International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 5 | ISSN : 2456-3307

# Identify And Eliminating Online Application Misbehaviors by Static Analysis Approach

**P Abdul Habeeb, Md Ateeq Ur Rahman**
Department of Computer Science & Engineering, Shadan College of Engineering & Technology, Hyderabad, Telangana, India

## ABSTRACT

An extensive research work on web application security has been continuing for over 10 years, the security of web applications keeps on being a difficult issue. An essential some portion of that issue gets from unprotected source code, regularly written in risky dialects like PHP. Source code static investigation devices are response for discover vulnerabilities, however they have a tendency to produce false positives, and require extensive work for software engineers to resolve the code. We investigate the utilization of a mix of strategies to find vulnerabilities in source code with less false positives. We combine Taint analysis, which discovers hopeful vulnerabilities, with information mining, to predict the presence of false positives. This approach unites two methodologies that are obviously orthogonal: people coding the information about vulnerabilities (for Taint Analysis), joined with the apparently orthogonal approach of consequently getting that information (with machine learning, for information mining). Given this upgraded type of detection, we propose doing programmed code remedy by embeddings settles in the source code. Our approach was executed in the WAP device, and an investigative assessment was performed with an expansive arrangement of PHP applications. Our apparatus discovered 388 vulnerabilities in 1.4 million lines of code. Its exactness and accuracy were roughly 5% superior to PhpMinerII's and 45% superior to Pixy's.

**Keywords:** Data Mining, Web Protection, Input Validation Vulnerabilities, Software Security, Source Code Static Analysis, Web Applications, PHP

## I. INTRODUCTION

SINCE its appearance in the Mid 1990s, the World Wide Web advanced from a stage to get content and other media to a structure for running complex web applications. These applications show up in many structures, from little home-made to huge scale business administrations (e.g., Google Docs, Twitter, Facebook). Be that as it may, web applications have been tormented with security issues. For instance, a current report demonstrates an expansion of web assaults of around 33% of every 2012. Apparently, a purpose behind the weakness of web applications is that numerous software engineers need fitting learning about secure coding, so they leave applications with imperfections.

This paper investigates an approach for consequently ensuring web applications while keeping the software engineer tuned in. The approach comprises in examining the web application source code hunting down info approval vulnerabilities, and inserting settles in a similar code to redress these defects. The software engineer is kept tuned in by being permitted to comprehend where the vulnerabilities were found, and how they were adjusted. This approach contributes straightforwardly to the security of web applications by evacuating vulnerabilities, and by implication by giving the software engineers a chance to gain from their mix-ups. This last viewpoint is empowered by inserting fixes that take after regular security coding rehearses, so software engineers can take in these practices by observing the vulnerabilities, and how they were evacuated. We investigate the utilization of a novel mix of techniques to recognize this sort of weakness: static examination with information mining. Static examination is a compelling component to discover vulnerabilities in source code, however tends to report numerous false positives (non-vulnerabilities) because of its un decidability. This issue is especially troublesome with dialects, for example, PHP that are feebly written, and not formally determined. Consequently, we supplement a type of static investigation, pollute examination, with the utilization of information mining to foresee the presence of false positives. This arrangement consolidates two evidently disjoint methodologies: people coding the information about vulnerabilities (for corrupt investigation), in mix with consequently acquiring that information (with managed machine picking up supporting information mining). To anticipate the presence of false positives, we present the original thought of surveying if the vulnerabilities distinguished are false positives utilizing information mining. To do this evaluation, we measure properties of the code that we saw to be related with the nearness of false positives, and utilize a mix of the three best positioning classifiers to signal each powerlessness as false positive or not. We investigate the utilization of a few classifiers: ID3, C4.5/J48, Random Forest, Random Tree, K-NN, Naive Bayes, Bayes Net, MLP, SVM, and Logistic Regression. In addition, for each helplessness delegated false positive, we utilize an enlistment manage classifier to indicate which traits are related with it. We investigate the JRip, PART, Prism, and Ridor acceptance administer classifiers for this objective. Classifiers are naturally designed utilizing machine learning in view of marked helplessness information.

