Informationsflussanalyse für Java

Erweitertes Abstract

Christian Hammer
Universität Karlsruhe (TH), hammer@ipd.info.uni-karlsruhe.de

1 Problemstellung

Eine Reihe von Mechanismen, die Software-bedingten Sicherheitsproblemen vorbeugen oder wenigstens seine negativen Auswirkungen verhindern sollen, sind heute fester Bestandteil des täglichen Lebens: Zugangskontrolle, Codesignierung, Kryptographie, Virenscanner, Firewalls, automatische Updates, etc.

All diesen Ansätzen ist jedoch gemein, dass sie nur punktuell auf Probleme reagieren, aber nie Sicherheit auf dem ganzen Lebensweg von Daten (end-to-end security) garantieren können. Mit klassischer Sicherheitstechnik lassen sich lediglich Identität und Herkunft einer Software überprüfen, semantische Eigenschaften werden nicht berücksichtigt. So kann beispielsweise die Zugangskontrolle des Betriebssystems nur festlegen, ob bestimmte Personen Zugang zu vertraulichen Daten bekommen dürfen. Nach Freigabe dieser Daten entzieht sich ihre weitere Verwendung (d.h. die eigentliche Programmsemantik) weitgehend der Kontrolle des Benutzers. Dieser kann i.A. nur anhand der Identität des Programmherstellers auf dessen Kompetenz und Gutartigkeit hoffen. Allerdings wurden in der Vergangenheit einige Beispiele bekannt, bei denen selbst namhafte Hersteller Daten über die Benutzer gesammelt und an sich weitergeleitet haben. Meist hat der Benutzer auf diese Aktivitäten keinen Einfluss.

2 Zielsetzung

Um end-to-end Security zu erreichen, wurde die Informationsflusskontrolle (information flow control, IFC) erfunden. Diese überprüft ein gegebenes Programm auf die Einhaltung einer bestimmten Sicherheitspolitik, normalerweise eine Variante von Nichtinterferenz. Dazu wird jedem Informationskanal ein gewisses Sicherheitslevel wie z.B. “geheim” oder “öffentlich” zugewiesen. Nichtinterferenz ist dann gegeben, wenn geheime Eingaben die öffentlichen Ausgaben nicht beeinflussen können. Eine Kombination von Nichtinterferenz und Zugangskontrolle garantiert die Sicherheit der Daten auf dem ganzen Verarbeitungspfad. Die klassische Nichtinterferenz ist jedoch für die Praxis zu restriktiv, weil geheime Daten in nichttrivialen Programmen immer auch die öffentlichen Ausgaben in gewissem Maß beeinflussen. Man denke nur an geheime Passwörter zum Login: Das Ergebnis der Überprüfung des eingegebenen Passworts muss veröffentlicht werden. Dazu wurde die sog. Deklassifikation eingeführt. Diese erlaubt unter geeigneten Umständen die Veröffentlichung von Daten, die mit Hilfe von geheimen Daten berechnet wurden. Deklassifikationen und Annotationen bilden die sog. Sicherheitspolitik.

Bisherige Ansätze zur Informationsflusskontrolle verwenden nur ein sehr eingeschränktes Repertoire der Programmanalyse: In der Literatur finden sich fast ausschließlich spezielle Typsysteme, die die klassischen Variablentypen um Sicherheitsannotationen wie “geheim”, “öffentlich” usw. erweitern. Typsysteme sind zwar elegant und effizient zu implementieren, allerdings benötigen sie dazu eine große Zahl von Benutzerannotationen. Dadurch ist der Aufwand, eine Sicherheitspolitik für ein vorhandenes Programm zu definieren, sehr hoch.

3 Lösungsansatz

Die hier vorgestellte Arbeit beschreitet einen neuen Ansatz zur effektiven Informationsflusskontrolle, welcher auf Programmabhängigkeitsgraphen und sog. Program Slicing basiert. Ein vor kurzem präsentiertes Theorem über den Zusammenhang von IFC und Program Slicing mit Abhängigkeitsgraphen lässt erstmals auch präzise Analysen von echten Sprachen zu.

Abhängigkeitsgraphen und Program Slicing sind grundsätzlich präziser als Typsysteme, da sie den Kontrollfluss durch das Programm berücksichtigen (Fluss-Sensitivität), verschiedene Aufrufkontexte einer Prozedur unterscheiden (Kontext-Sensitivität) und bei objektorientierten Programmen sogar zwischen den Feldern unterschiedlicher Objekte differenzieren (Objekt-Sensitivität). In dieser Arbeit untersuchen wir realistische Programme bzw. deren Sicherheitskerne, die in Java Bytecode vorliegen. Anhand der Bytecode-Semantik wird ein präziser Programmabhängigkeitsgraph erzeugt, der alle zur Laufzeit möglichen Informationsflüsse abbildet. Ausgehend von dem generierten Graphen wird eine auf Program Slicing basierende Analyse beschrieben, die klassische Nichtinterferenz garantiert. Außerdem wurde eine Möglichkeit zur Deklassifikation geschaffen, um an bestimmten Stellen im Programm die Sicherheitsstufe von Informationen herabzustufen.

Um die Bedienbarkeit zu erhöhen wurden die vorgestellten Techniken außerdem in Eclipse-Plugins integriert.