Automated vulnerability detection and prevention reporting of php-based web applications using PVRS and comparison between PVRS & RIPS

Santosh Naidu[#1], Subba Raju KV[#2]

[#1]*Computer Science and Engineering, MVGR College of Engineering, India*

[#2]*Computer Science and Engineering, MVGR College of Engineering, India*

[1]amsan2015@gmail.com

[2]srkakarlapudi@gmail.com

***Abstract*—Most of the web servers support some sort of scripting environment today to deploy dynamic web applications. PHP is a general-purpose server-side scripting language for creating dynamic webpages. Most people learn PHP syntax quite quickly and within short period of time they were able to write a script that works using mainly through online tutorials, references and books. The major problem is that most of the people forget the security aspect of PHP that one must consider while writing PHP based applications. Also mentioned are the common programming mistakes done by developers when building PHP web applications and necessary means to protect against such vulnerabilities. Presented are the most common PHP web application vulnerabilities and the necessary mechanisms required to compose secure code by leveraging PHP's unique features.**

So the paper here finally discusses about the tool PVRS which was specially developed for vulnerability detection of php-based web applications and also discusses in brief about the vulnerabilities that are mentioned in the tool and finally compares PVRS tool analysis results with the existing tool RIPS.

***Keywords*—PVRS, RIPS, vulnerability, SQL Injection, Cross-site Scripting, HTTP Banner Disclosure, Direct Object References, Scanner, Crawler**

I. GENERAL TERMS

Security, Vulnerability Assessment

II. INTRODUCTION

Developing PHP applications is passably easy. People clench the syntax rather quickly and within short period of time they will be able to develop a script that works using sessions, mentions & books.

One of the major problems is that most of the people forget the most important and serious aspects that one should consider while developing PHP based web-applications. Almost every beginner forgets the security aspect of PHP.

Some people instantly do some malicious activities and are seeking to do malicious activities on website. Those people scrutinize the application for security defects and exploit these holes. Several times the beginner doesn't know that these things would even be a problem and therefore it might be a problem to fix the holes.

The purpose of this research is to identify common PHP web application vulnerabilities & the necessary mechanisms to write code by leveraging PHP's unique features. As some people learn best by illustrations, I use such illustration vulnerable code and also live vulnerable websites to show exploitation of vulnerabilities in PHP applications by using PVRS (Php Vulnerability Reporting System).

III. RELATED WORK

- In 2007,(1) Ettore Merlo along with Dominic Letarte and Giuliano Antoniol describes 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.

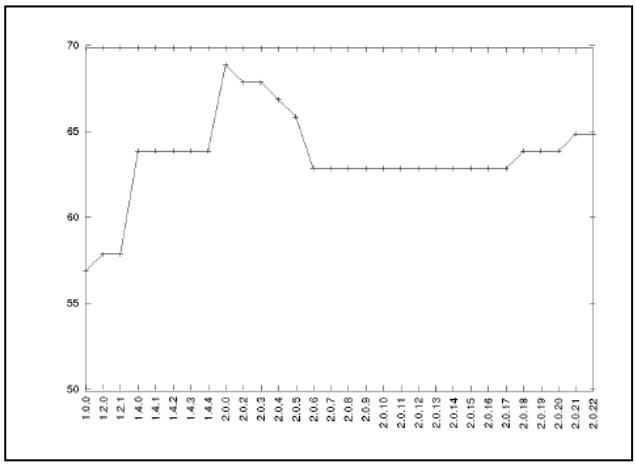

Fig.1: Percentage of vulnerable DB accesses

- In 2010,(2) Johannes Dahse developed a open source tool named RIPS which automates the process of vulnerability detection of php based web applications and implementation of RIPS and also the different kind of problems while building a static source code analysis tool for PHP.

Fig.2: Web Interface of RIPS with scan result & code viewer

- In 2009,(3) Nuno Seixas, Jose Fonseca, Marco Vieira and Henrique Madeira presented different web security vulnerabilities from programming language view in 20th International Symposium on Software Reliability Engineering. And also described about security patches are reported for a set of widely used web applications written in different languages (Java, C#, VB.NET) which are analysed in order to understand the fault types that are responsible for the vulnerabilities discovered (SQL injection and XSS).

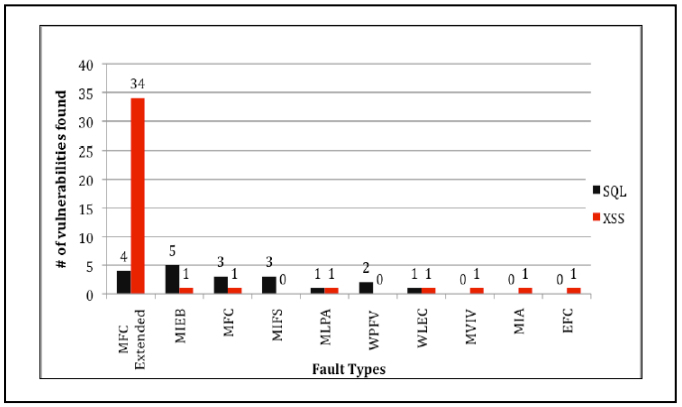

Fig.3: Vulnerabilities fault type summary

- In 2011, (4) Francisco José Marques Vieira proposed an architecture for vulnerability injection tool which allows the intromission of vulnerabilities in a program/script and is an extensible one that might support addition of new vulnerabilities to inject.
- In 2013, (5) Jamang Jayeshbha Bhalabha along with Amit Doegar and Poonam Saini have done some modifications/extension 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, (6) Francois Gauthier and Ettore Merlo designed and developed 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.

