2008

Web Security Detection Tool

Abhishek Agashe
San Jose State University

Web Security Detection Tool

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

by

Abhishek Agashe

December 2008

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

_____

Dr. Chris Tseng

Professor of Computer Science, San José State University

_____

Dr. Chris Pollett

Professor of Computer Science, San José State University

_____

Dr. Robert Chun

Professor of Computer Science, San José State University

_____

ABSTRACT

Web Security Detection Tool

by Abhishek Agashe

According to Government Computer News (GCN) web attacks have been marked as all-time high this year. GCN says that some of the leading security software like SOPHOS detected about 15,000 newly infected web pages daily in initial three months of 2008 [13]. This has lead to the need of efficient software to make web applications robust and sustainable to these attacks. While finding information on different types of attacks, I found that SQL injection and cross site scripting are the most famous among attackers. These attacks are used extensively since, they can be performed using different techniques and it is difficult to make a web application completely immune to these attacks. There are myriad detection tools available which help to detect vulnerabilities in web applications. These tools are mainly categorized as white-box and black-box testing tools.

In this writing project, we aim to develop a detection tool which would be efficient and helpful for the users to pinpoint possible vulnerabilities in his/her PHP scripts. We propose a technique to integrate the aforementioned categories of tools under one framework to achieve better detection against possible vulnerabilities. Our system focuses on giving the developer a simple and concise tool which would help him/her to correct possible loopholes in the PHP code snippets.

ACKNOWLEGEMENTS

I would like to thank Dr. Tseng for his guidance and support through out my writing project. I would also like to thank Dr. Chun and Dr. Pollett for being my committee members and giving me valuable suggestions.

I am grateful to my family and friends for their continuous encouragement during this project.

Table of Contents

List of Table and Figures

1. Introduction

Internet has become an imperative part in present industry and our lives. Due to its widespread use web security has become a vital issue since last decade. There has been an exponential increase in e-commerce and online transactions in past few years. This has lead to need for developers to build robust and sustainable web applications. The most common reason for a susceptible web application is sloppy coding techniques. It is found that though it is not easy to make web applications immune to theses attacks, one can definitely mitigate them by taking some precautions. Developers tend to ignore security precautions due to constant deadlines and time constraints. This has resulted in increasing number of exploits in web based applications.

This project is based on the current needs for web application developers to create high end applications without compromising on security. The aforementioned conditions have lead to increasing demand of vulnerability detection tools to aid developers to deliver time critical and robust software. This project focuses on developing a detection tools which would not only help the user detect possible vulnerability in his code, but will also help in locating the vulnerable line. The contemporary vulnerability detection tools are categorized as white box and black box testing. This project tries to exploit advantages of both these testing techniques by combining them under one framework. This technique helps the two separate tools to complement each others results and help application developer to better pinpoint vulnerabilities in their code.

This project accepts only PHP scripts and a URL to the web hoisted application. The system will give the user a formatted result for possible vulnerabilities in his/her scripts.

The rest of the report is organized as follows:

- Chapter 2 gives a detailed background information related to SQL injection (SQLI) and Cross-Site scripting (XSS). It contains the most commonly used SQLI and XSS attacks along with a few techniques to mitigate these attacks. This section would also enumerate the possible effects of these attacks on a target application. This section concludes with list of PHP functionality which could be used to debilitate the aforementioned attacks.
- Chapter 3 starts with an overview of open source projects used as detection tools for SQLI and XSS attacks. This section includes detailed description of the two testing tools including PIXY and WAPITI. There is also a comparison of these tools with other famous detection tools.
- Chapter 4 describes the working of the detection tool. This section starts with an overview and the architecture. Later it also discusses about the detection process adopted along with the output generated by this tool.
- Section 5 discusses the testing phase of our system along with analysis of the results.
- Section 6 concludes the report and discusses about further enhancement on this project.

2. Background

2.1 SQL Injection

SQL injection is a technique used by attackers to exploit a vulnerability found in the database of an application [2]. This technique is often used on web applications which acquire inputs from user to process them and present relevant information. It is found that this technique is very prevalent in acquiring confidential information from application login pages. SQL injection can be performed on any web application based on almost any web technology like ASP, ASP.NET and PHP with any flavor of SQL database at the back-end. SQL injection is typically performed on user login pages where, the user information namely username and password are requested from the users. Then these username and passwords are compared with the list of legitimate users in the database. Once a match is found an SQL query is fired behind the application to display necessary user information. Now, in SQL injection an attacker uses a specially crafted SQL query syntax and passes it to such applications through the login page.

Consider the following example; where the attacker uses specially crafted query like **admin' or 1=1#** as username. In this case the user input is not sanitized by the application and is directly used to build queries like:

"SELECT * FROM uname_pwd WHERE username='".$_GET['username']."' and password ='".($_GET['password'])."'";

Thus, in this case the query is rendered useless as the syntax causes username to be accepted as '**1=1'**, which is always true and the rest of the query is bypassed by SQL

comment syntax “#”. Following figure gives an instance of such attack

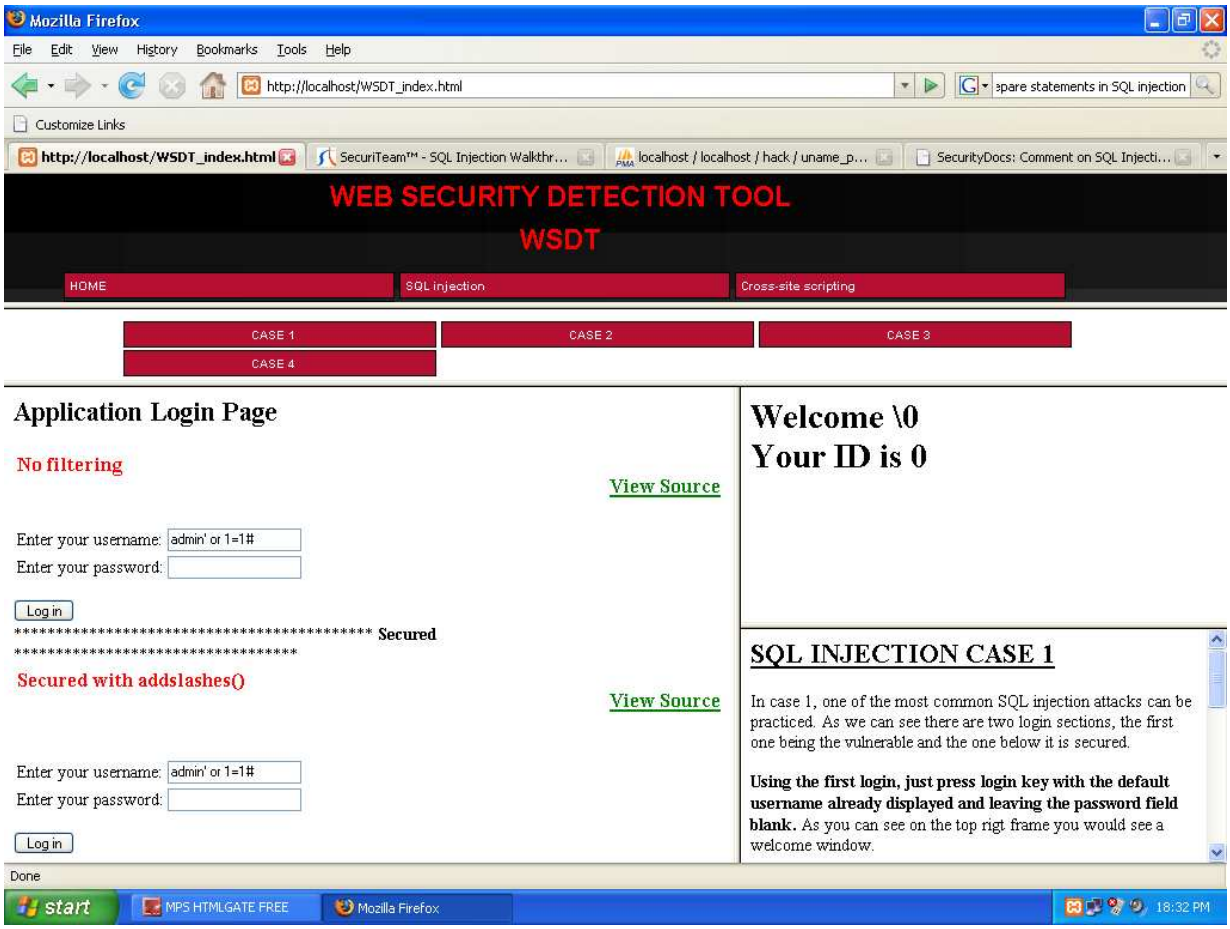

Figure 2.1: Screenshot of basic SQL injection attack

There are numerous ways in which SQL injection is performed. Each method varies in its syntax and the possible results. The most common methods in SQL injection are:

- Redirection and reshaping of the query.
- Error message based
- Blind injection

2.2 Cross-Site scripting (XSS)

Cross-site scripting sometimes is found to be very similar to SQL injection attacks. Though these attacks have similar attacking styles of injection, both are quite different from each other. SQL injection is a technique as mentioned before, which directly attacks the database of the application by using certain specially crafted queries. Cross-site scripting also performs similar code injection, but this attack is not performed to achieve access to the database and perform modification. XSS is used to attack other users of the web application by injecting malicious code in same, which is able to run in legitimate user's browser [5]. Thus, XSS attack is not against the web application, but it is against the application's users. An example of XSS attack with malicious JavaScript can be shown as follows:

```
<html>
<body>
<p> Hello friends, I am a student of Computer Science Department</p>
<script> MALICIOUS CODE </script>
<p> San Jose State University </p>
</body>
</html>
```

XSS attacks are categorized as:

- Non-persistent
- Persistent attacks.

