An Intrusion Detection System Using Ideas from the Immune System

Fabrício Sérgio de Paula
Computing Institute
State University of Campinas
Avenida Albert Einstein, 1251
CEP: 13084-971, Campinas/SP, Brazil
Email: fabricio@las.ic.unicamp.br

Leandro Nunes de Castro
Graduation Program on Informatics
Catholic University of Santos
Rua Dr. Carvalho de Mendonça, 144
CEP: 11070-906, Santos/SP, Brazil
Email: lnunes@unisantos.edu.br

Paulo Lício de Geus
Computing Institute
State University of Campinas
Avenida Albert Einstein, 1251
CEP: 13084-971, Campinas/SP, Brazil
Email: paulo@las.ic.unicamp.br

***Abstract*— This paper proposes an intrusion detection framework and presents a prototype for an intrusion detection system based on it. This framework takes architectural inspiration from the human immune system and brings desirable features to intrusion detection systems, such as automated intrusion recovery, attack signature extraction, and potential to improve behavior-based detection. These features are enabled through intrusion evidence detection. The prototype, called ADENOIDS, is designed to deal with application attacks, extracting signature for remote buffer overflow attacks. The framework and ADENOIDS are described and experimental results are presented.**

I. INTRODUCTION

The Internet was designed to be an open and distributed environment with mutual trust among users. Security issues are rarely given high priority by software developers, vendors, network managers or consumers. As a result, a considerable number of vulnerabilities raises constantly. Once explored by an attacker, these vulnerabilities put government, businesses, and individual users at risk [1], [2].

Intrusion detection systems (IDSs) are useful tools to improve the security of a computer system and, because of their importance, they have become an integral part of modern network security technology. An IDS acts by monitoring events in a computer system or network, analyzing them for signs of security problems [3]. Several techniques are used to achieve intrusion detection such as expert systems, state transition approaches, statistical analysis, and neural networks [3]. More recently, several approaches based on the immune system were proposed [4], [5], [6], [7]. Most of these approaches concentrate on building models and algorithms for behavior-based detection.

This paper presents a framework for intrusion detection and automated response inspired by the human immune system. This framework is intended to mimic, only at the architectural level, several human immune system features, some of them little explored in other works. Examples of these features are intrusion tolerance, intrusion evidence detection, automated attack signature extraction and system recovery mechanisms.

One of the most important aspects of this framework is its assumption that successful attacks are inevitable, and its strongest feature is its ability to deal with such situation. Note that this is also the case with the vertebrate immune system. Some disease-causing agents are successful in invading the organism and causing harm to it before the immune system can eliminate them. After that, the immune system learns to cope with this type of agent, and some repair strategy is taken to recover the damaged parts. In this way, the proposed framework is more related to a research in virus identification [8] than previous work in intrusion detection.

Based on the proposed framework, an IDS prototype, called ADENOIDS, was developed to detect intrusion evidences in running applications, restore the system after an intrusion using a file system undo mechanism, and extract the attack signature for remote buffer overflow attacks.

In fact, applications that provide publicly available services have been the most intended targets of attack in the last years [9]. Among several techniques employed to exploit application vulnerabilities, buffer overflow has been one of the most explored [9].

ADENOIDS was tested against two datasets and the experimental results are encouraging. The proposed signature extraction algorithm can find the attack signature and discard candidate signatures which do not correspond to an attack.

This paper is organized as follows. Section II describes the main concepts of the immune system used as inspiration for the development of the IDS framework, and briefly maps the framework into the immune system. Section III synthesizes the proposed framework, describing its components and the general functioning. Section IV presents the implementation aspects of ADENOIDS and experimental results are shown in Section V. Section VI concludes the paper.

II. INTRODUCTION TO THE IMMUNE SYSTEM

The immune system is a very complex bodily system, and we do not intend to provide a comprehensive view of its functioning in this paper. Instead, the description to be presented here focuses on those aspects that inspired the proposal and development of ADENOIDS. A more thorough description of the immune system can be found in [5], [10], [11].

A. General Concepts

All living beings have the ability to present resistance to disease-causing agents, known as pathogens (e.g., viruses and bacteria). The primary role of the immune system is to protect our bodies against infections caused by pathogens. The defense must occur in many levels and has to cover the whole body. Therefore, various levels of defense mechanisms and barriers have evolved in order to result in sufficient protection. The immune system can be divided into innate immune system and adaptive immune system, composed of diverse sets of cells, molecules and organs that work in concert to protect the organism. Each one of these systems recognize and respond to particular types of pathogens. They have different but complementary functions.

