A Hybrid Approach for Intrusion Detection System

Hussam A. Al-Ameen

University of Basrah / College of Education for Pure Science, Computer Science Department

Abstract: ***Buffer Overflow (BOF) have become the most common target for network-based attacks and on the other side many detection and prevention techniques have been developed to secure the systems and networks known Intrusion Detection Systems (IDS). The paper deals with the problem of BOF and proposes an IDS which is a combination of Host Intrusion Detection System (HIDS) and Network Intrusion Detection System (NIDS). It is designed to detect any attempt of BOF attack that use the Call/Jump Register technique depending on the use of set of available memory addresses of Call/Jump instructions for loaded DLL files uses them as a return addresses that point to the attacker malicious code being used to exploit the system. The proposed system generates two signature files, one for HIDS and the other for NIDS. The Monitoring and Detection Engine in the HIDS depend on On-Access-Scan technique to capture any file that contains the attack signature as they open and log them to a log file. Besides that, the Monitoring and Detection Engine in the NIDS depends on Snort system to sniff and capture any data packets in the network traffic that contain the attack signature and log them to another log file. An Analysis Engine applies a set of statistical operations and a Fuzzy Analysis System on log files in order to produce a set of reports in the form of PHP web sites that represent the analysis output that give the degree of BOF attack risk.***

Keywords: HIDS, NIDS, Buffer Overflow

1. Introduction

Computer Security is the property of computer systems and networks that specifies that the systems in question and their elements can be trusted to act as expected in safeguarding their owners' and users' information [1]. Security is a big issue for all networks in today's enterprise environment. Hackers and intruders have made many successful attempts to bring down important company networks and web services. Many methods have been developed to secure the network infrastructure and communication over the Internet, among them the use of firewalls, encryption, and virtual private networks.

Intrusion detection represents an important part of such techniques. Using intrusion detection methods, information from known types of attacks can be collected and used and find out if someone is trying to attack the network orparticular hosts. The information collected by this way can be used to strengthen the network security [2].

Buffer overflow attacks are one of the most serious security threats. Nearly 50% of today's widely exploited vulnerabilities are caused by buffer overflow and the ratio is increasing over time. Buffer overflow attacks may cause serious security problems to special purpose embedded systems as well as general purpose systems. With more embedded systems networked, it becomes an important research problem to defend embedded systems against buffer over-flow attacks [3].

To have a secure network, companies must realize that there are two important issues: prevention and detection. Most companies focus their efforts on prevention and forget about detection. For example, more than 90 percent of large companies have firewalls installed, which are meant to address the prevention issue. The problem is double,first, a company cannot prevent all traffic, so some will get through, possibly an attack. Second, most prevention mechanisms that companies put in are either not designed or not configured correctly, which means that they are providing minimal protection if any.

An intrusion can be defined as "*any set of actions that attempt to compromise the integrity, confidentiality or availability of resources*" [4]. The main categories of parties that could attack a computer system and compromise its security must be identified. Nowadays both commercial and open source products are available to defense network security. Many vulnerability assessment tools are also available in the market that can be used to assess different types of security holes present in a network.

A comprehensive security system consists of multiple tools, including:

- Firewalls that are used to block unwanted incoming as well as outgoing traffic of data. There are many firewall products available in the market both in open source and commercial products.
- Intrusion detection systems (IDS) that are used to find out if someone has gotten into or is trying to get into the network.
- Vulnerability assessment tools that are used to find and plug security holes present in the network. Information collected from vulnerability assessment tools is used to set rules on firewalls so that these security holes are safeguarded from malicious Internet users. These tools can work together and exchange information with each other. Some products provide complete systems consisting of all of these products pack together.

This paper presents an Intrusion Detection System which is a combination of two Intrusion Detection Systems, Host and Network Intrusion Detection Systems.

The proposed system is employed to catch and report any BOF attack attempt that depend on *Call/Jump Register Technique* as they occur providing the system administrator with the degree of risk in order to make the necessary actions. The proposed system with its two parts (HIDS and NDIS) is working in two main stages. The first stage is the Monitoring and Detection stage which detects any existence of *Call/Jump Register BOF* attack signature in both of opened files and network traffic then it logs the generated alerts to log files stored in a database. The second stage is the Analysis stage which is working on the log files and produce statistical tables that contain information about the degree of risk for each attack's alert. All outputs will be in the form of web pages.

The rest of the paper is organized as follow. Section 2 describes the main concepts of the intrusion detection system. The buffer overflow attack is then discussed in Section 3. The proposed IDS is presented in Section 4. The discussion of the results is given in Section 5 and Section 6 concludes the paper.

2. Intrusion Detection System

An Intrusion Detection System (IDS) is the high-technology equivalent of a burglar alarm, a burglar alarm configured to monitor access points, hostile activities, and known intruders. The simplest definition of IDS is a specialized tool that knows how to read and interpret the contents of log files from routers, firewalls, servers, and other network devices [4].

