WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Website Security Using Plugin

Priyanka N. Bhosale, Ruchita K. Mali, Aishwarya D. Jadhav, Vikas A. Singh, Prof. Sujeet More

UBachelor of EngineeringDepartment of Information Technology,Trinity College of Engineering & Research (TCOER), Pune

ABSTRACT:

In an attempt to support customization, many web applications allow the integration of third-party server-side plugins that offer diverse functionality, but also open an additional door for security vulnerabilities. In this paper we study the use of static code analysis tools to detect vulnerabilities in the plugins of the web application. The goal is twofold: 1) to study the effectiveness of static analysis on the detection of web application plugin vulnerabilities, and 2) to understand the potential impact of those plugins in the security of the core web application. We use two static code analysers to evaluate a large number of plugins for a widely used Content Management System. Results show that many plugins that are currently deployed worldwide have dangerous Cross Site Scripting and SQL Injection vulnerabilities that can be easily exploited, and that even widely used static analysis tools may present disappointing vulnerability coverage and false positive rates.

Introduction:

In a trial to support customization, several net applications permit the mixing of third-party server-side introduce that provide various practicality, however additionally open an extra door for security vulnerabilities. During this paper we have a tendency to study the utilization of static code analysis tools to discover vulnerabilities within the plug-in of the net application.

Weather The goal is twofold:

1) To review the effectiveness of static analysis on the detection of net application introduce vulnerabilities, and
2) To know the potential impact of these plug-in within the security of the core net application. we have a tendency to use 2 static code analysers to judge an outsized range of plug-in for a wide used Content Management System..

Results show that several plug-in that area unit presently deployed worldwide have dangerous Cross web site Scripting and SQL Injection vulnerabilities which will be simply exploited, which even wide used static analysis tools could gift unsatisfactory vulnerability coverage and false positive rates.

In This System our approach is to create registration of user when registration user will login to system when login user will enter the URL of web site and scrap the web site am fond of it can retrieve the all security plug-in from that URL.

Web scraping is the process of extracting and creating a structured representation of data from a web site. A company may for instance want to autonomously

Monitor its competitor's product prices, or an enterprising student may want to unify information on parties from all campus bar and dormitory web sites and present them in a calendar on her own web site. If the owner of the information does not provide an open API, the remedy is to write a program that targets the markup of the web page. A common approach is to parse the web page to a tree representation and evaluate an XPath expression on it. An XPath denotes a path, possibly with wildcards, and when evaluated on a tree, the result is the set of nodes at the end of any occurrence of the path in the tree. HTML, the mark-up language used to structure data on web pages, is intended for creating a visually appealing interface for humans. The drawback of the existing techniques used for web scraping is that the markup is subject to change either because the web site is highly dynamic or simply because the look-and-feel is updated. Even XPaths with wildcards are vulnerable to these changes because a given change may be to a tag which cannot be covered by a wildcard. In thisis we show how to perform web scraping using approximate tree Pattern matching. A commonly used measure for tree similarity is the tree edits distance which easily can be extended to be a measure of how well a pattern can be matched in a tree. An obstacle for this approach is its time complexity, so we consider if faster algorithms for constrained tree edit distances are usable.

For web scraping and we develop algorithms and heuristics to reduce the size of the tree representing the web page.

The aim of the project is to a develop a solution for web scraping that is

- Tolerant towards as many changes in the mark-up as possible,
- Fast enough to be used in e.g. a web service where response time is crucial, and
- Pose no constraints on the pattern, i.e. any well-formed HTML snippet should be usable as pattern.

Software Requirements:

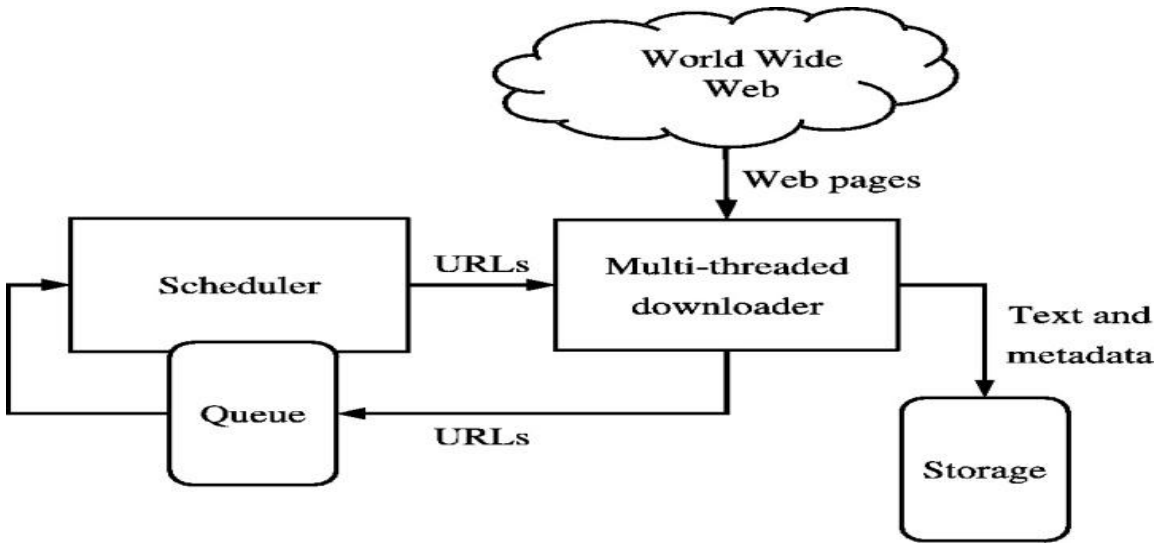
- Microsoft Visual Studio
- Heidi SQL for My SQL

