Diseño y Programación Seguras
Caso 2: AusCERT wrapper

Ignacio Nieto Vidaurrázaga
Máster en Ciberseguridad
Universidad de León
León, Spain
inietv00@estudiantes.unileon.es

***Abstract*—In this case it is shown the implementation of a wrapper that allows the control and processing the input of strings, more specifically the length of the string. The input produced by the user either unconsciously or in a malicious way can lead to a buffer overflow and very pernicious consequences. Another advantage of this wrapper is to isolate the program from user's input until it is treated and correctly formated, in this way it is prevented also from exploitation of the code.**

***Index Terms*—Design, simplicity, AusCERT, wrapper**

I. Summary

Information is the powerful and most valuable resource for every company. Aligned with that, information security has become the biggest concern in companies and organizations since they are exposed to huge amounts of losses including onerous fines imposed by governmental agencies. A good and smart design is key to protect our programs and the information, and so it is shown in the case of study of the AusCERT overflow wrapper.

The problem of the buffer overflow has been referred in so many prestigious books, like Seacord [1] or Graff and van Wyk [2] ones, but it had a milestone with an article called "Smashing the stack for fun and profit" that was published in 1996 by Aleph One [3]. In it were defined the stack management and some problems arising as a result of this, among which is the buffer overflow. This overflow is the result of stuffing more data into a buffer than it can handle. This often can culminate in an advantage to execute arbitrary code.

One year after this article was developed the AusCERT wrapper and this can give us an idea of how important was the issue in the mid years of that decade. But this problem remained until our time and will continue to be one of the main themes of secure coding. It may vary the programming language, the libraries included or the standards approved by the International Organization for Standardization (C89, C99, C11) [4], but the developers will always have to take the control over their code and limit its exploitation.

Since the wrapper and so many code is developed and fixed in C/C++ nowadays, It is very useful in this regard the SEI CERT C Coding Standard [5]. This standard is published by Carnegie Mellon University and provides rules for secure coding in the C programming language and the goal of these rules and recommendations is to develop safe, reliable, and secure systems, for example by eliminating undefined behaviors that can lead to undefined pro-gram behaviors and exploitable vulnerabilities. Considering the problem of string's length would be applicable the note STR31-C.

Notwithstanding the foregoing, if we are lazy as developers or suffocated by deadlines we could have a temptation, not to follow the pieces of advice given by standards. In this case we must trust on the acknowledgment of great institutions such as the mentioned CMU, and of course the AusCERT organization must be included into them. The Australian cybersecurity team has demonstrated over years its efforts and good practices, for example by organising every year an international conference with leading professionals in this sector or publishing newsletters and articles of very current interest.

II. Conclusions

Above all I would like to highlight two important aspects related to this wrapper.

A. Simple design

It is said that great ideas are usually simple ideas and so it is the AusCERT wrapper. We can find in few lines of code a solution that really works and its easy to understand. Often the best way to comment code is the code itself when its clean and simple.

B. Efficacy and durability

Not only good solutions in a concrete moment are needed when speaking of coding but also make this solution sustainable over time. And that's what you get when using a wrapper that is isolated from the main program and can be used in future releases and in other applications or programs.

III. References

References

[1] Robert C. Seacord, "Secure Coding in C and C++ ". Addison-Wesley. 2nd Edition 2013, pp. 35–86.
[2] Mark G. Graff, Kenneth R. van Wyk, "Secure Coding: Principles Practices". O'Reilly. 2003. Chapter 3.
[3] E. Levy (Aleph One), "Smashing the Stack for Fun and Profit". Phrack Magazine, 49. 1996.

[4] International Organization for Standardization. (1990). Programming languages — C — Technical Corrigendum 2 (ISO/IEC 9899:1990/COR 2:1996). Retrieved from https://www.iso.org/standard/27110.html.
[5] Carnegie Mellon University, “SEI CERT C Coding Standard. Rules for Developing Safe, Reliable, and Secure Systems”. Software Engineering Institute, Carnegie Mellon University, 2016 Edition, pp. 232–261.