2.1 Intrusion Detection Systems Classification

IDS systems vary according to a number of criteria. First, it is possible to distinguish IDSs by the kinds of activities, traffic, transactions, or systems they monitor. IDSs can be divided into network-based, host-based, and distributed. IDSs that monitor network backbones and look for attack signatures are called ***network-based IDSs***, whereas those that operate on hosts defend and monitor the operating and file systems for signs of intrusion are called ***host-based IDSs***. Some of IDSs functioning as remote sensors and reporting to a central management station are known as ***Distributed IDS*** (DIDS) [5]. Figure 1 shows number of IDS classification concepts.

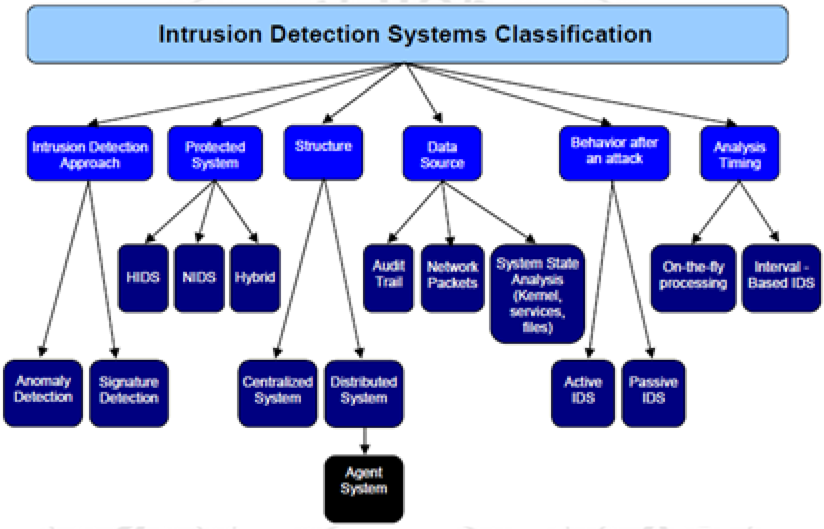


Figure 1: IDS classification concepts

In practice, most commercial environments use some combination of network, host, and/or application-based IDS systems to see what is happening on the network while also monitoring key hosts and applications more closely. IDSs can also be distinguished by their way of event analysis. Some IDSs primarily use a technique called ***signature detection*** [5]. This look like the way many antivirus programs use virus signatures to recognize and block infected files, programs, or active web content from entering a computer system, except that, it uses a database of traffic or activity patterns related to known attacks, called *attack signatures*. Signature detection is the most widely used approach in commercial IDS technology today.

Another approach is called ***anomaly detection***. It uses rules or predefined concepts about "normal" and "abnormal" system activity (called *heuristics*) to distinguish anomalies from normal system behavior and to monitor, report on, or block anomalies as they occur. Some anomaly detection IDSs use user profiles. These profiles represent the baselines of normal activity and can be build using statistical sampling, rule base approach or neural networks [6].

Hundreds of vendors offer various forms of commercial IDS implementations as well as an advanced method for interpreting IDS output. Most effective solutions combine network and host-based IDS implementations. The majority of implementations are primarily signature based, with only limited anomaly-based detection capabilities present in specific products or solutions.

Most modern IDSs include some limited automatic response capabilities, but these usually focus on automated traffic filtering, blocking, or disconnects as a response. Although some systems claim to be able to provide effective action against attacks, best practices indicate that automated identification and back trace facilities are the most useful aspects that such facilities provide and then most likely to be used.

2.2 Network-Based Intrusion Detection System (NIDS)

The NIDS derives its name from the fact that it monitors the entire network or network segment. Normally, a computer Network Interface Card (NIC) operates in nonpromiscuous mode. In this mode of operation, only packets destined for the NICs specific Media Access Control (MAC) address are forwarded up the stack for analysis. TheNIDS must operate in promiscuous mode to monitor network traffic not destined for its own MAC address. In promiscuous mode, the NIDS can eavesdrop on all communications on the network segment. Operation in promiscuous mode is necessary to protect the network. However, according to the privacy regulations, monitoring network communications is a responsibility that must be considered carefully.

In Figure 2, there is a network using three NIDS. The units have been placed on strategic network segments and can monitor network traffic for all devices on the segment. This configuration represents a standard security network topology where the subnets housing the public servers are protected by NIDSs. When a public server is compromised on a subnet, the server can become a launching platform for additional exploits. Careful monitoring is necessary to prevent further damage.

The internal host systems are protected by an additional NIDS to reduce exposure to internal compromise. The use of multiple NIDS within a network is an example of a defense-in-depth security architecture [6].

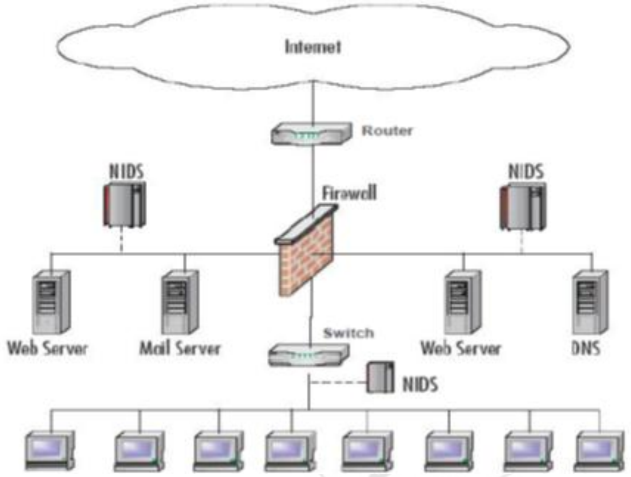


Figure 2: NIDS

2.3 Host-Based Intrusion Detection System (HIDS)

HIDS differ from NIDS in two ways. HIDS protects only the host system on which it resides, and its network card operates in nonpromiscuous mode. Nonpromiscuous mode of operation can be an advantage in some cases, because not all NICs are capable of promiscuous mode. In addition, promiscuous mode can be CPU intensive for a slow host machine.

