# ISA: A Source Code Static Vulnerability Detection System Based on Data Fusion

Deguang Kong, Quan Zheng, Chao Chen, Jianmei Shuai, Ming Zhu

School of Information Science and Technology

University of Science & Technology of China
+86-551-3647002

{kdg, jackchen}@mail.ustc.edu.cn

## ABSTRACT

Static analysis is a kind of effective method to detect the vulnerabilities in the software. Without running the programs, static analysis tools can be used to automatically discover unknown bugs. To cope with the problem of high false positives and false negatives in source code static analysis methods, this paper presents a source code static analysis technology for vulnerability detection based on data fusion. By parsing and making data fusion on the outcome of different static analysis methods, this technology lets different results validate each other, which greatly decreases the false positives and false negatives. Brief explanations are given to support this method. A prototype system of scalable source code analysis system (ISA for short) is designed and implemented which also can automatically search for the best result based on feedback of the user interaction. The whole system is scalable and platform-independent. It is proved by experiment that this method has a better performance with lower false positives and false negatives and higher efficiency compared with one single method.

## Categories and Subject Descriptors

G.3 [**Mathematics of Computing**]: Probability and statistics–*Statistical computing*;

D.2 [**Software engineering**]: Miscellaneous-Rapid prototyping, reusable software

## General Terms

Security, Design

## Keywords

Static analysis, Vulnerability, Data fusion

## 1. INTRODUCTION

As the increasement of information society's dependence on software systems, software security becomes a problem attracting more and more general concern. No matter in commercial software or open source software, software vulnerabilities can be found everywhere. Internet security threat research group X-force declared that, nearly 5,000 vulnerabilities are traced at the end of year 2005, and the figure rapidly increased to 7,000 at the end of year 2006. As an important means of detecting software vulnerabilities automatically, source code static analysis tools try to find the potential vulnerabilities and security problems in source code earlier by analyzing the source code before running the application.

There are two important guide lines for evaluating static analysis tools: (1) false negatives: the ratio of security problems ignored by the tool; (2) false positives: the ratio of inexistent security problems reported by the tool.

There have been heated researches on the static analysis tools. Existed source code static analysis tools can be divided to five kinds:
(1)tools based on annotation analysis, such as Splint/Lclint[5];
(2)tools based on lexical analysis, such as Its4[13], Rats[18], Flawfinder[9],etc.;
(3)tools based on grammar analysis, such as Boon[8];
(4)tools based on model checking detection, such as Mops[12], Verisoft[23], Codesurfer [16], etc.;
(5)tools based on type analysis, such as Cqual[3], etc.

However, nearly all these tools have a common weakness: producing many false negatives and false positives. On one hand, it is because of the limitation of static analysis tool itself, Rice Theorem has proved that static analysis is undecidable in the worst case, and some problems can not be solved just by static analysis; on the other hand, most static analysis tools' modeling is not precise enough, thus there are lots of differences between analysis model and practical program executive situation [2]. Moreover, many static analysis tools adopt the method of conservative analysis, in a manner of flow-insensitive or context-insensitive, which brings high false negatives and false positives.

How to reduce the false negatives and false positives of static analysis tools has become a hot problem of software vulnerability analysis. Although new source static analysis tools are continuously released, such as FaultMiner[17], a tool integrating data mining and static analysis; Oink[15], a C++ code vulnerability detection tool based on type analysis, etc., these tools still need to be improved to reduce false negatives and false positives.

Aiming at source code static analysis tool's weakness in high false negatives and false positives, this paper provides a source code static analysis method based on data fusion. This method integrates existed static analysis tools, parses and then makes data fusion on the result of different analysis, which lets different

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
INFOSCALE 2007, June 6-8, Suzhou, China
Copyright © 2007 ICST 978-1-59593-757-5
DOI 10.4108/infoscale.2007.910

output result verify each other to enhance the ratio of identifying real vulnerabilities and thus to attain better performance. This system can be used to detect vulnerabilities in the software also with higher efficiency.

Main contributions of this paper are:

(1) present a method based on data fusion to detect vulnerabilities to incorporate advantages of different tools ;

(2) feedback is introduced to adjust the parameter to get better result in the process of user interaction;

(3) design and implement a object-oriented system which is easily extended and platform independent;

(4) XML format output can be easily parsed and shared.

