Secure Coding als Bindeglied zwischen formalen Ansätzen um Safety in Infrastruktur-kritischen Systemen zu gewährleisten

Wolfgang Köppl, Dr. Gordon Rohrmair

Siemens AG, Corporate Technology
IC CERT
Otto-Hahn-Ring 6
81739 München
w.koeppl@siemens.com
gordon.rohrmair@siemens.com

Abstract: Safety und IT Security sind eng miteinander verknüpft. Safety schützt den Menschen vor Maschinen, während bei der IT Security ein System vor böswilligen Menschen geschützt wird. Der wichtigste Themenkomplex innerhalb der IT Security ist „Secure Coding" – die Kunst, sichere Applikationen zu erstellen. Die steigende Komplexität der Systeme führt dazu, dass vollständige formale Verifikation nicht mehr bezahlbar ist. 90% der erfolgreichen Angriffe auf IT Systeme basieren auf dem Ausnützen bekannter Programmierfehler. [Ko05] [Mi07] Durch Angriffe kann der Kontrollfluss des Gesamtsystems verändert werden. Um Safety wirklich herzustellen müssen die formalen Ansätze oder Modelle richtig auf der Implementierungsebene verankert werden. Secure Coding schließt die entstehenden Lücken und erhöht die Verlässlichkeit von Safety Funktionalität selbst bei absichtlicher Fehlbedienung des Systems wie beispielsweise einem Angriff. Die Autoren zeigen, warum Security und im speziellen Secure Coding in der Praxis eine Voraussetzung für Safety ist. Secure Coding stellt dabei auch die passenden Ergänzungen für aktuelle Programmiertrends zur Verfügung und integriert sich nahtlos in den Entwicklungsprozess.

1 Zusammenhang von Safety und Security

Safety und Security werden im Deutschen meist jeweils als Sicherheit übersetzt. Während Safety das Ziel hat, Schaden an Personen in jeder erdenklichen Situation abzuhalten, behandelt Security den Schutz vor Risiken vorwiegend durch Angreifer. IT Security soll sicherstellen, dass feindliche Aktionen niemals dazu führen, dass der vordefinierte Kontrollfluß verändert wird. Bei Änderung der Zustandsübergänge des Systems können auch Safety Maßnahmen ausgehebelt werden.

Wenn man Safety von einer theoretischen Seite her betrachtet, also das Sicherstellen, dass unter allen Umständen das System das Verhalten offeriert, das spezifiziert wurde – und nur das, dann kann Security als Unterpunkt von Safety gesehen werden.

In den vorigen Jahrzehnten wurde Safety vor allem dadurch erzielt, Systemsicherungen zu inkludieren und diese auf abstrakter Ebene zu verifizieren. Die Sicherungen sollten garantieren, dass das Gesamtsystem - unabhängig von dem Zustand in dem es sich befindet – menschliches Leib und Wohl nicht in Gefahr bringt.

Unabsichtliche Fehlbedienungen sind dabei die Hauptbedrohung. Heutzutage spielt aber auch IT Security eine zunehmend wichtigere Rolle für Safety, da die Angriffsoberfläche wächst. Personen, die Schaden verursachen wollen, haben dazu eine Vielzahl an Möglichkeiten. Die drei Hauptgründe für diese Beobachtung sind:

- Die zunehmende Vernetzung von kritischen Systemen
- Die steigende Komplexität
- Das immer weniger benötigte Know-How um gefährliche Angriffe auszuführen.

Medizinische Geräte wie Computer-Tomographen sind an Krankenhaus-Netzwerke angebunden. Jeder andere Rechner in diesem Netzwerk kann potentiell als Werkzeug für den Angriff verwendet werden. Dies sind in vielen Fällen einige hundert bis tausende Systeme.

Aufgrund des steigenden Kostendrucks wird auch bei kritischen Systemen auf Drittkomponenten zurückgegriffen. Komplexe Steuerungssysteme in der Energieversorgung integrieren meist einige Dutzend Fremd-Software-Teile und setzten fast immer auf Standard-Betriebssystemen auf. Bezogen auf das Produktportfolio der Siemens AG bedeutet das, dass Softwareprodukte bis zu 70 Komponenten von Drittherstellen aufweisen. Wenn eine Schwachstelle für einen dieser Komponenten bekannt wird, ist dadurch meist das gesamte Produkt angreifbar.

