ISSN:0975-9646

Wireless Intrusion Detection System

[1*]Keith Chettiar, [2]Akshay Patil, [3]Arpita Pradhan and [4] Ms. Sushama Khanvilkar

[1*, 2, 3, 4] *Computer Department, Xavier Institute of Engineering, Mahim, Mumbai, India.*

Abstract— **The rapid proliferation of wireless networks and mobile computing applications has changed the landscape of network security, the wireless networks have changed the way business, organizations work and offered a new range of possibilities and flexibilities; It is clear that wireless solutions are transforming the way we work and live. Employees are able to keep in touch with their e-mail, calendar and employer from mobile devices, but on the other hand they have introduced a new security threat. While an attacker needs physical access to a wired network in order to gain access to the network and to accomplish his goals, a wireless network allows anyone within its range to passively monitor the traffic or even start an attack. One of the countermeasures that can be used in order to allow us to know both the threats affecting our wireless network and our system vulnerabilities is the Intrusion Detection System. WLANs have a number of security-related issues. They are as follows:**
1] Data interception
2] Denial Of Service (DOS)
3] Rogue APs.
4] Other 802.11 related security threats.
WLANs typically encompass a relatively large physical coverage area. In this situation, many WAPs can be deployed in order to provide adequate signal strength to the given area. An essential aspect of implementing a Wireless IDS is to deploy it wherever a WAP is located. By providing comprehensive coverage of the physical infrastructure at all WAP locations, the majority of attacks and misuse can be detected.

Keywords— **Libpcap; Libnet; Boyer-Moore; Sniffer Mode; Packet Logger Mode; Intrusion Detection Mode; Barnyard2; ADODB; Unified2; BASE.**

I. Introduction

Threats to Wireless Local Area Networks (WLANs) are numerous and potentially devastating. Security issues ranging from misconfigured Wireless Access Points (WAPs) to session hijacking to Denial of Service (DoS) can plague a WLAN. Wireless networks are not only susceptible to TCP/IP-based attacks native to wired networks, they are also subject to a wide array of 802.11-specific threats. To aid in the defence and detection of these potential threats, WLANs should employ a security solution that includes an Intrusion Detection System (IDS).
A Wireless Intrusion Detection System (WIDS) monitors the radio spectrum for the presence of unauthorized, rogue access points and the use of wireless attack tools. The system monitors the radio spectrum used by Wireless LANs, and immediately alerts the system administrator whenever a rogue access point is detected. One such Intrusion Detection System that we have used is Snort.

II. Snort As An Ids

Snort is a free and open-source Network Intrusion Prevention System and Network Intrusion Detection System created by Martin Roesch in 1998; now maintained by Sourcefire. Snort's Intrusion Detection System has the ability to perform real-time traffic analysis and packet logging on Internet Protocol (IP) networks. Snort performs protocol analysis, content searching and content matching. These basic services have many purposes including application-aware triggered quality of service, to de-prioritize bulk traffic when latency-sensitive applications are in use.
Snort can be configured in three different modes: sniffer, packet logger and intrusion detection. In sniffer mode, the program will read network packets and display them on the console. In packet logger mode, the program will log packets to the disk. In intrusion detection mode, the program will monitor network traffic and analyze it against a rule set defined by the user. The program will then perform a specific action based on what has been identified.

III. Prerequisites For Snort

A. *Capturing Packets using Libpcap*

In the field of computer network administration, libpcap is used for capturing network traffic. Libpcap is almost the standard for grabbing packets off the wire in snort and is used by many protocol-decoding applications.

B. *Data Acquisition library*

Data Acquisition Library or DAQ, replaces direct calls into packet capture libraries like libpcap with an abstraction layer that makes it easy to add additional software or hardware packet capture implementations. The DAQ is essentially an abstraction layer and a suite of pluggable modules that can be selected at run-time. This makes switching from passive to inline mode easy, and does not require a recompile of the snort core.

C. *Libnet as an API*

Libnet is an API to help with the construction and handling of network packets. It is used in conjunction with libpcap and it provides framework for low-level network packet writing and handling. Libpcap includes packet creation at the IP layer and at the link layer as well as a host of supplementary and complementary functionality.

