Intrusion detection

Evaluation of the diagnostic capabilities of commercial intrusion detection systems

Hervé Debar,
Benjamin Morin

Hervé Debar, Expert Sénior à France Télécom R&D, est spécialiste en techniques et outils de détection d'intrusions, ainsi que dans les technologies complémentaires (analyse de vulnérabilités, leurres, audits de sécurité, gestion des alertes et des incidents).

Benjamin Morin, INSA Rennes 2000, prépare une thèse de doctorat à France Télécom R&D (Caen) sur les techniques de détection d'intrusion utilisant la corrélation des alertes.

Intrusion detection

Evaluation of the diagnostic capabilities of commercial intrusion detection systems

Abstract

Description of a test bed for comparative evaluation of intrusion-detection systems. It is shown that the benefit of the system is maximized when an accurate diagnostic of the malicious activity is provided. This leads to the proposal of an enhanced model.

This paper describes a testing environment for commercial intrusion-detection systems,shows results of an actual test run and presents a number of conclusions drawn from the tests. Our test environment currently focuses on IP denial-of-service attacks, Trojan horse traffic and HTTP traffic. The paper focuses on the point of view of an analyst receiving alerts sent by intrusion-detection systems. While the analysis of test results does not solely targets this point of view, we feel that the diagnostic accuracy issue is extremely relevant for the actual success and usability of intrusion-detection technology. The tests show that the diagnostic proposed by commercial intrusion-detection systems sorely lack in precision and accuracy, lacking the capability to diagnose the multiple facets of the security issues occuring on the test network. In particular, while they are sometimes able to extract multiple informations from a single malicious activity, the alerts reported are not related to one another in any way, thus loosing signicant background information for an analyst. The paper therefore proposes a solution for improving current intrusion-detection probes to enhance the diagnostic provided in the case of an alert, and qualifying alerts in relation to the intent of the attacker as percieved from the information acquired during analysis.

Introduction

Since the seminal work by Denning in 1987 [7], many intrusion-detection prototypes have been created. Intrusion-detection systems have emerged in the computer security area because of the difficulty of ensuring that an information system will be free of security aws.

Commercial intrusion-detection systems have been available since 1995; however, their performance has not been scientically studied. In early 2001, we decided to create a testing environment for commercial intrusion-detection systems, following a three phase approach. After a litterature survey getting information from vendors and the community about multiple intrusion-detection products, we selected a small number of them for internal testing and comparative evaluation. Early on, it became clear that the study should be restricted in scope in order to provide on network-based intrusion-detection commercial products, with probe components available worldwide as a remotely manageable appliance. We finally deployed four commercial intrusion-detection systems on a testbed and carried out a comparative evaluation. Partial results from this evaluation are presented in the paper.

The emphasis of this work is on ensuring that the benefit of deploying intrusion-detection technology is maximized by providing a detailed and accurate diagnostic of the malicious activity occuring on our networks. The objective is to ensure that operators with a good knowledge of their networks but little in-depth knowledge of vulnerabilities and intrusions can operate the probes, leaving only serious security breaches to trained analysts. Operational issues such as management of the probes, updates to software, signature and conguration, performance, are recognized as extremely important, but were given second priority to the diagnosis accuracy issue.

The remainder of the paper is organized as follows:

- "Introduction to intrusion-detection systems" proposes a short introduction to the area of intrusion-detection.
- "Background on testing intrusion-detection systems" (page 53) presents the goals and organisation of the tests.
- "The France Télécom R&D intrusion-detection testbed" (page 56) presents our testing principles and the testbed we designed.
- "Results obtained during the tests" (page 59) presents the results obtained and the lessons that we learned from the tests.
- "Proposed model for an intrusion-detection system" (page 66) draws from these lessons to propose a enhanced model for an intrusion-detection system that would emphasize diagnostic accuracy.

Introduction to intrusion-detection systems

Description of a generic intrusion-detection system

An intrusion-detection system acquires information about its environment to perform a diagnosis on its security status. The goal is to discover breaches of security, attempted breaches, or open vulnerabilities that could lead to potential breaches. A typical intrusion-detection system is shown in Figure 1.

An intrusion-detection system can be described at a very macroscopic level as a detector that processes information coming from the system that is to be protected (Fig. 1). This detector can also launch probes to trigger the audit process, such as requesting version numbers for applications.

This detector uses three kinds of information: long-term information related to the technique used to detect intrusions (a knowledge base of attacks, for example), configuration information about the current state of the system, and audit information describing the events that are happening on the system.

The role of the detector is to eliminate unneeded information from the audit trail. It then present either a synthetic view of the security-related actions taken during normal usage of the system, or a synthetic view of the current security state of the system. A decision is then taken to evaluate the probability that these actions or this state can be considered symptoms of intrusion or vulnerabilities.
A countermeasure component can then take corrective action to either prevent the actions from being executed or changing the state of the system back to a secure state.

Intrusion-detection systems can be classified by two properties, the information source used by the system and the technology used to analyze the data. The following sections cover these in more details.

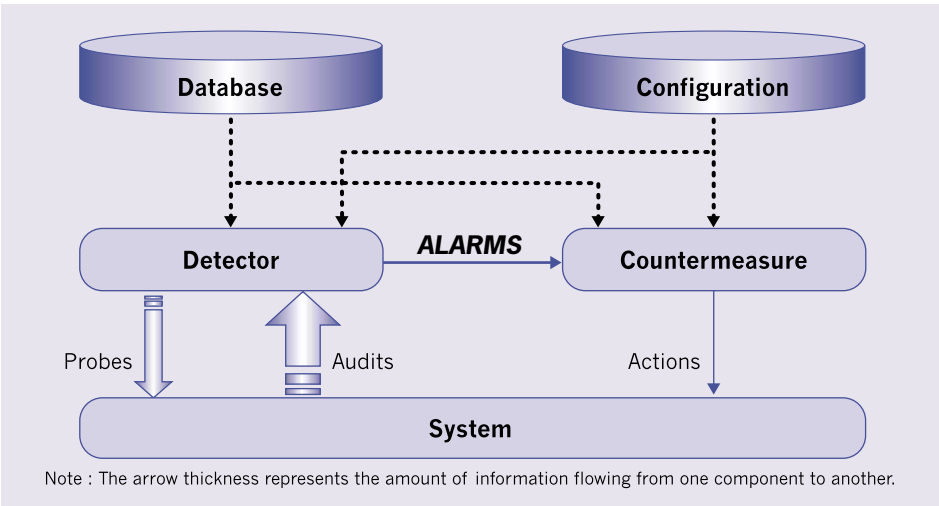


Figure 1 - Very simple intrusion-detection system.

Information sources for intrusion detection systems

Three categories of information sources have been used by intrusion-detection systems, network traffic for network-based intrusion detection, system logs for host-based intrusion detection, and application logs.

Network-based information sources

As the popularity of network sniffers for gathering information has grown in the attackers community, they are also regarded today as an efficient means for gathering information about the events that occur on the network architecture. This is consistent with the trend of moving from a centralized to a distributed computing model, and the pace of change has even increased with the widespread diversification of the Internet. Most accesses to sensitive computers today take place over a network, and therefore capturing the packets before they enter the server is probably the most efficient way to monitor this server.

It is also consistent with the occurrence of denial-of-service attacks. As companies put valuable information on the Internet, and even depend on it as a source of revenue, the prospect of simply shutting down a web site creates an effective threat to the organization running it. Most of these denial-of-service attacks originate from the network and must be detected at the network level, as a host-based intrusion-detection system does not have the capability to acquire this kind of audit information.

There is an inherent duality in network sniffers, which is also apparent in the rewall world with its differences between application-level gateways and ltering routers. If the analysis is carried out at a low level by performing pattern matching, signature analysis, or some other kind of analysis of the raw content of the TCP or IP packet, then the intrusion-detection system can perform its analysis quickly. This is a stateless approach that does not take session information into account because the latter could span over several network packets. If the intrusion-detection system acts as an application gateway and analyzes each packet with respect to the application or protocol being followed, then the analysis is more thorough, but also much more costly. This is a stateful analysis. Note that this analysis of the higher levels of the protocol also depends on the particular machine being protected, as implementations of the protocols are not identical from one network stack to another.

Using network sniffing as the information source has the following advantages:

- The detection of network-specific attacks. There are a number of network attacks, particularly denial-of-service attacks that cannot be detected in a timely fashion by searching for audit information on the host, but only by analyzing network traffic.
- The impact of auditing on the host performance. Information is entirely collected on a separate machine, with no knowledge of the rest of the network. Therefore, installation of such tools is facilitated because in terms of configuration and performance they do not impact the entire environment.
- The heterogeneous audit trail formats. The current de facto standardization towards TCP/IP facilitates the acquisition, formatting, and cross-platform analysis of the audit information.

But it also has a number of drawbacks:

- It is more difficult to identify the culprit when an intrusion has been discovered. There is no reliable link between information contained in the packets and the identity of the user who actually submitted the commands on the host.
- With switched networks (switched-Ethernet, switched Token-Ring, ATM), selecting an appropriate location for the sniffer is not obvious. Some tools are located on switches, others at gateways between the protected system and the outside world. The former gives better audit information but also has a higher cost. One has to realize, however, that switched networks are also much less vulnerable to sniffer attacks [20] and are actually a recommended solution to improve the security of a network.
- Encryption makes it impossible to analyze the payload of the packets, and therefore hides a considerable amount of important information from these tools. Also, even without encryption, it is possible to obfuscate the contents of the packet to evade detection [20] if the signatures are not suciently comprehensive.