The innate immune system is very important as a first line of defense against several types of pathogens and is also crucial for the regulation of the adaptive immune system. Cells belonging to the innate immune system are capable of recognizing generic molecular patterns (a type of molecular signature) that are only present in pathogens, and can never be found in the cells of the host. Once a pathogen has been recognized by a cell of the innate immune system, this cell signals (through chemical messengers) other immune cells, including those of the adaptive immune system, to start fighting against the pathogen. Therefore, the innate immune system plays a major role in providing co-stimulatory signals for the adaptive immune system. Co-stimulatory signals are usually provided by the innate immune system when the organism is being damaged in some way. For the most types of pathogens, the adaptive immune system cannot act without the co-stimulatory signals provided by the innate immune system.

However, not all pathogens can be recognized by the innate immune system. Some specific pathogens are only recognized by cells and molecules of the adaptive immune system, also called specific immune system. Any pathogenic pattern that can elicit an adaptive immune response is known as an antigen. Once the adaptive immune system is prepared to act, it can adapt to the invading pathogen and create specific molecular patterns to fight against the same or a similar future infection of this type. This is a remarkable feature of the immune system, from the biological and computational perspective: the adaptiveness to previously seen molecular patterns, and the generation and maintenance of stable memories of known patterns.

B. Using Ideas from the Immune System to Introduce the Proposed Framework

The proposed IDS framework specifies ten components (see Figure 1). Some of these components are well-known intrusion detection building blocks—such as Console, Data Source, Knowledge-Based Detector and Behavior-Based Detector[1]—and the others are specified by this framework. The components which bring ideas from the immune system are emphasized here.

The Evidence-Based Detector is part of the innate (computer) immune system. It is capable of providing its own generic response (through the Innate Response Agent) and generating danger signals indicating that damages are being caused due to an attack in progress. These signals will be responsible for co-stimulating the Behavior-Based Detector. Another important feature of this component is that it can stimulate an adaptive immune response (to be performed further by the Adaptive Response Agent) via the Signature Extractor, without requiring the intervention of the Evidence-Based Detector.

The Behavior-Based Detector performs the recognition of some "bad patterns" through a behavioral analysis. This component detects a possible anomalous behavior, but it will wait for a co-stimulatory signal from the Evidence-Based Detector before this profile is definitely identified as an attack. If this co-stimulatory signal is delivered then a maturation process takes place.

The Signature Extractor is responsible for the signature maturation process. It acts by exposing detected bad patterns to normal system events, like in the negative selection of the immune system. This process serves to eliminate some patterns which are false-positives. At the end of this maturation process the attack signature can be outputted.

The Knowledge-Based Detector corresponds to the adaptive immune memory. After the maturation process, only specific bad patterns are kept as memory. Once the Knowledge-Based Detector detects an attack then a specific response to the attack is initiated by the Adaptive Response Agent and the attack is blocked before damage can happen.

The difference between the Adaptive Response Agent and the Response Generator is that the later generates the type of response that will be performed by the Adaptive Response Agent.

Altogether, the Signature Extractor, the Response Generator, the Knowledge-Based Detector, and the Adaptive Response Agent, compose the adaptive immune system. The Evidence-Based Detector, Behavior-Based Detector and Innate Response Agent components compose the innate immune system, acting in a generic way for different attack types. The Console and Forensic Support Repository components were modeled mainly for operational reasons.

III. The Framework

This section introduces the proposed framework which can support several features desirable in a computer immune system. The main goals of this framework are:

1) Precise detection of known attacks and effective response against them.
2) Detection of previously unknown attacks by analyzing evidences of successful attacks to the system.
3) Ability to handle previously unknown attacks by:
    a) Providing countermeasures to maintain the system in acceptable conditions while a more detailed analysis is done.
    b) Learning about the attack in an attempt to extract a signature that matches the attack.

[1]The Knowledge-Based Detector and the Behavior-Based Detector are also known in the IDS community by the terms "Misuse-Based Detector" and "Anomaly-Based Detector", respectively.

    c) Storing attack-relevant information in a repository for future use.
    d) Restoring the affected system parts.
4) Precise detection and response to attacks that the system learnt to recognize.

To achieve these goals, this framework makes use of a set of components—each one with a particular function—and specifies the relationship between them. Figure 1 illustrates these components and the information flow between them. All its components are detailed in Section III-A.

