Vulnerability Detection of php applications using Php-Code Analyzer before deploying web application

Sandeep Guntreddy[1] Santosh Naidu P[2]
[2]Information Security Consultant
[1,2]Department of Computer Science and Engineering
[1]Vizag Institute of Technology [2]CyberQ Pvt. Ltd

Abstract— A Vulnerability Scanner is software application that assesses security vulnerabilities web sites and web applications and produces appropriate results after the scan. However, because both admins/developers and attackers can use the same tool for fixing or exploiting a system, admins/developers need to run a scan and fix problems before an attacker can do the same scan and exploit any vulnerability found. Php Code Analyzer can help you find and validate SQL Injection, Cross-Site Scripting (XSS), unintentionally disclosed sensitive information, and other vulnerabilities of the web applications that have located in local server before the final deployment in main server or in the web. It is written in php; application based, and can be deployed in Linux and Windows.

Key words: Php Code Analyzer, Cross Site Scripting, SQL Injection, LDAP Injection, X-Path Injection, File Disclosure, File Inclusion, Protocol Injection

I. INTRODUCTION

Php-code analyzer tests the web applications for common security problems such as code execution, command execution, file disclosure, fle inclusion, file manipulation, LDAP Injection, Php Object Injection, Protocol Injection, Reflection Injection, SQL Injection, XPath Injection, Cross-site scripting, HTTP response splitting, Session Fixtion and remote command execution vulnerabilities.

This tool crawls a web application and locates application layer vulnerabilities and weaknesses, by modifying or inspecting HTTP messages for suspicious attributes.

A large number of web application scanning tools are available both in commercial and open source.

Php-code analyzer is used to evaluate web application scanners on their ability to effectively test web applications and identify vulnerabilities. It covers areas such as crawling, parsing, session handling, testing, and reporting.

The goal of the Php-code analyzer is to report vulnerabilities to help guide web application security professionals during scanning of web applications.

II. RELATED WORK

In 2013, (8) XIAOWEI LI and YUAN XUE explained not only about different security aspects in web applications by systemizing the existing techniques which might be used for further research but also about input validation vulnerabilities, session management vulnerabilities and application logic vulnerabilities.

In 2011, (5) Francisco José Marques Vieira designed an architecture for vulnerability injection tool which allows the intromission of vulnerabilities in a program/script and is an extensible one which supports addition of new vulnerabilities to inject.

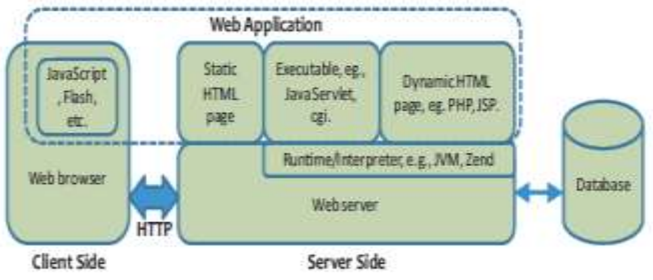


Fig. 1: Overview of web application

In 2013, (6) Jamang Jayeshbha Bhalabha, Amit Doegar and Poonam Saini have done some extensions to RIPS and proposed a new one RIPS plus modification for injection tool and also exploited some vulnerabilities in the latest versions of well-known PHP applications.

In 2012, (7) Francois Gauthier and Ettore Merlo designed a tool named ACMA (Access Control Model Analyzer) which detects access control vulnerabilities in PHP applications and uses a lightweight model checker to detect the privileges of the application.

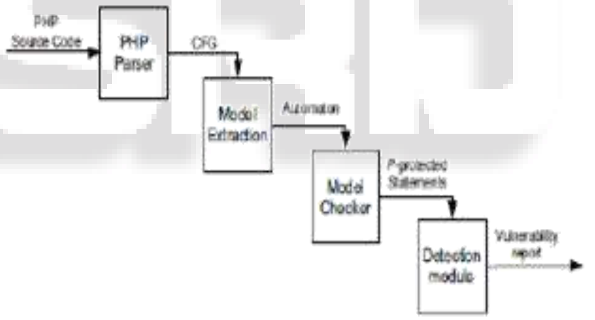


Fig. 2: ACMA Architecture

In 2007,(8) Ettore Merlo, Dominic Letarte and Giuliano Antoniol explained about the evolution of security related vulnerabilities detected by disseminating and combining CFG (Control Flow Graph) along with security level DB accesses w.r.t SQL Injection attacks.

