Antrag auf Einrichtung eines neuen DFG-Schwerpunktprogramms

Zuverlässig sichere Softwaresysteme

(gekürzte Version)

Initiatorengruppe:

Dr. Dieter Hutter, DFKI GmbH Bremen
Prof. Dr. Heiko Mantel (**Koordinator**), TU Darmstadt
Prof. Dr. Günter Müller, Universität Freiburg
Prof. Tobias Nipkow, Ph.D., TU München
Prof. Dr. Gregor Snelting, Universität Karlsruhe (TH)

15. November 2008

Inhaltsverzeichnis

1 Zusammenfassung

Das bisherige Bild der IT-Sicherheit (im Sinne von IT-Security) wird vor allem durch klare Grenzen zwischen einer vertrauenswürdigen internen und einer potentiell feindlichen äußeren Welt geprägt. Dementsprechend konzentrieren sich viele klassische Sicherheitsmechanismen auf den Schutz von Grenzen zwischen diesen Welten und begründen das Vertrauen in Artefakte in erster Linie anhand ihrer Identität und Herkunft. Jedoch führen Aspekte wie Vernetzung, Mobilität und dynamische Erweiterbarkeit dazu, dass Grenzen zwischen “Innen” und “Außen” zunehmend verschwimmen, also einer differenzierteren Sicht bedürfen. In heutigen Systemen werden Aufgaben zunehmend durch flexible, dynamische Kompositionen von mehreren Diensten und deren Interaktion in verteilten Systemen bearbeitet. Hieraus resultiert eine Komplexität, die in der Praxis sowohl das Verständnis als auch die zuverlässige Überprüfung von Sicherheitsgarantien auf einer technischen Ebene verhindert.

Das *SPP Zuverlässig sichere Softwaresysteme* zielt daher auf die Entwicklung eines neuen konzeptionellen und technischen Rahmens für Sicherheit, der die Zertifizierung von Sicherheitsgarantien auf ein semantisch fundiertes Verständnis von Programmen und Sicherheitsaspekten abstützt. Das *erste Leitthema* ist die Entwicklung präzise definierter (und daher überprüfbarer) Sicherheitseigenschaften. Hierdurch soll eine differenzierte Sicht auf Sicherheit möglich werden, die einerseits von technischen Implementierungsdetails abstrahiert und andererseits die adäquate und präzise Modellierung der vielfältigen Sicherheitsanforderungen und -garantien erlaubt. Das *zweite Leitthema* ist die Entwicklung von Analysemethoden und -werkzeugen zur zuverlässigen Überprüfung der Einhaltung von Sicherheitseigenschaften. Hierdurch wird die Basis für eine semantisch fundierte (und daher zuverlässige) Zertifizierung von Sicherheitsgarantien für Programme geschaffen. Das *dritte Leitthema* ist die Entwicklung von Konzepten, um Sicherheitsaspekte auch in komplexen Softwaresystemen erfassen und zertifizieren zu können, also für Sicherheit im Großen. Insbesondere sollen die in der Informatik gängigen Techniken der Abstraktion, Dekomposition und schrittweisen Verfeinerung für den Sicherheitsbereich adaptiert werden, wofür einige konzeptionelle und technische, aber sicherheitsspezifische Probleme zu überwinden sind. Hierdurch soll es beispielsweise möglich werden, abstrakte Sicherheitsanforderungen, wie sie auf der Ebene von Geschäftsprozessen auftreten (z.B. Need-to-Know oder Separation-of-Duty), aus den eher detaillierten Garantien der verwendeten Sicherheitsmechanismen abzuleiten.

Im SPP wird der Ansatz der *Informationsflusskontrolle* eine zentrale Rolle spielen: Informationsflusseigenschaften abstrahieren von technischen Implementierungsdetails, Informationsflussanalysen berücksichtigen die Programmlogik bis in die letzten Verästelungen und Kompositionalitätsresultate für sicheren Informationsfluss erlauben eine fundierte, modulare Sicht auf komplexe Systeme. Dieser Ansatz erfüllt die essentiellen Voraussetzungen für eine semantisch fundierte Zertifizierung von Sicherheitsaspekten in komplexen Softwaresystemen. Die Entwicklung weiterer, vergleichbar geeigneter Ansätze ist wünschenswert, wobei Informationsflusseigenschaften in Bezug auf Deklarativität und semantische Fundierung sowohl Vorbild als auch Maßstab für neuartige Sicherheitseigenschaften sein sollen.

Bis vor kurzem waren die für dieses interdisziplinäre Vorhaben notwendigen technischen Grundlagen noch nicht gegeben, sie wurden erst durch die jüngsten Fortschritte in den Bereichen Informationsflusskontrolle, Programmanalyse und Verifikation geschaffen. Gerade in Deutschland gibt es eine Reihe von international führenden Wissenschaftlern in diesen Bereichen, die einen Paradigmenwechsel in der IT-Sicherheitsforschung herbeiführen können. Dieses Synergiepotential soll durch das Schwerpunktprogramm freigesetzt werden.

2 Thematik und Vision des Schwerpunktprogramms

Der Sicherheit von Softwaresystemen (im Sinne von IT-Security) kommt in unserer Gesellschaft eine zentrale Rolle zu. Informationen sind zu Wirtschaftsgütern geworden, die zwar elektronisch verarbeitet aber in gleicher Weise wie physikalische Güter vermarktet und gehandelt werden. Die Geheimhaltung persönlicher Daten ist für den Schutz der Privatsphäre essentiell und z.B. bei elektronischen Wahlen genau wie die Integrität von Stimmzetteln und Auszählungsergebnissen sogar notwendig, um den Fortbestand unserer demokratischen Gesellschaftsordnung zu sichern. Der Wunsch nach sicherer Informationsverarbeitung wird auch am Umsatz von Sicherheitssoftware deutlich, der für 2008 bereits auf mehr als 10 Milliarden US-Dollar geschätzt wird [11]. Trotzdem ist die Informatik immer noch weit von einer akzeptablen Lösung der Sicherheitsproblematik entfernt, was durch die Unzahl an Angriffen und Sicherheitslücken eindrucksvoll belegt wird (siehe z.B. [52]).

2.1 Notwendigkeit eines Paradigmenwechsels

Im Unterschied zu anderen Gebieten der Informatik fehlen im Bereich der IT-Sicherheit angemessene Qualitätskriterien und zuverlässige Qualitätsüberprüfungen auf technischer Ebene für komplexe Softwaresysteme. Anstatt Garantien für die sichere Funktionsweise von Programmen zu etablieren, beschränkt sich die Zertifizierung sicherheitskritischer Systeme meist auf methodische Aspekte der Softwareentwicklung (z.B. im Rahmen der Common Criteria [6]). Hierdurch kann die Sicherheit der entwickelten Software zwar positiv beeinflusst, nicht aber zuverlässig garantiert werden. Darüberhinaus wird die Konkretisierung von Sicherheitsanforderungen in Form von Eigenschaften, die z.B. bei funktionalen Anforderungen oder Fehlertoleranzanforderungen selbstverständlich ist, bei der Entwicklung sicherheitskritischer Software meist übersprungen. Stattdessen wird direkt zu Realisierungsdetails wie den eingesetzten Sicherheitsmechanismen übergegangen, wodurch der für eine Sicherheitsüberprüfung unabdingbare Bezugspunkt auf Seiten der Anforderungen fehlt.

Das Fehlen zuverlässiger Sicherheitsgarantien wird in der Praxis momentan durch den intensiven Einsatz von Vertrauensmodellen kaschiert, wobei sicherheitsrelevante Entscheidungen meist ausschließlich anhand der Identität, Integrität und Herkunft von Programmen getroffen werden. Kryptografische Zertifikate garantieren nur, dass der Programmcode nach Auslieferung durch den Hersteller nicht unerlaubt verändert wurde, kommunizieren aber weder das Qualitätsniveau noch den Umfang der Sicherheitsüberprüfung. Sie bieten somit keine angemessene Grundlage für informierte sicherheitsrelevante Entscheidungen und schon gar nicht für allumfassendes Vertrauen in die Sicherheit der zertifizierten Programme, zumal die nutzerspezifischen Sicherheitsbedürfnisse den Herstellern zum Zeitpunkt der Sicherheitsüberprüfungen normalerweise nicht einmal bekannt sind. Darüberhinaus können nutzerspezifische Sicherheitsinteressen sogar im Konflikt mit herstellerseitigen Interessen stehen, wie in jüngster Zeit z.B. bei der Weitergabe persönlicher Daten durch Internet-Browser prominenter Hersteller deutlich wurde (siehe z.B. [16, 17]).

