

# ImmunesystemforthedetectionofintrusionsatHT TPprotocollevel

EfraínTorresMejía  
e-mail: [e.torres@javeriana.edu.co](mailto:e.torres@javeriana.edu.co)  
DepartmentofSystemsEngineering  
PontificiaJaverianaUniversity  
Colombia,May2003

## Summary

Thispapercoversthedevelopmentofasystemforthedetectionandpreventionof intrusionsatanHTTPprotocollevel,basedonpreviousresearchworkcarriedouton computerimmunesystemsandnewtechnologiesdevelopedforHIDSandNIDSwiththe aimofreducingthelimitationsofcurrentIDS,increasingtheirprecisionand reconsidering certainapproachesrelatingtothearchitecture,design,implementationand configurationof thistypeofsystem.

## Keywords:

IDS,IPS,HTTP,Elman,artificialneuralnetworks,computerimmunesystems,firewalls.

## 1.Introduction

Intrusiondetectionsystemsaregenerallydividedintotwotypes:intrusiondetection systemsoperatingatanetworklevel(NIDS)andhostlevel(HIDS).There isanewthird typewhichhasbeentermedanintrusiondetectionsystematapplicationlevelor “ApplicationFirewall”.Unfortunately,thesesystemscontinuetobasetheiroperationon therecognitionofsignaturesofpreviouslyknownattacks,whicharecompared,inthecase ofNIDS,withthenetworktrafficcapturedbyanestablishedsensor,orinthecaseof applicationfirewalls,withthenetworktraffiddirectedtowardsaparticularserver,generally aWebserver.

### 1.1LimitationsofcurrentIDS

Unfortunately,currentIDSbasetheirperformanceonastaticschema,whichmakesthem highlyinefficientsystemswhenanattackermodifiestheinputdata.Thedatabaseofattack signaturesuffersthesameinconveniencesasanti-virusystems:ifthedatabaseisnot updateditishighlylikelythatthesystemwillbeattackedwithoutitbeingdetected. Additionally,thetimethatlapsesbetweenthe discoveryofanewattack,itstopublication anditsinclusionintheIDSisquitelong,consideringthatthetimethatlapsesbetween the publicationofsaidattackandtheimproperuseonthe partofapossibleattackerisvery short.

Applicationfirewallsgenerallyworkbycoveringtheapplicationthatistobeprotected, whichmakesthemdependantontheapplicationandthesystemorarchitectureonwhich theapplicationtobeprotectedresides.Thisplacesagreatlimitationontheir implementationandcangiverisetonewsecurityproblemsasitincreasesthelevelof

complexity of the application to be protected, and thus increases the possibility that new security problems will arise.

## 1.2 Behaviour analysis

Having observed the limitations of current IDS, certain research centres around the world have begun to analyse behaviour in order to determine the existence of anomalies which can be used to identify attacks. Initial behaviour analysis focused on the development of statistical mechanisms to determine anomalies in the behaviour of users of a particular system. The great advantage of statistical mechanisms is that they are easily adaptable to new and changing conditions over time. But that adaptability is susceptible to progressive programmed changes which enable the inclusion of intrusive activities, thus avoiding their detection [1].

There are two types of behaviour analysis: behaviour analysis for anomaly detection and behaviour analysis for misuse detection. Anomaly detection can be defined as the attempt to detect intrusions by discovering significant deviations in normal behaviour, whilst misuse detection corresponds to the current approach which uses signatures of attacks previously introduced into the detection system, and compares them with activities in the system in order to detect an attack. In the last five years, studies such as "Learning Program Behavior Profiles for Intrusion Detection" [1], have demonstrated the necessity and effectiveness of analysing abnormal program behaviour instead of the approach currently used.

## 1.3 Computer immune systems

The immune system of vertebrates is an excellent example of an intrusion detection system which eliminates many of the problems stated above. It is a multi-level system, which is robust, dynamic, diverse and adaptable. This makes it an excellent basis for the protection of computer systems [2][3][7]. Additionally, it should be remembered that the immune system is a system of protection by levels, where the pathogenic element principally faces the skin as the main initial barrier and later confronts the immune system itself, whose functional mechanism is based on the recognition of patterns by the lymphocytes. This recognition does not depend on a permanent and static memory, but rather on an evolutionary memory, which is inherited and therefore dynamic.