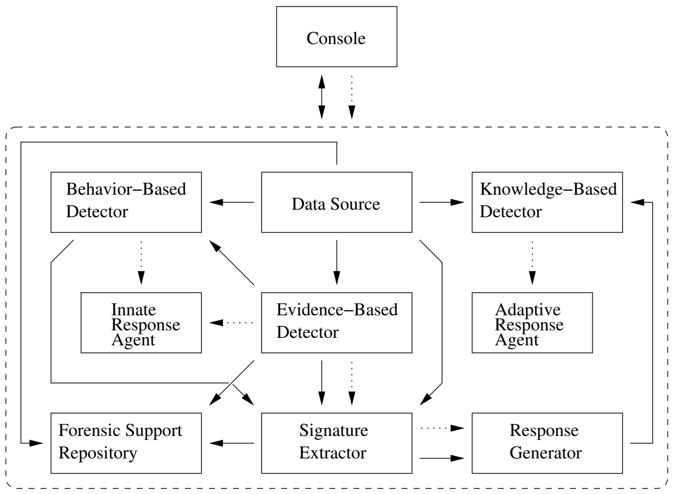

Fig. 1. The proposed framework. The components are represented by solid line rectangles. A grouping of components is represented by a dotted line rectangle. Solid directed lines indicate information flow and dotted directed lines show control flow. Each flow occurs between two components or between one component and all other components of the group.

A. Framework Components

Some framework components are well-known intrusion detection building blocks such as the Console, the Data Source, the Knowledge-based Detector and the Behavior-based Detector. The Console is an interface between the framework and the system administrator. The Data Source is not a component by itself but represents the source of all information needed for the correct IDS working. The remaining components are described as follows.

1) Evidence-Based Detector: Responsible for monitoring the computer system searching for events that indicate a successful attack. Roughly, this component monitors the computer system in search for "damage" that give higher privileges to the attacker. This component must have a precise detection mechanism, matching a security policy, in the sense that both false-positives and false-negatives can happen only at a near-zero rate. An Evidence-Based Detector with the given restriction can be developed because: evidences are invariably produced during an intrusion [12]; and a typical system attack will subvert a privileged process to have an unexpected access to the file system, edit system logs, enable unauthorized communication, start a new service, alter information in an unusual way or perform other related violations [13], [14]. Indeed, during an attack intruders usually modify the file system [15]. In this scenario, monitoring application-level events against a specified policy can be done basically through system call policies, at a low performance cost [16].

The Evidence-Based Detector is also responsible for initiating an innate response and the signature maturation process. Whenever this component detects a security policy violation the Innate Response Agent and Signature Extractor become active. Also, information related to the detected attack can be delivered to the Behavior-Based Detector, Forensic Support Repository and Signature Extractor components.

2) Knowledge-Based Detector: Like in other IDSs, the Knowledge-Based Detector holds a set of attack signatures and works by looking for attempts of these attacks. Once one attack is detected, the Adaptive Response Agent is activated.

3) Adaptive Response Agent: The Adaptive Response agent is responsible for the execution of specific countermeasures related to the attack signature. This component is activated only by the Knowledge-Based Detector and serves to block an attack before any damage can appear.

4) Behavior-Based Detector: This component acts just like in classical behavior-based detection. Whenever the Behavior-Based Detector component detects anomalous behavior exceeding a specified threshold, it activates the Innate Response Agent. The detection of the Behavior-Based Detector is very important for two reasons: 1) it can precede the detection of attack evidences and thus can activate the Innate Response Agent to minimize system damage; and 2) it can detect anomalous behavior and therefore can also be used to point to candidate attack signatures after attack evidences are found.

Taking advantage of the precise evidence-based detection, the Behavior-Based Detector can also improve detection when: 1) the actual behavior appears to be anomalous and evidence-based detection indicates absence of an attack (in this case, the actual behavior can be incorporated into the normal profile); 2) the actual behavior appears to be normal and evidence-based detection indicates an attack in progress (in this case, the actual behavior can be removed from the normal profile). In both cases it is possible to reduce further mistaken detection—false-positives and false-negatives.

5) Innate Response Agent: The Innate Response Agent is responsible for initiating a series of contention measures to slow down a probable attack. Its reaction is limited and general because the attack is not specifically identified yet. The goal of this response is to minimize damage and to save resources so as to allow the system to do a specific attack analysis. Example of these measures are limiting bandwidth or disk access, and changing the priority of a process. If this component is activated by the Evidence-Based Detector, the response can include a file system restoration, restarting daemons and even a machine reboot. The Evidence-Based Detector can also terminate a response in progress.

6) Signature Extractor: The purpose of the Signature Extractor is to analyze the system during or after an attack in an attempt to extract a signature that matches the attack. This signature will enable a more efficient detection of this attack in the future. This component gets information from the Data Source, Behavior-Based Detector and Evidence Based-Detector, and activates the Response Generator as soon as the signature is created.

A general algorithm for the signature extraction problem is proposed in this paper. It takes inspiration from the negative selection process of the human immune system and it is suitable for general attacks. The algorithm divides the signature extraction into two phases: the search for candidate signatures and the maturation of the candidates.

