2nd Hellenic Conf. on AI, SETN-2002, 11-12 April 2002, Thessaloniki, Greece,
Proceedings, Companion Volume, pp. 531-536

Applying the Blackboard Model in the Security Field

Simeon (simos) Xenitellis *

Information Security Group,
Royal Holloway University of London,
TW20 0EX United Kingdom
S.Xenitellis@rhul.ac.uk

Abstract. The blackboard model is a flexible problem-solving approach that can be used in domains where the data and the knowledge rules are strictly separated [7]. It is used in a variety of fields such as speech recognition or optical character recognition. In this document we discuss the use of the blackboard model in the security field and the automatic detection of programming mistakes.

Keywords: artificial intelligence, blackboard model, software security, dataflow analysis

1 Introduction

The blackboard model is a problem-solving approach that is well-suited in domains where the data and the knowledge rules are strictly separated [7]. It was first described by Newell in 1962 and became famous with the Hearsay II speech recognition system [3].

The blackboard model has been applied in such diverse fields as the automatic transcription of polyphonic music [8], parallel map recognition [16] and the cryptanalysis of monoalphabetic substitution ciphers [1].

In the cryptanalysis of monoalphabetic substitution ciphers, specific knowledge bases (KB) of the target language are constructed that can handle language specific information. Such KBs handle frequences of individual letters, bigrams, words of specific sizes and understand common prefixes, suffixes and syntactical structure. A controlling mechanism asks in every round the KSs to provide an estimation of the appropriateness of their best suggestion, based on the current state of the solution. The most promising suggestion gets permission to update the state of the solution. If a dead end is reached, the controlling mechanism backtracks the last suggestions, trying alternative ones for each round.

Another field of computer security that can benefit from the blackboard model is that of the vulnerability analysis of source code of software systems.

* The author's studies are funded by the State's Scholarship Foundation (SSF) of Greece.

Currently, to detect security vulnerabilities in the source code, manual inspection by a security specialist team appears to be the most successful solution [14].

In the case of automatic code inspection, an example of automated static analysis is the Meta-level Compilation (MC) project [2]. The MC project uses the GCC [6] suite of compilers to preprocess the source code into an intermediate form and subsequently applies simple checker rules to detect common software reliability vulnerabilities.

An extension to the current systematic approach of using flow analysis is the addition of the blackboard model. The coupling of flow analysis with the blackboard model is the contribution of this paper.

In section 2 we give a background of the vulnerabilities we try to tackle and in section 3 we describe the related work. In the following section we describe the process of security auditing that we will follow for the automated method we present. In section 5 we present the blackboard application of automated security auditing and we end with conclusions.

2 Background

There is a range of software reliability issues that can occur in applications that are written in programming languages like the C programming language. One cause of software reliability issues is the lack of a bounds checking mechanism when accessing strings in memory. Although a change of programming language can eliminate those vulnerabilities, this is not practical. A large number of software applications are already written in the C programming language and there are other features of the language that make it still worthy to use.

The C programming language does not provide built-in mechanisms to protect against buffer overruns for reasons of simplicity, speed and efficiency. Overrunning a buffer results in overwriting another area of memory that typically holds important internal structures of the running application. Knowledge of the internal structures can result in the exploitation of the buffer overrun and the execution of malicious code.

Recently, a software vulnerability affecting a special structure of the C language, the format strings, has been discovered [11, 12]. In this vulnerability, manipulating the contents of the format string can also overwrite areas of the internal memory of the running application, again with the potential to cause the execution of malicious code.

In both of the above cases, the overwriting of internal memory of the running application makes the application to malfunction and subsequently crash or hang [9, 5]. If this malfunction is caused due to crafted input by the attacker, it can lead to loss of availability (denial of service) or even worse to the execution of malicious code.

3 Related Work

There are several projects that aim in the detection of security vulnerabilities using the source code. The majority of those use static analysis, the analysis of problematic code structures in a flow-insensitive manner. Typically, such structures are specific function invocations or specific buffer operations.

The MC project [2] uses automated static analysis as part of the compilation procedure and ITS4 [15] statically scans source code for security vulnerabilities such as buffer overruns and race conditions. RATS [13] and flawfinder [17] are open-source projects that check for security vulnerabilities by statically scanning the source code for dangerous functions and trying to infer whether a buffer overrun may occur.

Splint [4] uses static analysis to find buffer overrun and format string vulnerabilities. Additionally, it performs a limited flow analysis.

4 Security auditing

Solving all types of software reliability problems using automated testing is extremely difficult [10]. However, focusing on a subset is more manageable. The subset of software reliability problems we are dealing with are those that undermine the integrity of an application. In this setting, the attacker tries to access without authorisation the assets of the system. The entrance for the attacker to the system is any type of input that can be manipulated and subsequently be exploited. The security vulnerabilities could be logical errors (when an empty username and password grants access) or programming mistakes (when a buffer os overflowed). In this paper we deal with the latter, assuming the former does not take place.

Thus, in order to eliminate the programming mistakes that lead to security vulnerabilities, the software security auditor should

- locate all inputs to the system that can be manipulated by an attacker,
- establish what input is legal and reject all others [18],
- follow the flow of the input data in the system,
- identify when a buffer overrun or format string mistake occurs,
- and finally fix the error.