There exist studies in which these immunological concepts have been used to create an intrusion detection model, where patterns of attack in system calls by privileged processes are recognised in a dynamic manner [4][5][6][7]. These studies demonstrated the feasibility of using the recognition of previously learned patterns in system calls to detect pathogenic agents which represent intrusions to the system.

The problem is that in managing information at this level, we lose highly valuable and specific information which is located at higher levels of abstraction. The optimum level for analysing this information corresponds to that level in  $(t) - 2(\text{udi}) - 1.99788(\text{e})4.00187[\text{TJ3ptntginginff4}(\text{na})4(1) - 22$

that, since it has not been detected as an anomalous (intrusive) behaviour, the process will already have infected the system. The same occurs with the systems currently developed in behaviour analysis research projects and in existing application firewalls, but the latter generally manage a special type of barrier which defines valid characteristics for a particular request, and if it manages to pass this filter, located at the application level, then the identification process continues. From the immunological perspective, this initial barrier serves as a “skin” for the system. If the request manages to get past this barrier, it will come up against the immune system which, through an algorithm for detecting previously learned patterns, will confirm whether this request corresponds to normal behaviour and as a result, can be processed. If this is not the case, then the request is rejected and may optionally pass to an additional stage of classification.

## 1.4 Web and HTTP protocol

A typical organisation with an Internet presence, “protected” by a firewall, allows HTTP or HTTPS traffic to its Web servers. The World Wide Web (WWW) has become a public service *par excellence*. In December 2002, the Spanish IT security firm S21 SEC carried out a study of the preceding five months (June–November). In this study, it was established that of 2,113 vulnerabilities published, 1,320 had their origin in Web applications [8]. This represents 62.5% of reported vulnerabilities, a percentage that reflects not only the high number of security problems in the applications supporting or managing this type of service, but also the high degree of risk to which organisations with an Internet-based Web presence are exposed. A risk which has not really been evaluated, considering that the majority of organisations are not aware of the limitations of protection mechanisms such as firewalls. The outlook becomes much more critical if we take into account the growing complexity of Web servers, which due to the great quantity of vulnerabilities involved in them and the applications they support, have become unauthorised points of access to any organisation. For this reason, the system developed is aimed directly at the HTTP protocol.

Currently, the HTTP protocol is in its 1.1 version. HTTP/1.0 specifies that a single TCP connection provides a single HTTP message. HTTP/1.1 enables multiple message sockets over a single TCP connection [9]. This difference does not affect intrusion detection at the application level, and therefore the latest version is used as a reference for this protocol.

## 2. IDS for HTTP protocol

### 2.1 Architecture

The great limitation of IDS at network level lies in the level of data acquisition and the technology used to obtain this data at this level. Its limitations include the inability to capture and analyse all the traffic that should be monitored. This shortcoming arises from the actual traffic analysers (sniffers), which are the system sensors. The system has to manage network problems such as data fragmentation, which requires the use of processing power in the manipulation of data, and therefore reduces the system's performance and the probability of it detecting an attack. In addition, because the data is being taken directly from the network level, the time factor becomes an anti-IDS technique, since different attacks separated by time frames higher than those processable by the NIDS Scan are carried

out[10]. This situation is even more bleak if the data at this level is codified with an anti-IDS technique or simply encrypted (HTTPS), in which case the NIDS becomes completely ineffective. Additionally, the sensor must be strategically positioned and must obtain the sum total of the traffic to be analysed, something which can be complicated in a network involving switches.

Taking into account the above-mentioned drawbacks, the first problem consists in defining an architecture which can eliminate the current limitations and offer additional advantages. In this way, the created architecture is based on a reverse proxy which eliminates the problems directly related with the network level for data acquisition, and allows the totality of the data directed toward the Web server to be obtained, automatically discriminating the other protocols, and if this data is encrypted, then at this point it can be decrypted and analysed before being redirected to the Web servers protected in the demilitarised zone (DMZ) or public service zone. Since the data manipulation takes place at application level, there is no fragmentation of the data and it can therefore be processed and analysed correctly and decisions can be taken which transform the system into one which prevents intrusions.

What is more, having separated the analysis and detection level in a reverse proxy, we eliminate the inherent risk of the additional complexity created by an application firewall, and we eliminate the dependence or compatibility problems of the IDS with respect to the protected Web servers and their system architectures. Basically, the IDS system becomes a generic system for the protection of any type of Web server/application.