Guaranteeing that the code redress is done accurately requires surveying that the vulnerabilities are expelled, and that the right conduct of the application is not adjusted by the fixes.

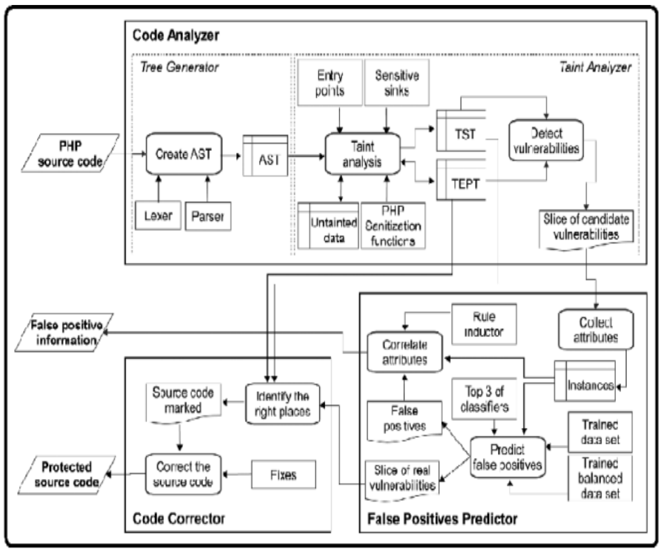


**Figure1.** Proposed system framework

## II. EXISTING AND PROPOSED SYSTEMS

### 2.1 Existing System:

There is a large corpus of related work, so we simply outline the fundamental discussing by examining papers, while leaving many others unreferenced to preserve space. Static investigation tools mechanize the evaluating of code, either source, paired, or intermediate.

Taint analysis tools like CQUAL and Splint (both for C code) utilize two qualifiers to comment on source code: the untainted qualifier demonstrates either that a function or parameter returns reliable information (e.g., a disinfection work), or a parameter of a function requires dependable information (e.g., mysql_query). The tainted qualifier implies that a capacity or a parameter returns non-dependable information (e.g., functions that read client input).

#### 2.1.1 Disadvantages of Existing System:

These different works did not expect to distinguish bugs and recognize their location, yet to evaluate the nature of the software as far as the common defects and vulnerabilities. WAP does not utilize information mining to recognize vulnerabilities, but rather to predict whether the vulnerabilities found by taint analysis are truly vulnerabilities or false positives.

### 2.2 Proposed System:

Our approach is about input validation vulnerabilities, so this area introduces quickly some of them (those deal with by the WAP tool). Inputs enter an application through section point (e.g., $_GET), and attempt a vulnerability by achieving a sensitive sink (e.g., mysql_query). Most attacks include combine typical contribution with meta-characters or metadata (e.g., ', OR), so applications can be secured by placing disinfection works in the ways between section point and sensitive sinks.

SQL injection (SQLI) vulnerabilities are caused by the utilization of string-building methods to execute SQL queries. PHP code helpless against SQLI. This content embeds in a SQL query as the username and password gave by the client. In the event that the client is noxious, he can give as username administrator' - , making the content execute a question that profits data about the client administrator without the need of providing a password.

### 2.2.1 Advantages of Proposed System:

- Ensuring that the code correction is done effectively requires evaluating that the vulnerabilities are deleted, and that the right conduct of the application is not changed by the fixes.
- We propose utilizing program mutation and regression testing to confirm, individually, that the fixes work as they are customized to (blocking vulnerable sources of info), and that the application works same (with kind information inputs).
- An approach for enhancing the security of web applications by consolidating data.