Unlike other works [5], [7], [17] which generate candidate detectors randomly, the proposed algorithm takes advantage from the evidence detection and selects events prior to the attack to be the candidates. Because the attack is a fact—by the evidence detection—some events related to this attack must happen before this evidence detection. The proposed approach seems to be more appropriate to search for good candidates than random generation. In fact, the most appropriate use of the negative selection can be as a filter for invalid detectors, and not for the generation of effective detectors [18].

The proposed algorithm is as follows. The input is composed of a real number $p \in]0;1]$, a set E of events prior to the evidence detection and a set N of events generated by the computer system during normal working, where $N \cap E = \emptyset$. The output is a set $C \subseteq E$ of events, which are the extracted attack signatures with estimated probability less than p of false-positives occurring during further detection. The steps of this algorithm are as follows:

```
1. Restore the computer system to a safe state.
2. Select a set $C$ of events to be the candidate signatures, where $C \subseteq E$.
3. $progress \leftarrow 0$.
4. While $progress < \left\lceil \frac{|C|}{p} \right\rceil$ do:
   4.1. Get a new event $n \in N$ during the normal computer system working.
   4.2. For all $c_i \in C$, if $c_i$ matches $n$, then $C \leftarrow C \setminus \{c_i\}$.
   4.3. $progress \leftarrow progress + 1$.
5. Return each signature in $C$. If $|C| = 0$, return null.
```

Step 2 involves the search for candidates and Step 4 performs the maturation of the candidates. Set C must be chosen to contain events related to the attack being analyzed. These candidate signatures can be obtained from the Behavior-Based Detector or by selecting all events prior to the evidence detection within a time interval.

The set N of normal events can be built in two ways:

1) By collecting events before the attack. In this case, collected events must be stored in such way that they can be retrieved in the future.
2) By collecting events after the attack. In this case, if a new attack evidence is found during or soon after the signature extraction process, the algorithm must be restarted with the initial set C, because this new attack can disregard relevant events of the prior attack.

In both cases, repeated attacks can create an opportunity to correlate their events and extract signatures efficiently.

The matching criterion of Step 4.2 must take into account the attack classes being considered by the IDS developer. The matches command can, for example, represent a perfect matching or a partial matching of one or more event attributes.

It should be noted that the running time of the signature extraction process is dependent mainly upon three factors: $|C|$, p and the generation rate of normal events. Therefore, this process may be long and it is not intended to provide a response in real-time.

7) Response Generator: Receives attack signatures and elaborates a set of specific countermeasures to this attack. It also delivers signatures with the countermeasures to the Knowledge-Based Detector in order to prevent the attack in the future. Example of countermeasures are to restart, kill or stop a process, and to close a connection or block a specific network traffic.

8) Forensic Support Repository: This component is designed to store information gathered during positive attack identification by the Evidence-Based Detector. It is modeled to provide support for manual forensic analysis by preserving data that cannot be corrupted even after a system restore [12].

B. Self-Maintenance

Since the presented framework makes the assumption that successful attacks are inevitable, its components must be protected in such a way that they cannot be damaged by the attack. This protection can be implemented basically at the operating system level, restricting the access from potential attack targets.

IV. The ADenoIdS IDS

The prototype ADenoIdS was developed based upon the proposed framework and it was designed to protect a single computer against application level attacks and to automate signature extraction for remote buffer overflow attacks. The attack evidences are detected in running processes at the system call level and the attack signatures are extracted at the network level.

Table I shows the ADenoIdS modules[2] and their relationship with the framework components. All modules were implemented in C over the Linux kernel version 2.4.19. Some framework components, such as the Response Generator, the Knowledge-Based Detector and the Adaptive Response Agent were not included in this prototype. Once attack signatures are extracted at the network level, the response must only block the patterns known to be ”bad”. Therefore these components can be replaced by a tool like Snort-inline [19].

All information required by ADenoIdS are distributed in three levels: system calls, network traffic and file system information.

The ADCON module is provided through a set of configuration files located in the `/etc/adenoids` directory and a set

[2]The term “module” was adopted to refer to an implemented framework component.

TABLE I

ADENOIDS MODULES AND RELATIONSHIP WITH THE FRAMEWORK.

ADENOIDS **Module**	**Relationship with the Framework**
ADCON	Console
ADEID	Evidence-Based Detector
ADBID	Behavior-Based Detector
ADIRA	Innate Response Agent
ADSIG	Signature Extractor
ADFSR	Forensic Support Repository
UNDOFS	File system restoration for ADIRA

of log files located in the /var/log/adenoids directory. The remaining modules are described as follows.