Hardware Requirement:

- Laptop with Basic Hardware
- Keyboard
- Mouse

Problem Statement:

Finding the current security plug in of the website and suggesting the more suggestion for the improving the security of website.
Finding the current security plug in of the website and suggesting the more suggestion for the improving the security of website



Existing Architecture:

Fig.5.1 Existing Architecture

Existing System Algorithm:

- VLDC String Matching Algorithm
- Pattern Matching Algorithm

VLDC String Matching Algorithm:

String matching (SM) problem is to find the occurrences of a pattern within a text. A vanable length don't care (VLDC) is a special symbol, not belonging to a finite alphabet Σ but in Σ^*. Each VLDC in the pattern can match any substring in the text. Given a run-length coded text of length 2n over Σ and a run-length coded pattern of length 2m over Σ^*, this paper first presents an O(1) time parallel SM algorithm for run-length coded strings with VLDCs on a reconfigurable mesh (RM) using O(nm) processors. Consider the hardware limitation in VLSI implementation. In order to be suitable for VLSI modular implementation, a partitionable parallel algorithm on the RM with limited processors is further presented. For N < n and M < m, the SM for run-length coded strings with VLDCs can be solved in O(X^Y^) time on the RM using O(NM)(= O((nm)/((X^Y^))) processors, where X^ = [(n – 1)/(N – 1)] and Y^ = [(m – 1)/(M – 1)].

A basic search operation on patterns is the string matching (SM). In many applications, using a special encoding method for representing strings is important and advantageous for saving storage and manipulating them. One well-known method that has been widely used in many fields and has played a valuable historical role in the development of data compression is run-length coding. The basic idea of this method is to replace sequences of identical consecutive symbols with that representative symbol and its multiplicity. For example, the run-length coded representation of the string

aaaabbbaaacccc is a4b a3c4 (a, 4) (b, 3) (a, 3) (c, 4), and the length is reduced from 14 to 8. A variable length don't care (VLDC) is a special symbol, not belonging to E but in E* Each VLDC in the pattern can match any substring in the text (possibly zero length). For example, given a text string 'cccaaaaabbaaabbbccccddaabb' and a pattern string 'aabb.cccddaa', where is the VLDC, the two matched positions are from 7 to 24 and from 12 to 24. The SM problem for run-length coded strings with VLDCs can be viewed as an extension of the classical SM problem and has many important applications [4] such as editing operations, pattern recognition, file retrieval, DNA matching, etc.

Pattern Matching Algorithm:

Web Scraping is the process of extracting content from human-readable websites in order to import it into local storage such as databases or CSV Files. The process of data extraction and its design is time-consuming requiring an analysis of the website, data representation of the objects comprising its structure (DOM), HTML tags, and the Cascading Style Sheets (CSS) classes. To support this process we aim at providing automation. A pattern mining technique to scrap websites by recognizing title and body based on a content structure pattern. This approach consists of three steps, i.e.: extracting new website structure, constructing a pattern of HTML content, and implementing the pattern as a set of rules in web scraping. Our approach is a simple, general, and straightforward way to extract articles that consist of the title, the body of any blogs, or news websites. Pattern matching, in its classical form, involves the use of one-dimensional string matching. Patterns are either tree structures or sequences. There are different classes of programming languages and machines which make use of pattern matching. In the case of machines, the major classifications include deterministic finite state automata, deterministic pushdown automata, nondeterministic pushdown automata and Turing machines. Regular programming languages make use of regular expressions for pattern matching. Tree patterns are also used in certain programming languages like Haskell as a tool to process data based on the structure. Compared to regular expressions, tree patterns lack simplicity and efficiency.