Non-persistent attacks need the user to specifically visit the specially crafted link. These

attacks are most commonly found in search engines e.g. Google, Ebay, Amazon, etc. The persistent XSS attacks store the malicious script in the application itself and get activated whenever a particular web page is accessed by the user. The most common example of this attack is in forums: accepting user comments and displaying the same to other users. The following example shows a pop-up appearing every time the blogg is read by any user. This happens due to injection of a malicious script in the application database, which executes on the user browsers every time data is retrieved from it.

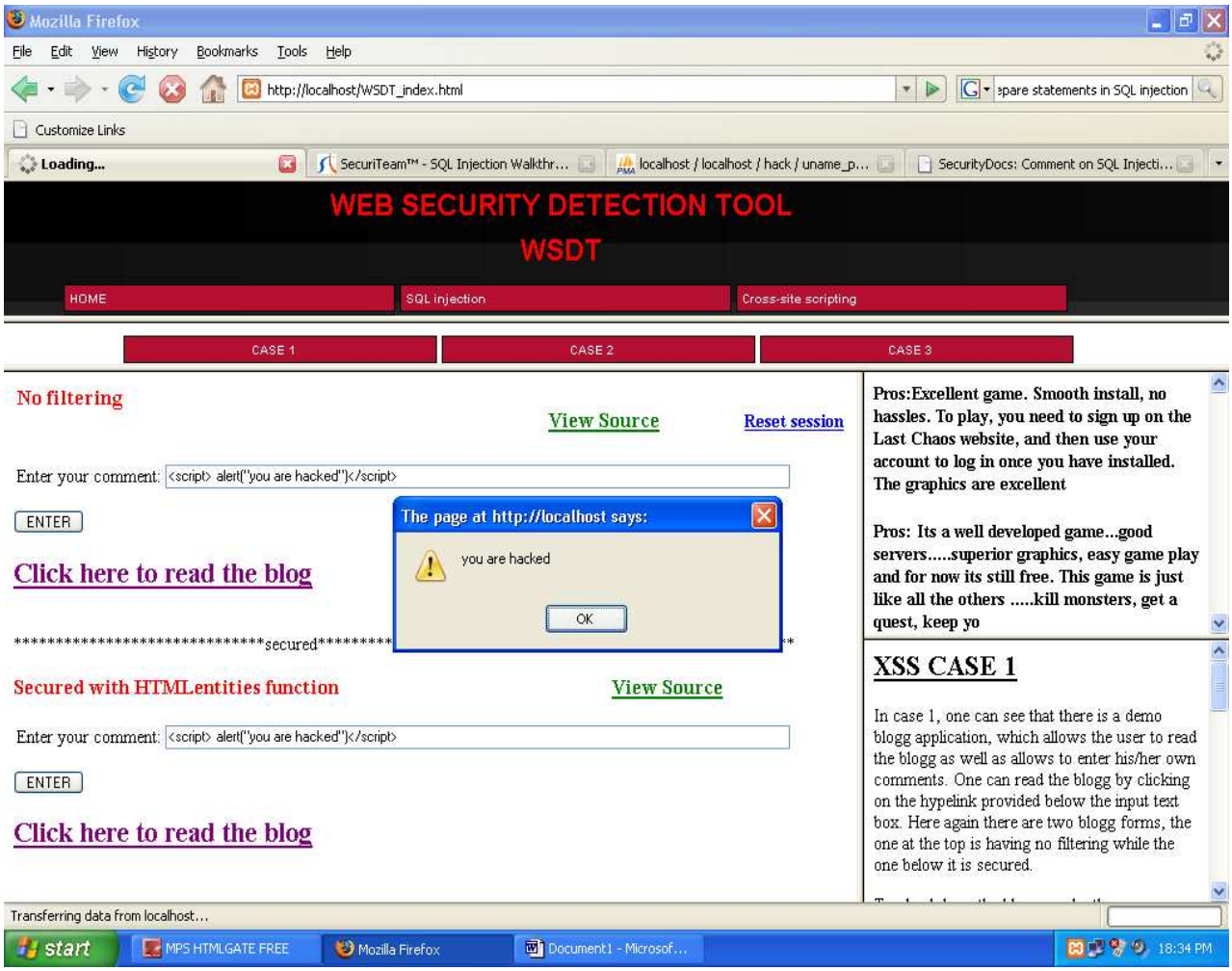

Figure 2.2: Screenshot of persistent XSS attack

2.3 Effects on the application

Effects of SQL injection can be fairly understood from the examples given in the previous sections. The attacker can get a full access to the database of a web application and might be able to insert, delete or even modify any of the original database entries. An SQL injection attack is widely known in the industry and even after its widespread use, some application tend to ignore their vulnerabilities and get exploited. SQL injection has the potential to affect all the four sides of security which include confidentiality, authentication, authorization, and integrity [2].

As mentioned before Cross-site scripting can be implemented in myriad ways. The attacker might use Java Script, PHP, or even HTML for code injection. Cross-site scripting attack can have different effects on the application and its user experience. XSS can be used by the attacker for varied purposes

An attacker can inject malicious scripts in the web application, which may display pictures, pop ups or even close the browsers window. Following is a list of commonly used XSS attacks:

- Users' browser might inadvertently execute the injected scripts form an attacker. This may display dynamically generated web pages created by the attacker which is not related to the legitimate website [5].

- Cross-site scripting attack may also be used by the attacker to connect a legitimate user of a website to a malevolent server.

- XSS also allows the attacker to take over legitimate user's session cookie before it expires.

- Cross-site scripting can also be used by the attacker to convince the user of an application to access a malicious URL. This URL might result in execution of the attacker scripts on the legitimate user's browser. This will allow the attacker to exploit the user privileges to access the application and issue SQL queries and exploit the underlying application [5].

Aforementioned are some of the most common attacks performed by the attacker using Cross-site scripting. It's not feasible to discuss every possible effect on the system due to such attacks, since these attacks can be performed in numerous forms each affecting a specific part of the application.

2.4 Solutions

While we develop web applications it's very important to sanitize all user input. If the inputs are sanitized it restricts the attacker from using certain special characters, which otherwise are not needed by legitimate users [3].