I. ALGORITHMIC EFFICIENCY

In earlier days, snort used brute force pattern matching which was slow and was seen as a place where performance could be improvement. The first thing done to boost performance was implementing a partial Boyer-

Moore pattern matching algorithm. After a couple of months a full implementation of Boyer-Moore was implemented[1].

The algorithm pre-processes the string being searched for (the pattern), but not the string being searched in (the text). It is thus well-suited for applications in which the pattern is much shorter than the text or does persist across multiple searches. The Boyer-Moore algorithm uses information gathered during the pre-process step to skip sections of the text, resulting in a lower constant factor than many other string algorithms. In general, the algorithm runs faster as the pattern length increases. The key feature of the algorithm is to match on the tail of the pattern rather than the head, and to skip along the text in jumps of multiple characters rather than searching every single character in the text.

The Boyer-Moore algorithm searches for occurrences of pattern text by performing explicit character comparisons at different alignments. Instead of a brute-force search of all alignments (of which there are m - n + 1), Boyer-Moore uses information gained by pre-processing pattern to skip as many alignments as possible.

The algorithm scans the characters of the pattern from right to left beginning with the rightmost one. In case of a mismatch (or a complete match of the whole pattern) it uses two pre-computed functions to shift the window to the right. These two shift functions are called the good-suffix shift (also called matching shift) and the bad-character shift (also called the occurrence shift)[4].

Assume that a mismatch occurs between the character x[i]=a of the pattern and the character y[i+j]=b of the text during an attempt at position j. Then, x[i=1 .. m-1]=y[i+j+1 .. j+m-1]=u and x[i]!=y[i+j]. The good-suffix shift consists in aligning the segment y[i+j+1 .. j+m-1]=x[i+1 ..m-1] with its rightmost occurrence in x that is preceded by a character different from x[i][4].

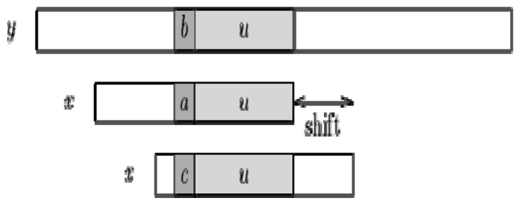


Figure 1: The good-suffix shift, u re-occurs preceded by a character c different from a.

If there exists no such segment, the shift consists in aligning the longest suffix v of y[i+j+1 .. j+m-1] with a matching prefix of x.

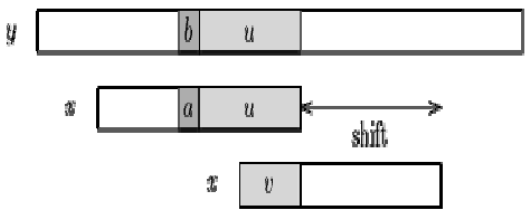


Figure 2: The good-suffix shift, only a suffix of u re-occurs in x.

The bad-character shift consists in aligning the text character y[i+j] with its rightmost occurrence in x[0 .. m-2].

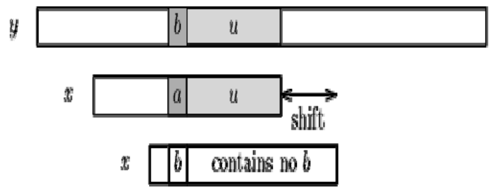


Figure 3: The bad-character shift, b does not occur in x.

If y[i+j] does not occur in the pattern x, no occurrence of x in y can include y[i+j], and the left end of the window is alignedd with the character immediately after y[i+j], namely y[i+j+1].

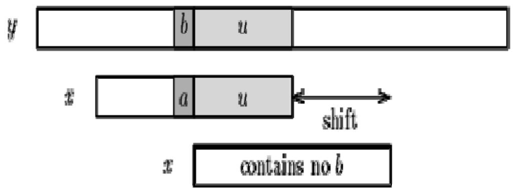


Figure 4: The bad-character shift, b does not occur in x.