Another advantage of HIDS is the ability to adjust the rule set to a specific need. For example, there is no need to examine multiple rules designed to detect Domain Name Services (DNS) exploits on a host that is not running. The reduction in the number of relevant rules enhances performance and reduces processor overhead.

Figure 3 shows a network using HIDS on specific servers and host computers. As previously mentioned, the rule set for the HIDS on the mail server is customized to protect it from mail server exploits, while the web server rules are customized for web exploits. During installation, individual host machines can be configured with a common set of rules. New rules can be loaded periodically to be used for new vulnerabilities [6].

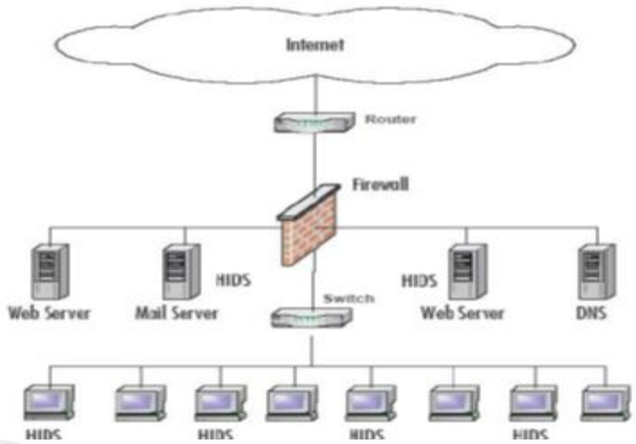


Figure 3: HIDS

3. Buffer Overflow Attack

Buffers are data storage areas, which generally hold a predefined amount of finite data. A buffer overflow occurs when a program attempts to store data into a buffer, and the data is larger than the size of the buffer. For example, an empty 50 ml. glass, this is similar to a buffer. This buffer (empty glass) can store 50 ml. of liquid (data). Now, a bottle, which is about 75 ml, of water wished to be transferred into the empty 50 ml. glass. As begin to fill the glass (buffer) with water (data), everything is fine until the end when water begins to spill over the glass and onto the table. This is an example of an overflow. Such an overflow is bad for water and unfortunately, even worse if such vulnerabilities exist in computer programs [7].

When the data exceeds the size of the buffer, the extra data can overflow into adjacent memory locations, corrupting valid data and possibly changing the execution path and instructions. The ability to exploit a buffer overflow allows one to possibly inject arbitrary code into the execution path. This arbitrary code could allow remote, system-level access, giving unauthorized access to not only malicious hackers, but also to replicating malware.

The majority of buffer overflow attacks involve corruption of procedure return addresses in the memory stack. During the execution of a procedure call instruction, the processor transfers control to code that implements the target procedure. Upon completing the procedure, control is returned to the instruction following the call instruction. This transfer of control occurs in a LIFO (Last In First Out) nested fashion. A procedure call stack, which is a LIFO data structure, is used to save the state between procedure calls and returns [8] [9].

3.1 Memory Organization and the Stack

Each process has its own private address space. The address space is initially divided into three logical segments: text, data, and stack as shown in Figure 4.

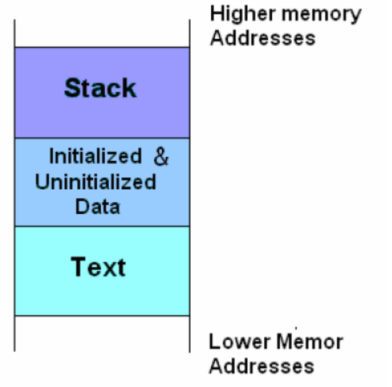


Figure 4: Process memory regions

The Text region is fixed by the program and includes code (instructions) and read-only data. This region corresponds to the text section of the executable file. This region is normally marked read-only and any attempt to write to it will result in a segmentation violation. The Data region contains initialized and uninitialized data. Static variables are stored in this region. In the Stack region, modern computers are designed with the need of high-level languages in mind. The most important technique for structuring programs introduced by high-level languages is the procedure or function. A procedure call alters the flow of control just as a jump does, but unlike a jump, when finished performing its task, a function returns control to the statement or instruction following the call. This high-level abstraction is implemented with the help of the stack [8].

The stack is also used to dynamically allocate the local variables used in functions, to pass parameters to the functions, and to return values from the function. A stack is a contiguous block of memory containing data. A register called the Stack Pointer (SP) points to the top of the stack. The bottom of the stack is at a fixed address. Its size is dynamically adjusted by the kernel at run time. The CPU implements instructions to PUSH onto and POP off of the stack.