Though sanitizing user inputs helps to avoid attacks, it is not always feasible to do so. In case of entering an email address the above technique of sanitizing might be useful but not always. There are a few common ways in PHP which can be used for sanitizing user inputs.. One such very common methodology is using addslash(), which adds a backslash

before occurrence of any single or double quote. The magicquotes() is the functionality which appeared in the 4.X and higher versions of PHP. Though being similar to addslashes(); magicquotes() also provides security for other syntax like '\r', '\n', and EOF along with those covered under addslashes(). The following example shows how mysql_real_escape_strings() a functionality in PHP works:

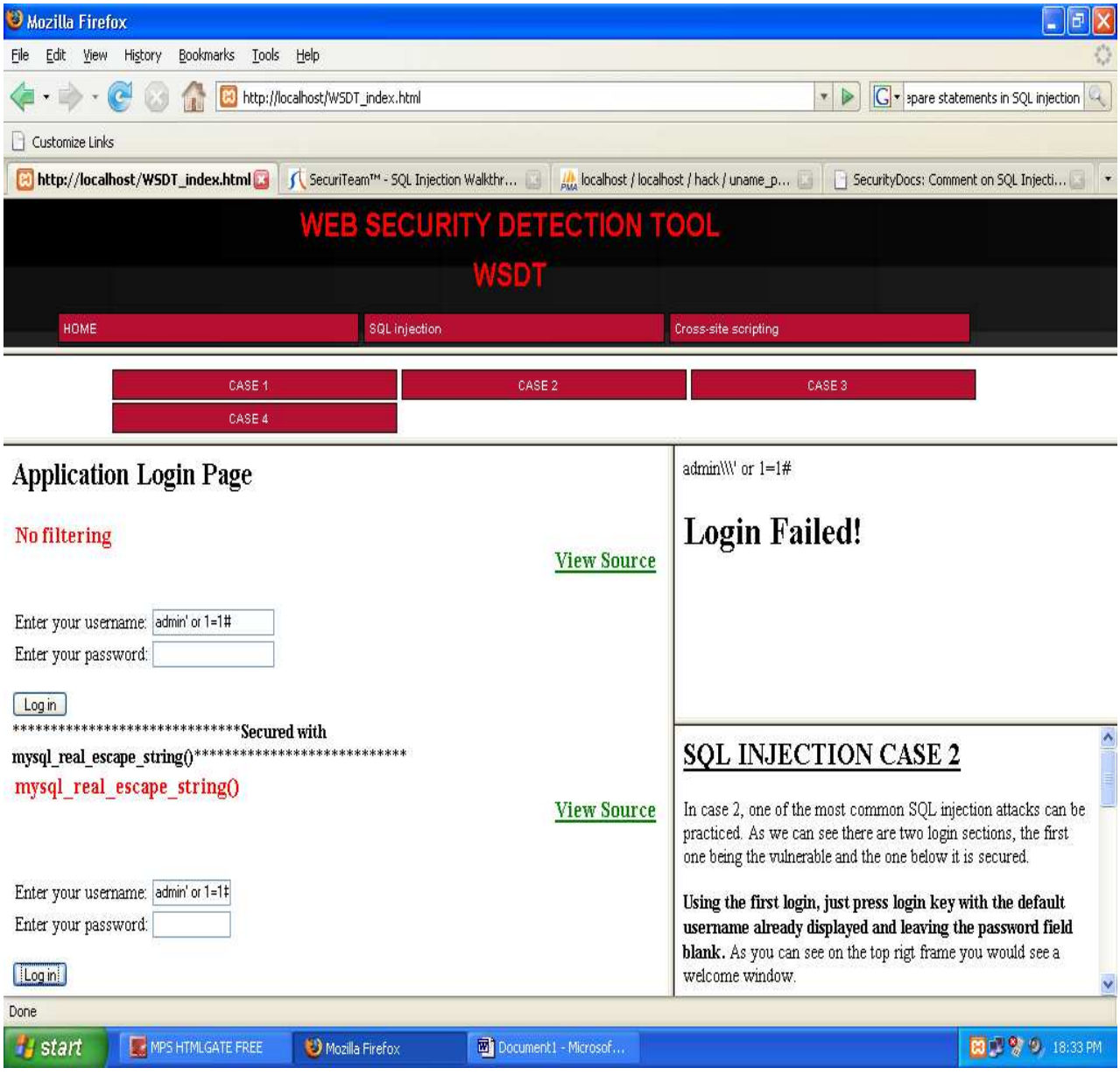

Figure 2.4: Application secured with mysql_real_escape_strings()

Here, in the above example consider the lower login application wherein, we have used the same SQL injection technique mentioned in section 2. When the application is secured with either addslashes or magicquotes, as you can see an input like:

USERNAME: admin' or 1=1 # doesn't give any form of access to the database. One can see in the right section of the above screenshot that an input mentioned above is debilitated by adding a backslash before the single quote. Even after we use these technique there is a possibility that the attacker might still get access since we are still considering the user inputs as SQL statements. This might cause a potential vulnerability.

There is a better way in which we can make our web applications more secured. We can use 'bound parameters' which are possible to be used in almost all databases. Bound parameters also called as the Prepare Statement allow us to create a SQL query with a place holder which is a question mark for each parameter [2]. In this way any inputs from user are not used directly to form a query behind the application. The prepare statement not only helps in making the application secure but also gives performance benefits. In case to mitigate XSS attacks one can use htmlentities(), which blocks any possible script tags to be executed over user's browser.

There are many open source soft-wares available which help to keep the code secure against SQL injection and XSS attacks. Few of such tools available are as follows:

- Nessus ➔ http://www.nessus.org/nessus/
- Nikto ➔ http://www.cirt.net/nikto2
- PHP Security Scanner ➔ http://securityscanner.lostfiles.de

The aforementioned tools can definitely be customized as per the user's requirements so as

to be used for personal use. These soft-wares come with documentation and are easily available over web. There are also commercial soft-wares available, which perform similar security measures.

3. Project design and components

3.1 Detection tools

As mentioned earlier there are myriad detection tools available which include many open source software. These detection tools though cannot guarantee that their use will debilitate the attacks completely, but they can definitely make the application more secured. These tools are categorized in two methodologies, the white box testing and black box testing.

3.2 White box testing

White box testing is a widely used testing tool which is used for detecting vulnerabilities in PHP scripts. This software allow the user to pass PHP files (scripts) to these software, wherein they are scanned for possible XSS or SQL injection vulnerabilities. Some of the most commonly used tools for white box testing are:

Nessus	http://www.nessus.org/nessus
Acunetics	http://www.acunetix.com/vulnerability-scanner/features.htm
PHP security	http://securityscanner.lostfiles.de/

All these tools have specific functions differing from each other; though have the same

concept of scanning the scripts for vulnerabilities. Our project uses one such similar tool called PIXY 3.02 (http://pixybox.seclab.tuwien.ac.at/pixy/).

3.2.1 PIXY

PIXY is a Java based vulnerability detection tool. This tool is very common to the other white box testing tools, as it accepts PHP scripts and scans them for possible vulnerabilities. PIXY understands that PHP script accepts data from the user using statements such as $_GET[], this data becomes harmful when used in other scripts or in SQL queries. It recognizes these evil lines of code as sensitive sinks. PIXY calls SQL injection and Cross Site Scripting vulnerabilities as tainted vulnerabilities and categorizes them weakly and strongly tainted.

Consider the following scenario when the PHP script takes user inputs and creates an SQL query using the user data achieved from $_GET statements.

```
$UNAME  = addslashes($_GET['username']);
$PWD = addslashes($_GET['password']);

mysql_query("SELECT * FROM some_table WHERE username=$UNAME AND password='$PWD'");
```

[12]

In the code snippet given above we take user inputs and create a SQL query. Now, as you can see the $UNAME is not embedded in (‘’) as in case of $PWD [12]. This opens up a loophole in the script which might get exploited by the attacker as:

SELECT * FROM some_table WHERE **username=1 or 1=1 AND password=''**

In this case even though we have used addslashes() function of PHP the script is open to vulnerabilities. PIXY considers such a scenario as potential exploitable loophole and calls it as **weakly tainted**. Strongly tainted are supposed to be harmful regardless of the structure of SQL query [12].

Pixy is a command line tool and is able to accept single PHP file as input and generates vulnerability results for it. Consider the following sample result from PIXY