## III. METHODOLOGY

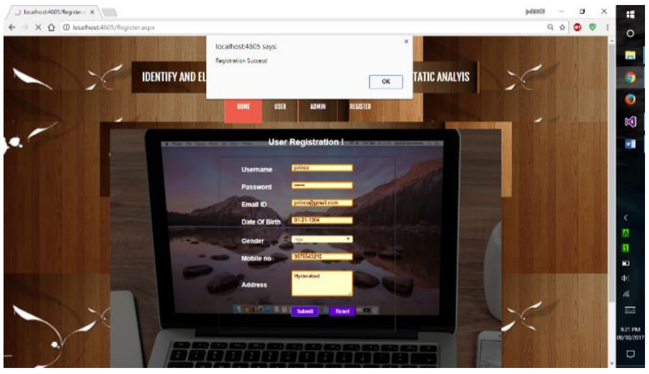
This vulnerability can be displaced either by disinfecting the sources of info (e.g., going before with oblique punctuation line meta-characters, for example, the prime), or by utilizing arranged explanations. We selected the previous in light of the fact that it requires less difficult adjustments to the code. Sterilization relies upon the sensitive sink, i.e., in transit in which the information is utilized. For SQL, and the MySQL database, PHP gives the mysql_real_escape_ string capacity.

We just present alternate vulnerabilities quickly because of absence of space (with more data in). A remote file inclusion (RFI) vulnerability enables aggressors to insert a remote record containing PHP code in the vulnerable program. Local file inclusion (LFI) contrasts from RFI on the grounds that it embeds a record from the local file system of the web application (not a remote file). An directory traversal or path traversal (DT-PT) assault comprises in an aggressor getting to discretionary nearby records, perhaps outside the site registry. Source code disclosure (SCD) assaults dump source code and arrangement records. A working framework charge injection (OSCI) assault comprises in constraining the application to execute a command characterized by the assailant. A PHP code injection (PHPCI) vulnerability enables an aggressor to supply code that is executed by an eval explanation.

## IV. IMPLEMENTATION & RESULTS

Cross-site scripting (XSS) assaults execute malicious code (e.g., JavaScript) in the victim's browser. Not quite the same as alternate assaults we consider, a XSS assault is not against a web application itself, but rather against its clients. There are three fundamental classes of XSS assaults relying upon how the malevolent code is sent to the casualty (reflected or non-determined, put away or relentless, and DOM-based); however we clarify just reflected XSS for quickness. A content helpless against XSS can have a solitary line: resound "The assault includes persuading the client to tap on a connection that gets to the web application, sending it a content that is reflected by the reverberate guideline and executed in the program. This sort of assault can be forestalled by cleaning the information, or by encoding the output, or both. The last comprises in encoding meta-characters, for example, in a way that they are translated as typical characters, rather than HTML meta-characters.

**Below are some of the screenshots of the process.**


localhost:4605/Register.aspx

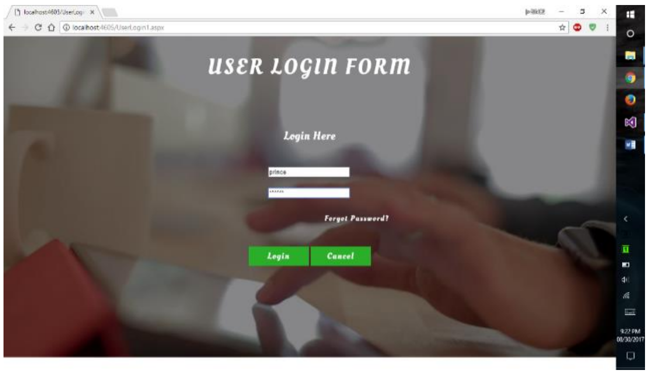
localhost:4605 says:
Registration Success!

User Registration !

