

## Flickwerk

Gunter Bitz, Dirk Fox

Vor gut sieben Jahren erheiterte die angeblich von Bill Gates provozierte Pressemitteilung von General Motors, die in 13 Punkten beschrieb, was wäre, wenn Autos mit Microsoft-Technologie gebaut würden. Zwei Kostproben: „Wenn man bestimmte Manöver ausführt, z. B. eine Linkskurve, stellt sich der Motor ab und weigert sich, neu zu starten. Man muss ihn dann neu installieren.“ Oder: „Das Airbag-System würde bei jedem Unfall fragen: ‚Sind Sie sicher?‘ bevor es auslöst.“

Tatsächlich: Zwischen der Qualität eines Autos und der von Software lagen Welten. Ein Auto, das immer wieder die Räder verliert? Oder ein Motor, der alle 100 km nur noch mit halber Leistung läuft? Undenkbar? Warum aber war und ist Software nicht so fehlerarm wie ein Auto?

Der Grund ist einfach: Die präzise Produktion von Achsen, Kolben und Getrieben hat im Maschinenbau eine weit über 100jährige Tradition. Anders als die Softwaretechnik – sie existiert noch keine 20 Jahre. Denn allen theoretischen Ansätzen zum Trotz gebiert noch immer Erfahrung den größten Fortschritt. So versagten Fahrzeuge vor 80 Jahren – bei niedrigerer Geschwindigkeit und Belastung – weit häufiger als heute.

Vermutlich wäre GM dennoch heute zurückhaltender mit ihrer Replik. Denn in modernen Fahrzeugen werkeln inzwischen bis zu 50 vernetzte Kleinstrechner, um Antrieb, Beleuchtung, Sicherheitseinrichtungen und Steuerung zu koordinieren und an die Umgebungsbedingungen anzupassen. Der Preis: 2004 war die Fahrzeugelektronik mit 36% Pannensache Nr. 1.

Wir werden auch weiterhin damit leben müssen, dass Software Fehler enthält – im Schnitt 20 je 1.000 Programmzeilen, darunter auch Sicherheitslücken. Zudem setzen wir Software heute unter weitaus „ungünstigeren“ Einsatzbedingungen ein als noch vor 10 Jahren. Statt geschlossener Systeme, die mit einem simplen Terminal zu bedienen sind, nutzen wir heute Web-Services<sup>1</sup>

über das Internet. Das ist eine neue Herausforderung, denn Angriffssoftware erlaubt Attacken mit völlig anderer Datenqualität und -menge als ein Mensch am Terminal. Ist das Entwickeln immer klar?

Damit bleibt bis auf Weiteres das „Flicken“ (Patches) eine wichtige Security-Disziplin. Allerdings unter verschärften Randbedingungen: Exploits sind inzwischen im Schnitt nach fünf Tagen verfügbar. Eine frische Windows 2000 (SP4) Installation „überlebt“ im Internet gerade noch 30 Sekunden. Aber schon mittelfristig ist Patchen keine geeignete Methode mehr, Sicherheitslücken in Software zu bekämpfen. Will man dem „Strudel“ des Wettlaufs zwischen „Lückenfindern“ und der Fehlerbehebung durch Hersteller entkommen, hilft nur, das Übel an der Wurzel zu packen – der Qualität der Softwareentwicklung. Zu oft noch steht die Sicherheit von Programmcode hinter Entwicklungsgeschwindigkeit („time to market“) und Funktionalität („Featuritis“) zurück. Das kann angesichts der weiter wachsenden Abhängigkeit von immer mehr betrieblichen und infrastrukturellen Prozessen und Leistungen zu erheblichen Schäden und, in der Folge, entsprechenden Schadensersatzforderungen führen.

Deswegen rufen nun auch legislative Organe immer lauter nach einer Regulierung der Softwareentwicklung. Hierzulande ist die EU vermittelnd tätig<sup>2</sup> – natürlich mit dem Ziel, die Verbraucher vor größeren Schäden durch mangelhafte Software zu schützen. Damit ist das Eigeninteresse von Softwarefirmen offensichtlich: Wollen sie dem Haftungsrisiko und einer ebenfalls nicht auszuschließenden Regulierung begegnen, die sie empfindlich treffen könnte, müssen sie präventiv handeln, statt wie bisher nur Nachzubessern.<sup>3</sup> Man fürchtet, dass eine Regulierung die Softwareindustrie unflexibler werden lässt, was nicht im Interesse ihrer Kunden wäre.

Das ist leichter gesagt als getan, denn die Komplexität von Software wächst unge-

bremst: Kam Windows 3.1 (1990) mit 2,5 Mio. Programmzeilen aus, benötigte XP (2002) schon 40 Mio. – vier Mal so viel wie die Software des Space Shuttle. Zugleich steigt die Programmvialität und somit die Gesamtzahl sicherheitskritischer Fehler. Deren schiere Menge relativiert bisher jede Qualitätsverbesserung im Entwicklungsprozess. Das zeigt nicht zuletzt der exponentielle Anstieg der gefundenen Bugs: Von 171, die das CERT/CC 1995 zählte, auf knapp 4.000 allein im ersten Halbjahr 2006. Die Frage der Testbarkeit solcher Codemengen ist heute bei Weitem noch nicht gelöst.<sup>4</sup> Es gibt Ansätze automatisierter Testverfahren, deren Qualität jedoch noch stark verbesserungswürdig ist.<sup>5</sup>

Immerhin werden zunehmend Sicherheitsmechanismen in Softwareprodukte integriert. Das ist grundsätzlich eine begrüßenswerte Entwicklung. Problematisch sind jedoch die handwerklichen und konzeptionellen Fehler, die sich dabei immer wieder beobachten lassen. Das beginnt mit Offensichtlichem wie der Klartextübermittlung von Passwörtern beim Login, geht weiter mit fehlerhaften Authentisierungsprotokollen bis zur Ableitung eines 256-Bit-Schlüssels aus einer sechsstelligen numerischen PIN (Schlüsselraum: 20 Bit).

Diese Fehler zeugen von mangelhaftem Verständnis der, zugegeben oft komplexen, relevanten Zusammenhänge – und wiegen den Anwender in falscher Sicherheit. Damit gewinnt „Security Engineering“ an Bedeutung, erstmals systematisch behandelt von Ross Anderson in seinem einschlägigen Kompendium. Jüngst trugen SAP und Microsoft dieser Entwicklung mit internen Maßnahmen<sup>6</sup> sowie der Veröffentlichung einer Fibel „Sicheres Programmieren“ Rechnung – vielleicht das wichtigste Ergebnis der Initiative „Deutschland sicher im Netz“.<sup>7</sup>

<sup>4</sup> Siehe Schreiber und Borrmann, in diesem Heft.

<sup>5</sup> Siehe Schumacher und Wiegenstein, in diesem Heft.

<sup>6</sup> Siehe Zahedani und Obert, in diesem Heft.

<sup>7</sup> Siehe Schimmer, in diesem Heft.

<sup>1</sup> Siehe Thompson, in diesem Heft.

<sup>2</sup> Siehe Paulus und Tegge, in diesem Heft.

<sup>3</sup> Siehe AppSIC, in diesem Heft.