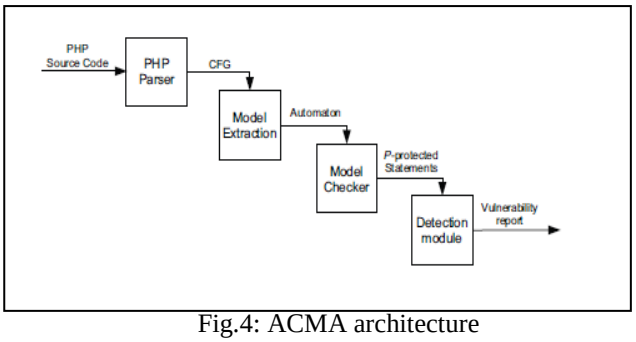

Fig.4: ACMA architecture

- In 2012, (7) Maureen Doyle and James Walden have done investigation on the evolution of vulnerabilities in PHP web applications and also calculating vulnerability densities.

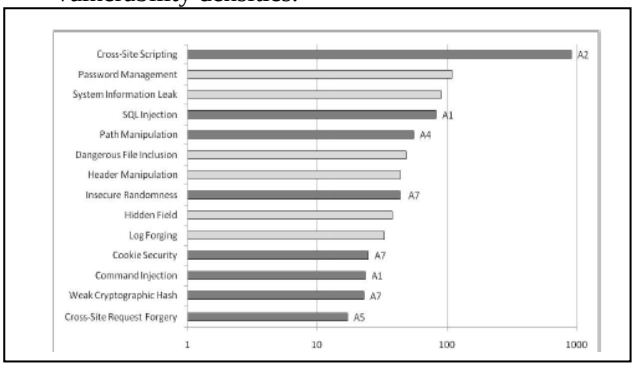

Fig.5: Aggregate Vulnerabilities by type

- In 2013, (8) XIAOWEI LI and YUAN XUE surveyed security aspects in web applications by systemizing the existing techniques which might be used for further research. Also described about input validation vulnerabilities, session management vulnerabilities and application logic vulnerabilities.

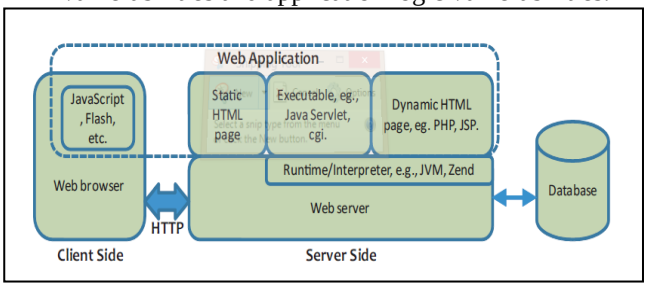

Fig.6: Overview of web application

IV. DISTINCTIVE WEB ATTACKS

1. Reflected Cross-Site Scripting

Cross-Site Scripting (XSS) is a vulnerability that allows code to be injected into the web application and viewed by users. The code may include HTML, JavaScript, or other client-side languages. Reflected XSS is a variation of XSS where user-controllable data is displayed back to the user, in the HTTP response of the request used for the attack, without being validated correctly.