The use of a reverse proxy as a security mechanism is not new[11]. Some people justify its use in that it forms a single point of entry to Web servers and, above all, because it conceals internal IP addressing. This is completely incorrect, as hiding the internal addresses, as performed by a NAT, is only functional and effective at a security level when a new request coming from an insecure network (Internet) is unable to reach protected hosts identified within valid IPs. In a reverse proxy, requests coming from the Internet always reach the protected Web servers, even if they have invalid IPs.

The use of a reverse proxy as a single point of entry is only efficient if requests are subjected to an analysis or treatment at a security level, and to date, this has simply followed the same static and inefficient focus on signatures for the possible detection of intruders.

## 2.2 Multilevel detection system

In IT security, in the same way that 100% protection does not exist, there are no global solutions to current security problems. For this reason, multiple levels of protection have been created, where each protection level is assigned to carry out its own specific tasks. This means that each level specialises in its work. In this way, the group of level-based protections and specialisations provide higher security "levels", minimising the impact of an existing security risk. The key to this perspective lies basically in providing as suitable a number of levels or layers, fully balancing or identifying the tasks that each level must fulfil (specialisation)[12].

However, academic research on this subject attempts to generate global solutions for a wide range of security problems. This is shown, for example, as a StackOverflow attack (SoF) is treated in the same way and at the same level for detection as an SQL injection. Although they both correspond to well-known security vulnerabilities, both work in a very different way and therefore need different treatment. In an HTTP request that includes an SQL injection code, it is possible to recognise this type of attack due to the fact that it has a well-defined pattern. However, an SoF, due to its characteristics, can be codified in such a manner that it is very difficult to identify [14]. Consequently, the process of identifying an SoF becomes as task equal to or exceeding that of detecting a polymorphic virus.

The system developed at the application level includes two levels. The first level corresponds to a filter which, according to the immunological perspective, forms the “skin” of the system. This level is responsible for rejecting those requests whose length exceeds a predetermined normal limit. The second level receives the requests from the filter in order that they can be analysed and any type of attack in them identified.

### 2.2.1 Filtering level

The IDS must not be approached within any particular paradigm. It must benefit from the characteristics which other systems have developed. In this way, taking into account the immunological perspective, the multi-level system would have its skin as a filter for anomalous characteristics.

This level is a logical adaptation of the tasks carried out by an application firewall; this type of IDS monitors the length and basic composition of the requests specifically determined by a URL, which in theory significantly reduces the risk of a server being compromised.

One of the most important characteristics of the filter is to limit the size of the different permitted HTTP data elements. These data elements include, for example:

Attribute	Max. recommended size (chars)	
	<u>www.eeye.com</u> SecureIIS 1.2.1	<u>www.flicks.com</u> Titan
URL	1024	2047
GET Query	1024	2047
POST Query	olons(64l)-00]TJT*[(P)Vraaeemnc	
Puery	1022800]TJT*[(P)Dtenc	

Accept- Language	256
Accept- Encoding	256
User-Agent	256
Host	256
Connection	256
Cookie	256
If-Modified- Since	256
If-None- Match	256
Authorisation	256

Table 1: Attributes vs. Recommended size.

There is no public evidence of any kind relating to the mechanism or bases used to determine the values recommended by the manufacturers of these and other products. However, due to the sizes of these limits, it can be understood that this protection mechanism against stack overflows is directed at protecting the Web server indirectly but not the applications developed on it (e.g. CGIs) [15].

What would happen if the Web server is susceptible to a buffer overflow in the host with approximately 257 characters and, in addition, a developed CGI uses this same field, but in its development was left vulnerable to a buffer overflow of 60 characters? In implementing the application firewall, the Web server would not be vulnerable to the first BoF (Buffer Overflow) but it would be vulnerable to the second.

It is very difficult to determine with exactitude the maximum recommended limits for the attributes, the difficulty lying in the fact that each Web server is unique. Unique in its users and unique in the type of applications it provides. Therefore, there is a need for a tool that allows the appropriate values to be determined for each environment. This statistical tool reports, in real time, information such as the maximum size of each of the attributes and its percentage of use. With this data, it is much more feasible to determine, for example, that the size of the Host variable never exceeds 45 characters and therefore it would be recommended to set a maximum limit of 45 characters instead of the initial 256. In addition, the percentage of use of the attributes enables us to know precisely the necessity of permitting only those attributes (e.g. Methods) that are actually being used, leaving aside many other options which may contain security problems.