Structure of this paper is arranged as follows. Section 2 introduces the principle of source code static analysis technology based on data fusion. Section 3 introduces design and implementation of prototype system based on the principle of data fusion. Section 4 gives the result of experiment. Sections 5 introduces some related work. Finally it is conclusion and future work.

## 2. PRINCIPLE OF SOURCE CODE STATIC ANALYSIS TECHNOLOGY BASED ON DATA FUSION

### 2.1 The Reason for this Method

Source code static analysis tools based on different mechanisms employ different analysis methods, and even tools of same kind based on similar mechanism have differences in design and implementation detail. For instance, different rules in vulnerability database, enlightening strategy in algorithm implementation, definition of dangerous levels of vulnerability entry each would influence the ratio of identifying real software vulnerabilities.

Take most common tools with high efficiency (Its4, Rats, Flawfinder) for example, David Pozza's research indicated[6] that, when being tested with the same software package, Flawfinder found the most vulnerabilities, but did not totally cover those found by Rats and Its4; numbers of vulnerabilities found by Rats and Its4 are close, but many of them are not the same. Thus if each static analysis tool's advantages can be made full use of, and output data of these tools' analysis results can be fused in a proper way, to let the vulnerabilities sets verify and complement each other, and surely a better result comes out. On one hand, it would reduce the ratio of false negatives of source code static analysis and find more existed vulnerabilities. On the other hand, if the level of the fusion's result could be reasonably evaluated, the false negatives would also be reduced. Based on these thoughts, this paper presents a source code static analysis method based on data fusion.

### 2.2 Formal Description and Explanation

Proposition:

(1)A vulnerability identified by most tools is highly possible to be a real vulnerability;

(2) the possibility that a vulnerability would be falsely reported by all tools is low.

Explanation: It is assumed that there are n static analysis tools, and the true positives of the i-th static analysis tool is TP(i), then the false positives is FP(i) = 1 - TP(i).

(1)From

$$\prod_{i=1}^{n} TP(i) < \min\{TP(i)\},$$

it's easily attained that the possibility that a vulnerability be reported by n tools is reduced. It is reasonable, since most codes are non vulnerable codes and the possibility of existed vulnerability itself is low.

(2)The possibility that a vulnerability be falsely reported by n tools is

$$\prod_{i=1}^{n} FP(i) < \min\{FP(i)\},$$

If $FP(i) < \frac{1}{2}$,

Then $\prod_{i=1}^{n} \mathrm{FP(i)} < (\frac{1}{2})^{n}$ 。

Obviously, the possibility that a vulnerability be falsely reported by n tools will be reduced when n increases.

If different results of static analysis can be compared, namely, be properly transformed to a general intermediate form, then a random variable X is defined for score corresponding to each vulnerability to evaluate and forecast the possibility that some vulnerability be a real vulnerability. Since the vulnerability reported by most tools is more likely to be a real vulnerability than the vulnerability reported by less tools, rules for defining random variable X for score corresponding to each vulnerability are listed below:

(1)score of the vulnerability entry that is reported by multiple tools should be raised;

(2)score of the vulnerability entry that belongs to a high dangerous level should be raised;

(3)score of the vulnerability entry that belongs to a low dangerous level or appears unusually should be reduced;

(4)the contribution to score from different tools' analysis results should be different, namely, it could be adjusted by the proportion specified by user and adjusted to a proper proportion when combining with feedback adaptive adjustment.

### 2.3 Detailed Introduction

As for vulnerability entry( vul for short), s(i) means the i-th tool's variable for score (1 <= i <= n), r(i) means the i-th tool's contribution to analysis result, rs(i) means the i-th tool's real score, vul(i) = 1 means the vulnerability vul is reported by the i-th tool (1 <= i <=n), and it satisfy:

$$\sum_{i=1}^{n} r(i) = 1;$$

$$rs(i) = \begin{cases} s(i) & vul(i) = 1 \\ 0 & other \end{cases};$$

$$E(X) = \sum_{i=1}^{n} rs(i) r(i).$$

The variable for score s(i) maps the dangerous level of vulnerability entry(vul) defined by tool i to the score defined by tool i. Compared to the vulnerability entry belongs to lower dangerous level, vulnerability entry belongs to higher dangerous level is more likely to be real vulnerability, and it should have higher priority level for output.

For example, Its4 provides three classes of dangerous levels of vulnerability entries: High; Medium; Default; if it needs to be divided to five levels, the following mapping can be made: f(High) = 5; f(Medium) = 3; f(Default) = 1.

