Auswahl des passenden Security-Programmierstandards

Sucht man nach Security-orientierten Programmierstandards, gibt es mehrere Quellen, angefangen bei den CERT Coding Standards über OWASP und CWE bis zu vielen unterschiedlichen Empfehlungen und Best Practices. Dazu gesellen sich weitere fachspezifische Standards wie AUTOSAR und MISRA, und andere auf Basis der IEC 61508 Norm. Aus dieser Fülle an Informationen genau den für das eigene Projekt am besten geeigneten Programmierstandard herauszufinden ist eine Mammutaufgabe. Muss man sie mitten im SDLC (Software Development Lifecyle) lösen, wenn also das Projekt schon angelaufen ist und die Anpassung von bestehender Software ansteht, ist die Herausforderung noch größer. Mit einer vorausschauenden Planung lässt sich die Aufgabe vereinfachen.

Standards für funktionale Sicherheit (Safety)

Um die funktionale Sicherheit von elektrischen, elektronischen und programmierbaren elektronischen sicherheitsrelevanten Systemen zu gewährleisten, wurde für alle Anwendungsgebiete die IEC 61508 Norm entwickelt. Sie deckt sämtliche Aspekte eines Computersystems ab, ihr Teil 3 (Software Requirements) befasst sich gezielt mit dem Software-Teil des Systems und enthält viele Anforderungen für sämtliche Phasen des Softwareentwicklungs-Lebenszyklus. Die IEC 61508-3, Tabelle A.9 (Software Verification) enthält eine Liste von Techniken, die für die Software-Verifikation empfohlen werden, darunter u. a. die explizite und dringende Empfehlung der statischen Analyse für die höheren SILs (Safety Integrity Levels).

Tabelle 1, Designs and Coding Standards von IEC 61508-3‘ enthält Softwaredesign-Techniken, die für bestimmte SILs empfohlen (Recommended – R) oder dringend empfohlen (Highly Recommended – HR) werden:

Bei der IEC 61508 handelt es sich um eine universelle Norm, die auf die unterschiedlichsten Branchen abzielt und Anwendung findet. Zusätzlich existieren auf bestimmte Einsatzgebiete ausgerichtete Normen wie diese Beispiele:

- Luftfahrt: DO-178C / ED-12C
- Automobil: ISO 26262
- Eisenbahn: IEC 62279 / EN 50128
- Kernkraftwerke: IEC 61513

Auch wenn diese Normen auf die spezifischen Besonderheiten eines

Technik / Maßnahme	SIL 1	SIL 2	SIL 3	SIL 4
Anwendung eines Codierstandards zur Verringerung der Fehlerwahrscheinlichkeit	HR	HR	HR	HR
Keine Verwendung dynamischer Objekte	R	HR	HR	HR
Keine Verwendung dynamischer Variablen	R	HR	HR	HR
Online-Prüfung der Installation dynamischer Variablen	--	R	HR	HR
Eingeschränkte Verwendung von Interrupts	R	R	HR	HR
Eingeschränkte Verwendung von Zeigern	--	R	HR	HR
Eingeschränkte Verwendung von Rekursionen	--	R	HR	HR
Kein unstrukturierter Kontrollfluss in Hochsprachen-Programmen	R	HR	HR	HR
Keine automatische Typumwandlung	R	HR	HR	HR

Tabelle 1

Autor:
Michal Rozenau zeichnet verantwortlich als Project Lead Engineer bei Parasoft. Er ist spezialisiert auf den Einsatz der Produkte von Parasoft in sicherheitsrelevanten Anwendungen gemäß Safety-Standards wie IEC 61508, ISO 26262, DO-178B/C und EN 50128.