Klassische Sicherheitstechnologie wird durch eine starke Fokussierung auf Sicherheitsmechanismen geprägt. Mechanismen wie Authentifizierungskomponenten, kryptografische Protokolle, Netzwerkfilter, Virenscanner und Zugriffskontrollen haben sich in vielen Bereichen gut bewährt. Diese Sicherheitsmechanismen werden in erster Linie eingesetzt, um eine Zone zu erschaffen, die vor dem Eindringen unbekannter, potentiell bösartiger Programme geschützt ist. Das vorherrschende Paradigma einer vertrauensabhängigen Zugangskontrolle, das Programme vor allem anhand ihrer Herkunft beurteilt, wird an der Metapher einer

mittelalterlichen Burg anschaulich: ein Innenhof wird durch unüberwindbare Mauern von der Außenwelt abgetrennt, und der Zugang ist nur durch wenige Burgtore möglich, die gut kontrolliert werden, um den Zugang auf als vertrauenswürdig bekannte Personen einzuschränken. Während diese Metapher für einfache Systeme einleuchtend sein mag, also für *Sicherheit im Kleinen*, ist sie für komplexe IT-Systeme nur äußerst eingeschränkt nützlich, um die Sicherheitsproblematik vollständig zu erfassen. Einerseits bieten offene Netze und erweiterbare Software eine Vielzahl an Zugangsmöglichkeiten, so dass neben der Kontrolle einzelner Zugänge auch das Zusammenspiel dieser Kontrollen zu koordinieren ist. Andererseits stehen Programmen im "Inneren" vielfältige Aktionen, Interaktionsmöglichkeiten und Ressourcen zur Verfügung, deren Verwendung sicherheitsrelevant sein kann, und daher auch nach einem Zugang zu kontrollieren ist. Sowohl Zugangs- als auch Zugriffskontrollen konfrontieren Nutzer mit einer Fülle an Detailanfragen wie beispielsweise "KLICKEN SIE AUF FORTSETZUNG, FALLS SIE DIESES PROGRAMM STARTEN WOLLEN: SETUP.EXE" oder "WOLLEN SIE DER ANWENDUNG BROWSER DEN LAN-ZUGRIFF ERLAUBEN?", die einfach scheinen (vergleichbar einer Zugangskontrolle am Burgtor), aber in komplexen IT-Systemen keine angemessene Grundlage für informierte Entscheidungen bieten. In beiden Beispielen bleibt unklar, welche Konsequenzen sich aus einer Zustimmung für die Sicherheit im Großen ergeben, z.B. in wie weit die Vertraulichkeit persönlicher Daten gefährdet sein könnte.

Während die Weiterentwicklung von Sicherheitsmechanismen enorme Fortschritte erzielt hat, ist in der letzten Dekade auch immer deutlicher geworden, dass eine zu starke Fokussierung auf Mechanismen, wie sie heute üblich ist, nicht geeignet sein wird, um der Komplexität moderner Softwaresysteme Herr zu werden, also *Sicherheit im Großen* zuverlässig zu erfassen. Dass das bisherige Paradigma der IT-Sicherheit an seine Grenzen gestoßen ist, lässt sich nicht nur analytisch begründen sondern auch empirisch beobachten. Benötigt werden Konzepte, die von technischen Details wie den verwendeten Sicherheitsmechanismen in einer fundierten Weise abstrahieren, um die Komplexität von Software in Sicherheitsanalysen zu beherrschen. Die Adaptierung der in anderen Bereichen üblichen Ansätze zur Reduktion von Komplexität durch Abstraktion, Dekomposition und schrittweise Verfeinerung wäre äußerst wünschenswert. Hierbei treten aber fundamentale, für den Sicherheitsbereich spezifische Probleme auf (wie z.B. das Verfeinerungsparadoxon [30]), die es zu bewältigen gilt.

2.2 Der Ansatz des SPPs

Im SPP sollen die konzeptionellen Grenzen der klassischen Sicherheitstechnologie überwunden und der dringend benötigte Paradigmenwechsel vom heute üblichen vertrauensabhängigen und mechanismenorientierten Zugang zu einem eigenschaftsorientierten und semantisch fundierten Zugang in der Sicherheitsforschung eingeleitet werden. Um die zuverlässige Zertifizierung von Sicherheitsgarantien zu ermöglichen, sollen vor allem in den nachfolgend skizzierten drei Hauptrichtungen neue Wege eröffnet und ausgestaltet werden:

Eigenschaftsorientierter Zugang zu IT-Sicherheit. Eine Konkretisierung von Sicherheitsaspekten durch präzise definierte Eigenschaften wird eine differenziertere Sicht auf die vielfältigen Sicherheitsbedürfnisse ermöglichen als sie heute üblich ist. Hierdurch soll eine fundierte Basis für die Analyse von Sicherheitsanforderungen (z.B. zur Identifikation widersprüchlicher Anforderungen) und für den Nachweis von Sicherheitsgarantien auf technischer Ebene gelegt werden. Während ein entsprechendes Vorgehen in anderen Bereichen der Informatik seit langem üblich ist, steht der Durchbruch im Bereich der IT-Sicherheit noch aus. Dieses ist umso erstaunlicher, da bereits seit vielen Jahren, sogar Jahrzehnten präzise

definierte Sicherheitseigenschaften bekannt sind, die als Grundlage für einen eigenschaftsorientierten Zugang geeignet scheinen. Besonders vielversprechend ist die Familie der sogenannten *Informationsflusseigenschaften*, die Sicherheitsaspekte durch Einschränkungen an den Informationsfluss während der Programmausführung charakterisieren. Mit diesem Ansatz lassen sich sowohl Vertraulichkeits- als auch Integritätsanforderungen ausdrücken, wodurch sogar eine einheitliche Sicht auf unterschiedliche Sicherheitsaspekte möglich wird. Informationsflusseigenschaften bieten sich auch bezüglich der Anforderungen der beiden anderen, nachfolgend beschriebenen Hauptrichtungen an. Bevor diese Lösungsansätze für einen eigenschaftsorientierten Zugang in praxistauglichen Lösungen realisiert werden können, muss aber eine Reihe grundlegender Fragestellungen geklärt und beantwortet werden. Zentrale Forschungsziele in diesem Bereich sind: (1) die Adaptierung von bekannten Sicherheitseigenschaften wie z.B. Noninterference [13] an praktische Erfordernisse, (2) die praktische Erprobung dieser Eigenschaften und (3) die bedarfsgerechte Entwicklung neuer Sicherheitseigenschaften, die sowohl präzise formal definiert als auch praktikabel sind.

Semantisch fundierte Sicherheitsanalyse. Eine fundierte Berücksichtigung der Semantik von Programmen wird zuverlässige Sicherheitsanalysen auf technischer Ebene ermöglichen. Eine Betrachtung einzelner Aktionen etwa der Zugriffsoperationen eines Programms ist hierfür alleine oft nicht ausreichend, sondern es müssen auch vollständige Programmabläufe analysiert werden. Für einige Sicherheitsaspekte ist es sogar notwendig, mehrere mögliche Programmabläufe miteinander zu vergleichen oder die Menge aller möglichen Programmabläufe als Ganzes zu betrachten. Hierin unterscheidet sich Security von anderen Anforderungen wie z.B. funktionaler Korrektheit oder Sicherheit im Sinne von Safety.

Als mögliche Ansätze für fundierte Sicherheitsanalysen kommen z.B. die explizite Berücksichtigung der Programmsemantik im Rahmen einer formalen Verifikation, die implizite Berücksichtigung durch Verwendung einer Programmlogik sowie die Fokussierung auf die für eine gegebene Sicherheitseigenschaft relevanten Aspekte der Semantik durch Verwendung eines spezialisierten Sicherheitstypsystems in Frage. Unabhängig vom gewählten Ansatz muss die Semantik der verwendeten Programmiersprache sorgfältig berücksichtigt werden, darf die Analyse nur logisch gültige Schlüsse ziehen, und sollte die Korrektheit der verwendeten Analysewerkzeuge auf Implementierungsebene sichergestellt werden. Nach einer erfolgreichen Analyse können die hergeleiteten Sicherheitsgarantien in Form von Zertifikaten formuliert werden, die im Unterschied zu kryptografischen Zertifikaten semantisch fundiert sind. Semantische Zertifikate könnten dann sogar als Grundlage von Haftungsvereinbarungen zwischen Herstellern und Nutzern dienen. Unter Ausnutzung des Proof-Carrying-Code Prinzips [39] könnte Nutzern darüberhinaus die Möglichkeit angeboten werden, die Korrektheit der Sicherheitsanalyse mit Hilfe eines Rechners ihrer Wahl nachzuprüfen. Zentrale Forschungsfragestellungen in diesem Bereich sind: (1) die Weiterentwicklung von semantisch fundierten Sicherheitsanalysen, um deren bisher sehr eingeschränkte Effizienz und Präzision signifikant zu verbessern, (2) die Entwicklung von zuverlässigen Werkzeugen zur Automatisierung von Sicherheitsanalysen, (3) die Schaffung von Infrastrukturkomponenten für eine semantisch fundierte Zertifizierung und (4) die Entwicklung von Ansätzen zur Reparatur von Programmen nach einer fehlgeschlagenen Sicherheitsanalyse.