Value of r(i) can be specified during the process of user interaction, and feedback adjustment can be made to optimize it. From previous definitions, it is known that E(X), the mathematical expectations of variable for score X reflects n tools' analysis results for a vulnerability entry, namely, evaluation value to a vulnerability entry after data fusion.

## 2.4 An Instance

For the same vulnerability entry in software source code package wu-ftpd-2:5:0, the analyzed results from Its4, Rats, Flawfinder are indicated in figure 1 and figure 2 (Flawfinder failed to find this vulnerability entry).During the course of data fusion process , let r(1) = 0:3; r(2) = 0:3; r(3) = 0:4, and rs(1) =5; rs(2) = 5; rs(3) = 0, then E(X) will be 3.5.

```
wu-ftpd-2.5.0/src/extensions.c:183: High: fprintf
Check to be sure that the non-constant format string passed as
argument 2 to this function call does not come from an
untrusted source that could have added formatting characters
that the code is not prepared to handle.
```

**Fig.1. the tool of Rats's analysis result**

```
wu-ftpd-2.5.0/src/extensions.c:183:(Urgent) fprintf
Non-constant format strings can often be attacked.
Use a constant format string.
```

**Fig.2. the tool of Its4's analysis result**

## 3. INTEGRATED SOFTWARE SOURCE STATIC ANALYSIS BASED ON DATA FUSION—ISA

Based on previous principle, an integrated software source code analysis (ISA for short) is implemented .Features of this prototype system contains:

(1)this system integrates advantages of many analysis tools and let different result sets verify each other to reduces false positives and false negatives;

(2)this system adds user interaction to data fusion and data output procedure, import feedback mechanism when setting parameter value to effectively instruct the vulnerability mining;

(3)this system adopts the object-oriented idea and method for the design and implementation of the system, so it has good extensibility and also is platform independent;

(4)output of the system can adopt the form of XML, thus it is easy to describe, parse and share data.

Architecture design and implementation details of the system will be given in the following part.

## 3.1 Architecture Design of the System

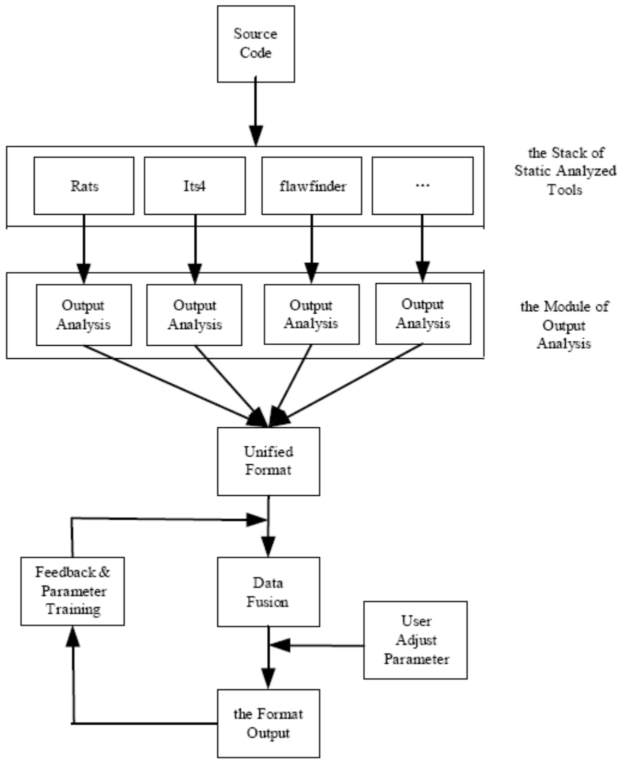


**Fig.3. the architecture of the system**

It consists of 6 major parts:

(1)After analysis on the program source code by static analysis tools in this static analysis tool stack, output results from different analysis tools can be attained. Static analysis tool stack is a framework integrating multiple analysis tools like TCP/IP protocol stack, each of which is independent and will not influence each other. Considering that the output results of Rats, Its4, Flawfinder are easy to compare and they are commonly and frequently used, this paper first implements the comparison and parsing of these three analysis tools. New static analysis tools can be easily added to the stack on demand, and the principle is similar.