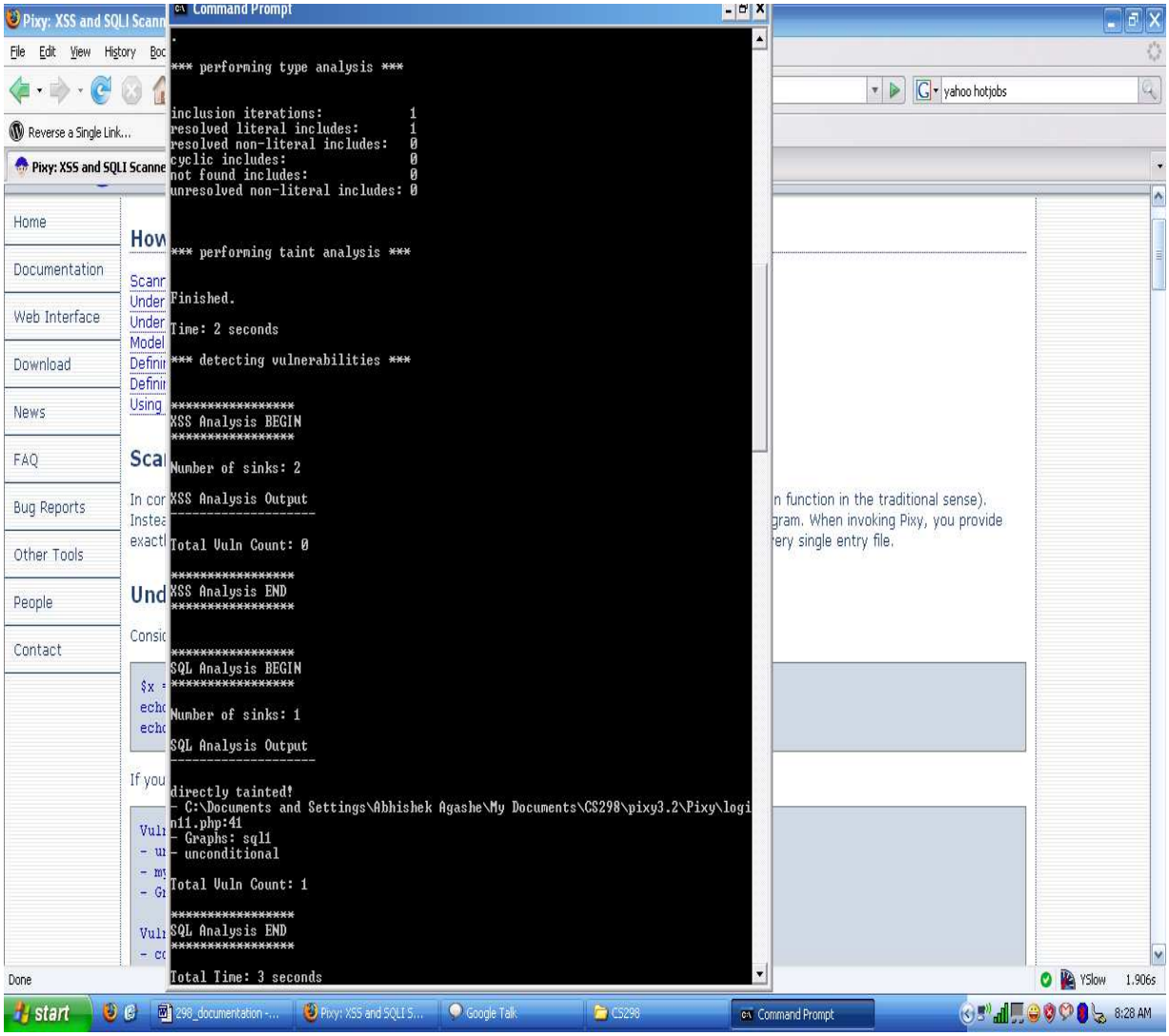

Figure3.2.1: Sample PIXY result

As we can see figure 1 depicts the results of PIXY analysis on a given PHP script which includes both SQLI injection as well as Cross Site Scripting analysis. In the above case there was no XSS threat found, whereas in SQL analysis PIXY detected a directly tainted vulnerability which is found to be at line 41 of login11.php script. Pixy at the end also displays the amount of time it took for the analysis, which in this case is 3 seconds.

Pixy as mentioned before is written completely in java and uses PERL script as a startup

script for calling the main java file and increasing the initial heap size.

3.2.2 Result analysis of PIXY

XSS analysis

As mentioned earlier PIXY categorizes the results as directly tainted and indirectly tainted.

Consider the following PHP code snippet:

```
1. $input = $_GET[‘input’];
2. echo $input;
3. echo $input1;
```
[12]

Now in this case Pixy might generate results as follows:

```
Vulnerability detected!
- unconditional
- x.php:2

Vulnerability detected!
- conditional on register_globals=on
- x.php:3
```
[12]

In this case as we can see Pixy detects two vulnerabilities in the php script. One being the unconditional, While second one is conditional. Here as we see the second

vulnerability depends conditionally on register_globals being set or not set, while first vulnerability is regardless of any specific option being set or unset.

SQL injection analysis

PIXY analyses SQL injection results on similar lines as XSS. Consider the following PHP code:

```
1:  $a = $_GET['a'];
2:  $b = addslashes($b);
3:  mysql_query("SELECT * FROM articles WHERE id = '1'");   // harmless
4:  mysql_query("SELECT * FROM articles WHERE id = '$a'");  // dangerous
5:  mysql_query("SELECT * FROM articles WHERE id = '$b'");  // harmless
6:  mysql_query("SELECT * FROM articles WHERE id = $b");    // dangerous
```

[12]

Given the above PHP script we get the following results from PIXY analysis:

```
directly tainted!
- myfile.php:4
- unconditional

indirectly tainted and dangerous!
- myfile.php:6
- unconditional
```

[12]

As we can see in this case again we use the $_GET and PHP function of addslashes() for sanitation. The two statements marked as dangerous are detected by PIXY as directly and indirectly tainted. In first case it's directly tainted since we are not at all sanitizing user inputs and directly creating an SQL query using the same. In second case we get an indirect tainted and dangerous warning. This is due to the fact that though we have used addslashes() for sanitizing user inputs, the query isn't framed correctly avoiding use of quotes for $b thus making a possible loophole to be exploited.

3.2.3 Comparing with other white box testing

White box testing is a very crucial component in building a secure web application. There are myriad white box testing tools available each with their pros and cons. Here is a list of some commonly used tools with their features and certain disadvantages.

Tools	Characteristics/ Features
Nessus	• Supports SQLI and XSS attack detection • Well documented • NO support for white box testing but not explicitly for PHP files. • Highly complex
Acunetics	• Supports SQLI and XSS attack detection. • Well documented. • Commercial software

PHP Security scanner	• Supports general PHP script vulnerable detection. • Support for scanning an entire directory. • Well documented. • Depends on database for tests and results.

3.3 Black-box testing

Black box testing is a very common tool found which can scan websites for possible vulnerabilities. Usually these tools have web crawlers or spiders in them that scan the website and all possible sub-links trying to detect vulnerabilities on the application running on server. Following are a few commonly used black box testing tools:

NIKTO 2	http://www.cirt.net/nikto2
Kayra	http://sourceforge.net/projects/kayra/
Space Monkey	http://sourceforge.net/projects/spacemonkey/
Agares Security	http://sourceforge.net/projects/agaressecurity/

All the aforementioned tools have spiders build in them to scan through entire websites performing an automated attack to detect possible attacks which might be successful. Some of these tools have databases to store in possible attack cases and some don't. Yet, they all

perform injection by injecting test cases through application forms.

3.3.1 Wapiti

Wapiti is a black box testing tool that scans through web pages in a web site using a web-spider. Wapiti then starts injecting payloads through forms accepting user inputs and detects if the script is vulnerable to SQL injection or Cross Site Scripting [3].

Wapiti can be used to detect multiple errors like:

- Database injection
- Cross site scripting
- HTTP response splitting
- HTTP 500 error
- Command execution detection

3.3.2 Result analysis of Wapiti

Wapiti determines XSS as either permanent or non-permanent. A non-permanent XSS also known as temporary XSS includes scripts which get executed only when the script code is used by the crafted user query. In other words the temporary XSS attack doesn't stay over the server and is returned immediately. The root cause of permanent XSS is un-sanitized user inputs which gets stored in the application's database.

Thus each time some one access this web application the code segment in the database gets executed in the user's browsers performing XSS.