There are many applications for pattern matching in computer science. High-level language compilers make use of pattern matching in order to parse source files to determine if they are syntactically correct. In programming languages and applications, pattern matching is used in identifying the matching pattern or substituting the matching pattern with another token sequence.

String matching algorithms have greatly influenced computer science and play an essential role in various real-world problems. It helps in performing time-efficient tasks in multiple domains. These algorithms are useful in the case of searching a string within another string. String matching is also used in the Database schema, Network systems.

Let us look at a few string matching algorithms before proceeding to their applications in real world. String Matching Algorithms can broadly be classified into two types of algorithms –

1. Exact String Matching Algorithms
2. Approximate String Matching Algorithms.

Implementation Diagram:

Analysis Model:

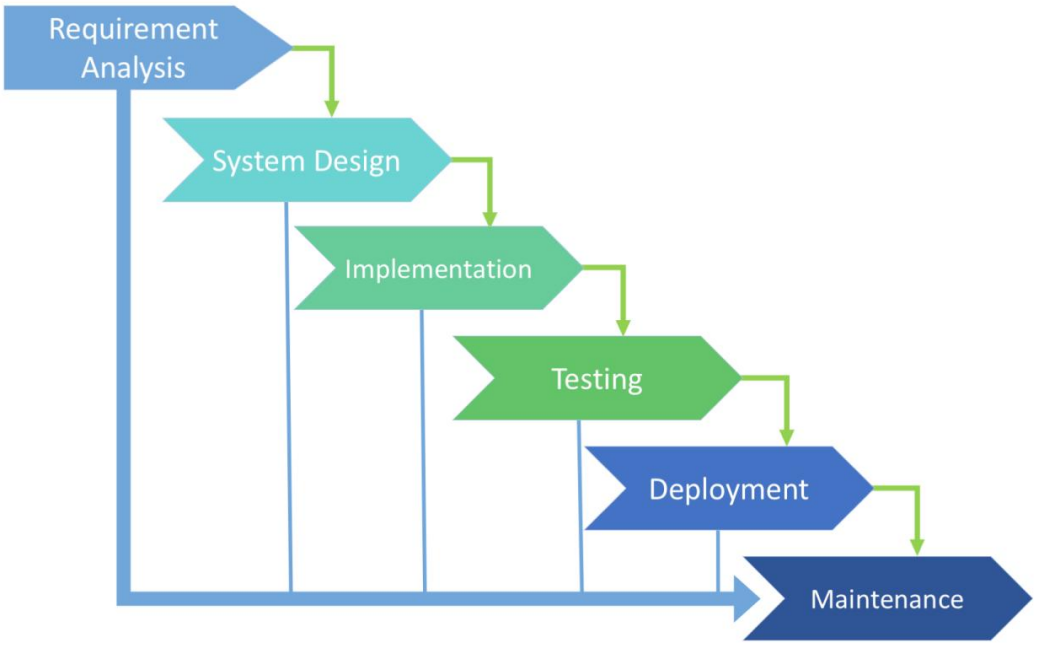
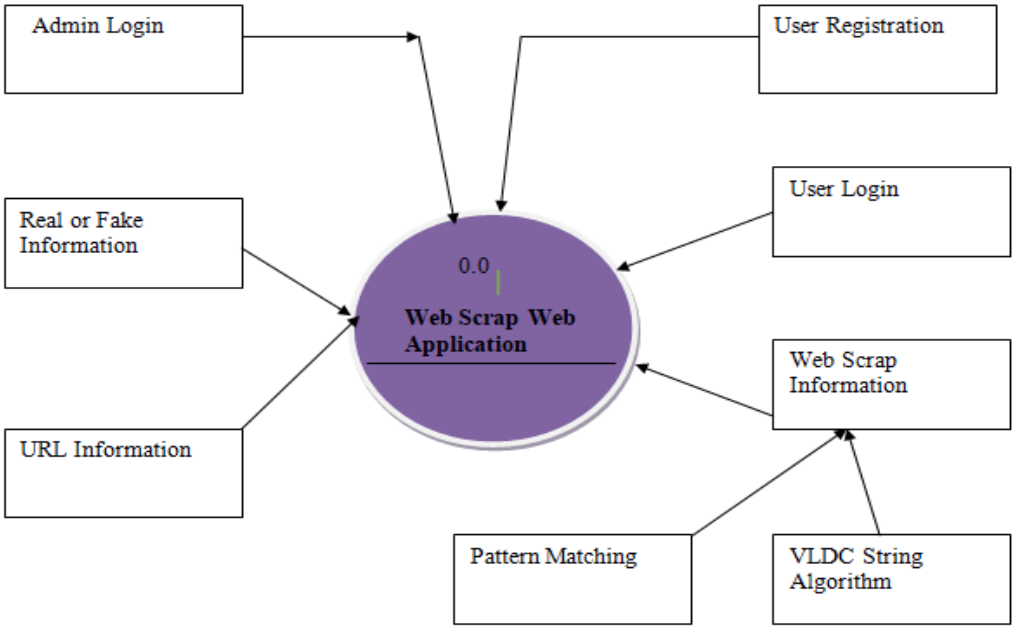


