Journal of Engineering and Applied Sciences 13 (Special Issue 6): 5276-5285, 2018
ISSN: 1816-949X

A Hybrid Vulnerability Analysis Tool Using a Risk Evaluation Technique

[1]Jong-Chul Park, [1]Sang-Ann Nam,[2]Jae-Pyo Park and [2]Yeun-Soo Choo
[1]Department of IT Policy Management,
[2]Department of Information Security, Soongsil University, 07027 Seoul, Republic of Korea

Abstract: Recent there have been many efforts to detect and analyze vulnerabilities using diverse analysis tools, removing them at the development stage. However, vulnerability analysis tools are prone to missed detections incorrect detections and over detection which reduces the accuracy of detection. In this study, a vulnerability detection technique is proposed that develops and manages safe applications and can resolve and analyze these problems. Risks due to vulnerabilities are computed and an intelligent vulnerability detection technique is used to improve accuracy and evaluate risks of the final version of the application. This helps the development and execution of safe applications.Through incorporation of tools that use both static and dynamic analysis techniques our proposed technique overcomes weak points at each stage and improves the accuracy of vulnerability detection. Existing vulnerability risk evaluation system only evaluate self-risks while our proposed vulnerability risk evaluation system reflects vulnerability self-risk and detection accuracy in a complex fashion to evaluate relative. Our proposed technique compares and analyzes existing analysis tools such as lists for detections and detection accuracy based on the top 10 items of SANS at CWE. Quantitative evaluation systems for existing vulnerability risks and proposed application vulnerability risks are compared and analyzed. Through, incorporation of tools that use both static analysis and dynamic analysis techniques. We developed prototype analysis tool using our technique to test the application's vulnerability-detection ability and show our proposed technique is superior to existing ones.

Key words: Vulnerability, degree of risk, vulnerability detection, accuracy of detection, application, complex

INTRODUCTION

In order to defend against cyber attacks and minimize damages, many have adopted diverse security solutions for protection yet damage incidents and their scale has still grown. Among the reason for increased cyberattacks is the fact that about 75% are due to software's weak points (Allodi and Massacci, 2014). The heart bleed bug of open SSL an open source code which 2/3 of all global web sites are currently using is just one of the typical cases. Microsoft has stated that development cost can be saved for projects that develop security operation systems which consider security issues from the beginning, rather than ones that do not consider them from the beginning and develop security patches as time passes (Anonymous, 2016). Therefore, attempting to find weak points in the development phase of software is a prudent way to reduce cyber attacks and to lower development costs. As techniques to detect vulnerabilities, there are static module based on source codes and dynamic module based on execution systems. However, these methods are prone to false detection, missing detections, over detection and other problems (Mouzarani *et al*., 2016; Ransbotham and Mitra, 2013). In this study, a vulnerability analysis technique is proposed to help resolve these problems. Our proposed technique is designed with an interaction analysis to increase detection accuracies using existing methods. This increases accuracies of the results as its interaction results re-confirm prior results. Our method has five distinct analysis stages the results that were generated by each stage get analyzed using the dynamic and static modules twice which increases overall accuracy. All results generated in each stage get analyzed first and weights are applied to evaluate the final degree of risks of vulnerability to the application. The existing analysis methods are used to evaluate functions of the proposed vulnerability analysis technique this research compares and analyzes functions using these existing analysis methods. Therefore, our intelligent vulnerability analysis technique is proposed to enhance the accuracy of detection and its application for risk evaluation to evaluate an application vulnerability status by analyzing its weak points. Our proposed technique also has the advantage of automatically executing inspection activities that humans currently perform manually.

Corresponding Author: Jae-Pyo Park, Department of Information Security, Soongsil University, 07027 Seoul, Republic of Korea

MATERIALS AND METHODS

Vulnerability analysis technique: The static analysis technique uses source codes to analyze vulnerability that is inherited in the application without execution. The static analysis technique is not only used to detect vulnerability that is inherited in the completed applications but also used to test patterns, API and others that are confirmed as codes to possibly cause vulnerability from the development phase this allows to prevent occurrence of vulnerability. Therefore, the static analysis technique is used to maintain and minimize repair costs of completed applications. Therefore, it is widely used to maintain and lower repair costs for completed applications (Chen *et al*., 2013). One of common methods of static analysis techniques is pattern matching i.e., parsing that uses syntax analysis, lexical analysis and others methods. Pattern matching detects vulnerability using string patterns; string patterns use forms of unsafe functions, dangerous variable names, particular code patterns. Since, this method categorizes all patterns that are determined by rules to weak points for their detection, all footnotes are also detected. Therefore, its downside is the high possibility of over detections when compared to dynamic analysis techniques (Fang *et al*., 2014; Christey, 2011).