Host-based information sources

Host audit sources are the only way to gather information about the activities of the users of a given machine. On the other hand, they are also vulnerable to alterations in the case of a successful attack. This creates an important real-time constraint on host-based intrusion-detection systems, which have to process the audit trail and generate alarms before an attacker taking over the machine can subvert either the audit trail or the intrusion-detection system itself.

System sources

All operating systems have commands to obtain a snapshot of information on what is happening. In a UNIX environment, examples of such commands are *ps, pstat, vmstat, getrlimit*. These commands provide very precise and focused information about the events because they examine the kernel memory directly. However, they are very difficult to use for continuous audit collection in intrusion-detection tools because they do not offer a structural way of collecting and storing the audit information.
However, this source of information is now implemented as dedicated kernel modules and does not use the commands referenced above anymore.

Syslog

Syslog is an audit service provided to applications by the operating system (UNIX and others). This service receives a text string from the application, prefixes it with a time stamp and the name of the system on which the application runs, and then archives it, either locally or remotely. Syslog is very easy to use, and that has incited many application developers to use it as their audit trail. A number of applications and network services use it, such as *login, sendmail, nfs, http,* and this also includes security-related tools such as *sudo, klaxon,* or *TCP wrappers*. Therefore, a few intrusion-detection tools have been developed that use information provided by the syslog daemon, an example of this approach being Swatch [11]. Although syslog is a lightweight audit source that does not generate a large amount of audit data per machine, a large network can generate a large number of messages, very few of which are security-relevant. Swatch [11] reduces the burden of the system administrator by correlating messages (several machines reporting that an nfs server is down would be aggregated into one) and highlighting security-related ones.

C2 security audit

The security audit records all potentially security-significant events on the system. As the US government has required that all computer systems it purchases be certified at the C2 level of the TCSEC, all operating system vendors competing in this area have had to include an "accountability" feature. This translates into security audit trails such as SUN's BSM and Shield packages, or AIX audit. All these security audit trails have the same basic principle. They record the crossing of instructions executed by the processor in the user space and instructions executed in the Trusted Computing Base (TCB) space. This security model postulates that the TCB is trusted, that actions in the user space cannot harm the security of the system, and that security-related actions that can impact the system only take place when users request services from the TCB.

In the UNIX environment, the TCB is basically the kernel. Therefore, the audit system records the execution of system calls by all processes launched by the users. Compared with a full system call trace, the audit trail provides limited abstraction: context switches, memory allocation, internal semaphores, and consecutive file reads do not appear in the trail. On the other hand, there is always a straightforward mapping of audit events to system calls.

The UNIX security audit record contains a great deal of information about the events. It includes detailed user and group identification (from the login identity to the one under which the system call is executed), the parameters of the system call execution (file names including path, command line arguments, etc.), the return code from the execution, and the error code.

The C2 security audit is the primary source of audit information for the majority of host-based intrusion-detection prototypes and tools because it currently is the only reliable mechanism for gathering detailed information on the actions taken on an information system. Work has been conducted by several groups [8, 17, 19, 27] to define what information should be included in the security audit trail as well as a common format for audit trail records, but this is an ongoing research effort.

Application log files

As the trend towards application servers becomes more pronounced, and the notion of operating system fades, application log files take a greater importance as a data source for intrusion detection.
Compared with system audits or network packets, the use of application log files has three advantages:

Accuracy

C2 audit data or network packets require processing before the intrusion-detection system can understand which information as actually been received by the application. This processing is based on interpretations of protocol specifications or API specifications, and it is very possible that the interpretation of the application developer is different from the interpretation of the intrusion-detection system developer. By obtaining the information directly from the log, the information is almost guaranteed to be accurate.

Completeness

C2 audit data or network packets require reassembly of several audit calls or several network packets, potentially on multiple hosts, to rebuild the session at the application level. This can be very difficult to achieve, and even simple reassembly, such as matching incoming http request and outgoing response to determine the success of a request, is not done by current tools. The application log contains all relevant information, even if the application is distributed over a cluster of machines (e.g web server or database server). In addition, the application can provide internal data that does not show up in audit trails or network packets.

Performance

By letting the application select which information is relevant for security purposes, the overhead induced by the collection mechanism is greatly reduced when compared with security audit trails.

There are two drawbacks in using application log files for intrusion detection:

Race condition

Attacks are only detected when the application log is written. If the attack can prevent the writing of the application log (this may be the case of many denial-of-service attacks), then the information needed by the intrusion-detection system is not there.

Low-level attacks

There are a number of attacks (again, particularly denial-of-service attacks), that tar-get the lower levels of the system software, such as network drivers. As these attacks do not exercise application code, they may not be seen in the application logs, or only the consequence of the attack (such as reboot), could be visible.

An example of such a tool is *WebWatcher* [1]. This tool monitors web server logs in real time and provides much mode detailed information about web server attacks than its network-based counterparts do.
A similar approach could be envisioned for database servers.

Misuse detection versus anomaly detection

There are two complementary detection technologies in intrusion detection, [1] to use the knowledge accumulated about attacks and look for evidence of the exploitation of these attacks, and [2] to build a reference model of the usual behavior of the information system being monitored and look for deviations from the observed usage. The first trend is often referred to as *misuse detection* [12, 13], but also as detection by appearance [26]. The second trend is referred to as anomaly detection [12] or detection by behavior [26].

Misuse detection

Misusedetection applies the knowledge accumulated about specific attacks and system vulnerabilities. The intrusion-detection system contains information about these vulnerabilities and looks for *attempts* to exploit these vulnerabilities. When such an attempt is detected, an alarm is raised. In other words, any action that is not explicitly recognized as a misuse of the information system is considered acceptable.

Advantages of the misuse detection approaches are that they have, in theory, very low false-alarm rates, and the contextual analysis proposed by the intrusion-detection system is detailed, making it easier for the security officer using this intrusion-detection system to understand the problem and to take preventive or corrective action.

Drawbacks include the difficulty of gathering the required information on the known attacks and keeping it up to date with new vulnerabilities and environments. Maintenance of the knowledge base of the intrusion-detection system requires careful analysis of each vulnerability and is therefore a time-consuming task. Knowledge-based approaches also have to face the generalization issue. Knowledge about attacks is very focused, dependent on the operating system, version, platform, and application. The resulting intrusion-detection system is therefore closely tied to a given environment. Also, detection of insider attacks involving an abuse of privileges is deemed more difficult because no vulnerability is actually exploited by the attacker.

In terms of techniques, knowledge-based intrusion-detection prototypes were first implemented using first-order logic and expert systems. Commercial products them mostly used a signature (i.e. pattern matching) approach. Additional techniques such as Petri nets and state-transition analysis have been proposed as well.

Anomaly detection

Anomaly detection assumes that an intrusion can be detected by observing a deviation from the normal or expected behavior of the system or the users. The model of normal or valid behavior is extracted from reference information collected by various means.
The intrusion-detection system later compares this model with the current activity. When a deviation is observed, an alarm is generated. In other words, anything that does not correspond to a previously learned behavior is considered intrusive.

Advantages of anomaly detection approaches are that they can detect attempts to exploit new and unforeseen vulnerabilities. They can even contribute to the (partially) automatic discovery of these new attacks. They are less dependent on operating-system-specific mechanisms. They also help detect “abuse of privilege” types of attacks that do not actually involve exploiting any security vulnerability.

The high false-alarm rate is generally cited as the main drawback of anomaly detection techniques because the entire scope of the behavior of an information system may not be covered during the learning phase. Also, behavior can change over time, introducing the need for periodic on-line retraining of the behavior profile, resulting either in unavailability of the intrusion-detection system or in additional false alarms. The information system can undergo attacks at the same time the intrusion-detection system is learning the behavior. As a result, the behavior profile will contain intrusive behavior, which is not detected as anomalous.

In terms of implementation techniques, intrusion-detection systems mostly use statistics. Prototypes using expert systems, neural networks, user intention identification and computer immunology have been proposed.

According to the previous taxonomy elements, all the commercial intrusion-detection system tested in this paper are network-based and follow a misuse detection paradigm oriented towards signature analysis.

Background on testing intrusion-detection systems

A number of papers related to testing intrusion-detection systems have been published in the litterature. In most cases testing methodology is proposed by the developpers of a given IDS method or tool, with the purpose of showing their own development at its best. This type of work is considered biased and was not studied in depth because it does not take into account commercially available intrusion-detection systems. During our litterature survey, we found three testing approaches which were sufficiently close to our preoccupations that we analyze them and decide if they are relevant to us.

The Lincoln Lab experiments

One of the best known testing experiments in the intrusion-detection community is the Lincoln Lab exper-iment [15, 14]. The initial purpose of this experiment is to test the various intrusion-detection technologies developped under DARPA funding, and to compare them in order to choose the most appropriate one.
This is achieved by simulating a network of workstations to create normal trafic and by inserting a set of attacks carefully chosen to cover various situations, summed up as remote attacks, local attacks and denial-of-service attacks.
This experiment has received scientific criticism from the community, most notably [16], and has evolved accordingly, up to the generation of a testing environment. This testing environment was not available at the time of testing, and we probably would not have used it for the following reasons:

■ **Focus on background traffic**
As the test includes behaviour-based intrusion-detection systems, realistic background data must be generated to ensure that these systems will be properly trained. However, these systems simply do not exist commercially and thus the need for background data does not exist for training. While background traffic generation is required for performance testing, the Lincoln Lab testbed does not provide a way of calibrating the characteristics of this background traffic and verifying their compliance with specifications.
Our testbed uses the LoadRunner tool from Mercury Interactive (1) to generate calibrated HTTP traffic.
This tool creates realistic HTTP requests to load the network links on the testbed. However, performance testing is not at the core of the paper and results are not included here.

