**Institut für Theoretische Informatik**
Awendungsorientierte Formale Verifikation
Prof. Dr. B. Beckert

# Themen Proseminar Sommersemester 2015

*"Desaster in der Softwareentwicklung"*

### 1) Wann sind Programme wirklich sicher? (DB)

Informationssicherheit bedeutet, dass Computersysteme keine geheimen Informationen veröffentlichen dürfen. Aber was bedeutet das formal? Insbesondere sobald wir nebenläufige Programme betrachten gehen die Definitionen im Detail auseinander. Können etwa parallel laufende Programme die Sicherheit beeinträchtigen? In dieser Arbeit sollen mehrere gebräuchliche Definitionen untersucht werden und Minimalbeispiele für Programme gefunden werden, die mal als sicher, mal als unsicher gelten. Daraus lassen sich Angriffsschemata für die beschriebenen Systeme generieren. Die angebene Literatur ist zwar umfangreich, die gesuchten Definitionen machen allerdings nur einen kleinen Teil des Umfangs aus und sollten intuitiv verständlich sein.

### 2) Der Heartbleed-Bug (TB)

Der Schutz sensibler Benutzerdaten ist eine wichtige Aufgabe sicherer Software. Trotz üblicher Qualitätssicherungsmassnahmen in der Softwareentwicklung gelingt es Angreifern in der Praxis aber, etwa durch Ausnutzen von Designfehlern in Protokollen oder Implementierungsfehlern, an Daten zu gelangen, die nicht für sie bestimmt sind. Ein prominentes Beispiel für einen solchen Implementierungsfehler ist der Heartbleed Bug, der es einem Angreifer erlaubt, sensible Daten im durch die OpenSSL-Bibliothek genutzten Speicher auszulesen.

Ziel des Vortrags ist es eine Übersicht (formaler) Methoden zu geben, die sich auf das Heartbleed-Problem anwenden lassen, d.h. mit deren Hilfe unerwünschte Informationsflüsse aufgedeckt oder verhindert werden können.

### 3) Android's, Java's and Python's Sorting Algorithm is broken (MU)

Tim Peters developed the Timsort hybrid sorting algorithm in 2002. It is a clever combination of ideas from merge sort and insertion sort, and designed to perform well on real world data. TimSort was first developed for Python, but later ported to Java (where it appears as

java.util.Collections.sort and java.util.Arrays.sort). TimSort is today used as the default sorting algorithm for Android SDK, Sun's JDK and OpenJDK. Given the popularity of these platforms this means that the number of computers, cloud services and mobile phones that use TimSort for sorting is well into the billions.
Fast forward to 2015. TimSort seemed to fit the bill for a verification challenge as it is rather complex and widely used. Unfortunately, the authors weren't able to prove its correctness. A closer analysis showed that this was, quite simply, because TimSort was broken and theoretical considerations finally led to a path towards finding the bug (interestingly, that bug appears already in the Python implementation).

### 4) Desaster der Datum- und Uhrzeit-Verarbeitung (VK)

Eine korrekte Verarbeitung von Datum und Uhrzeit ist unabdingbar für viele Softwareanwendungen. Das bekannteste Problem in diesem Bereich ist wahrscheinlich Y2K (das Jahr-2000-Problem) - es gibt aber auch viele andere Beispiele (Microsoft Azure, Versagen der EC-Karten 2010, Eurofighter, usw.). Das Ziel des Proseminars ist, anhand von Literatur und Fallberichten ein Überblick über die Problematik und die Gegenmaßnahmen zu erarbeiten und vorzustellen.
(Das ist ein Thema, das am Besten mit Interesse an Mathematik und selbständiger Recherche bearbeitet werden kann.)

### 5) Genau Dein Typ - fehlerfreiere Programme durch Typsysteme (MU)

Typsysteme helfen, kleine Desaster zu vermeiden, bevor sie entstehen.
Statische Typsicherheit ist ein Merkmal vieler moderner Programmiersprachen. Es gibt Studien, die sagen, dass dadurch bestimmte Fehlerquellen bereits durch den Übersetzer zur Entwurfszeit ausgeschlossen werden können; dass Typsysteme die Fehlerrate in Programmem minimieren.
In dieser Arbeit sollen einige neuartige und weitreichende Typsysteme vorgestellt werden und wie diese Typsysteme helfen können, Programmierfehler frühzeitig zu vermeiden.

### 6) Smart Card Vulnerabilities - an existing threat (AW)

RFID Smartcards halten immer mehr Einzug in jedermanns Leben: Personalausweise, Reisepässe, Kreditkarten, Nahverkehrskarten, Zugangsberechtigungen, Mensakarten, …
Dennoch sind diese Karten, die häufig mit Zahlungsverkehr zu tun haben, erstaunlich leicht zu korrumpieren. Wissenschaftler haben schon vor Jahren gezeigt, dass sich Karten kopieren und fälschen lassen. Und erst in den letzten Woche wurde erneut von einem erfolgreichen Angriff auf Smartcards berichtet. Sind Smartcards tickende Zeitbomben oder ist alles gar nicht so schlimm?
Die Aufgabe beinhaltet eine kleine Literaturrecherche zum Thema.