(2)Main task of output result parsing module is to pick up valid vulnerability information from specific format of analysis result of the tools by using the lexical analysis method in order to fit the need of data fusion module. In the design of output result parsing module, inherited pattern is adopted to optimize the realization process. Namely, the unified parent class Process deals with operations of different tools output which may be the same, the child classes inherited from parent class (such as Rats process, Its4 process, etc.) parse specific tools output result. It is convenient to extend new static analysis tools.

(3)After previous analysis module, a unified format of data structure form is acquired, a six-tuple sequence: (name of file which contains the vulnerability, number of line the vulnerability locates at, dangerous level, vulnerability type, function causes the vulnerability, score for vulnerability threat).

(4)Fuse the data of analysis results from multiple tools, according to the principles introduced in Section 2. For example, if tool i does not find the vulnerability entry, then the score of entry in this tool rs(i) = 0. It is easy to calculate the score of a vulnerability entry.

(5) After data fusion process, the sorting module of c corresponding vulnerability sets is called and all vulnerability entries will be arranged from high score to low score. The threshold (confidence value) can be attainted from user interaction, and all vulnerabilities are printed out in format reserving vulnerabilities of higher scores. The default value is also set if there is no user interaction. A possible alternative format is XML format, which is easy to read, parse and transfer in the Internet.

(6)Compare the result of previous analysis with practical vulnerability analysis result, find the difference of vulnerability distribution, feedback the information representing differences to data fusion module through parameter training module, adjust the weight of r(i), and fuse the data of n result sets again. Repeat these procedures until parameter r(i) is trained to a comparatively proper proportion to attain optimal result.

## 3.2 Detailed Design of the System

The following part will give some detailed information about the realization of the system in 3 respects.

(1) The whole implementation of the system adopts the object-oriented method, uses design pattern like object factory, singleton, policy,template and so on, has good extensibility and robustness, and also is implemented by language C/C++ for easier cross-platform use. Figure 4 presents a part of class diagram of the system. Main control is the main control class of system processing flow, controls the processing flow of ISA. Broken line stands for dependence relation. Main control class creates two parent classes, class ResFactory and class Process. ResFactory is the parent class of object factory, and the three child classes Its4Factory, Rats Factory, Flaw Factory take charge of corresponding object creation separately. For example, Its Factory takes charge of creating instance of Its4process. Class Process is the parent class for output result parsing, offers interface for unified call and encapsulation of same operations and the three child classes Its4process, Rats process, Flaw process take charge of the parsing of corresponding tools output result separately.

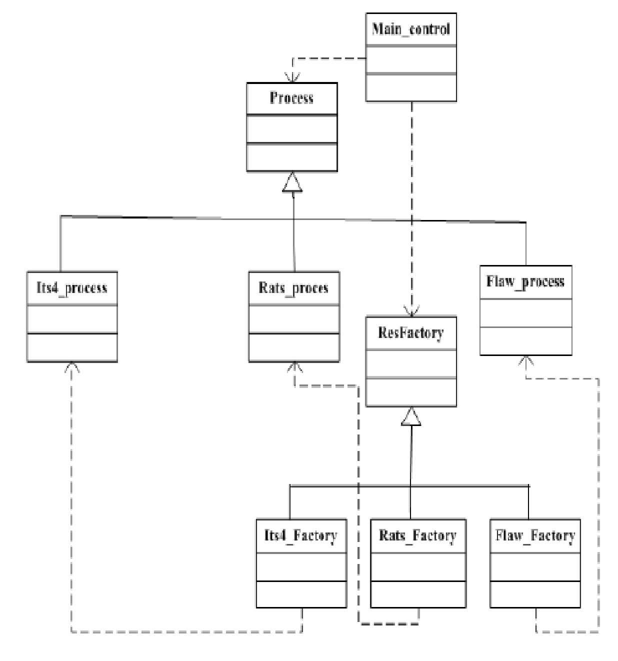
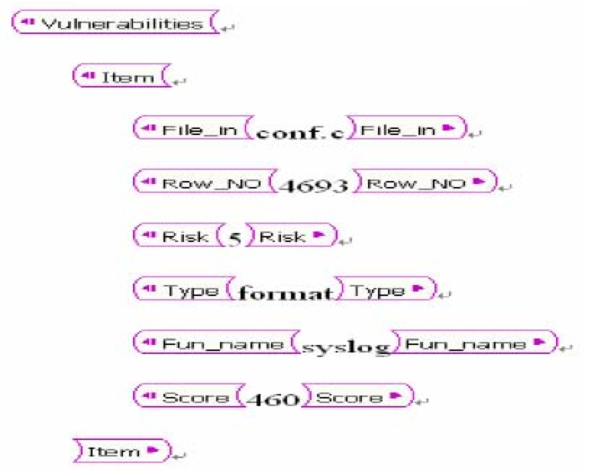


