Sicherheit lässt sich programmieren

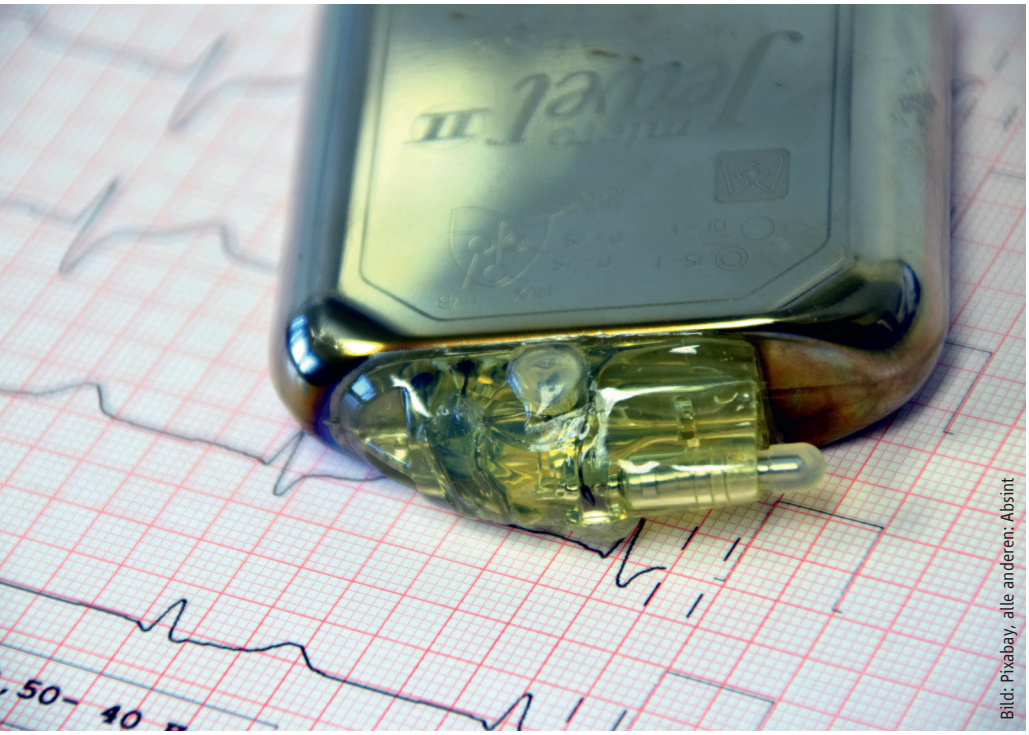


Bild: Pixabay, alle anderen: Absint

Eine Fehlfunktion implantierter Medizingeräte kann schwerwiegende Folgen haben. Die Sicherheit dieser Geräte sowohl für die Hardware als auch die Steuerungssoftware muss daher gewährleistet sein. Durch statische Codeanalyse lassen sich zahlreiche kritische Programmfehler und -verwundbarkeiten ausschließen.

Daniel Kästner
CTO bei Absint

Wurde Software vor einigen Jahren lediglich als kleiner Zusatz zur elektronischen Hardware wahrgenommen, übernimmt sie nun wichtige Teile der Funktionalität und trägt erheblich zur Wertschöpfung des Gesamtprodukts bei – auch in der Medizintechnik. Insbesondere wird immer mehr sicherheitskritische Funktionalität durch Software realisiert. Ein weiterer industrieübergreifender Trend ist die zunehmende Vernetzung von Geräten, die die Gefahr von Cyberattacken massiv erhöht. Eine Fehlfunktion oder bösartige Manipulation der Steuerungssoftware kann im Extremfall Menschenleben gefährden, aber auch in weniger kritischen Systemen hohe Kosten verursachen – nicht zuletzt durch Rückrufaktionen.