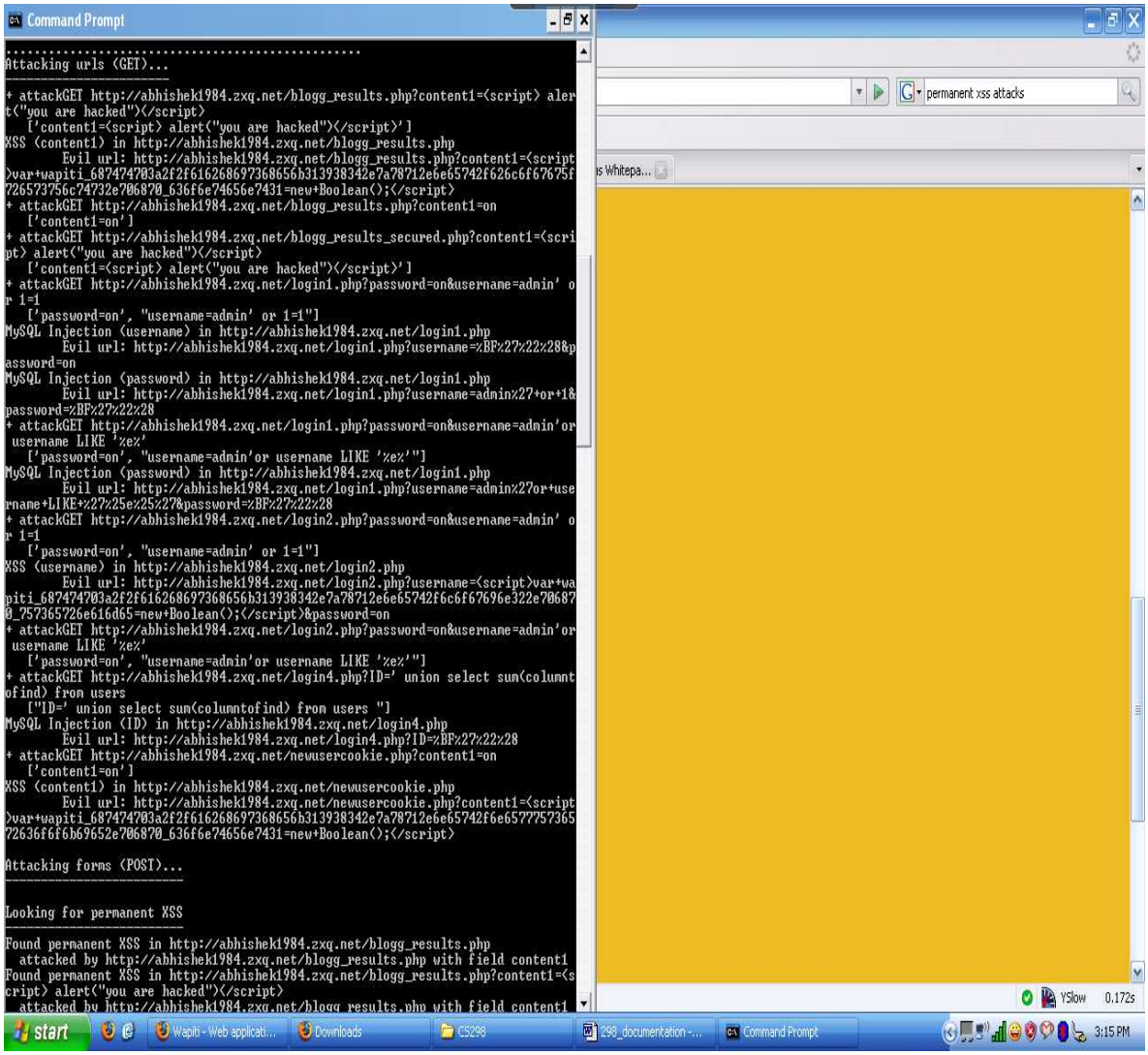

Figure 3.3.2 Wapiti sample output

Wapiti uses library called “lswww” which is a web spider. This requires html tidy (http://tidy.sourceforge.net/) since it accepts only well formatted html web pages.

Wapiti 1.1.5 provides options like verbose, timeout options, starting URL specifications, etc.

3.3.3. Comparison with other black-box tools

As mentioned before there are many black-box testing tools available both in open source and commercial domain. Most of these tools are efficient and widely used for testing purposes. The following chart gives an overview of few widely used black box testing tools along with Wapiti.

Nikto 2	• Open Source • Web Server scanning tool • Detects XSS and 404 errors • Provides and Intrusion Detection System • Username guessing plugin • Generates XML reports • Relies on database for test cases • Doesn't detect SQL injection explicitly
Kayra	• Open source • Detects XSS and SQL injection attacks • Provides GUI • No documentation • Doesn't detect permanent XSS • Doesn't detect server errors
Space Monkey	• Detects SQL injection and XSS attacks • Written in C/C++ • Platform dependent. • No documentation. • Doesn't detect permanent XSS • Not suitable for testing purposes.

Witko	• Witco detects XSS as well as SQI • Has many enhance features. • Well documented. • Written on non open source (.NET Framework) • Difficult to merge with other tools

4. Implementation

4.1 Overview

As mentioned above both; the black box testing as well as white box testing is used widely in detecting vulnerabilities in web applications. All these applications either provide a scanning for external web application or accept user's scripts and scan them for vulnerability. Both of these techniques have their pros and cons. White box testing gets complete access to server side files and can pin point the location of the vulnerability. Whereas, in case of black box, one can monitor possible loopholes in an application which is already deployed and how well it can sustain attacks.

In my project I am trying to combine these techniques to give user functionality of both methodologies in a single framework. This project uses two open source software:

- PIXY 3.2 (White box testing)
- Wapiti (Black box testing)

This project uses a JAVA wrapper code to combine these techniques and give a presentable result to the user.

4.1 Architecture:

The project uses JAVA as its base language to combine both testing techniques and small amounts of PERL is used for automation.

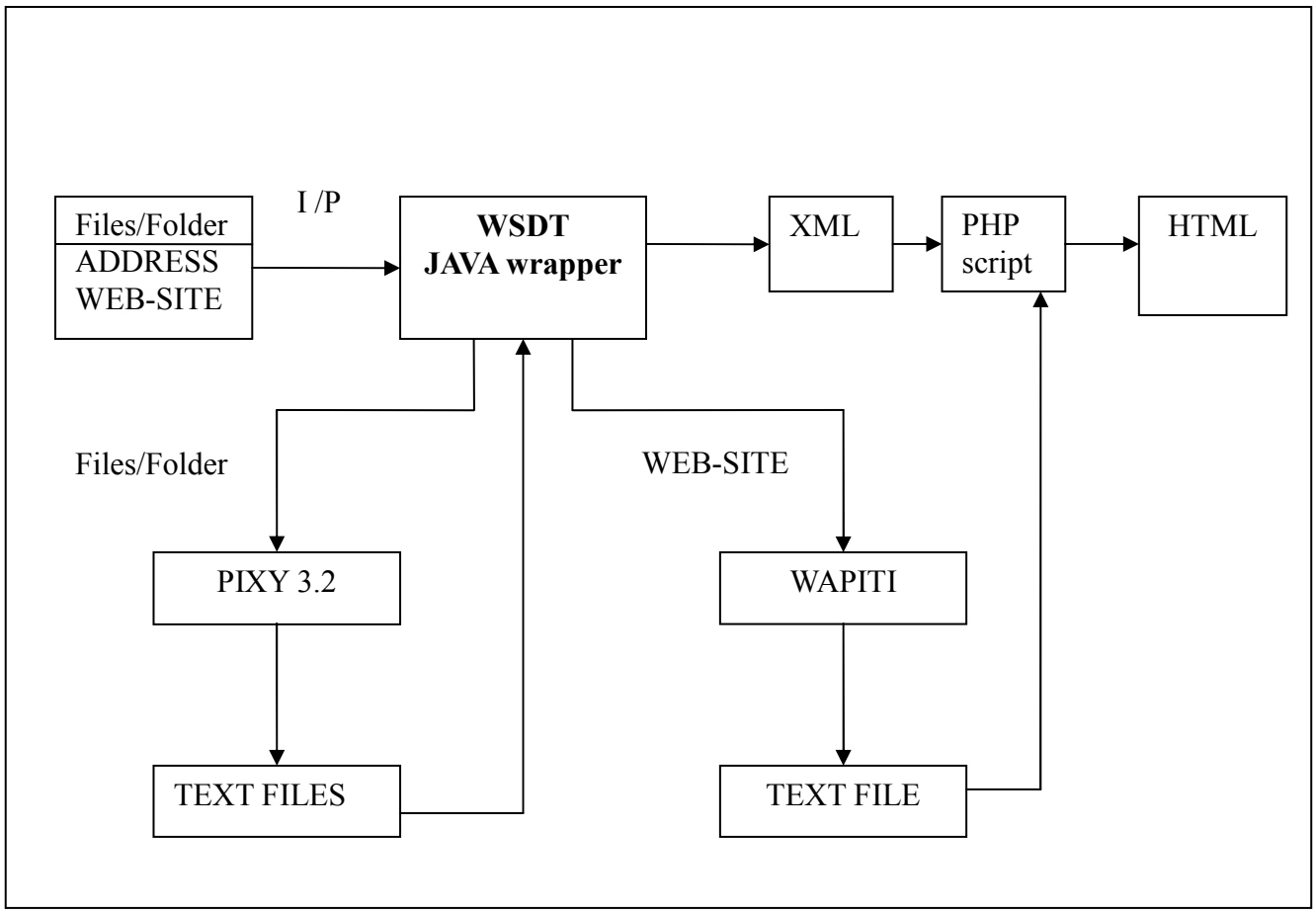

Figure 4.1: Architecture of Web Security Detection Tool

Figure 3 describes the architecture of the detection tool. As shown in figure 3 the main wrapper is built in JAVA, which is responsible for combining both the open source software. The Java wrapper includes enhancements for the open source and is responsible for processing the output generated by PIXY and Wapiti.