Other known static analysis techniques are 'Parsing' and 'Data Flow Analysis. The parsing classifies and interprets source code and source code gets expressed with abstract syntax tree. This parsing tree not only construct program but also analyze meanings. In order to accurately parsing and analyze diverse program, static analysis technique requires one key complier and compatible parse. The data flow analysis is a traditional complier technique to resolve 'Buffer Overflow' and 'Format String' issues. This technique can also be used to detect vulnerability. This technique collects information of possible values that can be calculated from diverse points. Easy way to execute data flow analysis is to set data-flow equations for nodes of control flow graph and calculate from input to output repeatedly until the whole system gets reached to stability (Castro *et al*., 2006).

The dynamic analysis technique is to analyze whether functions of application run properly or not data gets input into actual running application and outputted real results get analyzed for its vulnerability (Ernst, 2003). It is to test whether errors or unexpected results gets draw or not when the value gets input into application. Simply speaking when '1' and '2' gets input into the Add-function module, this dynamic analysis technique analyzes whether we get '3' as the result or not. At this time, if unintended 'letter character' gets input, dynamic analysis technique analyzes whether we get security problems or not. Dynamic analysis techniques can detect functional defects which static analysis technique cannot. Static analysis techniques detect vulnerabilities based on source codes; they cannot detect unexpected results during actual execution. However, dynamic analysis method's advantage is that all values are input for dummy purging cases this results in detecting all vulnerabilities that developers may not expect and relatively high inspection rates compared to static analysis methods. However, knowledge of input values and the many processes to detect many inputs are downsides in detecting vulnerabilities under this technique (Khurana *et al*., 2016; Khan and Khan, 2012).

International information for vulnerability management: ISO (International Organization for Standardization) 27005 defines cybersecurity weak points as a group of weaknesses that can be exploited by one or more threats. Also, the internet engineering task force defines weak points as defects and weaknesses from system designs, materializations, operations and management which can be exploited to violate system security policies (ISO., 2016). Weak points are known as security vulnerabilities to actual attacks among many weak points that exist in information assets in systems and solutions. These weak points go through confirmation, categorization, cure and a remedy process which requires routine management (Agawal and Singh, 2013). This type of systematic management of vulnerabilities is known as a vulnerability management system and all different vulnerabilities found from diverse agencies around the world are managed this way.

Application developers and users can make decisions of relevant application's adjustments and reuse, once it is known how dangerous applications arewhen they get to know where do managed vulnerability exists. Developers or users may not need to consider applicable vulnerability if vulnerability points do not significantly affect risks of applications in certain situations or conditions, even, if they were found using vulnerability analysis tools. However, a risk evaluation system is also required for vulnerability management systems because security patches or application adjustments are mandatory.

The Common Vulnerability and Exposures (CVE) is a database for cyber security weak points; it manages occurrences of those weak points by years, sequences and other characteristics. Many US government agencies and diverse companies such as Apple, Microsoft and others all participate in CVE; currently, the US Department of Homeland Security manages it. CVE was first created

by MITRE, a nonprofit organization in 1999 and the National Institute of Standards and Technology (NIST)'s coordination helped with the systematic managements. When weak points are detected, one can request a registration to MITRE, after which request the CVE management then discusses the candidate and potentially assigns a CVE-ID (FIRST., 2016).

The CWE/SANS 25 is the list that contains the most dangerous errors of software. MITRE under DOD and Software security experts from the US and Europe made CWE/SANS 25 in order to minimize damages that occur from severe errors of software. This list gets developed by categorizing top 25 vulnerability by assigning scores to vulnerability list provided by CWE using CWSS (Common Weakness Scoring System). The most recent list was released in 2011 and it gets sorted to 3 categories. These 3 categories are incomplete interaction among components such as "SQL injection", dangerous resource management such as "Buffer Overflow", porous defense system such as 'non-certify of important function's (Christy, 2017). Like this, 25-vulnerability list is the guideline for the safe software development and ways to minimize security damages due to software.

The Common Vulnerability Scoring System (CVSS) is a system to evaluate degree of risks to embedded vulnerabilities. Developed by Carnegie Mellon University and National Institute of Standards and Technology, it is currently managed by the Forum of Incident Response and Security Teams. The first official version was V2.0 presented in 2005; it was updated as V3.0 in June 2015 (Coley, 2016). CVSS has three evaluation items the basic group, time group and environment group. Each evaluation item has detailed evaluation tables to calculate weak points. The basic group evaluates general degrees of risk that are well-known. The time group and environment group are used under special circumstances for detected vulnerabilities from special time flow or other situations.

The Common Weakness Scoring System (CWSS) is the system that evaluates security vulnerabilities that software possesses quantitatively and the US government, academics and industry are all involved in its research. For quantitative evaluation, CWSS calculates scores of 3 sectors separately. The three sectors consist of the base finding metric group (a group that evaluates quantitative scores of the security vulnerability itself), the attack surface metric group (efforts that attackers need to put in during software attacks using the relevant security vulnerability) and the environmental metric group (environmental factors that alter the strength of the relevant security vulnerability). All 3 sectors have 16 detailed items, the midpoints of each sector get calculated and then those midpoints are multiplied to calculate the final CWSS value. The current CWSS is Version 1.0.1 and it was last updated in 2014.