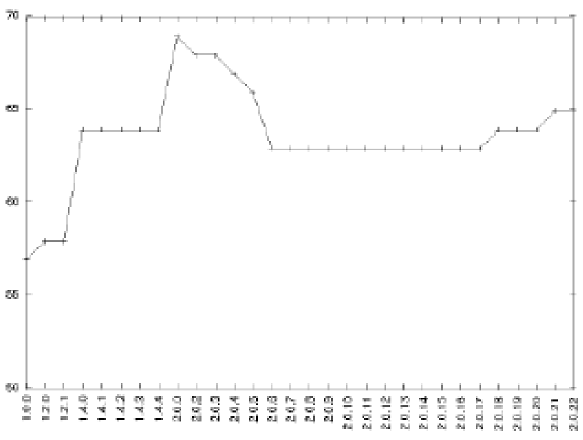


Fig. 3: Percentage of vulnerable DB accesses

In 2012, (9) Maureen Doyle and James Walden explained about the evolution of vulnerabilities in PHP web applications and also calculating vulnerability densities.

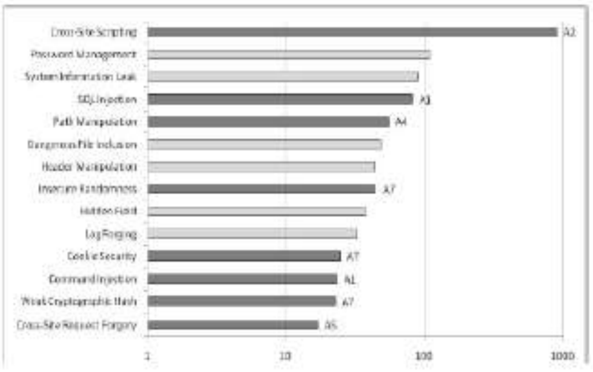


Fig. 4: Aggregate Vulnerabilities by type

III. Features

A. *Vulnerabilities*

1) *Server Side*

a) Code Execution

This vulnerability detects an HTTP request that attempts to alter PHP's command line parameters. Vulnerable installations of PHP on Apache using mod_cgi will obey these parameters, allowing the attacker to DoS the server or execute remote code.

b) Command Execution

An attacker can run arbitrary commands on the web server by executing library files and overwriting script variables that aren't properly initialized. It explores the process of “expanding” an exploit by leveraging it to attack backend applications, plant Trojan or backdoor code, or gather reconnaissance data.

c) Header Injection

This vulnerability allows a malicious user to control the remaining headers and body of the response the application intends to send, but also allow them to create additional responses entirely under their control.

d) File Disclosure

This vulnerability allow attackers to read arbitrary files present server-side file system and also outside the web root.

e) File Inclusion

File inclusion vulnerability allows an attacker to access unauthorized or sensitive files available on the web server or to execute malicious files on the web server by making use of the ‘include’ utility. This type of vulnerability arises mainly due to bad input validation mechanism; wherein the user’s input is passed to the file include commands without proper validation. This impacts to malicious code execution on the server or reveal data present in sensitive files, etc. g. File Manipulation.

f) LDAP Injection

LDAP injection is the technique which exploits web applications that use client-supplied data in LDAP statements without first stripping potentially harmful characters from the request.

g) SQL Injection

A SQL Injection attack is a form of attack that comes from user input that has not been checked to see whether it is valid or not. The objective is to exploit the database system into running malicious code that will reveal sensitive information or otherwise compromise the server.

h) Un-Serialize with POP

An attacker can exploit this issue to inject and execute arbitrary malicious PHP code in the context of the application which have been affected. This may facilitate in compromise the application and the underlying system; other attacks are also possible.

i) X-Path Injection

XPath (XML Path Language) injection is an attack technique similar to SQL injection that can be used to exploit Web applications using an XML database that construct run-time XPath queries from user-supplied input.