2. **Stored Cross-Site Scripting**
   Cross-Site Scripting (XSS) is a vulnerability that allows code to be injected into the web application and viewed by other users. The code may include HTML, JavaScript, or other client-side languages. Stored XSS is a variation of XSS where user-controllable data is stored and displayed to users at a later stage. For example, product reviews or posts on a forum.
3. **Standard SQL Injection**
   SQL injection is a technique in which a user submits SQL statements to a web application in an attempt to exploit the database layer of the application. This can be performed using a browser and entering the SQL statements in a form and submitting the form.
4. **Broken Authentication using SQL Injection**
   SQL injection is a technique in which a user submits SQL statements to a web application in an attempt to exploit the database layer of the application. This can be performed using a browser and entering the SQL statements in a form and submitting the form.The login form can be bypassed using a form of SQL Injection that manipulates the SQL query behind the login form so that it will return one or more results. Therefore, a malicious user can login to the web application without the knowledge of valid credentials.
5. **Autocomplete enabled on sensitive fields**
   'Autocomplete occurs when the browser caches data, such as a user's username and password for an application, so the user will not have to enter them any time they access the application. Forms that process sensitive data such as passwords should always have autocomplete disabled. If an attacker gains access to a user's browser cache, he could easily obtain the sensitive information which may be saved in plaintext.
6. **Direct Object References**
   'Exposing a reference to an internal implementation object is known as a direct object reference. An example of an internal implementation object would be a file, directory, or database key. If the reference can be edited by a user and sufficient control is not in place, the user could manipulate the reference and possibly access unauthorized resources. For example, a URL like the following is exposing a direct object reference:
   http://www.example.com/displayFile.php?file=stats.txt. However, a malicious user could replace the file name and re-request the URL again in order to try and obtain system passwords. For example: http://www.example.com/displayFile.php?file=/../../../../etc/passwd. Automated tools cannot typically identify such defects as they cannot make out what requires protection and what is secure or insecure. Therefore, this result is only indicating that values which look like direct object references are exposed and they may be insecure.
7. **Directory Listing Enabled**
   The contents of one or more directories can be viewed by web users. Therefore, when a user requests a directory such as http://www.example.com/directoryname/ using their browser, a list of all files and directories contained in the requested directory will be displayed to the user. This could possibly expose sensitive information such as executables, text files, documentation, and installation and configuration files. An attacker could use these to map out the server's directory structure and identify potentially vulnerable files or applications
8. **HTTP Banner Disclosure**
   'The application discloses information about the technologies used such as the web server, operating system, cryptography tools, or programming languages. An attacker could identify vulnerabilities in these technologies and use them to exploit the server, therefore, potentially exploiting the application.
9. **SSL Certificate not trusted**
   The web application is using a SSL certificate which has been checked against Mozilla's bundle of X.509 certificates of public Certificate Authorities and cannot be found. Therefore, the certificate cannot be validated and is not trusted.
10. **Invalidated Redirects**
    The application is redirecting the user to a page based on user-controllable data that is not validated correctly. An example of this would be a link that requests the following URL: http://www.example.com?redirect.php?redirect=user_can_replace_this.html. This could be edited to http://www.example.com?redirect.php?redirect=http://www.malicous-site.com, and if not validated correctly, it will redirect to the malicious site. Links like the latter could then be emailed to potential victims and they may click on them with no hesitation as www.example.com may be a trusted domain.

V. ABOUT PVRS

PVRS-Php Vulnerability Reporting System firstly crawls the target website to shoot out all URLs belonging to the website. It tests each URL for different vulnerabilities and generates detailed report in PDF format, once the scan is complete.
PVRS is a best-in-class web scanning solution that rapidly and accurately scans large, complex web sites and web applications to undertake web-based vulnerabilities. PVRS identifies application vulnerabilities as well as site exposure risk, ranks threat priority, produces highly graphical, intuitive HTML reports, and indicates site security posture by vulnerabilities and threat level.

VI. IMPLEMENTATION & DEPLOYMENT OF PVRS

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/)
   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)
   Function: It provides the functionality of using HTTP protocol in php code.
3. PHP Simple DOM Parser (http://simplehtmldom.sourceforge.net/)
   Function: It is one of the libraries which is useful for parsing Document Object Models such as HTML content in php
4. TCPDF (http://www.tcpdf.org/)
   Function:It is one of the libraries which are useful for generating reports in pdf format.
5. jQuery (http://jquery.com/)
   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 PVRS:

- 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.
- Create a database named "webvulscan" using phpmyadmin.
- Create four tables tests, test_results, users and vulnerabilities respectively in the database "webvulscan"
- Copy the whole project of PVRS into htdocs folder of XAMPP to make project executable.
- Start XAMPP server and run http://localhost/PVRS on your system.

VII. WORKING OF PVRS

Main functionality of PVRS is scanning of web applications for vulnerabilities. PVRS automates the scanning of applications and checks the above mentioned serious vulnerabilities in the applications and generates the report mentioning the risk level of the vulnerability in pie-chart and also the recommendations or necessary precautions to overcome those vulnerabilities. Below are some of the screenshots of how the scanning process actually works. For example test a sample sites http://schemaxitinfra.com/ and http://gitam.edu/WelcomePage.aspx for scanning. First of all register into the application and login with registered user credentials and go to Scanner tab. In that tab, under Enter URL box enter the website which we want to scan (http://schemaxitinfra.com/) and (http://gitam.edu/WelcomePage.aspx) respectively as shown in Fig 7 for all the vulnerabilities mentioned with check boxes. Check in boxes for the vulnerabilities to be scanned. By default all the vulnerabilities will be checked in.

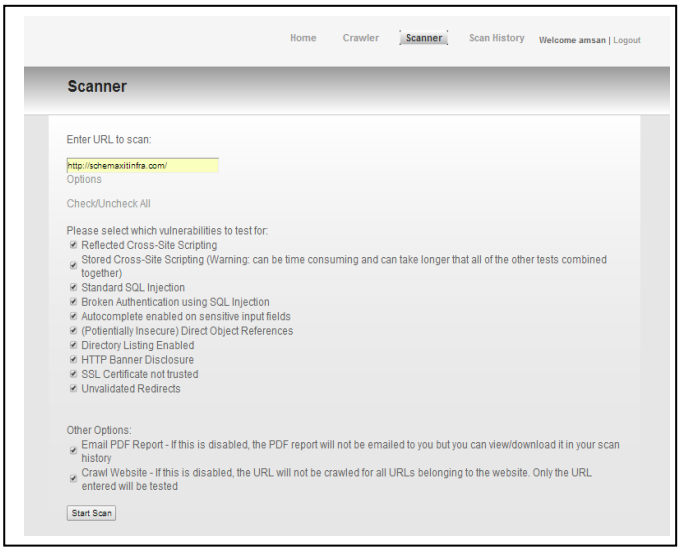

Fig 7: The above interface shows the the URL entered is ready for scanning by checking all the check boxes for vulnerabilities mentioned

