WEB APPLICATION RECOGNITION AND ERADICATING VULNERABILITIES IN DATA MINING WITH STATISTICAL ANALYSIS

[1]Dr. V. Bapuji, [2]B. Anvesh Kumar
[1]Associate Professor, [2]Assistant Professor, [1,2]Dept. of Master of Computers and Applications,
[1,2]Vaageswari College of Engineering, Karimnagar, Telangana.
E-Mail: [1]Bapuji.vala@gmail.com, [2]anveshbodddupalli@gmail.com

ABSTRACT

Although there has been significant research on online application security for more than ten years, the issue of web application security is still difficult to solve. Vulnerable source code, frequently written in dangerous languages like PHP, is a significant contributor to this issue. Code of origin Static analysis tools can help uncover vulnerabilities, but they frequently produce false positives and demand a lot of human work from programmers to patch the code. We investigate the usage of many techniques to find source code flaws with fewer false positives. In order to anticipate the existence of false positives, we integrate data mining with taint analysis, which identifies prospective vulnerabilities. This approach brings together two approaches that are apparently orthogonal: humans coding the knowledge about vulnerabilities (for taint analysis), joined with the seemingly orthogonal approach of automatically obtaining that knowledge (with machine learning, for data mining). Given this enhanced form of detection, we propose doing automatic code correction by inserting fixes in the source code. Our approach was implemented in the WAP tool, and an experimental evaluation was performed with a large set of PHP applications. Our tool found 388 vulnerabilities in 1.4 million lines of code. Its accuracy and precision were approximately 5% better than PhpMinerII's and 45% better than Pixy's.
Key words: CNN, RCNN, SSD, dataset, weapon detection.

INTRODUCTION

Agriculture is India's main source of welfare. Rainfall is necessary for agriculture to succeed. Additionally, it benefits water resources. The country's economy grows as a result of farmers being able to better manage their crops thanks to historical rainfall data. Precipitation forecasting is useful for avoiding flooding, which protects lives and property. Forecasting rainfall is difficult for meteorological experts because of variations in the timing and volume of precipitation. To develop a predictive model for precise rainfall, forecasting is one of the most difficult tasks for academics from a number of domains, including meteorological data mining, environmental machine learning, functional hydrology, and numerical forecasting. In these problems, a common question is how to infer the past predictions and make use of future predictions. A varietyof sub-processes are typically composed of the substantial process in rainfall. It is at times not promise to predict the precipitation correctlyby on its global system. Climate forecastingstands out for all countries around the globe in all the benefits and services provided by the meteorological department. The job is very complicated because it needs specific numbers and all signals are intimated without any assurance. Accurate precipitation forecasting has been an important issue in hydrological science as early notice of stern weather canhelp avoid natural disaster injuries and damage if prompt and accurate forecasts are made. The theory of the modular model and theintegrati2on of different models has recentlygained more interest in rainfall forecasting to address this challenge. A huge range of rainfall prediction methodologies is available in India. In India, there are two primary methods of forecasting rainfall. Regression, ArtificialNeural Network (ANN), Decision Tree algorithm, Fuzzy logic and team process of data handling are the majority frequently used computational methods used for weather forecasting The basic goal is to follow information rules and relationships while gaining intangible and potentially expensive knowledge. Artificial NN is a promising part ofthis wide field Rainfall prediction remains a serious concern and has attracted the attention of governments, industries, risk management entities, as well as the scientific community. Rainfall is a climatic factor that affects many human activities like agricultural production, construction, power generation, forestry and tourism, among others [1]. To this extent, rainfall prediction is essential since this variable is the one with the highest correlation with adverse natural events such as landslides, flooding, mass movements and avalanches. These incidents have affected society for years [2]. Therefore, having an appropriate approach for rainfall prediction makes it possible to take preventive and mitigation measures for these natural phenomena.

To solve this uncertainty, we used various machine learning techniques and models to make accurate and timely predictions. Thesepaper aims to provide end to end machine learning life cycle right from Data preprocessing to implementing models to evaluating them. Data preprocessing steps include imputing missing values, feature transformation, encoding categorical features, feature scaling and feature selection. We implemented models such as Logistic Regression, Decision Tree, K Nearest Neighbour, Rule-based and Ensembles. For evaluation purpose.