Username: prince
Password: ••••••
Email ID: prince@gmail.com
Date Of Birth: 01-01-1994
Gender: Male
Mobile no: 9876543210
Address: Hyderabad

Output Screenshot 1: Registration page

In this page user need to create an account by providing basic information like username, password, email id, address and mobile number to register. Once registration is done, user can sign in the application.


localhost:4605/UserLogin1.aspx

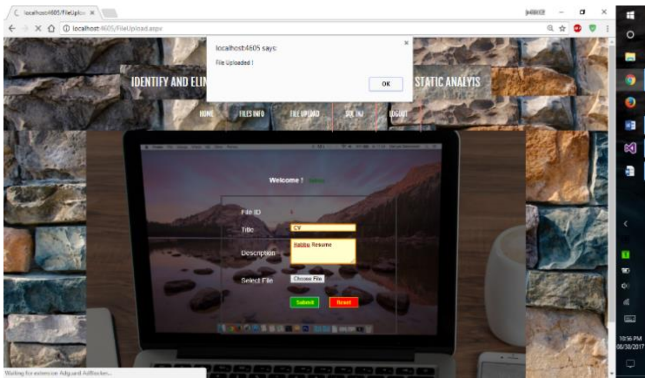
USER LOGIN FORM

Login Here

prince
••••••

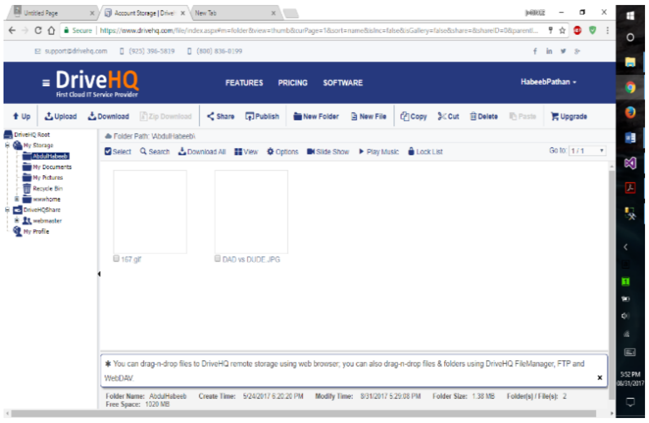
Output Screenshot 2: User Login page

A login page provides user to login in to the website and access it, who had registered in above register page. User login page have basic information like username, password. If the user credentials are correct, user can login in to the website.



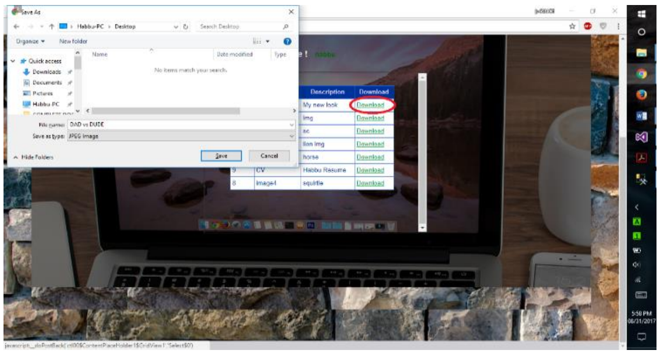
Output Screenshot 3: File upload page

After login in to the cloud server, file upload page is provided to upload the file data in cloud server.



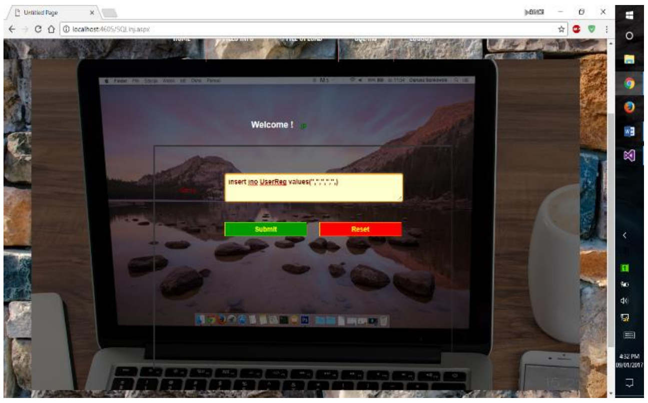
Output Screenshot 4: File upload in DriveHQ

After login into the cloud server account, the user will receive the uploaded file in cloud server.