RESULTS AND DISCUSSION

System architecture: Our proposed technique has 5 stages. The first stage is the basic analysis phase which detects vulnerability by using the static analysis tool and the dynamic analysis tool, these two tools minimize problems with the dynamic analysis tool through interaction. The second stage is the analysis stage of the basic vulnerability analysis results; it analyzes results from 1st stage, so that, results can be used during the third and fourth stages. The third stage is the generation of a customized rule set and in-depth test stage which executes an in-depth static analysis using information generated from second stage. Information used at this stage was analyzed from the second stage where it originated from the dynamic analysis module from the first stage. Results from the fourth stage go through in-depth tests after generating exploits using information from the second stage; this also went through the static analysis module of first stage. The relevant application's degree of risk gets evaluated by collecting results from the fifth stage. The fifth stage reports lists of detected vulnerabilities; the degree of risks is computed using detection accuracy of the vulnerability lists. From the above mentioned stages, results from the first stage static analysis module get analyzed once again at the in-depth test of the fourth stage to minimize over detections and incorrect detections of the static module of the first stage which increases detection accuracy.

Additionally, results from the dynamic analysis module of the first stage go through an in-depth test of the third stage to verify them at the source code. Both static and dynamic analysis modules then get cross checked to increase detection accuracy. This is the new technique that is proposed in this study. Moreover, using information of detected vulnerabilities, a new methodology that evaluates the level of dangers that applications are situated is also proposed. Figure 1 shows the general structure of the proposed intelligent vulnerability analysis technique structure.

Vulnerability analysis process

Basic vulnerability analysis stage: This stage analyzes application's weak points by using both static module and dynamic module. This stage is constructed as shown in Fig. 2. Source code collection (A) receives source codes of applications and prepare to perform static analysis. At this time, source code collection (A) sends

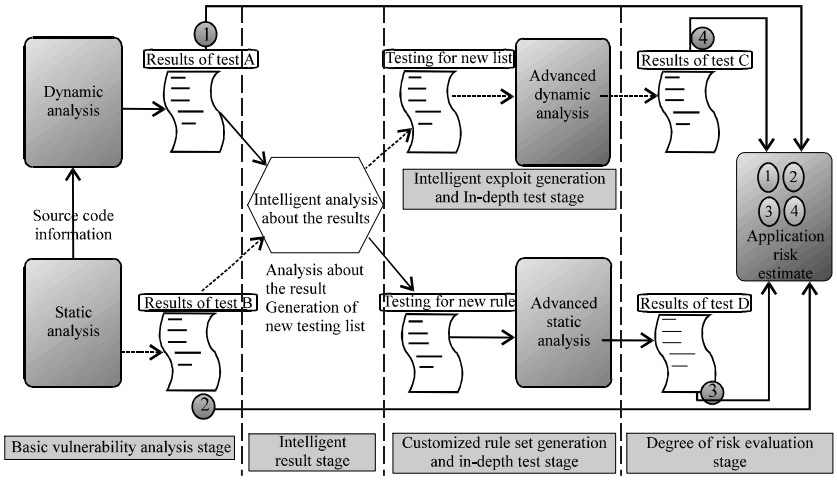


Fig. 1: Proposed vulnerability analysis technique structure

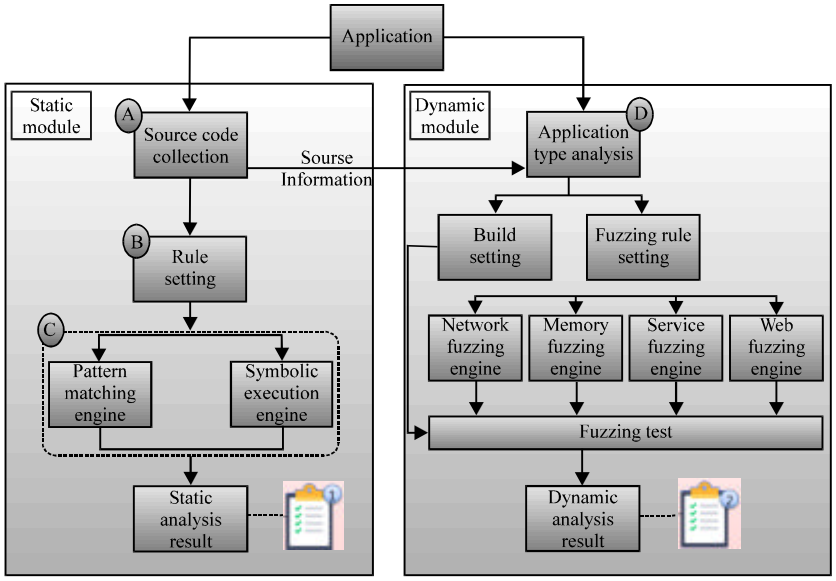


Fig. 2: Step 1; Process for basic vulnerability analysis

pre-collected information of source codes to application type analysis (D) of dynamic module to determine the type of application. Rule setting (B) sets rules for vulnerability that are to be analyzed and execute the rule testing at analysis engine (C).