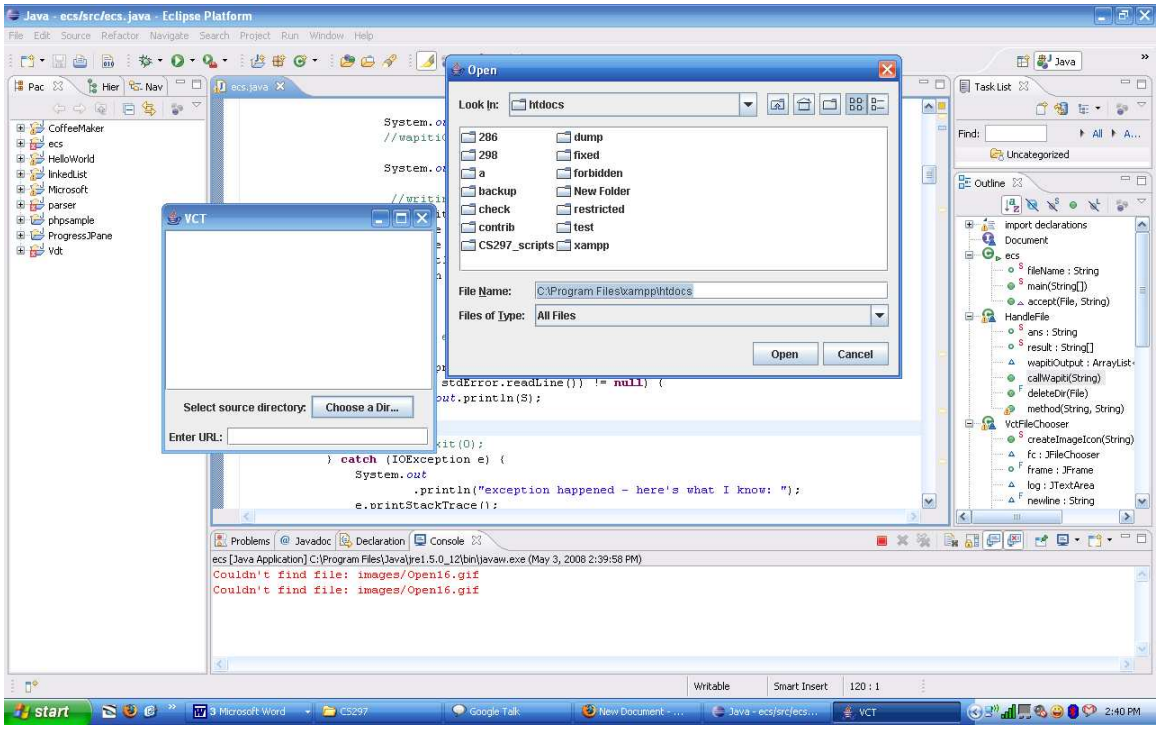

Figure 4.2: GUI to accept user inputs

Figure 4.2 depicts that WSDT (wrapper) gets two inputs from the user namely:

- Files/Folders
- Address of deployed web application

These as shown are passed to the open source tools. Once output is generated in XML and text files format, it's further processed by PHP scripts generating output in form of HTML page.

This architecture gives following enhancements over the contemporary software:

- PIXY as a standalone tool take a single PHP file as input. Our system now takes an entire folder and performs a search to find PHP files in the same.
- Both the aforementioned tools are based on command line. The tool developed gives user a GUI interface.

- Both the open source tools i.e. Wapiti and PIXY give unorganized results about the detected vulnerability on the command line. This result can get complex while dealing with huge applications. The proposed system converts the output generated by the open source software in XML format thus, making it easier to manipulate. Later the final results are presented to the user in HTML format.
- Pixy gives the vulnerability only at the point where the sink ends. Our system uses the Wapiti output to detect the point where the user inputs are accepted by the PHP scripts. Our system also detects possible permanent XSS found in the submitted PHP files.

4.2 Detection Process

WSDT accepts an entire folder consisting of PHP files and starts scanning the folder for all possible PHP files, which in turn are passed to PIXY. PIXY as mentioned before is the white box testing tool that accepts PHP files and scans them for possible XSS and SQL injection vulnerabilities. The result generated by PIXY is stored in form of text file which is different for each PHP file scanned. For simplicity the names of the PHP files are retained and the output text file of each PHP script is given the same name.

e.g. login.php ➔ WSDT➔ login.txt

A sample text output file would look as follows:

****

XSS Analysis BEGIN

****

Vulnerability detected!

- unconditional

- C:\Program Files\xampp\htdocs\blogg_results.php:72

Total Vuln Count: 1

****

XSS Analysis END

****

****

SQL Analysis BEGIN

****

Number of sinks: 2

SQL Analysis Output

---

directly tainted!

- C:\Program Files\xampp\htdocs\blogg_results.php:43

- unconditional

Total Vuln Count: 1

****

SQL Analysis END

****

While PIXY processes the PHP files for vulnerabilities the second user input which is web-

site address of the deployed application is passed by WSDT to Wapiti. The black box testing tool (Wapiti) starts crawling through the web pages and tries to inject payloads and detects any possible evil (vulnerable) links. Again this output generated by wapiti is stored in text files for further processing.

Once PIXY is done processing all the text files these text files are given as input back to WSDT(wrapper) for further processing. Now that WSDT has the entire output of all the files scanned by PIXY, it starts scanning these output files. As per the output generated for each file the files gets categorized as XSS vulnerable, SQL injection vulnerable or both. This entire categorization along with other details of each of the PHP files is stored in an XML file. The entire output is stored in XML so that it becomes easier to further process this information in systematic pattern.

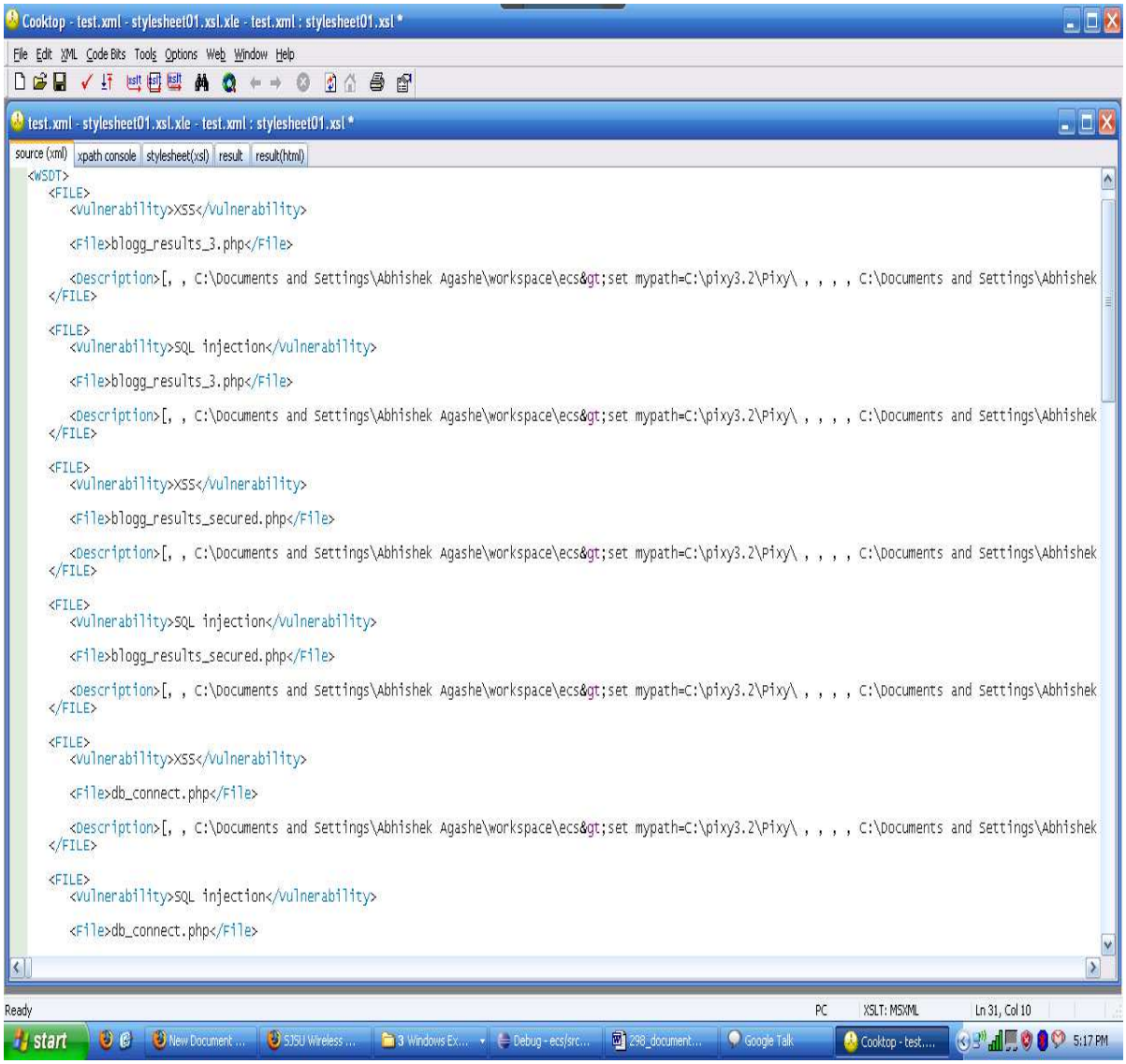

Figure.4.3: XML file generated by the wrapper

As shown in figure 4, the wrapper processes the PIXY's output files in XML form. Each of the "FILE" element shown above includes the output generated for the PHP files scanned by PIXY. As depicted above, the FILE element has basically three sub-elements embedded in it namely: Vulnerability, File, and description.