2) *Client Side*

a) Cross-Site Scripting

Client side XSS allows attackers to inject own script code to the application-side of the vulnerable online-service module. To protect the user’s environment from malicious JavaScript code, browsers use a different mechanism named as sand-box that limits a script to access only resources associated with its origin site.

b) HTTP Response Splitting

HTTP response splitting is a Web application input validation vulnerability that allows to exploit the HTTP headers of a Web application for initiating attacks leading to cross-site scripting (XSS), user/page hijacking, modification of cookies, website spoofing etc. The attacker initiates attack through the injection of a sequence of hex-coded Carriage-Return (CR) and Line-Feed (LF) characters in the HTTP header and then appending it with malicious HTTP Set-Cookie headers crafted to force the server to process and break the requests into two individual responses

c) Session Fixation

In a this type of attack, the attacker able to fix the user’s session ID before the user even logs into the target server, thereby eliminating the need to obtain the user’s session ID afterwards.

B. *Code Audit Interface*

– scan and vulnerability statistics
– grouped vulnerable code lines (bottom up or top down)
– vulnerability description with example code, PoC, patch
– exploit creator
– file list and graph (connected by includes)
– function list and graph (connected by calls)
– user input list (application parameters)
– source code viewer with highlighting
– active jumping between function calls
– search through code by regular expression
– 8 syntax highlighting designs

C. *Static Code Analysis*

– fast
– tokenizing with PHP tokenizer extension
– taint analysis for 232 sensitive sinks
– inter- and intra-procedural analysis
– handles very PHP-specific behavior
– handles user-defined securing
– reconstruct file inclusions

- detect blind/non-blind exploitation
- detect backdoors
- 5 verbosity levels
- over 100 test-cases

The entire proposed modeling and architecture of the current research paper should be presented in this section. This section gives the original contribution of the authors. This section should be written in Times New Roman font with size 10. Accepted manuscripts should be written by following this template. Once the manuscript is accepted authors should transfer the copyright form to the journal editorial office. Authors should write their manuscripts without any mistakes especially spelling and grammar.

IV. ABOUT PHP-CODE ANALYZER

Php-code analyzer is a tool written in PHP to find vulnerabilities in PHP applications using static code analysis.

By tokenizing and parsing all the headspring code files, php code analyzer is have the capacity to transform PHP source code into a program model and to detect potentially vulnerable functions that can be tainted by user input (influenced by a malicious user) during the program execution/compilation. Php-code analyzer offers not only structured output of found vulnerabilities but also it offers an integrated code audit framework for further manual analysis.

V. IMPLEMENTATION AND DEPLOYMENT OF PHP-CODE ANALYZER

The whole code is written in php by taking help of some third party software as mentioned below.

1) PHP Crawl (http://phpcrawl.cuab.de/) [1]
   Function: its main function is to search a website to identify all URL's belonging to that site.
2) PHP HTTP Protocol Client (http://www.phpclasses.org/package/3-PHP-HTTP-client-to-access-Web-site-pages.html) [2]
   Function: It provides the functionality of using HTTP protocol in php code.
3) PHP Simple DOM Parser (http://simplehtmldom.sourceforge.net/) [3]
   Function: It is one of the libraries which is useful for parsing Document Object Models such as HTML content in php
4) jQuery (http://jquery.com/) [4]
   Function: It is one of the java script's libraries which provides Java Scripts and Ajax functions.

Software requirements for PVRS are defined as follows:

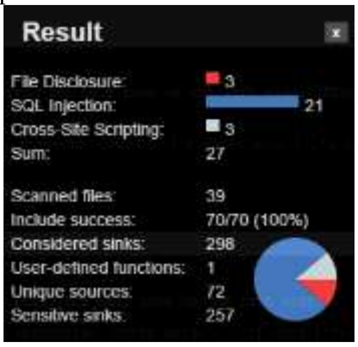
- Windows XP, 2000 and higher versions or any linux operating system
- XAMPP or WAMP servers to run the application developed in php and to setup databases required

Installation of Php-code analyzer:

- Install XAMPP server which automatically installs database of its own (phpmyadmin) on WINDOWS XP, 2000 or higher versions or any Linux operating systems depending on the requirement of the user.
- For this project there is no need of database to be created.
- Copy the whole project of Php=code analyzer into htdocs folder of XAMPP to make project executable.
- Start XAMPP server and run http://localhost/PCA on your system.