Parasoft
www.parasoft.com

Standard	Zahl der Richtlinien	Details
MISRA C 2012	173	156 Regeln, 17 Direktiven
CERT C	307	121 Regeln, 186 Empfehlungen
AUTOSAR	344	319 erforderlich, 25 angeraten
CERT C++	163	83 C++-Regeln, 80 relevante C-Regeln

Tabelle 2

bestimmten Einsatzbereichs abzielen, ist ihr Grundgedanke recht ähnlich – zumindest aus dem Blickwinkel der Software: Alle verlangen die Anwendung der statischen Analyse am entwickelten Code, neben anderen Verifikationstechniken. Zudem bieten sie allgemeine Richtlinien hinsichtlich der zu ergreifenden Maßnahmen, lassen aber gleichzeitig einen breiten Interpretationsspielraum. Normalerweise benennen sie keine bestimmten Codierungs-Konventionen oder Programmierstandards, die angewendet werden sollen, nur die ISO 26262 erwähnt MISRA C als Beispiel einer Codierrichtlinie für die Programmiersprache C.

Sichere Programmierstandards (Security)

Sicherer („secure") Code bedeutet, dass der geschriebene Code keine potenziell ausnutzbaren Schwachstellen oder Sicherheitslücken aufweist und damit nicht angreifbar ist. Zudem muss er einerseits gewisse sichere Muster (im Sinne von „safe") aufweisen und unsichere Muster vermeiden. Diese Muster unterscheiden sich von einer Programmiersprache zur anderen; es gibt keinen Satz „Goldener Regeln", deren Befolgung die Sicherheit der Software garantiert. Einige verbreitet eingesetzte Programmierstandards tragen dem Safety-Aspekt Rechnung:

- Für die C-Sprache: MISRA C und SEI CERT C Codierstandard (*2)
- Für die Sprache C++: MISRA C++, JSF AV C++ Codierstandard, SEI CERT C++ Codierstandard, AUTOSAR C++ Codierrichtlinien (*2)

Normalerweise werden die Regeln in den Standards in mehrere Kategorien eingeteilt – das erleichtert die Orientierung, denn die Zahl der Regeln in den einzelnen Standards kann recht umfangreich sein, wie diese Aufstellung zeigt (siehe Tabelle 2):

Von CVE bis zu den CWE Top 25

Im Jahr 1999 begann die MITRE Corporation (eine US-amerikanische, nicht auf Gewinn ausgerichtete Organisation, die von der US-Regierung gesponserte Forschungs- und Entwicklungszentren betreibt) mit der Dokumentation bekannter Software-Sicherheitslücken in der CVE-Liste (Common Vulnerabilities and Exposures). Zunächst bestand die Liste nur aus 321 Einträgen, aber diese wuchsen bis September 2018 auf mehr als 100.000 Einträge an. Die Pflege erfolgt von 92 CVE Numbering Authorities (CNAs /*1). Diese CVE-Liste findet weithin Anwendung, und zahlreiche Organisationen (u.a. NIST, DISA) empfehlen sie.

Mit dem Anwachsen der CVE-Liste wurde es notwendig, ihre Einträge je nach Problemtyp in Gruppen einzuteilen, um die Ursachen der meisten gängigen Probleme besser zu verstehen und geeignete Maßnahmen für die Vermeidung ähnlicher Probleme in der Zukunft zu treffen. Dazu begann die MITRE Corporation mit der Kategorisierung der bekannten Probleme. Hieraus ging das PLOVER-Dokument (Preliminary List of Vulnerability Examples for Researchers) hervor, und durch Zusammenführung dieser Arbeit mit anderen Forschungsarbeiten entstand die CWE-Liste (Common Weakness Enumeration) als eine formelle Aufstellung der verschiedenen Arten von Software-Schwachstellen.

