UDC 004.415.5

**O.V. POMOROVA, D.O. IVANCHYSHYN**

***Khmelnitskyi National University, Khmelnitskyi, Ukraine***

# ASSESSMENT OF THE STATIC ANALYZERS USAGE FITNESS FOR SOFTWARE VULNERABILITIES IDENTIFICATION

*Source code analysis (SCA) tools are allowed to analyze source code and/or compiled version of code in order to help find security flaws. Most SCA tools are capable of finding multiple types of flaws, but the capabilities of tools are not necessarily uniform across the spectrum of flaws they detect. Even tools that target a specific type of flaw are capable of finding some variants of that flaw and not others. Nowadays there is a lack of information concerning the quality of functioning results of the static code analysis tools. In order to overcome this the SCA usage for software security defects identification was investigated. The common SCA tools functioning results were considered. The effectiveness of SCA means to identify defects in the source code associated with software security was analyzed.*

***Key words:*** *software security, static code analysis, static code analysis efficiency.*

## Introduction

One of the definitions of software security is its property to function without showing negative effects on a computer system [1]. The level of software security is a probability that software during its operation under given conditions will returned functionally suitable result. The reasons that lead to functionally useless results can be different: failures of the computer systems, error of programmers and operators, defects in software. Harmful influence on software is performed with the purpose of privacy violation, integrity and availability of information. Software safety assurance includes two main aspects: development of secure software and control of its secure execution. First aspect requires implementation into life cycle measures for software quality and security assurance. This includes usage of the special tools, studies of developers, architects and engineers for safety requirements providing. The second aspect is providing of secure software execution. For example sandboxing code (as the Java virtual machine does), protecting against malicious code, obfuscating code, monitoring programs as they run (especially their input), enforcing the software use policy with technology, and dealing with extensible systems.

However, today there are serious problems with security software. The Veracode company is analyzed the source code of software in order to identify vulnerabilities. The report indicates the presence of serious problems with safety in approximately 60% of tested software. The 88% of tested by company software does not meet standards OWASP and CWE/SANS [2].

Vulnerabilities in software lead to significant financial losses. For example, a buffer overflow error in Microsoft IIS and Windows RPC Service were used by viruses Code Red and MS Blaster. They cost to Microsoft $3.26 and $1 billion respectively. Buffer overflow in Microsoft SQL Server, which was used by SQL Slammer virus has brought losses of $1.2 billion for 5 days [3]. IBM's reports indicate that each year about eight thousand new vulnerabilities in software are appeared [4].

Therefore, implementation of measures for software security assurance throughout its life cycle is important problem for today.

One of the methods of source code verification and automated detection of defects associated with the software security is a static code analysis (SCA). SCA is the process of detecting errors and defects in software's source code that is performed without actually executing programs [5]. Static code analyzers are used to uncover hard to find implementation errors before run-time, since they may be even more difficult or impossible to find and assess during execution. These tools can discover many logical, safety and security errors in an application without the need to execute the application.

SCA methods include model checking, control and data flow analysis and text-based pattern matching [6]. The diversity between analysis algorithms leads to significant differences and differentiation of their functioning results. SCA tools do not demonstrate equally effectiveness in detection of various types of weaknesses, because do not cover all flaw types [7].

Nowadays researches indicate the necessity of quality assessment of the static code analysis results. The presence of false positive and the lack of standardization for interpretation of SCA results creates obstacles in tools effectiveness assessment [8][9]. Evaluation of

SCA usage effectiveness and development of the method for SCA tools selection will allow to improve the quality of static code analysis and to get a universal mean for selecting an analyzer for a specific software project.

## 1. C and C++ security problems

Today, considerable part of the system software, software for mobile and embedded systems, control systems, and other mission-critical software developed with using of C and C++ programming languages [10]. Languages such as C and C++ are designed primarily for efficiency and portability assurance. Therefore, they provide little support to avoid or to deal with runtime errors. For example, there is no checking in C that read or write access to an array is within bounds, that dereferencing of a pointer variable is possible (that the variable is not null) or that type casting is well-defined. Such checks must therefore be enforced by the programmer. For software with high security requirements advanced memory management and potentially dangerous set of library functions (such as gets) lead to increase costs and time that are spent on the phases of implementation and testing.