Fig 8: Interface showing the scanning status after the scanning process is initiated

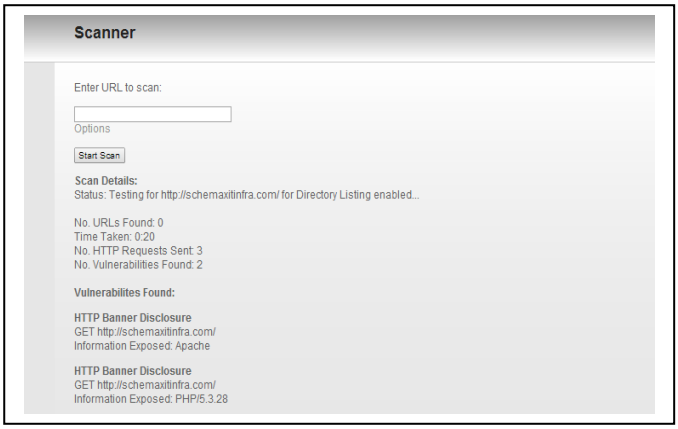

Fig 9: Interface showing the scanning status with vulnerabilities found till that time (http://schemaxitinfra.com/)

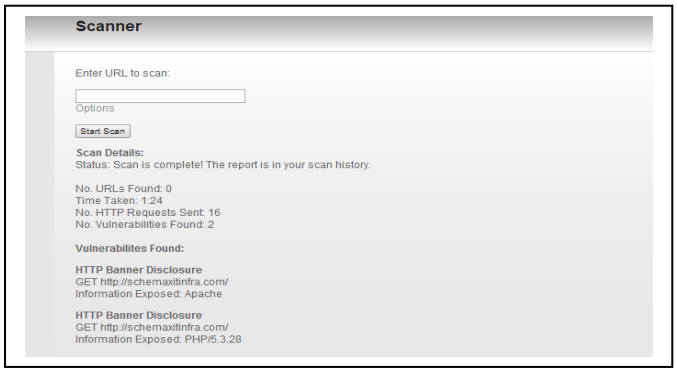

Fig 10: Interface showing the scanning status after the scanning process is completed (http://schemaxitinfra.com/)

Fig 11: Interface showing the scanning history along with pdf reports (http://schemaxitinfra.com/)

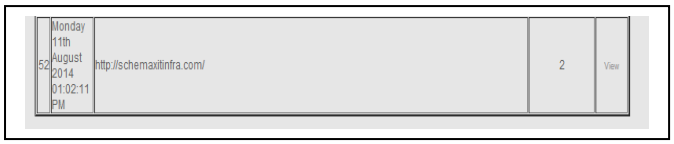

Fig 12: Interface showing the scanning history of our example website (http://schemaxitinfra.com/)

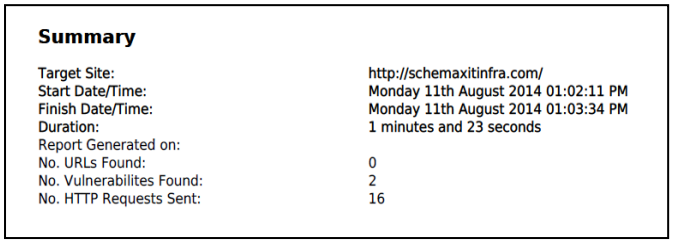

Fig 13: Pdf summary report of the scanning (http://schemaxitinfra.com/)

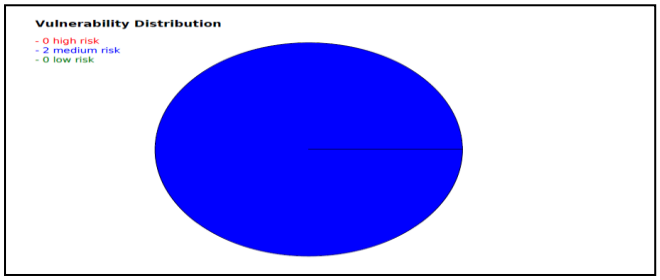

Fig 14: Pdf report showing risk level along with pie-chart representation (http://schemaxitinfra.com/)

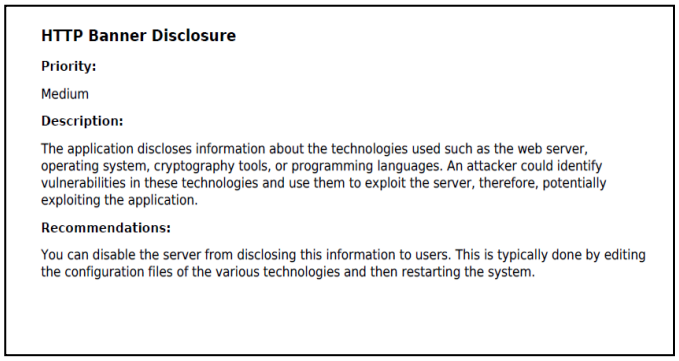

Fig 15: Pdf report showing vulnerability description and recommendations (http://schemaxitinfra.com/)