Denkwerkzeuge für Sicherheit im Großen. Die Adaptierung von Techniken zur Abstraktion und Modularisierung für den Sicherheitsbereich wird eine zuverlässige Zertifizierung von Sicherheitsgarantien auch für komplexe Softwaresysteme, also Sicherheit im Großen, ermöglichen. Die Formulierung von Sicherheitsgarantien als Eigenschaften, die deklarativ in

Termini der Schnittstellen von Systemkomponenten definiert sind (wie es z.B. mit Informationsflusseigenschaften möglich ist), bietet eine Abstraktion von technischen Implementierungsdetails innerhalb der Komponenten. Kompositionalitätsresultate bilden die Basis, um Garantien für Komponenten bei der Herleitung von Sicherheitsgarantien für komplexere, aus Komponenten zusammengesetzte Softwaresysteme auszunutzen. Um die Komplexität moderner Softwaresysteme zu beherrschen, ist eine Kombination von Abstraktion und Modularisierung nützlich. Erst hierdurch wird es möglich, beispielsweise aus Garantien für Sicherheitsmechanismen schrittweise zuerst Garantien für komplexere Dienste herzuleiten und dann aus diesen die gewünschten Sicherheitsgarantien für Geschäftsprozesse zu folgern. In einer fundierten Sicherheitsanalyse muss auch mit konzeptionellen Unterschieden zwischen Sicherheitseigenschaften umgegangen werden, insbesondere mit dem Übergang von den für Mechanismen üblichen, eher technikorientierten Sicherheitsgarantien zu den Sicherheitsgarantien für komplexe Softwaresysteme, die oft recht abstrakt sind (z.B. Need-to-Know oder Separation-of-Duty auf der Ebene von Geschäftsprozessen). Die fundamentalen Unterschiede zwischen Sicherheitseigenschaften und herkömmlichen Systemeigenschaften machen die Entwicklung neuer Abstraktionstechniken und Kompositionalitätsresultate notwendig, die jeweils durch die Herleitung zugehöriger Korrektheitsresultate zu fundieren sind.

Zentrale Forschungsfragen in diesem Bereich sind: die Entwicklung von (1) neuen Kompositionalitätsresultaten, um die Komplexität für Sicherheit im Großen zu reduzieren und fundierte Ende-zu-Ende Garantien herzuleiten, (2) Abstraktionen, die Sicherheitsgarantien auch für komplexe Anwendungen subjektiv verständlich werden lassen, (3) Werkzeugunterstützung für modulare Sicherheitsanalysen, die auch die Kombination unterschiedlicher Analysetechniken unterstützen, und (4) Ansätzen zur schrittweisen Softwareentwicklung, die die Einhaltung von Sicherheitsanforderungen bereits per Konstruktion berücksichtigen.

2.3 Ergebnisvision und Chancen eines fokussierten SPPs

Der angestrebte Paradigmenwechsel soll einen fundierten Umgang mit der Sicherheitsproblematik ermöglichen. Durch die im SPP entwickelten Sicherheitsmodelle und -analysen sollen zuverlässige Sicherheitsgarantien auf technischer Ebene möglich werden. Hierdurch soll z.B. auch eine flexiblere Nutzung erweiterbarer Architekturen möglich werden, inklusive der risikofreien Nutzung von Softwarekomponenten unbekannter Hersteller. Sicherheit im Großen soll einerseits technisch möglich und andererseits auch durch geeignete Denkwerkzeuge konzeptionell erfassbar und kommunizierbar werden. Die entwickelten Technologien und Methoden werden sowohl auf die Nutzung als auch auf die Administration und Konstruktion von Softwaresystemen einen substantiellen und nachhaltigen positiven Einfluss haben.

Die ultimative Vision ist ein fundamental besserer Umgang mit der Sicherheitsproblematik in Softwaresystemen, der nicht nur endlich ein angemessenes Sicherheitsniveau bietet, sondern auch eine weitgehende Automatisierung von sicherheitsrelevanten Entscheidungen erlaubt. Hierbei werden die semantisch fundierten Sicherheitszertifikate die Grundlage bilden, um sicherzustellen, dass Programme alle gewünschten Richtlinien einhalten. Sicherheitsrichtlinien werden durch semantisch fundierte Politiken spezifiziert, die hersteller-, nutzer- und anwendungsspezifische Anteile enthalten können, also flexibel zusammensetzbar sind. Durch die semantische Fundierung mit Sicherheitseigenschaften können Sicherheitspolitiken als Bezugspunkt für technische Sicherheitsüberprüfungen von Softwaresystemen dienen und auch zur Identifikation von widersprüchlichen Anforderungen. Deklarativ definierte Sicherheitseigenschaften werden nicht nur fundierte Beschreibungen von Sicher-

heitsgarantien ermöglichen, sondern auch die subjektiv verständliche Kommunikation solcher Garantien unterstützen, so dass Qualitätsunterschiede im Sicherheitsbereich für Nutzer erkennbar und für Hersteller besser vermarktbar werden. Erst hierdurch wird ein wirklich informierter und mündiger Umgang mit der Sicherheitsproblematik möglich.

Mit dem angestrebten Paradigmenwechsel werden also langfristig auch Veränderungen in der Benutzerinteraktion einhergehen. Die Resultate des SPPs werden aber schon in der heute üblichen Interaktion spürbare Verbesserungen erzielen, wie nachfolgend illustriert:

Es wird möglich sein, Nutzer deutlich besser über Sicherheitsaspekte zu informieren, so dass diese Entscheidungen angemessen informiert treffen können. Während kryptografische Zertifikate einem Nutzer heute wenigstens die Möglichkeit bieten, den Hersteller eines Programms zu bestimmen und die Integrität des Programmcodes zweifelsfrei zu überprüfen, muss er bisher aber immer noch blind darauf vertrauen, dass das Programmverhalten zu seinen Sicherheitsanforderungen konform ist. Durch die im SPP entwickelten Sicherheitsmodelle und -analysen wird es endlich möglich werden, Konformität auf technischer Ebene zu überprüfen und somit z.B. auszuschließen, dass private Daten durch ein Programm verbreitet werden. Beispielsweise könnte eine Anfrage wie "Das Programm finanzverwaltung.exe versucht, auf das Internet zuzugreifen (Ziel-IP:199.2.456.0). Möchten Sie dieses zulassen?" optional weitere Informationen anbieten, wie z.B. "Die hierbei versendeten Daten sind unabhängig von Ihren persönlichen Daten in den Verzeichnissen MyDocuments, MyPictures und MyMusic.", "Bei diesem Zugriff werden nur Informationen versendet, die aus Ihrer heutigen Interaktion mit dem Programm entstanden sind." bzw. "Bei diesem Zugriff werden potentiell Informationen über Ihre Konten und Aktiendepots versendet". Dem Nutzer werden also neben den zur Verfügung stehenden Handlungsoptionen auch deren mögliche Konsequenzen mitgeteilt. Die Anzeige der versendeten Daten (eine heute bereits übliche Option) bietet diese Information nicht, da es prinzipiell unmöglich ist, alleine aus den Daten die darin kodierte Information zuverlässig zu erschließen. Hierzu wäre zusätzlich die Kenntnis der Interpretation nötig oder wenigstens die Kenntnis aller Quellen aus denen die Daten generiert wurden. Deshalb sind semantisch fundierte Informationsflussanalysen unabdingbar.

Das Entscheidungsspektrum der Nutzer kann durch neue Optionen sinnvoll erweitert werden. Beispielsweise könnte vor der Installation eines Programms, z.B. eines Browser-Plugins, neben der Auswahl "Wollen Sie das Plugin installieren?" zusätzlich eine Option wie z.B. "Plugin installieren, aber alle Features deaktivieren, die Informationen über die von Ihnen besuchten Webseiten weiterleiten würden." angeboten werden, oder sogar "Wählen Sie alle Quellen aus, deren Daten durch das Plugin weitergegeben werden dürfen. Alle nicht konformen Features des Plugins werden deaktiviert". Es ergeben sich also völlig neue Möglichkeiten, die von der bedarfsgerechten Konfiguration von Programmen bis zur automatischen Reparatur unsicherer Programme reichen werden. Durch die Adäquatheit der Sicherheitseigenschaften und die Korrektheit der Sicherheitsanalysen wird Sicherheit zuverlässig garantiert, so dass sich Nutzer in Zukunft auf die Einhaltung Ihrer Sicherheitsrichtlinien wirklich verlassen können.