2. LITERATURE SURVEY

1) WASP: protecting web applications using positive tainting and syntax aware evaluation AUTHORS: W. Halfond, A. Orso, and P. Manolios

Many software systems have evolved to include a Web-based component that makes them available to the public via the Internet and can expose them to a variety of Web-based attacks. One of these attacks is SQL injection, which can give attackers unrestricted access to the databases that underlie Web applicationsand has become increasingly frequent and serious. This paper presents a new highly automated approach for protecting Webapplications against SQL injection that has both conceptual and practical advantages over most existing techniques. From a conceptualstandpoint, the approach is based on the novel idea of positive tainting and on the concept of syntax-aware evaluation. From a practical standpoint, our technique is precise and efficient, has minimal deployment requirements, and incurs a negligible performance overhead in most cases. We have implemented our techniques in the Web application SQL-injection preventer (WASP) tool, which we used to perform an empirical evaluation on a wide range of Web applicationsthat we subjected to a large and varied set of attacks and legitimate accesses. WASP was able to stop all of the otherwise successful attacks and did not generate any false positives.

2) Defending against injection attacksthrough context-sensitive string evaluation

AUTHORS: T. Pietraszek and C. V. Berghe Injection vulnerabilities pose a major threat to application-level security. Some of the more common types are SQL injection, cross- site scripting and shell injection vulnerabilities. Existing methods for defending against injection attacks, that is, attacks exploiting these vulnerabilities, rely heavily on the application developers and are therefore error- prone.

3) SigFree: A signature-free buffer overflowattack blocker

AUTHORS: X. Wang, C. Pan, P. Liu, and S. Zhu We propose SigFree, an online signature-free out-of-the-box application-layer method for blocking code-injection buffer overflow attack messages targeting at various Internet services such as Web service. Motivated by the observation that buffer overflow attacks typically contain executables whereas legitimate client requests never contain executables in most Internet services, SigFree blocks attacks by detecting the presence of code. Unlike the previous code detection algorithms, SigFree uses a new data-flowanalysis technique called code abstraction that is generic, fast, and hard for exploit code to evade.

4) Vulnerability removal with attackinjection

AUTHORS: J. Antunes, N. F. Neves, M. Correia, P. Verissimo the increasing reliance put on networked computer systems demands higher levels of dependability. This is even more relevant as new threats and forms of attack are constantly being revealed, compromising thesecurity of systems. This paper addresses this problem by presenting an attack injection methodology for the automatic discovery of vulnerabilities in software components. The proposed methodology, implemented in AJECT, follows an approach similar to hackers and security analysts to discover vulnerabilities in network-connected servers. AJECT uses a specification of the server's communication protocol and predefined test case generation algorithms to automatically create a large number of attacks. Then, while it injects these attacks through the network, it monitors the execution of the server in the target system and the responses returned to the clients. The observation of an unexpected behavior suggests the presence of a vulnerability thatwas triggered by some particular attack (or group of attacks). This attack can then be used to reproduce the anomaly and to assist the removal of the error. To assess the usefulness of this approach, several attack injection campaigns were performed with 16 publiclyavailable POP and IMAP servers. The results show that AJECT could effectively be used to locate vulnerabilities, even on well-known servers tested throughout the years.

5) Fast black-box testing of system recoverycode

AUTHORS: R. Banabic and G. Candea Fault injection---a key technique for testing the robustness of software systems---ends up rarelybeing used in practice, because it is labor- intensive and one needs to choose between performing random injections (which leads to poor coverage and low representativeness) or systematic testing (which takes a long time to wade through large fault spaces). As a result, testers of systems with high reliability requirements, such as MySQL, perform fault injection in an ad-hoc manner, using explicitly- coded injection statements in the base source code and manual triggering of failures.

Existing System

There is a large corpus of related work, so we just summarize the main areas by discussing representative papers, while leaving many others unreferenced to conserve space. Static analysis tools automate the auditing of code, either source, binary, or intermediate. Taint analysis tools like CQUAL and Splint(both for C code) use two qualifiers to annotatesource code: the *untainted* qualifier indicates either that a function or parameter returns trustworthy data (e.g., a sanitization function), or a parameter of a function requires trustworthy data (e.g., *mysql_query*). The *tainted* qualifier means that a function or a parameter returns non-trustworthy data (e.g., functions that read user input).