Note that the bad-character shift can be negative, thus for shifting the window, the Boyer-Moore algorithm applies the maximum between the good-suffix shift and bad-character shift. More formally the two shift functions are defined as follows.

The good-suffix shift function is stored in a table *bmGs* of size *m*+1.

Let us define two conditions:

Cs(*i*, *s*): for each *k* such that $i < k < m$, $s \geq k$ or $x[k-s]=x[k]$ and

Co(*i*, *s*): if $s < i$ then $x[i-s] \neq x[i]$

Then, for $0 \le i < m$: $bmGs[i+1]=\min\{s>0 : Cs(i, s) \text{ and } Co(i, s) \text{ hold}\}$ and we define *bmGs*[0] as the length of the period of *x*. The computation of the table *bmGs* use a table *suff* defined as follows:

for $1 \le i < m$, $suff[i]=\max\{k : x[i-k+1 .. i]=x[m-k .. m-1]\}$

For *c* in Σ: $bmBc[c] = \min\{i : 1 \le i < m-1 \text{ and } x[m-1-i]=c\}$ if *c* occurs in *x*, *m* otherwise.

Tables *bmBc* and *bmGs* can be pre-computed in time $O(m+\sigma)$ before the searching phase and require an extra-space in $O(m+\sigma)$. The searching phase time complexity is quadratic but at most 3*n* text character comparisons are performed when searching for a non periodic pattern. On large alphabets (relatively to the length of the pattern) the algorithm is extremely fast. When searching for $a^{m-1}b$ in b^n the algorithm makes only $O(n / m)$ comparisons, which is the absolute minimum for any string-matching algorithm in the model where the pattern only is pre-processed[4].

IV. WORKING WITH RULES

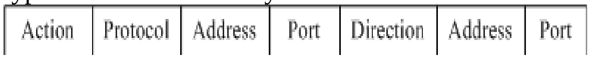
A. Structure of a Rule

All snort rules have two logical parts: rule header and rule options.

Rule Header	Rule Options

Figure 5: Structure of a rule

The rule header contains information about what action a rule takes. It also contains criteria for matching a rule against data packets. The options part usually contains an alert message and information about which part of the packet should be used to generate the alert message. The options part contains additional criteria for matching a rule against data packets. A rule may detect one type or multiple types of intrusion activity.



Action	Protocol	Address	Port	Direction	Address	Port

Figure 6: General structure of a rule

The action part of the rule determines the type of action taken when criteria is met and a rule is exactly matched against a data packet. Typical actions are generating an alert or log message or invoking another rule.
The protocol part is used to apply the rule on packets for a particular protocol only. This is the first criterion mentioned in the rule. Some examples of protocols used are IP, ICMP, UDP etc.
The address part defines source and destination addresses. Addresses maybe a single host, multiple hosts or network addresses. You can also use these parts to exclude some addresses from a complete network.
In case of TCP or UDP protocol, the *port* part determines the source and destination ports of a packet on which the rule is applied. In case of network layer protocols like IP and ICMP, port numbers have no significance.
The direction part of the rule actually determines which address and port number is used as source and which as destination.

V. Running Snort

As mentioned earlier, snort runs in three different modes.

A. *Sniffer Mode*

For sniffing network traffic, there are three command line switches:
-d : Dump or display application layer information.
-e : Dump or display data link layer information.
-v : Be verbose.
The -v switch by itself simply outputs TCP headers to the screen. Adding the -d switch includes the payload information in addition to the headers. Lastly, the -e switch includes the data link information.
The command line syntax for running Snort in this mode is as follows:

```
#~ snort -vde
```

If you will be using Snort in this mode, it is important to note that the -d and/or -e switches requires you to use the -v switch. The effect of not doing so is the same as not specifying any switches which force Snort to look in your home directory for configuration file. If one is not found it will simply display a listing of the possible command line options followed by the message, "Uh, you need to tell me to do something...".
If you have initiated a packet sniffing session with Snort, you can end it by issuing a <Ctrl + c>.