Sicherheitsrelevante Entscheidungen werden zuverlässig automatisierbar. Anstatt Entscheidungen in jedem Einzelfall zu treffen, soll die Möglichkeit angeboten werden, die eigenen Präferenzen in Form semantisch fundierter Sicherheitspolitiken zu erfassen. Basierend auf einer vorab definierten Sicherheitspolitik und der Auswahl einer eher optimistischen oder pessimistischen Entscheidungsstrategie (je nach persönlicher Präferenz) könnten Zustimmungen oder Ablehnungen weitgehend automatisch errechnet werden. Die Zuverlässigkeit

der getroffenen Entscheidungen wäre durch die Sicherheitsanalysen garantiert, obwohl nur in Ausnahmefällen Eingaben des Nutzers benötigt würden. Sogar eine inkrementelle Definition der eigenen Sicherheitspolitik wäre denkbar, so dass bei Auftreten eines Sonderfalls die Option angeboten wird, die Sicherheitspolitik so anzupassen, dass diese und ähnliche Entscheidungen in Zukunft automatisch getroffen werden können.

Diese Verbesserungen werden erst durch die Kombination von Ideen, Technologien und Werkzeugen möglich, die unterschiedlichen Teildisziplinen der Informatik zuzuordnen sind. Mit dem SPP soll eine interdisziplinäre Kooperation von Wissenschaftlern aus den Bereichen IT-Sicherheit, Programmanalyse und Verifikation etabliert werden und die hierfür notwendigen Rahmenbedingungen geschaffen werden. Die Entwicklung zuverlässiger Sicherheitsanalysen ist erst durch jüngste Fortschritte in diesen drei Bereichen realistisch geworden (siehe Abschnitt 3). Beispielsweise stehen heute mächtige Verifikationswerkzeuge und gut skalierende Programmanalysetechniken zur Verfügung, die noch vor wenigen Jahren unerreichbar und vielleicht sogar kaum vorstellbar waren. In Deutschland bieten sich ideale Voraussetzungen für dieses interdisziplinäre Forschungsvorhaben [. . .].

3 Stand der Forschung und Herausforderungen

Wir diskutieren den Stand der Forschung und die sich daraus ergebenden Herausforderungen anhand der beteiligten Kerndisziplinen: Sicherheit, Programmanalyse und Verifikation.

3.1 Sicherheit (Security)

Nach einem kurzen Abriss über traditionelle Zugriffskontrolle und Proof-Carrying Code wird Informationsflusskontrolle allgemein und in zwei wichtigen Ausprägungen skizziert.

Zugriffskontrolle. Sicherheitspolitiken beschreiben die Sicherheitsanforderungen an ein System. Zugriffskontrollen (z.B. Mandatory Access Control, Discretionary Access Control oder Role-Based Access Control) bilden die üblichen Schutzmechanismen für vertrauliche Daten, indem Berechtigungen der Subjekte für die einzelnen Daten und den darauf operierenden Operationen definiert werden. Bekannte Sicherheitspolitiksprachen (z.B. [27, 9, 42]) erlauben die Spezifikation solcher Zugriffskontrollen und erweitern sie hinsichtlich Handlungsverpflichtungen (Obligationen) oder Delegation von Rechten. Die Zugriffskontrollen operieren auf den Behältern, in denen die Daten gespeichert werden. Informationen, die aus dem Systemverhalten abgeleitet werden und damit quasi außerhalb der kontrollierten Container auftreten, sind damit nicht geschützt. Klassischerweise werden Zugriffskontrollen durch Referenzmonitore implementiert, die die Ausführung eines Programms überwachen und dabei Aktionen unterbinden, die gegen die Sicherheitspolitik verstoßen. Der Referenzmonitor trifft seine Entscheidungen auf Basis des aktuellen Systemzustands und der Art des Zugriffs. Damit können im Allgemeinen allerdings Sicherheitsanforderungen, die mehr als nur die bei der Überwachung des Systems erhältlichen Informationen benötigen, nicht garantiert werden. Ein Beispiel ist hierfür das Verbot von Informationsflüssen zwischen Systemteilen, da diese von der Menge aller möglichen Systemabläufe abhängen.

Proof-Carrying Code (PCC). Da Referenzmonitore zu erheblichen Effizienzeinbußen führen, wurden Ansätze entwickelt, durch Programmanalyse (etwa mit Typsystemen) und Programmtransformation Sicherheitskontrollen in das Programm selbst zu integrieren, z.B. Überprüfung von Array-Grenzen. Eine Programmanalyse ermittelt diejenigen Programmstellen,

an denen es zur Laufzeit zu einer Verletzung der Sicherheitspolitik kommen könnte und fügt in das Programm geeignete Laufzeittests ein, um die gewünschten Sicherheitsanforderungen zu garantieren. Welche Anforderungen bereits statisch garantiert werden können, hängt von der Mächtigkeit der Analysetechniken ab. Um Sicherheitsgarantien ohne aufwändige Analysen direkt überprüfen zu können, wurde das Prinzip des Proof-Carrying Code [39] entwickelt. Hier werden Programme mit maschinell überprüfbaren Beweisen dafür angereichert, dass sie die Sicherheitsanforderungen erfüllen [50]. Diese Beweise (auch Zertifikate genannt) werden von zertifizierenden Compilern bei der Übersetzung mitgeneriert (oder im Extremfall auch interaktiv bewiesen). PCC ist immer dann relevant, wenn eine Neuanalyse zu teuer ist, z.B. auf Smart Cards, oder wenn man den sicherheitskritischen Kern des Systems minimieren möchte. Denn ein Zertifikatprüfer ist meist um Größenordnungen einfacher als ein Analysewerkzeug, weshalb er auch einfacher zu verifizieren ist. Eine Herausforderung dieses SPPs ist die Übertragung dieses Prinzips auf Informationsflusseigenschaften.

Informationsflusskontrolle (IFC). IFC dient der Kontrolle des Informations- und Datenflusses in einem System. Informationen sind dabei beispielsweise Variablenwerte oder gerade ausgeführte Aktionen und können sowohl explizit durch Kopieren von Daten als auch implizit durch Kombination von statischem Systemwissen mit Beobachtungen des dynamischen Systemverhaltens nach außen dringen. IFC erlaubt die Modellierung sowohl von Integrität, indem der Einfluss von Informationen auf das Ergebnis von Berechnungen kontrolliert wird, als auch von Vertraulichkeit, indem der Informationsfluss in der Zukunft auf bestimmte Informationswege beschränkt wird. Nichtinterferenz bildet ein zentrales Kriterium für die Umsetzung von IFC und verlangt, dass geheime Daten oder Aktionen keinen Einfluss auf das sichtbare Systemverhalten haben, damit ein Angreifer aus seinen Beobachtungen keine Rückschlüsse auf Geheimnisse ziehen kann. Die Fähigkeiten eines Angreifers hängen von seinen Möglichkeiten ab, verschiedene Systemzustände zu unterscheiden. Fasst man die für einen Angreifer jeweils nicht-unterscheidbaren Systemzustände in Klassen zusammen, so charakterisiert die erhaltene Äquivalenzrelation das beobachtbare Systemverhalten. Für ein sicheres (deterministisches) Systems muss dieses daher unabhängig von der Variation vertraulicher Eingabewerte stets dieselben Zustandsklassen durchlaufen. Ein System ist insbesondere (bzgl. Nichtinterferenz) dann sicher, wenn es sich für einen Beobachter deterministisch verhält [45].

Sprachbasierte IFC. Sprachbasierte Ansätze untersuchen den Informationsfluss auf Programmebene. Unterteilt man Ein- und Ausgabe eines Programms in sichtbare und vertrauliche Anteile, so ist ein Programm dann sicher, wenn es für je zwei Eingaben mit gleichen sichtbaren Anteilen die gleiche sichtbare Ausgabe erzeugt. Somit kann ein Angreifer aus der sichtbaren Ausgabe nicht auf den vertraulichen Anteil der Eingabe schließen. Die Entwicklung von Sicherheitstypsystemen zur Informationsflussanalyse ist ein sehr aktives Forschungsgebiet. Es begann mit einem Basiskalkül [55] für eine einfache imperative Sprache mit Zuweisungen, Schleifen und Bedingungen. Die nachfolgende Entwicklung erfolgte entlang verschiedener, orthogonaler Richtungen: der Erweiterung des Einsatzgebiet auf weitere Programmierkonzepte, der Erweiterung der Fähigkeiten eines Beobachters und der Modellierung von Sicherheit hinsichtlich der Abschwächung der Nichtinterferenz-Anforderungen.