Allein im Zusammenhang mit Infusionspumpen wurden nach Aussage der FDA (US Food and Drug Administration) zwischen 2005 und 2009 mehr als 500 Tote registriert. Im gleichen Zeitraum wurden aus Sicherheitsgründen 87 Rückrufe von Infusionspumpen durchgeführt. Eine Analyse der schwerwiegenden Ereignisse durch die FDA hat Softwarefehler als eine der Hauptursachen identifiziert. Seitdem ist die Zahl weiter gestiegen: Aufgrund von Software-Defekten wurden zwischen 2011 und 2015 durch die FDA 627 Software-Geräte zurückgerufen, zwölf davon mit der höchsten Gefährdungsstufe. Zusätzlich steigt durch die zunehmende Vernetzung elektronischer Geräte die Gefahr von Cybersecurity-Risiken an. Der jüngste relevante Fall ist eine im Juli 2019 von der FDA veröffentlichte Gefährdungswarnung vor Insulinpumpen aufgrund von Cybersecurity-Verwundbarkeiten. Besonders schwerwiegend sind solche Verwundbarkeiten in Geräten, bei denen durch Cyberangriffe Gesundheit und Leben von Patienten gefährdet werden können, was insbesondere bei implantierten Geräten oft der Fall ist. In solchen Fällen stellt ein Cybersicherheits-Risiko auch ein Risiko im Sinne der funktionalen Sicherheit dar.

Dies unterstreicht die zentrale Bedeutung einer systematischen Verifikation und Validierung von Medizinsoftware. Das Ziel der Software-Verifikation liegt in der Bereitstellung eines objektiven Nachweises, dass die relevanten Anforderungen an die Software erfüllt worden sind und keine Sicherheitsverletzungen auftreten können. Die Software-Validierung prüft zusätzlich, ob die gestellten Anforderungen konsistent und vollständig sind und das gewünschte Verhalten abbilden.

Normenlandschaft zur funktionalen und Cyber-Sicherheit

Der Software-Entwicklungsprozess wird durch nationale und internationale Normen und Vorschriften reguliert. Bei Medizinprodukten ist in Deutschland der Nachweis der Übereinstimmung mit dem Medizinproduktegesetz (MPG) erforderlich. Darin wurde die Richtlinie 2007/47/EG in deutsches Recht übernommen, die in Abs. 20) explizit fordert, Software für Medizinprodukte – sowohl als eigenständiges Element wie auch als Bestandteil eines Medizinproduktes – in Übereinstimmung mit dem Stand der Technik zu validieren. Grundsätzlich unterliegen alle elektrisch-/elektronischen Systeme der Norm

IEC 61508, von der zum Beispiel im Automobilsektor die branchenspezifische Norm ISO 26262 oder im Bahnsektor die Norm EN 50128 abgeleitet wurden. Die höchsten Anforderungen sind mit der Norm DO-178C im Luftfahrtbereich gestellt; auch die Zertifizierungsprozesse sind dort deutlich strikter als im Medizintechnikbereich.

Im Medizinbereich formulieren EN 60601-1 und IEC 61010-1 allgemeine Anforderungen für die Herstellung funktional sicherer Medizingeräte. Der Schwerpunkt der IEC 62304 liegt auf Software für Medizingeräte und definiert insbesondere einen Lebenszyklus für Softwareentwicklung mit Fokus auf komponentenorientierte Software-Architekturen und Software-Wartung. Die Vorgaben für Verifikation und Validierung sind jedoch sehr allgemein gehalten und im Detailgrad nicht mit IEC 61508, ISO 26262 oder EN 50128 vergleichbar. EN 45502-1 und ISO 14708-1 geben spezifische Leitlinien für aktive implantierte Geräte und verweisen dabei bezüglich Software-Anforderungen auf die IEC 62304. Mit dem Risikomanagement für Medizingeräte befasst sich die ISO 14971. Die Norm ISO 13485 betrachtet allgemeines Qualitätsmanagement. Zu den relevanten nationalen Richtlinien im Bereich Verifikation und Validierung zählen insbesondere auch die US Quality System Regulations (siehe Title 21 CFR Part 820 und 61 FR 52602) und die FDA General Principles of Software Validation.