A security problem can occur in any of the attributes managed, regardless of the type of vulnerability (heap overflow, buffer overflow, format bug, etc.). Therefore, limitation of the use of attributes should not be confined just to certain attributes. All the parameters which have their values contained in the URL must be filtered. With the help of statistical analysis of their use and length, it is possible to determine exactly which of the parameters need to be permitted for use.

The mechanism of limiting the lengths of parameters and their values is a tool that allows the great majority of problems related to SoF to be neutralised. Nevertheless, current application firewalls provide a minimum of protection over these, at the level of the application supported on the supposedly protected Web servers. Additionally, in limiting the lengths of the parameters in a request, based on statistical information captured in real time and adjusted by the system administrator, this level identifies the presence of common chains that can identify an attempted attack, for example, “/etc/passwd” or “/repair/sam.\_”.

## 2.2.2 Detection level

One of the main objectives of this level is to be able to detect previously unknown attacks. In this way, an analysis of a sample of 3,000 security problems reported since 1994 in Web servers, CGIs and other related applications [16] showed up, worryingly, that the security problems in HTTP protocol repeat themselves and basically correspond to:

1-Security problems that have been previously published but present themselves in the same form in another application or Web server.

2-Variations of a previously published security problem.

3-Groups of various previously published security problems.

In conclusion, there are no radically new techniques in the arsenal of weapons for exploiting security problems in Web servers, CGIs, or any other applications working with HTTP protocol. However, they continue to appear.

Taking this into account, the detection of unknown attacks is based on making the name, type of application or vulnerable parameter independent from the vulnerability itself. Additionally, a mechanism that is able to generalise based on basic previously learned attack patterns can generalise in order to detect new ones. This type of mechanism is offered by artificial neural networks.

In 1990, in his work “Finding Structure in Time” [17], Dr. Jeffrey L. Elman developed a neural network based on previous work carried out by Jordan (1986). The main objective was to find a solution with respect to the use of neural networks in environments where time was a relevant aspect in addition to inputs. By taking time into consideration, an architecture was developed which manages the context of the input patterns. In this way, a pattern depends on the previous one, and so on. Accordingly, time is no longer represented as an explicit part of the input, but rather it is now represented by the effect it has in the processing. This means giving the processing system dynamic properties that answer to temporary sequences, basically, a memory.

The architecture of an Elman artificial neural network allows the use of his memory to be considered for a variety of problems in which input processing is involved, these inputs naturally being presented in a sequence. An Elman artificial neural network is the principal core of the detection level.

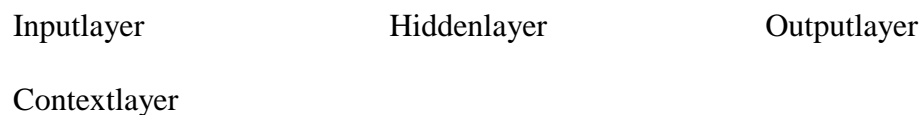


Figure 1: Elman artificial neural network.

The decision about the type of neural network to be used was based on previous studies [1] [18] [19], which demonstrated their effectiveness for the detection of attack patterns in a collection of data (system calls) and their use in recognising patterns in a language [17]. We need to remember that a communication protocol, such as HTTP/1.1, is an adopted language in itself. Additionally, the Elman neural network manages a memory level (context) which takes previous inputs into account when processing a new input.

The inputs come directly from the filtering level and are processed by the Elman neural network to identify an attack in the request.

### 2.3 Detection, identification and classification of attacks

Identifying an attack within a given classification is just as important, or even more important, than detecting the attack itself. Being able to identify an attack within a classification provides additional information to the people responsible for security in an organisation, allowing them to make appropriate additional decisions according to the type of attack occurring. Detection is implicit in the classification, and this is why the neural network developed classifies attacks.

The main problem that presented itself was that of determining a system of classification that would be effective, simple, clear and restricted by well-defined limits. In revising multiple attack classifications presented in similar studies, we were able to establish that the classifications used in the great majority of cases do not have well-defined limits [2] [21], which means that an attack can be included in two or more categories at the same time. And considering that this classification corresponds to the output data of a neural network, it can cause problems not only in the identification of attacks, but also in the net training process. The great majority of research bases its classifications on the direct effects of the vulnerability, which is completely erroneous. For example, a classification which manages Denial of Service (DoS), Command Execution and BoF complicates the identification of the latter, since, if it fails, it may generate a DoS (e.g. overwrite the return address with an arbitrary value), and if it is effective, it can execute system commands or execute its own more complex programs specified in its shellcode. Maybe this type of classification arises from the common classification that is given to certain vulnerabilities when they are published on specialised mail lists.