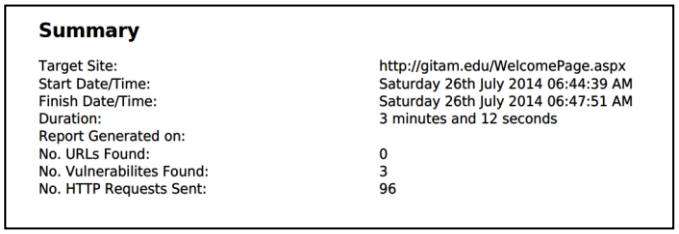

Fig 16: Interface showing the summary report for another website (http://gitam.edu/WelcomePage.aspx)

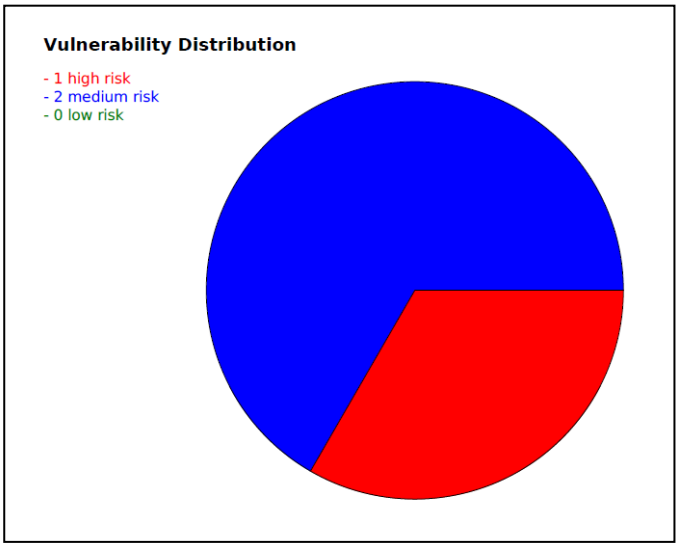

Fig 17: Pdf report showing risk level along with pie-chart representation (http://gitam.edu/WelcomePage.aspx)

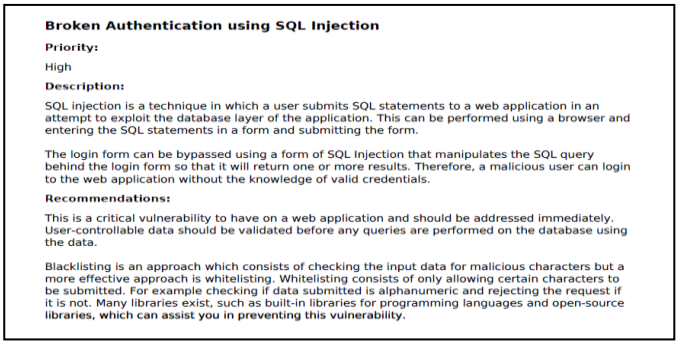

Fig 18: Pdf report showing vulnerability description and recommendations (http://gitam.edu/WelcomePage.aspx)

VIII. FEATURES OF PVRS

- **Crawler:** Searches a website to discover and show all URLs belonging to the website.
- **Scanner:** Searches a website and skims all URLs found for above mentioned vulnerabilities.
- **Scan History:** Permits a user to view or download PDF reports of previous scans that they performed.
- **Register:** Permits a user to register with the application of PVRS
- **Login:** Permits a user to login to the application of PVRS
- **Options:** Permits a user to select which vulnerabilities they wish to test for scanning.
- **PDF Generation:** PDF report is generated dynamically once the scanning is completed.
- **Report Delivery:** The PDF report thus generated is mailed to the user.
- Precise, comprehensive web application scanning
- Exposure prioritization by risk level
- Overall menace Analysis
- Web services support