Version 3.1 der CWE-Liste enthält rund 700 Arten von Schwachstellen, die in 250 Kategorien und Views eingeteilt sind. Wegen der hohen Anzahl führt die CWE zusammen mit dem SANS Institute eine Liste der 25 gefährlichsten Softwarefehler, also die am weitesten verbreiteten und kritischsten Fehler, die zu gravierenden Schwachstellen in Software führen können. Auf den ersten Plätzen dieser „Top 25"-Liste finden sich:

1. CWE-89: Unkorrekte Neutralisierung spezieller Elemente, die in einem SQL-Befehl verwendet werden (SQL Injection)
2. CWE-78: Unkorrekte Neutralisierung spezieller Elemente, die in einem Betriebssystem-Befehl verwendet werden (OS Command Injection)
3. CWE-120: Kopieren in den Puffer, ohne die Größe des Eingabewerts zu prüfen (der klassische Pufferüberlauf)

Best Practices für die Herangehensweise

Sobald man an die Absicherung seines Codes geht, kommt es auf die richtige Wahl an. Muss die Software nach einem bestimmten Functional-Safety-Standard zertifiziert werden (z. B. ISO 26262 für Automotive- oder DO-178C für Luftfahrt-Anwendungen), ist die erste Entscheidung bereits gefällt. Ungeachtet dessen muss man aber über den/die Programmierstandard(s) oder die Teilmenge des Standards entscheiden, den die entwickelte Software zwingend einzuhalten hat. Dies kann (muss aber nicht) einer der zuvor erwähnten Standards sein.

Im nächsten Schritt geht es um die Auswahl eines geeigneten statischen Analysetools, denn es ist nicht möglich, die Einhaltung all dieser Regeln auf manuellem Weg zu verifizieren. Auf dem Markt finden sich sowohl quelloffene als auch kommerzielle statische Analysetools. Hier einige Auswahlkriterien:

- Wird der bzw. werden die gewählte(n) Codierstandard(s) von dem Tool ganz oder teilweise unterstützt?
- Wenn der betreffende Funktionssicherheits-Standard die Tool-Qualifikation verlangt, ist festzustellen, ob das Tool zertifiziert ist, und ob es ein Qualification Kit bietet?
- Kann das Tool die Analyse-Reports in der für die Konformitäts-Analyse benötigten Form vorlegen?
- Kann es die Analyse-Reports in einer für die Entwickler leicht lesbaren Form erzeugen?
- Lässt sich das Tool einwandfrei in die verwendeten IDEs, Build- und CI-Systeme integrieren?
- Gestattet es eine selektive Analyse von neuem, geändertem oder bestehendem Code?
- Unterstützt das Tool eine flexible Konfiguration der Analyse?

Regel	Schweregrad	Wahrscheinlichkeit	Kosten für Abhilfe	Priorität	Level
FIO30-C	hoch	Likely (wahrscheinlich)	mittel	P18	L1
FIO32-C	mittel	Unlikely (nicht wahrscheinlich)	mittel	P4	L3
FIO34-C	hoch	Probable (wahrscheinlich)	mittel	P12	L1
FIO37-C	hoch	Probable (wahrscheinlich)	mittel	P12	L1
FIO38-C	gering	Probable (wahrscheinlich)	mittel	P4	L3

Tabelle 3

Literaturverzeichnis

[1] Hicken Arthur, Parasoft, Embedded Cybersecurity Through Secure Coding Standards CWE and CERT, veröffentlicht auf https://alm.parasoft.com/embedded-cybersecurity-through-secure-coding-standards-cwe-and-cert
[2] MITRE Corporation, Common Vulnerabilities and Exposures (CVE), veröffentlicht auf https://cve.mitre.org/
[3] MITRE Corporation, Common Weakness Enumeration (CWE), veröffentlicht auf https://cwe.mitre.org/
[4] IEC, IEC 61508-3:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Software requirements
[5] ISO, ISO 26262-6:2011(en) Road vehicles - Functional safety - Part 6: Product development at the software level
[6] Carnegie Mellon University, SEI CERT C Coding Standard, veröffentlicht auf https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+CODIERSTANDARDCoding+Standard
[7] Carnegie Mellon University, SEI CERT C++ Coding Standard, veröffentlicht auf https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682

- Nutzt das Tool Risk-Scoring-Algorithmen als Hilfestellung beim Priorisieren der gefundenen Defekte?