**Fig.4.class diagram of *ISA* (partly)**



```
Vulnerabilities
    Item
        File_in conf.c File_in
        Row_NO 4693 Row_NO
        Risk 5 Risk
        Type format Type
        Fun_name syslog Fun_name
        Score 460 Score
    Item
```

**Fig.5 a XML description of one vulnerability entry**

(2)Th [illegible] multiple description languages including XML format to satisfy different user demand. Being a general description language in the Internet age, data in XML format makes it convenient to describe, parse and share vulnerabilities. Fig 5 displays a sketch map of a vulnerability entry after a DTD (Document Type Define) is given.

(3)The whole system front-end depends on specific tools in static analysis tool stack, and the manner of parsing will change when the tools change. Other parts become the back-end. Compile ISA's code to the form of executable files, and the whole system can be executed both in Linux and Windows platform by the drive of the running scripts.

## 4. ANALYSIS AND COMPARITION OF THE EXPERIMENT RESULT

### 4.1 Experiment Result

Three software packages (wu-ftpd, Net-tools, Pure-ftpd) are employed for test. They are practical tools in the real world and thus have some representative, and they are related to network application. Some kinds of vulnerabilities including buffer overflow kinds have been found in these software packages, and also relative information can be found in database like CVE (Common Vulnerabilities and Exposure)[4],etc. Experiment environment for test is: Pentium 1.6G CPU, 256M RAM, Linux (Red Hat 9.0). Chosen source code static analysis tools are the prototype system ISA, Rats, Its4, Flawfinder.

**Table 1.Signature comparison of different source code package**

| Program | Ver. | LOC | LOE | LOE/LOC |
|---|---|---|---|---|
| wu-ftpd | 2.5.0 | 13582 | 64 | 0.47% |
| Net-tools | 1.46 | 4146 | 50 | 1.21% |
| Pure-ftpd | 1.0.17a | 25230 | 1 | 3.96E-5 |

**Table 2.The comparison of false positives of different tools**

| Program | ISA | Rats | Its4 | Flawfinder |
|---|---|---|---|---|
| wu-ftpd | 76.10% | 93.12% | 94.28% | 90% |
| Net-tools | 74.07% | 94.25% | 94.37% | 88.18% |
| Pure-ftpd | 100% | 100% | 100% | 100% |

**Table 3.The comparison of false negatives of different tools**

**Table 4.The comparison of work efficiency of different tools**

Detailed test data are given below:

Table 1 lists signature comparison of different source code package (Ver stands for the version number of software, LOC stands for the number of source code package code lines excluding blank lines and commented lines, LOE stands for the number of lines of real vulnerabilities).

Table 2 gives the comparison of false positives of different

| program | ISA | Rats | Its4 | Flawfinder |
|---|---|---|---|---|
| wu-ftpd | 28.1% | 39.1% | 39.1% | 29.6% |
| Net-tools | 6% | 26% | 20% | 14% |
| Pure-ftpd | 100% | 100% | 100% | 100% |

tools (the threshold of ISA is set to 0.21).

Table 3 gives the comparison of false negatives of different tools (the threshold of ISA is set to 0.21).

Table 4 gives the comparison of work efficiency of different tools.

TP represents the number of real vulnerability entries detected by tools, NUM represents the number of output result entries detected by tools, TP* represents the number of real vulnerability entries detected by tools after assigning the threshold of ISA, NUM* represents the number of output result entries after assigning the threshold of ISA, RTP represents the number of real vulnerability entries detected by first (NUM*ratio) entries after assigning tools ratio.

Let

$$\mathit{false\ \ negatives} = 1 - \frac{TP}{LOE};$$

$$\mathit{false\ \ positives} = 1 - \frac{TP}{NUM};$$

for tools such as Its4,etc. But for ISA,

$$\mathit{false\ \ positives} = 1 - \frac{TP^*}{NUM^*}.$$

As for the execution efficiencies of tools, for ISA it should be the number of vulnerabilities found by progressive scan the output result analysis after assigning the threshold, namely, for ISA,

$$\mathit{efficiency} = \frac{TP^*}{NUM^*}.$$

For other tools, since the output set of vulnerabilities is large and there is no corresponding sorting for vulnerability priority, the possibility of finding vulnerabilities by progressive scan is low. Let

$$\mathit{ratio} = 0.30;$$

$$\mathit{efficiency} = \frac{RTP}{NUM * \mathit{ratio}}.$$

Through training and adjustment procedure during user interaction, the threshold of ISA is set to 0.21.