IX. BENEFITS OF PVRS

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

X. RECOMMENDATIONS TO THE VULNERABILITIES IN PVRS

1. **Reflected Cross-Site Scripting**
   This can be high risk vulnerability and can be underestimated. Mitigating this vulnerability uses a two-fold approach. Ensure all user-controllable data is validated after it is inputted and again before it is outputted to users. Blacklisting is an approach which consists of checking the input data for malicious characters but a more effective approach is whitelisting. Whitelisting consists of only allowing certain characters to be submitted. For example checking if data submitted is alphanumeric and rejecting the request if it is not. You can use an approach like this after data is submitted and then perform a similar approach before data is outputted to the user.
2. **Stored Cross-Site Scripting**
   This is high risk vulnerability. Mitigating this vulnerability uses a two-fold approach. Ensure all user-controllable data is validated before it is stored and again before it is outputted to users. Blacklisting is an approach which consists of checking the input data for malicious characters but a more effective approach is whitelisting. Whitelisting consists of only allowing certain characters to be submitted. For example checking if data submitted is alphanumeric and rejecting the request if it is not. You can use an approach like this after data is submitted and then perform a similar approach before data is outputted to the user.
3. **Standard SQL Injection**
   'This is a critical vulnerability to have on a web application and should be addressed immediately. Validation should be done for user-controllable data in the earlier stage of any query performance on the database using the data. Blacklisting is an approach which consists of checking the input data for malicious characters but a more effective approach is whitelisting. Whitelisting consists of only allowing certain characters to be submitted. For example checking if data submitted is alphanumeric and rejecting the request if it is not. Many libraries exist, such as built-in libraries for programming languages and open-source libraries, which can assist you in preventing this vulnerability.
4. **Broken Authentication using SQL Injection**
   This is a critical vulnerability to have on a web application and should be addressed immediately. User-controllable data should be validated before any queries are performed on the database using the data. Blacklisting is an approach which consists of checking the input data for malicious characters but a more effective approach is whitelisting. Whitelisting consists of only allowing certain characters to be submitted. For example checking if data submitted is alphanumeric and rejecting the request if it is not. Many libraries exist, such as built-in libraries for programming languages and open-source libraries, which can assist you in preventing this vulnerability.
5. **Autocomplete enabled on sensitive fields**
   Disable the autocomplete attribute of input fields that hold sensitive data. This can be done by placing autocomplete=\\"off\\" inside the tags of the input field. You can also disable autocomplete for an entire form by placing it inside the tags of the form.
6. **Direct Object References**
   Use indirect object references. For example, using the above scenario, pass an integer to the URL and this could be mapped to a file name once the request is made. If direct object references must be used, have an access control check in place to ensure the user is authorized to view the requested resource.
7. **Directory Listing Enabled**
   This can be high risk vulnerability. This is typically enabled in the server's configuration file but can sometimes arise from vulnerability in particular applications. You can eliminate this vulnerability by disabling directory listing in the server's configuration file and restart the server. The location and name of this file differs depending on what web server you are using.

8. **HTTP Banner Disclosure**
   You can disable the server from disclosing this information to users. This is typically done by editing the configuration files of the various technologies and then restarting the system.
9. **SSL Certificate not trusted**
   Ensure the SSL certificate has been issued by a trusted authority.
10. **Invalidated Redirects**
   The target site that the user is being redirected to should not be exposed. If there is no way around this or it is simply too much effort to edit this design, ensure the user-controllable data is validated before redirecting to it. One countermeasure is to maintain a list of safe URLs that can be redirected to and check the user-controllable value against this list before performing the redirect. Another good counter-measure is to pass an integer to the URL that is redirecting. For example:
   http://www.example.com?redirect.php?redirect=3.
   This integer acts as an array index and an array of safe URLs is maintained by the web applications.

XI. METRICS OF PVRS

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" (9)
X axis denotes the time interval of test run. Y axis is different for various graphs. For example:

Active users : The number of active users is displayed on the right side of the chart.

Pages per second : The number of pages executed per second is displayed on the left side of the chart.

1) Performance:

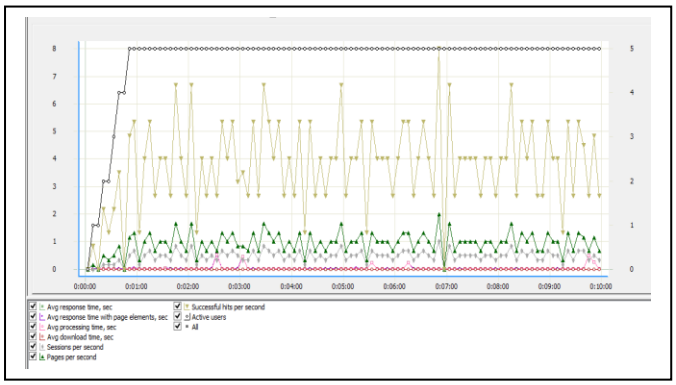

Fig 19 : Performance Analysis graph of PVRS

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 behaviour 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.

2) Average Bandwidth:

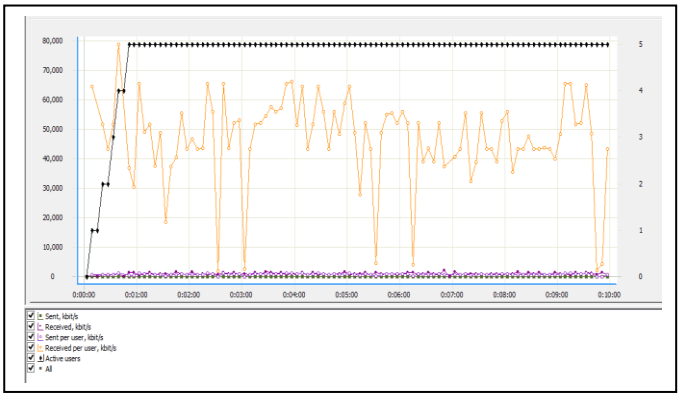

Fig 20: Bandwidth utilization graph of PVRS

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.