The stack consists of logical stack frames that are pushed when calling a function and popped when returning. A stack frame contains the parameters to a function, its local variables, and the data necessary to recover the previous stack frame, including the value of the instruction pointer at the time of the function call.

Depending on the implementation, the stack will either grow down (towards lower memory addresses), or up. On many computers including the Intel, Motorola, SPARC and MIPS processors, a stack grows down. The stack pointer (SP) is also implementation dependent. It may point to the last address on the stack or to the next free available address after the stack. In this paper, we'll assume it points to the last address on the stack [10].

In addition to the stack pointer, it is often convenient to have a Frame Pointer (FP) which points to a fixed location within a frame. Also it can be called a local base pointer (LB). In principle, local variables can be referenced by giving their offsets from LB. As words are pushed onto the stack and popped from the stack, these offsets change. Although in some cases the compiler can keep track of the number of words on the stack and thus correct the offsets, in some cases it cannot, and in all cases considerable administration is required. Furthermore, on some machines, such as Intel-based processors, accessing a variable at a known distance from SP requires multiple instructions [8].

Many compilers use a second register, FP, for referencing both local variables and parameters because their distances from FP do not change with PUSHs and POPs. On Intel CPUs, BP (EBP) is used for this purpose. Because the way the stack grows, actual parameters have positive offsets and local variables have negative offsets from FP.

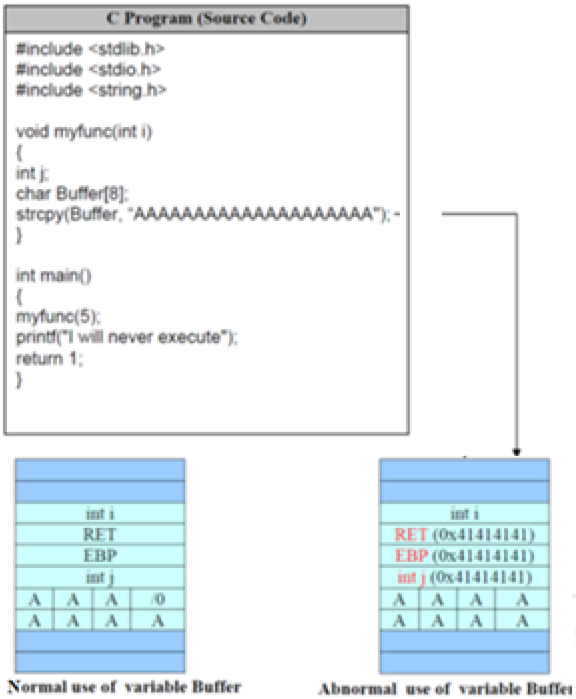
The first thing a procedure must do when called is to save the previous FP or EBP (so it can be restored at procedure exit). Then it copies SP into FP to create the new FP, and advances SP to reserve space for the local variables. This code is called the procedure prolog. Upon procedure exit, the stack must be cleaned up again, something called the procedure epilog. The Intel ENTER and LEAVE instructions have been provided to do most of the procedure prolog and epilog work efficiently.

3.2 Buffer Overflow

A buffer is simply a contiguous block of computer memory that holds multiple instances of the same data type. C programmers normally associate with the word buffer arrays. Most commonly, character arrays. Arrays, like all variables in C, can be declared either static or dynamic. Static variables are allocated at load time on the data segment. Dynamic variables are allocated at run time on the stack [11].

A buffer overflows when too much data is put into it. C language (and its derivatives, like C++), offer many ways to cause more to be put into a buffer than was expected.As local variables can be allocated on the stack. This means that there is a buffer of a set size sitting on the stack somewhere. Since the stack grows down and there are very important pieces of information stored there, when more data are put into the stack allocated buffer than it can handle then like the glass of water, it overflows.

Figure 5 represents a simple example of an uncontrolled overflow. This is not really exploitable. This demonstrates a more commonly made programming error, and the bad effects it can have on the stability of the program. The program simply calls the *myfunc* function. In the *myfunc* function, a string of 20 A's is copied into a buffer that can hold 8 bytes. What results is a buffer overflow. The *printf* in the main function will never be called.



C Program (Source Code)

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void myfunc(int i)
{
int j;
char Buffer[8];
strcpy(Buffer, "AAAAAAAAAAAAAAAAAAAA"); -
}