Ein Angreifer benötigt nur die Versionsnummer einer Komponente des Angriffsziels und schon kann er über öffentlich zugängige Quellen Informationen über Schwachstellen und sogar Angriffsprogramme herunterladen. Das benötigte Know-How wird dabei auf einen einfachen „Doppelklick“ reduziert. IT Security ist aus diesen Gründen in vielen Fällen zu einer Vorraussetzung geworden, um Safety erreichen zu können.

2 Safety ohne Security scheitert in der Praxis

Wenn IT Security nicht ernst genommen wird, kann dies zu beträchtlichen Bedrohungen für Menschen führen. Einige auftretende Vorfälle werden öffentlich bekannt, die meisten aber verschwiegen.

- Im Januar 2008 schaffte es ein polnischer Teenager, eine Fernseh-Fernbedienung so zu manipulieren, dass er S-Bahn steuern konnte. Vier S-Bahnen in Lodz entgleisten und mehrere Personen wurden dabei verletzt [He08].
- Angestellte eines deutschen Krankenhauses berichteten dem Siemens CERT im Rahmen eines Projektes von einem andernen Vorfall. Eine kritische Operation musste abgebrochen werden, da die nötigen Informationen des Patienten und die Ergebnisse der Voruntersuchungen nicht verfügbar waren. Ein Wurmbefall hatte die

medizinischen Geräte außer Betrieb gesetzt. Der Patient war bereits narkotisiert und auf dem Operationstisch.

Derartige Vorfälle können in den letzten Jahren von Siemens CERT zunehmend beobachtet werden. Es wird dabei immer offensichtlicher, dass Safety ohne Security in der Praxis bei komplexen Systemen meist scheitert.

3 Security ergänzt modellbasierte und formale Ansätze

In der Ausschreibung zu diesem Workshop wurden verschiedene Trends im Software-Engineering aufgezeigt und deren Relevanz und Anwendbarkeit hinterfragt. Jeder dieser Ansätze hat substantielle Schwächen, die durch IT Security Maßnahmen ausgebessert werden können.

3.1 Modellbasierte Entwicklung

Modellbasierte Entwicklung ist sowohl für funktionale Anforderungen als auch für Sicherheitsanforderungen wie Zugangskontrolle eine gute Hilfe. Allerdings kann ein Modell keine spezifischen Probleme einer Zielplatform behandeln. Dies reicht nur aus, wenn beweisbare Sprachen wie ein Subset von Ada [Gu90] und keine Fremdkomponenten verwendet werden. Für alle anderen Anwendungsfälle, die die Mehrheit darstellen, ist Security Wissen über die Zielplatform essentiell.

90 % der erfolgreichen Angriffe basieren auf dem Ausnutzen von bekannten Implementierungsfehlern [Ko05]. Da diese Fehler ihren Ursprung nicht im Design finden, können sie auch nicht über das Entwickeln von korrekten funktionalen Modellen vermieden werden. Für eine beträchtliche Anzahl von Schwachstellen kann allerdings ein IT Security Konzept modelliert und erstellt werden, das bei Auftreten von Softwarefehlern deren Einfluss auf das Gesamtsystem vermindert (z.B. Buffer Overflow Protections [Mi06] [HK01].

3.2 Formale Methoden

Viele kritische Infrastrukturen sind sehr komplex. Wodurch die formale Verifikation meist zu kostenintensiv wird. Daher werden zum einen nur Teile formal verifiziert und nicht das gesamte System. Zum anderen wird meist nicht auf Implementierungsebene verifiziert sondern eine Abstraktion des Source Codes als Beschreibung für das formale Modell herangezogen.