VI. WORKING OF PHP-CODE ANALYZER

Main functionality of Php-code analyzer is scanning of web applications for vulnerabilities. Pho-code analyzer automates the scanning of applications and checks for the mentioned vulnerabilities in the applications and generates the report mentioning the risk level of the vulnerability in graph. Below are some of the screenshots of how the scanning process actually works. For example test a locahost site: H:\xampp\htdocs\clinic.

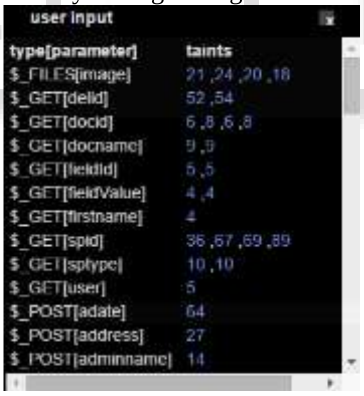


```
File Disclosure:          3
SQL Injection:            21
Cross-Site Scripting:     3
Sum:                      27

Scanned files:            39
Include success:          70/70 (100%)
Considered sinks:         298
User-defined functions:   1
Unique sources:           72
Sensitive sinks:          257
```

Fig. 5: Overview of the result of found vulnerabilities in the local web application

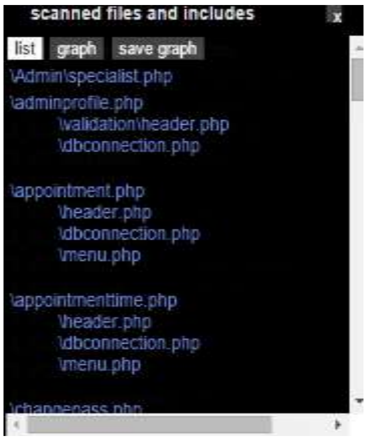
Description: The above figure clearly depicts the overview of vulnerabilities found in the web application and display a pie chart by distinguishing vulnerabilities



```
type[parameter]        taints
$_FILES[image]         21 ,24 ,20 ,18
$_GET[delid]           52 ,54
$_GET[docid]           6 ,8 ,6 ,8
$_GET[docname]         9 ,9
$_GET[fieldid]         5 ,5
$_GET[fieldValue]      4 ,4
$_GET[firstname]       4
$_GET[spid]            36 ,67 ,69 ,89
$_GET[sptype]          10 ,10
$_GET[user]            5
$_POST[adate]          64
$_POST[address]        27
$_POST[adminname]      14
```

Fig. 6: User Input Parameters

Description: The above figure clearly depicts the input parameters of the user



```
\Admin\specialist.php
\adminprofile.php
        \validation\header.php
        \dbconnection.php

\appointment.php
        \header.php
        \dbconnection.php
        \menu.php

\appointmenttime.php
        \header.php
        \dbconnection.php
        \menu.php

\changepass.php
```

Fig. 7: Scanned files list

Description: The above figure clearly depicts the list of scanned files that have been scanned by php-code analyzer.

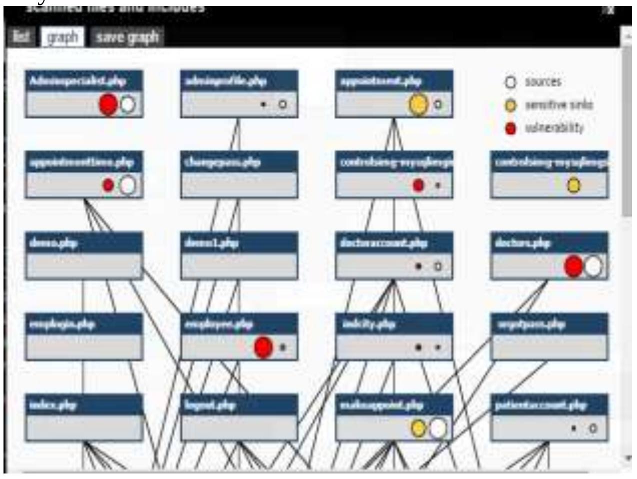
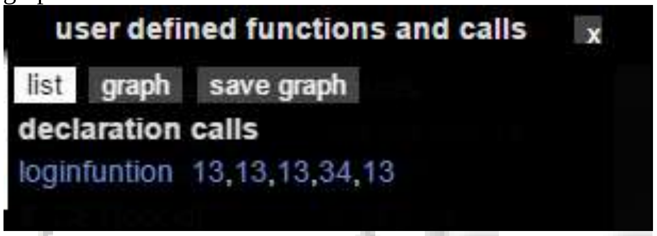