int main()
{
myfunc(5);
printf("I will never execute");
return 1;
}
```

Figure 5: Uncontrolled Buffer Overflow

3.3 General Exploit Concepts [10]

To perform exploitation, a bit of planning and explanation are needed in order to take the overflows to the stage where EIP (Extended Instruction Pointer) can be controlled and then the advantage of this situation can be taken to gain control of the machine.

After processor control is gained, and thetransfer control of the code has been chosen, the EIP will be pointed to the code that had been written, either directly or indirectly. This is known as ***payload***.

The ***Buffer Injection Vector*** is the custom operational code needed to actually control the instruction pointer EIP on the remote machine. The main role of the injection vector is to get the payload to execute. The payload, on the other hand, is like a virus: it should work anywhere, anytime, regardless of how it was injected into the remote machine.

The payload does not have to be located in the same place as the injection vector, it is just easier to use the stack for both. When using the stack for payload and injection vector, considerations should be taken about the size of payload and how the injection vector interacts with the payload. If these problems become too complex, then the payload should be put somewhere else.

All programs will accept user input and store it somewhere. Any location in the program a buffer can be stored becomes a candidate for storing a payload. The trick is to get the processor to start executing that buffer.

Some common places to store payloads include:
1- Files on disk which are then loaded into memory.
2- Environment variables controlled by a local user.
3- Environment variables passed within a Web request.
4- User-controlled fields within a network protocol.

After injecting the payload, the task is simply to get the instruction pointer to load the address of the payload. The advantage of storing the payload somewhere other than the stack is that it makes the attacker free from constraints on the size of the payload.

3.3.1 Call Register Technique to Execute Payload

There are some techniques used to decide what to put into the saved EIP on the stack to make it finally point to the desired code. One of these techniques is called ***Call Register Technique. In this technique***, if a register is already loaded with an address that points to the payload, the attacker simply needs to load the EIP to an instruction that performs a "call <*register*>" depending on the desired register. For example:

- call EAX FF D0
- call EBX FF D3
- call ECX FF D1
- call EDX FF D2
- call ESI FF D6
- call EDI FF D7
- call ESP FF D4

The usable addresses for these useful pairs can be found by searching the DLL files like (KERNEL32.DLL).These pairs can be used from almost any normal process.

Since these are part of the kernel interface DLL, they will normally be at fixed addresses. They will likely differ between Windows versions, and possibly depending on which Service Pack is applied.

4. The proposed IDS

The proposed Intrusion Detection System is used to detect any attempt to perform BOF attack on the host system connected to the network using *Call-Jump-Ret* technique discussed in previous section. (This attack will be called as *Call Register BOF attack or CR-BOF).*

The proposed system will protect the host system and the network from this type of attack by implementing two intrusion detection modules at the same time. The first one is HIDS that will work to monitor any file access operation and scan this file searching for the attack signature. *On-Access* scan operation will be simulated by using FileSystemWatcher [12] class in the proposed system program. The second module is NIDS that is monitoring the network traffic and capture any packet that contains the attack signature and produce alerts. This part depends on Snort system [13] which is configured to work as NIDS for this type of attack by building a suitable set of detection rules.

Both of HIDS and NIDS will work together to detect the *CR-BOF attack* in what will be called the *Monitoring and Detection Engine*. This can be any computer with good capabilities (called sensor) or can be a server.

During the monitoring and detection session the system log any alert related to finding animus activity into a data base or equivalent means of storing alert data. After a period of monitoring and detection session which is determined by system administrator, the role of *Analyses Engine* will start. Analyses Engine works on the data logged into the Data Base tables and apply number of SQL queries and statistical operation to produce a set of statistical values and then apply a fuzzy system to produce a set of information that will be useful for system administrator to evaluate any anomalous activity in the host computer or through the network traffic. These information reflect the degree of the dangerous and can be used to make a decision in order to protect the host computer and the network. Figure 6 shows the main parts of the proposed system.

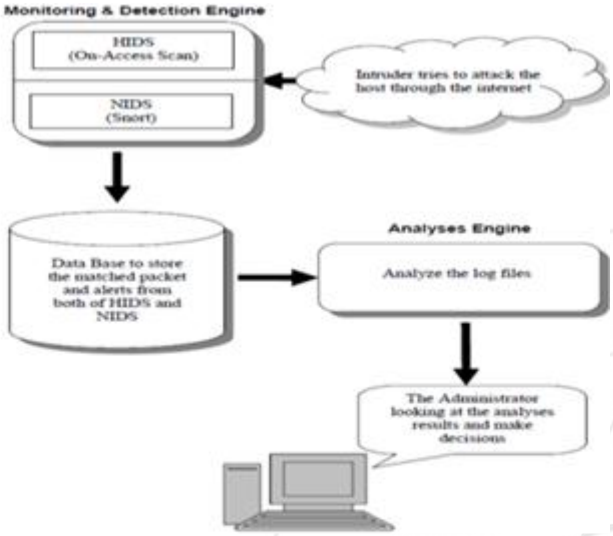
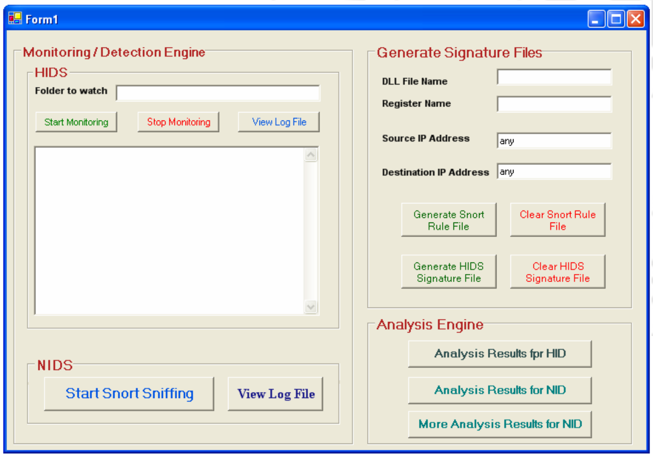

Monitoring & Detection Engine
HIDS
(On-Access Scan)
NIDS
(Snort)
Intruder tries to attack the host through the internet
Data Base to store the matched packet and alerts from both of HIDS and NIDS
Analyses Engine
Analyze the log files
The Administrator looking at the analyses results and make decisions

Figure 6: The proposed IDS main parts

The following figure (Figure 8) shows the User Interface of the proposed system:



Source IP Address any

Destination IP Address any

Figure 8: The Proposed IDS user interface

4.1 Monitoring and Detection Engine

The proposed system works to detect the anomalous intrusion from both of host and network side. This part consists of the following two sub parts:

4.1.1 NIDS Part

Network Intrusion Detection Part will play important role in the proposed system because most of intrusion attacks come from remote computers through the network as e-mails, files or websites etc. The main function of this part is to monitor and scan data flowing on the network and apply some predefined filters on network traffic packets in order to catch any packet that contains one of the *CR-BOF attack* signatures and then generate the desired alerts. For that, a means that is able to sniff, capture and apply filters on network data traffic is needed. Snort is the best choice to do all that. It is used as a tool to build a user defined NIDS by providing deferent working modes and a powerful set of commands and keywords.

Snort uses rules stored in text files that can be modified by a text editor. Rules are grouped in categories and the rules belonging to each category are stored in separate files. These files are then included in a main configuration file called Snort.conf. Snort reads these rules at the start-up time and builds internal data structures or chains to apply these rules to captured data. Snort comes with a rich set of pre-defined rules to detect intrusion activity and we are free to use these built-in rules or remove some of them to avoid false alarms and avoid the need of more processing power that is required to process the captured activity [14] [15].

In the proposed system, one of C++ program will generate Snort rules file which will be applied on captured data. This part works as the following:

- Generating Rules file and place it in Snort\Rules Directory.
- Starting Snort in NID Mode by running this command from the proposed system.