A. ADEID

The ADEID module monitors running applications in the search for events which violate pre-specified access policies. Each access policy specifies a set of operations which can be performed by a specific process. The events analyzed by ADEID are:

- Files, directories and links: opening, creation, erasing, renaming, truncation and attribute changing (owner, group and permissions).
- Process: creation and execution.
- Kernel modules: creation and deletion.
- Communication: signal sending, TCP connection creation and acceptance, and UDP datagram sending and receiving.

The monitoring policies must be specified obeying the following structure:

```
policy_name[/fully/qualified/program/pathname]
{
    fs_acl{
        list of pathnames and access permissions
    }
    can_exec{
        list of programs which can be executed
    }
    max_children = maximum number of children processes
    can_send_signal = yes | no
    can_manip_modules{
        list of kernel modules which can be
        created and posteriorly deleted
    }
    connect_using_tcp = yes | no
    send_using_udp = yes | no
    accept_conn_on_ports{
        list of port ranges which can be used to
        accept connections
    }
}
```

Although the system call policies proposed in [16] can be more powerful, the ADEID policies make the specification a simpler task. For building a good monitoring policy it is necessary to know about the Linux file system hierarchy and the main purpose of the application intended to be monitored. ADEID has been used to monitor named, wu-ftpd, amd, imapd and httpd applications for two months. It has demonstrated to be very efficient to detect attacks, being free of false-positives and false-negatives during the tests.

ADEID is implemented as a kernel patch by rewriting some system calls which deal with the monitored events. The new system calls do their original work and call the detection procedure. By detecting attack evidences, ADEID analyzes only successful system calls. This feature—which characterizes the evidence detection, unlike [16]—also helps to reduce the false-positive rate because unauthorized actions will not be analyzed.

The policies are read from disk and loaded into memory during the system startup. A new system call was also created for test reasons to update the policies after the system initialization. Whenever a new process is executed through the system call sys_execve(), ADEID searches for a related access policy. If there is one defined for this process it is attached to the process. After this moment, the process has all relevant events monitored by ADEID. Once a process becomes monitored, all new children processes will also be monitored. In this case, if a child process does not have a specific policy it will be monitored according to the parent process policy.

Whenever ADEID detects some attack evidence the ADIRA module becomes active by calling a kernel procedure and, after that, a SIGUSR1 is sent to ADSIG and information about the attack—current process and violated policy—are also delivered. For testing purposes an user can disable these activation mechanisms.

Preliminary results show that the performance cost imposed by ADEID is imperceptible for users. A general benchmark for the most expensive operation—opening and reading cached files—showed that this cost is, on average, lower than 5% in an Athlon XP 1900+ with 512MB RAM. Table II summarizes the average performance penalty imposed by the ADEID module. This penalty was calculated by comparing the execution time of applications running over an unpatched kernel and the execution time of these applications being monitored over the ADEID patched kernel. Each test was repeated ten times and was used a total of 800 files equally distributed in eight categories of size: 100, 1000, 5000, 10000, 50000, 100000, 500000 and 1000000 bytes.

TABLE II

AVERAGE PERFORMANCE PENALTY IMPOSED BY ADEID.

Operation	**Average Penalty (%)**
System startup and halt	0.52
Open and read cached files	4.32
Create and write to files	0.09
Get cached files via httpd (Apache/2.0.40)	1.78

B. ADBID

The ADBID module is responsible for analyzing network traffic in search for incoming information which appear to be anomalous.

ADBID captures packets through pcap library and delivers them to application-specific procedures. An application-specific procedure decodes the related application-level protocol delivering the request[3] to be analyzed.

[3]The term “request” is was adopted to refer to application-level protocol data.

The role of ADBID is to point to the candidate attack signatures. Because ADENOIDS focuses on signature extraction for remote buffer overflow attacks, ADBID works by building a statistical profile to detect requests whose length are less probable to be found during normal operation and are found in overflow attacks. Actually, ADBID detects requests whose length is greater than $\mu + 2s$, where μ is the arithmetic mean of requests length and s is the standard deviation of requests length.

ADBID writes all decoded requests to a file and all abnormal decoded requests to a separate file.

The extensible ADBID implementation makes it possible to add a new analysis procedure by creating this new procedure and required data structures, and setting a function pointer at the ADBID initialization. A new application can have your traffic analyzed by adding some information about the application in the code such as packet filter expression, application-level protocol decoder procedure, ADEID monitoring policy and application name.

C. ADIRA

The ADIRA module is implemented as a kernel patch and works by restoring the computer system after an attack. Actually ADIRA does not implement the contention measures proposed by the Innate Response Agent and, consequently, it is only activated by ADEID.

ADIRA restores the computer system through the following steps:

1) Block all user processes.
2) Restore the file system through UNDOFS module.
3) Restart the monitored applications.
4) Kill the attacked process.
5) Unblock all blocked processes.

