the existing attacks and keeping it up to date with new vulnerabilities and environments.

[2] Note that we use this term to describe a different concept than the one presented in [10].

- **Behavior-based** intrusion detection techniques entail the assumption that an intrusion can be detected by observing a deviation in the behavior of the system. This IDS acquires information about the normal behavior of the system and compares it with the current activity. When a deviation is observed, the intrusion detection tool generates an alarm.

  In other words, anything that does not correspond to a previously observed and learned behavior is considered unacceptable.

  Advantages of the behavior-based approaches are that they can take into account new and unexpected vulnerabilities. They can even contribute to the (semi)automatic discovery of these new attacks. They rely less on the underlying operating system-specific mechanisms, and can help detect abuses of privileges that do not involve actually attacking the environment via security vulnerabilities.

  The high false alarm rate is generally cited as the main drawback of behavior-based techniques.

Various tools can be used to implement both families. For example, statistics [11], expert systems [5], signature analysis [8], neural networks [3], and model-based reasoning [7] have been used to implement various research prototypes and commercial products.

These intrusion detection tools share a common set of properties. They all have a detector component that makes use of a database and a configuration file to process audit information and to generate alarms. In complex cases, the intrusion detection system might be considered to contain several of these detectors with their associated input, output, configuration file, and database. These components and the way they interact are depicted in Figure 1.

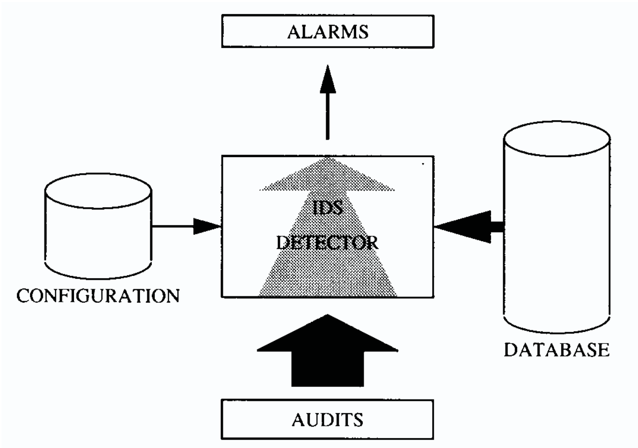

Figure 1: Generic representation of an intrusion detection system

The IDS detector processes large amounts of audit information generated by the environment and provides a small stream of alarms. The database of the detector contains its knowledge about the system it monitors. For a *knowledge-based* intrusion detection system, this is the set of attack signatures, or the set of rules describing attacks. For a *behavior-based* IDS, this is the profile of normal user behavior, a statistical profile, neural network, etc. The configuration file contains information about the environment monitored by the IDS.

3 Design principles

3.1 Introduction

In this section, we outline the most important requirements concerning the workbench and how they have influenced its design:

1. The workbench must generate a wide variety of normal usage patterns. These normal usage patterns are important for both behavior-based and knowledge based intrusion detection systems. Therefore, we have included in the current version of the workbench two methods for generating normal user behavior.

2. The workbench must cope with a **heterogeneous environment**. Some intrusion detection systems are specific to one platform. These platform-specific aspects must be taken into account for generating both normal user activity and attacks in a "portable" way. The generic architecture of the workbench is shown in Section 3.4.

3.2 Simulating user behavior

As mentioned above, the workbench aims at generating normal user behavior for the training phase of behavior-based intrusion detection systems. It also generates normal user behavior to verify the false alarm rate of intrusion detection systems operating in a "safe" environment regardless of the technology they implement.

The ideal situation for simulating user behavior is to have real users conducting activities on the workbench. There are a few reasons, however, why we would not want to do this, one of them being that we do not want to make vulnerable servers accessible on an internal network (where the real users are), that is also accessible from other company locations and over the internet. Furthermore, running the learning phase in a real environment exposes the workbench to the risk of an attack. This could potentially allow let the system to consider an attack a *normal* activity!

Therefore, the workbench runs on a physically secure network without outside connections, and we have resorted to simulation techniques to create user activities. An additional benefit that we draw from this approach is some sort of reproducibility of the user behaviors that we generate. Three approaches for simulating user behavior are envisaged in the following subsections.