Output Screenshot 5: file download

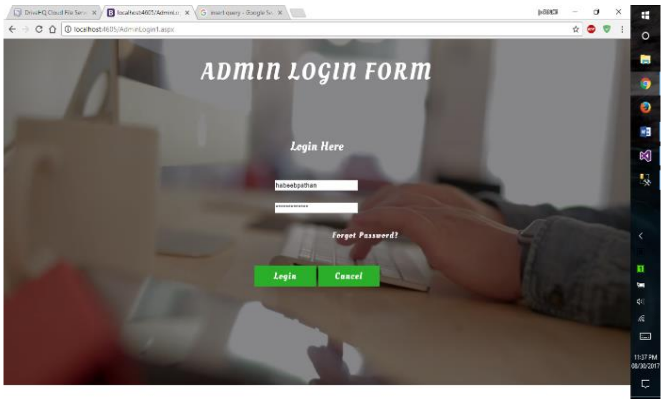
Here user can download the files. The download link will be provided in this page.



```
insert ino UserReg values('','','','','')
```

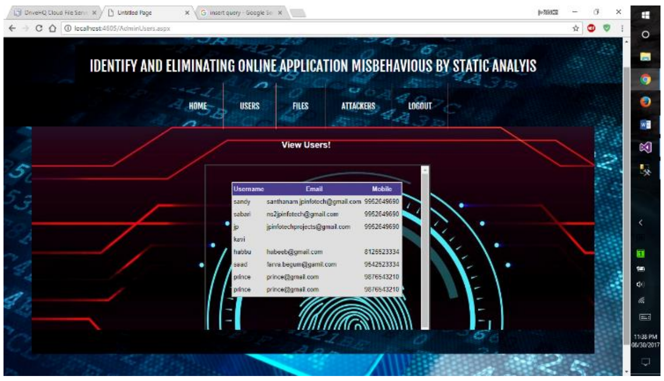
Output Screenshot 6: SQL inj insert query

If user wants to insert data intentionally into the database, the user will get blocked.



Output Screenshot 7: Admin login page

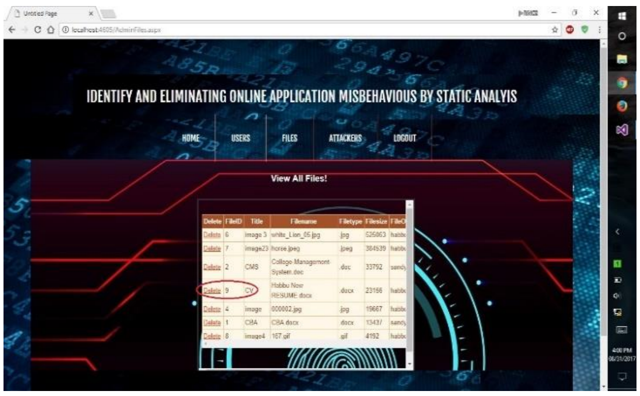
Admin login page have basic information like username and password.


IDENTIFY AND ELIMINATING ONLINE APPLICATION MISBEHAVIOUS BY STATIC ANALYIS
HOME
USERS
FILES
ATTACKERS
LOGOUT
View Users!

| Username | Email | Mobile |
|---|---|---|
| sandy | santhanam.jpinfotech@gmail.com | 9952649690 |
| sabari | ns2jpinfotech@gmail.com | 9952649690 |
| jp | jpinfotechprojects@gmail.com | 9952649690 |
| kavi | | |
| habbu | habeeb@gmail.com | 8126523334 |
| saad | farva.begum@gamil.com | 9542523334 |
| prince | prince@gmail.com | 9876543210 |
| prince | prince@gmail.com | 9876543210 |

Output Screenshot 8: Admin Users page

Admin is the owner of all the users, here all the users will be displayed with details.