Ende der 90er Jahre erfolgten Erweiterung und Übertragungen des Basiskalküls auf die Behandlung von Prozeduren [53] und Ausnahmen [38], sowie die Behandlung funktionaler [44] und insbesondere objektorientierter Programmiersprachen [38, 3]. Für Programme

mit Multithreading ist das Verhalten des Schedulers sicherheitskritisch. Dies führte zu Arbeiten, in denen weitgehende Anforderungen an das Verhalten des Schedulers bzw. der Thread-Synchronisation gestellt werden [46]. Für die Behandlung der Kommunikation bei Multithreading wurden verschiedene typbasierte Ansätze [32] entwickelt. Obwohl die meisten Sprachmittel durch Typsysteme abgedeckt werden, bleibt die Herausforderung, diese zu geeigneten Typsystemen für reale Programmiersprachen zu kombinieren. Die Komplexität der dabei entstehenden Typkalküle wächst mit der Reichhaltigkeit der Programmiersprache und macht eine mechanische Verifikation der Kalküle unverzichtbar. Neue Programmierparadigmen, wie beispielsweise eine dynamische Serviceplanung und -komposition, erfordern die Entwicklung neuer Techniken, die beispielsweise den Einfluss vertraulicher Informationen auf die Planung der Services mit berücksichtigen.

Eine adäquate Modellierung der Fähigkeiten eines Angreifers ist entscheidend für den Ausschluss verdeckter Kanäle in einem zu untersuchenden System. Typsysteme wurden daher um die Erfassung von Zeitaspekten [54, 47] erweitert, und Programmtransformationen wurden zur Eliminierung sicherheitsrelevanter Laufzeitunterschiede [23] entwickelt. Damit sind allerdings bei weitem nicht alle Möglichkeiten verdeckter Kanäle erfasst. Weitere Beispiele sind etwa der Strom- oder anderer Ressourcenverbrauch eines Systems. Zur Vermeidung von Informationsflüssen durch Analyse der statistischen Verteilungen von Systemverläufen wurden eine Reihe von Techniken vorgeschlagen [36, 48]. Der Begriff der Nichtinterferenz stößt an seine Grenzen, wenn öffentlich zugängliche Informationen von geheimen Daten intensional abhängen, wie beispielsweise bei der Freigabe ehemals geheimer Informationen, bei der Überprüfung von Passwörtern oder bei der verschlüsselten Übertragung geheimer Daten über öffentliche Kanäle. Zur Lösung dieser Probleme wurden in der Vergangenheit verschiedene Mechanismen entwickelt. In der quantitativen IFC [28, 34] wird der Informationsfluss nicht vollständig verboten sondern nur auf eine festgelegte Anzahl von Bits beschränkt. Demgegenüber erlauben Deklassifikationsverfahren das explizite Herunterstufen vertraulicher Informationen [49]. Andere Ansätze ersetzen Nichtinterferenz durch komplexitätstheoretische Abschätzungen [24]. Eine Integration solcher Ansätze mit Multithreading ist noch völlig ungelöst.

Spezifikationsbasierte IFC. Sprachbasierte Ansätze setzen auf der Implementierungsebene auf und sind daher für die Sicherheitsmodellierung auf abstrakten Entwurfsebenen ungeeignet. Spezifikationsbasierte Ansätze verwenden Automaten (oder auch Prozessalgebren) und deren Abläufe (Traces) als Systemmodelle. Verschiedene Sicherheitsstufen werden durch Domänen repräsentiert und Transitionen sind jeweils einer Domäne zugeordnet. Eine Politik legt fest, zwischen welchen Domänen Informationen fließen dürfen. Ein System ist sicher, wenn die Ausgabe eines Systems mit der für einen Beobachter in einer Domäne sichtbaren (Teil-)Folge von Transitionen konsistent ist, ohne dass er auf zusätzliche, nicht-sichtbare Transitionen schließen kann [13]. Seine Beobachtung muss daher mit unterschiedlichen Traces verträglich sein. Damit sind Sicherheitspolitiken typischerweise Abschlusseigenschaften über Mengen von Traces, die aus Sicht eines Beobachters das gleiche Verhalten zeigen. Die Frage, welche Abschlusseigenschaften für die Sicherheit adäquat sind, führte zur Entwicklung von Rahmenwerken [37, 29], mit denen der Sicherheitsbegriff auf individuelle Systeme angepasst werden kann. Eine durchgängige Methodik für eine formale Modellierung solcher Sicherheitsbegriffe [20] oder von Covert Channels [33] fehlt aber noch. Der Nachweis, dass ein System die Politik erfüllt, erfolgt im Großen durch Kompositionalitätsresultate [31] und im Kleinen typischerweise durch induktives Beweisen. Auch hier

fehlen geeignete Entwicklungsmethodiken für eine effiziente deduktionsgestützte Analyse.

Um von der Entwurfsebene zu einer korrekten Implementierung zu gelangen, sieht die formale Softwareentwicklung die schrittweise Verfeinerung vor. Jeder Verfeinerungsschritt soll dabei die gewünschten Systemeigenschaften erhalten. Leider erhalten die meisten bekannten Verfeinerungsmechanismen Sicherheitseigenschaften nicht. Die Herausforderung ist hier, das Prinzip der schrittweisen Verfeinerung auch bei sicherheitskritischen Applikationen anwenden zu können. Erste konzeptionelle Ansätze gibt es im Bereich der Prozessalgebren und Automaten bei Verfeinerungen im Sinne einer Reduktion von Indeterminismus oder bei Verfeinerungen von Aktionen oder Datentypen. Für eine Umsetzung solcher Konzepte auf realistische Applikationen fehlen allerdings noch entsprechende Lösungen.

3.2 Programmanalyse

Programmanalyse wurde ursprünglich als Grundlage von Codeoptimierungen in Compilern entwickelt. Erste intraprozedurale Analysen und Optimierungen stammen aus den 60er Jahren; in den 70ern wurden nicht nur eine Vielfalt zunehmend präziser Analysen, sondern auch theoretische Fundamente wie monotone Datenfluss-Rahmenwerke [21] und abstrakte Interpretation [7] entwickelt. Ab den 80er Jahren gab es eine Fülle neuer Verfahren, die insbesondere präzise interprozedurale Analysen ermöglichten. So ist interprozedurale Konstantenpropagation [5] ausweislich empirischer Untersuchungen deutlich wirksamer als rein intraprozedurale. Als eine weitreichende Standardanalyse, die für viele Zwecke nutzbar ist, wurde die Static Single Assignment Form [8] entwickelt.

Gleichzeitig wurde das Abwägen zwischen Präzision und Kosten auf eine ingenieurwissenschaftliche Grundlage gestellt: Zu vielen Analyseproblemen gibt es präzise, aber aufwändige Algorithmen wie auch schnelle, aber ungenaue; es hängt von der Anwendung und den Anforderungen an Skalierbarkeit ab, welche der Ingenieur wählt. Die entscheidenden Kategorien dazu sind Flusssensitivität (flussinsensitive Verfahren beachten nicht die Reihenfolge von Anweisungen), Kontextsensitivität (der Aufrufkontext von Prozeduren wird beachtet – sehr teuer) und Objektsensitivität (Attribute werden je nach “beherbergendem” Objekt unterschieden – noch teurer).

Als Beispiel sei die Points-to Analyse genannt. Analyse von realistischen Java-Programmen ist unmöglich bzw. nutzlos ohne präzise Points-to Analyse. Viele Jahre wurde nach guten Algorithmen gesucht, denn anfangs war sogar flussinsensitive Analyse nur auf kleine Programme anwendbar. Erst seit 2000 gelang der Durchbruch (siehe z.B. [26]). Viele Anwendungen – auch in der IFC – verwenden aber weiterhin ganz unzureichende Points-to Analysen.