```
Snort –dev –l c:\IDS\Log –c c:\Snort\etc\Snort.conf –i1
```

According to this command, Snort will start working in Network Intrusion Detection Mode (by –c option) [14] [16] in which Snort Detection Engine applies rules given in generated rules file on all captured packets and log the matched packet (by –l option) in the specified log directory and generate alerts. The result log file will be used later in the analysis part of the proposed system to make the decisions.

- Each rule in the generated rule file has the following format:

```
alerttcp 10.30.0.112 any -> 10.30.0.85 any (content: "|7B 1A 80 7C|"; msg: "7C801A7B";)
```

The following is the meaning of each part of the rule [14] [16]:

- *alert*is the action part in Snort rule header.
- *tcp*is the protocol part that shows on which type of packet the rule will be applied.
- *10.30.0.112* is the source IP address part in Snort rule header.
- *any*is the source port number part in Snort rule header that can be a port number or 'any'.
- represent the direction of the packet so the IP address and port number to the right of it will be the destination IP address and destination port number.
- *content*is one of Snort keywords that can be used in the options part of Snort rules to find a data pattern inside a packet.

As can be seen, the parameter of the *content* keyword is one of the found addresses (useful jump point) that we are looking for as a *CR-BOF attack* signature. Also, the parameter of *msg*keyword is the same address used in *content* part of the rule and this means that the alert message will give the address when it is found in one of captured packet.

In Snort configuration file (snort.conf), the Output Module can be defined to control the output from Snort Detection Engine. Snort has the following output modules [15]:

1) The Database Output Module: This module is used to store Snort output data in databases.
2) The SNMP Output Module: This module can be used to send Snort alerts in the form of traps to a management server.
3) The SMB alerts Output Module: This module can send alerts to Microsoft Windows machines in the form of pop-up SMB alert windows.
4) The syslog Output Module: This module logs messages to the syslog utility. Using this module, messages can be logged to a centralized logging server.
5) The XML Output Module: This Module is used to save data in XML so they can be read and interpreted by any XML-based interpreter or browser.
6) The CSV Output Module: This Module is used to save data in *Comma Separated Value* files. The CSV files can then be imported into databases or spreadsheet software for further processing or analysis.

In the proposed system, *CSV Output Module* is used. This module was chosen for the following reasons:

- It is easy to define what information (fields) should be stored in the CSV file and in what order.
- It is useful when someone wants to import data into other software packages like databases and spreadsheets, e.g., Microsoft Excel. To configure Snort to use the CSV output format, the following line should be added in the Snort configuration file (Snort.conf):

output csv: <filename><formatting_options>

This configures Snort to create a CSV log file named <filename> in the configured logs directory using the formatting options that specify what output fields we want to store in the CSV file . One can use only a few of these options in the CSV file as required. In the NIDS part of the proposed system, the following line in *Snort.conf* file will be used that will record only *timestamp*, *msg*, *src*and *dst* fields in CSV file called (*Snortalert.csv*)

outputalert_csv: Snortalert.csv timestamp,msg,src,dst

where:

- **timestamp**= Time stamp including date and time.
- **msg**= Message which is taken from the msg option of the rule.
- **src**= Source IP address.
- **dst**= Destination IP address.

At this point, the role of NIDS is finished giving the log file that will be used as an input to the Analysis Engine.

4.1.2 HIDS Part

This part will function as a second protection level in the proposed system. If the intruder succeed in putting his payload code in the victim computer through the network (by different available means) or putting the payload directly in the computer or by using a payload that is already available in the victim computer, then, this part will play a good role to detect the using of this payload. The proposed system will work on the case in which the payload is in the form of file that will be used by the target program as an input file, then the proposed HIDS part should monitor the access process to any file in real time and check this file for the *CR-BOF attack* signature. This process is being used by most of Anti Virus programs and it is called *On-Access Scan*. In the proposed system, a simulation of this On-Access Scan process was programmed by using the FileSystemWatcher Class provided by Microsoft Visual Studio.NET programming languages.