■ **Focus on research prototypes**
The Lincoln Lab tests were commissioned by DARPA to evaluate DARPA-funded research work. No commercial intrusion-detection product was ever evaluated or taken into ac-count. For example, intrusion-detection products provide configuration management features, that we want to evaluate, and these aspects are not available with the Lincoln Lab tests.
The testbed includes reporting on the installation, management and integration for each tool. These results are extremely dependent on our procedures and environment and are considered out of scope for this paper.

■ **Focus on a broad set of attacks**
The Lincoln Lab tests aimed at exercising the largest possible set of attacks for the largest possible set of intrusion-detection systems. Our objective is to focus on network traffic close to firewalls, therefore the Lincoln Lab tests are too wide for our use. Also, attacks that are qualified as *local to root* are less relevant in telecommunication environment where monitoring is located on the wires.
Our testbed focuses on specific types of applications that are representative of the traffic profiles seen on our networks.

■ **Lack of reference point or baseline**
The Lincoln Lab tests compare prototypes in a closed circle. There is no notion of a minimal set of requirements that a tested tool has to meet, only relative data comparing them with each other.
Our testbed uses *Snort* [24] as a baseline.

The lack of a baseline that all tools would have to fulfill was felt as particularly lacking in the Lincol Lab experiment.

1. http://www.heva.mercuryinteractive.com/products/loadrunner/

The university and research work

The most representative work concerning university tests has been carried at UCDavis [3, 22, 21], with related and de facto similar work at IBM Zurich [5].

The objective of the UC Davis test is to simulate the activity of normal users and the activity of attackers, using script written in the *expect* language.
Expect simulates the presence of a user by taking over the input and output streams of a tty-based application, matching expected responses from the application with the displayed output, and feeding appropriate input as if the user typed it on its keyboard.

A similar approach was followed at IBM Zurich; in addition to user-recorded scripts the IBM approach introduced software testing scripts from the DejaGnu platform (also in *expect*) to ensure that all aspects of an application were exercised, regardless of the fact that they were obscure features of an application or not.

This testing methodology is closer to our own. In particular, the fact that tests are automated and reproducible is felt to be a very important property. However, we feel that the following points reduce the effectiveness of the test environment.

■ **Heavy to manage**
To achieve a significant level of background activity, the testbed must contain a significant number of machines, each of them piloting a number of users generating activity. This creates a complex environment to manage, and induces the risk of repetitiveness. Also, calibration to obtain data points at regularly spaced traffic rates is not taken into account.
Our testbed includes centralized distribution of software and management scripts that automate test run and result analysis.

■ **Limited in attack testing**
Requiring that attacks be scriptable using expect makes it unfeasible to use a number of exploit scripts collected from "underground" sources. Also, verifying the actual execution of each attack, verify its effect, and correlate with the IDS system is a manual process.
Our testbed does not solve this issue; installing and configuring the vulnerable software, meeting precon-ditions, running the attack, verifying the results and restoring the environment for future tests is manual as well.

■ **Applicability to commercial tools**
The UC Davis testbed has been designed primarily for research prototypes, and no information is available as to how commercial intrusion-detection systems would be included in such testing.

Commercial tests

Tests classified as commercial regroup the tests published by commercial test laboratories mandated by a particular company, and tests carried out by independant publications.

Several test labs have published test results related to intrusion-detection systems. In particular, Mier Communications has released at least two test reports, a comparative of BlackICE Sentry, RealSecure and NetProwler on one hand, and a test of Intrusion.com SecurenetPro Gigabit appliance. Appropriate queries on mailing list archives show that these test results have been the subject of many controversies. The general feeling is that it is appropriate for the testing laboratory to provide results that highlight the advantages of the product sponsored by the company financing the tests.
Even if direct test manipulation is not suspected, at least test configurations for the competing products is likely not to be handled by experts, thus resulting in unfair comparison.

Our feeling is that even this cannot be determined, as the description of the tests is very sparse and does not provide information that would allow an impartial judgement of the tests. Normal traffic generation, for example, is done using commercial products and the test report is considered complete with only the mention of the name of the traffic generator, without any information on the kind of traffic it generates or its configuration.

Journalists have also been involved in the testing of intrusion-detection systems. The most interesting article that we have found is by Mueller and Shipley [18].
It is the only comparative test of commercial intrusion-detection systems that we have found, where a number of intrusion-detection products are compared on equal footing and on live traffic, hence with a higher probability of either missing attacks or triggering flase positives.
The main drawback of this kind of test is reproductibility: when observing alerts it is quite difficult to reproduce the conditions in which these alerts are generated.
Also, we believe that the tuning phase carried out during testing initiation is problematic; the testers describe a process in which alerts that they believe are false are turned off. Our approach has been the opposite, keeping a maximum number of signatures active and tabulating both appropriate and false alerts. Doing this enables us to demonstrate the current trade-off of signature-based (and probably misuse-detection only systems), that a large number of patterns catches more attacks but also generates more false alarms.

IDS Evasion

Another, related work, is the work on *detection evasion*, particularly pointed by Ptacek and Newsham [20], and enhanced by Vern Paxson and al.[10]. We consider this to be of extreme relevance to our work since network-based intrusion-detection systems periodically are shown vulnerable to evasion techniques in one protocol or the other.

However, we also believe that carrying out this kind of tests is extremely difficult, and that we would not be able to add much value to the state of the art. Therefore, our testbed focuses on evasion at the application protocol layer, because we believe that application protocol analysis is still an area of improvement for commercial products.
There are in fact two issues with application-layer protocols:

■ **Misunderstanding of the protocol states or properties**
Sometimes, vulnerabilities are only applicable to certain states of the application layer protocols, or certain fields in the exchange. For example, *Sendmail* vulnerabilities usually apply only to the SMTP command mode. Intrusion-detection systems need to recognize these states and verify that state information is adequate before applying signatures.
Unfortunately, this sometimes require keeping state information, which can be costly. As a corollary, rarely-used states are sometimes ignored by the vendors for the sake of performance, and result in evasion opportunities for attackers.

■ **Misunderstanding of the protocol encodings**
Sometimes, protocols can encode data in a way which hides the information to the intrusion-detection system if it does not have the ability to decode it.
For example, unicode encoding on HTTP requests [9] can result in false positives. Worse, implementers of applications sometimes have hidden features or encodings that deviate from the published protocol specifications, such as the infamous *%u unicode encoding* issue with Microsoft Internet Information Server, which resulted in a lot of intrusion-detection systems not able to correctly parse HTTP requests and detect attacks [2, 25].

We take these two issues seriously. Our testbed currently uses Whisker [23] as a support for HTTP scanning and therefore has knowledge of HTTP evasion techniques. We are examining a transition to the libwhisker library which provides additional evasive capabilities.

The France Telecom R&D intrusion-detection testbed

From this background, we decided to develop our own testbed, reusing the interesting ideas and trying to improve where we felt there were weaknesses. Our testbed is segmented in five areas, described in page 57. Each of these areas contains a set of tests that can be executed with different parameters. Our testbed repeats as many executions of each test set with as many parameter combinations as relevant for the expected results.

Objectives of the test

While designing the testbed, we set a number of objectives that this design must meet when performing the comparative tests.

■ Fairness

The first objective of our testbed is *fairness*. The testbed must ensure that all products are treated equally, receive the same input and have a chance to correctly detect the attack.

To ensure that this objective is met, all network-based intrusion-detection systems being tested are located on the same hub and receive exactly the same traffic. Each appliance is equipped with two network interfaces, one for data acquisition purposes and one for management purposes. The two LANs to which these appliances reside are physically separated from one another. While this setup may not be feasible in all environments, it is actually the way other appliances are deployed, with a physical network dedicated to management purposes.

Also, during installation of the products on the testbed a maximum coverage policy is enabled to ensure that detection capabilities are at a maximum. This may be seen as unrealistic because of performance and operator overloading issues. While we share these two concerns, wa also believe that *reducing the capabilities of the intrusion-detection appliance actually shifts the burden from the machine to the analyst trying to make sense of the alert data generated*. The sparser the information delivered, the longer it will take the analyst to understand what is going on and assess the severity of the alert. As an example, certain HTTP attacks are dangerous only if the attacker uses the POST method; correlating the alert with the method helps the analyst assess the attacker's intention and success, therefore having the intrusion-detection system provide the data is faster than having the analyst search for it.

■ Repeatability

The second objective of the test is *repeatability*. In the intrusion-detection world, updates to both the detection software and the knowledge base are frequent, and linked together. Most vendors will deliver updates on a monthly basis; since recent attacks are usually more dangerous than older ones because of the time it takes to actually patch systems, and because of the inherent attractiveness of new exploits, updates must be taken into account as fast as possible.

We therefore expect that the testing process will have to be repeated on a regular basis, to ensure that the tools deployed in the field still perform as expected, and to apply regression testing to discover whether "older" vulnerabilities are still being detected, whether the false alarm rate has been improved, and whether performance is still comparable to the initial data. We also do expect that vendors shift focus, and that the ratings obtained at the end of the test may change over time. Regression testing will detect this shift, and allow us to either alert vendors or change product choices.

■ Automation

As the testing process is likely to be repeated, the automation objective seems self explanatory. However, it covers two different areas of the testbed, running the tests and exploiting the results.

Automating the execution of the tests on the testbed and collecting the data has been an important goal of our design. We use a collection of scripts centralized on a source server; every component of the testbed connects to the source server to obtain the latest version of the testbed software. If modifications are made on one of the components, upon commiting these changes to the script repository all other components benefit from the update. Since the entire battery of tests can take up to 3 days continuous execution, automation is indeed usefull. Note, however, that a few tests described in page 64 have not been automated. The main reason for this manual execution is the difficulty of checking all possible error conditions and verifying that the test actually succeeded.

