Attack Evidence Detection, Recovery and Signature Extraction with ADenoIdS

Fabrício Sérgio de Paula and Paulo Lício de Geus*

Computing Institute, State University of Campinas, Campinas, SP, Brazil
{Fabricio, Paulo}@las.ic.unicamp.br

Abstract. This paper presents the ADenoIdS intrusion detection system (IDS). ADenoIdS takes some architectural inspiration from the human immune system and automates intrusion recovery and attack signature extraction. These features are enabled through attack evidence detection. This IDS is initially designed to deal with application attacks, extracting signature for remote buffer overflow attacks. ADenoIdS is described in this paper and experimental results are also presented. These results show that ADenoIdS can discard false-positives and extract signatures which match the attacks.

1 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 [4–6]. Most of these approaches concentrate on building models and algorithms for behavior-based detection.

This paper presents the ADenoIdS IDS which is intended to mimic, mainly at the architectural level, several human immune system features, some of them little explored in other works. Examples of these features are intrusion tolerance, attack evidence detection, automated attack signature extraction and system recovery mechanisms.

One of the most important aspects of this IDS is its assumption that successful attacks are inevitable, and its strongest feature is its ability to deal with such

* The authors would like to thank the FAPESP agency and the State University of Mato Grosso do Sul (UEMS) for supporting this research.

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, this IDS is more related to a research in virus identification [7] than previous work in intrusion detection.

ADenoIdS was developed to detect attack evidences in running applications, restore the system after an attack 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 [8]. Among several techniques employed to exploit application vulnerabilities, buffer overflow has been one of the most explored [8].

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.

In this paper will not be discussed the immune system features and its analogies with security systems, and the reader is referred to [9] for an introduction to these issues.

This paper is organized as follows. Sect. 2 presents an overview of the ADenoIdS IDS. Sect. 3 describes the main implementation aspects of this IDS and experimental results are shown in Sect. 4. Sect. 5 concludes the paper.

2 ADenoIdS Overview

ADenoIdS was designed to monitor a single computer in such way to detect application-level attacks and automate signature extraction for remote buffer overflow attacks. The attack evidences are detected in running process at the system call level and the attack signatures are extracted at the network level. Fig. 1 illustrates the ADenoIdS modules and the communication flow between them. In this figure, a short name for some modules is indicated inside parenthesis.

Some of these modules are well-known intrusion detection building blocks such as the Console, the Data Source and the Behavior-Based Detector. The role of each module is as follows:

- Data Source: is not a module by itself but represents the source of all information needed for the correct IDS working.
- ADCON: is an interface between the IDS and the system administrator.
- ADEID: is responsible for monitoring the computer in search for events that indicate a successful attack. Although all modules are designed to work concurrently, ADEID can initiate an automated response (by activating ADIRA) and the signature extraction process (by activating ADSIG).
- ADIRA: is responsible for restoring the computer system after an attack.
- ADBID: performs anomaly detection by monitoring incoming network traffic in search for candidate attack signatures.

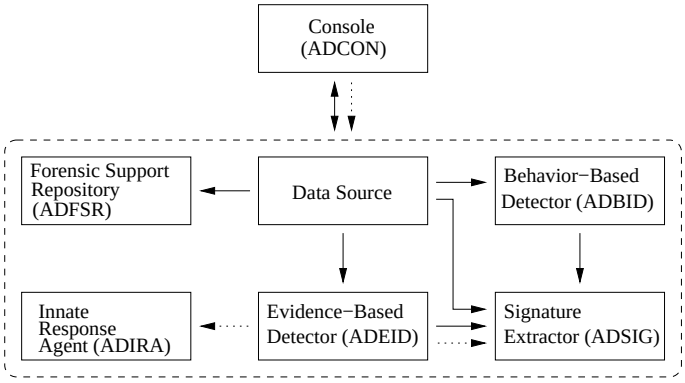

Fig. 1. The ADenoIdS modules. Each module is represented by solid line rectangles. Solid directed lines indicate information flow and dotted directed lines show control flow. Each flow occurs between two modules or between one module and all other modules of the grouping (represented by a dotted line rectangle)

- ADSIG: analyzes the collected candidate signatures in attempt to: 1) discard false-positives; and 2) extract signatures which match the attack.
- ADFSR: it is only modeled to provide support for manual forensic analysis by preserving data that cannot be corrupted even after a system restore.