Analysis of rules gets performed using pattern matching engine and symbolic execution engine. Application type analysis (D) of dynamic module uses information received from Source code collection (A) to analyze types such as applet, web application independent systems and others. Then, fuzzing rule for dynamic analysis and build setting for execution get set. Lastly, based on types of application, vulnerability list gets generated through basic dynamic tests using network fuzzing, memory fuzzing and other fuzzings.

Results stage for basic vulnerability analysis: The second stage of the intelligent vulnerability analysis technique, i.e., the analysis stage, analyzes results from the first stage and generates information for the third and fourth stage's weak-points analysis. Figure 3 illustrates the processes of second stage.

When results from the basic vulnerability weak-points analysis stage are sent to the second stage,it analyzes whether those results are detected from the same locations of the source codes or not. Due to the

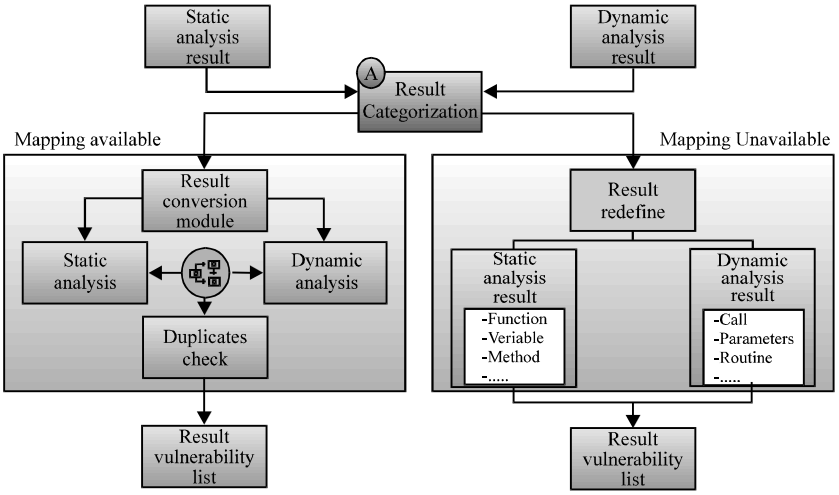


Fig. 3: Step 2; Analysis process for basic vulnerability analysis results

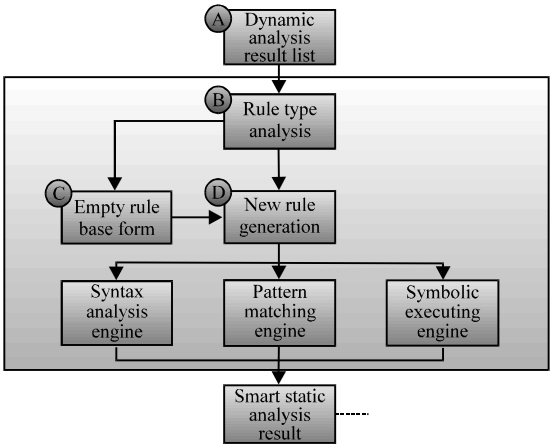


Fig. 4: Step 3: Process for customized rule set generation and in-depth test

characteristics of weak points, some cannot be detected using either static or dynamic analysis methods. Therefore, result categorization (A) develops a list of top 10 of the CWE/SANS 25 that both modules detect commonly and also a list that any one module can detect.

Customized rule set generation and in-depth test stage: A three staged customized rule set generation and In-depth test stage is the static analysis module that goes through in-depth tests to enhance accuracy of results from dynamic module received from the second stage. Figure 4 illustrates this three staged processes.

The vulnerability list (A) that comes to the third stage is the same as the ones that completed dynamic analysis but were not decided as the same causes with static analysis. From rule type analysis, (B) the vulnerability list gets analyzed and categorized as to which rule type it corresponds to. Separate from the existing 10 items, a new empty rule base form (C) gets generated. To re-check the delivered vulnerability lists, customized rules are generated from the new rule generation (D). At this stage in order to find common detection results between results from static analysis and dynamic analysis, detected vulnerability gets categorized. This stage is to find vulnerability list that is detected as common vulnerability from same code location of applications; except common lists, results of static analysis uses dynamic technique, results of dynamic analysis uses static technique to re-analyze vulnerability of application.

Exploit generation and in-depth test stage: Exploit generation and in-depth test stage is the fourth stage and it is opposite to the third stage. Vulnerability lists that were either mapped of not mapped from detection results of dynamic analysis module get transferred. These vulnerabilities were detected using the static analysis methods; there is a chance of them being due to false or over detection; it is required to re-check the results using dynamic analysis module to increase the detection accuracy. Figure 5 illustrates the four-stage process.