PROPOSED SYSTEM

This paper explores an approach for automatically protecting web applicationswhile keeping the programmer in the loop. The approach consists in analyzing the web application source code searching for input validation vulnerabilities, and inserting fixes in the same code to correct these flaws. The programmer is kept in the loop by being allowed to understand where the vulnerabilities were found, and how they were corrected. This approach contributes directly to the security of web applications by removing vulnerabilities, and indirectly by letting the programmers learn from their mistakes. This last aspect is enabled by inserting fixes that follow common security coding practices, so programmers can learn these practices by seeing the vulnerabilities, and how they were removed. We explore the use of a novel combination of methods to detect this type of vulnerability: static analysis with data mining. Static analysis is an effective mechanism to find vulnerabilities in source code, but tends to report many false positives (non-vulnerabilities) due to its undesirability

MODULE DESCRIPTIONS

Taint Analysis:

The taint analyzer is a static analysis tool that operates over an AST createdby a lexer and a parser, for PHP 5 in our case. In the beginning of the analysis, all*symbols* (variables, functions) are *untainted* unless they are an entry point. The tree walkers build a *tainted symbol table* (TST) in which every cell is a program statement from which we want to collect data. Each cell contains a subtree of the AST plus some data. For instance, for statement $x = $b + $c; the TST cell contains the subtree of the ASTthat represents the dependency of $x on

$b and $c. For each symbol, several data items are stored, e.g., the symbol name, the line number of the statement, and the taintedness.

Predicting False Positives

The static analysis problem is known to be related to Turing's halting problem, and therefore is undividable for non- trivial languages. In practice, this difficulty is solved by making only a partial analysis of some language constructs, leading static analysis tools to be unsound. In our approach, this problem can appear, for example, with string manipulation operations. For instance, it is unclear what to do to the state of a tainted string that is processed by operations that return a substring or concatenate it with another string. Both operations can untaint the string, but we cannot decide with complete certainty. We opted to let the string be tainted, which may lead to false positives but not false negatives.

Code Correction:

Our approach involves doing code correction automatically after the detection of the vulnerabilities is performed by the taint analyzer and the data mining component. The taint analyzer returns data about the vulnerability, including its class (e.g., SQLI), and the vulnerable slice of code. A fix is a call to a function that sanitizes or validates the data that reaches the sensitive sink. Sanitization involves modifying the data to neutralize dangerous Meta characters or metadata, if they are present. Validation involves checking the data, and executing the sensitive sink or not depending on this verification.

Testing:

Our fixes were designed to avoid modifying the (correct) behavior of the applications. So far, we witnessed no cases in which an application fixed by WAP started to function incorrectly, or that the fixes themselves worked incorrectly. However, to increase the confidence in this observation, we propose using software testing techniques to a program to determine for instance if the program in general contains errors, or if modifications to the program introduced errors. This verification is done by checking if these test cases produce incorrect or unexpected behavior or outputs. We use two software testing techniques for doing these two verifications, respectively: 1) program mutation, and