Fig.6.1 Analysis Model



Data Flow Diagram:

Fig6.2 Data Flow Diagram

ER Diagram:

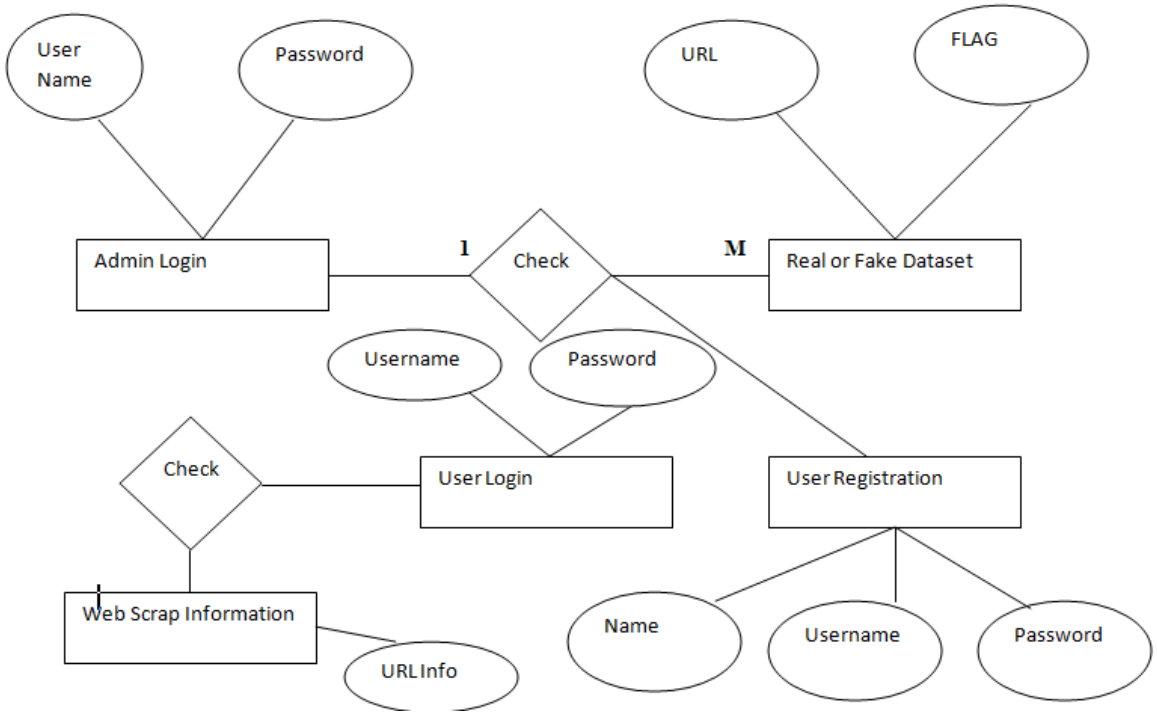
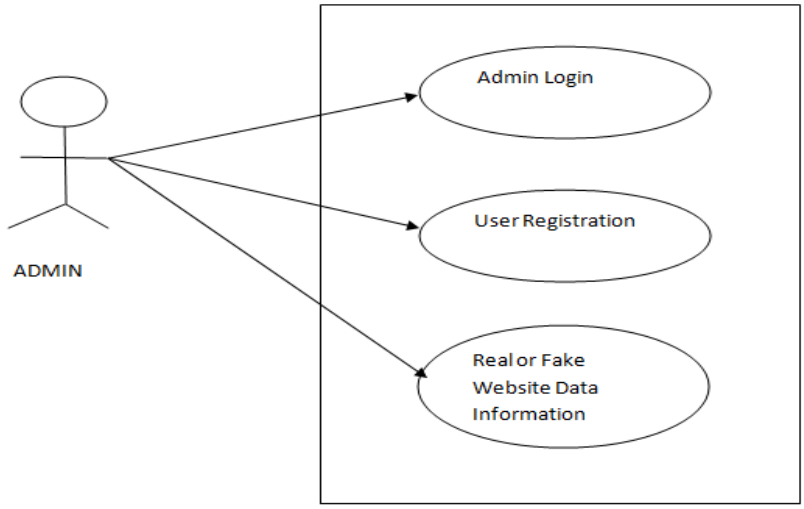
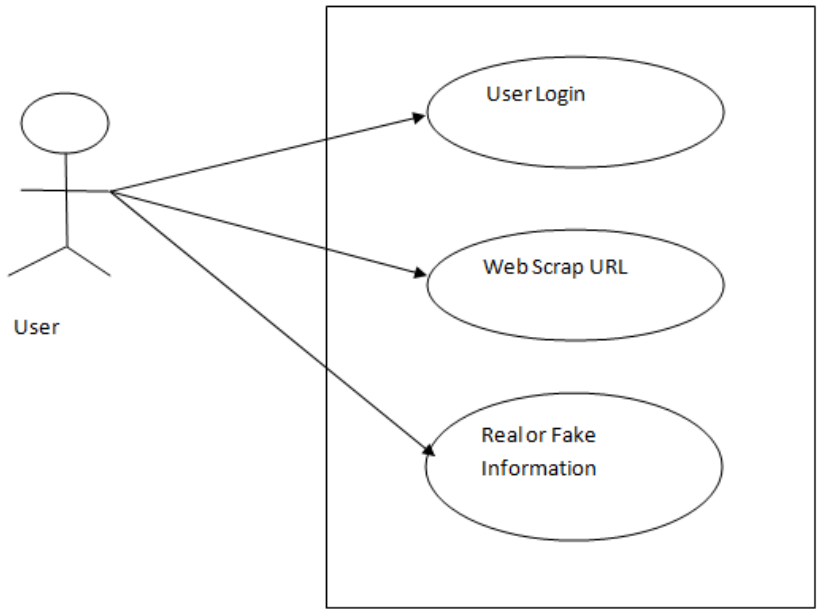