Die Erfahrungswerte der IT Security, welche Angriffe bei bestimmten Umgebungen wahrscheinlich sind, stellen somit eine essentielle Ergänzung dar. Sie geben einerseits Auskunft darüber, welche Komponenten mit großer Wahrscheinlichkeit angegriffen werden, andererseits geben sie eine Einschätzung über den erreichbaren Sicherheitslevel in einem bestimmten Abstraktionsniveau. Somit ermöglichen sie auch für nicht formal verifizierte Teile einen adäquaten Security und Safety Level zu erreichen.

3.3 Testgenerierung

Wenn Tests nur automatisiert aufgrund der Funktionalitätsbeschreibung eines Produktes erstellt werden, werden meist mögliche Hacker-Angriffe übersehen. Schwachstellen der Zieltechnologie werden dann nicht abgedeckt. Bei C/C++ sind Tests auf Buffer Overflows beispielsweise extrem wichtig. Bei Java wäre diese Bedrohung nicht relevant. Diese Bedrohung würde daher in einer abstrakten und Technologie-unabhängigen Funktionsbeschreibung nicht erwähnt werden. Hier haben nur Erfahrungen aus der IT Security die Chance, Angriffspunkte auf Safety zu finden.

3.4 Codegenerierung

Direkte automatisierte Code-Generierung, beispielsweise aus einem Modell, ist eine vielversprechende Technologie. Viele Schwachstellen, die Programmierer potentiell erzeugen, können so vermieden werden. Dies gilt vor allem in funktioneller Hinsicht.

IT Security betrachtet vor allem aber auch Schwachstellen, die sehr spezifisch für die Zielumgebung sind. Somit sollte Secure Coding Wissen in die Codegenerierungs-Funktionalität einfließen, damit sicher gestellt wird, dass sicherer Code generiert wird. Beispielsweise kann das Template für Datenbank-Abfragen so erstellt werden, dass SQL Injections nicht mehr möglich sind.

Generatoren erstellen meist nicht 100% des Source Codes. Der Programmierer muss noch eingreifen oder neuen Code ganz hinzufügen. Es ist essentiell, dass auch er keine Schwachstellen verursacht. Wichtig dabei ist, festzulegen, ab wann diese Ansätze im Produktlebenszyklus greifen müssen.

4 Integration von IT Security in den Entwicklungsprozess

Der beste Weg, um adäquate IT Security bei besonders kritischen Applikationen zu garantieren, ist die Integration von Maßnahmen in den Entwicklungsprozess. Abbildung 1 zeigt den Good Practice Ansatz, der aufgrund der Erfahrungen des Siemens CERT kreiert wurde und sich am Siemens Referenzprozesshaus orientiert.

Es ist wichtig, bereits in den Plan und Define Phasen umfassenden Fokus auf IT Security Maßnahmen zu legen. Eine Schwachstelle im Design auszumerzen kostet nur etwa ein Achzigstel von dem, was die Beseitigung im Betrieb kostet [Ib03].

4.1 Plan

Die ersten wichtigen Schritte sind Gesetze und Standards zu sichten und eine für IT Security verantwortliche Person zu benennen. Regulatorien zu beachten ist nicht nur für Safety essentiell, sondern spielt auch für IT Security eine zunehmend wichtige Rolle. In den USA definieren beispielsweise die NERC Standards [Ne08] umfassende IT Security Vorgaben für Energieerzeuger und -verteiler.