Furthermore, considering the observations made previously (see 2.2.2) about these security problems in HTTP protocol, by analysing the sample and logs with normal traffic, it is feasible to create a clear, simple and effective classification of security problems, based on their cause and not their effects. This gives rise to the following classification:



1. Cross Site Scripting (XSS).
2. Command Injection.
3. SQL Injection.
4. Stack Overflows (SoF).
5. PATH manipulation.
6. NORMAL

There is a special case, which corresponds to applications with vulnerabilities as simple as, for example,

`Prueba.cgi?file=/etc/passwd`

In reality, this application does not have a vulnerability in itself as the characteristics of the inputs (file) are not being modified in order to modify the behaviour of the CGI. This example CGI is in practice implemented in order to provide the file. It would be counterproductive to train a neural network with this type of pattern, as we would be associating a vulnerability with a filename. Problems of this kind are managed at the filtering level, where the occurrence of chains such as “/etc/passwd” imply the presence of an attack but not a type of associated vulnerability.

## 2.4 Design and training of the Elman neural network

The generation of datasets was carried out using the samples of published vulnerabilities and databases supplied by scanners of available vulnerabilities.

The first approximation for the development of the neural network was that of managing input of variable dimensions for the training stage. This approximation involved many difficulties and was not very effective, and therefore it was decided to use inputs of static dimensions in the training stage and to manage a sliding window for the presentation of requests (URL) when it came to normal operation.

With regard to the presentation of the data, in each pattern, represented by a character chain of fixed size, each character was codified to a value between [0,1] translating each character to its full ASCII value and normalising the vector by dividing each value by 255. This type of codification impeded the learning process, due to the fact that the values generated by the codification operation were so close to each other that it was difficult for the neural network to differentiate between them [22] (Figure 1).

Figure 2: Training codification ASCII/255.

Moreover, the datasets with this codification were regenerated on the basis of base patterns obtained from the initial sample. Each base pattern was processed to generate new training patterns including random alphabetical chains. This in turn generated training datasets which were very large and not very effective.

Subsequently, a codification was developed which allowed a pattern to be represented in binary values. This codification was based on the binary representation of the ASCII value

of each character, converting the input to the neural network into a matrix having a length of  $(8 \times \text{Length\_window})$ , which generates a very large neural network that is difficult to process and consumes a large amount of computer resources. Having said that, the neural network still behaved much better than the first models, as can be seen in Figure 2.

Figure 3: Training codification 8bits.

In order to reduce the size of the neural network, the inputs were processed at a high level, a value of 255 being assigned to each alphanumeric chain, leaving aside program extensions, if they exist, and special characters, which are in the end those that determine the presence of an attack. In this way, a URL is processed and condensed into a basic structure. For example:

*Chain:*

nombre.exe?param1=..\..\archivo

*Type of attack:*

PATH manipulation

*Codification:*

(255)(46)(101)(120)(101)(63)(255)  
(61)(46)(46)(92)(46)(46)(92)(255)

*8-bit Matrix:*

```
11111111
00101110
01100101
....
11111111
```

The decision to maintain the application extensions is due to the fact that many vulnerabilities, whilst not depending on the application itself, do depend on the type of application, which is specified by the extension in most cases. (e.g. manipulation of the PATH of library location in PHP). This codification made it possible to eliminate the dependency on program names and to focus training on generic patterns of attack. It reduced the size of the neural network, as well as the size of the training dataset, which are now composed basically of codified generic patterns, thus achieving shorter training time (Figure 3).

Example of training dataset (without codification to 8bits, '@'=255):

```
<Classification>;<pattern>
COMMAND_INJECTION;
/@;@/@/@|
COMMAND_INJECTION;
;@/@/@|?@
COMMAND_INJECTION;
```

Figure4: Training codification: 8-bit normalised and condensed.

