POSTER: Transforming Code to Drop Dead Privileges

Xiaoyu Hu
BitFusion.io Inc.

Jie Zhou
Department of Computer Science
University of Rochester

Spyridoula Gravani
Department of Computer Science
University of Rochester

John Criswell
Department of Computer Science
University of Rochester

***Abstract*—To help programmers write programs that follow Saltzer and Schroeder's Principle of Least Privilege, modern operating systems divide the power of the administrative user into separate privileges which applications can enable on demand and permanently discard when no longer needed. However, using such privileges requires tedious reasoning of program behavior.**

We present a compiler, named AutoPriv, that uses whole-program analysis to transform programs to remove unneeded privileges during their execution. We tested AutoPriv on several privileged open-source programs that typically run as root. Our results show that AutoPriv increases optimization time by 19% on average but that transformed programs exhibit practically no overhead.

This poster is for an accepted paper.

I. Introduction

Saltzer and Schroeder's Principle of Least Privilege [1] dictates that software should only have the privileges that it needs to operate correctly. Permanently removing privileges is especially important for programs written in type-unsafe programming languages. If such a program has a memory safety error, an attacker could use a code-reuse attack [2], [3] to use privileges that should be disallowed.

To help programmers adhere to the Principle of Least Privilege [1], modern operating systems like Linux [4] and Windows [5] divide the power of the administrative user into separate privileges. However, it's difficult to manually write code that meets the principle because determining when a program can safely remove a privilege requires reasoning about the program's use of system calls across function boundaries and compilation units. Programmers need an automated tool that locates program points at which privileges can be disabled permanently.

In this poster, we present a tool named AutoPriv that transforms a program to eliminate privileges when no longer needed. AutoPriv utilizes inter-procedural compiler analysis to determine where privileges are used and at what program points they can be safely removed.

II. Design and Implementation

AutoPriv uses iterative inter-procedural data-flow analysis. To determine at which points in a program which privileges can be removed, AutoPriv must determine which privileges can still be used at each point in the program. We define the *live privileges* at a program point p to be the privileges that may still be used along some path in the program. Our live privileges definition is analogous to the definition of *live variables* in live variable analysis [6].

We have developed an inter-procedural, flow-insensitive, context-insensitive live privilege analysis based on the standard iterative data-flow analysis framework developed by Kam and Ullman [6]; this analysis computes the live privilege sets at the beginning and end of each basic block. Our compiler then uses this information to locate points in a program at which the live privilege sets change in order to locate at which program points privileges can be safely removed.

In summary, given a program, AutoPriv will analyze the privilege use, compute the sets of privileges that are live on entry and exit to each basic block, add code to remove privileges when they are no longer needed, and generate a final executable. We implemented AutoPriv as a set of new compiler passes for LLVM 3.7.1 [7].

III. Performance Experiments

We studied both AutoPriv's performance when compiling programs and the performance of programs transformed by it. We used five Linux applications: `ping`, `thttpd`, `sshd`, `passwd`, and `su`. We selected these programs because they run as the `root` user on Unix systems in order to override one or more of the Unix access controls.

To measure analysis time, we used the LLVM `opt` tool to run our global live privilege analysis passes on each test program. Our results show that AutoPriv induces an average overhead of 19% across our benchmarks on optimization time.

To measure the performance overhead that AutoPriv induces on the programs it compiles, we compiled test programs with and without AutoPriv's transformation passes and evaluated their performance. Our results show that AutoPriv incurs no overhead on program performance.

IV. Conclusion

This paper describes the AutoPriv compiler which analyzes privilege use in applications and transforms them to permanently remove privileges when no longer needed. AutoPriv incurs, on average, 19% overhead during optimization and induces practically no overhead in the programs that it compiles.

References

[1] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[2] A. One, “Smashing the stack for fun and profit,” *Phrack*, vol. 7, November 1996. [Online]. Available: http://www.phrack.org/issues/49/14.html
[3] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, “Return-oriented programming: Systems, languages, and applications,” *ACM Transactions on Information Systems Security*, vol. 15, no. 1, pp. 2:1–2:34, Mar. 2012.
[4] D. P. Bovet and M. Cesati, *Understanding the LINUX Kernel*, 2nd ed. Sebastopol, CA: O’Reilly, 2003.
[5] M. E. Russinovich and D. A. Solomon, *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server(TM) 2003, Windows XP, and Windows 2000 (Pro-Developer)*. Redmond, WA, USA: Microsoft Press, 2004.
[6] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of Program Analysis*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999.
[7] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis and transformation,” in *Proceedings of the Conference on Code Generation and Optimization*, San Jose, CA, USA, Mar 2004, pp. 75–88.