### 4.2 Brief Analysis of the Experiment Result

After analyzing the previous data, it is found that the false negatives and false positives of ISA is reduced, except for Pure-ftpd. Since its source code package has only one vulnerability entry, none of the three origin tools found it, this vulnerability is still ignored through the corresponding algorithm of data fusion, so the false negatives and false positive are both 100%.

| program | ISA | Rats | Its4 | Flawfinder |
|---|---|---|---|---|
| wu-ftpd | 23.9% | 0 | 2.5% | 17.0% |
| Net-tools | 25.9% | 0 | 1.7% | 18.4% |
| Pure-ftpd | 0 | 0 | 0 | 0 |

The comparison of efficiency embodies the sorting of vulnerability priority, so it takes user less time to mine more real vulnerabilities. In conclusion, three advantages of the tool are:

(1) by fusing the data of result sets of multiple tools, increase the number of detected vulnerabilities and reduce false negatives.

(2) Through setting threshold by manual interaction and feedback adjustment, find more vulnerabilities in less time, namely, raise the mining efficiency.

(3) In the condition of having proper threshold, increase the correct rate and reduce false negatives.

## 5. RELATED WORK

As static analysis is important in eliminating security vulnerabilities in the programs, thus there has been lots of research relevant to static analysis for vulnerability detection. LCLint [5] is a kind of static program analysis tools which need programmer to annotate the source code extensively, but it is not accurate enough. Lexical tools such as Rats[18], Its4[13], Flawfinder[9] are commonly used to find misuse of dangerous function calls in source files. Their design and implementation is not so difficult as the other kinds of static analysis tools, but they nearly failed to understand the language semantics; thus these several tools are high in false positives and false negatives. Boon[8] is a kind of static analysis tool based on grammar analysis, and it builds a model of the program execution and tries to reduce the program to a simpler system to check buffer overflow vulnerability as a set of integer constraint problem. Its analysis is not accurate enough for it ignores the the execution order of programs statement and fails to handle the pointer alias problem, and generates huge false positives and negatives. Static analysis tools based on model checking detection, such as Mops [12], Verisoft[23] depend greatly on the specification the model needs to check, and also these tools are often bothered with too large state space. Take Mops for example ,the Push Down Automation(PDA) model is used to model the problem execution which ignores the dataflow analysis leading to inaccuracy of result. Also the specifications must be expressed to check one by one, which is also a source of errors. Tools based on type analysis, such as Cqual[3], etc. use type inference to categorize the data into trusted and untrusted data for detecting format string vulnerabilities, and also bring out false positives and negatives.

Recently static analysis tool such as FaultMiner[17] combines data mining technology to detect the vulnerabilities in the software package especially for the unknown invariant. As the data sets are very small, the mining result is not so credible, and inevitably it produces false positives and negatives. Oink[15] is also a static analysis tool based on type inferences to detect vulnerabilities especially for C++ language, which also has defects.

There have also been some work in dealing with the result sets of static analysis tools. Ted Kremenek's group [19] try to decrease the false positives and false negatives by analyzing the correlation between the output of one specific static analysis tool. They make cluster analysis to find the corrections and dependencies in the analysis outcome, and baysian network model is employed to train the data sets and get the better result. But different with this papers' idea, it is just based on one specific static analysis tool, and the training process could not guarantee the better result, and it is also inaccurate, besides this static analysis tool itself is likely in high false positives and false negatives. T. Kremenek's group [22] have presented a Z-ranking technique to rank error reports emitted by static program checking analysis tools. As a statistic method, the large data sets are required and also the experience of handling data is important , both of which may lead to inaccuracy.

Liusheng Huang and his team[20][21]present a common vulnerability markup language for easily realization of vulnerability detection. This paper gives a XML description of vulnerability entry for the use of vulnerability detection process to attain better result for scalability which is different from their method in [20][21].

## 6. CONCLUSION AND FUTURE WORK

This paper advances a source code static analysis method based on data fusion for vulnerability detection, designs and implements a prototype source code static integrated analysis system (ISA), by fusing the data of results of multiple static analysis tools, integrate advantages of the tools and acquire better performance.