3.2.1 Generic session generation tool

The first approach is based on a finite state automaton that implements all possible actions that can be performed by a user. Generating normal behavior means simply running the automaton. An example of this is given in Figure 2, which shows a partial representation of the required finite state machine to simulate the client of an ftp daemon. Figure 2 depicts only a very limited subset of some 70 different commands (without options!) that the ftp daemon can analyze, and not all paths are represented (e.g. one may issue the *ls* command after a *put* command).

However, only a very limited subset of these commands is actually known to users and used by them. Therefore, it would make sense to enrich these arcs with transition probabilities. As we are interested in the logic of this process, a natural solution for this kind of modelization would be to use stochastic Petri nets [14], the coverage graphs of which are Markov graphs [12].

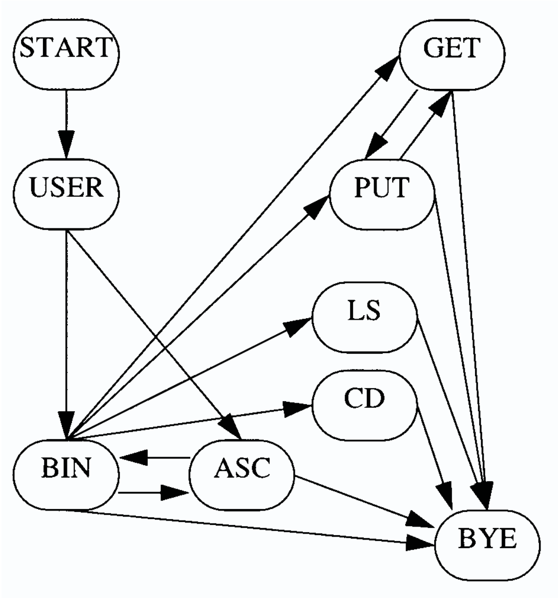

Figure 2: Transition graph for an ftp client

We could then use those Markov graphs to generate test suites using, for example, classical statistical software testing methods [18].

However, this approach is very difficult to implement for three main reasons:

1. The values of the transition probabilities are difficult to determine.

2. This approach is only suitable when the set of commands is limited. It can simulate an ftp client but not a normal user shell session.

3. Even in simple cases such as ftp clients, implementation problems can arise. Indeed, ftp clients on different platforms sometimes have a different syntax, and the commands they generate make the ftp server react in different ways. Also, maintaining the coherence of the environment is quite difficult, as the automaton must analyze the return information given by the ftp daemon to decide which parameters it can insert in the next command. For the automaton to generate meaningful sequences of commands, it has to analyze the information returned by the ftp server and create the subsequent command accordingly, otherwise the number of error messages generated by the ftp daemon would be much higher than what the normal usage pattern shows. These difficulties encountered in modeling ftp activity are true for other services as well.

3.2.2 Using test suites

A simpler approach consists of using the test suites provided by the development laboratories that develop operating systems. In our case, we can apply the test suites used in the AIX environment to which we have access. These test suites are usually created to test whether the implementation complies with its specification and to evaluate the performance of the various services offered by the operating system. These test suites do not offer a *realistic* view of what users usually do but, at least, they let us see how the system should behave when responding to *normal* requests.

This approach is particularly interesting to evaluate the false alarm rate because the test suite will make the network daemons generate all sorts of events that do not (or very infrequently) occur in real-life systems. In this case, every alarm is for sure a false alarm.

If the IDS has rules governing the normal behavior of the network service, running such a test suite allows us to check the completeness of the normal behavior model and also to interpret alarms generated by the IDS as false alarms. However, the capability to fully train a behavior-based IDS on this kind of data is still an open issue, as the profiles may contain many "borderline" behaviors.

3.2.3 Recording live data

The third approach consists of recording normal activities of various people and replay them inside the workbench. This is described in more detail in Section 4.2. One constraint set on these recordings is that we must have a sufficient number of recordings to create heavy loads on the servers and services we are testing by playing several recordings in parallel. We must also be able to separate training and testing phases for behavior-based IDSs. The IDS is trained with a subset of the recordings. During the testing phase, other recordings are played to verify that the behavior model created by the IDS has certain generalization capabilities.