So for languages C and C++ much attention require weaknesses introduced in the source code. This requires the use of measures for weaknesses detection at the phase of implementation. Because of this source code security analysis is effective.

At present, there are a wide range of commercial and free tools for C and C++ static code analysis. Although C and C++ are different programming languages, they are treated as a single unit since C++ is a generally a superset of C. In addition, the most of software assurance tools support both C and C++.

## 2. SCA tools and test samples

Security department of U.S. National Institute of Standards and Technology provides a list of SCA tools that can be used for detecting and reporting about weaknesses that can lead to security vulnerabilities [11]. Table 1 shows the SCA tools for C and C++ that were selected for research and their brief descriptions.

Table 1

SCA Tools description

| Tool | Characteristics | |
|---|---|---|
| | Price | Main features |
| PVS-Studio | $4585 | There are 3 sets of rules included in PVS-Studio: diagnosis of 64-bit errors (Viva64) diagnosis of parallel errors (VivaMP) general-purpose diagnosis . |
| PC-Lint | $389 | Memory leaks, invalid STL usage, overlapping data in sprintf, division by zero, null pointer dereference, unused struct member, passing parameter by value, etc. |
| Goanna Studio | $999 | memory corruptions, resource leaks, buffer overruns, null pointer dereferences, C++ hazards, etc. |
| Cpp check | free | Customization facilities, pre-compiled headers, dimensional strong types, value tracking, semantic specifications (-sem), multi-thread support. |

Three sets of test samples for quality of the SCA assessment are selected. First set is taken from site of the U.S. department of homeland security. It includes 23 test samples designed specifically for SCA tools testing [12]. These example programs demonstrate flaws that may be detected by security scanners for C/C++ software. The examples are small, simple C/C++ programs, each of which is meant to evaluate some specific aspect of a security scanner's performance. The total number of vulnerabilities that should have been detected by SCA tools was 25.

Second set includes 14 test samples. It is taken from site of Security department of U.S. National Institute of Standards and Technology. Each case as in first group is small C/C++ program, which includes some defect, for example buffer overflow or memory leak. Set is selected for comparative analysis with the results for the first test set and assessment of the SCA results for different software.

The third test set contains 10 applications with only one type of defect. The widespread defect in the source code CWE (Common Weakness Enumeration) 476 CWE null pointer dereference was selected [13]. In C/C++, a null pointer does not point to any object. Null pointer dereference causes runtime errors and abnormal program termination.

Null pointer dereferences can result in the following security impacts: denial of service (denial-of-service attack is an attempt to make a machine or network resource unavailable to its intended users) and in some circumstances malicious code execution.

Null pointer dereferences generally impact the availability of an application. However, in some cases they can also impact confidentiality and integrity.

Each of considered SCA has rule for identification of such class of defects. So the third test set is aimed on detection the differences between the methods used by developers to identify the stated defects.

## 3. SCA Results

The following source code contains two defects of null pointer dereferences:

```
#1      #include <stdio.h>
#2      void function(int *ptr) {
#3            int value = *ptr; }
#4      int main(void)
#5      {
#6            int *p = NULL;
#7            int x;
#8            x = *p;
#9            function(p);
#10           return 0;
#11     }
```

Defect in the line 8 was detected by all SCA tools, but defect in the line 3 wasn't detected by any SCA tool. The difference in first and second situations is usage of dereferencing pointer operation in function body. Analyzers in the investigation do not track such situations. Thus, detection of a defect does not depend only on availability of the rule in analyzer's base. The structure of the source code also impact on results. Type of software and programming style determine differences of the source code, which influence on SCA applying effectiveness in general.