FileSystemWatcher is a very powerful component, which allows programmer to connect to the directories and watch for specific changes within them, such as creation of new files, addition of subdirectories and renaming of files or subdirectories. This makes it possible to easily detect when certain files or directories are created, modified or deleted. It is one of the members of *System.IO Namespace*. The component can watch files on a local computer, network drive or a remote machine [12] [17] [18].

In the proposed system's program, the change in File Access Time is monitored as a notification filter in order to catch the file when any program attempt to open it, and then, the On-Access Scan process is simulated. In the proposed system's program, we have to determine the folder or the drive we want to monitor its contents and the type of files we want to watch by specifying the extension .The proposed system interface enables the administrator to specify the directory or drive we want to watch. After catching the file, the path of this file can be got and the file can be opened and scanned for the desired signature. In this part of the proposed system, the first signature file that contains the addresses only will be used. Also, more than one type of files can be monitored by creating more than one FileSystemWatcher object and making each one use deferent file extension.

The output of this part will be a log file that contains the name and path of the files that contains one or more of the *CR-BOF attack* signature and the found addresses (signatures). This log file will be used later as an input of the analysis part of the system.

4.2 Data Base

This part is used to store the information that logged from the Monitoring and Detection Engine. This Database can be in different forms according to the mode being used in Monitoring and Detection Engine and according to the type of analysis operations and results required from the Analysis Engine and finally according to system administrator needs.

In the proposed system, the CSV files represents the Database part .We have two CSV log files ,the first one stores HIDS log data as a table that contains *Address* and *Path/File Name* fields. The second one stores NIDS log data as a table that contains *Date/Time*, *Signature*, *Source IP Address* and *Destination IP Address* fields. MySQL Database is another type of Database that can be used in this part. MySQL Database can be used to store the logged data. In this case, we need to use MySQL Database Server which enable us to use all of MySQL capabilities like Distributed and Network Database and all the power of SQL Language [19].

4.3 Analysis Engine

This part of the proposed system works on the logged data stored in the Database and after one or many sessions of monitoring and detection done by Monitoring and Detection Engine. The output of this engine will be a set of statistical tables in the form of PHP web pages that enable the administrator to make the necessary decisions according to the information gained from these tables. the *Detection Engine* will log all the alerts into two main log files , the first one is related to HID part and the second one is related to NID. The *Analysis Engine* will take these two files as its input and perform number of statistical operations in order to produce a number of statistical values. These result values will be used as an input to a Fuzzy analysis system [20] in order to classify the intrusion attempts to the levels of BOF alerts, each level represents the degree of risk given as a percentage in the final statistical table that represents the output of the *Analysis Engine* (Figure 7).

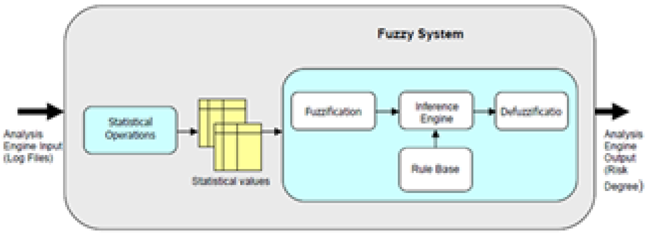


Figure 7: Analysis Engine part of the proposed IDS

5. Result and Analysis

Intrusion detection is a technique employed to catch and report attacks as they occur. IDS rely on data sources which can be drawn from OS audit logs, sniffed network packets, logs from components such as routers or applications. There are a number of associated problems such as :

1) How can IDS's be evaluated?
2) Do an IDS produce too many false positives?
3) Can IDS be used to detect unknown attacks?
4) How can reports of multiple IDS's be combined to detect attacks?
5) How to integrate intrusion detection and automated response?

The present paper proposed and implement a two parts IDS. The first is HIDS and the second is a NIDS. The system can be considered to partially addresses problems 3,4 and 5 listed above.

- The system may be evaluated (problem 1) if experimentation is dedicated for problem 2.
- The system is fully entitled to consider problem 3 for BOF attacks. Its detection criteria is based on "important addresses" not on base signature.