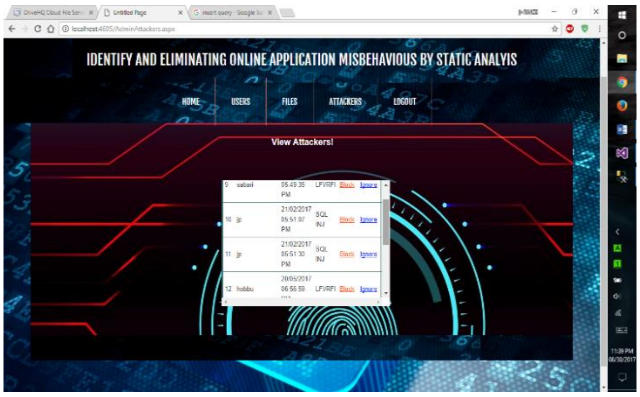
IDENTIFY AND ELIMINATING ONLINE APPLICATION MISBEHAVIOUS BY STATIC ANALYIS

View All Files!

| Delete | FileID | Title | Filename | Filetype | Filesize | FileO |
|---|---|---|---|---|---|---|
| Delete | 6 | image 3 | white_Lion_05.jpg | .jpg | 525063 | habb |
| Delete | 7 | image23 | horse.jpeg | .jpeg | 384539 | habb |
| Delete | 2 | CMS | College Management System.doc | .doc | 33792 | sand |
| Delete | 9 | CV | Hobbu New RESUME.docx | .docx | 23156 | habb |
| Delete | 4 | image | 000002.jpg | .jpg | 19667 | habb |
| Delete | 1 | CBA | CBA.docx | .docx | 13437 | sand |
| Delete | 8 | image4 | 167.gif | .gif | 4192 | habb |

Output Screenshot 9: Files deleted page

Admin is authorized to delete the selected file.



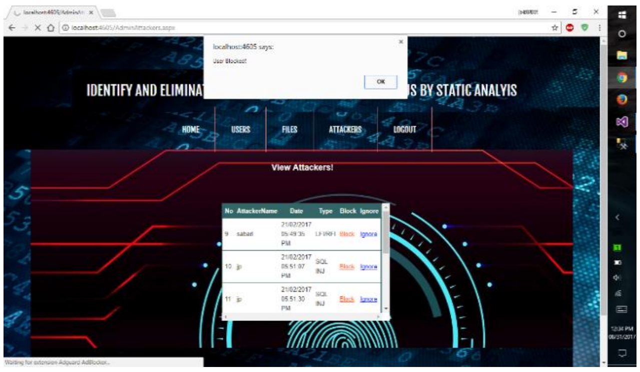
IDENTIFY AND ELIMINATING ONLINE APPLICATION MISBEHAVIOUS BY STATIC ANALYIS

View Attackers!

| | | | | | |
|---|---|---|---|---|---|
| 9 | sabari | 05.49.35 PM | LFI/RFI | Block | Ignore |
| 10 | jp | 21/02/2017 05:51:07 PM | SQL INJ | Block | Ignore |
| 11 | jp | 21/02/2017 05:51:30 PM | SQL INJ | Block | Ignore |
| 12 | habbu | 20/05/2017 06:56:59 | LFI/RFI | Block | Ignore |

Output Screenshot 10: Admin Attackers page

All the users who uploaded the source code file will get displayed in the Admin Attackers page.



localhost:4605 says:

User Blocked!

IDENTIFY AND ELIMINA S BY STATIC ANALYIS

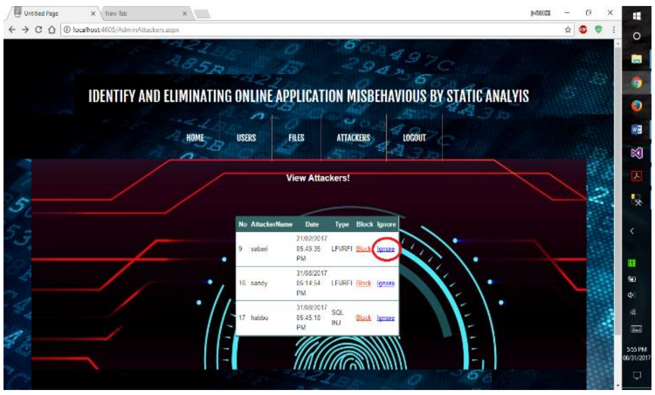
View Attackers!