Aktuelle Richtlinien im Bereich Cybersicherheit sind vornehmlich für den Bereich der Datenverarbeitung ausgelegt. Die Norm ISO 15408 (Common Criteria) definiert allgemeine Kriterien für die Bewertung der Sicherheit von Informationstechnologie, der Standard ISO/IEC 27001 regelt die Anforderungen für Informationssicherheits-Managementsysteme im Organisationskontext, IEC 62443 fokussiert auf Cybersicherheit für industrielle Kommunikationsnetze. Die Interaktion zwischen funktionaler Sicherheit und Cybersicherheit ist Gegenstand aktueller Normungsaktivitäten. Im Automobilbereich wird funktionale Sicherheit durch die ISO 26262 geregelt; im Bereich Cybersicherheit wurde 2016 der Praxisleitfaden SAE J3061 veröffentlicht, eine ISO-Norm wird derzeit von der ISO-Arbeitsgruppe ISO/SAE 21434 entwickelt. Ähnliche Aktivitäten gibt es auch in anderen Industriebereichen, zum Beispiel IEC TR 63069 im Bereich der industriellen Mess-, Steuer-, Regel- und Automatisierungstechnik.

Verifikation und Validierung

Gemeinsam ist allen Safety-Normen und -Regulierungen im Software-Bereich, dass sie die Identifikation funktionaler und nicht-funktionaler Gefahrenstellen fordern sowie den Nachweis, dass die Software die relevanten Sicherheitsziele erfüllt. Das gilt ohne Einschränkung auch für den Bereich der Medizinsoftware, auch wenn der Detailgrad der Anforderungsbeschreibung deutlich hinter anderen Industrienormen liegt.

Es wird gefordert, Codier-Richtlinien einzuhalten, die das Risiko von Programmierfehlern minimieren. Verbreitete Standards sind insbesondere MISRA C, SEI CERT C, ISO/TS 17951 und CWE (Common Weakness Enumeration). Hierbei gibt es eine

EXKURS THERAC-25: DIE PROGRAMMIERTE KATASTROPHE

Das Therac-25 war ein computergesteuertes Bestrahlungsgerät zur Behandlung von Tumoren. Es war das dritte Gerät der Serie, entwickelt und gebaut von der kanadischen Regierungsfirma Atomic Energy of Canada Limited (AECL). Insgesamt wurden elf Geräte vom Typ Therac-25 in den USA und Kanada vertrieben.

Eine Kombination aus Software-Fehlern und mangelnder Qualitätskontrolle endete beim Therac-25 fatal. So schrieb ein einziger Programmierer den Code für das Gerät, basierend auf den Programmen, die in Vorgängermodellen verwendet wurden. Er ging davon aus, dass Software nicht »altern« könne. Tatsächlich führte das überarbeitete Betriebsprogramm dazu, dass die Strahlendosen des Therac-25 zu hoch waren: Jedes Mal, wenn der behandelnde Arzt bei der Eingabe der Parameter eine bestimmte Pfeiltaste benutzte, führte das bei der Bestrahlung zur »Fehlfunktion 54« (Bildschirmmeldung). Doch weil die mitgelieferte Dokumentation diesen Fehler nicht beschrieb, wurden die Patienten einfach weiter beschossen – mit überdosierter Strahlung (25 Millionen Elektronen-Volt).

Zwischen 1985 und 1987 kam es insgesamt zu sechs massiven Überdosen, die bei drei Patienten zum Tode führten. Wobei die ersten Vorfälle seitens der Regierung und des Herstellers nicht untersucht wurden, da sie eine Fehlfunktion des Gerätes ausschlossen. Erst 1987 reagierte der Hersteller und rief das Gerät zurück.

starke Überlappung zwischen Safety- und Security-Anforderungen, wobei die meisten der genannten Codierstandards beide Bereiche abdecken. Eine automatisierte Prüfung durch Softwarewerkzeuge (wie RuleChecker von Absint) hat sich in allen sicherheitskritischen Industriebereichen als fester Teil des Entwicklungsprozesses durchgesetzt.