D. UNDOFS and ADFSR

The UNDOFS module implements a general mechanism to provide file system restoration by applying *undo* techniques. It is also developed as a kernel patch and provides undo and redo features to any file system which can support both, reading and writing data.

This mechanism is activated before the following operations over files, directories and links can be done: creation, erasing, renaming, writing, truncation and attribute changing. For each operation a specific undo log is created. This log holds the necessary information in such way that the operation can be reversed in the future. Redo logs are created by operations performed during the undo process.

A kernel procedure can be called to request a file system undo or redo up to a defined checkpoint. Actually the checkpoints are inserted automatically during the system startup.

A configuration file states what directories are covered by this mechanism. The default UNDOFS configuration file includes the directories `/bin`, `/boot`, `/dev`, `/etc`, `/initrd`, `/lib`, `/sbin`, `/usr` and `/var/named`. These directories contain the most important binaries and configuration files needed for the correct system working.

The UNDOFS performance depends on the operation to be done. Appending bytes to a file adds only a fixed-size log. File truncation requires to read the bytes to be truncated and to write them into the log file. File erasing and the overwrite operation are also expensive.

It should be noted that the default UNDOFS configuration file includes vital directories which are rarely modified and, therefore, the imposed cost is very acceptable.

The ADFSR module implements an interface to provide step-by-step redo by calling UNDOFS procedures after a system reboot. In this way, the manual analysis of all file system events happening during an attack is enabled; if a system administrator or forensics specialist want to do that.

E. ADSIG

The ADSIG module is responsible for extracting attack signatures from network traffic in such a way that this attack can be efficiently identified and blocked in the future.

This module works exactly as proposed in Section III-A.6. Once activated by ADEID, ADSIG reads the delivered information about the violated policy—policy name and related process—and inserts this information in a queue.

Whenever the current signature extraction terminates, ADSIG gets the information about the next one from the queue. If there is one into the queue, ADSIG begins a new signature extraction by opening the related ADBID abnormal requests file and loading the candidate signatures into a proper data structure. Actually ADSIG loads the candidates which were captured within the last 24 hours. The next step consists of opening the ADBID decoded requests file and to begin the signature maturation process.

In the signature maturation process ADSIG considers only requests which arrived after the attack. A matching criterion was chosen to discard the candidates which are most probable to be found during normal operation. In the actual ADSIG implementation all candidates whose length is lower than or equal to a normal request length are discarded. This works well for buffer overflow attacks because if a request is normal—and probably does not overflow a buffer size—a candidate whose length is at most equal to the request length probably will not overflow this buffer size too and can also be considered normal[4]. At the end of the signature maturation process ADSIG outputs the extracted signatures.

The current implementation does not take advantage of subsequent attacks and the current signature extraction process is restarted when new evidences are found.

The default value for the parameter p is 0.0001, what means that the extracted signatures altogether will probably generate less than one false-positive in ten thousand requests.

F. ADENOIDS *Self-Maintenance*

ADENOIDS implements a simple self-maintenance mechanism, which consists of denying access to ADENOIDS modules, data and configuration files from the processes being

[4]A more complete analysis should consider a different buffer size for each request type.

monitored. In this way ADENOIDS considers that all possible attack target—usually server applications—must be monitored.

V. Experimental Results

This section presents experimental results obtained by testing the ADENOIDS IDS. The main objectives of the tests were to evaluate the ability of evidence-based detection, behavior-based detection and signature extraction mechanisms.

The tests were done on an Athlon XP 1900+ with 512MB DDR RAM, 80GB Ultra-DMA IDE 7200 RPM hard disk running Linux kernel 2.4.19. This host system was used to attack a User-mode Linux virtual machine running ADENOIDS over a guest kernel 2.4.19. This guest system was customized from a Red Hat Linux 6.2 to provide vulnerable named, wu-ftpd, amd and imapd applications. All these applications can be successfully attacked through buffer overflow exploits collected around the world.

The ADEID, UNDOFS and ADFSR modules have been used for two months whereas the ADBID, ADIRA and ADSIG modules were tested for two weeks.

Each ADEID monitoring policy was built in two steps by observing the reported violations:

1) Initial policy establishment. This step spent about half an hour of intensive work.
2) Policy refinement. This step spent about two days of sparse work.

After these steps, the ADBID demonstrated to be very efficient to detect attacks, being free from false-positives and false-negatives during the tests.

The complete ADENOIDS IDS was tested against the 1999 Darpa Offline Intrusion Detection Evaluation dataset—available at http://www.ll.mit.edu/IST/ideval/index.html—and against a dataset collected at our research laboratory (LAS dataset).