Auf der Grundlage interprozeduraler Analyse, Points-to Analyse und anderer Verfahren wurden seit den 90er Jahren Programmabhängigkeitsgraphen [19] zu einer Standard-Datenstruktur, die vielfältige weitergehende Analysen, insbesondere Program Slicing und Abhängigkeitsanalyse ermöglichen. Diese sind per Konstruktion flusssensitiv, aber erst 2004 waren kontextsensitive, objektsensitive Slicer für volles Java verfügbar [15]. Kommerzielle Slicing-Implementierungen für C und C++ sind heute erhältlich. Später wurde die immer noch fragile Präzision durch Pfadbedingungen verbessert [51], die auch erstmals Abhängigkeitsgraphen für IFC nutzten.

Erst allmählich wird das Potential moderner Programmanalyse außerhalb einer engeren Community erkannt. Als Beispiel einer inzwischen großindustriellen Anwendung sei die Vorhersage des Cache-Verhaltens der Airbus-Software durch abstrakte Interpretation genannt

[57]. Das Potential von Programmanalyse für IFC ist hingegen noch nicht ausgeschöpft, da heutige IFC-Verfahren meist nicht fluss-, kontext-, oder objektsensitiv sind, oder nicht für reale Sprachen skalieren; Einbindung moderner Analysealgorithmen wird Fehlalarme reduzieren und reale Systeme behandelbar machen. Dennoch wird sprachbasierte IFC zukünftig vor einer Reihe von Herausforderungen stehen, auch wenn sie das Potential heutiger Typsysteme oder Programmanalysen voll ausschöpft:

Präzision/Skalierbarkeit: Für wirklich große Systeme reichen die heutigen Analyseverfahren nicht aus. Das Beispiel Points-to Analyse zeigt aber, dass algorithmische Durchbrüche bei intensiver Anstrengung zu erwarten sind. So wurden in letzter Zeit pfadsensitive Analysen vorgeschlagen, die an einem Programmpunkt berücksichtigen, wie man dort hingekommen ist. Dies bringt abermals einen Präzisionsschub und hat hohes Potential für Sicherheitsanalysen; skalierende, pfadsensitive IFC ist aber eine technische Herausforderung.

Kompositionalität/Modularität: Heutige Verfahren analysieren immer das ganze System; in einer Welt mobiler Softwarekomponenten und dynamisch konfigurierbarer Systeme muss es aber möglich sein, ohne Präzisionsverlust Komponenten isoliert zu analysieren und die Zwischenergebnisse später zusammenzusetzen. Sicherheitstypsysteme haben diese Eigenschaft, aber für flusssensitive (geschweige denn kontext-, objekt- oder pfadsensitive) Verfahren ist Kompositionalität ein ungelöstes Problem.

Symbolische Analysen/Einsatz von Solvern: Alle klassischen Analysen arbeiten auf abstrakten Werten, seien es Typen oder Datenflussinformation. Rechnet man stattdessen mit symbolischen Ausdrücken, so ergeben sich völlig neue Möglichkeiten für die Präzision, aber auch die semantische Charakterisierung von IFC-Situationen. Bereits jetzt kann man aus Programmen Gleichungssysteme oder Pfadbedingungen extrahieren, deren Lösung genaue Randbedingungen (sog. Zeugen) für Fluss liefern. Um dies in großem Maßstab anzuwenden, ist aber der Einsatz von Computeralgebra und der seit kurzem verfügbaren, neuen Generation von Solvern (z.B. SAT-Löser) notwendig; die Skalierbarkeit ist völlig offen.

3.3 Verifikation

Das (automatische) maschinelle Beweisen mathematischer Sätze war ein Traum aus der Anfangsphase der KI, der sich bis in die 80er Jahre hielt, aber nie wirklich substanzielle Erfolge vorweisen konnte. Erst die konsequente Einbeziehung des Menschen in den Beweisprozess brachte hier mit den *interaktiven Beweisern* den Durchbruch [4, 14, 41]. Der Mensch entwickelt die Grobstruktur der Beweise, die Maschine überprüft die logische Korrektheit und versucht, Einzelschritte selbst zu ergänzen. Dies war in den Anfängen extrem zeitaufwändig für den menschlichen Benutzer. Nach etwa 20 Jahren Forschung sind aber zwei initiale Hemmnisse erheblich entschärft. Erstens existieren nun bereits große Bibliotheken an mathematischen Grundlagen, etwa im Mizar System oder HOL Light. Zweitens verfügen interaktive Beweiser inzwischen über mächtige Beweisprozeduren (Entscheidungs- oder auch nur Semientscheidungsverfahren) für viele wichtige Theorien, von der Aussagenlogik bis zu reell abgeschlossene Körper [35]. Daher ist es inzwischen auch möglich, umstrittene mathematische Beweise wie die des Vier-Farben-Satzes und der Keplerschen Vermutung (partiell) zu verifizieren [40].

Die primäre Anwendung von Beweisern in der Informatik liegt aber in der Software-Modellierung und -Verifikation. Hier existieren seit etwa 5 Jahren signifikante Anwendungen von einer Größenordnung, die vorher undenkbar war: Programmiersprachen, Compiler und Betriebssystemkerne sind formalisiert und verifiziert worden [22, 25, 18, 1]. Es ist genau die-

ses Potenzial, das wir nutzen wollen, um die Sicherheitsanalysewerkzeuge und Methoden selbst zu verifizieren. Dies ist ob der Komplexität sowohl realer Programmiersprachen als auch von Programmanalysen und Typsystemen unumgänglich, um die Verlässlichkeit der Werkzeuge zu garantieren. Erste Schritte hierzu sind Arbeiten zur Verifikation traditioneller Programmanalysealgorithmen [43, 56]. Die Herausforderung besteht nun darin, die doch sehr vielfältigen Ansätze zu Semantik, Typen und Programmanalysen zu einer umfassenden, hinreichend allgemeinen und abstrakten Theorie zusammenzufassen, und vor allem, diese Theorien in einem Theorembeweiser bereitzustellen, um auf dieser Basis Dinge wie Typsysteme und Analysealgorithmen für IFC zu verifizieren. Neben der abstrakten mathematischen Ebene ist auch die Korrektheit der daraus automatisch oder per Hand generierten Programme eine große Herausforderung. Denn automatische Generierung effizienten Codes aus abstrakten funktionalen Beschreibungen erfordert weiterhin domänenspezifische Techniken. Im Bereich Programmanalyse gibt es hierzu noch keine tragfähigen Ansätze.

Neben der Hinwendung zu interaktiven Beweisern hat es in der letzten Dekade eine zweite Entwicklung gegeben, die die Softwareverifikation erheblich voran gebracht hat: Die Konzentration auf den Beweis einfacher Eigenschaften von Quellprogrammen mit Hilfe automatischer Beweiser [10], sogenannter SMT-Löser [2]. Diese beruhen auf einer Kombination von Aussagenlogik (basierend auf extrem effizienten SAT-Lösern, die noch Formeln im Bereich von 10^6 Variablen schnell lösen können), linearer Arithmetik, speziellen Theorien wie Bitvektoren, und Heuristiken zur Instanziierung von Quantoren. Wir sehen hier ein doppeltes Anwendungspotenzial: Die automatische Verifikation von Informationsflusseigenschaften von Programmen statt der traditionellen Safety-Eigenschaften, und die Nutzung von SMT-Lösern bei der interaktiven Verifikation von Analysewerkzeugen. Eine längerfristige Herausforderung ist es, hier spezielle Entscheidungsprozeduren für erststufige Formeln zu entwickeln, die über Quantorenelimination für rein arithmetische Formeln hinausgehen.

4 Arbeitsprogramm

Die Arbeiten im SPP werden sich an den drei Hauptrichtungen (Abschnitt 2.2) orientieren. Während die ersten beiden Richtungen jeweils in eigenen Arbeitsbereichen (A und C) angegangen werden, hat die dritte Hauptrichtung einen querschnittlichen Charakter. Abbildung 1 skizziert den Datenfluss zwischen den Arbeitsbereichen des SPPs (als Ovale dargestellt).

Der orange unterlegte Ausschnitt verdeutlicht die Arbeitsbereiche, die für die angestrebte Zielsetzung einer semantisch fundierten Zertifizierung zuverlässig sicherer Softwaresysteme unabdingbar sind. Im Arbeitsbereich MODELLIERUNG VON SICHERHEITSANFORDERUNGEN (A) werden z.B. Techniken zur Anforderungsanalyse, ein Katalog von Sicherheitseigenschaften (insbesondere praktikable Varianten von Noninterference), und illustrierende Beispiele für die Spezifikation von Sicherheitsanforderungen mit Hilfe dieser Eigenschaften (inklusive fundierter Adäquatheitsargumente) entstehen. Im Arbeitsbereich SICHERHEITSANALYSE (C) werden neue Analysetechniken sowie -werkzeuge entwickelt und Korrektheitsbeweise geführt, um deren Zuverlässigkeit zu garantieren. Damit Softwaresysteme auf den verschiedenen Ebenen von abstrakten Spezifikationen bis hin zum Programmcode betrachtet werden können, werden im Arbeitsbereich (B) Techniken zur Modellierung sicherheitskritischer Systeme und formale Charakterisierungen von Programmsemantiken entstehen. Die aus der Sicherheitsanalyse resultierenden Zertifikate werden auf Grund der semantischen Fundierung letztendlich die angestrebten zuverlässigen Sicherheitsgarantien bieten.