Zum Korrektheitsnachweis für funktionale Programmeigenschaften haben sich automatische und modellbasierte Testverfahren neben formalen Methoden wie Model Checking (Modellprüfung) etabliert. Bei der Systemkorrektheit spielen jedoch auch sicherheitsrelevante nichtfunktionale Qualitätseigenschaften eine wichtige Rolle. Es ist offensichtlich, dass ein Programm nicht wegen Speicherfehlern abstürzen darf, nicht aufgrund von Security-Verwundbarkeiten angreifbar sein darf, und dass die Reaktionen von Echtzeitsoftware innerhalb der vorgegebenen Zeit erfolgen müssen. Die Folgen nichtfunktionaler Fehler können schwerwiegend sein – bekannte Beispiele sind die Explosion der Ariane 5, die unbeabsichtigte Beschleunigung im Toyota Camry sowie die fehlerhafte Strahlendosisberechnung beim Therac-25-Unfall (*Kasten S. 43*).

Statische Analyse durch Abstrakte Interpretation

Ein statischer Analysator ist ein Softwarewerkzeug, das zur Ermittlung der Analyseergebnisse keine Ausführung des untersuchten Programmes benötigt. Statische Analysatoren auf Basis der Abstrakten Interpretation haben sich in den letzten Jahren zum Stand der Technik bei der Verifikation nicht-funktionaler Eigenschaften entwickelt. Diese sogenannten sicheren statischen Analysatoren zählen zu den formalen Verifikationsmethoden.

Die Abstrakte Interpretation ist eine semantikbasierte Methodik für Programmanalysen. Sie ermöglicht volle Kontroll- und Datenabdeckung und kann einen formalen Nachweis von Programmeigenschaften erbringen. Sogenannte False Negatives, also übersehene Fehler, sind ausgeschlossen. Statische Analysen lassen sich leicht automatisieren und in den Entwicklungsprozess integrieren und sie ermöglichen es, Fehler in frühen Entwicklungsphasen zu entdecken. Moderne statische Analysatoren skalieren gut und erlauben die Analyse vollständiger sicherheitskritischer Industrieanwendungen. Die Abstrakte Interpretation eignet sich sehr gut, um nicht-funktionale Softwarefehler auszuschließen. Beispiele von Absint sind sichere statische Analysatoren zum Nachweis der maximalen Ausführungszeit (aiT) und des maximalen Stackverbrauchs (StackAnalyzer) sowie Astrée zum Ausschluss des Auftretens von Laufzeitfehlern und Datenwettläufen.