The next fragment of source code is more complex and contains a set of problems related to memory leaks, null pointer dereferencing and buffer overflows:

```
#1  class A {
#2   public:
#3    A() {
#4    char* p = new char[10];
#5    p = new char[10];
#6    char* a = (char *)0;
#7    *a = 0;
#8    char c[10];
#9    c[10] = 0;  }
#10  ~A() {
#11   delete p;
#12   char* a = new char[100];
#13   return;
#14   delete[] a; }
#15  private:
#16   char* p; };
#17  int main() {
#18   A a;
#19   return 0; }
```

Dereferencing null pointer a and array out of range c were found by all analyzers. Memory leaks associated with the pointers p and a were found by CppCheck and PC-lint only. Overall PVS-Studio and Goanna Studio found on 2 defects while CppCheck and PC-Lint found on 4. But the resulting reports contained 4, 4, 8 and 30 messages respectively. Therefore SCA allows to detect and to remove the real defects in the software. But the effectiveness of the tools usage is different.

## 4. SCA effectiveness

For quality of the SCA results assessment the set of metrics was chosen. Criteria that reflect the quality of SCA results are Precision and Recall. They are based on the number of true positive (TP), false positive (FP), and false negative (FN) in the analyzer's reports.

Precision means the ratio of weaknesses reported by a tool to the set of actual weaknesses in the analyzed code. It is defined as the number of weaknesses correctly reported (TP) divided by the total number of weaknesses actually reported (TP plus FP):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

The Recall metric represents the fraction of real flaws that were reported by a tool. Recall is defined as the number of real flaws that a tool reported (TP), divided by the total number of real flaws that existed in the code (TP plus FN):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

SCA tools reports contained messages with different importance levels: style, informational, elective notes, etc. These messages are not related to vulnerabilities that were presented in the test samples and can be suppressed in the results. Therefore, for Precision and Recall metrics calculation such messages did not taken into account. Only error and warnings messages for tools reports were taken.

In addition to the Precision and Recall metrics, an F_score was calculated by taking the harmonic mean of the Precision and Recall values. The F_score provides weighted guidance in identifying good static analysis tool by capturing how many of the weaknesses were found (TP) and how much noise (FP) was produced. F_Score is a measure of the SCA results quality:

$$\text{F\_Score} = 2 \times \frac{\text{SCA\_Precision} \times \text{SCA\_recall}}{\text{SCA\_Precision} + \text{SCA\_recall}}.$$

Usage of such metrics is a complex task for real software with a large amount of the source code. It is a difficult task to estimate accurately the number of actual defects in software. Small test samples do not cover all defects that may be present in real projects, but they allow to estimate the number of existing defects and to calculate the value of TP, FP, FN for the SCA results. Table 2 shows Precision, Recall and F_Score metric's values that were obtained for all test sets.

Table 2

SCA Results

| | CppCheck | PVS-Studio | Goanna | PC-Lint |
|---|---|---|---|---|
| | Test set 1 | | | |
| Recall | 0,64 | 0,24 | 0,12 | 0,48 |
| Precision | 0,89 | 1 | 1 | 0,41 |
| F_score | 0,74 | 0,39 | 0,21 | 0,44 |
| | Test set 2 | | | |
| Recall | 0,18 | 0,18 | 0,27 | 0,27 |
| Precision | 1 | 1 | 1 | 0,43 |
| F_score | 0,31 | 0,31 | 0,35 | 0,33 |
| | Test set 3 | | | |
| Recall | 0,6 | 0,5 | 0,5 | 0,6 |
| Precision | 1 | 1 | 1 | 0,75 |
| F_score | 0,75 | 0,67 | 0,67 | 0,67 |

Test samples contains an average no more than several tens lines of the source code. So, analysis Precision value is high enough. PVS-Studio and Goanna Studio analyzers Precision metric is equal to 1 for all test sets. Recall metric shows opposite results. It characterize small percentage of defects detection. Figure 1 shows metric F_Score change depending on the test set.

