Garantiert kein Stacküberlauf

Alle aktuellen Sicherheitsstandards fordern den Nachweis oberer Schranken des maximalen Stackverbrauchs. Das Fehlen eines solchen Nachweises kann als Indiz für Fahrlässigkeit gewertet werden. Mit dem StackAnalyzer von AbsInt kann der maximale Stackverbrauch berechnet werden – und zwar meist bytegenau.

Der Laufzeitstack ist ein grundlegendes Ausführungskonzept aller moderner Programmiersprachen: er wird vom Laufzeitsystem zur Verwaltung der aufgerufenen Funktionen und zur Auswertung von Ausdrücken verwendet. In sicherheitskritischen Systemen ist der Stack in der Regel der einzige dynamisch verwaltete Speicherbereich. Dennoch muss der maximale Stackverbrauch jeder Task statisch bekannt sein, da er zur Konfigurationszeit des Systems festgelegt werden muss. Wird der maximale Stackverbrauch unterschätzt, kann es zu Stacküberläufen kommen. Stacküberläufe sind schwerwiegende Fehler, die oft schwer zu reproduzieren und zu beheben sind, siehe hierzu [1].

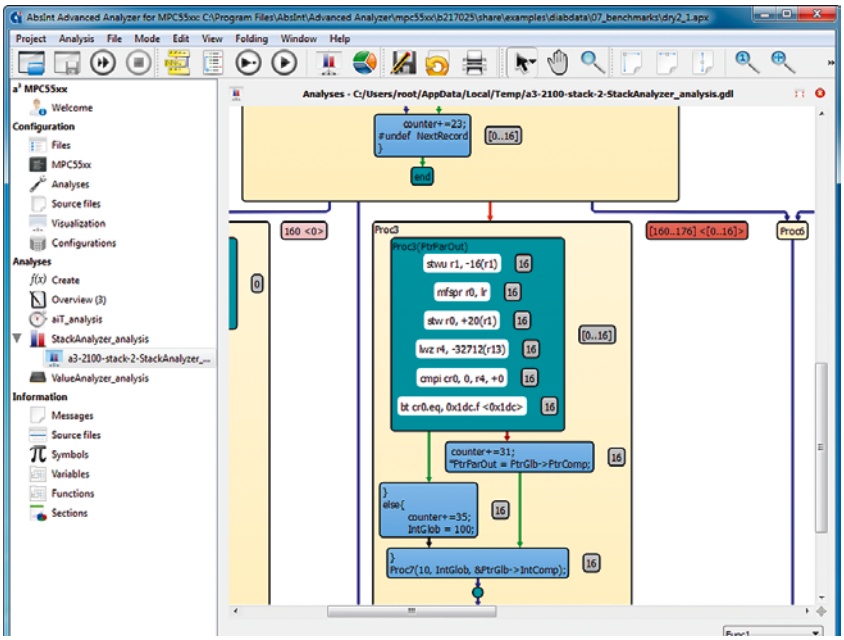

Bild 1: Visualisierung des Kontrollflussgraphen mit Stackhöhenannotationen.

Eine Verwendung dynamischer Testverfahren zur Bestimmung des maximalen Stackverbrauchs ist problematisch, da keine sicheren Testende-Kriterien zur Verfügung stehen. Stacküberläufe treten oft in Randfällen auf, z. B. im Kontext von Fehlerbehandlungsmechanismen, die schwer zu stimulieren sind und in der Regel nicht vollständig getestet werden können [2]. Quellcode-Analysatoren sind zur Untersuchung des Stackverbrauchs ungeeignet, da der Einfluss des Compilers nicht präzise berücksichtigt werden kann. Innerhalb eines Compilers stehen in der Regel präzise Informationen über das Stacklayout zur Verfügung; manche Compiler bieten die Möglichkeit, diese Informationen zu exportieren. Sie können allerdings nicht die Auswirkungen von Inline-Assembly oder Linker-Optimierungen berücksichtigen. Industrieanwendungen, z. B. im Automobilbereich, beinhalten in der Regel Bibliotheken und Objektcode verschiedener Zulieferer. Die Stackeffekte dieser Softwareteile können vom Compiler nicht präzise berücksichtigt werden. Zudem ist bei einer Verwendung von Stackschätzungen des Compilers keine Unabhängigkeit zwischen Codegenerator und Verifikati- »

» **Der StackAnalyzer arbeitet auf Maschinencode-Ebene, d. h. die Eingabe besteht aus einer vollständig gelinkten ausführbaren Binärdatei, dem Executable des Zielsystems“.**

Dr. Daniel Kästner, CTO der AbsInt Angewandte Informatik GmbH.