Automating the exploitation of results is also extremely important. the usage of the testbed shows that a test batch generates several thousand alerts, from the 5 intrusion-detection systems being tested and the script running the attacks. All this information has to be correlated and tabulated to establish which tool caught which attack, and how it caught it. We have been able to automate most if this task through the use of a central alert repository accessed via syslog. While syslog lacks reliability, it works well in practice as long as the management network is not overloaded. Comparison between the content of the global syslog server and individual syslog files hosted on the intrusion-detection probes has shown that all data is correctly transferred. Also, syslog is easy to set-up and operate for most probes, and the one-line format eases post-processing.

Result exploitation is done through a set of perl scripts generating CSV tables. The analysis of the messages provided by the attack machine provides us with a linear outline of the complete test. Each of the parameter combinations is aggregated on one line to tabulate the results.

Baseline

We believe that meaningful tests require a baseline against which the products are tested. Not only do we want to compare intrusion-detection products, we also wish to establish if they are actually better than what the security community provides and maintain for free. The actual push towards open source in the security community is important; having the possibility to view the signatures and understand why an alert is triggered is of the utmost importance during analysis, and *Snort* [24] provides this facility.

Note that using *Snort* for baselining does not make it a competitor for our choice of an intrusion-detection platform, because it is not an appliance and does not have an integrated management platform. At the time of testing, the SourceFire company was forming and it is quite possible that a repetition of the tests at this time would see the open source *Snort* baselining the SourceFire appliance.

Description of the test protocol

We prioritized a set of 5 tests related to HTTP traffic, Trojan horse traffic and IP low level vulnerabilities and implemented them on the testbed. This priorities were set in accordance with the traffic distribution observed at our network boundaries.

The testbed layout is shown in Fig. 2. Two local area networks are used, the *control* for management and alert traffic and the *live* for observation of normal and malicious traffic. The *source server* is the central source repository, running a CVS server. The *victim* is the target of malicious activity. It runs an Apache web-server masquerading as both Apache and IIS, and has a set of vulnerable cgi scripts installed to carry out actual attacks. The *client* machines generate normal, mostly non-malicious traffic (2). The *Attacker* machine contains the attack scripts and drive the test sequences. The *Syslog server* receives alert data for analysis. The *Console* manages the intrusion-detection systems.

The following five test sets have been implemented in the testbed:

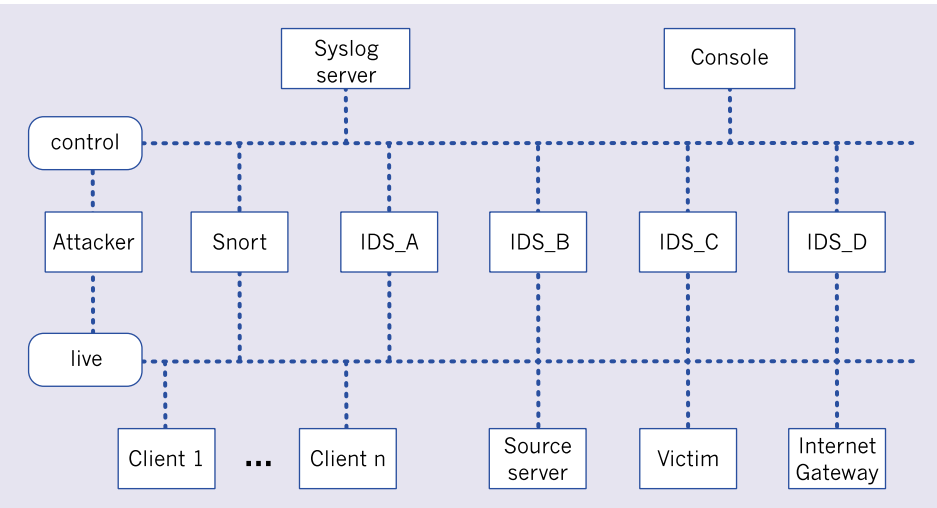


Figure 2 - The France Telecom intrusion-detection testbed.

IP Manipulation

Tests These tests are related to low level manipulations of the IP packet, such as *targa* or *winnuke*, that result either in denial-of-service or evasion of the intrusion-detection system. For these tests, we selected seventeen different vulnerabilities for which we have been able to find an adequate exploit tool, and to verify that this exploit tool indeed manipulates the TCP-IP packet sent over the wire. The reader is refered to vulnerability databases such as SecurityFocus (3) searching for the attack keyword identifying the attack tool.
Note that the objective of these tests is remote denial-of-service and does not targets the evasion methods described in [20], although some of these attacks can be relevant to these techniques.

Trojan horse traffic

These tests are related to the installation of Trojan horses on client machines, either classic, remote-command Trojans such as *BackOrifice* or *NetBus*, or distributed-denial-of-service Trojans such as *Tribal Flood Network* or *Trinoo*. Note that these tests concentrate on the detection of management traffic, and as such do not carry out denial-of-service attempts against the server. We installed these four Trojans on the testbed, ensuring that all the components were indeed available and running. The Trojans run unmodified, and as such use default publicly known communication ports, default command sets and unencrypted traffic. This test set could certainly be improved at least by the usage of non-default communication ports.

2. This area of the testbed is under development. An early version was used for performance evaluation, using the LoadRunner HTTP traffic simulation engine to load the live network.

3. http://www.securityfocus.com/

■ Whisker vulnerability scanning

These tests use the freely available whisker cgi scanner, repeating the scan with the default database of vulnerabilities with multiple evasion parameters. Note that our server is specially configured to answer appropriately when whisker attempts to verify pre-conditions before running the actual test, so that the maximum number of requests from the default database are run.
We expect intrusion-detection vendors to use whisker in their test suites, as we believe it is a tool actively in use for information gathering purposes, even though it is a bit old by Internet standards. Unfortunately, page 60 shows that the diagnostic of whisker scans is still lacking.

The same test database is used for 21 test runs, the first four with the GET method (- M command line flag), the next 9 with the HEAD method and the evasion parameter (- I command line flag) incremented from 1 to 9, and the next 7 with the HEAD method, and the evasion parameter set to - I 1 - I n with n varying from 2 to 9. Note that the - M command line switch is overriden for certain tests in the default database. Also, some tests are not carried out in some evasion modes, which results in a smaller number of requests reported by the Attacker.

■ Live cgi attacks

These tests carry out real attacks against our vulnerable HTTP server. These attacks have varying results, some of them giving a shell with httpd user privileges and the others allowing to visualize files. At this stage, this set of tests has to be carried out manually to ensure that pre-conditions are met, that compromise effectively happens and to restore the server to its original state.
Our file target is the /etc/passwd file, to ensure that all intrusion-detection products would have a chance to see abnormal activity. While other files are worthy of an attacker's attentions, the only common one that we have found registered accross all intrusion-detection systems is the unix password file.

■ Whisker signature evaluation

These tests use the freely available *whisker* cgi scanner, with a specially crafted database taking into account the list of HTTP vulnerabilities that all products document, and constructing creative requests to evaluate the extend and diagnostic capability of each signature. The goal of this test is to verify that signatures listed in the product documentation do trigger and to approximate the trigger that sets the alert off. We expect this information to be very valuable for analysts assessing the alerts representing the diagnostic of the tested intrusion-detection systems, particularly if detailed alert description is not available (4).

The test is constructed around a specific whisker database, containing a set of URLs to test for each vulnerability/signature that we wish to evaluate. This set contains URLs related to different attacker activity, such as scanning (attempting to infer whether the vulnerability is present on the system; this is the general operation of whisker), normal activity (attempting to reproduce requests that users earnestly trying to use the vulnerable application as specified would submit), outright malicious activity that would directly exploit the vulnerability, and abnormal activity that either extends the direct exploits with our own knowledge or derives from these direct exploits. Clearly, a lot of time is spent analyzing each of the vulnerabilities and figuring out devious ways of either using the vulnerability but with different goals than the "default" exploits published, or transforming these so that the requests look similar without being exactly the same. Examples are given in page 65 that covers the results of this set of tests.

Clearly, this does not cover all possible intrusive activity. Future extensions of the testbed include generating normal and abnormal DNS, mail (SMTP, POP and IMAP) RPC and file (NFS, Samba) traffic to broaden the coverage of the test. Our emphasis on HTTP is justified by its status today as an ubiquitous transport protocol, used not only for page serving, but also for additional traffic simply because it can traverse the firewall protecting most organizations today. Also, most content is served from web servers even if the server only acts as a mediation portal.

4. This is the case for most vendors.

Results obtained during the tests

This section presents the results obtained during a complete run of the tests. The intrusion-detection systems will be identified as IDS-A, IDS-B, IDS-C and IDS-D, representing four of the five commercial leaders in the field. We decided against explicitely naming the product because all of them exhibited significant (although not the same) shortcomings and we do not wish these results to be interpreted as an endorsement of these products. In fact, the shortcomings identified lead us to believe that none of them would be satisfactory for our demanding environment.

Results of the IP manipulations tests

The following table shows the results obtained. The first column identifies the attack, the next five the number of different alerts that were considered valid for each IDS, and the final five the number of alerts that were considered false alerts for each attack. The summary counts the number of events flagged and the total number of different alerts for each tool in each category.

The first observation from the results is that none of the tested intrusion-detection systems detects the entire set of attacks. In fact, two of them, *gwese* and *octopus*, are not detected at all; this is not surprising since both of them are very specific, windows-95 denial-of-service attacks and as such are probably rare in the field. Note, however, that the best tool only detects ten out of seventeen trials; we believe that this number is quite low and should be improved.