For the first test set effectiveness of SCA tools varies significantly. CppCheck is the most effective in this case. Static analyzer allows to detect the vast majority of defects present in test samples and to generate small percentage of false positives. For the second and third sets values are similar. All tools demonstrate analogous results. For CWE 476 vulnerability test set rate of the F_Score is the same for three analyzers. The result for CppCheck SCA is differed slightly.

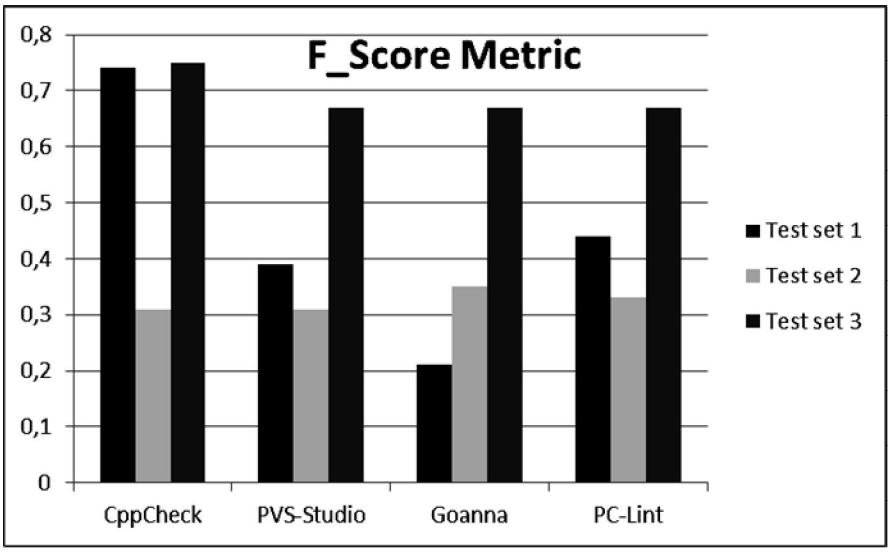


Fig. 1. F_Score metric for all test sets

Thus, the metric F Score reflects effectiveness of SCA. However, the study found that the effectiveness of static analysis is a complex measure that depends on the complexity and implementation details of the testing source code. The value of the F_Score metric is useful for comparison of the results for only single target software. The high results for particular project do not guarantee a similar result for any other software.

## Conclusion

Static code analysis is effective measure for software security assurance. The use of SCA technology in the software development requires a small amount of resources and allows to eliminate the real defects. However, selecting of the specific SCA tool is an actual problem. Different SCA tools do not demonstrate the same effectiveness in vulnerability identification. Effectiveness depends on the rule's bases, used methods, complexity and characteristics of the software source code.

The development of general approach for source code security analyzers choosing requires further study. This will help to improve the quality of SCA and to get a universal means for static analyzer selection for particular projects.

## References

*1. Kazarin, O.V. Software security for computer systems. Monograph. [Text] / O.V. Kazarin. – M.: MGUL, 2003. – 212 p.*

*2. Veracode Inc. State of Software Security Report: Volume 1 [Text] / Veracode Inc. – March 1, 2010. – 32 p.*

*3. Moore, D. Inside the Slammer Worm [Text] / D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, N. Weaver // IEEE Security and Privacy, – Vol. 1, No. 4. – July 2003. - P. 33 – 39.*

*4. IBM Security Systems. X-Force 2012 Mid-year Trend and Risk Report [Text] / IBM Security Systems. – 2012. – 6 p.*

*5. Lopes, R. Static Analysis tools, a practical approach for safety-critical software verification. [Text] / R. Lopes, D. Vicente, N. Silva // Critical Software SA Parque Industrial de Taveiro. Coimbra, Portugal. – 2009. - 12 p.*

*6. Nilsson, U. A Comparative Study of Industrial Static Analysis Tool [Text] / U. Nilsson, P. Emanuelsson // Electronic Notes in Theoretical Computer Science (ENTCS). Volume 217. – July, 2008. - P. 5 – 21.*

*7. Argen, M. Static Code Analysis For Embedded Systems [Text] / M. Argen. // Department of Computer Science and Engineering, Chalmers University of technology, university of gotenburg. Göteborg, Sweden. - August 2009. – 29 p.*