### 3. Results

It is important to bear in mind that the tests undertaken were carried out using scanners which perform real attacks and do not simply verify the existence of a file (e.g. CGI) that is potentially vulnerable. This type of verification is not an attack; rather, it is carried out via a normal request and is therefore identified as normal traffic. Associating the existence of an application with a vulnerability that is commonly related with it is the main cause of false positives. At present, the Elman neural network at detection level has obtained the following results:

*Detection Percentage:*

94.3%

*Identification Percentage:*

90.1%

*False Positives:*

4.7%

*False Negatives:*

0.9%

The identification percentage increases as the quality of the datasets improves, and the datasets currently comprise approximately 740 basic patterns.

In order to determine the quality of the results, the same tests were carried out on the NIDS Snort version 2.0 (<http://www.snort.org>) so that the resulting detection percentages could be compared. It should be mentioned that although Snort is designed to detect attacks in additional protocols other than HTTP, the attacks were only carried out using the same tools (nikto v1.23, nessus v2.0.5) in the same conditions in which the tests on the prototype were generated, in order to guarantee coherent and realistic results. It must be emphasised that the identification percentages are not compared because each IDS has its own classification system. The results for the Snort NIDS were the following:

*Detection Percentage:*

Nessus 59.4%

Nikto 74.9%

The following graph illustrates these results more clearly:

Figure5: Detection Percentages  
Prototype vs. Snort.

#### **4. Conclusions**

Research perspectives into IDS development must be restructured so that they are effective

- [9]NetworkWorkingGroup,“HypertextTransferProtocol-HTTP/1.1”,RFC 2616. <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- [10]LoWNOISE,“Wmapv1.4webscanner”, <http://pwp.007mundo.com/etorres1/>
- [11]ArtStricek,“AReverseProxyIsAProxyByAnyOtherName”,2002, [http://www.sans.org/rr/web/reverse\\_proxy.php](http://www.sans.org/rr/web/reverse_proxy.php)
- [12]DoreneL.Kewley,JohnLowry,“Observationsontheeffectsofdefenseindepthon adversarybehaviorincyberwarfare”,Proceedingsofthe2001IEEE,Workshopon InformationAssuranceandSecurity,UnitedStatesMilitaryAcademy,WestPoint,NY, June,2001.
- [13]FermínJ.Serna,“PolymorphicShellcodesvs.ApplicationIDSs,NextGeneration SecurityTechnologies”, <http://www.ngsec.com>
- [14]FozZy,«AdvancedShellcodes”,DefconX,LasVegas,USA,2002.
- [15]EeyeSECUREIISv1.2.1USERMANUAL <http://www.eeye.com>
- [16]Bugtraq,SecurityFocus, <http://www.securityfocus.com>
- [17]Elman,Jeffrey."FindingStructureinTime".CognitiveScience,14.179-2111990.
- [18]Lippman,R.Conningham,R."ImprovingIntrusionDetectionPerformanceUsing KeywordSelectionandNeuralNetworks".MITLincolnLaboratory,2000
- [19]Draelos,T.Collins,Michael."ExperimentsonAdaptiveTechniquesforHost-Based IntrusionDetection"SandiaNationalLaboratory,SAND2001-3065,2001
- [20]Kumar,S.andSpafford,E., "APatternMatchingModelforMisuseIntrusion Detection,"ProceedingsoftheSeventeenthNationalComputerSecurityConference, pp. 11--21(Oct.1994). <http://citeseer.nj.nec.com/kumar94pattern.html>
- [21]Husmoua,“NeuralNetbasedHost/ApplicationAnomalydetection systems” <http://www.securityfocus.com/archive/96/257986/2002-02-17/2002-02-23/1>
- [22]Farkas,Jennifer.“DocumentClassificationandRecurrentNeuralNetworks”,Industry Canada,CentreforInformationTechnologyInnovation.

## 6.Author

EfraínTorresisastudentofSystemsEngineeringatthePontificiaJaverianaUniversityin Bogotá,Colombia.Forthelastnineyears,hehasbeeninvolvedintheareaofITsecurity, mainlyinthepublicationandexploitationofvulnerabilitiesindifferentsystems, andthe developmentoftoolsforinfiltrationtestsandmethodologies.Heisoneoftheteamofmanagers in

contributorstotheOSSTMM(OpenSourceSecurityTestingMethodologyManual–ISECOM, <http://isecom.org>),andiscurrentlydevelopingthesubjectofthispaperaspartof hisgraduatethesis.

E-mail: [e.torres@javeriana.edu.co](mailto:e.torres@javeriana.edu.co)  
[et@cyberspace.org](mailto:et@cyberspace.org)