3.3 Running attacks

An important functionality of the workbench is to enable us to run attacks and verify that the tested IDSs have detected them. In the following subsections we envisage two important issues: (i) where can we find *exploit scripts*[3] and (ii) how can we run them automatically against various kinds of servers.

3.3.1 Obtaining exploit scripts

Our main source of information for attacks is a vulnerability database that we maintain internally. This vulnerability database contains sorted information concerning vulnerabilities and attack techniques. It also contains hundreds of exploit scripts that we can use for testing purposes on the workbench. The database is updated daily and resides on the same protected network as the workbench (disconnected from the rest of the world).

3.3.2 Recording and replaying attacks

Replaying attacks also restricts the kind of attacks that we would like to have on the workbench. We consider our workbench to be a heterogeneous environment. However, some attacks clearly depend on the environment. The typical case is a buffer overflow attack, where the attacker has to insert opcodes for the processor of interest at a precise place to achieve the desired execution. This reduces the generality of our attack generator to target a specific kind of host, operating system, or architecture.

To record an attack, we create attack scripts that follow the various steps required in order to gain superuser privileges. At the end of the script execution, the actual privileges are checked to make sure that the desired goal has been achieved. This attack script can then be run at any time during the experiment, at the same time the normal behavior is generated, or separately if required.

[3] An exploit script is a program written to take advantage of a known vulnerability.

3.4 Description of workbench elements

Figure 3 shows the various components of the workbench. The server machine supports the network services (SERVICES box, representing the server's operating system network services) that act as the target for normal behavior and attacks. It generates the corresponding audit information. If the IDS is host-based, the server can also run the IDS (IDS box), or it can send its audits to another machine for processing.

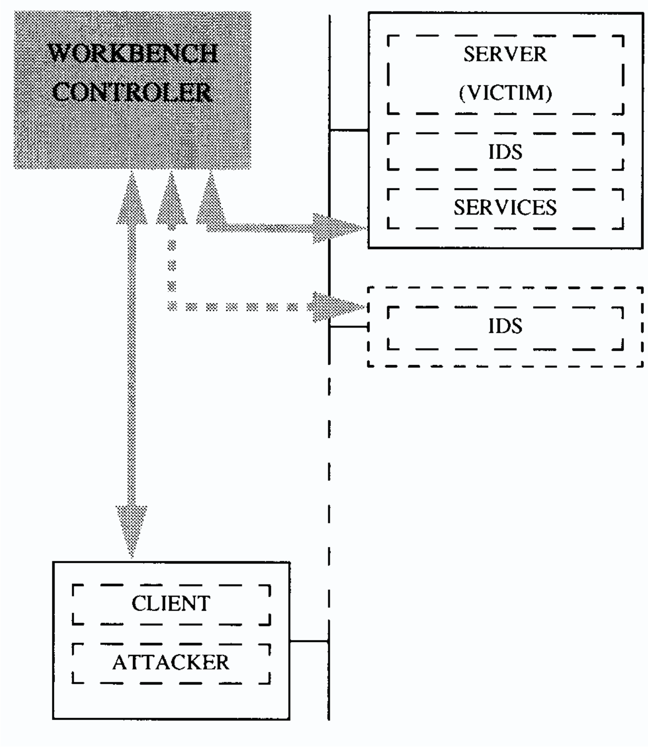

Figure 3: Workbench elements

Physically close to the server and on the same subnetwork, the workbench has an optional second machine. In cases where the host-based IDS does not run on the server, it runs on this optional machine and retrieves information from the server via the network. In cases where the IDS is network-based, it runs on the optional machine with the network adapter in promiscuous mode.

The client machine supports the generation of user behavior and attacks. It can be far away from the other machines, as indicated by the dotted line.

The last element of the workbench is the workbench controller. It interacts with the three other components. On the server and the optional machine, the controller configures the IDS (configuration file, Figure 1) and the server (creates a vulnerable server, starts audits collection). It also displays information returned by the IDS concerning the alarms that it generates. On the client, the workbench controller triggers and tunes the generation of normal behavior and attacks by interacting with the client and the attacker tools. The workbench controller interacts with the environment in a way that does not perturb the client and the server.

4 Current implementation

The current implementation of the workbench is limited to creating normal activity on and attacks against an ftp server. Ftp has been chosen as our first network service because it is widespread and easy to find on various platforms. Also, there have been several well-known attacks against it.