The emphasis of future research will be how to integrate more vulnerability analysis tools to this prototype system, and developing data fusion algorithm with higher efficiency to continuously reduce false negatives and false positives produced in vulnerability analysis. The utilization of feedback control principle combined with the linear system model may be effectively improving the initial analysis output and leads to a much better result.

## 7. ACKNOWLEDGEMENT

First of all, the authors would thank a friend from Italy, whose name is Ashish Aggarwal. Due to his help for providing the important information about the data sets concerning about the vulnerabilities of the tested software above. The authors would like to thank Xin Gan, Zuochun Tang for their useful suggestions; also thank Jianfeng Pan for his kind help and sharing developing experience in "IceSword" software with us. This work have been supported by the National 863 high technology research and developing project (2006AA01Z449).

## 8. REFERENCES

[1] Alan Shalloway, James R.Trott. *Design patterns explaineda new perspective on object-oriented design*,China Machine Press, p101-143, 2006

[2] Brian Chess, Gray McGraw, *Static Analysis for Security*:IEEE Security & Privacy 04,2004, pp. 32-36

[3] Cqual: *http://www.cs.umd.edu/ jfoster/cqual/*

[4] CVE: Common Vulnerabilities and Exposure. *http://cve.mitre.org/*

[5] David Evans, John Guttag, Jim Horning. *LCLint: A Tool for Using Specifications to Check Code*. SIGSOFT Symposium on the Foundations of Software Engineering. December, 1994.

[6] David Pozza,Riccardo Sisto.*Comparing Lexical Analysis Tool for Buffer Overflow Detection in Network Software*. Communication System Software and Middleware First International Conference, Jan.2006

[7] D. Wagner. *Static Analysis and Computer SecurityNew technique for Software Assurance*. Ph.D Dissertation,Fall 2000.

[8] D.Wagner, J. Foster, E. Brewer and A. Aiken. *A first step towards automated detection of buffer overrun vulnerabilities*. In The 2000 Network and Distributed Systems Security Conference. February,2000.

[9] Flawfinder: *http://www.dwheeler.com/flawfinder*

[10] Giacomo Della Riccia, Hanz-Joachim Lenz, Rudolf Kruse Wien. *Data fusion and perception*, Springer,p67-85, c2001

[11] G. McGraw, *Software Security*, IEEE Security & Privacy, vol. 2, no.2, 2004, pp. 80-83.

[12 ] H. Chen, D. Wagner. *MOPS:An Infrastructure for Examining Security Properties of Software*, Proc. 9th ACM Conf. Computer and Communications Security (CCS2002),ACM Press, 2002,pp.235-244.

[13 ] John Viega, J. T. Bloch, Tadayoshi Kohno. *ITS4: A Static Vulnerability Scanner for C and C + + Code.* Annual Computer Security Applications Conference. December 2000.

[14] Kelly Carey and Stanko Blatnik. *XML: content and data,*p46-76, Prentice Hall, 2002.
[15 ] Oink(Cqual++): *http://oink.me.uk/*
[16] Paul Anderson ,Mark Zarins. *the CodeSurfer Software Understanding Platform*, Proceedings of the 13th International Workshop on Program Comprehension(IWPC 05)
[17] Rajeev Gopalakrishna, Eugene H.Spafford, Jan Vitek. *Fault-Miner: Discovering Unknown Software Defects using Static Analysis and Data Mining*, CERIAS TR 2006-07
[18] Rough Auditing Tool for Security: *http://www.scans.org./top20.html*
[19 ] Ted Kremenek, Ken Ashcraft, Junfeng Yang ,Dawson Engler.
*Correlation Exploitation in Error Ranking*. SIGSOFT04/FSE12, Nov, 2004
[20] Tian HT, Huang LS, Shan JL, et al. *Automated vulnerability management through web services,* LECTURE NOTES IN COMPUTER SCIENCE 3032: 1067-1070 2004
[21] Tian HT, Huang LS, Zhou Z, et al. *Common vulnerability markup language* LECTURE NOTES IN COMPUTER SCIENCE 2846: 228-240 2003
[22] T. Kremenek ,D. Engler. *Z-Ranking: Using statistical analysis*
*to counter the impact of static analysis approximations.* In SAS 2003.
[23 ] Verisoft: *http://cm.bell-labs.com/who/god/verisoft/*