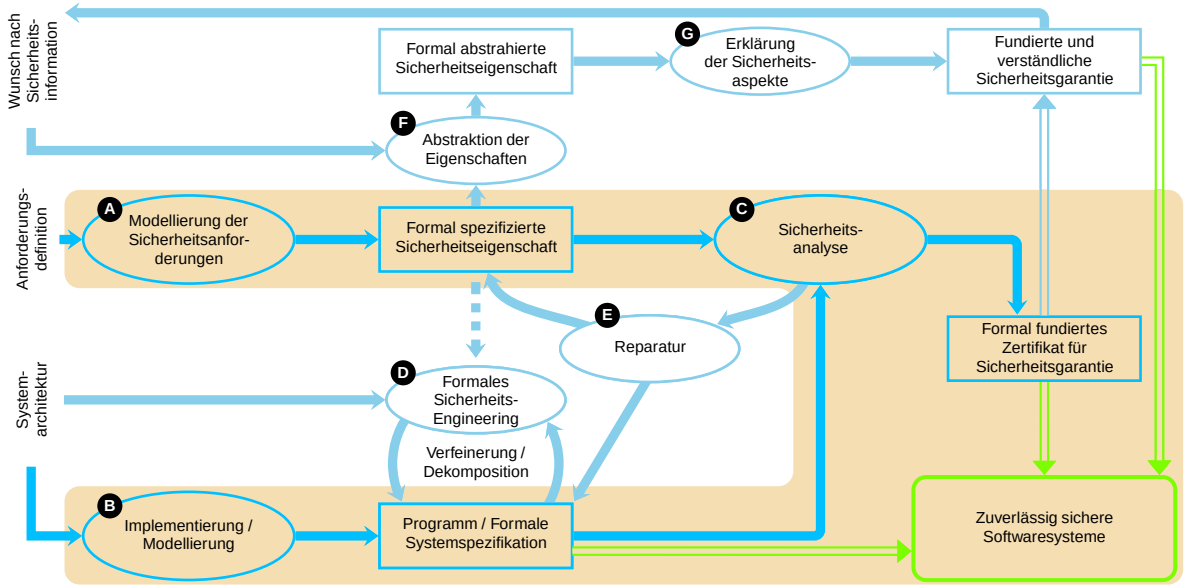

Abbildung 1: Struktur des Arbeitsprogramms und Datenfluss zwischen Arbeitsbereichen

Die übrigen, im Diagramm nicht unterlegten Arbeitsbereiche beschäftigen sich mit Fragen, die zwar aus theoretischer Sicht für eine semantisch fundierte Zertifizierung von Sicherheit nicht notwendig, für die praktische Realisierung der Vision des SPPs aber unabdingbar sind. Die Arbeitsbereiche (D) und (E) befassen sich mit Fragen der Softwaretechnik, genauer der schrittweisen Entwicklung zertifizierbar sicherer Software bzw. der Reparatur unsicherer Programme basierend auf den Informationen aus einer fehlgeschlagenen Sicherheitsanalyse. Die Arbeitsbereiche (F) und (G) zielen gemeinsam auf eine subjektiv verständliche, aber fundierte Kommunikation der zertifizierten Sicherheitsgarantien.

4.1 Exemplarische Forschungsvorhaben und Fragestellungen

Die folgende Liste möglicher Forschungsvorhaben skizziert exemplarisch wie Teilprojekte die Arbeitsbereiche des SPPs mit konkreten Forschungsfragen ausgestalten werden, wobei in Klammern auf die jeweils relevanten Arbeitsgebiete aus Abbildung 1 verwiesen wird:

Abstraktionen für Informationsflusskontrolle (ACEFG). Abstraktionen werden gesucht, die eine vereinfachte Sicht auf komplexe Softwaresysteme bieten und dadurch z.B. die Sicherheitsanalyse oder die Reparatur unsicherer Programme vereinfachen. Es soll auch ausgelotet werden, inwieweit Abstraktionen gezielt generiert werden können, um semantisch fundierte Sicherheitsgarantien für einen Nutzer subjektiv verständlicher werden zu lassen.

Automatische Codegenerierung für Informationsflusskontrollalgorithmen (C). Es werden Techniken entwickelt, die aus funktionalen Beschreibungen von Informationsflussanalysen automatisch Implementierungen gewinnen können. Diese sollen sowohl effizient als auch per Konstruktion korrekt sein. Es soll insbesondere untersucht werden, wie dabei mutierbare Datenstrukturen wie z.B. Arrays verwendet werden können.

Informationstheoretische Analyse verdeckter Kanäle (AB). Unerwünschte Informationsflüsse werden insbesondere durch verdeckte Kanäle in der Ausführungsumgebung von Programmen ermöglicht. Für typische Ausführungsumgebungen (z.B. Browser, Virtualisierungssoftware, Betriebssystem) werden formale Sicherheitsmodelle entwickelt, die eine qualitative und quantitative Bewertung der Gefährdung durch verdeckte Kanäle ermöglichen.

Präzise Informationsflusskontrolle für mobile Softwarekomponenten (C). Für die Informationsflusskontrolle von verteilten und mobilen Softwarekomponenten werden präzise, kompositionale Analysemethoden entwickelt und formal verifiziert. Darüberhinaus sollen die entwickelten Methoden implementiert und ihre Skalierbarkeit empirisch validiert werden.

Requirements Engineering für Informationsflusssicherheit (AD). Die Ergebnisse forensischer Analysen werden genutzt, um unsichere Informationsflüsse und, darauf aufbauend, erwünschte Informationsflusseigenschaften zu identifizieren. Im Anschluss werden die Eigenschaften sprachlich präzise, in Teilen sogar formal definiert, um als Ausgangspunkt für die Entwicklung neuer, besserer Systeme dienen zu können.

Zuverlässig sichere Service-orientierte Architekturen (ABD). Ausgehend von einem existierenden Anwendungszenaro im Bereich Krisenleitstände werden eigenschaftsbasierte Modellierungen für Sicherheitsanforderungen an Dienste entwickelt. Darüberhinaus wird untersucht, wie sich diese Anforderungen im Rahmen einer Serviceplanung zu Sicherheitsanforderungen an Hilfsdienste dekomponieren lassen.

4.2 SPP-weite Anwendungsszenarien und Demonstratoren

Synergieeffekte zwischen Teilprojekten werden durch die Bearbeitung verwandter Fragestellungen, durch den Wettbewerb zwischen Analysewerkzeugen und Modellierungstechniken sowie durch die intensive Kommunikation und interdisziplinäre Zusammenarbeit innerhalb des SPPs ([. . .]) entstehen. Darüberhinaus werden drei gemeinsame Fallstudien die hochschulübergreifende Zusammenarbeit fördern, die basierend auf der Ausrichtung der bewilligten Teilprojekte zu Beginn des SPPs ausgewählt und präzisiert werden. Neben der Zusammenarbeit soll hierdurch auch die Interoperabilität und Durchgängigkeit der entwickelten Methoden und Werkzeuge gefördert werden. Diese Anwendungsszenarien könnten z.B. folgenden Anwendungsbereichen entstammen: INTERNET BROWSER (Sicherheit erweiterbarer Softwaresysteme), E-HEALTH UND TELEMATIK (Compliance, Sicherheit verteilter Systeme), E-VOTING UND WAHLMASCHINEN (Sicherheit mobiler und eingebetteter Systeme), E-BUSINESS APPLIKATION (Sicherheit Service-orientierter Architekturen).

Die Ergebnisse der im SPP durchgeführten Fallstudien werden nicht nur der Evaluation der entwickelten Methoden und Werkzeuge dienen, sondern hieraus sollen auch Demonstratoren entstehen, mit denen die Ergebnisse des SPPs insbesondere interessierten Vertretern aus Industrie und öffentlicher Verwaltung illustriert werden können. Mehrere Firmen und Behörden haben ihr Interesse gegenüber den Initiatoren bekundet. [. . .]

5 Organisatorischer Rahmen und Kommunikation

[. . .]

Literatur

[1] E. Alkassar, N. Schirmer, and A. Starostin. Formal pervasive verification of a paging mechanism. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 109–123. Springer, LNCS 4963, 2008.