Second, note that the best IDS in terms of detection, IDS-A, is also the one generating the largest number of false alarms. This reveals an issue with signatures, that are probably not tuned enough to differentiate the real attack from symptoms that would also exist with other vulnerabilities, but are not significant simply by themselves. *Clearly, the tradeoff between accuracy of diagnostic and coverage has been set towards accuracy for* IDS-B *and coverage for* IDS-A; hence the *better* intrusion-detection system depends on the

Table 1 : Results of the IP manipulation tests

Attack Name	Appropriate alerts					Irrelevant alerts				
	Snort	IDS-A	IDS-B	IDS-C	IDS-D	Snort	IDS-A	IDS-B	IDS-C	IDS-D
papabroadcast	1	1	1	0	0	0	0	0	0	0
pinger	2	1	0	0	0	2	0	0	0	0
gewse	0	0	0	0	0	1	1	0	0	0
nestea	1	1	1	0	0	1	0	0	0	0
newtear	1	1	1	0	0	1	0	0	0	0
targa2-bonk	0	0	0	1	0	1	0	0	0	0
targa2-jolt	1	3	0	0	0	0	1	0	0	0
targa2-land	0	0	1	1	0	1	1	0	0	0
targa2-syndrop	1	1	1	0	0	0	0	0	0	0
targa2-winnuke	0	1	1	1	1	0	0	0	0	0
targa2-1234	0	2	1	0	0	0	0	0	0	0
targa2-sayhousen	0	3	0	0	2	0	0	0	0	0
targa2-oshare	0	0	0	0	0	0	0	0	0	0
kkill	0	0	1	0	0	0	2	1	0	0
octopus	0	0	0	0	0	0	0	0	0	0
overdrop	0	0	0	1	0	0	0	0	0	0
synful	0	1	1	0	0	1	1	0	0	0
Number of events flagged	6	10	9	4	2	7	5	1	0	0
Number of alerts sets	7	15	9	4	3	8	6	1	0	0

particular needs of an organization. We would add that we are not happy by either side of the tradeof, which means on one side more analyst time and on the other side missed attacks. Also, the Snort yardstick shows that IDS-C and IDS-D are not up to date with the possible attacks against an IP stack, letting the door open to potential denial-of-service attacks.

Third, note that table 1 counts the number of different alerts for each intrusion detection system, not the total number of alerts generated per attempt. This number of different alerts is important for an analyst because it gives him more information about the ongoing attack, and gives more meat to a correlation system.
Using this measure shows that not only does IDS-A detect the largest number of attempts, it also is the one giving us the most information about the malicious activity going on. The conclusion reached here is similar to the one in the previous paragraph.

Finally, the table does not measure the number of alerts generated for each attack. Some attacks generate a large number of packets, each of them carrying the anomalous characteristic and triggering an alert. This characteristic has already been shown in [6], and we confirm this result; clearly this is a case where aggregation of consecutive, similar alerts is desirable and should be performed by the probe since it has all the elements to do so.

Results of the Trojan horses tests

Concerning the Trojan horse tests, all Trojans were detected by all intrusion-detection systems except IDS-D which did not detect any of them and generated one false alarm. Snort in addition generated two false alarms. Since default ports and keywords were in use, these results are exactly what's expected and the only valid test result is that functionality specified in the documentation

Results of the Whisker vulnerability scanning

The results are presented in Table 2. The first two columns indicate the total number of whisker request and the number of requests that actually deliver information outside of directory existence. Then, for each intrusion-detection system, the table gives the total number of alerts generated, the number of whisker requests that triggered an alert, and the number of whisker directory events that triggered an alert.

The table is incomplete, because of a number of circumstantial issues during testing. We were unable to configure IDS-D to send events to our syslog server; given that this test generates several thousands of events, it is impossible to manually reincorporate the results of IDS-D into the log file. It also proved impossible to use custom log files to carry out this task and we finally gave up, having the impression from screen observation that the box was performing in about the same way as the other ones. Also, during the final rounds of the tests our data collection network broke down and therefore the last three lines of the experiment should be discounted. Since such a test run takes more than 3 days, we were not able to keep the intrusion-detection systems reliably running for that amount of time, at a rate of about one alert per second each. We propose the following comments for these results:

■ **Missed events**
The obvious remark from this table is that many scan events are not flagged as anomalous; the commercial intrusion detection systems flag between 10 % and 20 % of the scan events, and Snort goes up to 30 %, but at the cost of many false alarms. This actually is a very reasonable tradeoff, because this offers some resistance to alert flooding. The most verbose intrusion-detection probe across the board is Snort, which systematically generates 3 to 6 times more alerts than the other probes.

■ **Missed summary of scan activity**
Note that this test carries out a scan, not actual attacks. As such, there is no real attack traffic going on the network, only suspicious activity. The best response from an intrusion-detection probe facing this kind of traffic would be a single event, or the same event repeated at regular intervals with update information. What we see here is a chain of uncorrelated events (uncorre-lated in the sense that the intrusion-detection probe does not link them, because an obvious correlation chain from IP address and monotonically increasing port number exists in the logs) that requires an operator manual analysis, not the complete diagnostic that would bring the expected value from this tool. Snort is the only tool to indicate that, with some evasion modes, it is facing a whisker scan; this takes the form of additional alerts in evasion modes - I 3, - I 4, - I 5 either alone or combined. IDS-A and IDS-C without explicitly mentioning whisker at least give an indication of the evasion mode for some of them.

■ **Missed identification of targets**
A follow-up observation is that since this is merely a scan and not actual attacks, all of the alerts generated could be considered false alarms, from the point of view of an operator who would be looking for successful intrusions, or even intrusion attempts. A scan here attempts to assert that superficial traces of a vulnerability exist on the scanned web server. It does not attempt to use the potential vulnerability to break into the server. This unfortunately points out the fact that the intrusion-detection signatures implemented are attached to very superficial characteristics of the attacks and do not take into account symptoms indicating a really malicious attempt. The only anomalous symptom that the probes seem to look for is a request for the UNIX password file at the well-known location /etc/passwd. Our scan included attempts against other sensitive files (.rhost, /etc/exports, /etc/shadow) without the probes noticing them.

Table 2 : Results of the Whisker vulnerability scanning

Whisker		Snort			IDS-B			IDS-A			IDS-C		
Total	Scans	Total	Event	Dir	Total	Event	Dir	Total	Event	Dir	Total	Event	Dir
548	347	145	144	7	75	63	5	33	33	4	131	77	5
548	347	145	145	7	30	28	2	33	33	4	131	77	5
548	347	145	145	7	30	28	2	33	33	4	130	77	5
548	347	145	145	7	30	28	2	33	33	4	132	77	5
548	347	145	145	7	75	65	5	33	33	4	197	67	3
548	347	92	92	5	75	65	5	47	33	5	1366	69	3
548	347	556	538	205	75	65	5	568	33	5	1176	70	3
548	347	872	539	322	75	65	5	33	33	4	319	60	5
548	347	554	542	203	75	65	5	33	33	4	117	69	3
548	347	144	144	6	75	65	5	33	33	4	121	70	4
241	71	26	26	6	1	1	0	2	2	0	17	9	1
522	325	76	60	6	73	65	5	17	17	4	118	66	3
95	68	792	95	27	9	8	1	7	7	1	220	17	2
548	347	100	92	5	75	65	5	47	33	5	1146	52	1
548	347	556	541	203	75	65	5	570	33	5	1171	69	3
548	347	874	541	319	75	65	5	33	33	4	326	57	4
548	347	553	541	203	75	65	5	33	33	4	116	69	3
548	347	144	144	7	75	65	5	33	33	4	120	70	4
241	71	26	26	6	1	1	0	2	2	0	17	9	1
522	325	76	60	6	73	65	5	17	17	4	111	66	3
39	35	344	38	4	4	3	0	2	2	0	0	0	0

■ Ignorance of directory scanning

As shown in the *Dir* columns, Most probes fail to detect directory scanning. This is a reasonable way to practice unless directory browsing is enabled on the web server.

■ Continuing issues with evasion modes

Attack detection is very much dependant of the kind of evasion mode that is activated. Evasion mode - I 8 is particularly deadly, as it carries out most of the scanning activity with a very low detection probability. IDS-C resists better to the various evasion modes that the others, which indicates that the analysis engine has at least some notion of what an HTTP session should look like.
The reason why this change from "/" to "\" is deadly for the probes is that many of them use the "/" character in the signatures to anchor the pattern;
a signature pattern for the phf vulnerability could look like "/phf?" to indicate that the phf string is at the end of the request string and needs arguments. If the translation from "\" to "/" is not done (although IIS at least and probably all web servers running on windows platforms must do it) then the trigger is missed.
Even though it also shows failures from the probes, mode - I 7 is not as deadly for the server, as it applies mostly to directory scanning; this gives an attacker some information about the directory structure of the server and as such some information about potential weaknesses, but it would still require a lot of work to actually break in the site.

Reaction of the various intrusion-detection probes to evasion varies greatly. Snort and IDS-C (and IDS- A to a lesser extend) become extremely verbose with certain modes. In the *long URL mode* (- I 3, - I 4 or - I 5), they also generate a number of false alarms because the random characters inserted match existing, too simple signatures.

This test shows that even though progress has been made by the intrusion-detection probes with respect to interpreting the various subtleties of HTTP protocol encoding (a similar test two years ago showed all probes failing detection even with the simplest encoding tricks), they could still be improved.