The purpose of the signature extraction process is to enable a more efficient detection of this attack in the future by using a signature-based IDS like Snort-inline [10]. This process involves the ADBID and ADSIG modules and a general algorithm for the signature extraction problem is proposed in Sect. 2.1.

2.1 The Signature Extraction Algorithm

This section proposes a signature extraction algorithm which 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 [4, 6] which generate candidate detectors randomly, the proposed algorithm takes advantage from the evidence detection of ADEID and selects anomalous events prior the attack to be the candidates.

The proposed approach seems to be more appropriate for searching good candidates than randomly 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 [11].

The proposed algorithm is as follows. The input is composed of a real number $p \in]0; 1]$, a set E of events prior 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. The system restoration (Step 1) is provided by ADenoIdS through the ADIRA module. The set E comprehends the incoming network traffic prior to ADEID detection and the initial set C is built by collecting the anomalous traffic detected by ADBID from E. Because ADenoIdS focuses overflow attacks, ADBID works by detecting large requests[1] in the network traffic. The set N is built by collecting related incoming network traffic after the system restoration. The matching criterion of Step 4.2 takes into account the size of requests into the network traffic. If a new attack evidence is found during or soon after the signature extraction process, the algorithm is restarted with the initial set C, because this new attack can discard relevant events of the prior attack.

By considering the matching operation to be dominant and m to be the size of the initial set C, the running time of this algorithm is, in the worst case, $O\left(\frac{m^2}{p}\right)$. However, it should be noted that the real execution time is also dependent upon the generation rate of normal events. Therefore, this process may be long and it is not intended to provide a response in real-time.

3 ADenoIdS Implementation

ADenoIdS was implemented in C over the Linux kernel version 2.4.19. All information required by this IDS are distributed in two levels: system calls and network traffic.

The ADCON module is provided through a set of configuration files and a set of log files. The remaining modules are described as follows.

3.1 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 an specific process. The events analyzed by ADEID are the following:

[1] The term "request" is was adopted to refer to both application-level command and response.

– Files, directories and links: open, 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 child process
    can_send_signal = yes | no
    can_manip_modules { list of kernel modules which can be created and 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 [12] 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 from 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 policies are read from disk and loaded into kernel memory. Whenever a new process is executed the related monitoring policy is attached to the process, if any is defined.

By detecting attack evidences, ADEID analyzes only successful system calls. This feature—which characterizes the evidence detection, unlike [12]—also helps to reduce the false-positive rate because unauthorized actions will not be analyzed.

Whenever ADEID detects some attack evidence the ADIRA module becomes active by calling a kernel procedure and, after, 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.

3.2 ADBID

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 to be analyzed.

Because ADenoIdS focuses on signature extraction for remote buffer overflow attacks, ADBID works by building an 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. The detection procedure and parameters can also be easily changed.

3.3 ADIRA and ADFSR

The ADIRA module is implemented as a kernel patch and works by restoring the computer system after an attack. This restoration is done through the following steps: 1) block all user processes; 2) restore the file system; 3) restart the monitored applications; 4) kill attacked process; and 5) unblock all blocked processes.

The file system restoration is implemented by applying *undo* techniques into 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 is created an specific undo log. This log holds the necessary information in such way that the operation can be reversed in the future. Redo logs are created by operations done 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 undo 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 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 redo procedures after a system reboot. In this way, it is enabled the manual analysis of all file system events done during an attack, if a system administrator or forensics specialist want to do that.