B. *Packet Logger Mode*

The command line syntax for running Snort in this mode is as follows:

```
#~ snort -i eth0 -l /var/log/snort -K ascii
```

In the example above, the -l option instructs Snort to put the log data in the directory specified. In this case: /var/log/snort. If Snort is in ASCII logging mode, it will actually create a directory structure in the specified directory that contains a directory for each IP address it captures for the duration of the session. In the IP subdirectory, if places files named after the protocol and, if applicable, the source and destination port numbers. You can open these files with the standard command (cat, more, less, etc.) and view their contents. These are flat ASCII text files.
You can also use Snort to log data directly to a PCAP readable binary format with the -b option. This has the advantage of relieving the Snort process from having to convert the output to human readable format as it outputs the capture data. Another advantage is that the PCAP format can be re-read into the Snort engine with the -r option.

C. *Intrusion Detection Mode*

Snort is mostly used for its intrusion detection capabilities. In a production capacity, Snort would normally be called up from the command line in 'Detection' mode even if another instance is already running.
The only difference between running Snort in packet capture/logging mode and detection mode is the addition of an option to call up the configuration file: snort.conf. This file is typically placed in the /etc directory structure.
The command line syntax for running Snort in this mode is as follows:

```
#~ snort -c /etc/snort/snort.conf
```

The -c option allows you to print the location of the configuration file. The above given example starts Snort in detection mode, and implements any configuration settings that has been made in the snort.conf file.

VI. Database Output And Graphical Analysis

A. *Unified2 Output Formats*

Snort has the ability to produce a fast, binary output format called the unified2 format. The idea behind this capability is to have other applications do the work of processing Snort output, thus relieving the Snort process. This makes the Snort run more efficiently since it can concentrate more of its efforts on processing packets rather than having to also worry about output.
Unified2 output can produce three types of files:
alert_unified2
log_unified2
unified2
The alert s simply information about the alert, which includes some of the packet header information in addition to the alert information, such as alert message, SID and revision number if so configured in the rule. The packet log file contains the full packet information that triggered the alert which also includes the alert

information. Unified includes both logging styles in a single, unified file.

B. *Barnyard2*

Barnyard2 takes advantage of the unified output that is created and allows to configure the output in a variety of ways. Like Snort, Barnyard2 can produce output in many formats including ASCII, PCAP or database output. Barnyard2 creates and stores all its logs in /etc/log/barnyard2.

The command line syntax for running barnyard2 is as follows:

```
#~ barnyard2 -c /etc/snort/barnyard2.conf -d /var/log/snort -f snort.u2 -w /var/log/snort/barnyard2.waldo \ -g snort -u snort
```

Barnyard2 will startup and then it will process the alerts in the /var/log/snort, write them to both the screen and the database, and then wait for more events to appear in the /var/log/snort directory.

C. *Base*

BASE stands for Basic Analysis and Security Engine, which provides a web front-end to the database of Snort events. Additionally, we use ADODB package to provide an interface between the GUI and the MySQL database.

Configuration of BASE is done through its web page:

Browse to "http://<snort-ip-address>/base/index.php" and click on "setup page" link.

Click on "Create BASE AG" button on the upper right of the page.

Click on the "Main Page" line.

BASE is now configured and all the events generated through Snort should be visible.

VII. CONCLUSION

The motivation of this project is to create an application to prevent hackers from exploiting a wireless network. To achieve this objective, a rule based Intrusion Detection System is used. This Intrusion Detection System either can be used as a stand-alone application over a given network or can be coupled with other third-party interfacing tools for administration, reporting, performance and log analysis. Apart from BASE, there are other analysis consoles available such as SGUIL, Snorby, Snort Report which can be equally be used as a console in a production environment suiting one's own needs.

ACKNOWLEDGEMENTS

We are thankful to our project guide and project coordinator for guiding us in making these project a success.

REFERENCES

[1] http://www.linuxsecurity.com/resource_files/intrusion_detection/Increasing_Performance_in_High_Speed_NIDS.pdf

[2] http://en.wikipedia.org/wiki/Snort_%28software%29

[3] http://linton.tw/2014/08/17/Install-Snort-from-source-on-Ubuntu/

[4] http://www-igm.univ-mlv.fr/~lecroq/string/node14.htm