4.1 Environment configuration

The environment, represented in Figure 4, consists of a double network of workstations that support both token ring and ethernet protocols. Ethernet is either switched Ethernet (switch not shown) or thin Ethernet, which is mainly for *sniffering* purposes as switched ethernet does not allow the retrieval of all the network traffic. All machines are connected to both networks except one of the servers for which we do not have a token ring adapter.

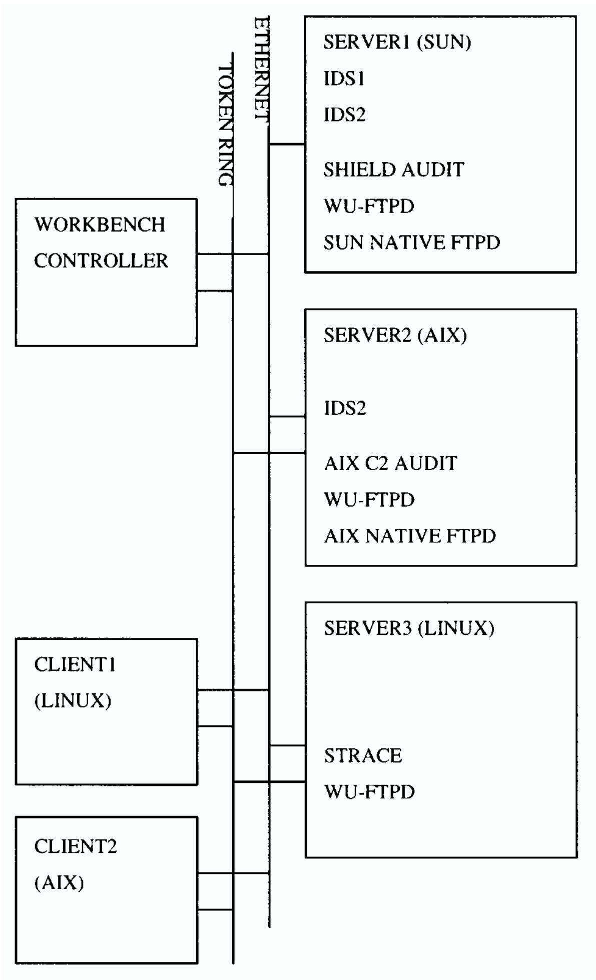

Figure 4: Workbench current components

4.1.1 Description of the configuration

The principle of an experiment is to test a set of IDSs against a given server. The configuration of the server and the various IDSs can be modified with the workbench controller. The controller connects to the server via a web server and cgi-bin scripts. It can run on one of the clients, on one of the servers, or on a separate machine. The fact that we have two networks gives us some degree of freedom in lessening the impact of the controller on the other elements of the workbench as we can use one for client/server traffic and the other for control traffic.

Currently, we have two clients and three servers configured to generate normal behavior. The attack scripts target only the LINUX and the AIX servers, although it should be possible to run them against the SUN server. All three servers run the ftp daemon from the University of Washington, using a version that is vulnerable to a number of attacks (see Section 4.3). Furthermore, the Sun and the AIX boxes also run their native ftpd daemon.

4.1.2 Intrusion detection systems in the workbench

The IDSs installed on our workbench are all host-based, i.e. they reside on the host and take their input data from it. There is no obstacle to install a network-based IDS. However, we have not yet experimented with any existing network-based IDS at this stage.

IDS1 is a research prototype freely available from a university. It runs on SUN workstations, uses the SHIELD security package and the audit provided by SUN, and implements a knowledge-based intrusion detection technique.

IDS2 is a commercial product, available on several platforms, e.g. SUN and AIX. It is derived from a research prototype and implements a knowledge-based intrusion detection technique as well as tools for managing the audit trails in a heterogeneous environment.

IDS3 represents a set of four prototypes developed at the IBM Zurich laboratory. They are all compliant with the model described in Section 3.2. These prototypes use behavior-based techniques for intrusion detection but different underlying techniques.

IDS1 and IDS2 have been installed on the workbench but they are not yet compliant with the model described in Section 3.2, i.e. the controller cannot yet interact with them. Therefore, we have checked their functionalities but have not used them in our comparative experiments so far.

