Development of a Source Code Analysis Environment Focusing on Security

L.O. Duarte[1] and A. Montes[2]

[1]Laboratory for Computing and Applied Mathematics - LAC
Brazilian National Institute for Space Research - INPE
C. Postal 515 – 12245-970 – São José dos Campos - SP
BRAZIL

[2]Information Systems Security Division - DSSI
Renato Archer Research Center – CENPRA
13069-901 – Campinas - SP
BRAZIL

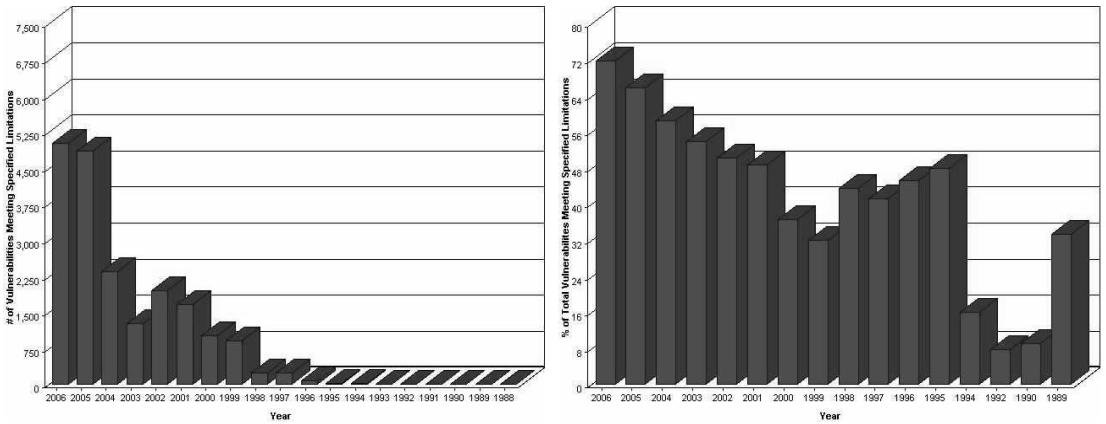
E-mail: duarte@lac.inpe.br, antonio.montes@cenpra.gov.br

Keywords: secure programming, syntactic analysis, secure coding, weak functions, security.

In the past years, the expressive increase in the development of computer technologies has been followed by an increasing number of systems being probed, infected, compromised and used to launch attacks on other systems. Successful attacks against computer systems are due to the exploitation of some kind of vulnerability. Poor programming techniques are the main cause of vulnerabilities. The more common vulnerabilities are:

- **Buffer Overflow:** It is the most common vulnerability present in software. It often occurs when more data is inserted in a buffer than it can handle. (Pincus, 2004), (Aleph One, 1996)
- **Race Condition:** This vulnerability occurs when two or more processes try to access the same resource simultaneously. One process can intentionally change a resource in such a way that a second process behaves unexpectedly. (Cowan *et* al, 2001a)
- **Injections:** It is a vulnerability associated with poor software input validation causing situations such as unexpected instruction or SQL command execution.
- **Format String Vulnerability:** It is a vulnerability in the format string of functions like printf() and syslog() that can allow information leak, unauthorized accesses to memory locations or access to the system by an attacker. (Cowan *et* al, 2001b)

Statistics of NVD/NIST (National Vulnerability Database / National Institute of Standards and Technologies) show that the number of discovered vulnerabilities in software is increasing. The number of indexed vulnerabilities from January to September of 2006 was bigger than all 2005 year vulnerabilities (Figure 1). Input validation errors are the main cause of those vulnerabilities. The buffer overflow vulnerability is related to validation input error and are the focus of this work.



The left graphic represents the number of indexed vulnerabilities. The percentage of indexed vulnerabilities due to input validation errors is shown in the right graphic.

The efforts to mitigate the vulnerabilities found in software are divided into three approaches:

- Compiler dependent approach; (Etoh, 2001) (Vendicator, 2000)
- System dependent approach;
- Software dependent approach.

This work is based on a software dependent approach. In this approach the software must be secure by itself, without the need of a specific operating system or compiler. To assure that a software is secure, one must observe the entire software development process to avoid programming errors.

The software audit processes look for problems inserted in the coding stage. This audit can be done in a static or dynamic way. The software dynamic analysis aims to discover problems during the software execution, often without analyzing the source-code. This technique demands that all the execution paths of the software are covered and exhaustively tested with all types off possible inputs. Although effective, analyzing all possible paths is not always feasible. Therefore, other techniques like fuzzying(Miller *et* al., 1990) are used.

The static analysis aims to determine the software properties by inspecting its source-code. The software is not executed in this method. The supposed advantage of this method is that it detects errors that could be too difficult to find using others methods. The tools that help the static analysis process vary among functions spotters, that are not more sophisticated than the grep tool, and precompilers. New approaches consist of vulnerability detection using constraint optimization.

This work presents a proposal of a source code analysis environment focused on security. Its main goal is to help developers to find vulnerabilities in their own software. The proposed environment analyzes a software source code to find buffer overflow vulnerabilities through a preventive and software-dependent approach, in a syntactic level. To achieve it, the environment tries to supply the limitations found in other tools. Some of these limitations were spotted by (Wilander & Kamkar, 2003).

REFERENCES

Cowan, C., Barringer, M., Beattie, S., Kroah-Hartman, G.,Frantzen, M. and Lokier, J.(2001) *FormatGuard: Atomatic Protection From printf Format String Vulnerabilities.* Paper presented to Conference: Usenix Security Symposium, 10th., Washington, United States of America.

Cowan, C.; Beattie, S., Wright, C. and Kroah-Hartman, G.(2001) *RaceGuard: Kernel Protection From Temporary File Race Vulnerabilities.* Paper presented to Conference: USENIX Security Symposium, 10th., Washington DC, United States of America.

Etoh, H.(2001) *GCC extension for protecting applications from stack-smashing attacks*. Online at: <http://www.trl.ibm.com/projects/security/ssp>

One, A.(1996) *Smashing The Stack For Fun And Profit.* Phrack Magazine, v. 49, n. 14. Online at: <http://www.phrack.org/phrack/49/P49-14>

Pincus, J.; B., B.(2004) *Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns*. IEEE Security & Privacy, v. 2, n. 4.

Vendicator(2000) *Stack Shield: A stack smashing technique protection tool for Linux*, 2000. Online at: <http://www.angelfire.com/sk/stackshield/>.

Wilander, J., Kamkar, M.(2003) *A Comparison of Publicly Avilable Tools for Static Intrusion Prevent.* Paper presented to Symposium: Network and Distributed System Security Symposium, 10th., San Diego, California, United States of America.

Miller, P. B., Fredriksen, L., So, B., (1990) *An empirical study of the reliability of UNIX utilities.* Comuninications of the Association for Computing Machinery.