The 1999 Darpa Offline IDS Evaluation dataset is composed of training and test datasets which include network traffic data, event logs and other audited data. Several attack types are present in this evaluation, including buffer overflow attacks. ADENOIDS was tested only against named buffer overflow attacks because this dataset does not provide training data for the imapd overflow and the vulnerable sendmail daemon was not available. To compensate this, some buffer overflow attacks in the test data for the wu-ftpd daemon were inserted. Because ADENOIDS analyzes only events produced by one host the test was done considering the network traffic destined to hosts separately.

The LAS dataset was collected under normal conditions at our external DNS server during 43 days. This dataset was chosen by two factors: 1) named is a very important application and often vulnerable; and 2) DNS queries can be replayed easily. This dataset was first analyzed before the test phase and was verified to be free of attacks.

Table III summarizes the results for the 1999 Darpa Offline IDS Evaluation and the LAS datasets. An appropriate label is placed before the beginning of each dataset results. The first column describes the target daemon being considered and the second column shows the average number of requests per day to the considered target host in the whole dataset. The third column presents the number of requests spent in the ADBID training. The fourth column shows the number of requests prior to the attack which were captured in the last 24 hours. Each test was performed by considering an exclusive set of prior events. For the LAS dataset, it was used a fixed number of 10,000 requests prior to the attack, exceeding the average number of requests per day. The fifth column shows the number of ADBID candidate signatures extracted from each of these sets. The sixth column presents the number of requests required by the complete signature extraction process. In some tests the final ADSIG output can be known by using only 1,000 normal events in the maturation process, but to satisfy the p parameter (indicated between parenthesis) the process must be continued. The seventh column indicates the number of requests outputted by ADSIG at the end of the maturation process. Some attacks can present more than one attack signature and the eighth column indicates if the main overflow request is found by ADSIG. The last column shows the number of false-positives after the signature extraction process.

The ADBID module was very efficient to found the candidate signatures. Its detection was capable of selecting fewer candidates and the main buffer overflow request was always inside the candidates' set.

The ADSIG module has also demonstrated to be very appropriate to discard erroneous candidates. The overflow requests were the only candidates at the end of the signature extraction process in seventeen out of twenty one attacks analyzed. The wu-ftpd false-positives were probably produced due to a fewer number of requests in the dataset and, consequently, in the maturation process. The first named false-positive was not also an overflow, but it looks like a malformed host name query.

Although some false-positives can happen, the signature generation algorithm claims that extracted signatures which are valid requests will be probabilistically rare events in further detection.

VI. Conclusions and Future Works

This paper proposes a new security framework which takes inspiration from the human immune system. The analogies are mainly kept at the architectural level and several well-known intrusion detection techniques are integrated. In addition, new components are specified. The result is a security framework that is very appropriate to computer security problems.

An IDS, called ADENOIDS, based upon the proposed framework was developed. This IDS is originally intended to deal with application attacks, extracting attack signatures for remote buffer overflow attacks. ADENOIDS is composed of seven modules, as described in the paper.

This IDS was tested against the Darpa 1999 Offline IDS Evaluation dataset and against another collected dataset. The experimental results presented were very encouraging. The proposed signature extraction algorithm can find the attack

TABLE III

EXPERIMENTAL RESULTS FOR THE 1999 DARPA OFFLINE IDS EVALUATION DATASET AND FOR THE LAS DATASET.

Target Daemon	Average # of Requests/Day	# Requests ADBID Training	# Requests Last 24 Hours	# Candidates Last 24 Hours	Required # of Normal Events	# Extracted Requests	Signature Found?	# False-Positives
Experimental results for the 1999 Darpa Offline IDS Evalutaion dataset								
named	174559	50000	58942	6	10000 ($p = 0.0001$)	1	yes	0
named	174559	50000	267336	11	10000 ($p = 0.0001$)	1	yes	0
named	174559	50000	266995	8	10000 ($p = 0.0001$)	1	yes	0
wu-ftpd	1575	2000	1873	29	4342 ($p = 0.003$)	13	yes	1
wu-ftpd	1575	2000	1603	36	4008 ($p = 0.003$)	12	yes	0
wu-ftpd	827	2000	918	22	3340 ($p = 0.003$)	10	yes	0
wu-ftpd	827	2000	795	22	3674 ($p = 0.003$)	11	yes	1
wu-ftpd	761	2000	670	24	4008 ($p = 0.003$)	12	yes	0
wu-ftpd	761	2000	1097	38	4008 ($p = 0.003$)	12	yes	0
Experimental results for the LAS dataset								
named	8590	40922	10000	25	10000 ($p = 0.0001$)	1	yes	0
named	8590	40922	10000	7	10000 ($p = 0.0001$)	1	yes	0
named	8590	40922	10000	14	21099 ($p = 0.0001$)	2	yes	1
named	8590	40922	10000	20	10000 ($p = 0.0001$)	1	yes	0
named	8590	40922	10000	18	10000 ($p = 0.0001$)	1	yes	0
named	8590	40922	10000	15	10000 ($p = 0.0001$)	1	yes	0
named	8590	40922	10000	10	10000 ($p = 0.0001$)	1	yes	0
named	8590	40922	10000	18	10000 ($p = 0.0001$)	1	yes	0
named	8590	40922	10000	20	20000 ($p = 0.0001$)	2	yes	1
named	8590	40922	10000	25	10000 ($p = 0.0001$)	1	yes	0
named	8590	40922	10000	20	11640 ($p = 0.0001$)	1	yes	0
named	8590	40922	10000	23	30955 ($p = 0.0001$)	1	yes	0