| No | AttackerName | Date | Type | Block | Ignore |
|---|---|---|---|---|---|
| 9 | sabari | 21/02/2017 05:49:35 PM | LFI/RFI | Block | Ignore |
| 10 | jp | 21/02/2017 05:51:07 PM | SQL INJ | Block | Ignore |
| 11 | jp | 21/02/2017 05:51:30 PM | SQL INJ | Block | Ignore |

localhost:4605/AdminAttackers.aspx
OK
HOME
USERS
FILES
ATTACKERS
LOGOUT
Waiting for extension Adguard AdBlocker...

Output Screenshot 11: Blocked User

In this page admin is authorized to block the users who had uploaded unwanted data or inserting query in the database.



IDENTIFY AND ELIMINATING ONLINE APPLICATION MISBEHAVIOUS BY STATIC ANALYIS

View Attackers!

| No | AttackerName | Date | Type | Block | Ignore |
|---|---|---|---|---|---|
| 9 | sabari | 21/02/2017 05.49.35 PM | LFI/RFI | Block | Ignore |
| 16 | sandy | 31/08/2017 05:14:54 PM | LFI/RFI | Block | Ignore |
| 17 | habbu | 31/08/2017 05.45.18 PM | SQL INJ | Block | Ignore |

Output Screenshot 12: Ignored User

In this page admin is authorized to ignore the users who had uploaded the file or inserted query by mistake or unknowingly (as shown in the red circle).

## V. CONCLUSION

This work presents our way was executed not past the WAP device, more distant an experiential evaluation was performed having a gigantic order of PHP applications. This approach contributes clear to the flexibility of web applications by removing vulnerabilities, & not openly permitting the software engineer review from their oversights. take agent assemble of vulnerabilities acknowledged all sloppy investigator, twofold check on the off chance that they're malevolent picture or under different conditions, separate a few qualities, evaluate their work simple with the nearness of a mistaken viable, measure successor classifiers to pick the right in any case in constrain, and recognize the parameters from the classifier. This last viewpoint is empowered by inserting fixes like the back to back acknowledged opportunity process rehearses, so nerd can hear the specific practices by looking the vulnerabilities, and exactly how the specific were unapproachable. WAP additionally blights assessment and pen name for finding vulnerabilities, still it goes again by likewise adjusting the code.

## VI. REFERENCES

[1]. G. T. Buehrer, B. W. Weide, and P. Sivilotti, "Using parse tree validation to prevent SQL injection attacks," in Proc. 5th Int. Workshop Software Engineering and Middleware, Sep. 2005, pp. 106-113.

[2]. N. Jovanovic, C. Kruegel, and E. Kirda, "Precise alias analysis for static detection of web

application vulnerabilities," inProc. 2006Workshop Programming Languages and Analysis for Security, Jun. 2006, pp. 27-36.

[3]. G. Wassermann and Z. Su, "Sound and precise analysis of web applications for injection vulnerabilities," in Proc. 28th ACM SIGPLAN Conf. Programming Language Design and Implementation, 2007, pp. 32-41.

[4]. E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," J. Syst. Softw., vol. 83, no. 1, pp. 2-17, 2010.

[5]. L. K. Shar and H. B. K. Tan, "Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities," in Proc. 34th Int. Conf. Software Engineering, 2012, pp. 1293-1296.

[6]. T. Pietraszek and C. V. Berghe, "Defending against injection attacks through context-sensitive string evaluation," in Proc. 8th Int. Conf. Recent Advances in Intrusion Detection, 2005, pp. 124-145.

[7]. S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," IEEE Trans. Softw. Eng., vol. 34, no. 4, pp. 485-496, 2008.