- The system deals with host attacks and with network attacks at the same session. Integrating Report on both channels is a matter of DBMS. MY-SQL solves this easily.
- The mechanism for automated response is API application. The response of the proposed system is in the form of an *alert* to the System Administrator. If the proposed system is fully evaluated, the response automation is a programming problem.

6. Conclusion

By understanding the concepts of BOF attack we can find that the key parameter behind BOF attack is the "Return Address" and how to change it to be pointed to the exploit code. The proposed system can be considered as following:

- Host and Network (Hybrid) IDS according to protected system.
- Almost signature based IDS according to the Intrusion detection approach.
- Network packets and system state analysis IDS according to data source.
- Passive IDS according to behavior after attack. That is because it generates alerts and log files without a real time response to the attack.
- Interval based IDS according to analysis timing.

Although the proposed IDS considered as a signature based IDS and this means that it is used to detect known attacks only and cannot detect new or unknown attacks. The signatures that it looks for, doesn't represent a signature of known attack but a signature of call register technique that can be used to perform any attack depend on BOF vulnerability.

References

[1] Robert Courtney, James Burrows, F. Lynn McNulty, Stuart Katzke, Irene Gilbert, and Dennis Steinauer, "*An Introduction to Computer Security*". National Institute of Standards and Technology NIST, 1996.

[2] RebeccaBace , "*An Introduction to Intrusion Detection and Assessment*". Infidel, Inc. for ICSA, 2000.

[3] James C. Foster, VitalyOsipov, Nish Bhalla and Niels Heinen, "*Buffer Overflow Attacks: Detect, Exploit, Prevent*". Syngress Publishing, Inc., 2005.

[4] Amrita Anand, Brajesh Patel, "*An Overview on Intrusion Detection System and Types of Attacks It Can Detect Considering Different Protocols*" , International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, Issue 8, pp. 94-98, 2012.

[5] IndraneelMukhopadhyay, Mohuya Chakraborty, SatyajitChakrabarti, "*A Comparative Study of Related Technologies of Intrusion Detection & Prevention Systems*". Journal of Information Security, Vol. 2, No. 1, pp.28-38, 2011.

[6] KarenScarfone, Peter Mell, "*Guide to Intrusion Detection and Prevention Systems (IDPS)*". National Institute of Standards and Technology (NIST) Special Publication 800-94 Revision 1, 2012.

[7] EricChien and PéterSzör, "*Blended Attacks Exploits, Vulnerabilities and Buffer-Overflow Techniques in Computer Viruses*". Symantec Corporation ,Virus Bulletin Conference ,USA , 2002.

[8] MartinVuagnoux, "*An Intruduction to Buffer overflow Attacks*". Swiss Federal Institute of Technology, Computer Security and Cryptography Laboratory, 2005.

[9] Sahel Alouneh, HebaBsoul, MazenKharbutli, “*Protecting Binary Files from Stack-Based Buffer Overflow*”, Springer-Verlag Berlin Heidelberg, Information Science and Applications chapter, Lecture Notes in Electrical Engineering series, Vol. 339, pp 415-422, 2015.

[10] Ryan Russell, Dan Kaminsky, Rain Forest Puppy, Joe Grand, K2, David Ahmad, Hal Flynn, IdoDubrawsky, Steve W. Manzuik, Ryan Permeh, "*Hack Proofing Your Network Second Edition*". Syngress Publishing, Inc., 2002.

[11] Tzi-ckerChiueh and Fu-Hau Hsu, "*RAD: A Compile-Time Solution to Buffer Overflow Attacks*". Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS), 2001.

[12] JulianTempleman,"*Microsoft Visual C++/CLI Step by Step*" , Microsoft Press, 2013.

[13] G. D. Kurundkar, N. A. Naik and S. D. Khamitkar, "*Network Intrusion Detection using SNORT*", International Journal of Engineering Research and Applications (IJERA), Vol. 2, Issue 2, pp. 1288-1296, 2012.

[14] Jay Beale and James C. Foster, "*Snort 2.0 Intrusion Detection, Second Edition*". Syngress Publishing, Inc., 2004.

[15] Rafeeq Ur Rehman, "*Intrusion Detection Systems with Snort, advanced IDS Techniques Using Snort, Apache, MySQL, PHP, and ACID*". Pearson Education, Inc.,2003.

[16] Mukta Garg, "*Intrusion Detection System in Campus Network: SNORT – the most powerful Open Source Network Security Tool*", International Journal of Advancement in Engineering Technology, Management & Applied Science, Vol.1, Issue 5, 2014.

[17] Bruce Johnson, "*Professional Visual Studio 2013*" , John Wiley & Sons, Inc. , 2013.

[18] Michael Halvorson, "*Microsoft Visual Basic 2013 Step by Step (Step by Step Developer)*", Microsoft Press, 2013.

[19] Michael Steele, "*Snort 1.8.7 for Windows NT Server / 2000/ XP using Apache Webserver, MySQL and Acid to view and alerts*".http://www.forum-intrusion.com/snort/snort2.html, 2002.

[20] Mostaque Md., Morshedur Hassan, "*Current Studies On Intrusion Detection System, Genetic Algorithm And Fuzzy Logic*", International Journal of Distributed and Parallel Systems (IJDPS) Vol.4, No.2, 2013.