4.1.3 Tools

We use the following UNIX tools on the workbench:

- **Script**: Script allows user sessions to be recorded.
- **Expect**: Expect is a Tcl-based tool that allows interactive sessions to be replayed by simulating user input on the keyboard. We use this tool to send commands to the ftp client and to analyze the output received from the server.
- **Perl**: Perl is a scripting language heavily used in the workbench implementation. We use it for cgi scripting and as a wrapper to put pieces together.

4.2 Normal behavior

Normal behavior in the workbench is created using recorded scripts of user sessions and replaying them to simulate user activity in the environment. As stated above, our objective was to generate normal behavior of daemons. The ftp daemon was chosen for our first experiments.

The third approach described in Section 3.3 has been implemented and tested. This tool creates a database of scripts to simulate ftp sessions with the server. We also can tune the replaying of scripts to vary the load of the server, and thus obtain a simulated activity containing as many cases as can be found in real life.

It is worth noting that this modelization effort has to be done for each new service of interest. Experience shows that this procedure, despite its apparent simplicity, is very time-consuming. Therefore, the use of test suites[4] developed for testing the functionalities of operating systems looks much more promising on a larger scale.

Figure 5 shows the mechanisms used for recording and replaying user behavior. When a user records an ftp session, it first starts a perl script that records everything appearing on the screen using the unix command *script*. Once the session is terminated, another perl script (*translator*) translates the obtained screen dump into an *expect* script, also removing some control characters and inserting information that is not displayed on the screen (password). The script is then stored in the database.

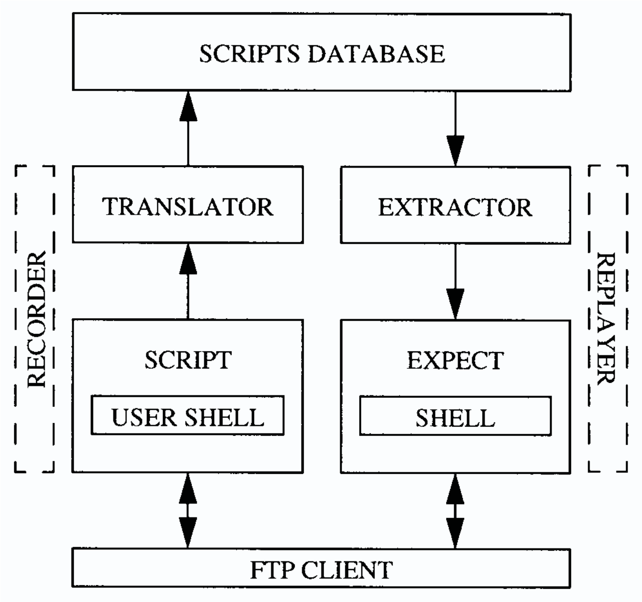

Figure 5: Normal behavior generation

For replaying, the *extractor* module on the client selects a script in the database and verifies that the corresponding user is not already active for consistency reasons. It then creates the environment for this user on the server and runs the script using the *expect* program to simulate interactive activity. When the execution is completed, the *extractor* module cleans up the user environment and removes the locks that prevented concurrent runs.

The average time between the initiation of two ftp sessions is a parameter that can be modified in the *extractor* module. Setting this time to a low value allows us to create many sessions in parallel and thus to simulate a heavily loaded ftp server.

It is important to maintain the consistency of the environment in which we are running the scripts. Before running one of the expect scripts, we have to restore the original environment in which it was originally created. This is done by "out of band" means, consisting currently of an NFS mount of the server's disk on the client's machine to copy the appropriate files. This solution is appropriate for the moment, at the cost of some performance.

[4] as explained in Section 3.2.2

4.3 Attacks

This section describes attacks implemented against our ftp servers in the workbench. By exploiting well-known vulnerabilities of tools available on the net. Exploits for these vulnerabilities have been available for a long time. The reader is referred to well-known mailing list archives (e.g.[19]) for exact explanations of the mechanisms underlying these attacks, and to books on the subject (e.g.[6]).