Fig. 8: Graph of scanned files

Description: The above figure clearly depicts the graph structure of the scanned files.



user defined functions and calls x

list graph save graph

declaration calls

loginfuntion 13,13,13,34,13

Fig. 9: user defined function calls

Description: The above figure clearly depicts the user defined function calls that are defined in the code.

File: H:\xampp\htdocs\clinic\Admin\specialist.php

SQL Injection

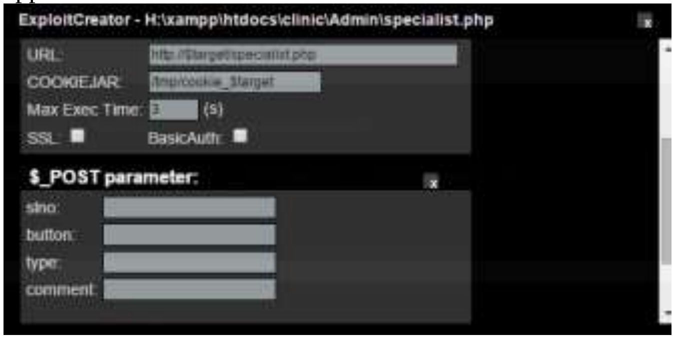
[illegible]

SQL Injection

[illegible]

Fig. 10: Vulnerability found screenshot

Description: The above figure clearly depicts the display of vulnerability found in scanning the web application.



ExploitCreator - H:\xampp\htdocs\clinic\Admin\specialist.php

URL:	http://$target/specialist.php
COOKIEJAR:	/tmp/cookie_$target
Max Exec Time:	3 (s)
SSL: ☐	BasicAuth: ☐

$_POST parameter:

slno:	
button:	
type:	
comment:	

Fig. 11: exploit creator

Description: The above figure clearly depicts the screenshot of exploit creator at the backend of the web application.

VII. BENEFITS OF PHP-CODE ANALYZER

1) Gentle, precise scanning of websites and applications
2) Less complicated, more effective redress
3) Machine-driven assessment process
4) Increased web security and protection
5) Fast, flexible deployment
6) Unparalleled service and support

VIII. METRICS OF PHP-CODE ANALYZER

Graphs are generated by taking every element into consideration as shown in the respective graphs of each metric.

Note: All the graphs are generated through automated performance analysis tool name "WAPT" (10) X axis denotes the time interval of test run. Y axis is different for various graphs. For example:

A. *Performance*

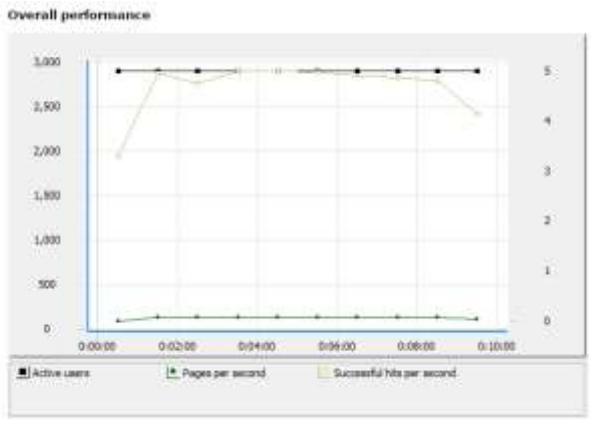


Fig. 12: Performance Analysis graph of Php-code analyzer

Avg response time: Depicts the values of response time averaged through all user profiles. This is response time for main page requests (without page elements).