Table 3 : Structure of the alerts of the Whisker vulnerability scanning

	Snort					IDS-B					IDS-A					IDS-C				
L	T	G	A	B	E	T	G	A	B	E	T	G	A	B	E	T	G	A	B	E
0	145	140	5	0	0	75	1	3	0	71	33	24	9	0	0	131	60	16	1	0
1	145	140	5	0	0	30	12	15	0	3	33	24	9	0	0	131	59	17	1	0
2	145	140	5	0	0	30	12	15	0	3	33	24	9	0	0	130	59	17	1	0
3	145	140	5	0	0	30	12	15	0	3	33	24	9	0	0	132	59	17	1	1
4	145	140	5	0	0	75	1	3	0	71	33	24	9	0	0	197	53	13	1	51
5	92	88	4	0	0	75	1	3	0	71	47	24	9	0	14	1366	53	15	1	743
6	556	124	5	0	427	75	1	3	0	71	568	24	9	0	535	1176	53	15	2	553
7	872	105	3	8	756	75	1	3	0	71	33	24	9	0	0	319	40	16	4	112
8	554	138	5	1	410	75	1	3	0	71	33	24	9	0	0	117	53	15	1	0
9	144	139	5	0	0	75	1	3	0	71	33	24	9	0	0	121	50	19	1	0
10	26	24	2	0	0	1	1	0	0	0	2	2	0	0	0	17	7	2	0	0
11	76	50	4	0	22	73	1	1	0	71	17	14	3	0	0	118	54	11	1	3
12	792	0	0	0	792	9	1	3	0	5	7	7	0	0	0	220	10	7	0	102
13	100	88	4	0	8	75	1	3	0	71	47	24	9	0	14	1146	43	8	1	641
14	556	124	5	0	427	75	1	3	0	71	570	24	9	0	537	1171	53	15	1	551
15	874	96	5	4	769	75	1	3	0	71	33	24	9	0	0	326	41	14	2	121
16	553	138	5	0	410	75	1	3	0	71	33	24	9	0	0	116	53	15	1	0
17	144	139	5	0	0	75	1	3	0	71	33	24	9	0	0	120	50	19	1	0
18	26	24	2	0	0	1	1	0	0	0	2	2	0	0	0	17	7	2	0	0
19	76	50	4	0	22	73	1	1	0	71	17	14	3	0	0	111	54	11	1	0
20	344	0	0	0	344	4	1	3	0	0	2	2	0	0	0	0	0	0	0	0

Table 3 analyzes in more detail the structure of the alerts generated by the intrusion-detection systems.
The total number of alerts is given in columns *T*. Then, alerts are classified into 4 groups: *G* for good alerts giving meaningful information about the event (although maybe not as exhaustive as what we whould like), *A* for approximate, giving information that is correct but marginally relevant for the analysis of the actual event, *B* for bad alerts that are clearly false alarms, and *E* for alerts related to the evasion method used rather than the actual event. The definition of these groups is configured into the result analysis tool.
Concerning the *E* (Evasion) column, it means different things for each probe.

For IDS-B, almost all alerts classified in that category indicate that the probe has caught a HEAD request. Such an alert is not systematically generated for every HEAD request. The most likely hypothesis at this stage is that there is a second factor which is required for the creation of the alert, yet unidentified.
Some scan events not generating an alert in modes 1-3, but fairly close in patterns (e.g. perl and perl5) do give a *HEAD* alert. A second hypothesis is then that IDS-B understands that there is an anomaly with the request and generates alerts with a slightly broader coverage in terms of pattern matching. Finally, the HEAD alert is sometimes doubled, e.g. two messages arrive in the syslog file for only one scan event. As such, it is unclear whether it is a bad alert or a useful one.

For IDS-A, evasion almost exclusively means that it has seen the infamous "../.." string in the URL. This pattern is inserted by whisker's mode 4 and is flagged by IDS-A as an attempt to exploit an IIS vulnerability, which is obviously an erroneous interpretation (even though the alert should not, in our opinion, be classified as bad because the request indeed presents the required characteristics).

For IDS-C and Snort, it is a mixed bag of things, "../.." strings and also specific messages targeting whisker modes such as the directory traversal of mode 4 and the splicing technique of mode 9. The first observation from this table is that Snort and IDS-C are the only intrusion-detection probes that generates obviously-false alarms (not that the others never do, but testing for false alarms was beyond the purpose of the tests). For Snort, this is due to the overly simplistic signature database that matches on strings found in the *long URL* evasion mode (7 and 15). For IDS-C, the same *long URL* mode matches on pornographic signatures. IDS-C also matches "perl" on the "perlshop.cgi" scan event; this is considered a false alarm because it matches on part of the file name only and should be corrected.

The second diagnostic is that the classification of alerts in *G*, *A* or *B* is not influenced significantly by the evasion type (except for IDS-B). The total count drops a bit or stays steady, and only the *E* column increases. This means that filtering the evasive alerts would make the diagnostic acceptable for an operator, without loosing too much accuracy.

As a comparison between IDS-B and the other probes, only evasion modes 1 to 3 can be used (due to the *HEAD* phenomenon mentioned earlier). This shows that both IDS-B and IDS-A are almost equivalent in terms of performance. IDS-C has a slightly better score, with the cost of additional false alarms.

Although the "perl" signature from IDS-C is overly large, while other signatures are very precise and distinguish between locations. For example, it matches on /mlog.phtml, but not /cgi-bin/mlog.html, whereas Snort matches on both. This indicates that the signatures from IDS-C can probably be enhanced to better take into account the attack conditions, which is an important possibility to reduce the number of false positives.

Results of the live cgi attacks

The results of these tests are presented in Table 4. Note that when an attack is not caught through generation of one or several alerts, an actual compromise of our victim web server is not diagnosed and reported. Therefore, missed attacks are counted as a very bad point, especially since the attacks were carried out using parameters that are known to be embedded in signatures, such as /etc/passwd.

Our test includes 18 attacks. IDS-A and *Snort* are the two best probes on this test, generating at least one alarm for 15 of the 18 attempts. This could look like a very nice result, but is actually not so. In cases where the tools give us only one alert they only catch the presence of the /etc/passwd string on the request; they do not identify the vulnerability itself. Quite clearly, the fact that a request targets the password file is important and must be flagged, even if the vulnerable CGI script is not known to the intrusion-detection system. However, the fact that only the password file is known as an anomalous symptom shows that the designers of intrusion-detection systems have little imagination in terms of attackers targets.

Also, this ranking changes when analyzing the accuracy of the diagnostic proposed by the intrusion-detection system. IDS-C and IDS-D do a much better job at evaluating the extend of an attack. They both diagnose multiple aspects of the attack, such as the name of the vulnerable script used as the attack vector, directory traversal activity, request for /etc/passwd and indicate whether the attacker's actions have been successful. IDS-C in particular in one instance indicated us that the HTTP request for the password file was successful, and in another instance indicated that something that looked like a password file was being sent out of the protected network. These two elements show that extended diagnostic while not generally available is indeed possible and that this extended diagnostic eases the analyst's work.

Table 4 : Results of the Whisker vulnerability scanning

Attack Attack name	Appropriate alerts					Irrelevant alerts				
	Snort	IDS-A	IDS-B	IDS-C	IDS-D	Snort	IDS-A	IDS-B	IDS-C	IDS-D
accesscounter	0	0	0	0	0	0	0	0	0	0
aspseek-xpl	0	0	0	0	1	0	0	0	2	0
bizdb	1	1	0	0	1	0	0	0	0	0
clickrespond	0	1	0	0	0	1	0	0	0	0
clipper	1	1	1	2	2	0	0	0	0	0
coldfusion	1	0	0	0	0	0	0	0	0	0
finger	1	1	1	1	1	0	0	0	0	0
handler	1	2	0	3	1	0	0	0	0	0
htdig	1	1	1	1	1	0	0	0	0	0
htgrep	1	1	1	1	1	0	0	0	0	0
phf	1	2	2	3	3	0	0	0	0	0
php-nuke	1	1	0	0	0	0	0	0	0	0
php	1	2	2	2	2	0	0	0	0	0
search	1	1	1	1	2	0	0	0	0	0
search	2	1	1	2	2	0	0	0	0	0
viewsource	1	2	2	2	3	0	0	0	0	0
webspirs	1	1	1	3	2	0	0	0	0	0
whois	1	1	0	2	0	0	0	0	0	0
Number of attacks found	15	15	10	12	13	1	0	0	1	0
Diagnostic accuracy	16	19	13	23	22					

Of course, this extended diagnostic still does not meet our expectations.
In particular, it does require an analyst to manually correlate the three or four alerts related to the attack, as the intrusion-detection probe merely provides the alerts side by side without eliciting any relationship between them. In addition, each alert keeps the evaluation of its own severity, whereas the most relevant information would be an aggregated and ponderated severity of all related alerts.

Results of the signature evaluation tests

The results presented here are related to three vulnerabilities out of the 60 that the testbed currently checks.
They are representative of all observed behaviours.

Figure 3 shows an extract of the whisker database file for the evaluation of the *add.exe* vulnerability.
For this vulnerability, four requests are sent, although the first two could be considered equivalent be-cause the HTTP standard does not differentiate between the absence of arguments to a CGI script and the empty argument. Therefore, the first two requests result in starting the add.exe script with an empty QUERY_STRING. The third request passes an argument to the CGI script in an attempt to simulate normal usage and the fourth one constitutes the effective attack.
For each of the requests the testbed receive the same ADD.EXE alert.
This suggest that the pattern detecting the malicious activity for all of the intrusion detection systems is similar to (in perl-like regular expression syntax) "/add\.exe".

```
3.1: scan () @roots >> add.exe
3.2: scan () @roots >> add.exe?
3.3: scan () @roots >> add.exe?foo
3.4: scan () @roots >> add.exe?C:\inetpub\iissamples\default\samples.asp
```

Figure 3 - Signature evaluation for the add.exe *vulnerability*

In each case, the attacker receives different information. In each case, the diagnostic provided by the intrusion-detection systems is the same. As such, we consider this diagnostic incomplete. A normal request to the script is flagged as anomalous by the intrusion-detection system with exactly the same severity as a scan or an intrusive attempt. Our conclusion is that for every alert generated, valuable analyst time must be spent on assessing the activity whereas the intrusion-detection system could automate the process.