Transforming exploit scripts found in our database into attack scripts requires some work but setting up a reliable and vulnerable server has also proved to be a difficult task! We have found on the web an old version of the wu-ftpd program known for the vulnerabilities it contained. Unfortunately, this version also contains other bugs not related to security issues, that cause the server to crash from time to time. Therefore, we patched this old version, carefully applying selected fixes that improved the reliability of wu-ftpd while retaining the security vulnerabilities. We ended up with a hybrid version of the wu-ftpd daemon which was sufficiently recent to be compiled, installed, and run reliably, but which still contained the vulnerabilities of interest.

In the next subsection, we give a brief overview of the four categories of attacks on the wu-ftpd daemon.

4.3.1 Put-dot-file attacks

These attacks work on any UNIX machine with an improperly configured ftp site. When the ftp home directory of a UNIX host is writable, a remote intruder can, for example upload a .rhosts or .forward file to gain access to the system.

These attacks work in two phases. First, the cracker uploads a dot-file. Second, he uses this uploaded file to attack the host (by sending a mail or by logging in).

Two attack scripts related to this family of attacks have been implemented.

4.3.2 Site-exec attacks

A problem exists with certain configurations of the University of Washington ftp daemon, which may allow root access from any account on the system. In its original form, the vulnerability was not enabled by default. However, certain distributions of LINUX contain a wu-ftpd that has been compiled with a vulnerable configuration. This vulnerable configuration is distributed and run by default.

The precompiled binaries shipped with old Linux Slackware distributions are vulnerable. The variable PATH_EXECPATH was set to /bin in the configuration file src/pathnames.h when the distribution binary was built. Therefore, a cracker with an account on this machine can execute all programs via ftp by using the 'site exec' command. On some servers, the command run with the help of 'site exec' is executed as root.

Two attack scripts belonging to this category have been implemented.

4.3.3 Tar-exec attacks

These attacks also enable anonymous users to gain root access if certain criteria are met:

- The site must have GNU tar in the SITE EXECable directory.
- There must be a world writable directory for the cracker to upload a file.
- The site must not disable chmod.

The core of the problem is that the "site exec" command on certain wu-ftpd servers does not reset the real uid before executing a command. Thus, all commands are executed with a real uid/gid of 0. The way to exploit this problem is to compile a small program that does, among other things, a setuid(0), calls chmod(0755, "/bin/ftp-exec") and copies its standard input to stdout. The cracker can now upload this program to the /incoming directory along with a chosen tar archives (we call it tools.tar). Then the cracker has to chmod the program (we call it hackme) to 0755 with 'site chmod', he has to go to /bin/ftp-exec and invoke tar with the following options:

"site exec tar –use-compress-program /incoming/hackme -xf /incoming/tools.tar"

By doing so, the cracker has two possibilities to attack the ftp site:

- He can execute a program and obtain a root shell.
- He can copy his attack tools into the /bin/ftp-exec directory and then run them via ftp by using the "site exec" command.

Four attack scripts belonging to this category have been implemented.

4.3.4 Passive attack

This attack exploits a vulnerability of the University of Washington ftp daemon concerning the passive command. When a user opens a connection to an ftp site, but does not log in and executes the passive command, a core dump is created which contains some subset of shadowed passwords. Then, this file can be retrieved and, thanks to the *"Crack"* program [13], legitimate passwords can be discovered.

One attack script implements this attack.

4.4 Controller

The user interface of choice is a web browser that interacts with the workbench using cgi-scripts written in perl. This is a choice of convenience but we do not rule out enhancing the user interface by using tools related to network management. The user interface allows the workbench operator to create the testing environment and to configure the IDSs. Then, the user can start and stop tests. He can also visualize the results in an easy-to-compare format.

The user interface of the workbench is separated into three main components displayed on top of one another. The lower third of the user interface is the workbench configuration, the middle third allows the configuration of the IDSs to be tested, and the upper third displays and compares the alarms generated by the IDSs during the test execution. The user interface is represented in Figure 6. This snapshot shows the workbench running IDSs in parallel.

4.4.1 Configuration of the workbench

This feature allows the controller operator to perform the following actions:

- Choose how many IDSs will be tested in parallel. The workbench then allows the operator to configure each of them.
- When working with a behavior-based IDSs, choose whether one is working with the learning phase (learning the workbench behavior) or with the detection phase.
- Start and stop IDSs.