3.4 ADSIG

This module works exactly as proposed in Sect. 2.1. Once activated by ADEID, ADSIG reads the delivered information about the violated policy (policy name and related process) and begins a new signature extraction by loading the candidate signatures into a proper data structure. By default, the set E is built by selecting requests for the last 24 hours prior to ADEID detection. Therefore the initial set C contains the candidates for the last 24 hours.

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 the request length probably will not overflow this buffer size too and can also be considered normal[2]. At the end of the signature maturation process ADSIG outputs the extracted signatures.

3.5 ADenoIdS Self-protection

It is implemented a simple self-protection mechanism, which consists of denying access to ADenoIdS modules, data and configuration files from the processes being monitored. In this way ADenoIdS considers that all possible attack target—usually server applications—must be monitored.

4 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 test system were customized from a Red Hat Linux 6.2 to provide vulnerable named, wu-ftpd, amd and imapd applications over the Linux kernel 2.4.19. All these applications can be successfully attacked through buffer overflow exploits collected around the world and these attacks are launched from an external machine.

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 ADEID 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

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

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 1 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 ($|E|$). Each test was performed by considering an exclusive set of prior events. For the LAS dataset, it was used a fixed number of 10000 requests prior to the attack, exceeding the average number of requests per day. The fifth column shows the number of ADBID candidate signatures extracted (the initial $|C|$) 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 1000 normal events in the maturation process, but to satisfy the p parameter (indicated inside 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.

Table 1. Experimental results for the Darpa and LAS datasets

Target Daemon	**Average # of Reqs/Day**	**# Reqs Training**	**# Reqs (24 hours)**	**# Candidates (24 hours)**	**Required # of Normal Events**	**# Outputted 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

5 Conclusions and Future Works

This paper presents the ADenoIdS IDS which takes inspiration from the human immune system. This IDS is originally intended to deal with application attacks, extracting attack signatures for remote buffer overflow attacks.

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

Future work includes new tests considering other vulnerable applications, correlation of subsequent attacks, and an study about ADenoIdS generalization capability.

Although the ADenoIdS signature extraction mechanism covers only buffer overflow attacks it 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.

References

1. Garfinkel S., Spafford G.: Practical UNIX & Internet Security. 2nd edn. O'Reilly and Associates (1996)
2. Pethia, R.: Computer Security. Cert Coordidantion Center. Available on the web at http://www.cert.org/congressional_testimony/Pethia_testimony_Mar9.html (2000)
3. Bace, R.: Intrusion Detection. 1st edn. Macmillan Technical Publishing (2000)
4. Hofmeyr S., Forrest, S.: Architecture for an Artificial Immune System. Evolutionary Computation, Vol. 8 (2000) 443-473
5. Dasgupta, D.: Immunity-Based Intrusion Detection System: A General Framework. Proceedings of the 22nd National Information System Security Conference (1999) 147-160
6. Kim, J., Bentley, P.: An Artificial Immune Model for Network Intrusion Detection. Proceedings of the 7th European Congress on Intelligent Techniques and Soft Computing (1999)
7. Kephart, J.: A Biologically Inspired Immune System for Computers. Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems (1994) 130-139
8. CERT Coordination Center: CERT Summaries 1995-2003. Available on the web at http://www.cert.org/summaries (2004)
9. de Castro, L.N., Timmis, J.: Artificial Immune Systems: A New Computational Intelligence Approach. 1st edn. Springer-Verlag (2002)
10. Haile, J., McMillen, R.: Snort-inline tool. Available on the web at http://project.honeynet.org/papers/honeynet/tools (2004)
11. Kim, J., Bentley, P.: Evaluating Negative Selection in an Artificial Immune System for Network Intrusion Detection. Proceedings of the Genetic and Evolutionary Computation Conference (2001) 1330-1337
12. Provos, N.: Improving Host Security with System Call Policies. Proceedings of the 12th USENIX Security Symposium (2003)