Now, here the Vulnerability element tells us the type of vulnerability detected in that particular PHP file. The File element gives us the name of that file and description element

has all the detailed result of that particular PHP file generated by PIXY.

4.3 Output format

Once we have all the data that we need in the XML form now we can present the results to the user in a better format. The output text file generated from Wapiti and the aforementioned XML file forms the basic raw data which is processed further. This project also includes a PHP script which is used for presenting the data generated from WSDT. This PHP script parses the XML data and according to the Vulnerabilities element generated for each of the files, it enters the PHP filenames given by the user in a table form. This helps in determining which files have either or both of the vulnerabilities.

This tabular form can be shown as follows:

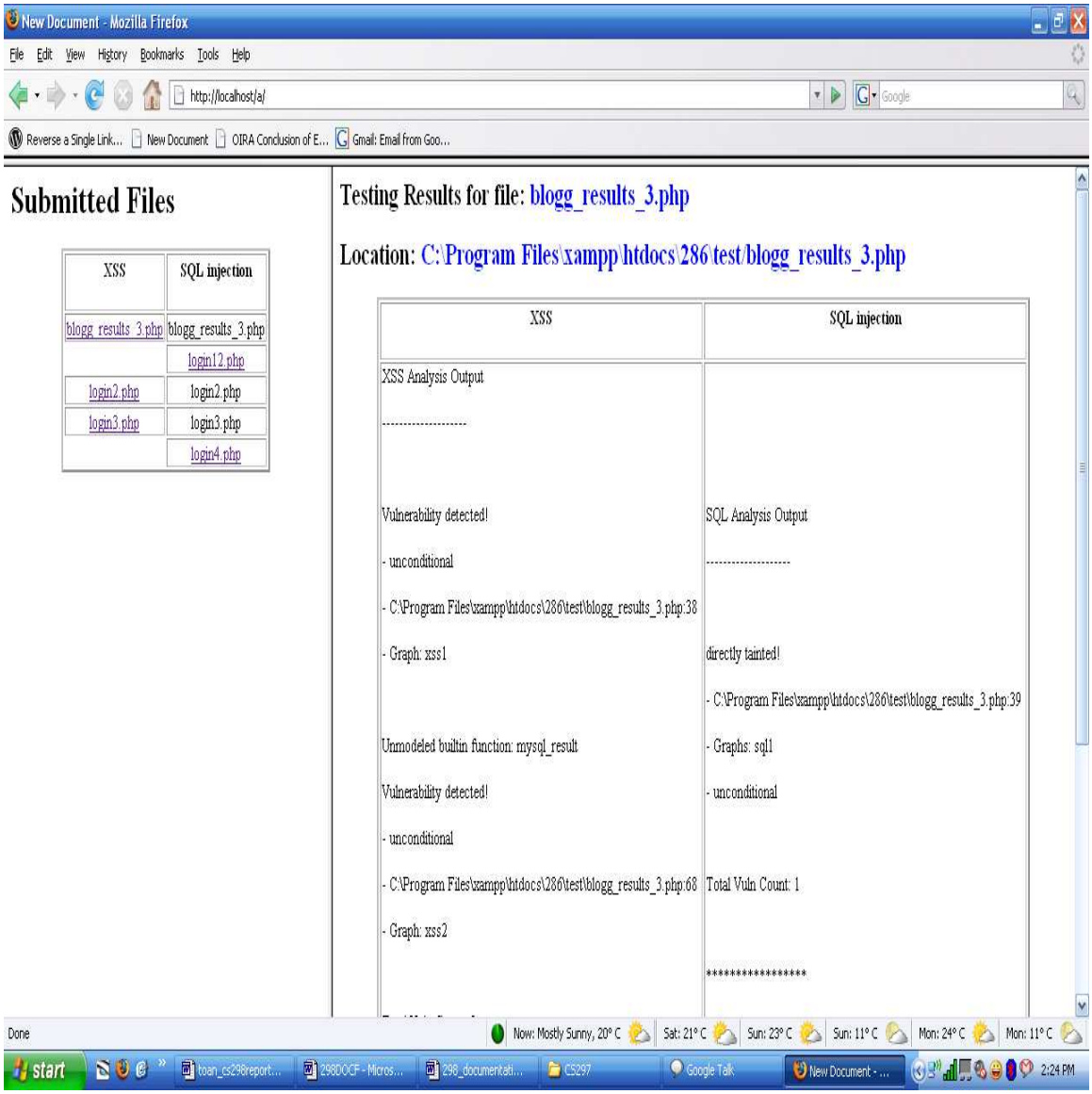

Figure 4.3 PHP files categorized as per vulnerabilities

Figure 5 shows the tabular form in which the files are categorized as per the vulnerabilities detected. The user would be able to see the analysis details by clicking on the particular PHP file in the table. While this depicts the PIXY results, the Wapiti results will also be presented to the user on the same page. This would give information on the possible vulnerable variable in the scripts along with detection of permanent XSS. For instance,

consider an HTML form accepts user's comments and stores it in database. Now, this user data is passed as a parameter to the PHP scripts using either GET, POST or REQUEST methodology. This is the actual point where the vulnerability actually entered the script. Using Wapiti we are able to detect this vulnerable parameter and find the point where the user input can be sanitized.

5.0 Results and Analysis:

5.1 Results

This project has two sections one being the web hoisted test application, which basically acts as a tutorial for understanding SQL injection and Cross-site scripting. The web application is hoisted at http://abhishek1984.zxq.net . This application has been tried and tested by many of my friends and also have been utilized by other students in their academic projects. The second section of my project is basically the detection tool which is developed in Java and PHP to combine the black-box and white box testing techniques. The results of this tool are generated by parsing through the user's PHP scripts located locally on his/her machine, at the same time crawling the live application deployed on the server. The results generated by the tool include a table categorizing all the submitted files to the tool into SQL injection and cross-site scripting respectively.

Each of these files when clicked gives a combined result of PIXY and wapiti in the right frame displaying the possible vulnerable line found in the particular script with respect to SQLI or XSS. It also gives information about the vulnerable variable which might have been used by the script to accept user input with sanitizati