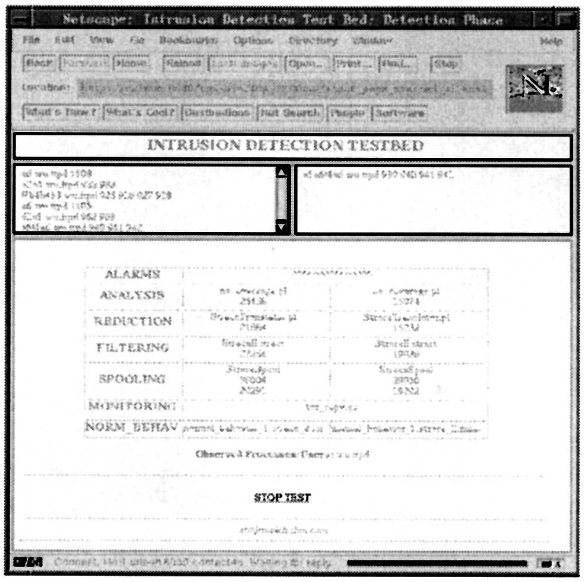

Figure 6: Workbench user interface

- Select the input source for the IDS as live information generated by the workbench or from previously recorded experiments.
- Replay normal user sessions, attacks, or manually started sessions to generate information for the workbench.

4.4.2 Configuration of the intrusion detection systems

This feature allows the workbench operator to connect various modules to test different combinations of modules and compare them with one another.

Each IDS is represented by a stack of modules. For each cell of the stack, the workbench operator selects the module of interest from predefined choices. The modules must be able to generate output information comprehensible to higher modules.

Currently, the user interface allows us to configure only our own prototypes of individual modules. We plan to add configuration facilities for the IDSs that we have installed on the workbench. For IDS1, it will take the form of a configuration file editor. For IDS2, it will fire its own user interface.

4.4.3 Comparison of the alarms generated

This feature allows the controller operator to perform the following actions:

- display the output of several IDSs side-by-side as text.
- trace the origin of each alarm to the audits and visualize the set of audits that led to the creation of the alarm by the IDS.

The bottom part of the screen represented in Figure 6 displays the STOP_TEST button. This allows the workbench operator to start or stop the experiment. Clicking on this button lets the operator configure the workbench by selecting which machine is the target, which users or processes are involved, and how the normal activity is created (from files or by activating the client with given parameters).

The middle part organized in tables indicates which IDSs are running. Each column represents an IDS, with its modules stacked on top of one another, and the associated process numbers. The two lower lines indicate from which files the normal data and the attack data are taken. They would contain information about the client if it were an interactive run.

The upper part displays the alarms generated by IDSs. These alarms contain information that allow the user to click on the audit record references (sequence of numbers at the right of each line) and bring up on the screen the area in which suspicious activity has been detected.

4.5 Results

The aim of the workbench is to compare the efficiency of various IDSs. In recent months, we have conducted many different kinds of experiments with it. They have helped us enhance the prototypes we are working on, the description of which is beyond the scope of this paper. We refer the reader to [2] for further information.

On the other hand, there are a few important results related to the workbench itself that are worth briefly mentioning:

1. The workbench exists. It has been implemented and used to compare four different prototypes of behavior-based intrusion detection techniques.

2. The model described in Section 3.2 offers a precise, yet flexible framework to integrate other interesting IDSs. It has proved to be expressive enough so far to implement four different techniques.

3. Our experiments indicate that there are simple ways to reduce dramatically the false-alarm rate of behavior-based approaches by using them in parallel on the same data source.

4. The behavior of an ftp server can be dependent on the behavior of the client.[5] This has important consequences on the way the learning phases of the behavior-based approaches have to be carried out.

5. Generating realistic, normal behavior is a time-consuming task when working in a heterogeneous environment.

5 Conclusions and future work

The purpose of developing an IDSs workbench was to enable an easy comparison of the effectiveness of various intrusion detection prototypes under development at the IBM Zurich Research Laboratory. The design of this workbench is such that it allows us to envisage in the near future its integration into an existing IDS, for testing purposes. Therefore, we hope to cooperate with other research teams interested in using this workbench. The abstract model we have defined in this paper and successfully used with our prototypes provides a well-defined, yet flexible framework that will certainly facilitate such an integration process.