3) Errors:

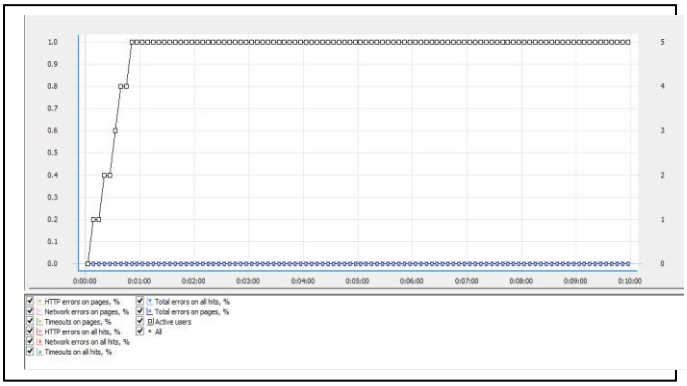

Fig 21: Error control graph of PVRS

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.

XII. COMPARISON BETWEEN PVRS AND RIPS

Comparison is done by taking few elements into consideration as shown in the respective graphs of each metric.

1. <u>In terms of Performance:</u>

a) PVRS

Fig 22: Overall performance analysis of graph of PVRS by considering few elements for metric

As seen in the both graphs from Fig 22 and Fig 23, we can clearly observe that performance is much better in PVRS than RIPS. In the graph we can observe successful hits per second works much better in PVRS when compared to RIPS. So this might be one of the advantages of PVRS when compared to RIPS.

b) RIPS

Fig 23: Overall performance analysis of graph of RIPS by considering few elements for metric

2. In terms of Bandwidth:

a) PVRS

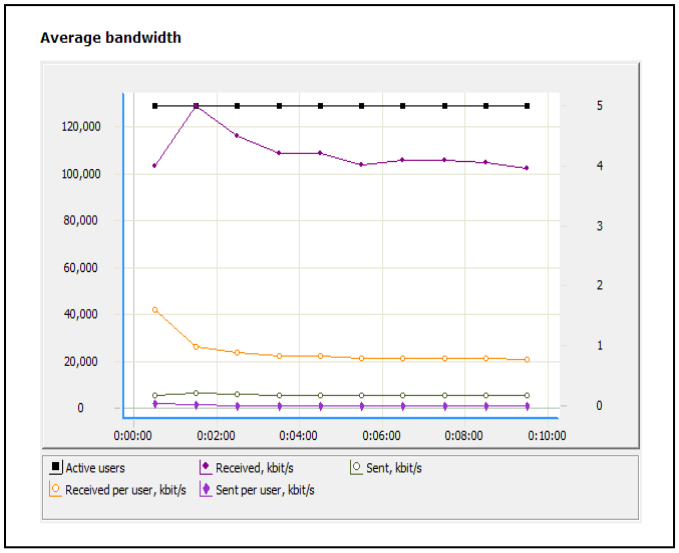

Fig 24: Overall performance analysis of graph of RIPS by considering few elements for metric

As seen in the both graphs from Fig 24 and Fig 25, we can clearly observe that bandwidth utilization is much lesser in PVRS than RIPS. In the graph we can observe bits received per user always goes almost in a linear fashion using lesser bandwidth, whereas in PVRS there are much deviations in data received per user and doesn't goes in a linear fashion, in spite it goes in non-linear with lot of modulations and deviations in the graph using more bandwidth when compared to PVRS. So this might be one more advantages of PVRS when compared to RIPS.

b) RIPS

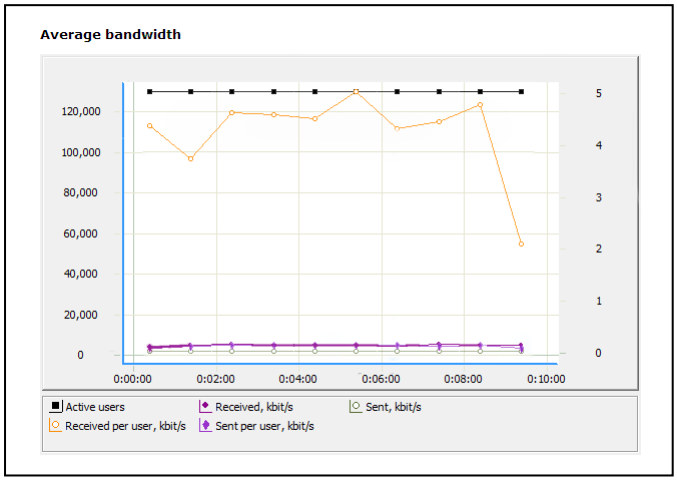

Fig 25: Overall performance analysis of graph of RIPS by considering few elements for metric

3. In terms of Average Response Time:

a) PVRS

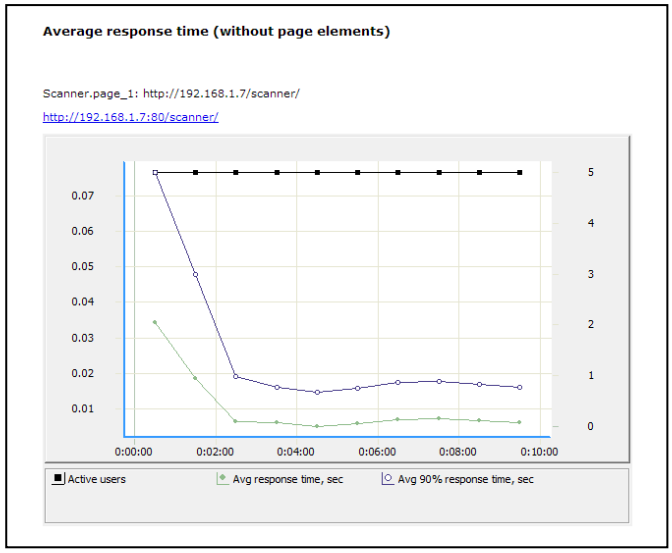

Fig 26: Overall performance analysis of graph of RIPS by considering few elements for metric