Figure 4 shows an extract of the whisker database file for the evaluation of the *Cart32* vulnerability.
Again, the first two requests are related to scanning and the third one to normal activity (although normal activity is more complex than that). The two last lines are direct malicious attempts exploiting two different *Cart32* vulnerabilities.

The tested intrusion-detection systems generate an alert only on the last request (line 4.5). Neither the scanning activity of the 2 first requests or the attack attempt from the fourth generate any noise, altough any analyst will consider that the one before last (line 4.4) looks extremely suspicious. This case illustrates that the signature for the Cart32 vulnerability is too restrictive, because additional vulnerabilities discovered more recently and affecting the same base software are not covered by the signature.

```
4.1: scan () @roots >> c32web.exe
4.2: scan () @roots >> c32web.exe?
4.3: scan () @roots >> c32web.exe?foo
4.4: scan () @roots >> c32web.exe?TabName=Cart32%2B &Action=Save+Cart32%2B+Tab &SaveTab=Cart32%2B
&Client=foobar &ClientPassword=e%21U%23 %25%28%5D%5D%26%25*%2B-a &Admin= &AdminPassword=
&TabToSave=Cart32%2B &PlusTabToSave =Run+External+Program &UseCMDLine=Yes
&CMDLine=cmd.exe+%2Fc+dir+%3E+c%3A%5Cfile.txt
4.5: scan () @roots >> cart32.exe/cart32clientlist
```

Figure 4 - Signature evaluation for the Cart32 *vulnerability*

```
5.1: scan () / >> vti bin/shtml.exe
5.2: scan () / >> vti bin/shtml.exe?
5.3: scan () / >> vti bin/shtml.exe?toto
5.4: scan () / >> vti bin/shtml.exe/prn
5.5: scan () / >> vti bin/shtml.exe?prn
```

Figure 5 - Signature evaluation for the shtml *vulnerability*

This does not contradict our analysis of the *Add.exe* vulnerability test (too simple signatures giving too many alerts). Our analysis proceeds from the same logic, namely that in both cases the diagnostic provided is extremely imprecise and requires manual analysis to verify the exact circumstances of the attack and assess the extent of the damage. Assessing the alert with the appropriate adjacent and related acivity would allow broader signatures (hence less misses) with varying severity levels (hence facilitating the analyst's work).
This analysis is expanded in page 68.

Figure 5 shows an extract of the whisker database file for the evaluation of the *shtml.exe* vulnerability. The real attack is on line 4.4, line 4.5 representing a variation of ours to check whether the signature implemented in the tested intrusion-detection systems includes the slash between the components or not.

The interesting result from this experiment is that one of the tested commercial intrusion-detection sys-tems crashed hard on the four requests not specifically targetting the vulnerability while correctly identifying the malicious attempt.
This illustrates the difficulty of testing network-based intrusion-detection systems. Our suspicion is that vendors of these network-based intrusion-detection systems test them using a database of network frames representative of instances of the execution of a given vulnerability, without rebuilding the vulnerable environment and testing with live attacks. Therefore, attack variants that could be introduced accidentally because the operating system handles network traffic a bit differently, or intentional variants such as ours, are not tested and end up introducing faults in the software.

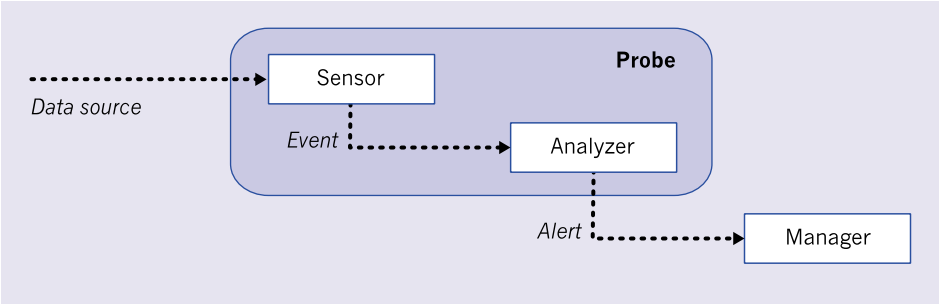
Proposed model for an intrusion-detection system

Our test results clearly show that there is still a lot of work to be carried out within the intrusion-detection community to improve the systems that are being deployed today. We believe that the most serious issue highlighted by our tests is the *diagnostic* issue, namely the lack of details in the information transmitted to the analyst whereas useful information could be automatically extracted from the data source. From this, we take a look at the current model for intrusion-detection systems proposed by the intrusion-detection working group (IDWG) of the IETF and propose several improvements that would result in better diagnostic.

The IDWG model of an intrusion-detection system

The model of an intrusion-detection system proposed by the IDWG in its requirements document [28] has several components and the interested reader is refered to the draft for the complete description. We really are interested here in the analysis (boxes) and data (italics) components as shown in Fig. 6.

Figure 6 - The IDWG intrusion-detection system model



Concerning the data components, the *data source* provides raw information tapped by the sensor. The sensor parses and formats information available in the *data source* and into *events* that are sent to the ANALYZER. The ANALYZER processes the events to realize the actual detection process and provide *alerts* indicating that malicious activity has been discovered. The *alerts* are sent to the MANAGER for management and further processing. Note that the model provides for multiple flows of information, for example for multiple SENSORS to send *events* to one ANALYZER. Also, a component could play both the role of an ANALYZER for dialogs with upstream components and the role of a MANAGER for downstream components.

Let's take the example of one of the tested network-based intrusion-detection systems. The *data source* is the network packets that are sent on the wire. The SENSOR taps the network and formats the information retrieved, for example applying the anti-evasion techniques presented in [20] and possibly further decomposing the information to retrieve protocol states. The ANALYZER then takes this information and decides according to its attack signatures to generate an alert. Note that this example collapses the SENSOR and ANALYSER components together into an opaque box that performs both data acquisition, formatting and analysis.

Our proposal for an enhanced model

Figure 7 presents our extension to the IDWG model. It is indeed a very simple extention, consisting of the insertion of a feature extraction mechanism (extractor taking *events* and providing *features* to the analyzer. While this seems like a minor change, we do believe it is important particularly in light of the test results shown in page 64 and 65. The intrusion-detection systems tested to not provide a complete analysis, but in fact limit themselves to the feature extraction stage. Each feature is extracted from the data source and sent to the management console independantely of the others, regardless of the fact that the features may be linked and loosing the link information in transit.

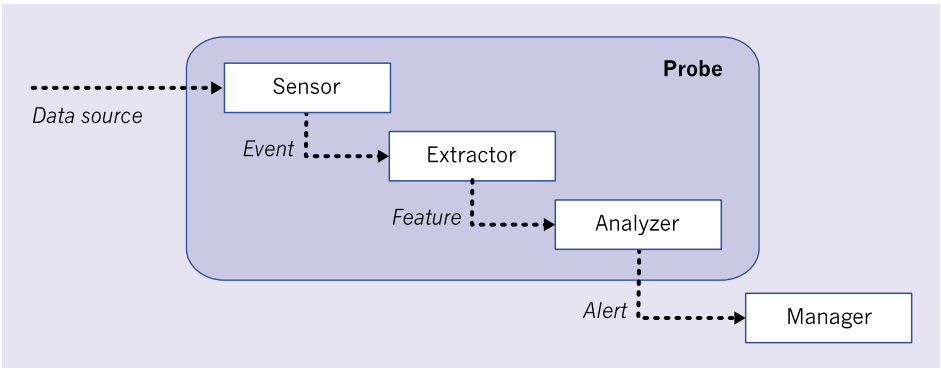
In addition, the quest for performance seems to push intrusion-detection vendors into not performing the complete feature extraction stage. Matching multiple patterns has a cost; exiting early from the search loop allows vendors to claim higher processing rates and gigabit capacity, which is often needed by customers. However, customers often do not realize that trading performance for accuracy will deter them from actually using the intrusion-detection system. This quest for performance also impacts the sensor, limiting the reconstruction of network data to IP and TCP fragmentation rather than analyzing the actual TCP session from establishment to teardown.

Let's take the example of an HTTP request for the password file using the *phf* vulnerability, against a vulnerable server. The request sent on the wire could look like GET /cgi-bin/phf?/etc/passwd HTTP/1.0. From this request, all tested intrusion-detection systems provide us with two alerts, PHF and PASSWORD, representative of the two saliant features of the URL. However, the two alerts are juxtaposed, not linked, event though they are extracted from the payload of the same packet.

IDS-C and IDS-D in addition to these two provide SUCCESS and/or PASSWORD-EXTERNAL, indicating that they are monitoring not only the incoming request but also the response from the server, and that they extract additional features from the response. This shows that monitoring sessions rather than packets is feasible and informative.

The limitation we see is that all the alerts are shown on the interface without indication that they come from the same TCP session, or even from the same IP packet; the analyst must manually notice that IP, port and url information is identical and that timing is identical or very close, and infer that they are related. Similarly for the response, port and IP information is reversed but the analyst matches it. We believe that the intrusion-detection probes have a much better case than the analyst to make the correlation between the alerts. Not only do they have the information displayed, they also definitively know if the features were in the same packet, or in the same TCP session due to matching sequence numbers. *We see no valid reason for the probes not to carry out this simple task, and give us a single alert fully assessing the damage created by each malicious activity.*

Figure 7 - Our proposed intrusion-detection system model



Damage assessment and alert qualification

This notion of *assessment* also allows to go beyond the current proposed severity level, which only takes into account the intrinsic damage caused by a successful exploit of a vulnerability. We wish that the diagnostic would propose at least one of the following qualifiers:

■ **Scan**
The attacker tries to determine the existence of a vulnerability. The main characteristic of a *scan*-qualified alert is that no actual compromise can result from the attacker's actions even though he gains information about the environment.