[5] As an example, the LINUX ftp client systematically generates the "ls -1" when the user submits an "ls" command, whereas the AIX ftp client really sends out the command submitted. The difference is important for behavior-based IDS, because the simple "ls" is implemented internally in the ftp server, whereas the "ls -1" requires forking another process to retrieve the information.

On the other hand, we will continue our experiments with other remote services. In order to speed up the process of generating normal-user behavior, the use of normal test suites used by the AIX development laboratory is considered.

Finally, it is worth noting that the definition of a set of criteria to evaluate the effectiveness of existing IDSs remains an open issue which we will have to address if we want to perform pertinent comparative studies.

References

[1] M. Chung, N. Puketza, R.A. Olsson and B. Mukherjee, "Simulating Concurrent Intrusions for Testing Intrusion Detection Systems: Parallelizing Intrusion", *Proc. of the 1995 National Information Systems Security Conference.* Baltimore, MD, October 10-13, 1995, pp. 173-183.

[2] M. Dacier, H. Debar, M. Nassehi and A. Wespi, *Comparing behavior-based intrusion detection techniques*, in preparation.

[3] H. Debar, M. Becker, D. Siboni, "A neural network component for an intrusion detection system", *Proc. of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1992, 1-11.

[4] D. Denning, "An intrusion-detection model", *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 222-232, Feb. 1987.

[5] M. Esmaili, R. Safavi-Naini and J. Pieprzyk, *Computer Intrusion Detection: A comparative Survey*, Center for Computer Security Research, University of Wollongong, Australia, Technical Report, Ref. 95-07, May 17, 1995, 11 pages.

[6] S. Garfinkel and E. H. Spafford, *Practical Unix and Internet Security*, O'Reilly and Associates, Second Edition, April 1996, 971 pages.

[7] T. D. Garvey and T. F. Lunt, "Model-Based Intrusion Detection", *Proc. of the 14th National Computer Security Conference*, Washington DC, October 1991, 14 pages.

[8] S. Kumar and E. H. Spafford., "A pattern matching model for misuse intrusion detection", *Proc. of the 17th National Computer Security Conference*, pages 11-21, October 1994.

[9] S. Lampart, *Creation of an Intrusion Detection Test Bed*, Professional Thesis, Eurecom Institute, Sophia Antipolis (France), July 1997, 85 pages.

[10] T. F. Lunt, R. Jagannathan, R. Lee, A. Whitehurst, "Knowledge-based Intrusion Detection", *Proc. of the 1989 AI Systems in Government Conference*, Washington DC, March 1989.

[11] T. F. Lunt, R. Jagannathan, D. Anderson, C. Dodd, F. Gilham, C. Jalali, H. Javitz, P. Neuman, A. Tamaru and A. Valdez, "System Design Document: Next-Generation Intrusion Detection Expert System (NIDES)", SRI International Technical Report, March 1993.

[12] M. Molloy, "Performance analysis using Stochastic Petri Nets", *IEEE Trans. on Computers*, vol. 39, no. 9, pp. 913-917, 1982.

[13] A. D. E. Muffet, *Crack Version 5.0, User Manual*, available for anonymous ftp from cert.sei.cmu.edu in /pub/tools/crack

[14] S. Natkin, *Les reseaux de Petri Stochastiques*, These de Docteur Ingenieur, CNAM, Paris 1980.

[15] N. F. Puketza, K. Zhang, M. Chung, B. Mukherjee and R. A. Olsson, "A Methodology for Testing Intrusion Detection Systems", *IEEE Transactions on Software Engineering*, Vol.22, No.10, October 1996, pp.719-729.

[16] M. Sobirey, *Michael Sobirey's Intrusion Detection Systems page*, http://www-rnks.informatik.tu-cottbus.de/ sobirey/ids.html.

[17] A. Sundaram, "An introduction to Intrusion Detection", *Crossroads: The ACM Student Magazine*, Vol. 2, No. 4, April 1996.

[18] P. Thevenod-Fosse, "Software validation by means of statistical testing: retrospect and future direction", in *1st IFIP Working Conference on Dependable Computing for Critical Applications*, Santa Barbara, CA, pp.15-22, 1989.

[19] *Bugtraq mailing list archives*, http://www.geek-girl.com/bugtraq/.