As seen in the both graphs from Fig 26 and Fig 27, we can clearly observe that response time is much lesser in PVRS than RIPS. In the graph we can observe average response time always goes on decreasing, whereas in PVRS there are many deviations in average response time and doesn't degrade but increases dynamically, so PVRS response time is very less when compared to RIPS. So this might be one more advantages of PVRS when compared to RIPS.

b) RIPS

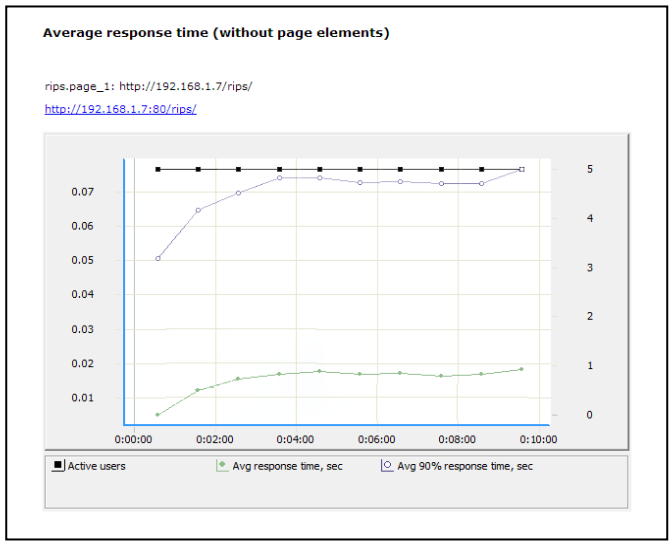

Fig 27: Overall performance analysis of graph of RIPS by considering few elements for metric

Tabular Comparison between PVRS and RIPS:

Tool	Overall Performance	Average Response Time	Average Bandwidth	Errors
PVRS	**Better performance results when compared with RIPS**	**Less Response Time when compared to RIPS**	**Less bandwidth when compared to RIPS**	**No errors**
RIPS	**Overall Performance is good**	**Response Time is more**	**More Bandwidth Utilization**	**No errors**

Table 1: Comparison between PVRS and RIPS showing PVRS is advantageous than RIPS

XIII. CONCLUSIONS AND FUTURE WORK

The main contribution of this paper is to show how easy it is for attackers to automatically discover and exploit application-level vulnerabilities in a large number of web applications. To this end, we presented PVRS, a generic and modular web vulnerability scanner that analyses web sites for exploitable SQL and some input validation vulnerabilities. We used PVRS to identify a large number of potentially vulnerable web sites. Moreover, we selected one hundred of these web sites for further analysis and manually confirmed exploitable flaws in the identified web pages. Among our victims were well-known global companies, computer security organizations, and governmental and educational institutions. We believe that it is only a matter of time before attackers start using automated vulnerability scanning tools such as PVRS to discover vulnerabilities that they can exploit. (10)

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.

For the future, we are planning to implement simultaneous scanning of websites at a time which reduces scanning time. Also, there is certainly some room for improvement in the performance and throughput of the tool.

XIV. ACKNOWLEDGEMENTS

I would like to thank Department of Computer Science and Engineering staff and management of MVGR College of Engineering for their support in doing this work.

REFERENCES

1) SQL-Injection Security Evolution Analysis in PHP by Ettore Merlo*, Dominic Letarte, Giuliano Antoniol © 2007 IEEE
2) RIPS - A static source code analyser for vulnerabilities in PHP scripts by Johannes Dahse Seminar Work at Chair for Network and Data Security
3) Looking at Web Security Vulnerabilities from the Programming Language Perspective: A Field Study by Nuno Seixas, José Fonseca, Marco Vieira, Henrique Madeira in 20th International Symposium on Software Reliability Engineering
4) Realistic Vulnerability Injections in PHP Web Applications by Francisco José Marques Vieira , MESTRADO EM SEGURANÇA INFORMÁTICA 2011
5) 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
6) Fast Detection of Access Control Vulnerabilities in PHP Applications by Franc¸ois Gauthier, Ettore Merlo in 2012 19th Working Conference on Reverse Engineering
7) 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
8) A Survey on Server-side Approaches to Securing Web Applications by XIAOWEI LI-Vanderbilt University, YUAN XUE-Vanderbilt University, ACM Transactions on Computing Surveys, Vol. V, No. N, November 2013
9) http://www.loadtestingtool.com/help/summary-graphs.shtml
10) SecuBat: A Web Vulnerability Scanner by Stefan Kals, Engin Kirda, Christopher Kruegel, and Nenad Jovanovic