3 possible variants of Response time graph behavior are shown below: (9)

1) Flat (or very slight growth): It is an ideal result. The increase of load on the server doesn't lead to the increase of response time (or leads to very slight growth).
2) Gradual growth (essential growth): The increase of load on the server leads to gradual growth of response time. It means that the server can handle the growing level of load until the load exceeds some maximum value. Possible reasons of such situation are problems with server hardware, for example, insufficient network bandwidth or low productivity of the server.
3) Sharp growth: If response time graph exhibits a sharp growth beginning from some level of user load while download time graph doesn't grow essentially, it means that the server provides a poor performance when the load reaches this level, or even cannot cope with such load. Users will see that the server responds very slowly, or doesn't respond at all.
   – Avg response time with page elements: Depicts values of average response time for pages including page elements.
   – Avg processing time: Depicts values of processing time averaged through all user profiles. WAPT Pro measures processing time without page elements.
   – Avg download time: Depicts values of download time averaged through all user profiles. Sessions

per second: Depicts the number of sessions executed per time scale unit (second, minute or hour).
- Pages per second: Depicts the number of pages executed per time scale unit.
- Successful hits per second: Depicts the number of hits executed without errors per time scale unit.
- Active users: Depicts the number of virtual users participated in the test. All: Exhibits all graphs on this tab.

B. Errors

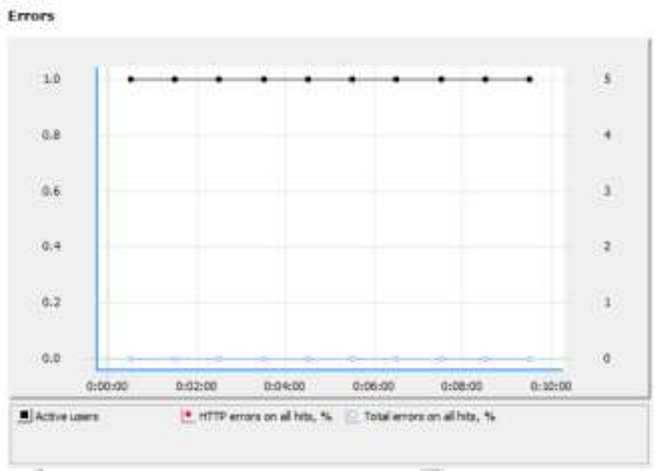


Fig. 13: Error control graph of Php-code analyzer

- HTTP errors on pages, %: Depicts the percentage of responses with HTTP errors from the total number of responses.
- Network errors on pages, %: Depicts the percentage of responses with network errors from the total number of responses.
- Timeouts on pages, %: Depicts the percentage of responses with timeouts from the total number of responses.
- HTTP errors on all hits, %: Depicts the percentage of responses with HTTP errors from the total number of hits, including the errors of page elements.
- Network errors on all hits, %: Depicts the percentage of responses with network errors from the total number of hits, including the errors of page elements.
- Timeouts on all hits, %: Depicts the percentage of responses with timeouts from the total number of hits, including the errors of page elements. JavaScript errors: Depicts the number of JavaScript errors occurred during test run. These are the errors of JavaScript operators and functions included in your profiles.
- Total errors on all hits, %: Depicts the percentage of all responses with errors from the total number of hits, including the errors of page elements.
- Total errors on pages, %: Depicts the percentage of all responses with errors from the total number of responses. Active users: Depicts the number of virtual users participated in the test.

All: Exhibits all graphs on this tab.

This graph will help us to know how error rate changes during a test when the number of virtual users is increasing. Error rate is the most valuable result of stress testing where you need to find the maximum number of users that can be served correctly, without errors. One also need to watch error rate during reliability/endurance tests to verify that it is inacceptable range even after a long run.