■ **Exploit**
The attacker tries to exploit the vulnerability, possibly with well known attack scripts, well known targets or publicly available information. An *exploit*-qualified alert indicates a clearly identified attack that can result in actual compromise if the target is vulnerable.

■ **Variant**
The attacker tries to exploit the vulnerability by using the application outside of its specifications, but the target or exploit could not be definitely recognized.
A *variant*-qualified alert indicates that there is a possibility for compromise or leakage of information.

In addition, the *success* of the malicious activity and its *relevance* to the monitored information system should be included in the alert. In some of our operations, we are simply not interested in unsuccessful malicious activity, because there is simply too much. We just want to know when they succeed, and then be able to react immediately and accurately. This would enable an evaluation of the severity of each alert based not only on the intrinsic characteristics of the vulnerability exploited but also on the actual exploit circumstances.

Note that these notions have been included in the message format of the IDWG working group [4] for some time, but have not yet found their way into products, although the relevance of attacks is starting to be evaluated in post-mortem commercial analysis products by crossing vulnerability assessment reports with intrusion-detection alerts.

Impact on alert correlation

Alert correlation is currently being pushed heavily in the research community as being the way to solve the diagnostic accuracy issue exhibited by many current commercial intrusion-detection systems. Indeed, we also are part of this trend and working towards alert correlation as a way to reduce the number of alerts that are being processed by operators and analysts. Adding the *feature* extraction in our model pushes some correlation functionalities down in the probes, thus reducing alert trafic and ensuring that the MANAGER is devoted to alert correlation from different probes.

We believe that the EXTRACTOR layer in the probe is in fact a fairly simple engine, extracting *features* from a single *event*; we do not envision a feature extraction mechanism that would search for a feature accross two events. In such case, the *feature* should be split in two sub-*features*, each of them covering one event, the recomposition of the two sub-*features* being realized by the ANALYZER. As such, the analyzer could provide a diagnostic based on a single *event* as well as multiple *events*.

A number of research projects have looked at using existing intrusion-detection alerts and correlating them to improve the quality of the diagnostic and lower false positives. An exemple of this trend is the Tivoli RiskManager product [6]. Looking at it from the feature extraction viewpoint, a much better solution is to improve the probes themselves, for the following reasons:

- They have more data internally than in the alerts they provide, in particular for assessing that multiple features come from the same data source, and the same occurence within the data source such as a single HTTP request or a single network packet.
- Exchange of information between different components of an intrusion-detection system is minimized, enabling remote management on smaller links or minimizing bandwidth required for in-band management.
- Communications with management consoles can be prioritized more easily, sending only what needs to be acted upon immediately and batching the alerts related to reporting and trending activities.

With more intelligent probes in place, work on correlation can concentrate on the most interesting aspects of crossing alerts provided by multiple probes while resting sure that each probe provides a trustworthy diagnostic.

Conclusion

In this paper, we have shown the design of a test bed for comparative evaluation of intrusion-detection systems. This test bed has been used to compare four commercial intrusion-detection systems with each other and with the open-source lightweight *Snort*. The results show that there is room for improvement in the tested probes.

Concerning future work, the test bed is under development to introduce additional applications and services and vary the traffic profile in order to make the task of the tested intrusion-detection systems more difficult. Also, we are developping a prototype probe to validate the EXTRACTOR concept and verify that the diagnostic is indeed improved; this prototype will be used as an additional yardstick along *Snort* in the testbed.

References

[1] Almgren, M., Debar, H., and Dacier, M. A lightweight tool for detecting web server attacks. In *Proceedings of NDSS 2000, Network and Distributed System Security Symposium* (San Diego, CA, February 2000), The Internet Society, pp. 157–170.

[2] Cert Coordination Center. Multiple intrusion detection systems may be circumvented via %u encoding. Cert-CC Vulnerability Note VU#548515, July 2001. http://www.kb.cert.org/vuls/id/548515.

[3] Chung, M., Puketza, N. J., Olsson, R. A., and Mukherjee, B. Simulating concurrent intrusions for testing intrusion detection systems: Parallelizing intrusions. In *Proceedings of the 1995 National Information Systems Security Conference. Baltimore, Maryland, October 10-13, 1995, pp. 173-183.* (Baltimore, MD, October 1995), pp. 173–183.

[4] Curry, D., and Debar, H. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition. Internet Draft (work in progress), December 2001 http://search.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-06.txt.

[5] Debar, H., Dacier, M., and Wespi, A. Reference Audit Information Generation for Intrusion–Detection Systems. internal RZ 2997, IBM Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland, March 1998.

[6] Debar, H., and Wespi, A. Aggregation and correlation of intrusion-detection alerts. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)* (Davis, CA, USA, October 2001), W. Lee, L. Mé, and A. Wespi, Eds., no. 2212 in Lecture Notes in Computer Science, Springer, pp. 85–103.

[7] Denning, D. An intrusion-detection model. *IEEE Trans. Softw. Eng. 13, 2* (1987), 222–232.

[8] Habra, N., Charlier, B. L., Mounji, A., and Mathieu, I. Asax: Software architecture and rule-based language for universal audit trail analysis. In *Proceedings of ESORICS92* (Toulouse, France, November 1992).

[9] Hacker, E. Ids evasion with unicode. http://online.securityfocus.com/infocus/1232, January 2001.

[10] Handley, M., Kreibich, C., and Paxson, V. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th USENIX Security Symposium* (Washington, DC, August 13–17 2001).

[11] Hansen, S. E., and Atkins, E. T. Automated system monitoring and notification with swatch. In *Proceedings of the seventh Systems Administration Conference (LISA '93)* (Monterey, CA, November 1993).

[12] Jagannathan, R., Lunt, T., Anderson, D., Dodd, C., Gilham, F., Jalali, C., Javitz, H., Neumann, P., Tamaru, A., and Valdes, A. System design document: Next-generation intrusion detection expert system (NIDES). Tech. Rep. A007/A008/A009/A011/A012/A014, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025, March 1993.

[13] Kumar, S., and Spafford, E. A pattern matching model for misuse intrusion detection. In *Proceedings of the 17th National Computer Security Conference* (October 1994), pp. 11–21.

[14] Lippman, R., Haines, J. W., Fried, D. J., Korba, J., and Das, K. Analysis and results of the 1999 darpa off-line intrusion detection evaluation. In *Proceedings of the third international workshop on Recent Advances in Intrusion Detection* (Toulouse, France, October 2000), H. Debar, L. Mé, and S. F. Wu, Eds., no. 1907 in Lecture Notes in Computer Science, Springer, pp. 162–182.

[15] Lippmann, R., Fried, D., Graf, I., Haines, J., Kendall, K., McClung, D., Weber, D., Webster, S., Wyschogrod, D., Cunninghan, R., , and Zissman, M. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition* (January 2000), IEEE Computer Society Press.

[16] McHugh, J. The 1998 lincoln laboratory ids evaluation, a critique. In *Proceedings of the third international workshop on Recent Advances in Intrusion Detection* (Toulouse, France, October 2000), H. Debar, L. Mé, and S. F. Wu, Eds., no. 1907 in Lecture Notes in Computer Science, Springer, pp. 145–161.

[17] Mounji, A. *Languages and Tools for Rule-Based Distributed Intrusion Detection.* Doctor of science, Facultés Universitaires Notre-Dame de la Paix, Namur (Belgium), September 1997.

[18] Mueller, P., and Shipley, G. To catch a thief. *Network Computing* (August 2001). http://www.nwc.com/1217/1217f1.html.

[19] Price, K. E. Host-based misuse detection and conventional operating systems' audit data collection. Master of science thesis, Purdue University, Purdue, IN, December 1997.

[20] Ptacek, T. H., and Newsham, T. N. Insertion, evasion, and denial of service: Eluding network intrusion detection. Tech. rep., Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, January 1998.

[21] Puketza, N. J., Chung, M., Olsson, R. A., and Mukherjee, B. A software platform for testing intrusion detection systems. IEEE Software 14, 5 (September–October 1997), 43–51.

[22] Puketza, N. J., Zhang, K., Chung, M., Mukherjee, B., and Olsson, R. A. A methodology for testing intrusion detection systems. *IEEE Trans. Softw. Eng. 22,* 10 (October 1996), 719–729.

[23] Rain Forest Puppy. A look at whisker's anti-ids tactics. http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html, 1999.

[24] Roesch, M. Snort - lightweight intrusion detection for networks. In *Proceedings of LISA'99: 13th Systems Administration Conference* (Seattle, Washington, USA, November 7-12 1999).

[25] SecurityFocus. Multiple ids vendor encoded iis attack detection evasion vulnerability. http://www.securityfocus.com/bid/3292, September 2001.

[26] Spirakis, P., Katsikas, S., Gritzalis, D., Allegre, F., Darzentas, J., Gigante, C., Karagiannis, D., Kess, P., Putkonen, H., and Spyrou, T. SECURENET: A network-oriented intelligent intrusion prevention and detection system. *Network Security Journal 1,* 1 (November 1994).

[27] Staniford-Chen, S., Tung, B., Porras, P., Kahn, C., Schnackenberg, D., Feiertag, R., and Stillman, M. The common intrusion detection framework - data formats. Internet draft draft-ietf-cidf-data-formats-00.txt, March 1998. Work-in-progress.

[28] Wood, M., and Erlinger, M. Intrusion detection message exchange requirements. Internet draft (work in progress), February 2002. http://search.ietf.org/internet-drafts/draft-ietf-idwg-requirements-06.txt.