Die CWE Community betreibt zusätzlich das CWE Compatibility and Effectiveness Program, einen formellen Prüf- und Evaluierungsprozess für ein Produkt oder einen Service. Die Liste mit Produkten und Services, die als „Officially CWE-Compatible“ eingestuft werden, umfasst rund 55 Einträge. Wird das gewählte Tool in dieser Liste geführt, ist sichergestellt, dass es die finale Stufe des formellen CWE Compatibility Program der MITRE-Organisation erreicht hat.

Sind der Programmierstandard und die erforderlichen Tools festgelegt, sollte die weitere Arbeit für die von Grund auf neu gestarteten Softwareprojekte einfach sein: Man muss nur das Tool in das CI-System integrieren und sicherstellen, dass kein Defekt unberücksichtigt bleibt.

Software in der Etnwicklung

In aller Regel aber müssen Programmierstandards auch für Software durchgesetzt werden, die sich bereits in der Entwicklung befindet. Dann kann das blinde Ausführen der Analyse mit der gesamten Codebasis zu Hunderten von Defektmeldungen führen, die in ihrer Gesamtheit schwierig zu handhaben sind. Glücklicherweise stehen mehrere Techniken zur Verfügung, um diesen Vorgang zu vereinfachen. Am besten ist es, sich zunächst auf den neu geschriebenen oder modifizierten Code zu konzentrieren. So lässt sich sicherstellen, dass nach Einrichtung des Programmierstandards zumindest keine neuen Defekte entstehen.

Eine weitere bewährte Vorgehensweise ist das Einordnen der gemeldeten Defekte nach ihrer Wichtigkeit, so dass die gravierendsten Fehler als erste bearbeitet werden. Zum Beispiel gibt es für die CERT C- und CERT C++-Regeln eine Risikobewertung, die eine einfache Priorisierung der gefundenen Defekte gestattet (siehe Tabelle 3).

Auch hilfreich ist eine nach Priorität geordnete Liste der Defekte im Verbund mit der konfigurierbaren Liste der durchzusetzenden Regeln, um sich zunächst auf die Regeln mit dem höchsten Schweregrad zu konzentrieren und die Zahl der Regeln auszuweiten, nachdem die gravierendsten Fehler behoben sind.

Praktische Umsetzung

Am wichtigsten ist es sicherzustellen, dass sich geeignete Maßnahmen des Entwicklungsteams an die Analyse anschließen. Denn auch die besten Berichte der statischen Analyse sind nutzlos, wenn sie von den Entwicklern nicht zum Reparieren des Codes genutzt werden, um Softwareprodukte hervorzubringen, die in Sachen Safety und Security besser sind.

Erläuterungen

(*1) CVE Numbering Authorities (CNAs) sind unterschiedliche Organisationen (u. a. Schwachstellen-Erforscher, Produktanbieter sowie Bug-Bounty-Programme, die gleichsam ein Kopfgeld für die Aufdeckung von Programmfehlern aussetzen) auf der ganzen Welt, die die Berechtigung haben, erkannte Probleme mit CVE-IDs zu versehen.

(*2) MISRA C und MISRA C++ wurden von der MISRA (Motor Industry Software Reliability Association) entwickelt, die AUTOSAR C++ Codierrichtlinien dagegen von der AUTOSAR (AUTomotive Open System ARchitecture) Entwicklungspartnerschaft. Für die Entwicklung des JSF AV C++ Codierstandards zeichnet die Lockheed Martin Corporation verantwortlich. Die SEI CERT C und C++ Codierstandards enthalten allgemeine Regeln, die die Safety, Zuverlässigkeit und Security von Softwaresystemen, die in den Programmiersprachen C und C++ entwickelt wurden, gewährleisten sollen. ◄