2) Regression testing

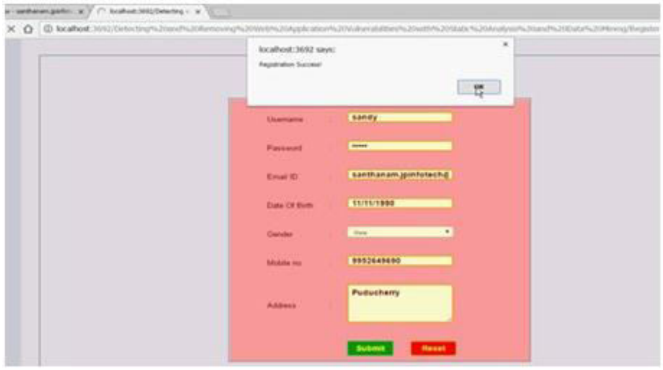

Fig.1. Login page

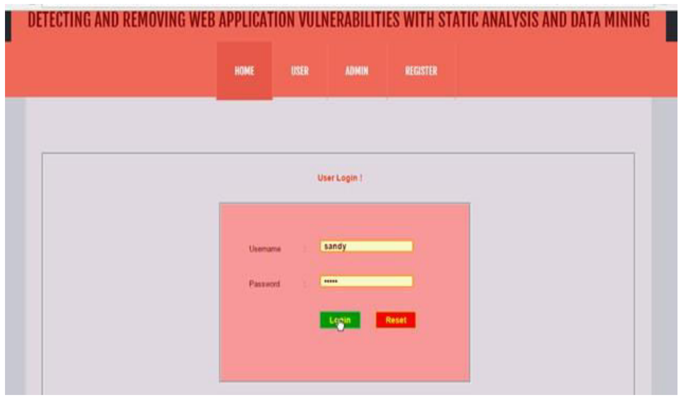

Fig.2. Home page

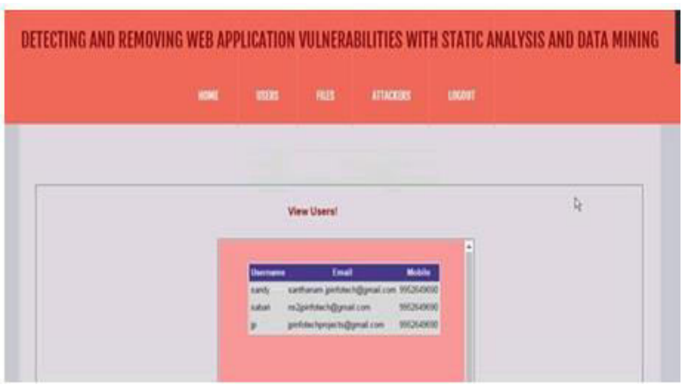

Fig.3. User page

Fig.4. Login code page

CONCLUSION

The method for identifying and fixing vulnerabilities in online applications is presented in this work, along with a tool that applies the method to PHP programmers and input validation flaws. Static source code analysis and data mining are used in the method and the tool to look for vulnerabilities. The top 3 machine learning classifiers are employed to identify false positives, and an induction rule classifier is utilized to confirm their existence. All classifiers were chosen after carefully weighing all available options. It's crucial to remember that this mix of detecting methods can't always produce accurate results. The static analysis problem is undividable, and resorting to data miningcannot circumvent this undesirability, but only provide probabilistic results. The tool corrects the code by inserting fixes, i.e., sanitization andvalidation functions. Testing is used to verify if the fixes actually remove the vulnerabilities and do not compromise the (correct) behavior of the applications. The tool was experimented with using synthetic code with vulnerabilities inserted on purpose, and with a considerable number of open source PHP applications. It was also compared with two source code analysis tools: Pixy, and PhpMinerII. This evaluation suggests that the tool can detect and correct the vulnerabilities of the classes it is programmed to handle. It was able to find 388 vulnerabilities in 1.4 million lines of code. Its accuracy and precision were approximately 5% better than PhpMinerII's, and 45% better thanPixy's.

REFERENCES

1. Symantec, Internet threat report. 2012 trends, vol. 18, Apr. 2013.
2. W. Halfond, A. Orso, and P. Manolios, “WASP: protecting web applications using positive tainting and syntax aware evaluation,” IEEE Trans. Softw. Eng., vol. 34, no. 1, pp. 65–81, 2008.
3. T. Pietraszek and C. V. Berghe, “Defendingagainst injection attacks through context- sensitive string evaluation,” in Proc. 8th Int. Conf. Recent Advances in Intrusion Detection, 2005, pp. 124–145.
4. X. Wang, C. Pan, P. Liu, and S. Zhu, “SigFree: A signature-free buffer overflow attack blocker,” in Proc. 15th USENIX SecuritySymp., Aug. 2006, pp. 225–240.
5. J. Antunes, N. F. Neves, M. Correia, P. Verissimo, and R. Neves, “Vulnerability removal with attack injection,” IEEE Trans. Softw. Eng., vol. 36, no. 3, pp. 357–370, 2010.
6. R. Banabic and G. Candea, “Fast black-boxtesting of system recovery code,” in Proc. 7th ACM Eur. Conf. Computer Systems, 2012, pp. 281–294.
7. Y.-W. Huang et al., “Web application security assessment by fault injection and behavior monitoring,” in Proc. 12th Int. Conf. World Wide Web, 2003, pp. 148–159.
8. Y.-W. Huang et al., “Securing web application code by static analysis and runtime protection,” in Proc. 13th Int. Conf. World Wide Web, 2004, pp. 40–52.
9. N. Jovanovic, C. Kruegel, and E. Kirda, “Precise alias analysis for static detection of web application vulnerabilities,” inProc. 2006Workshop Programming Languages andAnalysis for Security, Jun. 2006, pp. 27–36.
10. U. Shankar, K. Talwar, J. S. Foster, and D. Wagner, “Detecting format string vulnerabilities with type qualifiers,” in Proc. 10th USENIX Security Symp., Aug. 2001, vol. 10, pp. 16–16.