signatures and discard candidate signatures that would only produce false-positives.

Future work includes new tests considering other vulnerable applications, addition of new features in the IDS, a more detailed analysis of the experimental results reported here, and a study about ADENOIDS generalization capability. It is also the authors' intent to study the Danger Theory of the immune system [20], [21] and check if it can add to the framework.

Although the ADENOIDS signature extraction mechanism covers only buffer overflow attacks, the proposed framework is extensible to other classes of attacks. The ideas described here can also have straight applications in other areas, such as honeypot automation and forensic analysis.

ACKNOWLEDGMENT

The authors would like to thank the FAPESP agency for supporting this research.

REFERENCES

[1] S. Garfinkel and G. Spafford, "Practical UNIX & Internet Security", 2nd ed., O'Reilly and Associates, 1996.

[2] R. Pethia, "Computer Security", *Cert Coordidantion Center*, Available on the web at http://www.cert.org/advisories/CA-2002-27.html, 2000.

[3] R. Bace, "Intrusion Detection", 1st ed., Macmillan Technical Publishing, 2000.

[4] S. Forrest, A. Perelson, L. Allen and R. Cherukuri, "Self-nonself discrimination in a computer", in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 202-212, 1994.

[5] S. Hofmeyr and S. Forrest, "Architecture for an Artificial Immune System", in *Evolutionary Computation*, vol. 7, pp. 45-68, 2000.

[6] D. Dasgupta, "Immunity-Based Intrusion Detection System: A General Framework", in *Proceedings of the 22nd National Information System Security Conference*, pp. 147-160, 1999.

[7] J. Kim and P. Bentley, "An Artificial Immune Model for Network Intrusion Detection", in *Proceedings of the 7th European Congress on Intelligent Techniques and Soft Computing*, 1999.

[8] J. Kephart, "A biologically inspired immune system for computers", *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pp. 130-139, MIT Press, 1994.

[9] CERT/CC, "CERT Summaries 1995-2003", *Cert Coordination Center*, Available on the web at http://www.cert.org/summaries, 2004.

[10] L. N. de Castro and J. Timmis, "Artificial Immune Systems: A New Computational Intelligence Approach", 1st ed., Springer-Verlag, 2002.

[11] C. Janeway, P. Travers, M. Walport and J. Capra, "Immunobiology: The Immune System in Health & Disease". 4th ed., Garland Publishing, 1999.

[12] P. Stephenson, "Investigating Computer-Related Crime", 1st ed., CRC Press, 1999.

[13] CERT/CC, "CERT/CC Overview: Incident and Vulnerability Trends", *CERT Coordination Center*, Available on the web at http://www.cert.org/present/cert-overview-trends, 2004.

[14] W. Kruse II, and J. Heiser, "Computer Forensics: Incident Response Essentials", 1st ed., Addison Wesley, 2002.

[15] G. Kim and E. Spafford, "The Design and Implementation of Tripwire: A File System Integrity Checker", in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, 1994.

[16] N. Provos, "Improving Host Security with System Call Policies", in *Proceedings of the 12th USENIX Security Symposium*, 2003.

[17] J. Kim and P. Bentley, "Negative Selection and Niching by an Artificial Immune System for Network Intrusion Detection", in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 149-158, 1999.

[18] J. Kim and P. Bentley, "Evaluating Negative Selection in an Artificial Immune System for Network Intrusion Detection", in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1330-1337, 2001.

[19] J. Haile and R. McMillen, "Snort-inline tool", Available on the web at http://project.honeynet.org/papers/honeynet/tools, 2004.

[20] P. Matzinger, "Tolerance, Danger and the Extended Family", in *Annual Review of Immunology*, vol. 12, pp. 991-1045, 1994.

[21] U. Aickelin, P. Bentley, S. Cayzer, J. Kim and J. McLeod, "Danger Theory: The Link between AIS and IDS?", in *Proceedings of the 2nd International Conference on Artificial Immune Systems*, pp. 147-155, 2003.