Fig.6.3 ER Diagram



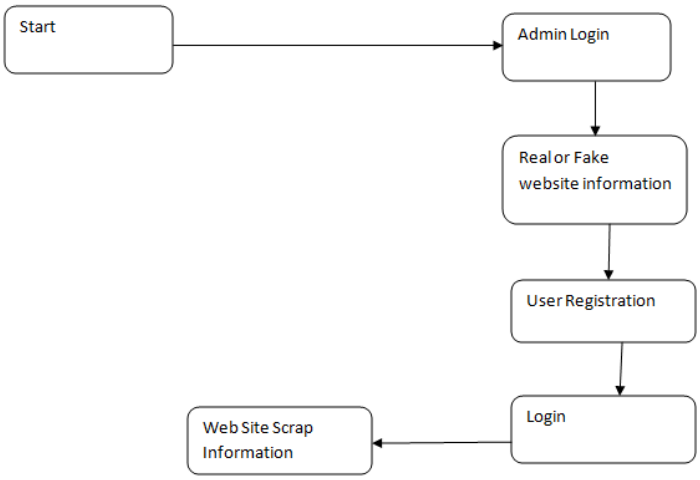
Admin Use Case Diagram:

Fig.6.4 Admin Use Case Diagram



User Use Case Diagram

Fig.6.5 User Use Case Diagram



State Chart Diagram

Fig.6.6 State Chart Diagram

Component Diagram

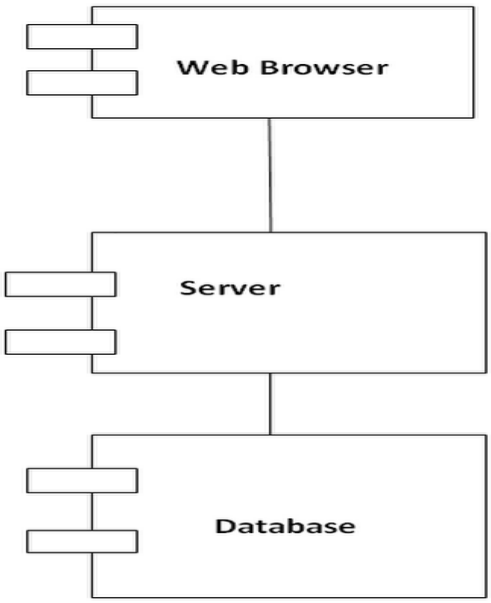


Fig.6.7 Component Diagram

Deployment Diagram

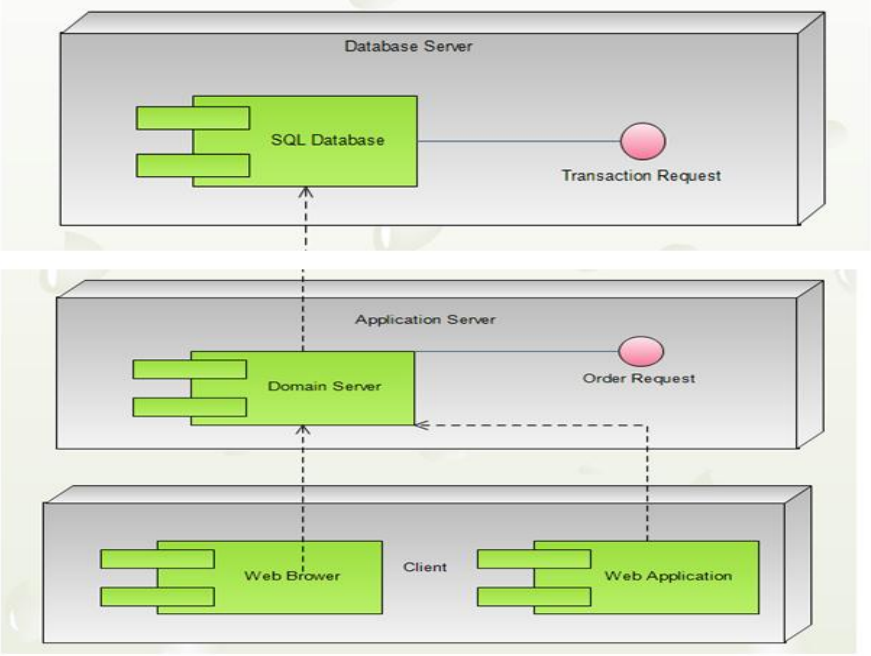


Fig.6.8. Deployment diagram

System Implantation

METHODOLOGY USED:

- Now a days as we can clearly see that hacker's attack are increasing day by day and it is causing threat in people, So we decided to build security plug in to overcome this to some extent Our main motto behind building security plug in is to provide security and threat free access to people And also it will be add on to our knowledge to learn many new things.
- Here we extract the security plug in from the URL and suggest additional plug-in for that URL

1. Preparation of the experiments: create the conditions for running the static analyzers on top of relevant plugins. Two steps are needed:

a. Identify a representative web application that allows the integration of plugins, and select a large set of widely used plugins for that application;

b. Decide on the types of vulnerabilities to be the target of the study and select representative static analyzers able to detect those vulnerabilities;

2. Execution of the static code analyzers: analyze the plugins using the tools. This includes two steps, whose results are later processed and compared:

a. Perform a generic analysis of the plugins using the typical configuration of the analyzers, i.e., not taking into account the fact that the target files are plugins for a specific web application;

b. Run a targeted analysis in which the configuration of the analyzers is tuned for the specific context of the target web application;

The correlation of the results from these two steps al- lows studying the performance of the static code analyzers in detecting vulnerabilities when they are configured for the specific context of the web application plugins and when not, with respect to two key figures of merit: coverage and false positives. This may give insights to how these tools should evolve in the future, e.g., helping to understand whether the tools should be explicitly pre- pared to handle the analysis of plugins or just need to follow a more generic approach.

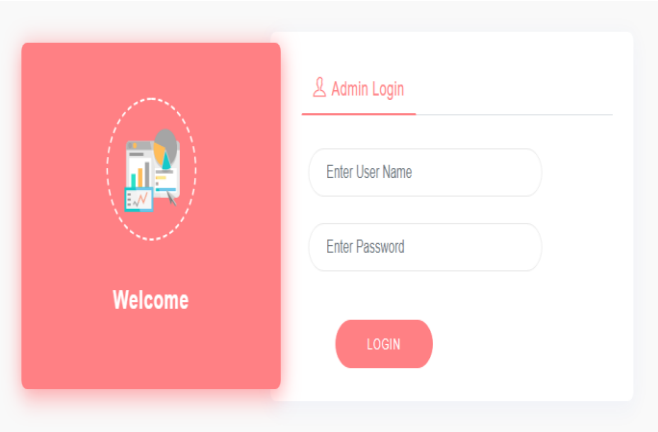
Analysis of the results:

Collect the reports of the tools and process the information gathered. This includes two steps:

a. Manual verification of the vulnerabilities reported to confirm the true vulnerabilities and discard the false positives;

b. b.Analysis of data to understand the impact of plugins in the application security and study the relative effectiveness, strengths and weaknesses, of the static analyzer tools. confirm the true vulnerabilities and discard the false positives;

c. Analysis of data to understand the impact of plugins in the application security and study the relative effectiveness, strengths and weaknesses, of the static analyzer tools.