5 The Blackboard Application

The blackboard application is composed of a set of knowledge sources that capture diverse knowledge domains, the blackboard that holds the current state of analysis and a control mechanism that manages the interactions between the previous two.

5.1 Knowledge Representation

The blackboard contains information about all possible entry points to the system that the attacker has access to. Such entry points could be

- command-line parameters,
- environment variables,
- input files,
- inherited file descriptors,
- or data from a network connection

It is assumed that any sort of input value can enter through the entry points. For example, a simple text editor may open binary files.

In a specific setting, different entry points may be considered to be available. A word processor on a standalone workstation has no entry points while when it is used to open documents downloaded from the Internet, the entry point is the input file that was received.

Additionally, the blackboard contains tree structures where the roots represent the entry points of the input data and each node represents a change to input data being passed to other variables. A node can have several subnodes, depending on changes to the flow of execution, such as conditional cases and function invocations.

5.2 Knowledge Sources

The knowledge sources are divided in the following groups, capturing different domains of the process of identifying security vulnerabilities.

- **Input Type Knowledge** These knowledge sources parse the source code in an attempt to identify if their input type is present. If it is present, they make an entry to the blackboard.
- **Flow Analysis Knowledge** These knowledge sources follow the input data while they are processed, during their whole lifetime until program termination.
- **Data Manipulation Knowledge** These knowledge sources can understand different types of data manipulation that can occur to the input data, such as the effect of library functions.
- **Impact Assesment Knowledge** These knowledge sources are called to determine whether the input data that trigger the buffer overflow or any other input related vulnerability, can be exploited to produce either a denial of service or most importantly the execution of malicious code.

5.3 Control Mechanism

The control mechanism orchestrates the execution of the KSs in order to fill the blackboard and represent all the input dataflow paths. The control mechanism is independent of the individual KSs. Depending on the information needed on the

blackboard, the controlling mechanism triggers the KSs to provide estimations of the appropriateness of their contribution. The most promising contribution gets the chance to update the blackboard.

The controlling mechanism is KS-agnostic. This is an important feature that ensures the efficiency of adding new knowledge material, as it gets discovered.

The goal of the controlling mechanism is to explore all dataflows of the input data and check whether a security vulnerability arises. Once all dataflows have been explored, the program is deemed secure for the specific set of knowledge sources.

6 Conclusions

Currently there is enormous need for methods to ensure that software is delivered without security vulnerabilities. Such an avenue appears to be the combination of artificial intelligence with dataflow analysis of the source code. The promising component of artificial intelligence that we propose is the blackboard model. In this paper we described the knowledge domains, the blackboard structure and the control mechanism of the blackboard application.

Acknowledgments

The author would like to thank Chris Mitchell for reviewing this paper.

References

[1] Grady Booch. *Object-Oriented Analysis and Design with Applications*. Object-Oriented Software Engineering. Addison-Wesley Publishing Company, Rational, Santa Clara, California, 2nd edition, 1994.

[2] Dawson Engler, et al. The Meta-level Compilation (MC) project. http://hands.stanford.edu/, 2001.

[3] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. In B. L. Webber and N. J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 349–389. Kaufmann, Los Altos, CA, 1981.

[4] David Evans and David Larochelle. Improving security using extensible lightweight static analysis. *IEEE Software*, January/February 2002.

[5] Justin E. Forrester and Barton P. Miller. An Empirical Study of the Robustness of Windows NT Applications Using Random Testing. 4th USENIX Windows Systems Symposium, August 2000.

[6] GCC. GNU Compiler Collection. http://gcc.gnu.org/, 2002.

[7] Magnus Kempe. A Framework for The Blackboard Model. ftp://lglftp.epfl.ch/pub/Papers/kempe-Blackboard-Overview.ps, 1995.

[8] Keith Martin. Automatic transcription of simple polyphonic music: Robust front end processing. Technical Report 399, M.I.T. Media Lab, 1996.

[9] Barton P. Miller, Cjin Pheow Lee, Vivekananda Maganty, Ravi Murthy, Ajitkumar Natarajan, and Jeff Steidl. Fuzz revisited: A re-examination of the reliability of unix utilities and services. Technical report, Computer Sciences Department, University of Wisconsin, 1995.

[10] Glenford J. Myers. *The art of software testing.* Business Data Processing. John Wiley and Sons, Inc., 1979.

[11] Tim Newsham. Format string attacks. http://www.guardent.com/docs/FormatString.PDF, September 2000. Guardent, Inc.

[12] scut. Exploiting format string vulnerabilities. http://teso.scene.at/articles/formatstring/, September 2001. TESO.

[13] Secure Software Solutions. Rats - rough auditing tool for security. http://www.securesw.com/rats/, 2001.

[14] Theo de Raadt et al. The OpenBSD free operating system. http://www.openbsd.org, 2002.

[15] John Viega, J. T. Bloch, Tadayoshi Kohno, and Gary McGraw. ITS4: A static vulnerability scanner for c and C++ code. In *Proceedings of the Annual Computer Security Applications Conference*, December 2001.

[16] Li Wang. A Framework for Parallel Map Recognition Based on Blackboard Model. citeseer.nj.nec.com/11673.html, 1996.

[17] David A. Wheeler. flawfinder. http://www.dwheeler.com/flawfinder/, 2001.

[18] David A. Wheeler. Secure Programming for Linux and Unix HOWTO. http://www.dwheeler.com/secure-programs/, 2002.