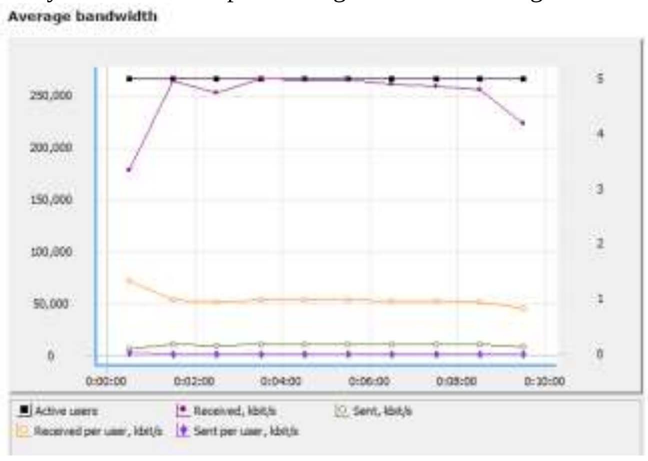


Fig. 14: Bandwidth utilization graph of Php-code analyzer

- Sent: Depicts how many kilobits per second were sent to the server.
- Received: Depicts how many kilobits per second were received from the server.
- Sent per user: Depicts the sending speed per virtual user (in kilobits per second).
- Received per user: Depicts the receiving speed per virtual user (in kilobits per second).
- Active users: Depicts the number of virtual users participated in the test. All: Exhibits all graphs on this tab.

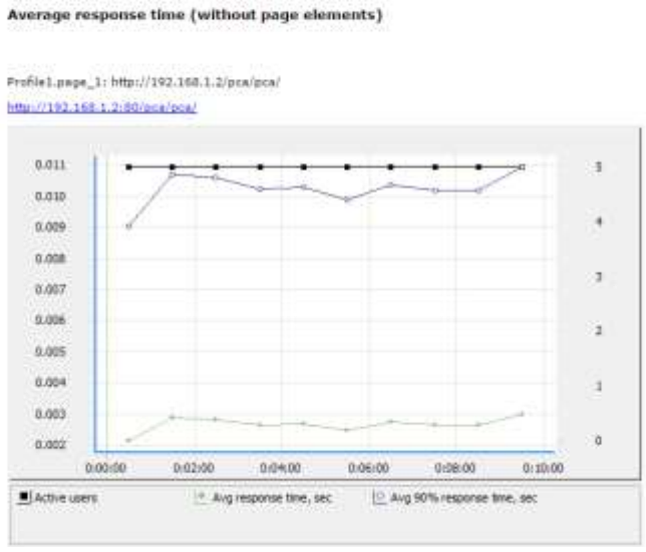


Fig. 15: Bandwidth utilization graph

IX. CONCLUSION AND FUTURE WORK

The main contribution of this paper is to show how easy it is for attackers to discover and exploit application level vulnerabilities in php-based web applications. So we developed a tool Php-code analyzer analyses php-based websites/web applications before deployment for SQL and input validation vulnerabilities. We scanned lots of php based web applications for further analysis and manually confirmed exploitable flaws in identified web pages. Such vulnerabilities, for example, could be used to launch phishing attacks that are difficult to identify even by technically more sophisticated users. With this paper, we hope to raise awareness and provide a tool available to web site administrators and web developers to proactively audit the security of their applications before deploying over the web. In future, we are planning to implement simultaneous scanning of websites and also scanning the websites which were already deployed over the web.

REFERENCES

[1] PHPCrawl webcrawler library/framework (http://phpcrawl.cuab.de/)
[2] PHP HTTP protocol client: HTTP client to access Web site pages (http://www.phpclasses.org/package/3-PHP-HTTP-client-to-access-Web-site-pages.htm)
[3] PHP Simple HTML DOM Parser (http://simplehtmldom.sourceforge.net/)
[4] http://jquery.com/
[5] Realistic Vulnerability Injections in PHP Web Applications by Francisco José Marques Vieira , MESTRADO EM SEGURANÇA INFORMÁTICA 2011
[6] Securing PHP Based Web Application Using Vulnerability Injection by Jamang Jayeshbha Bhalabha,, Amit Doegar and Poonam Saini, International Journal of Information and Computation Technology.ISSN 0974-2239 Volume 3, Number 5 (2013), pp. 391-398
[7] Fast Detection of Access Control Vulnerabilities in PHP Applications by Franc¸ois Gauthier, Ettore Merlo in 2012 19th Working Conference on Reverse Engineering
[8] SQL-Injection Security Evolution Analysis in PHP by Ettore Merlo*, Dominic Letarte, Giuliano Antoniol © 2007 IEEE
[9] An Empirical Study of the Evolution of PHP Web Application Security Maureen Doyle, James Walden, Department of Computer Science, Northern Kentucky University, Highland Heights, KY 41099
[10] http://www.loadtestingtool.com/help/summary-graphs.shtml