Exploit process (A) for each vulnerability is defined after analyzing the vulnerability list from second stage. Since, the dynamic module does its analysis during the execution of the application, codes that were detected with vulnerabilities can be executed at this stage. At this time, build information for the applications and application itself get executed first, then information which can reach towards vulnerability code gets delivered (B). After the process is defined, the exploit form is completed by

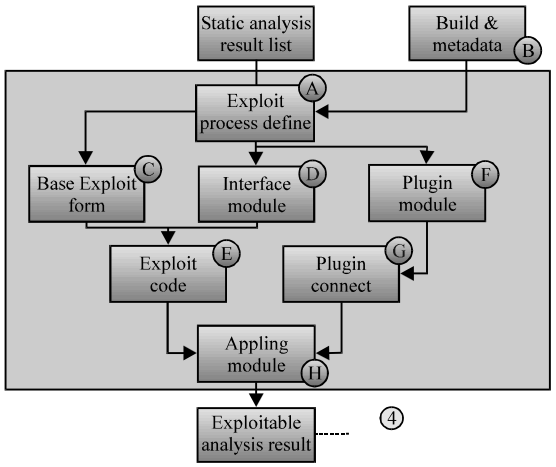


Fig. 5: Step 4; Exploit generation and in-depth test stage

sending information to the Base exploit form (C). Once, the exploit form is completed; the scripts and execution codes are linked at the interface module (D) to generate the exploit code (E). If security solutions (keyboard security module, ActiveX, etc.) are already running, the same environment is generated in the plugin module (F) and plugin connect (G). Exploit code prepared from the appling module (H) get applied to check whether the vulnerability is properly detected or not.

Stage for evaluation of degree of risk: Degree of risk evaluation stage is the stage where final degree of risk of application gets evaluated using results from 1st-4th stages. Final degree of risk gets finally evaluated using self-degree of risk of vulnerability detection lists from each stages and detected frequencies of accuracy of vulnerability and detected vulnerability from applications. Figure 6 shows how vulnerability individual risk score gets calculated by using phased vulnerability lists through 1-4 steps.

At this time, modified CWSS is used to score calculation. Detected lists by each step get applied with weights to calculate final risk score for relevant application. Modified CWSS, weights and final risk score calculation methods will be explained in section C.

Evaluation technique for degree of risk

Detection accuracy using proposed technique: Figure 7 describes the proposed technique. Vulnerability lists that were detected from each stage are indicated in numbers. After 1st stage, we can get ¨ç, ¨è results; we get intersection(①∩②) by going through 2nd stage. We can get differences(②-(①∩②)) from 3rd stage using specially set rules using static module.

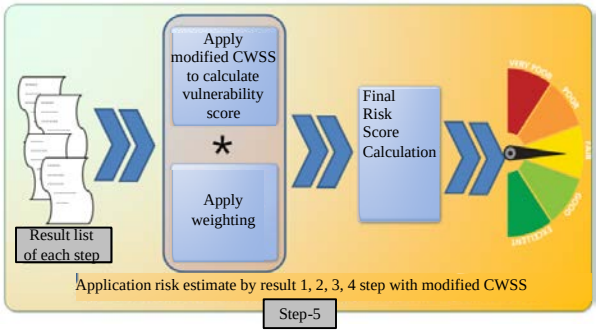


Fig. 6: Step-5: Process for evaluation of degree of risk

Table 1: Detection accuracy of results by stages

Results by stages	Accuracy	Probabilities (%)
①∩②	High	81
①-(①∩②)	Low	64
②-(①∩②)	Mid	69
③	Very high	88
④	Very high	88

Result ③ is assessed to have high accuracy than result ②. Moreover, results of differences (①-(①∩②)) from 4th stage goes through module analysis to enhance detection accuracy in results of ④. Accuracy that were detected by each stage is illustrated in Table 1. Test bed was structured first and results were drawn. Detection accuracy of each vulnerability is quite similar to ones that human analyzed manually.

Evaluation scores and mid scores of vulnerability self degree of risk: The vulnerability self degree of risk uses portions of the CWSS calculation logic. The CWSS's security vulnerability degree of risk score calculation uses the base finding metric group, attack surface metric group and environmental metric group. The environmental metric group applies external factors such as type of application, detection possibilities of the relevant vulnerability, degree of spread and other external causes. However in this study, the environmental metric group does not affect vulnerability significantly and it is removed from the vulnerability evaluation. Therefore in this study, only the base finding metric group and attack surface metric group are multiplied independently in order to evaluate the degree of risks of the vulnerability. Equation 2 illustrates the scoring methods of the base finding metric group:

Scores of degree of risk of vulnerability =
Base finding*Attack surface

Table 2 shows the abbreviations used in the following equations:

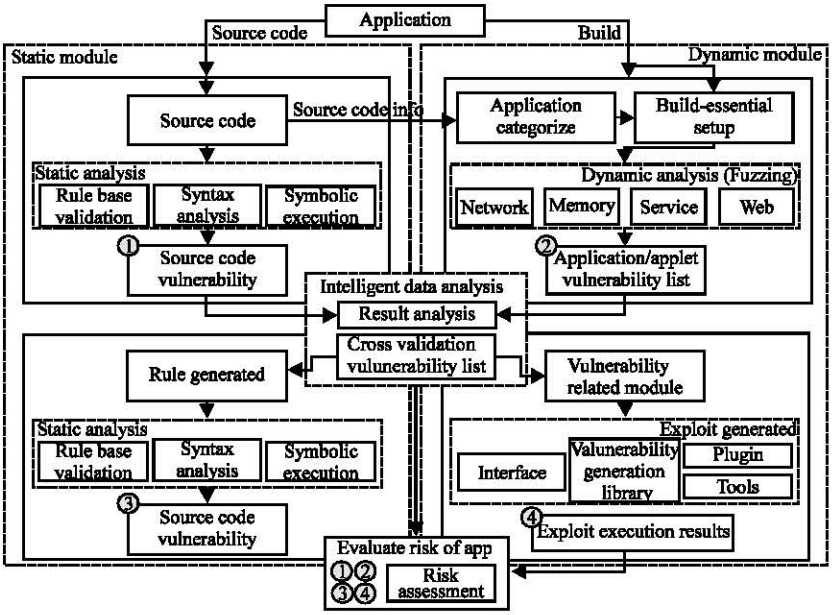


Fig. 7: Details of proposed intelligent vulnerability analysis technique

Table 2: Abbreviations used in formulas

Abbreviations	Full words
TI	Technical Impact
AL	Acquired privilege Layer
IC	Internal Control effectiveness
RL	Required privilege Layer
SC	Deployment Scope
AS	Authentication Strength
AP	Acquired Privilege
FC	Finding Confidence
RP	Required Privilege
AV	Access Vector
IN	Level of Interaction

$$\text{Base finding} = \{(10*TI+5*(AP+AL)+5*FC)*f(TI)*IC\}*4.0$$

The weighted score of 10 is used for technical affect due to the relevant vulnerability; weighted scores of 5 are used for reports of successful attacks and reliability about whether the relevant vulnerability actually exists or not is used to compute the scores. A unique characteristic is that the scores for the relevant vulnerability is not computed once the technical affect is decided to be 0. The score for AL is '1'; since, FC is also notified of the vulnerability, score is also '1'. Also, in order not to use IC, a score of '1' gets computed for the computing base finding scores.Attack surface metric score:

$$\text{ASM score} = \{20*(RP+RL+AV)+20*SC+15*IN+5*AS\}/100$$

In order to use the relevant vulnerability, the RP, RL, AV and SC are weighted each with 20%; the IN and AS are weighted with 15 and 5%, respectively. RL was not used in this study and we computed it as '1'. The Mid score uses the risk score that was computed from Eq. 1 and the accuracy of the relevant vulnerability. The mid score gets computed as the degree of risk of vulnerability score times the weights for accuracy. Detection accuracy is categorized into four types, based on the methods of the proposed technique. Table 3 shows the weights for accuracy.

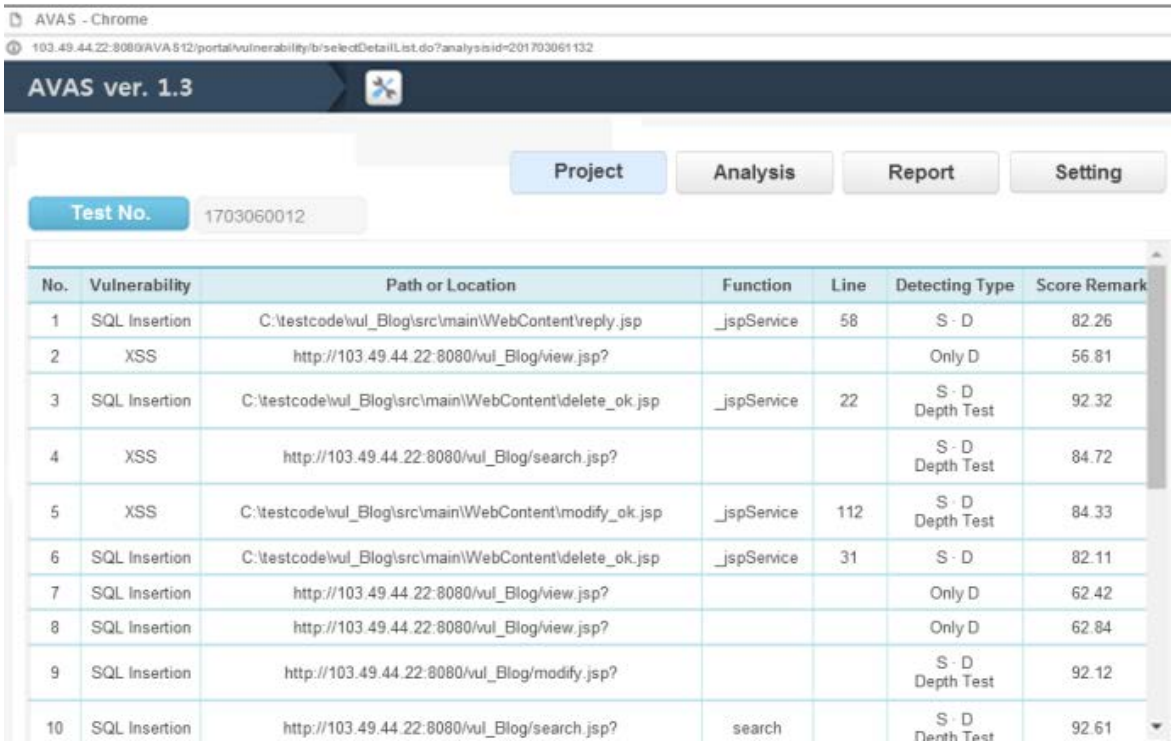
Table 3: Weights for vulnerability detection accuracy