### 7) Security Testing for GSM (Fuzzing) (TB)

Mobilfunknetze basierend auf dem in den 90er Jahren eingeführten GSM-Standard sind weltweit im Einsatz -- auch nach Einführung neuer Standards in den nachfolgenden Mobilfunkgenerationen wie UMTS und LTE.

Bekannt gewordene Schwachstellen in den verwendeten GSM-Protokollen, mit der Auswirkung, dass Telefonate abgehört und Mobilfunkteilnehmer imitiert werden können, sind (immernoch aktuelle) Sicherheitsprobleme. Weiterhin existieren, wie bei vielen komplexen Softwaresystemen, aber auch Fehler in der Implementierung der an der Kommunikation beteiligten Komponenten, die sich teilweise etwa dazu ausnutzen lassen, beliebige Programme auf diesen Komponenten (Handys, Router (?), ...) auszuführen.

Ziel des Seminarvortrags ist es, einen kurzen Einblick in die Sicherheitsdefizite von Mobilfunknetzen zu geben, um dann das Verfahren des sogenannten "fuzz testing" als eine Möglichkeit vorzustellen, solche Implementierungsfehler zu entdecken.

### 8) Das THERAC-Desaster (AW)

Bei dem Therac-25 handelt es sich um ein medizinisches Bestrahlungsgerät zur Behandlung von Tumoren im menschlichen Körper. Zwischen Juni 1985 und Januar 1987 kam es zu diversen Zwischenfällen in den USA und Kanada, bei denen insgesamt sechs Patienten bei der Behandlung mit dem Therac-Bestrahlungsgerät massiv überbestrahlt wurden.

Die Seminaristin oder der Seminarist soll das Desaster analysieren und wissenschaftliche Ausarbeitung zusammentragen, die sich mit dem Vermeiden des Desaster mittels Test oder Verifikationtechniken auseinandersetzen.

### 9) Entwicklung sicherer Systeme mit modellgetriebener Entwicklung (SG)

Um zu vermeiden, dass Systeme angreifbar sind sollte bereits zu einem frühen Zeitpunkt der Entwicklung Sicherheit systematisch betrachtet werden. Um den Mehraufwand in Grenzen zu halten empfiehlt es sich, die Sicherheitsbetrachtung in den Systementwurf zu integrieren.
Eine beliebte Sprache zur modellgetriebenen Systementwicklung bietet UML, eine Erweiterung der UML zur Betrachtung von Sicherheitseigenschaften ist UMLsec [1].
Ziel dieses Seminarvortrages ist es einen Überblick über UMLsec zu liefern, insbesondere mit Fokus auf die formalen Sicherheitsgarantien sowie die Arten von Fehlern, die durch UMLsec vermieden oder offenbart werden können.

### 10) Patriot Rocket Disaster (MH)

Am 25. Februar 1991 versagte das Patriot-Raketenabwehrsystem, das während des ersten Golfkrieges in Dhahran, Saudi Arabia stationiert war. Eine ankommende irakische Scud-Raketen wurde vom System nicht erkannt und nicht abfangen. Diese Rakete traf daraufhin eine Baracke der US-Army und töte 28 Menschen.

Schuld an diesem Fehlverhalten war ein Softwarebug: Sein Grund ist ein Problem beim Speichern von Zeit in einem 24-bit Register. Sie wurde als Fixed-Point-Zahl mit einer Akkuranz von 1/10s gespeichert. Die Präzision des Registers reichte jedoch nicht aus, um die Zeit so zu speichern und der Fehler konnte sich über die Zeit akkumulieren, bis die Abweichung nach einigen Stunden schließlich zu groß war und das System nicht mehr funktionierte.
In diesem Seminar soll die Literatur zu diesem Desaster recherchiert werden und über den konkreten Vorfall hinaus, Bugs mit Dezimalberechnungen beleuchtet werden und ggf. die Möglichkeiten, diese im Entwurf zu vermeiden.

### 11) Verification versus Validation/Mode Confusion(SaG)

Verifikation und Validierung sind zwei komplementäre Begriffe in der Softwarequalitätssicherung. Verifikation bedeutet, sicherzustellen, dass Systeme sich so verhalten wie sie spezifiziert sind. Validierung bedeutet, sicherzustellen, dass die Spezifizierung der Intention der Benutzergruppe entspricht.
In diesem Seminarbeitrag soll an Beispielen aus der Luftfahrt gezeigt werden, wie fatal es sein kann, dass Software korrekt funktioniert, aber ihre Spezifikation nicht der Erwartung des Piloten entspricht.

### 12) Desastervermeidung durch Prävention im User-Interface(SaG)

Das User-Interface ist die Kopplung zwischen den technischen Abläufen in Hard- und Software und dem Benutzer, sei es der Chirurgin während der Operation oder Pilotin im Flugzeug, die komplexe Systemabläufe abwickeln müssen. Während der Benutzung von gefährlichen Maschinen unterlaufen dem Menschen Fehler, die lebensbedrohlich sein können.
Gut designte Schnittstellen können somit helfen Desaster zu verhindern.
Zum einen gibt es Anforderungen an Benutzerschnittstellen, die sich aus der ISO-Norm 9241ergeben, zum Anderen gibt es Versuche Benutzerschnittstellen zu verifizieren .