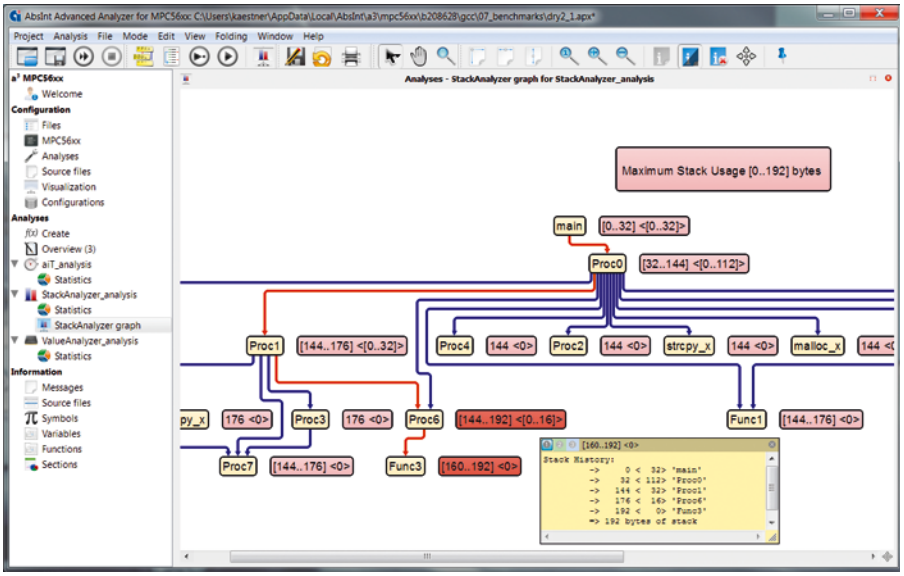

Bild 2: Visualisierung des Callgraphen mit Anzeige des kritischen Pfades.

on des Stackverbrauchs gegeben, was für die funktionale Sicherheit ebenfalls ein relevanter Aspekt sein kann.

Statische Analyse

Durch statische Programmanalyse auf Basis der abstrakten Interpretation können sichere obere Schranken des maximalen Stackverbrauchs jeder Task und jeder Interrupt-Service-Routine (ISR) bestimmt werden. Ein solcher statischer Analysator ist der StackAnalyzer von AbsInt [3]. Er arbeitet auf Maschinencode-Ebene, d. h. die Eingabe besteht aus einer vollständig gelinkten ausführbaren Binärdatei, dem Executable des Zielsystems. In einem ersten Schritt wird das Executable decodiert und eine Liste der möglichen Analysestartpunkte erzeugt, die den Eintrittspunkten von Tasks und ISRs entsprechen. Anschließend wird der Kontrollflussgraph der gewählten Tasks oder ISRs erzeugt, der alle möglichen Ausführungspfade innerhalb der Task oder ISR repräsentiert. Die darauffolgende Wertebereichsanalyse berechnet statisch die möglichen Wertebereiche von Registerinhalten und Speicherzellen für jeden möglichen Programmpunkt in jedem möglichen Ausführungskontext. Dies ist die Voraussetzung, um die Adressen von Datenzugriffen, berechneten Sprüngen oder indirekten Funktionsaufrufen vorhersagen zu können und um ungültige Pfade zu entdecken. Zusätzlich wird eine Schleifengrenzenanalyse für stack-modifizierende Schleifen durchgeführt, z. B. wenn Funktionsargumente in einer Schleife auf den Stack gelegt werden. Mit diesen Informationen kann der maximale Stackverbrauch berechnet werden – und zwar meist bytegenau [2].

Die Herausforderung für Binärcode-Analysatoren liegt in der Minimierung notwendiger Benutzerinteraktionen, z. B. zur Angabe der möglichen Zielfunktionen von Funktionspointer-Aufrufen, falls diese nicht statisch bestimmt werden können. Zur Minimierung dieses Interaktionsaufwands muss die Analyse sehr präzise sein, und der Annotationsmechanismus muss flexibel und leicht zu verwenden sein. StackAnalyzer arbeitet daher kontextsensitiv, d. h., bei jedem Funktionsaufruf wird präzise die Stackhöhe an der jeweiligen Aufrufstelle berücksichtigt. Zudem werden Wertebereichsanalyse und Decoding in einer Feedbackschleife durchgeführt, die es ermöglicht, Ergebnisse der Wertebereichsanalyse in der Decoding-Stufe zu verwenden, z. B. zur Berechnung der Ziele von Funktionspointer-Aufrufen. Sind Benutzerannotationen nötig, steht hierzu die Annotationssprache AIS zur Verfügung, die eine knappe und prägnante Spezifikation der benötigten Informationen ohne Veränderung des Executables ermöglicht [2].

Stackmaximum bei AUTOSAR

Während StackAnalyzer bei synchronen Systemen mit statischen Schedules direkt das globale Stackmaximum berechnet, muss bei dynamisch geplanten Systemen das Task-Scheduling berücksichtigt werden. Beispiele sind die vom OSEK [4] bzw. AUTOSAR [5] definierten Systemarchitekturen, die im Automobilbereich verbreitet eingesetzt werden. Ein Basiskonzept des OSEK-Standards ist, dass Tasks mit statischen Prioritäten unter Verwendung des Priority Ceiling Protokolls angeordnet werden. Zusätzlich können mehrere Prioritätsstufen für Interrupts vergeben werden, die ebenfalls statisch zugeordnet werden können. Aus der OSEK-Konfiguration ergibt sich eine Formel, die angibt, wie aus den Stackmaxima der einzelnen Tasks bzw. ISRs, ihren Prioritäten und der Größe der Zwischenframes bei Kontextwechseln das globale Stackmaximum ermittelt werden kann [2]. StackAnalyzer bietet die Möglichkeit, die Systemformel zu spezifizieren und