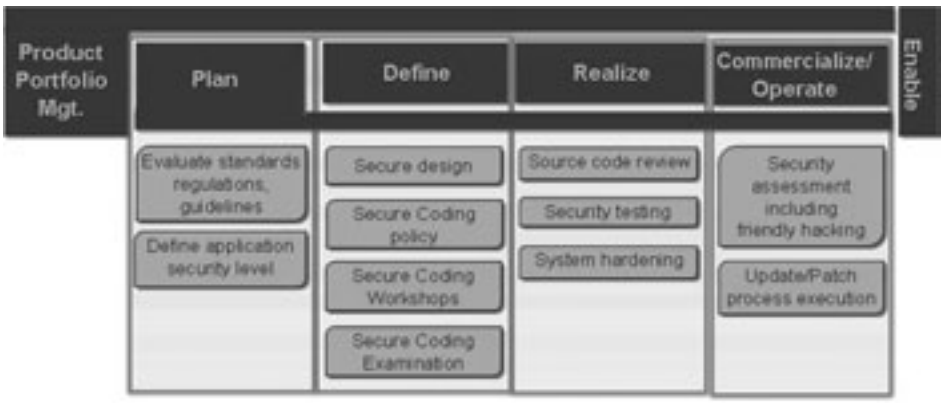

Abbildung 1: Integration von IT Security in den Entwicklungszyklus

4.2 Define

Die Define Phase kann als die wichtigste für IT Security gesehen werden. Nicht nur ein sicheres Design ist essentiell, sondern vor allem müssen in dieser Phase die Vorraussetzungen geschaffen werden, dass die Entwickler bei der Umsetzung die Schwachstellen von vorneherein vermeiden können. Einerseits sollen Programmiervorgaben (Secure Coding Policy) definiert werden, andererseits sollten diese aber auch durch Workshops für Entwickler ergänzt werden. Nur wenn die Programmierer die Fallen verstehen, können sie diese auch vermeiden. Bei kritischen Applikationen sollte man dieses Wissen auch durch Prüfungen verifizieren (Secure Coding Examination).

4.3 Realize

Bei der Implementierung soll die Einhaltung der Secure Coding Policy verifiziert werden. Dies geschieht einerseits im Rahmen des Source Code Reviews und andererseits im Rahmen des Tests der Applikation. Da meist eine Vielzahl von Fremdapplikationen eingesetzt werden, muss sichergestellt werden, dass diese richtig installiert, konfiguriert und aktualisiert werden (System hardening).

4.4 Commercialize/Operate

Bevor das Produkt an den Kunden geliefert oder beim Kunden in Betrieb genommen wird, sollte noch eine Sicherheitsüberprüfung aus Angreifer-Sicht gemacht werden. Oft tauchen Probleme erst dann auf, wenn Produkte in das Zielumfeld integriert werden. Im Gegensatz zu formal verifizierbaren Teilen einer Anwendung, kann man keine 100%-ige IT Security für das Gesamtsystem erreichen. Daher ist es auch wichtig, eine technische und prozessmäßige Möglichkeit zu schaffen, die Applikation zu aktualisieren, wenn Probleme bekannt werden. Dies ist besonders auch für Drittkomponenten relevant.

5 Zusammenfassung

Formale Ansätze sind lebensnotwendig für kritische Infrastrukturen. Allerdings sind diese Methoden in der Praxis nicht immer nahtlos ineinander integrierbar und anwendbar. Die entstehenden Lücken werden durch IT Security, im speziellen durch Secure Coding, geschlossen. Secure Coding Know-how liefert den formalen Ansätzen Erfahrungswerte darüber, welche Komponenten in welchem Abstraktionsgrad untersucht werden müssen. Technologie-spezifische Herausforderungen können durch formale Ansätze meist nicht ausreichend beantwortet werden. Secure Coding sorgt als Ansatz auf Implementierungsebene dafür, dass das Design sicher umgesetzt wird.

Literaturverzeichnis

[Gu90] Guaspari, D. et al, Formal verification of Ada programs, 1990

[He08] Heise Security Artikel, URL: http://www.heise.de/security/news/meldung/101674, [Stand 2008-01-25]

[HK01] Eto Hiroaki, Yoda Kunikazu, ProPolice: Improved stack-smashing attack detection, IEIC Technical Report, 2001

[Ib03] IBM Systems Sciences Institute, Security Defect Cost Multiplier, 2003

[Ko05] Eggün Kocatürk, Patch-Management vermeidet Flickwerk, Computerwoche, 2005

[Mi06] A detailed description of the Data Execution Prevention (DEP) feature in Windows XP Service Pack 2, Windows XP Tablet PC Edition 2005, and Windows Server 2003, URL: http://support.microsoft.com/kb/875352 [Stand 2008-01-25]

[Mi07] MITRE, Common Vulnerabilities and Exposures, http://cve.mitre.org/, 2007

[Ne08] NERC, URL: http://www.nerc.com/, 2008