Automatically Detecting and Diagnosing Use-After-Free Vulnerabilities present in Binary Files.

John Galea
john.galea.10@um.edu.mt

Use-after-free (UAF) vulnerabilities are caused by the use of dangling pointers, which are pointers left referring to deallocated memory [1]. Since their exploitation may allow attackers to hijack control-flow or leak sensitive information, their removal is of vital importance in relation to software security. However, manually identifying and diagnosing UAF vulnerabilities are difficult tasks to accomplish, especially for large and complex programs, such as script-hosting applications. Typically, vulnerability analysts require examining binary files by utilizing low-level tools (e.g. interactive debuggers), and are faced with inspecting large sections of machine code and dynamic program states. A need exists for tools that facilitate vulnerability analysis in order to detect dangling pointer errors before attackers exploit them.

One promising approach set in this direction is implemented in an existing tool called Undangle [2], which is capable of detecting UAF vulnerabilities by analysing the dynamic information flow of a program's execution. More specifically, Undangle carries out dynamic taint analysis [4] to inspect interesting data flows that create, propagate, dangle, and dereference pointers. It examines the execution trace of a program to track information related to pointers, such as their locations and whether they are live (point to live memory) or dangling (point to freed memory). Through this information, Undangle checks whether dereferenced pointers are dangling, and as a result, enables the detection of UAF vulnerabilities. In the case that a vulnerability is identified, a detailed report, describing the creation and use of the dangling pointer, is produced in effort to aid analysis.

Although the potential of Undangle has already been demonstrated, it leaves ample space for improvement in terms of scope and effectiveness. Firstly, Undangle takes an offline approach to inspect dynamic information flows by making use of pre-generated execution traces of the program under analysis. As a result, it requires performing instruction emulation in order to recover program states that are no longer available during analysis time. In addition, the original work lacks clear specification on how Undangle examines information flows, thus rendering its understanding and replication very difficult to achieve. Lastly, memory management functions, from which pointer data is introduced, need to be manually specified by the analyst. If some memory management functions are not specified for monitoring, possibly due to the analyst being unaware that the application uses an undocumented custom allocator, false negatives may occur.

We propose a tool called SUDUTA (**S**cript **U**AF **D**etection **U**sing **T**aint **A**nalysis) [5], which builds upon Undangle, addressing its limitations. One contribution is that SUDUTA performs analysis in an online fashion, and therefore, unlike Undangle, avoids the need to perform instruction emulation as values can be directly obtained from the current machine context at runtime. Moreover, SUDUTA's technique is described as a set of formal rules that clearly specify how dynamic taint propagation is performed. Lastly, SUDUTA goes through a preliminary stage to identify undocumented custom memory allocators with the aim of increasing pointer coverage. It achieves this by employing a set of heuristics proposed in previous work [3], which are based on various characteristics of memory management functions.

Experimentation results show that SUDUTA managed to identify UAF vulnerabilities present in a number of real-world script-hosting applications, including Internet Explorer 6 (IE 6) and MS Excel 2003, without generating any false positives. These results validate SUDUTA's formal taint propagation rules and online detection capabilities. Interestingly, SUDUTA only manages to detect the vulnerability in IE 6 when enabling the identification of custom memory allocators, thus showing an improvement in detection coverage. Future work will deal with enhancing SUDUTA to support 64-bit applications in order to conduct further evaluation of the technique.

Acknowledgments

The research work disclosed is partially funded by the Master it! Scholarship Scheme (Malta). This Scholarship is part-financed by the European Union – European Social Fund (ESF) under Operational Programme II – Cohesion Policy 2007-2013, "Empowering People for More Jobs and a Better Quality Of Life."

The final publication, titled "SUDUTA: Script UAF Detection Using Taint Analysis", is available through Springer at: http://rd.springer.com/chapter/10.1007%2F978-3-319-24858-5_9

References

[1] J. Afek and A. Sharabani, "Smashing the pointer for fun and profit," in *Watchfire*, 2007.

[2] J. Caballero, G. Grieco, M. Marron, and A. Nappa, "Undangle: Early detection of dangling pointers in use-after-free and double-free vulnerabilities." in *ISSTA*, M. P. E. Heimdahl and Z. Su, Eds. ACM, 2012, pp. 133–143.

[3] X. Chen, A. Slowinska, and H. Bos, "Who allocated my memory? Detecting custom memory allocators in C binaries," in *Reverse Engineering (WCRE), 2013 20th Working Conference on*, Oct 2013, pp. 22–31.

[4] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 317–331.

[5] Galea, John, and Mark Vella. "SUDUTA: Script UAF Detection Using Taint Analysis." Security and Trust Management. Springer International Publishing, 2015. 136-151.