[2] C. Barrett, L. de Moura, and A. Stump. Design and results of the 1st satisfiability modulo theories competition. *Journal of Automated Reasoning*, 35(4):373–390, 2005.

[3] G. Barthe, T. Rezk, and A. Basu. Security types preserving compilation. *Comp. Languages, Systems and Structures*, 33(2):35–59, 2007.

[4] Y. Bertot and P. Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer, 2004.

[5] D. Callahan et al. Interprocedural constant propagation. In *ACM Symp. on Compiler Construction*, pages 152–161. ACM, 1986.

[6] Common Criteria for information technology security evaluation. http://www.commoncriteriaportal.org/, 9/2006.

[7] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Symp. on Principles of Programming Languages*, pages 238–252, 1977.

[8] R. Cytron et al. Efficiently computing static single assignment form and the control dependence graph. *ACM Trans. Program. Lang. Syst.*, 13(4):451–490, 1991.

[9] N. Damianou et al. The Ponder policy specification language. In *Int. Works. on Policies for Distributed Systems and Networks*, pages 18–38. Springer, LNCS 1995, 2001.

[10] C. Flanagan et al. Extended static checking for Java. In *PLDI*, pages 234–245, 2002.

[11] Gartner Inc. Gartner predicts worldwide security software revenue to grow 11 percent in 2008. Pressemeldung `http://www.gartner.com/it/page.jsp?id=653407`, 22. April 2008.

[12] M. Gilliot, G. Müller, and S. Wohlgemuth. Abschlußbericht des Schwerpunktprogramms Sicherheit in der Informations- und Kommunikationstechnik (SPP 1079). `http://www.telematik.uni-freiburg.de/spps`, 2007.

[13] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symp. on Security and Privacy*, pages 11–20, 1982.

[14] M. Gordon and T. Melham, editors. *Introduction to HOL: a theorem-proving environment for higher order logic*. Cambridge University Pess, 1993.

[15] C. Hammer and G. Snelting. An improved slicer for Java. In *ACM Works. on Program Analysis for Software Tools and Engineering*, pages 17–22. ACM Press, 2004.

[16] Heise Online. Angreifer können Liste besuchter Webseiten auslesen. Meldung im Newsticker `http://www.heise.de/newsticker/meldung/86113`, 02. März 2008.

[17] Heise Online. Chrome ruft Google. Meldung im Newsticker `http://www.heise.de/newsticker/meldung/115537`, 05. September 2008.

[18] G. Heiser et al. Towards trustworthy computing systems: taking microkernels to the next level. *Operating Systems Review*, 41(4):3–11, 2007.

[19] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Syst.*, 12(1):26–60, 1990.

[20] D. Hutter, H. Mantel, I. Schäfer, and A. Schairer. Security of multi-agent systems: A case study on comparison shopping. *Journal of Applied Logic. Special Issue on Logic Based Agent Verification*, 5(2): 303-332, 2007.

[21] J. Kam and J. Ullman. Monotone data flow analysis frameworks. *Acta Informatica*, 7(3):305–317, 1977.

[22] G. Klein and T. Nipkow. A machine-checked model for a Java-like language, virtual machine and compiler. *ACM Trans. on Programming Languages and Systems*, 28(4):619–695, 2006.

[23] B. Köpf and H. Mantel. Transformational typing and unification for automatically correcting insecure programs. *Int. Journal of Information Security*, 6(2/3):107–131, 2007.

[24] P. Laud. Semantics and program analysis of computationally secure information flow. In *European Symp. on Programming*, pages 77–91. Springer, LNCS 2028, 2001.

[25] X. Leroy. Formal certification of a compiler backend or: programming a compiler with a proof assistant. In *ACM Symp. on Principles of Programming Languages*, pages 42–54, 2006.

[26] O. Lhoták and L. Hendren. Evaluating the benefits of context-sensitive points-to analysis using a BDD-based implementation. *ACM Trans. Softw. Eng. Methodol.*, 18(1):1–53, 2008.

[27] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In *UML 2002*, pages 426–441. Springer, 2002.

[28] G. Lowe. Quantifying information flow. In *15th IEEE Comp. Security Foundations Works.*, pages 18–31, 2002.

[29] H. Mantel. Possibilistic definitions of security – an assembly kit. In *IEEE Comp. Security Foundations Works.*, pages 185–199, 2000.

[30] H. Mantel. Preserving information flow properties under refinement. In *IEEE Symp. on Security and Privacy*, pages 78–91, 2001.

[31] H. Mantel. On the composition of secure systems. In *IEEE Symp. on Security and Privacy*, pages 88–104, 2002.

[32] H. Mantel and A. Sabelfeld. A generic approach to the security of multi-threaded programs. In *14th IEEE Comp. Security Foundations Works.*, pages 126–142, 2001.

[33] H. Mantel and H. Sudbrock. Comparing countermeasures against interrupt-related covert channels in an information-theoretic framework. In *20th IEEE Comp. Security Foundations Symp.*, pages 326–340, 2007.

[34] S. McCamant and M. Ernst. Quantitative information flow as network flow capacity. In *ACM Conf. on Programming Language Design and Implementation*, pages 193–205, 2008.

[35] S. McLaughlin and J. Harrison. A proof-producing decision procedure for real arithmetic. In *Automated Deduction — CADE-20*, pages 295–314. Springer, LNCS 3632, 2005.

[36] J. McLean. Security models and information flow. In *IEEE Symp. on Security and Privacy*, pages 180–187, 1990.

[37] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *IEEE Symp. on Research in Security and Privacy*, pages 79–93, 1994.

[38] A. Myers. JFlow: Practical mostly-static information flow control. In *ACM Symp. on Principles of Programming Languages*, 1999.

[39] G. Necula and P. Lee. The design and implementation of a certifying compiler. *ACM SIGPLAN Notices*, 33(5):333–344, 1998.

[40] T. Nipkow, G. Bauer, and P. Schultz. Flyspeck I: Tame graphs. In *Automated Reasoning*, pages 21–35. Springer, LNCS 4130, 2006.

[41] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer, LNCS 2283, 2002.

[42] OASIS. WS-security policy 1.2. http://docs.oasis-open.org/ws-sx/ws-securitypolicy, 2007.

[43] D. Pichardie. Building certified static analysers by modular construction of well-founded lattices. In *1st Int. Conf. on Foundations of Informatics, Computing and Software*, ENTCS vol. 212, pages 225–239, 2008.

[44] F. Pottier and S. Conchon. Information flow inference for free. In *ACM Int. Conf. on Functional Programming*, pages 46–57, 2000.

[45] A. Roscoe. CSP and determinism in security modelling. In *IEEE Symp. on Security and Privacy*, pages 114–127, 1995.

[46] A. Sabelfeld. The impact of synchronisation on secure information flow in concurrent programs. In *4th Int. Andrei Ershov Memorial Conf. on Perspectives of System Informatics*, pages 225–239. Springer, LNCS 2244, 2001.

[47] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *13th IEEE Comp. Security Foundation Works.*, pages 200–215, 2000.

[48] A. Sabelfeld and D. Sands. A PER model of secure information flow in sequential programs. *Higher Order and Symbolic Computation*, 14(1):59—91, 2001.

[49] A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *18th IEEE Comp. Security Foundations Works.*, pages 255–269, 2005.

[50] F. Schneider, G. Morrisett, and R. Harper. A language-based approach to security. In *Informatics: 10 Years Back, 10 Years Ahead*, pages 86–101. Springer, LNCS 2000, 2000.

[51] G. Snelting, T. Robschink, and J. Krinke. Efficient path conditions in dependence graphs for software safety analysis. *ACM Trans. Softw. Eng. Methodol.*, 15(4):410–457, 2006.

[52] Symantec Corporation. Symantec global internet security threat report. http://www.symantec.com/content/de/de/about/downloads/PressCenter/ISTR_XIII.pdf, 2008.

[53] D. Volpano and G. Smith. A type-based approach to program security. In *TAPSOFT'97*, pages 607–621. Springer, LNCS 1214, 1997.

[54] D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. *Journal of Comp. Security*, 7(2/3):231–253, 1999.

[55] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Comp. Security*, 4(3):167–187, 1996.

[56] D. Wasserrab and A. Lochbihler. Formalizing a framework for dynamic slicing of program dependence graphs in isabelle/hol. In *21st Int. Conf. of Theorem Provingin Higher Order Logics*, pages 294–309. Springer, LNCS 5170, 2008.

[57] R. Wilhelm et al. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. on Embedded Computing Systems*, 7(3):1–53, 2008.