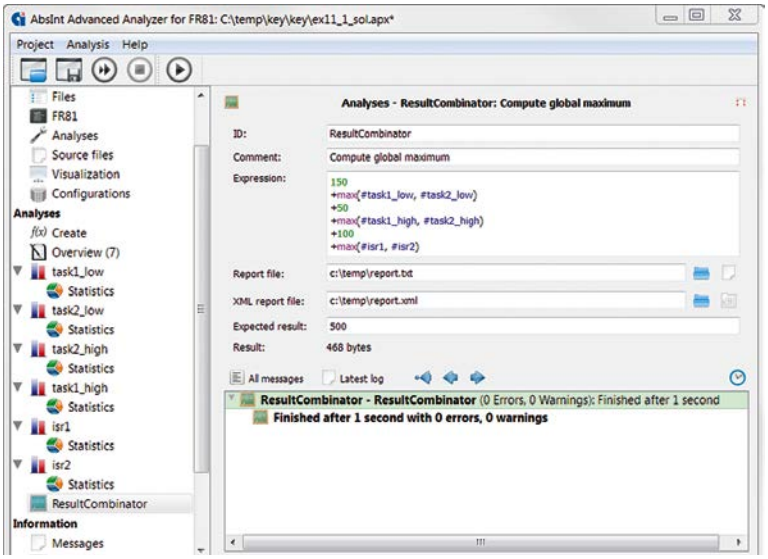

Bild 3: Ansicht des ResultCombinators zur Berechnung des Systemmaximums aus den Stackmaxima der einzelnen Tasks und ISRs.

automatisch aus den Stackmaxima der einzelnen Tasks und ISRs das Systemmaximum zu berechnen.

Neben der Berechnung der maximalen Stackhöhe liefert StackAnalyzer auch Feedback zur Optimierung des Stackverbrauchs, z. B. durch eine Visualisierung des kritischen Pfades in Aufruf- und Kontrollflussgraph mit annotierten Stackhöhen. Durch ein offenes XML-basiertes Austauschformat (XTC-Format [6]) kann der StackAnalyzer nahtlos in andere Entwicklungswerkzeuge integriert werden. Unter anderem existieren Tool-Kopplungen mit den modellbasierten Codegeneratoren Esterel SCADE [7] und dSpace TargetLink [8], die es ermöglichen, Stackprobleme frühzeitig im Entwicklungsprozess zu erkennen und auf Modellebene zurückzumelden. StackAnalyzer arbeitet ausschließlich auf den Maschineninstruktionen des generierten Zielcodes und benötigt daher weder Code-Instrumentierung noch Debug-Informationen. Linker-Optimierungen, Inline-Assembly und Bibliotheksaufrufe werden sicher berücksichtigt.

Fazit

Das Verfahren der abstrakten Interpretation ermöglicht den formalen Beweis, dass der maximale Stackverbrauch der analysierten Task/ISR niemals unterschätzt wird. Die Gefahr von Stacküberläufen im laufenden Betrieb kann somit sicher vermieden werden. Für StackAnalyzer sind Qualification Support Kits verfügbar, die eine automatische Toolqualifizierung nach allen gängigen Sicherheitsnormen ermöglichen [2]. Die Konfiguration des Analysators wird einmal für jedes Softwareprojekt erstellt; danach kann die Analyse automatisch ausgeführt werden. Dies ermöglicht eine kontinuierliche Verifikation, z. B. im Rahmen automatisierter nächtlicher Analyseläufe. ■ (oe)

» www.AbsInt.com

Literatur/Quellen

[1] Michael Dunn. Toyota's killer firmware: Bad design and its consequences. EDN Network, October 28, 2013.

URL: http://www.edn.com/design/automotive/4423428/Toyota-s-killer-firmware--Bad-design-and-its-consequences

[2] D. Kästner, C. Ferdinand. Proving the Absence of Stack Overflows. In SAFECOMP ,14: Proceedings of the 33th International Conference on Computer Safety, Reliability and Security (SAFECOMP), Florence, 2014.

[3] AbsInt GmbH. StackAnalyzer Website. URL: http://www.absint.com/stackanalyzer

[4] OSEK/VDX. OSEK/VDX Operating System. Version 2.2.3., 2005.

[5] AUTOSAR Development Partnership. Automotive Open System Architecture (AUTOSAR). URL: http://www.autosar.org.

[6] AbsInt GmbH. XTC Language Specification Version 2.1. URL: http://www.absint.com/xtc.

[7] Esterel Technologies. SCADE Suite. URL : http://www.esterel-technologies.com/products/scade-suite.

[8] dSpace GmbH. TargetLink Website. URL: http://www.dSpace.com/go/TargetLink.

Dr. Daniel Kästner ist Mitgründer und CTO der Firma AbsInt Angewandte Informatik GmbH.