*8. National Security Agency Center for Assured Software. On Analyzing Static Analysis Tools. [Text] - July, 2011. – 13 p.*

*9. Pomorova, O.V. Features of the Static Code Analyzers Usage for Different Types of Software [Text] / O.V. Pomorova, D.O. Ivanchyshyn // Proc. of the International Conference on Computer Science and Information Technologies (CSIT 2012). - 2012. – P. 134 – 136.*

*10. Lockheed Martin Corporation. Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program. [Text] – 2005. – 142 p.*

*11. National Institute of Standards and Technologies (NIST) SAMATE project [Electronic resource]. – Source Code Security Analyzers. – Available to: http://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html. – 18.01.2013 y.*

*12. Build Security In [Electronic resource]. – Source Code Analysis Tools - Example Programs. – Available to: https://buildsecurityin.us-cert.gov/bsi/articles/tools/code/498-BSI.html. – 18.01.2013 y.*

*13. A community developed dictionary of software weakness type [Electronic resource]. – Common Weakness Enumeration. – Available to: http://cwe.mitre.org. – 18.01.2013 y.*

*Надійшла до редакції 21.02.2013, розглянута на редколегії 13.03.2013*

**Рецензент:** д-р техн. наук, проф. каф. комп'ютерних систем і мереж А.В. Горбенко, Національний аерокосмічний університет ім. М.Є. Жуковського «ХАІ», Харків, Україна.

## ОЦІНКА ПРИДАТНОСТІ ЗАСТОСУВАННЯ СТАТИЧНИХ АНАЛІЗАТОРІВ ДЛЯ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ В ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

***О.В. Поморова, Д.О. Іванчишин***

Інструментальні засоби статичного аналізу вихідного коду (СА) дозволяють аналізувати вихідний код або відкомпільований код з метою виявлення недоліків безпеки. Розробники засобів СА декларують можливість виявлення значного переліку дефектів, проте ефективність ідентифікації для різних типів помилок є не однаковою. В статті досліджено застосування статичного аналізу вихідного коду для виявлення дефектів пов'язаних з безпекою програмного забезпечення. Розглянуто результати функціонування поширених на сьогодні інструментальних засобів СА вихідного коду. Проведено аналіз ефективності застосування інструментальних засобів СА для виявлення дефектів у вихідному коді пов'язаних з безпекою ПЗ.

**Ключові слова**: безпека ПЗ, статичний аналіз, ефективність статичного аналізу.

## ОЦЕНКА ПРИГОДНОСТИ ПРИМИНЕНИЯ СТАТИЧЕСКИХ АНАЛИЗАТОРОВ ДЛЯ ВЫЯВЛЕНИЯ УЯЗВИМОСТЕЙ В ПРОГРАМНОМ ОБЕСПЕЧЕНИИ

***О.В. Поморова, Д.А. Иванчишин***

Инструментальные средства статического анализа исходного кода (СА) позволяют анализировать исходный код или откомпилированный код с целью выявления недостатков безопасности. Разработчики средств СА декларирует возможность выявления значительного перечня дефектов, однако эффективность идентификации для различных типов ошибок является не одинаковой. В статье исследовано применение статического анализа исходного кода для выявления дефектов связанных с безопасности программного обеспечения. Рассмотрены результаты функционирования распространенных на сегодня инструментальных средств СА исходного кода. Проведен анализ эффективности применения инструментальных средств СА для выявления дефектов в исходном коде связанных с безопасностью ПО.

**Ключевые слова**: безопасность ПО, статический анализ, эффективность статического анализа.

**Поморова Оксана Вікторівна** – доктор технічних наук, професор, завідувач кафедри системного програмування Хмельницького національного університету, e-mail: o.pomorova@gmail.com.

**Іванчишин Дмитро Олександрович –** аспірант кафедри системного програмування Хмельницького національного університету, e-mail: dmytro_ivanchyshyn@ukr.net.