Outcomes:

Fig 8.1 Admin Login

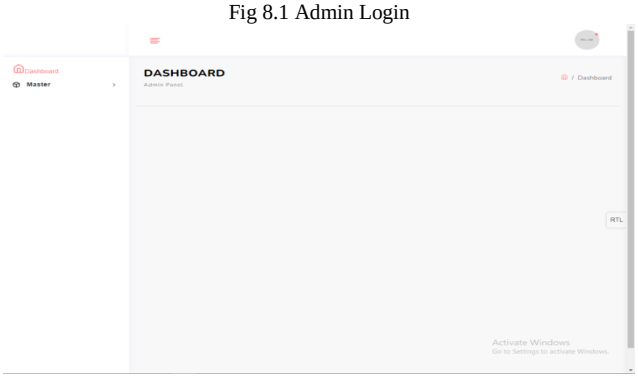


Fig. 8.2 Dashboard

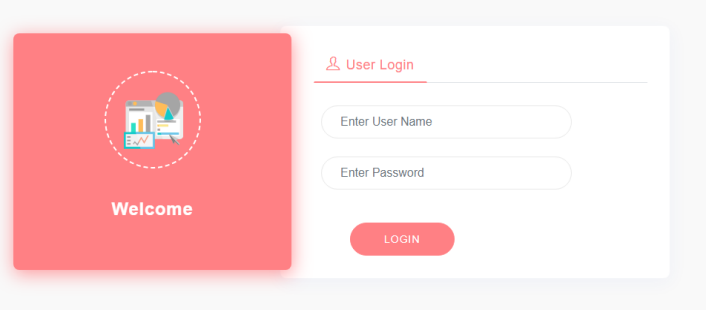


Fig 8.3 User Login

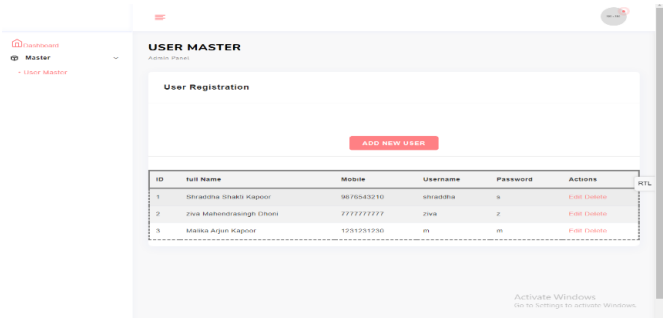


Fig8.4 User Registration

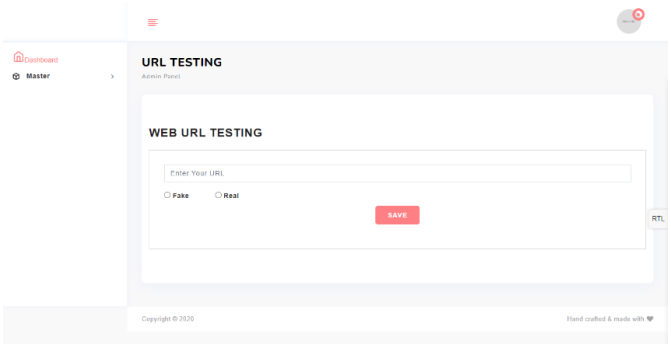


Fig 8.5 Real or fake information adding

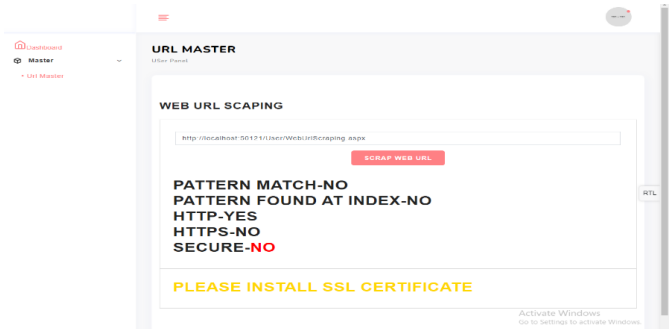


Fig 8.6 Pattern Matching-NOT

Conclusion

The main goal of this article was to explain how to use web scraping techniques to gather data from the web and display it in a meaningful way. We were able to accomplish this goal by using data from URL to create a meaningful suggestions page personalized to our needs. It only took a few hours to create and will save a great deal of loading time (especially on mobile devices) over the course of a year. In fact, if you checked the URL only once per day and each page took 10 seconds to load, you would have saved yourself 4 hours over the course of a year. Once again, we see that taking the time to learn how to write software can pay off in really big ways!