Detection methods	Accuracy	Weights
③, ④	Very high	0.88
①∩②	High	0.81
②-(①∩②)	Mid	0.69
①-(①∩②)	Low	0.64

$$\text{Mid score} = \text{Degree of risk of vulnerability score} * \text{weights for accuracy}$$

Evaluation scores and mid scores of vulnerability self degree of risk: The final evaluation of degree of risk of the application is decided by considering the number of detected vulnerabilities and frequency of occurrences. In order to reflect these factors, the scope of vulnerability self-risk scores is categorized into three sections. The first section is for under 65 indicated with 1~9; The second section is over 66 and under 85 as indicated with 10~90 and the third section is over 85 indicated with 100~200 (Table 4).

Degree of risk presents the number of vulnerability detected at application using relative concept. Empirically risk weight was assigned-single vulnerability with high



No.	Vulnerability	Path or Location	Function	Line	Detecting Type	Score Remark
1	SQL Insertion	C:\testcode\vul_Blog\src\main\WebContent\reply.jsp	_jspService	58	S · D	82.26
2	XSS	http://103.49.44.22:8080/vul_Blog/view.jsp?			Only D	56.81
3	SQL Insertion	C:\testcode\vul_Blog\src\main\WebContent\delete_ok.jsp	_jspService	22	S · D Depth Test	92.32
4	XSS	http://103.49.44.22:8080/vul_Blog/search.jsp?			S · D Depth Test	84.72
5	XSS	C:\testcode\vul_Blog\src\main\WebContent\modify_ok.jsp	_jspService	112	S · D Depth Test	84.33
6	SQL Insertion	C:\testcode\vul_Blog\src\main\WebContent\delete_ok.jsp	_jspService	31	S · D	82.11
7	SQL Insertion	http://103.49.44.22:8080/vul_Blog/view.jsp?			Only D	62.42
8	SQL Insertion	http://103.49.44.22:8080/vul_Blog/view.jsp?			Only D	62.84
9	SQL Insertion	http://103.49.44.22:8080/vul_Blog/modify.jsp?			S · D Depth Test	92.12
10	SQL Insertion	http://103.49.44.22:8080/vul_Blog/search.jsp?	search		S · D Depth Test	92.61

Fig. 8: Report of detail results for application vulnerability analysis

Table 4: Ways to describe final vulnerability scores by sections of mid scores

Section of mid scores	Scores of degree of risk
Below 65	1st digit: 1~9
65~84	10th digit: 10~90
Above 85	100th digit: 100~200

scores of CWSS is more dangerous than much vulnerability with low scores. Namely, degree of risk is the adjusted value to clarify application's degree of risk intuitively bases on mean scores.

Implementation: In order to test proposed technique, vulnerability analysis system was implemented as prototype. System was made up with both static and dynamic modules; 2 static analysis tools and 7 dynamic analysis tools were used. Table 5 indicates analysis tools for test bed.

The system used a CPU quad core 2.66G Hz, RAM 4 GB, Windows 7, HD 500 G which was used to install all analysis tools and evaluation page was used in jsp for the reports on each analysis tool's linkages. Data set for assessment includes 15,277 Java source file known as Juliet code, 2 static analysis technique and 7 dynamic analysis technique. Using them, vulnerability detection test is performed by using and without using proposed technique. The application analysis data was composed of analyzed items, project items to show analyze results,

Table 5: Vulnerability analysis tools used for test bed

Items	Static module	Dynamic module
Analysis	PMD	Zap
Tools	FindBugs	Wfuzz
		Wapiti
		Spike
		Scratch
		Rotos
		Peach puzzer

report items to show project results and setting items to set up vulnerability items for detections. Figure 8 is a screen shot of the embodied system's vulnerability analysis.

The above shows the detailed results after analysis of the particular open source's vulnerability that is uploaded to the system. Detailed information of the vulnerability such as where errors are found or the vulnerable function's details can be shown by selecting the relevant vulnerability.

Performance evaluation: Table 6 presents the number of total detections, proper detections and false detections of the existing static analysis tools (PMD, Find Bugs) and proposed technique for top 10 items of CWE/SANS 25. Excluding only one item, the proposed technique presents many proper detections from all items, showing that its capabilities for detecting vulnerability is superior over existing static analysis tools.