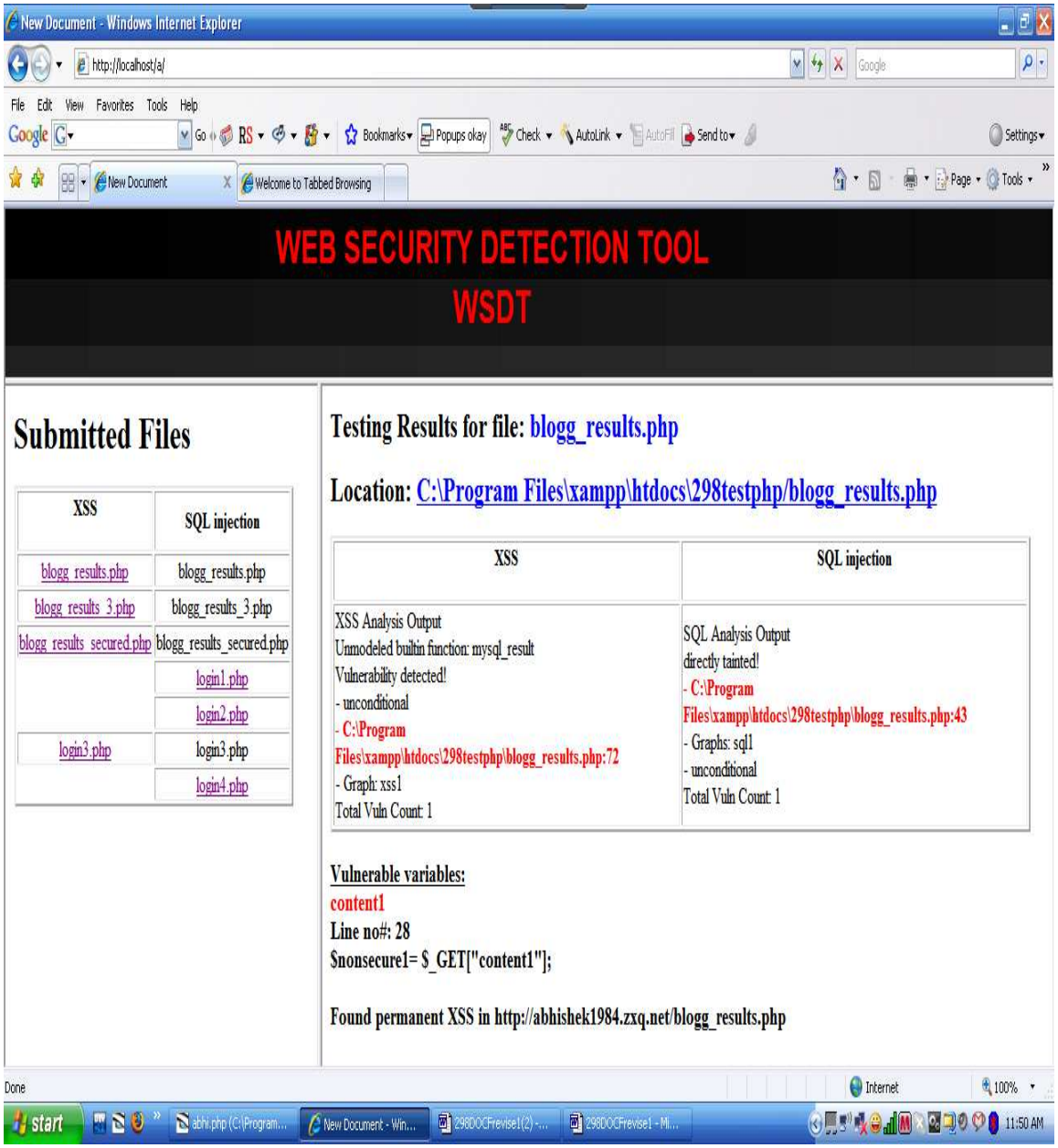

Figure 5.1 Results generated by the tool.

These results would be useful for the application developer to better pinpoint the vulnerability in his/her scripts. Since, we now get information on the point of entry where the vulnerability first occurred as well as the point to which it sank in script it would help to better mitigate the possible attacks. This tool also indicates the user if the submitted script is

vulnerable to persistent cross-site scripting vulnerability. This would also help user to get an indication that his/her database might be uploaded with malicious scripts performing XSS attacks. Persistent XSS indicates that some kind of script like JavaScript can be injected in his/her database and used for attacking others by its execution on legitimate user's browser.

5.2 Testing and Analysis

To test this system, we used a pool of PHP files and various other websites to determine its efficiency. Since this detection tool works on scripting language like PHP. It was found that many a times PIXY as well as Wapiti tends to overlook certain vulnerabilities in code. The major reason for this is the flexibility of scripting languages which allows a single implementation in numerous ways. This tool gives better efficiency for a well formatted HTML web pages and PHP scripts.

While testing the output generated by the tool it's found that using the start and sink point suggested by the tool it becomes easier to track down possible vulnerabilities in the scripts. Though, this tool might not be able to pin point the exact and all the locations which might have vulnerabilities, while testing it was found that it definitely gives better results than using the two testing techniques individually. For testing this tool the aforementioned tutorial website http://abhishek1984.zxq.net was used extensively along with matching PHP scripts available locally. We found that wapiti being a black box testing tools many a times overlooks possible vulnerabilities in web application. Since, it uses a fuzzer technique and needs well formed HTML pages it tends to overlook certain vulnerabilities. We found while

testing that a very efficient fuzzer independent of well formed HTML page would have been helpful as most of the PHP scripts developed do not generate well formed HTML pages.

6.0 Conclusion and Enhancements

6.1 Conclusion

As mentioned before most of the open source tools provide either the black box testing or the white box testing. This tool would give benefits of both the methodologies under a single roof. Many a time's vulnerabilities cannot be detected just by scanning the PHP scripts. Using black box testing we are able to test an already deployed website. This is useful in projects where the website updates is an ongoing process.

White Box testing helps developer to know the actual attacks which might be successful on the already deployed website. The White box testing would allow the developer to pin point the vulnerability in the PHP scripts. This would help the developer have complete control on the security loopholes present or one which, might be found on his/ her website. In case of PIXY, it gives only the second and final level sink points of the user inputs. It's vital that user inputs should be sanitized before any further operation Pixy sometimes gives false alarms as it has access only to source files. Wapiti output can be used to reduce these false alarms.

The black-box and white-box techniques can be used to complement each others results and mitigate aforementioned vulnerability. Though this tool might not be very useful for unformatted HTML pages and PHP scripts, it can definitely have added advantage over using both the tools used independently.

6.2 Future work

While discussing about this project with my other peers and friends I came across some added features which might be developed in future.

- Updating the test cases is a matter of concern in Vulnerability detection tool. There are options of maintaining a database for test cases, but it adds to the overhead for processing multiple files.

- As of now this system accepts only PHP files, but there are many other server side languages like JSP and PERL which are used extensively and should be incorporated for detection.

- The system could give possible PHP snippet which could mitigate the vulnerability in developer's script.

- Vulnerability correction tool would be definitely be useful and feasible if it would be possible to create a regular expression, which could be compatible with the flexibility of scripting languages like PHP and PERL

7. References

[1] Engin Kirda, Cristopher Kruegel, Giovanni Vigna, Nenad Jovanovic, "*Noxes: a client-side solution for mitigating cross-site scripting attacks*", 2006, Pages: 330 – 337, Proceedings of the 2006 ACM symposium on Applied computing, ACM Press.

[2] Wei K, Muthuprasanna M, Suraj Kothar "*Preventing SQL injection attacks in stored procedures*" 18-21 April 2006 Pages: 8 pp. Software Engineering Conference, 2006. Australian, IEEE.

[3] Muthuprasanna M, Ke Wei, Kothari S, "*Eliminating SQL Injection Attacks - A Transparent Defense Mechanism*" Sept. 2006, Pages: 22 -32 ,Web Site Evolution, 2006. WSE '06, Eighth IEEE International Symposium.

[4] Rubin, A.D. Geer, D.E., Jr. ," *A Survey of Web Security*", Sept. 1998 Volume: 31 , Issue: 9, Pages: 34 – 41,computer, ISSN: 0018-9162 IEEE computer society.

[5] Benjamin Livshits, Úlfar Erlingsson, "*Using web application construction frameworks to protect against code injection attacks*", 2007, Pages: 95 – 104, Proceedings of the 2007 workshop on Programming languages and analysis for security, ACM press.

[6] Shanmugam,J. Ponnavaikko, M.," *A solution to block Cross Site Scripting Vulnerabilities based on Service Oriented Architecture*", 11-13 July 2007, Pages: 861-866, Computer and Information Science, 2007, ICIS 2007.6th IEE/ACIS international conference.

[7] Di Lucca, G. Fasolino, A. Mastoianni, M. Tramontana, P.," *Identifying Cross Site Scripting Vulnerabilities in Web Applications*", 2004,Pages: 71-80, Proceedings of the Web Site Evolution, Sixth IEEE International Workshop.

[8] Buehrer, G. Weide, B. Sivilotti, P.," *Using parse tree validation to prevent SQL injection attacks"*, 2005, Pages: 106-113, Proceedings of the 5th international workshop on Software engineering and middleware.

[9]SQL injection.(2007).Retrieved May/09, 2008, from http://www.owasp.prg/index.php/SQL_injection

[10] Cross-site scripting. (2007). Retrieved May/09, 2008, from http://www.owasp.org/index.php/Cross-site-scripting

[11] Wapti. (2006). Retrieved May/09, 2008, from http://wapiti.sourceforge.net/

[12] PIXY. (2007). Retrieved May/09, 2008, from http://pixybox.seclab.tuwien.ac.at/pixy/documentation.php

[13] Government computer news. (2008). Retrieved May/08, 2008, from http://www.gcn.com/