References:

List all the material used from various sources for making this project proposal

Research Papers:

1. Krotov, V., Silva, L. (2018). Legality and ethics of web scraping.
2. Parikh, K., Singh, D., Yadav, D., &Rathod, M. (2018). DETECTION OF WEB SCRAPING USING MACHINE LEARNING.
3. Thelwall, M. (2001). A web crawler design for data mining. Journal of Information Science, 27(5), 319-325.
4. Wed, J. (n.d.). Can Scraping Non-Infringing Content Become Copyright Infringement... Because Of How Scrapers Work? Retrieved October 20, 2020, from https://www.techdirt.com/articles/20090605/2228205147.shtml
5. SysNucleus. (n.d.). WebHarvy Web Scraper. Retrieved October 20, 2020, from https://www.webharvy.com/articles/what-is-web-scraping.html
6. Cooley, R., Mobasher, B., Srivastava, J. (1997, November).Web mining: Information and pattern discovery on the world wide web. In Proceedings ninth IEEE international conference on tools with articial intelligence (pp. 558-567). IEEE
7. D. Stuttard, and M. Pinto, "The Web Application Hacker's Hand- book: Discovering and Exploiting Security Flaws", Wiley, 2007
8. ESA, "ESA Guide for Independent Software Verification &Valida- tion", ftp://ftp.estec.esa.nl/pub/wm/anonymous/wme/ecss/ESAISVVGuideI ssue2.029dec2008.pdf, visited in November 2013
9. http://blog.sucuri.net/2013/04/wordpress-plugin-social-media- widget.html, visited in November 2013
10. http://codex.wordpress.org/Data_Validation, visited in November 2013
11. http://community.websense.com/blogs/securitylabs/archive/2012/03/0 2/mass-injection-of-wordpress-sites.aspx, visited in November 2013
12. http://en.wordpress.com/stats/, visited in November 2013
13. http://seclists.org/fulldisclosure/2012/Dec/242, visited in November 2013
14. http://us3.php.net/manual/en, visited in November 2013
15. http://wp.tutsplus.com/tutorials/creative-coding/data-sanitization-and- validation-with-wordpress/, visited in November 2013
16. http://www.darkreading.com/database/hackers-timthumb-their-noses- at-vulnerab/231902162, visited in November 2013
17. https://github.com/nikic/PHP-Parser, visited in November 2013
18. IBM Global Technology Services, "IBM Internet Security Systems X-Force® 2010 Trend & Risk Report", IBM Corp., 2011

19. IEEE Computer Society, "1012-2004 - IEEE Standard for Software Verification and Validation", June 2005
20. J. Dahse, November 2013, "RIPS", http://rips- scanner.sourceforge.net/
21. J. Fonseca, M. Vieira, H. Madeira, "Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks", Pa- cific Rim International Symposium on Dependable Computing, De- cember 2007
22. J. Fonseca, M. Vieira, H. Madeira, "The Web Attacker Perspective – A Field Study", IEEE 21st International Symposium on Software Reliability Engineering, November 2010
23. J. Fonseca, M. Vieira, "Mapping Software Faults with Web Security Vulnerabilities", IEEE/IFIP Int. Conference on Dependable Systems and Networks, June 2008
24. J. Fonseca, November 2013, "phpSAFE", https://github.com/JoseCarlosFonseca/phpSAFE
25. K. Ivan, "Software vulnerability analysis", PhD Thesis, Purdue University, 1998
26. M. Howard, D. LeBlanc, Writing Secure Code, Microsoft Press, 2003
27. M. Howard, D. LeBlanc, and J. Viega, "19 Deadly Sins of Software Security: Programming Flaws and How to Fix Them", McGraw-Hill Osborne Media, 2005
28. N. Jovanovic, C. Kruegel, E. Kirda, "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities", IEEE symposium on security and privacy, pp. 258-263, 2006
29. N. Patwardhan, E. Siever, S. Spainhour, "Perl in a Nutshell", Second Edition, O'Reilly, ISBN 0-596-00241-6, December 1998
30. N. Sam, www.owasp.org/images/7/7d/Advanced_Topics_on_SQL_I njection_Protection.ppt, 2006
31. Netcraft, "Web Server Survey", http://news.netcraft.com, visited in November 2013
32. NSA, "Defense in depth", http://www.nsa.gov/ia/_files/support/defenseindepth.pdf, 2004
33. NTA, March, 2011, http://www.nta-monitor.com/posts/2011/03/01- tests_show_rise_in_number_of_vulnerabilities_affecting_web_applications_with_sql_injection_and_xss_most_common_flaws.html, vi- sited in May 2013
34. OWASP Foundation, "OWASP top 10", July 2010
35. R. Zhang, S. Huang, Z. Qi, H. Guan, "Static program analysis as- sisted dynamic taint tracking for software vulnerability discovery" Computers & Mathematics with Applications Journal, Vol. 63 Issue 2, pp. 469-480, January 2012
36. S. Christey, R. Martin, "Vulnerability Type Distributions in CVE",
37. Mitre report, May, 2007
38. S. Neuhaus, T. Zimmermann, "Security Trend Analysis with CVE Topic Models", International Symposium on Software Reliability Engineering, pp. 111-120, 2010
39. w3techs,
40. http://w3techs.com/technologies/overview/content_management/all/, visited in November 2013
41. w3techs,
42. http://w3techs.com/technologies/overview/programming_language/all
43. visited in November 2013