Ausschluss von Laufzeitfehlern

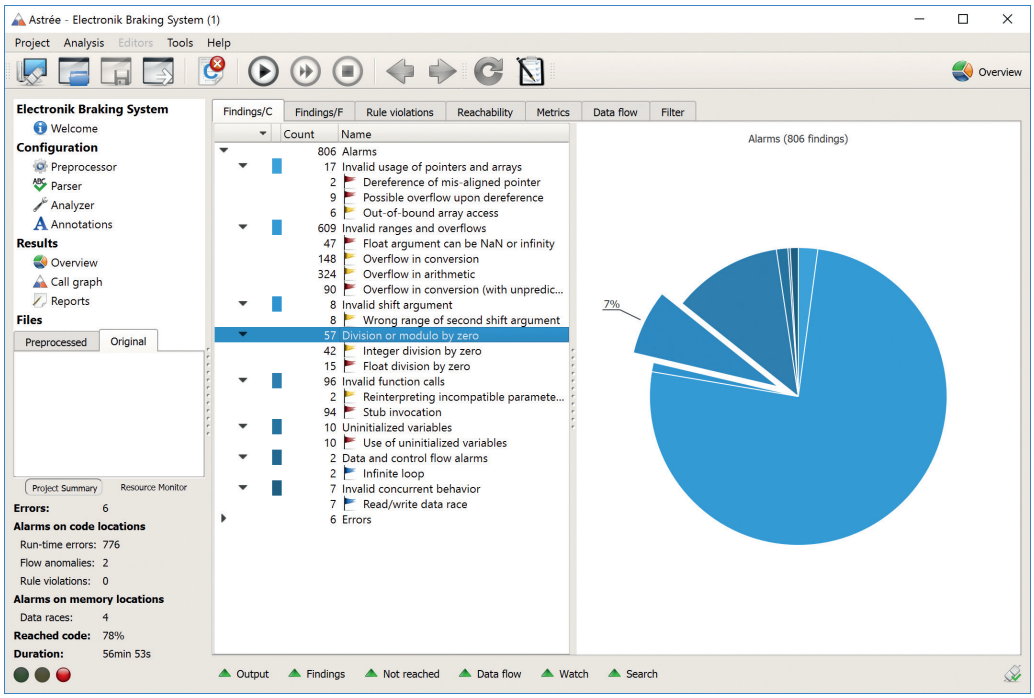
Eine wichtige Klasse kritischer Fehler sind Programmierfehler aufgrund undefinierten oder nicht spezifizierten Verhaltens der Programmiersprache, die sogenannten Laufzeitfehler. Hierzu zählen Feldgrenzenverletzungen, Pufferüberläufe, Nullzeiger und hängende Zeiger, aber auch Datenwettläufe oder Deadlocks. Solche Fehler können die Datenintegrität zerstören, falsche Systemreaktionen verursachen und zu einem Software-Absturz führen. Sie sind gleichzeitig die kritischsten Einfallstore für Cybersecurity-Attacken auf Code-Ebene.

Ein Beispiel für einen statischen Laufzeitfehler-Analysator auf Basis der Abstrakten Interpretation ist das Tool Astrée, das alle potenziellen Laufzeitfehler in C-Programmen entdeckt und damit den sicheren Nachweis der Abwesenheit solcher Fehler ermöglicht. Astrée verfügt über einen leistungsstarken Analysekern, der eine hohe Analysepräzision erreicht, aber dennoch für komplexe Industrieanwendungen skaliert. Beispielsweise konnte eine vollständige kommerzielle Flugzeugsteuerung von über 500.000 Zeilen Code ohne jeden Fehlalarm analysiert werden; das bislang größte analysierte Programm besteht aus 110 Millionen Zeilen präprozessierten Codes, für die knapp zwei Tage Analysezeit erforderlich waren.

