Diss. ETH No. 20442

Safe Loading and Efficient Runtime Confinement: A Foundation for Secure Execution

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by
Mathias J. Payer
Master of Science ETH in Computer Science, ETH Zurich
born April 29, 1981
citizen of the Principality of Liechtenstein

accepted on the recommendation of
Prof. Dr. Thomas R. Gross, examiner
Prof. Dr. Srdjan Capkun, co-examiner
Prof. Dr. Steve Hand, co-examiner

2012

Abstract

Protecting running applications is a hard problem. Many applications are written in a low-level language and are prone to exploits. Bugs can be used to exploit the application and to run malicious code. A rigorous code review is often not possible due to the size and the complexity of the applications. Even a detailed code review does not guarantee that all bugs in the application are found.

This thesis presents a model for the secure execution of untrusted code. The model assumes that the application code contains bugs but that the application is not malicious (i.e., malware). The application is safe if the model protects from all attack vectors through code-based or data-based exploits in the untrusted code. The model verifies all code prior to execution and ensures that no unchecked control flow transfers are possible. An important design decision is to use a dynamic approach for the implementation with minimal impact on the original applications. Binary only applications are executed without static recompilation or changes to the compiler toolchain (e.g., no recompilation is needed and features like dynamically loaded libraries, lazy binding, or hand written assembly code are still usable).

A dynamic, transparent sandbox in user-space loads and verifies code using binary translation. A secure loader starts the sandbox and bootstraps the application and all needed libraries in the sandbox. The sandbox checks the application code before it is executed and adds security guards during the translation. The combination of the secure loader and the sandbox protects from code-oriented exploits. System calls are redirected by the sandbox to a policy-based system call authorization layer that verifies every system call towards a policy. Every control flow transfer in the application code is verified using a dynamic control flow model. Control flow transfers to illegal locations or instructions that are not legal in the application stop the program. The combination of a system call policy and control flow integrity protects the application from code-based and data-based exploits.

A prototype implementation is used to evaluate the performance and effectiveness of the proposed model. We show that the overhead for our prototype implementation is low and that the model protects from all code-based exploits. The control flow model restricts the attack space for data-based attacks and restricts control flow transfers of the application to well-known and valid locations. The small and modular trusted computing base enables code reviews and allows additional security modules (e.g., a module that detects file-based race conditions).

Zusammenfassung

Der Schutz von laufenden Anwendungen ist ein komplexes Problem. Viele Anwendungen wurden in einer systemnahen Sprache geschrieben und sind fehleranfällig. Diese Programmierfehler können ausgenützt werden um eine Anwendung anzugreifen und schadhafte Befehle auszuführen. Eine detaillierte Überprüfung des Quellcodes ist, wegen der Grösse und Komplexität der Applikationen, oft nicht möglich. Ausserdem garantiert eine detaillierte Überprüfung nicht, dass alle Fehler in der Applikation gefunden wurden.

Diese Doktorarbeit präsentiert ein Modell für die sichere Ausführung von ungesichertem Programmcode. Das Modell trifft die Annahme, dass der Programmcode der Anwendung Fehler enthalten kann, die Anwendung selbst jedoch nicht bösartig ist (d.h. die Anwendung ist selbst keine Malware). Die Anwendung gilt als gesichert, falls das Modell vor allen Angriffsvektoren durch Code-basierte und Daten-basierte Angriffe im ungesicherten Programmcode schützt. Das Modell stellt sicher, dass jede Codesequenz geprüft wird, und garantiert, dass keine ungeprüfte Verzweigung möglich ist. Eine wichtige Designentscheidung ist es, einen dynamischen Ansatz für die Implementierung zu wählen, welcher nur minimalen Einfluss auf die Anwendung hat. Ausführbare Programme ohne Quellcode werden ohne statische Kompilierung oder Veränderungen des Kompilierers ausgeführt. Dieser Ansatz ermöglicht die Ausführung von geschützten Programmen ohne erneute Kompilierung und erlaubt Funktionen wie dynamische Bibliotheken, verzögertes Laden und handoptimierten Assemblercode.

Eine dynamische, transparente Sandbox im Userland lädt und verifiziert den Programmcode mittels binärer Übersetzung. Ein abgesichertes Programm startet die Sandbox und initialisiert die Anwendung und alle benötigten Bibliothekten innerhalb der Sandbox. Die Sandbox überprüft den Programmcode bevor er ausgeführt wird und fügt während der Übersetzung dynamische Sicherheitschecks hinzu. Die Kombination von sicherem Startprogramm und der Sandbox schützt vor allen Code-orientierten Angriffen. Systemaufrufe werden von der Sandbox zu einer regelbasierten Autorisierung für Systemaufrufe umgeleitet. Jede Verzweigung im Programmcode wird anhand eines Kontrollflussmodells verifiziert. Verzweigungen zu illegalen Zielen oder Instruktionen beenden die Anwendung. Die Kombination einer regelbasierten Autorisierung für Systemaufrufe mit den Kontrollflusstests schützt die Anwendung vor Code-basierten und Daten-basierten Angriffen.

Die Effektivität und Effizienz des Modells wird anhand einer Prototypenimplementierung evaluiert. Wir zeigen, dass der zusätzliche Berechnungsaufwand klein ist und dass das Modell vor allen Code-basierten Angriffen schützt. Das Kontrollflussmodell verringert die Angriffsmöglichkeiten von Daten-basierten Angriffen und beschränkt die möglichen Verzweigungen auf bekannte und gültige Ziele. Der kleine und modulare Prototyp ermöglicht eine detaillierte Überprüfung des Quellcodes und erlaubt ausserdem zusätzliche Sicherheitsmodule (z.B. ein Modul, welches zeitliche Angriffe auf das Dateisystem erkennt).