Table 6: Comparison of analysis results of existing static analysis tool and static and dynamic mixed analysis results that apply proposed technique

	PMD			Find bug			Apply proposed method (PMD+peachpuzzer)		
Top 10 of SANS 25	Total	Proper detect	False detect	Total	Proper detect	False detect	Total	Proper detect	False detect
SQL input	482	299	183	463	281	182	563	384	179
OS Command input	3	3	0	3	2	1	11	3	8
Buffer overflow	0	0	0	0	0	0	0	0	0
Cross site script	61	45	16	58	43	15	457	410	47
Allowance of critical functions without proper authorization	0	0	0	0	0	0	0	0	0
Inadequate authorization	14	14	0	14	13	1	18	18	0
Hard coded critical information	112	107	5	115	101	14	50	48	2
Clear text transmission and save of critical information	2	2	0	2	2	0	54	48	6
File uploads	2	2	0	2	2	0	9	8	1
Inappropriate input value used in security function decisions	16	11	5	14	9	5	19	14	5

Because the result of mix usage of PMD and peach puzzer was the most superior over all other mix of techniques; results of mix usage of PMD and peach puzzer is used in this research.

CONCLUSION

In order to increase accuracy of detecting vulnerabilities that are embedded into the application, this study presented a vulnerability detection technique which increases accuracy of results, A new technique was proposed, i.e., the customized static analysis rule gets generated and an intelligent exploit code is generated to re-verify both static and dynamic modules to increase the accuracy of vulnerability detection. Moreover, new scores for application risk evaluation using self-risk scores of vulnerability are provided by CWSS and proposed detection accuracy is also suggested.

We found that the vulnerability detection capabilities of our proposed technique have better results than the capabilities of existing static and dynamic analysis tools. Moreover, the risk evaluation scores of applications using the proposed methods were suggested. This relevant application shows level of dangers at a look this proposed technique has strong points over existing analysis methods.

RECOMMENDATIONS

For the future research, environmental matrix calculation which was not used in this study will be added to find out its impact, its spread, its attack-ability and others this will result in full details of category and to improve reliability of weak degree of risk calculations.

REFERENCES

Agawal, M. and A. Singh, 2013. Metasploit Penetration Testing Cookbook. 2nd Edn., Packt Publishing, Birmingham, England, UK., ISBN:9781782166795, Pages: 320.

Allodi, L. and F. Massacci, 2014. Comparing vulnerability severity and exploits using case-control studies. ACM. Trans. Inf. Syst. Secur., 17: 1-20.

Anonymous, 2016. Common vulnerabilities and exposures: The standard for information security vulnerability names. US Department of Homeland Security,Washington,DC., USA. http://cve.mitre.org/.

Castro, M., M. Costa and T. Harris, 2006. Securing software by enforcing data-flow integrity. Proceedings of the 7th Symposium on Operating Systems Design and Implementation, November 06-08, 2006, USENIX Association, Seattle, Washington, ISBN:1-931971-47-1, pp: 147-160.

Chen, T., X.S. Zhang, C. Zhu, X.L. Ji and S.Z. Guo *et al*., 2013. Design and implementation of a dynamic symbolic execution tool for windows executables. J. Software Evol. Process, 25: 1249-1272.

Christey, S., 2011. CWE/SANS top 25 most dangerous software errors. MSc Thesis, MITRE Institute, Bedford, Massachusetts.

Coley, S.C., 2016. Common Weakness Scoring System (CWSS). Master Thesis, MITRE Institute, Bedford, Massachusetts.

Ernst, M.D., 2003. Static and dynamic analysis: Synergy and duality. Proceedings of the International Conference on Software Engineering (ICSE) and Workshop on Dynamic Analysis (WODA'03), May 03-10, 2003, University of Portland, Portland, Oregon, pp: 24-27.

FIRST., 2016. Common vulnerability scoring system version 3.0 calculator. FIRST.Org, Inc. Morrisville, NorthCarolina.https://www.first.org/cvss/calculator/3.0.

Fang, Z., Y. Zhang, Y. Kong and Q. Liu, 2014. Static detection of logic vulnerabilities in Java web applications. Secur. Commun. Netw., 7: 519-531.

ISO., 2016. Introduction to ISO 27005 (ISO27005). ISO, Geneva, Switzerland.

Khan, M.E. and F. Khan, 2012. A comparative study of white box, black box and grey box testing techniques. Intl. J. Adv. Comput. Sci. Appl., 3: 12-15.

Khurana, P., A. Sharma and P.K. Singh, 2016. A systematic analysis on mobile application software vulnerabilities: Issues and challenges. Indian J. Sci. Technol., Vol. 9, 10.17485/ijst/2016/v9i32/100190

Mouzarani, M., B. Sadeghiyan and M. Zolfaghari, 2016. A smart fuzzing method for detecting heap-based vulnerabilities in executable codes. Secur. Commun. Netw., 9: 5098-5115.

Ransbotham, S. and S. Mitra, 2013. The Impact of Immediate Disclosure on Attack Diffusion and Volume. In: Economics of Information Security and Privacy, Schneier, B. (Ed.). Springer, New York, USA., ISBN:978-1-4614-1980-8, pp: 1-12.

SDLP., 2016. More secure software. SDL PLC, Maidenhead, England, UK.