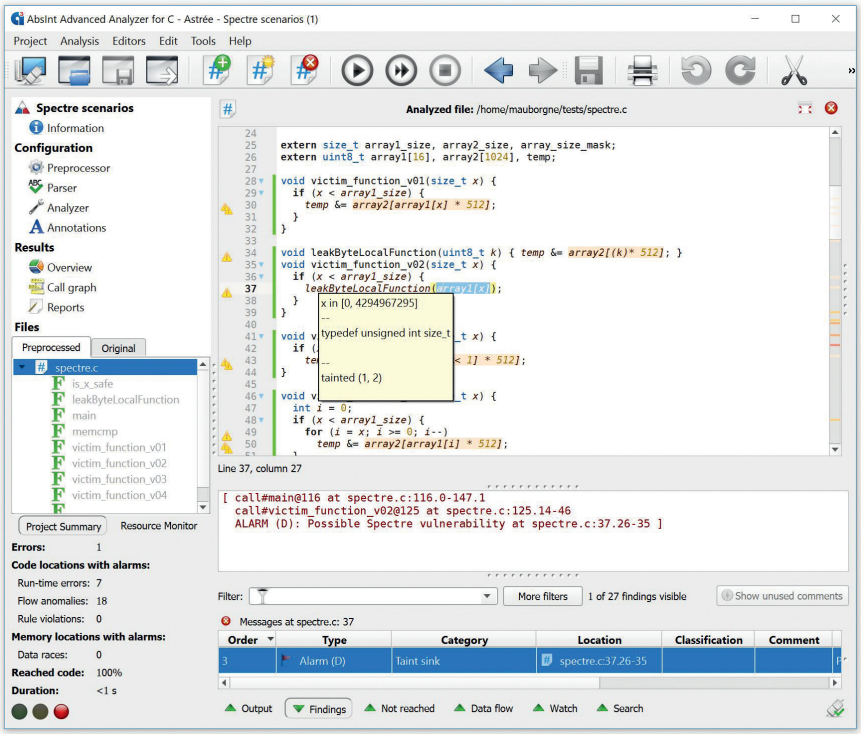
Stack-Überläufe und Laufzeitfehler sind zwei Hauptursachen software-verursachter Speicherkorruption. Die dritte Hauptursache solcher Fehler ist die Fehlcompilierung. Der Compiler erzeugt fehlerhaften Code für ein korrektes Eingabeprogramm. Durch eine formal bewiesenen Compiler wie CompCert kann Fehlcompilierung ausgeschlossen werden.

Daten- und Kontrollfluss-analyse

Wichtig für funktionale Sicherheit und Cybersicherheit ist die Berechnung des Kontroll- und Datenflusses eines Programmes. Kontrollflussgraphen geben Aufschluss, welche Funktionen welche anderen Funktionen aufrufen können, und in welchen Threads diese



Ergebnisdarstellung mit Übersicht der Laufzeitfehler-Kategorien in Astrée.



Astrée-Alarm zur Meldung einer Spectre-Verwundbarkeit.

Aufrufe erfolgen. Datenflussinformationen geben Aufschluss über Zugriffe auf die globalen oder statischen Variablen, von welchen Funktionen oder Threads diese Zugriffe erfolgen, und ob auf eine Variable von mehreren Threads aus zugegriffen wird oder sogar einem Datenwettlauf unterliegt.

Darüber hinaus gestatten statische Analysemethoden, spezifische Daten- und Kontrollflüsse zu berechnen. Program Slicing etwa identifiziert die Programmteile, die den Wert einer Variable an einem bestimmten Programmpunkt beeinflussen können und beantwortet somit die Frage, ob eine gegebene Berechnung von möglicherweise nicht-vertrauenswürdigen Quellen abhängt. Die Auswirkungen von Datenkorruption können durch Taint-Analyse bestimmt werden. Sie ermöglicht es nachzuvollziehen, welche Programmteile durch möglicherweise korrupte Eingangsdaten beeinflussbar sind. Program Slicing und Taint-Analyse werden beispielsweise vom statischen Analysator Astrée unterstützt, der auch eine auf Taint-Analyse basierende Erkennung von Spectre-Verwundbarkeiten bietet.

Zusammenfassung

Sicherheitsorientierte Codierrichtlinien ermöglichen einen auf Minimierung von Programmierfehlern ausgerichteten Programmierstil. Funktionale Korrektheit kann durch formale Verifikationsverfahren bewiesen oder durch automatisierte Testverfahren geprüft werden. Die Abwesenheit von sicherheitskritischen nicht-funktionalen Fehlern, darunter Stacküberläufe, Zeitüberschreitungen, Laufzeitfehler und Datenwettläufe, kann durch statische Analysatoren auf Basis der Abstrakten Interpretation bewiesen werden. Hierdurch können auch zahlreiche kritische Cybersicherheits-Verwundbarkeiten vermieden werden.

Formal verifizierte Compiler stehen zur Verfügung, bei denen mathematische Beweise vorliegen, dass es keine Fehlcompilierung geben kann.

LITERATUR

Das umfangreiche Literaturverzeichnis erreichen Sie über diesen QR-Code: