Unumgehbarkeit, Tamper Resistance und Trusted Computing Base

PG 450
Entwurf und Implementierung eines Prototyps zur zustandsabhängigen Zugriffskontolle und -rechteverwaltung

Marc Seitz
MarcSeitz@gmx.net
Dortmund

Christian Blichmann
pg450@blichmann.de
Dortmund

Universität Dortmund
April 2004

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Bestehende IT-Infrastruktur . . . 1
  - 1.2.1 Ungesicherter Bootvorgang . . . 1
  - 1.2.2 Ungesicherter Betriebssystemstart . . . 2
  - 1.2.3 Ungesicherte Laufzeitumgebung . . . 2
- 1.3 Angriffsszenarien . . . 2
- 1.4 Lösungsansätze . . . 3

2 Personal Secure Booting **4**
- 2.1 Ziele . . . 4
- 2.2 Arbeitsweise AEGIS/sAEGIS . . . 4
- 2.3 Bootstrap Prozess in 7 Schritten . . . 4
- 2.4 Grundannahmen . . . 6
- 2.5 Angriffsszenarien . . . 6
- 2.6 Aktueller Stand . . . 7

3 TCG und Trusted Computing Base **8**
- 3.1 Organisation . . . 8
  - 3.1.1 Was ist die TCG? . . . 8
  - 3.1.2 Entwicklung und Mitglieder . . . 8
- 3.2 TCG-Spezifikationen . . . 9
  - 3.2.1 TPM-Basisfunktionen . . . 9
  - 3.2.2 Funktionen im Detail . . . 10
  - 3.2.3 Trusted Software Stack . . . 11
- 3.3 Missverständnisse . . . 11
- 3.4 Ausblick . . . 12
- 3.5 Microsofts NGSCB . . . 12
  - 3.5.1 Überblick . . . 12
  - 3.5.2 Schlüsselfunktionen . . . 13
  - 3.5.3 TCG vs. NGSCB . . . 13
  - 3.5.4 Ausblick . . . 14

4 Java und .Net **15**
- 4.1 Architektur . . . 15
- 4.2 Java 1.0: Sandboxing . . . 16
- 4.3 Java 1.1: Digital signierte Klassen . . . 17
- 4.4 Java 1.2: Policies . . . 17
- 4.5 Schlussfolgerungen für SDSD-System . . . 18

5 Anwendungsfälle im SDSD-System **19**
5.1 Ideale Umgebung 19
5.2 Störfaktoren und deren Lösungen 19
5.3 Fazit 20

6 Schlussbetrachtung **21**

Glossar **23**

Literaturverzeichnis **25**

Zusammenfassung

Kaum ein anderes Thema wird in der letzten Zeit in den Medien so hochstilisiert wie IT-Sicherheit. Um die bestehende IT-Infrastruktur sicherer gegen Angriffe von außen zu machen und Anwendern wie Systemadministratoren die Möglichkeit zu geben, Software auf ihre Herkunft/ihre Integrität zu überprüfen, gibt es Bestrebungen, eine „Trusted Computing Base“ auf breiter Basis zu etablieren. Eine solche Plattform muss verschiedene Voraussetzungen erfüllen, unter anderem Unumgehbarkeit (d.h. das System darf nicht zu untergraben sein) und Fälschungssicherheit (d.h. Computersysteme sollen „vertrauenswürdig“ werden, indem sie vor Manipulationen geschützt werden).

Kapitel 1

Einleitung

1.1 Motivation

Die bestehenden Personal Computing Architekturen mit konventionellen Betriebssystemen (Windows, Linux, MacOS, FreeBSD, ...) sind auf Grund ihrer hohen Komplexität stark fehleranfällig und lassen sich - durch konzeptionelle Schwächen bedingt - nicht ausreichend vor Manipulationen absichern.
Obwohl es einige Bestrebungen gibt, auch momentan eingesetzte Betriebssysteme abzusichern (beispielsweise durch internationale Standards wie z.B. die Common Criteria Certification mit ihren Evaluation Assurance Level), gibt es bislang nur wenige Betriebssysteme, die bereits in der Designphase auf Sicherheit optimiert wurden (Multics, EROS, Birlix, ...).
Viele der Angriffe erfolgen außerdem von „innen“, d.h. aus dem Inneren einer Organisation, so dass hier auch eine Kontrolle durch den Benutzer erwünscht ist. Im Regelfall ist der Benutzer der Attacke hilflos ausgeliefert und kann keine Gegenmaßnahmen einleiten.
Um eine PC-Architektur abzusichern, muss die Integrität des zugrunde liegenden Betriebssystems sowie aller relevanten Hardware-Module gewährleistet werden können. Diese Absicherung muss einem so genannten Trusted Path folgen, der bereits beim Einschalten des Rechners von der Firmware der Peripherie bis hin zum Start des Kernels eine umfassende Integritätskontrolle durch den Einsatz von kryptographisch sicheren Signatur- und Verschlüsselungsalgorithmen ermöglicht.

1.2 Bestehende IT-Infrastruktur

Die bestehende IT-Infrastruktur hat im Groben vier Angriffsstellen. Von diesen vier Angriffsstellen ist die erste der Benutzer selbst, der nicht verantwortungsbewusst mit Passwörtern, Benutzerkonten etc. umgeht. Diese Schwachstelle ist mit technischen Mitteln (leider) nicht zu beheben und erfordert ein Umdenken der Verantworlichen innerhalb der jeweiligen Organisation.
Die anderen drei Systemschwachpunkte sind:

- Ungesicherter Bootvorgang
- Ungesicherter Betriebssystemstart
- Ungesicherte Laufzeitumgebung

1.2.1 Ungesicherter Bootvorgang

Bei den heutigen bestehenden Systemen erfolgt ein ungesicherter Bootvorgang. Es wird nicht geprüft, ob Hardware modifiziert oder ausgetauscht worden ist.
Ebenso kann der Benutzer nach Belieben selbst Hardware austauschen, ohne diese vorher zertifizieren zu müssen.
Eine Lösung für dieses Problem bietet das Personal Secure Booting [1].

[1] siehe Kap. 2

1.2.2 Ungesicherter Betriebssystemstart

Nachdem nun der Bootvorgang gesichert abläuft, erfolgt jedoch ein ungesicherter Betriebssystemstart. Daraus folgt, dass zwar die Hardware nicht mehr manipuliert werden kann, die Software jedoch vor Manipulationen durch Dritte nicht geschützt ist. Manipulationen werden im Regelfall auch noch durch den Benutzer unterstützt, indem der System-Login meistens ungesichert ist (kein Passwort) oder die Passwörter leicht zu erraten oder an einer prominenten Stelle zu finden sind.
Dieses Problem ist mit den Spezifikationen der TCG [2] lösbar.

1.2.3 Ungesicherte Laufzeitumgebung

Da es keine gesicherte Laufzeitumgebung gibt, ist normale Software zur Laufzeit kompromittierbar. Ebenso können Speicherschutzmechanismen ausgehebelt oder Buffer-Overflows und dergleichen herbeigeführt werden.
Darüber hinaus gibt es auf dem Markt eine Vielzahl von Software-Tools, um die jetzigen Sicherheitsmechanismen der PCs/Workstations/Mainframes/etc. zu umgehen.

1.3 Angriffsszenarien

Die Angriffsszenarien können sehr vielfältig sein; auf verschiedenen Plattformen werden auch unterschiedliche Arten des Angriffes genutzt.
Die Angriffe können auf die Hard- und Software des jeweiligen PCs erfolgen. Die meisten Angriffe erfolgen außerdem auf Microsoft Windows Betriebssysteme, da diese einen sehr hohen Marktanteil haben (momentan ca. 85%). Jedoch sind Angriffe auf andere Betriebssysteme ebenso möglich, auf Grund des niedrigeren Marktanteils aber nicht so häufig. Exemplarisch zählen wir nun einige der möglichen Angriffsszenarien auf:

Viren – Bösartige, sich selbst verbreitende (z.B. über das Inter-/Intranet) Programme

Trojaner – eigentlich „Trojanisches Pferd“, Programme die beim Benutzer den Anschein erwecken, ein seriöses Programm zu sein, nach dem Aufruf aber z.B. Viren installieren oder Daten ausspionieren.

Würmer – Meist harmlose, sich selbst verbreitende Programme, die aber durch ihre - meist schwierige - Entfernung hohe Kosten verursachen

Social Engineering – Das Ausspionieren von Passwörter oder Login-Daten durch Ausnutzung der typischen menschlichen Schwächen. Kann über alle verbreiteten Kommunikationsmittel erfolgen. Meist ein Anruf, bei dem der Angreifer sich als Administrator ausgibt.

Backdoors – Durch den Software-/Hardware-Hersteller oder einen bösartigen Programmierer eingebaute Hintertüren, also absichtliche Schwächen in Protokollen oder Masterpasswörter.

Password-/Network-Sniffer/Keylogger – Programme die Passwörter durch abhören des Netzwerkverkehrs oder Tastenanschlägen

Masterpasswörter – Vom Hersteller meist zu Testzwecken eingebaute Standardpasswörter, die sich im finalen Produkt wiederfinden (aktuelles Beispiel: Masterpasswörter in Ciscos WLSE und HSE[3]).

[2] Trusted Computing Group
[3] `http://www.cisco.com/warp/public/707/cisco-sa-20040407-username.shtml`

1.4 Lösungsansätze

In den vorangegangenen Absätzen sind bereits einige Lösungsansätze erwähnt worden. Das Spektrum der möglichen Lösungen ist genauso breit die möglichen Angriffe.
Zu erwähnen wären da z.B. proprietäre Systeme, die von der Designphase an auf Sicherheit optimiert worden sind. Diese sind unter anderem Multics[4], EROS[5] und Birlix[6].
Ferner gibt es die so genannten Common Criteria[7] nach ISO/IEC 154080, in denen erprobte Standardmaßnahmen definiert werden, wie gängige Betriebsystem-Software-Kombinationen abzusichern sind. Die höchste aktuelle Sicherheitsstufe bei Betriebssystemen ist EAL3[8] (zum Vergleich: die Bankkarten-Sicherheitsstufe ist EAL4 oder EAL4+).
Eine weitere Möglichkeit ist das so genannte Sandboxing, also das Einkapseln von Software in eine virtualisierte Umgebung. Kein echtes Sicherheitskonzept ist hingegen „Security through obscurity“, das darauf baut Implementationsdetails einer Software/eines Algorithmus geheim zu halten.
Wir wollen uns jedoch auf die für uns relevanten Sicherheitskonzepte beschränken. Dieses sind Personal Secure Booting und die Umsetzung der Spezifikationen der TCG.

[4] `http://www.multicians.org/`
[5] `http://www.eros-os.org/`
[6] `http://citeseer.ist.psu.edu/context/59373/0`
[7] `http://www.bsi.de/cc/`
[8] Evaluation Assurance Level 3

Kapitel 2

Personal Secure Booting

2.1 Ziele

Hauptziel des Personal-Secure-Booting-Konzepts ist, dem Benutzer einen sicheren Bootvorgang des Systems zu ermöglichen – bis zu dem Zeitpunkt, an dem das Betriebssystem die Kontrolle übernimmt. Es beruht auf der Tatsache, dass alles Hardwarekomponenten des Systems zertifiziert sind und durch die vorangegangene Komponente autorisiert werden. Auch eine Speicherung der Zertifikate auf Smartcards ist vorgesehen.
Grundproblem bei den Zielen ist, daß der Benutzer der ersten Komponente trauen muss[1].

2.2 Arbeitsweise AEGIS/sAEGIS

Im Folgenden beschreiben wir die Arbeitsweise des AEGIS/sAEGIS-Bootstrap-Prozesses.

Der Administrator oder autorisierte Benutzer generiert einen Hashwert (MAC[2]) der am Bootstrap-Prozess beteiligten Komponenten. Aus dem Hashwert wird ein Zertifikat C generiert.
Ein autorisierter Dritter (z.B. das Trustcenter) signiert C mit seinem privaten Schlüssel.
Anschließend wird C in der Komponente selbst gespeichert, wenn dies technisch möglich ist. Ansonsten weicht man auf den Flashspeicher der Motherboards aus.
Der komponenten-interne Code (z.B. Firmware, Grafik-BIOS) wird nur ausgeführt, wenn

- C nicht abgelaufen ist
- Signatur von C gültig ist
- Hashwert übereinstimmt

AEGIS/sAEGIS kann die Integritätsgarantie nur für den Systemstart übernehmen.

2.3 Bootstrap Prozess in 7 Schritten

Der Bootstrap-Prozess lässt sich in 7 Schritte aufteilen[3].
Im Folgenden werden die einzelnen Schritte des Prozesses erläutert:

[1] siehe auch Kap. 2.4
[2] Message Authentication Code
[3] siehe Abb. 2.1

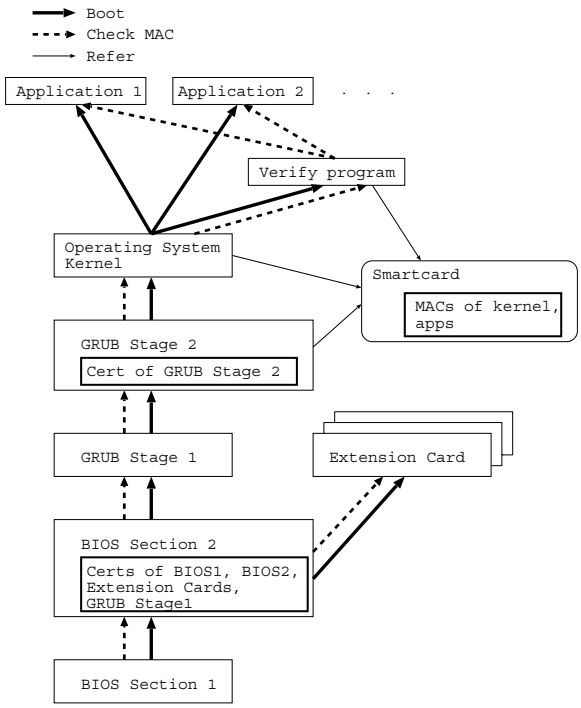

Abbildung 2.1: Bootstrap Prozess [IAPR01]

1. Power-On Self Test
   - Prozessor führt Selbsttest durch
   - Startet die Bootsequenz
2. BIOS Section 1
   - Enthält Grundfunktionen zur Integritätsprüfung (MD5, SHA1 und RSA)
   - Überprüft sich selbst und BIOS Section 2
   - Bootet BIOS Section 2 falls erfolgreich
3. BIOS Section 2
   - Überprüft ROM der Erweiterungskarten durch Zertifikate
   - Startet diese bei erfolgreicher Prüfung
   - Lädt den Primary Boot-Block (Grub Stage 1)[4] von der Festplatte und überprüft diesen
4. GRUB Stage 1
   - Lädt und verifiziert GRUB Stage 2
   - Benutzt die Routinen aus BIOS Section 1
5. GRUB Stage 2
   - Lädt den Betriebssystemkern und verifiziert diesen
   - Verifiziert die „Verification Tools“ des Kernels
   - Bindet das Dateisystem ein
   - Lädt die MAC von der Smartcard und verifiziert damit die Startdateien

[4] Bei der Intel x86-Architektur ist der der Master Boot Record (MBR) auf 512 Byte beschränkt, daher die Unterteilung in zwei Stages

6. Kernel
    - Benutzt das „Verify Program“ zum verifizieren der wichtigen Systemdateien
    - Startet die Systemdienste

Damit ist der Bootstrap-Vorgang abgeschlossen.

2.4 Grundannahmen

Auch das AEGIS/sAEGIS-Konzept kommt nicht ohne einige Grundannahmen aus:

1. Die erste Komponente des Gesamtsystems (hier also das BIOS) ist als „vertrauenswürdig“ eingestuft worden. Dies ist insofern problematisch, als dass hier der Benutzer einem Trustcenter glauben muss; er kann die Integrität der Komponente nicht weiter überprüfen. Dieses Problem soll durch die Smartcard-Lösung vom sAEGIS behoben werden, die einen Hashwert des BIOS auf der Smartcard speichert und verifiziert. Momentan ist dieser Ansatz aber nur prototypisch vorhanden.
2. Die BIOS Section 1 ist nicht manipulierbar. Kann durch den Einsatz von nur einmal beschreibbarem, fest aufgelötetem Flash-Speicher erreicht werden.
3. Alle verwendeten kryptographischen Hashfunktionen arbeiten kollisionsfrei und der RSA-Algorithmus kann nicht kompromittiert werden. Für den praktischen Einsatz erscheint diese Annahme vernünftig.
4. Alices[5] privater Schlüssel ist Mallory[6] unbekannt.
5. Mallory hat keinen Zugang zur Smartcard. Diese potentielle Sicherheitslücke liegt allein in der Hand der Benutzer im Speziellen und dem Sicherheitsbewusstsein/-bedürfnis der Organisation im Besonderen.
6. Die Kommunikation mit der Smartcard per Definition sicher. Auch dies ist für den praktischen Einsatz keine große Einschänkung, da man spezielle Smartcard-Leser mit eigener Tastatur zur PIN-Eingabe verwenden kann und die Kommunikation mit dem Rechner ebenfalls über kryptographisch gesicherte Protokolle laufen könnte.

2.5 Angriffsszenarien

Mallory könnte als Systemadministrator agieren, damit hätte er theoretisch vollen Zugriff auf das System.
Dieser klassische Angriff „von Innen“ ist durch den Einsatz von Smartcards vermeidbar, da diese per Definition als sicher eingestuft werden. Somit würden nicht autorisierte Veränderungen am System entdeckt.

Es könnten Systemkomponenten modifiziert werden. Da nur der Startvorgang gesichert ist, würde dieses dem Benutzer im laufenden Betrieb nicht auffallen (Bedrohung durch Viren, Trojaner etc.). Der Benutzer kann die Systemintegrität allerings jederzeit durch einen Neustart wiederherstellen. Leider ist diese Lösung bei z.B. Servern wenig praktikabel, da das Hochfahren oft mit erheblichem Zeit/Kostenaufwand verbunden ist.

Der komplette PC könnte gestohlen werden. Das Verändern bzw. Auslesen von Daten könnte man durch den Einsatz von Smartcards, PIN sowie kryptographischen Protokollen unterbinden.

[5] Name des legitimen Benutzers in kryptographischen Szenarien
[6] Angreifer in kryptographischen Szenarien

2.6 Aktueller Stand

Es existiert bereits eine erste funktionsfähige Version, welche mit Hilfe des Linux-BIOS-Projektes[7] realisiert und implementiert wurde. Die Veröffentlichung dieses Projektes erfolgt unter dem Namen SEBOS [8] und ist bootfähig für viele Betriebssysteme (aktuell Linux, FreeBSD und Windows 2000). Die vorgestellte sAEGIS-Variante ist erst prototypisch vorhanden.
Es ist bis jetzt jedoch noch kein offizieller Release erfolgt – dieser soll aber bald unter der GPL[9] erfolgen.

[7] `http://www.linuxbios.org/`
[8] Security Enhanced Bootloader for Operating Systems, `http://www.missl.cs.umd.edu/Projects/sebos/main.shtml`
[9] General Public Licence

Kapitel 3

TCG und Trusted Computing Base

3.1 Organisation

3.1.1 Was ist die TCG?

Die Trusted Computing Group (TCG) versteht sich selbst als Standardisierungsgremium der IT-Branche und besteht aus Hard- und Software-Herstellern die „daran interessiert sind, die Sicherheit der bestehenden IT-Infrastruktur über verschiedene Plattformen und Geräte hinweg zu erhöhen“[1].

Ziele der TCG sind laut eigenen Aussagen unter anderem:

- Entwicklung und Verbreitung eines offenen Industrie-Standards für vertrauenswürdige Computerhardware
- Förderung des so genannten „ubiquous computing“ durch Unterstützung möglichst vieler Plattformen wie z.B. PCs, Server, PDAs oder Handys
- Sicherere Datenhaltung, B2B[2]-Geschäftsbeziehungen und Ecommerce-Transaktionen
- Schutz der Privatsphäre und Recht auf informationelle Selbstbestimmung

3.1.2 Entwicklung und Mitglieder

Die jetzige Trusted Computing Group ist eine Umbildung die aus der TCPA[3] hervorgegangen ist. Diese Neugründung war auf Grund von strukturellen Schwierigkeiten notwendig geworden. So besaß zum Beispiel jedes Mitglied ein Vetorecht, und alle Entscheidungen mussten einstimmig gefällt werden.

An der Liste der Mitglieder hat sich durch die Neuformation jedoch (fast) nichts geändert:

- Die so genannten Promoter-Members, die treibenden Kräfte: AMD, Hewlett-Packard, IBM, Intel, Microsoft, Sony, Sun
- Contributor- und Adopter-Members: ARM, ATI, Dell, Fujitsu Siemens, Infineon, Motorola, Nokia, NVIDIA, RSA Security, Transmeta, VeriSign, VIA, Vodafone, ...

An der weiteren Organisationstruktur (siehe Abb. 3.1) ist besonders auffällig, dass Microsoft – in der breiten Öffentlichkeit oft gleichgesetzt mit den Drahtziehern hinter der TCG - keine direkte Führungsrolle innehat.

[1] Übersetzung von `https://www.trustedcomputinggroup.org/home`
[2] Business-To-Business
[3] Trusted Computing Platform Alliance

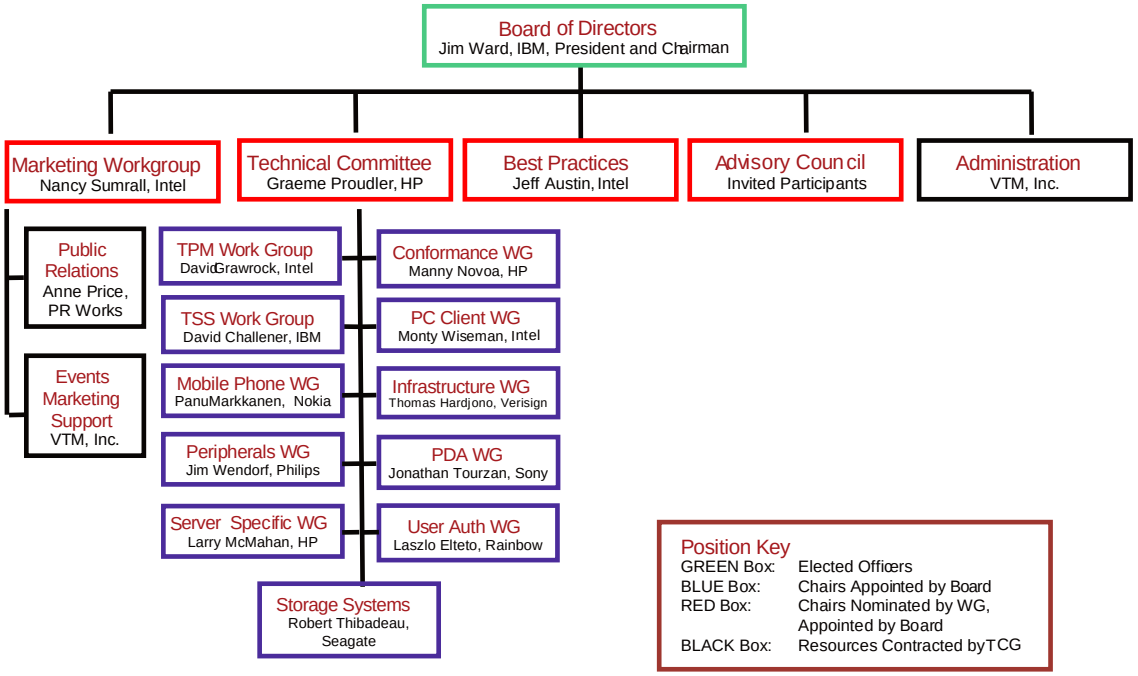

Abbildung 3.1: Organisationsstruktur der TCG

3.2 TCG-Spezifikationen

Im folgenden gehen wir genauer auf die einzelnen Spezifikationen der TCG ein, angefangen mit dem Trusted Platform Module als Grundlage des Gesamtkonzeptes über den Trusted Software Stack als Überbau, wobei wir auch auf häufige Missverständnisse eingehen. Abschließend behandeln wir kurz die Microsoft-Implementation, die noch in einer zu frühen Phase steckt, als dass man allzu Konkretes hierzu schreiben könnte.

3.2.1 TPM-Basisfunktionen

Kernstück der Trusted Computing Base der TCG ist das so gennante TPM[4]. Diese Modul ist eine zusätzliche Hardware-Komponente die – in die Systeme eingebaut - eine ganze Reihe von kryptographischen Funktionen bereitstellt (siehe Abb. 3.2):

- Gesicherter, nicht-flüchtiger Speicher zur Speicherung von Zertifikaten und ähnlichem
- Platform Control Registers (PCR), die beim gesicherten Booten ein Rolle spielen
- Speicherung und Verifizierung – auch multipler - Identitäten
- Kryptographisch starker Zufallszahlengenerator
- Berechnung von SHA-1-Hashwerten
- Verschlüsselung und Entschlüsselung mittels RSA
- Generierung von Schlüsselpaaren für asymmetrische Verschlüsselung
- So gennantes Opt-In, also das bewusste Ein- und Ausschalten von Seiten des Benutzers

Außerdem verfügt der TPM-Chip über Boot-Code, der nach dem Starten des CRTM[5] die Kontrolle übernimmt.

[4] Trusted Platform Module
[5] siehe Abschnitt 3.2.2

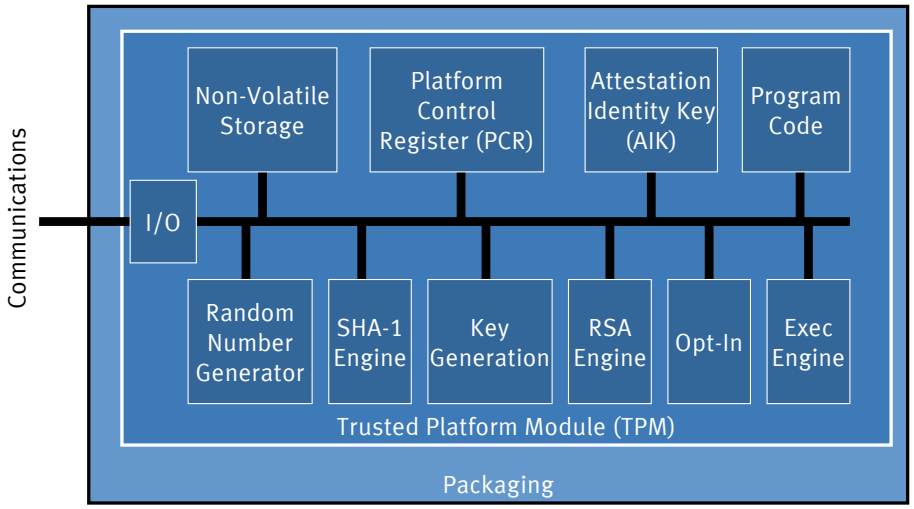

Abbildung 3.2: Blockschaltbild eines Trusted Platform Modules

3.2.2 Funktionen im Detail

Beim Bootvorgang des Systems wird – ebenso wie bei AEGIS/sAEGIS - eine Vertrauenskette initiiert. Diese fängt mit dem Core Root of Trust Measurement (CRTM) an. Diese BIOS-Erweiterung hat die Aufgabe, festzustellen, ob das TPM aktiviert ist. Ist das nicht der Fall, startet der normale Bootprozess, wie bei heutigen Systemen auch. Andernfalls wird der gesicherte Bootvorgang eingeleitet:
Logisch fängt dieser mit dem so genannten Endorsement Key (EK) an. Dieser Schlüssel ist vom Hersteller digital signiert worden (daraus ergibt sich das Endorsement Certificate, welches ebenfalls im TPM gespeichert ist) – alle Daten des Chips sind damit verschlüsselt. Nach dem Willen der TCG sollen EK und das zugehörige Zertifikat den Chip nie verlassen.

Die sichere Speicherung der nicht-flüchtigen Daten übernehmen die Data Integrity Registers (DIR), auf die nur über die definierten Schnittstellen des TPM zugegriffen werden kann. Ein Auslesen von außen soll unmöglich oder zumindest sehr schwierig sein.

Attestation Identity Keys (AIK) dienen der Identitätsüberprüfung – ein TPM-Chip kann mehrere Identitäten speichern, je nachdem, welche Rolle der aktuelle Benutzer gerade innehat - zu deren Erzeugung unterstützt das TPM spezielle Funktionen. Die AIKs besieren auf dem Endorsement Key und sind von einem System auf das Andere migrierbar.
Jeder auf diese Weise gebildeten Identität wird bei deren Erzeugung ein Satz von Konfigurationsdaten zugeordnet, die sich in den Plattform Control Registers (PCR) wiederfinden.

Die PCRs speichern die Hashwerte der einzelnen Systemkomponenten und ermöglichen es dem TPM, beim Systemstart Veränderungen festzustellen.
Im Betrieb des Systems dienen die PCRs dazu, Speicherbereiche zu „versiegeln“.

Weitere Funktionen sind:

Transportsicherheit – Stromverschlüsselung und Hash-Log aller durchgeführten Transaktionen. Der Chip generiert so genannte Nonces, eine Komibnation aus dem aktuellen Zeitstempel und einem Zufallswert.

Revisionssicherheit – Die einzelnen Logs werden über Monotonic Counters miteinander verbunden und sichern so deren Linearität und Integrität.

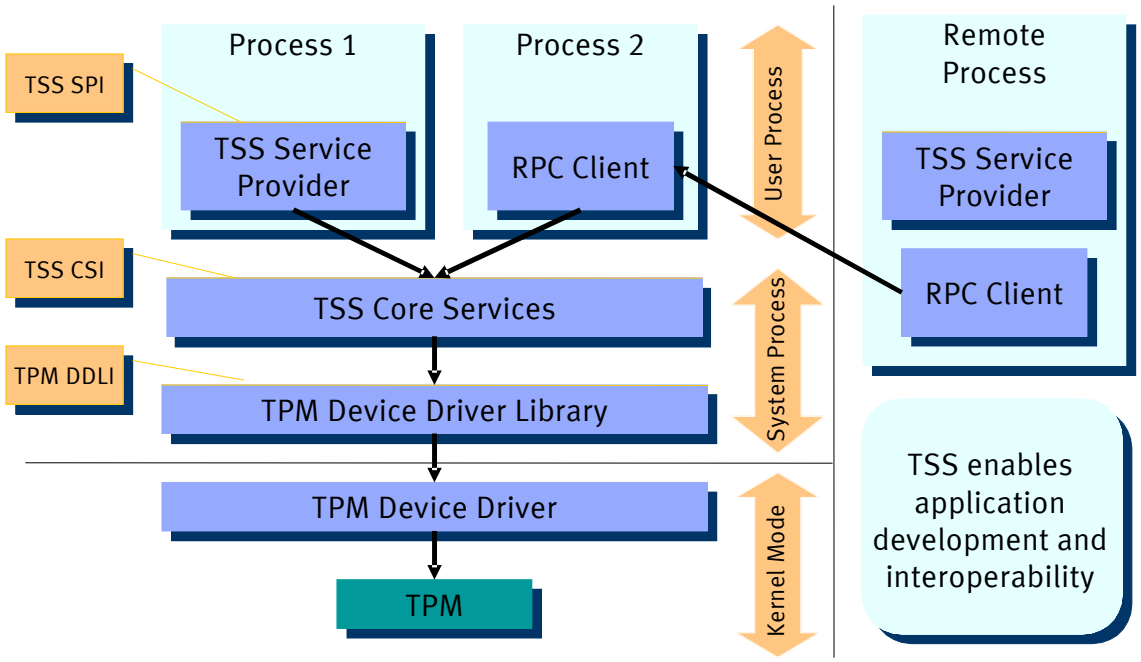

Abbildung 3.3: Die einzelnen TSS-Komponenten

3.2.3 Trusted Software Stack

Der Trusted Software Stack (TSS) stellt die Software-Seite des TCG-Sicherheitskonzepts dar. Die aktuelle Spezifikation ist Revision 1.1 und stellt einen standardisierten Stack für folgende Funktionalität bereit (siehe Abb. 3.3):

TPM Device Driver – Kommunikation mit dem TPM, kapselt dessen Funktionalität und läuft direkt im Kernel-Mode.

TPM Device Driver Library – Stellt das TPM DDLI[6] bereit, dessen Aufgabe im Wesentlichen darin besteht, eine zusätzliche Abstraktionsebene zum Kernelmodul zur Verfügung zu stellen (wohl ein Zugeständnis an Microsoft, die auf diese Weise Ihren TCG-fähigen, „Nexus“ genannten Kernel austauschbar machen können).

TSS Core Services – Stellt high-level Funktionen bereit, also etwa das Versiegeln eines Speicherbereiches oder das Anlegen einer neuen Identität.

TSS Service Provider – Dies ist schließlich die Anwendungsschnittstelle über die sämtliche Schreib-/Lese-Anfragen an den Chip abgewickelt werden. Eine Anwendung fordert nur noch bestimmte Funktionen an und bekommt das Ergebnis. Mit den dazu erforderlichen Zwischenschritten muss sie sich nicht beschäftigen.

Auf alle diese Funtionen wird über das jeweils passende Interface zugegriffen. Dies ist deshalb erwähnenswert, da nur diese Interfaces standardisiert sind – jeder Hersteller von TPMs darf laut Spezifikation die Funktionalität jedes einzelnen Moduls erweitern, um z.B. andere/neue Verschlüsselungsschemata (bei Microsoft ist dies AES) zu unterstützen.

3.3 Missverständnisse

Das TCG-Konzept wird in der Öffentlichkeit viel diskutiert dabei kommt es immer wieder zu einer Reihe von Missverständnissen, die nicht zuletzt auch auf die – gelinde gesagt - eher zögerliche Informationspolitik der TCG zurückzuführen sind:

[6] Device Driver Library Interface

Digital Rights Management ist ausdrücklich kein Ziel der TCG-Bestrebungen. Glaubhaft wird dies durch einen generellen Interessenkonflikt innerhalb der TCG (Bsp. Sony als Plattenfirma auf der einen, Hersteller von CD/DVD-Rohlingen auf den anderen Seite) und dadurch, das sich das TPM jederzeit vom Anwender deaktivieren lassen soll (beispielsweise bei IBM-Notebook einfach möglich).
Außerdem bietet das TPM keinerlei Schutz vor physischer Manipulation, wenn diese in Zukunft auch wohl wesentlich schwieriger sein wird, da es laut TCG-Vorschlag in eine Zwischenschicht auf dem Motherboard eingebettet werden soll.

Keine Kontrollen, Messungen oder Überwachungen Die Überwachung des Benutzers ist weder vorgesehen, noch direkt möglich, da das TPM nur den sicheren Speicher für Maschinen- und Identitätsbezogene Daten bereitstellt.
Das TPM kann außerdem nicht wissen, welche Software gerade läuft.

Benutzer kontrolliert TPM Durch so genanntes Opt-in beim Systemstart durch die Management-Funktionen des TPM ist gewährleistet, dass der Chip deaktiviert werden kann.

Keine Spezifikation bzgl. Betriebssystem, Systemkomponenten, Anwendungen Die TCG-Spezifikationen verzichten bewusst auf die Bezugnahme auf ein spezielles Betriebssystem. Hierbei muss klar sein, dass viele Möglichkeiten (aber auch kritisierte Gefahren) eines TCG-konformen Systems ohne ein sicheres Betriebssystem sehr eingeschränkt sind. Was die Spezifikation festschreibt, sind spezielle Anforderungen die ein Betriebssystem erfüllen muss, um bestimmte Funktionen wahrzunehmen.

3.4 Ausblick

Während bereits IBM-Notebooks seit Anfang 2003 mit TPM-Chips nach Spezifikation 1.1 ausgeliefert werden, arbeiten die Hersteller daran die Version 1.2 in Produkte zu integrieren.
Die wichtigsten Punkte sind hierbei:

- Unterstützung neuer Plattformen (PDAs, Handys, Server) unter Beibehaltung der Abwärtskompatibilität. Unterstützt den Gedanken des „ubiquous computing“.
- Einrichtung Best Practices Group für Datenschutzfragen auf Vorschlag der EU. Diese Gruppe hat z.B. das Rollenbasierte Konzept der Identitäten erarbeitet, dass Organisationen und Benutzer in die Lage versetzt, verschiedenen Sicherheitsanforderungen gerecht zu werden.
- Sichere Bootsequenz (ähnlich. zu Kap. 2) – Version 1.1 der Spezifikation beschränkt sich auf die Reine Datensicherheit (hier im Sinne von Datenverschlüsselung)
- Direct Dynamic Attestation, also das Beglaubigen eines Benutzers/einer Identität ohne ein Trustcenter. Basiert auf einer Zero-Knowledge-Beweistechnik (DirectProof von Intel).

3.5 Microsofts NGSCB

3.5.1 Überblick

Microsofts Implementierung eines Trusted Software Stack nach TCG-Spezifikation trägt den etwas kryptischen Name NGSCB[7], vormals unter „Palladium“ bekannt.
Im Unterschied zur TCG-Architektur versteht sich Microsofts Konzepts als Erweiterung der bestehenden Win32/Win64-Architektur. Dies hat gleich mehrere Vorteile:

- Kompatibilität zu bestehenden Anwendungen; „alte“ Anwendungen nutzen einfach die zusätzliche Funktionalität nicht.

[7] Next Generation Secure Computing Base

- Investitionssicherheit für Anwendungsentwickler, sie müssen nicht auf eine neue Architektur umsteigen, können aber neue Sicherheitsfunktionen nutzen.
- Erhaltung des Marktanteils, der für Microsoft wohl wichtigste Punkt. Man kann es sich nicht erlauben, ein gut eingeführtes Produkt zu gunsten eines völlig neuen und inkompatiblen aus dem Programm zu nehmen.
- Separate Entwicklung der Sicherheitsarchitektur, austauschbarer Sicherheitskernel.

3.5.2 Schlüsselfunktionen

NGSCB bietet vier Schlüsselfunktionen für den sicheren Betrieb des Systems und zum Schutz vor z.B. viralen Aktivitäten oder „Malware“:

Starke Isolation von Prozessen Prozesse sind untereinander und sogar vor schädlichen Aktivitäten aus dem Kernel geschützt. Benötigt Unterstützung vom Prozessor.

Sealed Storage Speichert vertrauliche Informationen, Zugriff erhalten ausschließlich der „Nexus“ und das speichernde Programm.

Sicherer Kommunikationspfad zum Benutzer „Aussperrung“ von Trojanern und Keyloggern durch vertrauenswürdige Ein- und Ausgabe(-geräte).

Nachweisbarkeit und Beglaubigung (Attestation) Flexible Authentifizierung von Hard- und Softwarekomponenten. Nicht autorisierte Veränderungen werden erkannt.

3.5.3 TCG vs. NGSCB

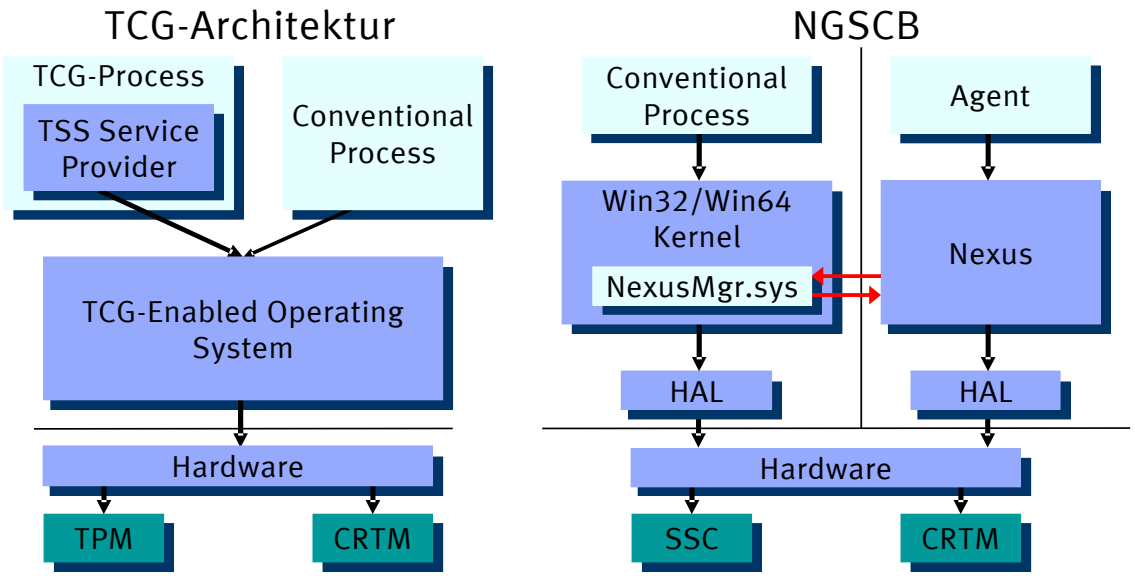

Abbildung 3.4: Die Konzepte der TCG und NGSCB im direkten Vergleich

Ein direkter Vergleich der TCG-Architektur und der Microsoft Implementierung verdeutlicht die zugrunde liegenden Paradigmen. Während die TCG von einem TSS/TPM-fähigen Betriebssystem spricht, ist der Microsoft-Ansatz – aus Kompatibilitätsgründen - zweigeteilt: Zusätzlich zum bestehenden Windows-Systemkern läuft ein zweiter Kernel, der so genannte Nexus. Dieser besitzt eine eigene Speicherverwaltung, eine eigene Hardware-Abstraktionsschicht (HAL) und kann direkt auf die (sichere) Hardware zugreifen.
Die Interprozesskommunikation und die Kommunikation mit dem herkömmlichen Kernel (etwa für den gemeinsamen Scheduler) gibt es einen besonderen Treiber im Windows-Kernel.

Neue Anwendungen – bei Microsoft Agents genannt - laufen im „Nexus-Space“, einer der vier Quadranten der NGSCB-Landkarte. Konventionelle Prozesse können wie bisher im normalen User-Space laufen ohne sich weiter um die – um Sicherheit erweiterten - anderen Komponenten zu kümmern.

Dem TPM der TCG-Spezifikation entspricht die Security Support Component (SSC), die Microsoft als erweiterte Variante der jeweils aktuellen TPM-Spezifikation entspricht.

3.5.4 Ausblick

Da Microsofts NGSCB sich noch nicht einmal im Alpha-Stadium der Entwicklung befindet sind konkrete Aussagen über das finale Produkt schwierig. Festzustehen scheint immerhin folgendes:

- Erstes Release frühestens 2006 mit Windows Codename „Longhorn“. In derzeit im Netz kursierenden Pre-Alpha-Versionen ist von NGSCB allerdings noch nichts zu sehen.
- Nicht weiter verwunderlich ist, dass die neue und sichere Architektur Teil von Microsofts strategischer Planung ist.
  Es ist davon auszugehen, dass man – ähnlich wie seinerzeit mit Windows 2000 - die beiden System-Pfade zusammenführen wird. Irgendwann wird es also nur noch die NGSCB geben.
- Standardmäßig soll der Nexus und alles, was dazu gehört deaktiviert ausgeliefert werden. Genauso liefert IBM derzeit auch Notebooks mit TPM aus.
- Die momentane Planung sieht einen austauschbaren Sicherheitskernel vor. Daraus ergeben sich neue Möglichkeiten für Open Source Projekte oder Institutionen mit besonderen Sicherheitsinteressen (etwa Geheimdienste oder Regierungsbehörden).

Kapitel 4

Java und .Net

4.1 Architektur

Abbildung 4.1 zeigt die bereits bekannte Architektur der beiden System im groben:

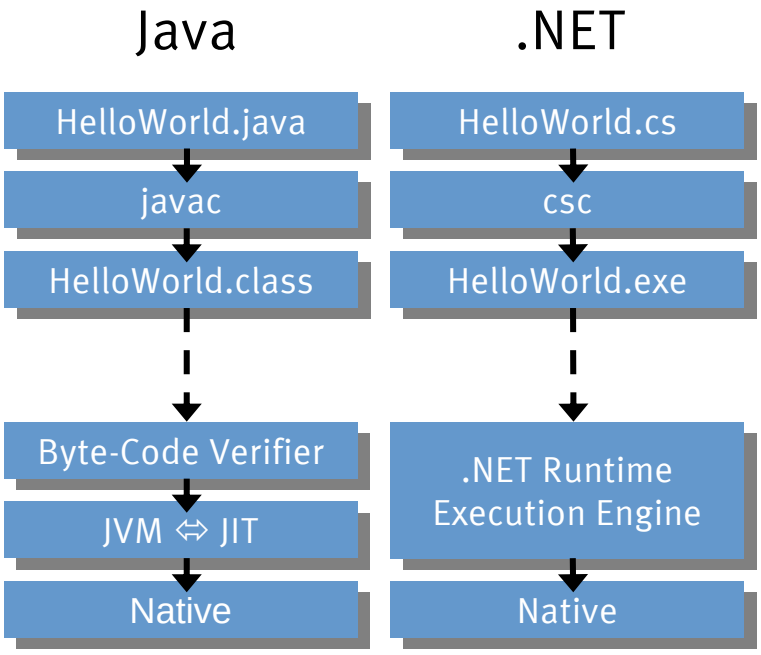

Abbildung 4.1: Grober Überblick über die Architekturen von Java und .NET

1. Quelltext wird geschrieben, Anwendung wird entwickelt.
2. Compiler übersetzt Quelltext in...
3. ...Byte-Code, der auf das...
4. ...Zielsystem transportiert wird (z.B. durch Download).
5. Ein Byte-Code-Verifier überprüft den Byte-Code auf potentielle Sicherheitsrisiken.
6. Maschinen-Code wird zur Laufzeit vom Just-In-Time-Compiler erzeugt und schließlich zur...
7. ...Ausführung gebracht.

Suns Java und Microsofts .NET-Framework sind sich – prinzipbedingt - sehr ähnlich. Daher lassen sich alle in diesem Kapitel gemachten Aussagen ohne weitere Schwierigkeiten auch auf .NET übertragen.

Der wesentliche Unterschied zu Java besteht darin, dass das .NET-Framework nicht nur eine, sondern mittlerweile über 40 Programmiersprachen unterstützt.

Unter anderem sind dies:

- Die von Microsoft unterstützten Sprachen C#, JScript.NET, Visual Basic.NET und Visual J#.NET, sowie C++ mit Managed Extensions
- Die für alte Enterprise-Software wichtige Sprache Cobol.NET z.B. von Fujitsu
- Pascal-basierte Sprachen, wie Delphi.NET von Borland, Oberon.NET und Modula.NET der ETH Zürich
- Viele weitere Programmiersprachen

Zwar gibt es auch Compiler, die beispielsweise C++ nach Java Byte-Code übersetzten; aber diese haben sich bis jetzt noch nicht auf breiter Front durchgesetzt – nicht zuletzt wohl auch deshalb, weil Sun ausschließlich das eigene Java propagiert.
Ein weiterer Unterschied ist, dass bei .NET der Byte-Code-Verifier und der Just-In-Time-Compiler in einer Komponente untergebracht sind.
Häufig verwendeter Code wird in nativen Maschinencode kompiliert und im Global Assembly Cache (GAC) gespeichert, während „normale“ Anwendungen ins Portable-Executable-Format kompiliert werden, was nichts anderes ist als, Byte-Code in einer Standard Windows Exe-Datei zu speichern. Neuere Java-Versionen bieten allerdings einen ähnlichen Caching-Mechanismus.
Zu guter letzt ist das von .NET verwendete Registermaschinen-Modell ein anderes als das von Java. Somit sind – natürlich - auch die Byte-Codes der beiden Architekturen inkompatibel zueinander.

4.2 Java 1.0: Sandboxing

In der ersten Java-Version galt aller lokal installierter Code als implizit vertrauenswürdig, aller aus dem Netzwerk geladener Code dagegen als unsicher und musste daher im so genannten „Sandkasten“ der Virtuellen Maschine laufen.
Bevor jedoch Code in die Sandbox geladen wird, durchläuft diese den Byte-Code-Verifier (siehe Abb. 4.1), um verhindern, dass Buffer-Overflows oder Exploits in der virtuellen Maschine ausgenutzt werden.
Hat der Code die Prüfung bestanden, steht einer Ausführung nur noch die – sehr flexibel ausgelegte - Zugriffskontrolle aus dem Paket `java.security` im Wege, die gleichzeitig das Herz der Java Sicherheitsarchitektur ist.
Ausgeführter Code kann nur über die definierten Schnittstellen der Virtuellen Maschine auf das System zugreifen, wird also von dieser eingekapselt. Besondere Rechte, wie etwa das Lesen und Schreiben auf den Datenträger und dergleichen müssen vom `SecurityManager`-Objekt explizit angefordert werden (hier zum Beispiel über `SecurityManager.checkRead()` und `SecurityManager.checkWrite()`). Der Benutzer kann dann selbst entscheiden, ob er die Aktivität zulässt oder nicht.

Werden keine Rechte angefordert, gelten die Standardeinstellungen für unsicheren Code:

- Es können keine Dateien gelesen, geschrieben, umbenannt oder gelöscht werden. Es kann weder die Länge noch das Änderungsdatum einer Datei abgefragt werden, noch überhaupt festgestellt werden, ob eine Datei existiert. Entsprechend ist auch kein Zugriff auf Verzeichnisse möglich.
- Es können keine Verbindungen zu anderen Computern aufgebaut werden, als zu dem, von dem der Code heruntergeladen wurde und solche Verbindungen auch nicht angenommen werden. Es können keine privilegierten Port verwendet werden (also Ports bis 1024).
- Nicht-vertrauenswürdiger Code kann keine Systemfunktionen wie das Laden einer nativen Bibliothek, das Starten eines neuen Prozesses oder das Beenden des Java-Interpreters ausführen. Er kann keine Threads oder Threadgruppen manipulieren, wenn diese nicht vorher selbst erzeugt

wurden. Ab Java 1.1 kann das Reflection API nicht verwendet werden, um Informationen über die nicht-öffentlichen Member einer Klasse zu bekommen, sofern diese Klasse nicht vom Code selbst heruntergeladen wurde.

- Bestimmte Grafik- und GUI-Funktionen sind nicht benutzbar. Es kann kein Druckauftrag abgesetzt und nicht auf die Zwischenablage zugegriffen werden. Außerdem zeigen alle erzeugten Fenster deutlich an, dass sie „unsicher“ sind, damit nicht vorgetäuscht werden kann, eine andere Anwendung zu sein.

4.3 Java 1.1: Digital signierte Klassen

Java 1.1 behält das Sandbox-Modell von Java 1.0 bei, fügt aber dem Paket `java.security` neue Funktionen für digitale Signaturen hinzu. Damit können Java-Klassen digital signiert und Java-Installationen so konfiguriert werden, dass sie heruntergeladenem Code dann vertrauen, wenn dieser eine gültige digitale Signatur eines vertrauenswürdigen Absenders trägt. Solcher Code wird dann behandelt, als wäre er lokal installiert, und bekommt damit vollen Zugriff auf die Java-APIs. In dieser Version verwaltet das Programm `javakey` die Schlüssel und signiert JAR-Dateien mit Java-Code.
Trotz dieser Erweiterungen wird das bewährte Modell beibehalten: Vertrauenswürdiger Code bekommt vollen Zugriff, nicht-vertrauenswürdiger Code unterliegt sämtlichen Einschränkungen der Sandbox.

4.4 Java 1.2: Policies

Mit Java 1.2 werden umfangreiche neue Funktionen zur Zugriffskontrolle in die Sicherheitsarchitektur von Java eingeführt. Wiederum wurden neue Klassen in `java.security` übernommen.
Die Klasse `Policy` ist eine der wichtigsten: Sie definiert eine Sicherheits-Policy in Java. Ein `Policy`-Objekt bildet `CodeSource`-Objekte auf zugehörige Mengen von `Permission`-Objekten ab. Ein `CodeSource`-Objekt repräsentiert die Herkunft eines „Stückchens“ Java-Code – hierzu gehört auch der URL der Klassendatei (der auch eine lokale Datei bezeichnen kann), sowie eine Liste derjenigen Entitäten, die die Klassendatei signiert haben.
`Permission`-Objekte, die im `Policy`-Objekt mit einem `CodeSource`-Objekt verbunden sind, definieren die Zugriffsrechte, die Code aus einer bestimmten Quelle gewährt werden. Von `Permission` abgeleitete Klassen repräsentieren verschiedene Zugriffsrechte.

Mit diesem neuen Modell ist der `SecurityManager` immer noch die zentrale Klasse. Anfragen an bestimmte Zugriffsrechte werden jetzt allerdings an eine Klasse `AccessController` delegiert, die die Rechte auf Basis der `Permission`/`Policy`-Architektur vergibt oder verweigert.
Die Zugriffskontrolle von Java 1.2 (und höheren Versionen) hat also folgende wichtige Merkmale:

- Code verschiedener Herkunft kann verschiedene Zugriffsrechte haben. Es werden nun also feinere Zugriffsabstufungen unterstützt. Insbesondere kann jetzt auch lokal installierter Code als unsicher oder nur teilweise vertrauenswürdig deklariert werden. Damit besitzen nur die Systemklassen vollen Zugriff.
- Policies können vom Administrator mit dem Programm `policytool` konfiguriert werden.
- Die neue Architektur ist nicht auf eine fest vorgegebene Menge an Zugriffsrechten beschränkt. Neue, von `Permission` abgeleitete Klasse können problemlos definiert werden, um neuen Anforderungen gerecht zu werden.

Ein Beispiel für die Zugriffskontrolle in Java

Angenommen, ein Applet möchte die Datei `/etc/passwd` lesen.
Zunächst ruft das Applet den Kontruktor der `FileInputStream`-Klasse auf. Dieser ruft nun die Methode `java.lang.Security.checkRead()` auf.

Deren Default-Verhalten sieht so aus: Es wird ein neues `FilePermission`-Objekt erzeugt, das den gewünschten Zugriff repräsentiert. Das Objekt hat das Ziel (`target`) „`/etc/passwd`“ und die Aktion (`action`) „read“. Dann wird das `FilePermission`-Objekt an die statische Methode `checkPermission()` der Klasse `java.lang.AccessController` übergeben.
Diese Klasse leistet nun die eigentliche Arbeit der Zugriffskontrolle. Die Methode bestimmt für jede aufrufende Methode die `CodeSource` und verwendet das aktuelle `Policy`-Objekt, um die `Permission`-Objekte zu bestimmen, die dazu gehören. Mit diesen Informationen kann der `AccessController` bestimmen, ob der lesende Zugriff auf `/etc/passwd` zulässig ist (bei den allermeisten Systemen sicherlich nicht).

4.5 Schlussfolgerungen für SDSD-System

Die abgesicherte Umgebung der Java-Sandbox bietet hinreichende Sicherheit. Eine Verbindung mit AEGIS/sAEGIS erscheint jedoch sinnvoll, da man auf der Grundlage eines sicheren Systemstarts dann das Signatur- und Zerfizierungs-Konzept von Java aufsetzen kann. Dafür ist es erforderlich das gesamte SDSD-System digital zu signieren.

Zusätzliche Sicherheit erreicht man – vom Design her – durch Implementierung mit möglichst niedrigen Zugriffsrechten.
Hierzu passt dann auch die Ableitung eigener `Permission`-Klassen etwa

- `issi.sdsd.AdministrationPermission` (Recht, das System zu administrieren) oder
- `issi.sdsd.TokenPermission` (grundlegendes Recht, das Security-Token anzunehmen)

in einem gedachten `issi\sdsd`-Paket. Ferner macht es Sinn, die einzelnen Rechner des Systems mit eigenen Policies auszustatten, um fremdem Code nicht blind zu vertrauen.

Für weiter gehende Erwägungen siehe das nun folgende Kap. 5.

Kapitel 5

Anwendungsfälle im SDSD-System

5.1 Ideale Umgebung

Im Rahmen einer idealen Umgebung (also ein komplett sicheres System) wird das SDSD-System so ausgeführt, wie von uns vorgesehen. Es kann zur Laufzeit nicht manipuliert werden und Angriffe werden erkannt und abgewehrt.

Um den gesicherten Start unseres SDSD-Systems zu gewährleisten, sind folgende Überprüfungen/Regelungen notwendig:

- SDSD-System wird beim Start auf (böswillige) Veränderungen des Codes überprüft. Bei Veränderungen wird der Start verweigert bzw. dem Nutzer eine Warnmeldung angezeigt, die bestätigt werden muss.
- Die Prozesse, die den SDSD-Code ausführen, müssen isoliert werden („Nexus“). Ebenso die nebenläufigen Prozesse, die Zugang zu SDSD-Code haben.
  Speicherbereiche, in denen der SDSD-Code ausgeführt wird, müssen ebenfalls vor unberechtigtem Zugriff geschützt werden.
- Das zu kontrollierende Objekt darf nur über das SDSD-System angesprochen werden – anderen Prozessen muss der Zugriff auf das Objekt verweigert werden. Dafür ist eine Kapselung durch das SDSD-System notwendig.

Diese Bedingungen werden durch ein sicheres Betriebssystem und dem SDSD-System sichergestellt. Natürlich setzt dieses ein sicheres System voraus, welches mit den Mitteln des Personal Secure Booting sowie den TCG-Spezifikationen geschaffen werden kann.

5.2 Störfaktoren und deren Lösungen

Um alle Störfaktoren auszuschalten, müssen wir ein sicheres System zugrunde legen. Damit das System sicher ist, muss die folgende Kausalitätskette erfüllt sein:

1. Sicheres Betriebssystem. Setzt voraus:
2. Sicheres Booten des Betriebssystems. Setzt voraus:
3. Sichere Hardware. Setzt voraus:
4. Sicherer Bootvorgang. Setzt voraus:
5. Sicheres Bios.

Um die Kausalitätskette zu erfüllen, benötigt man von Anfang an ein sicheres System. Dieses fängt beim manipulationssichern BIOS an und geht weiter mit dem sicheren Booten. Diese beiden Prozesse

können zum einen über das Personal Secure Booting oder über die TPM Spezifikation 1.2 sichergestellt werden.

Ebenso verhält es sich mit dem gesicherten Start des Betriebssystems. Hier bietet das Personal Secure Booting nur den gesicherten Start des Systems, während die TPM Spezifikation das System auch zur Laufzeit sichert.

Die Manipulationssicherheit zur Laufzeit wird dann schließlich durch die vorgestellten Techniken der TCG realisiert. Bis jetzt bezogen sich alle vorgestellten Maßnahmen auf ein Einzelplatzsystem. Im

Netzwerk sind jedoch auch noch andere Sicherheitskontrollen möglich. So können sich z.B. die Rechner untereinander Verifizieren bzw. das zweite System erkennt, wenn am Ersten etwas geändert worden ist. Solche Kontrollen sind auch per Hashwert durchführbar, so dass wir hier nicht unbedingt auf Personal Secure Booting oder TCG zurückgreifen müssen. Natürlich ist diese Absicherung nicht so manipulationssicher wie die beschriebenen Verfahren, aber schon ein guter Ausgangspunkt, bis die Sicherheitssysteme am Markt verfügbar sind.

5.3 Fazit

Letztendlich kann man sagen, dass der Idealfall noch Illusion ist. Weder Personal Secure Booting noch die TCG-Umsetzung ist am Markt verfügbar.
Jedoch werden mit den beiden Verfahren gute Voraussetzungen für sichere Systeme geschaffen. Um die das SDSD-System bei Verfügbarkeit von TCG konform deren Richtlinien zu gestalten, sind nach jetzigem Stand nur geringe Änderungen erforderlich.
Jedoch sollte man natürlich die Entwicklung von eigenen Sicherheitsmaßnahmen nicht vernachlässigen.
Die Kontrolle im Netzwerk über Hashwerte ist z.B. ohne die vorgestellten Sicherheitsmethoden anwendbar. Zwar bietet diese nicht die Sicherheit wie die Sicherheitssysteme, aber es wird damit eine gute Basis geschaffen, auf der man aufbauen kann.

Je mehr von Sicherheitsmaßnahmen später am Markt verfügbar sind, desto sicherer können wir das SDSD-System gestalten.

Kapitel 6

Schlussbetrachtung

Abschließend können wir sagen, dass das TPM 1.1 schon in einigen Systemen (erfolgreich) im Einsatz ist.
IBM zum Beispiel liefert Notebooks sowie die NetVista PC-Serie mit einem TPM-Chip aus. Dieser ist im BIOS standardmäßig ausgeschaltet. So folgt man auch hier nach wie vor dem Grundsatz der TCG, dem Benutzer die Kontrolle zu überlassen, ob er den TPM-Chip nutzen möchte oder nicht.
Wenn man sich dafür entscheidet, wird im momentanen Entwicklungsstadium nur die reine Datensicherheit gewährleistet. Durch das TPM werden die Daten an die Hardware des Gerätes gebunden. Entscheidender Nachteil ist hierbei, dass bei einer Zerstörung des TPM-Chips auch die Daten verloren sind.
Damit das TPM z.B. unter Windows XP läuft, muss dort der Login-Manager ausgetauscht werden, was jedoch nur einen kleinen Systemeingriff darstellt (Netzwerke auf Basis von Novell Netware benötigen eine ähnliche Modifikation).
Um weiter gehende Sicherheit à la TCG zu bekommen, benötigt man die Revision 1.2 des TPM-Chips, welche sich bereits in Planung befindet und bald am Markt verfügbar sein wird.
Desweiteren soll bald SEBOS unter der GPL erscheinen. Im Zusammenspiel mit den Vorgaben der TCG würde sich in der Kombination ein sicheres Gesamtsystem ergeben.
Zwar beinhalten auch die Whitepaper der TCG einen gesicherten Bootvorgang, doch bis dato sind dazu keine Details bekannt, so dass das SEBOS eine gute, verfügbare und freie Alternative darstellt. Ein weiterer Vorteil ist, dass im Gegensatz zum TCG-Modell hierfür keine neue Hardware benötigt wird.

In diesem Zusammenhang sei noch erwähnt, falls ein SEBOS-Release im Zeitraum unserer Projektarbeit erfolgt, eine Einbindung dieser Technik in unser System eventuell möglich wäre. Entsprechende Szenarien wurden in den vorangegangenen Kapitel erläutert.

Ein Kritikpunkt an der TCG ist die mangelhafte Informationspolitik. In der breiten Masse gilt die TCG als ein Synonym für Microsoft und dem damit verbundenem Ruf, den Nutzer nur überwachen und kontrollieren zu wollen.
Doch das Gegenteil ist der Fall: Microsoft hat in der TCG keine führende Position, die TCG möchte den Nutzer auch nicht „gängeln“ oder bevormunden; sie möchte ihm mächtige Werkzeuge in die Hand geben, mit denen ein beliebiges Betriebssystem gegen Manipulation abgesichert werden kann.
Das DRM ist ausdrücklich kein Ziel der TCG.
Ein großes Fragezeichen bilden die so genannten Trustcenter. Sind diese „vertrauenswürdig“? Kann bzw. will der Benutzer diesen Vertrauen? Wer steckt hinter den Trustcentern?
Hier werden gesetzliche Rahmenbedingungen sinnvoll, damit es bei den Trustcentern nicht zu Interessenkonflikten kommt, und die ausgegebenen Zertifikate auch wirklich vertrauenswürdig sind.

Nach wie vor ist der größte Schwachpunkt am System der Benutzer. Zum einen will er/sie ein einfach zu bedienendes System mit maximaler Sicherheit – dies ist leider *so nicht möglich*. Auch die Gewohn-

heit der Benutzer, bei Passwörtern bekannte Daten (wie z.B. Geburtsdaten, Name(n) der Kinder, Autokennzeichen etc.) zu wählen, ist ein großes Sicherheitsrisiko.

Glossar

Dramatis Personae (nach [Sch96])
Personen in der Kryptographie

Alice	Erster legitimer Teilnehmer in Protokollen, Sender
Bob	Zweiter legitimer Teilnehmer in Protokollen, Empfänger
Carol	Teilnehmer in Protokollen mit drei oder vier Parteien
Dave	Teilnehmer in Protokollen mit vier Parteien
Eve	Abhörer
Mallory	Böswilliger Angreifer
Trent	Vertrauenswürdiger Vermittler
Walter	„Wächter“, schützt Alice und Bob in einigen Protokollen
Peggy	Prüfer
Victor	Verifizierer

Begriffe und Abkürzungen

AES	Advanced Encryption Standard – Symmetrischer Verschlüsselungsalgorithmus mit einer Blocklänge von 128 Bit, Nachfolgestandard von DES
AIK	Attestation Identity Key – Kryptographiscer Schlüssel zur Identitätsüberprüfung mit dem TPM
Backdoor	Nicht dokumentierte Zugangsmöglichkeit (über ein Computernetz) zu einem PC
BIOS	Basic Input/Output System – Schnittstelle zwischen Hard- und Software
Buffer Overflow	Ungeprüfter Puffer bei Kommunikation, führt meist zum Überschreiben des Stacks
CRTM	Core Root of Trust Measurement – TCG BIOS-Erweiterung, stellt fest ob das TPM aktiv ist und startet im positiven Fall die gesicherte Bootsequenz
DDA	Direct Dynamic Attestation – Auf der Zero-Knowledge-Beweistechnik aufbauendes Verfahren zur Indentitätsbeglaubingung ohne Trustcenter
DRM	Digital Rights Management – Verfahren mit dem die Urheberrechte (vor allem an Filmen und Tonaufnahmen, aber auch an Software) auf elektronischen Datenverarbeitungsanlagen gewahrt und Raubkopien verhindert werden sollen
EAL	Evaluation Assurance Level – Sicherheitsstufe der Common Criteria
Exploit	Angriff auf ein Computersystem unter Ausnutzung einer Sicherheitslücke der darauf installierten Software
GRUB	Grand Unified Bootloader – Open Source Bootloader
HAL	Hardware Abstraction Layer – Zusätzliche Abstraktionsebene unterhalb eines Betriebssystemkerns
MAC	Message Authentication Code – Hashwert einer Nachricht
Malware	Zusammenziehung aus „malicious“ (bösartig) und „Software“ zur übergreifenden Bezeichnung für störende und/oder schädliche Programme und Viren
MD5	Message-Digest Algorithm 5 – Einweg-Hash-Funktion die einen 128 Bit Hashwert einer Nachricht berechnet, gilt als nicht mehr sicher
Nexus	Sicherer Betriebssystemkern der NGSCB
NGSCB	Next Generation Secure Computing Base – von Microsoft geplante TCG-konforme Implementation des TSS

Begriffe und Abkürzungen

Nonce	Kunstwort aus „number“ und „once“; Zufallszahl oder aktueller Zeitstempel oder Kombination aus Beidem
POST	Power On Self Test – Selbsttest des Systems vor Auführung des BIOS
RSA	Rivest, Shamir und Adleman Algorithmus – Weit verbreitetes asymmetrisches Verschlüsselungsschema für Public Key Kryptographie
Sandboxing	Einkapseln von Software in eine virtualisierte Umgebung; Zugriffe auf das zugrunde liegende System erfolgen ausschließlich indirekt
Security through obscurity	Sicherheit durch Geheimhaltung von Implementierungsdetails (eines Protokolls/eines Algorithmus/einer Software)
SDSD-System	State Dependend Security Decison System – Sicherheitssystem das auf endlichen Automaten basiert und kontextabhängige Sicherheitsentscheidungen ermöglicht
SEBOS	Security Enhanced Bootloader for Operating Systems – Sicherer Open Source Bootloader, basiert auf GRUB und LinuxBIOS
SHA-1	Secure Hash Algorithm 1 – Einweg-Hash-Funktion die einen 160 Bit Hashwert einer Nachricht mit einer maximalen Länge von 2^{64} Bit (ca. 2 Milliarden GB) berechnet
Social Engineering	Versuch eine Person zu beeinflussen und sie dazu zu bringen, sensible Informationen (zum Beispiel ein Passwort) preiszugeben
SSC	Security Support Component – TPM-Chip der NGSCB, entspricht momentan TPM 1.2
TCG	Trusted Computing Group – Industriezusammenschluss von Hard- und Software-Herstellern
TCPA	Trusted Computer Platform Alliance – siehe auch TCG
TPM	Trusted Platform Module – sicherer Speicher für Maschinenbezogene Daten zusammen mit grundlegenden Kryptographiefunktionen
Trojaner	besser: „Trojanisches Pferd“ – Programm, das oberflächlich nützliche Funktion anbietet, im Hintergrund jedoch zerstörerische oder schädigende Aktivitäten ausführt
TSS	Trusted Software Stack – von der TCG vorgeschlagene Softwarearchitektur
VM	Virtual Machine – Laufzeitumgebung beim Sandboxing

Literaturverzeichnis

[Arb99] W. Arbaugh, *Chaining layered integrity checks*, 1999

[BLP03] Biskup, Leineweber, Parthe, *Administration Rights in the SDSD-System*, 2003, Universität Dortmund

[Eck03] Claudia Eckert, *IT-Sicherheit: Konzepte - Verfahren - Protokolle*, 2. Aufl., Oldenbourg, München, Wien, 2003

[ELM+03] Paul England, Butler Lampson, John Manfredelli, Marcus Peinado and Bryan Willman, *A trusted open platform*, IEEE Computer 36 (2203), no. 7, 55-62

[EU04] Die Gruppe für den Schutz der Rechte von Personen bei der Verarbeitung personenbezogener Daten (Europäische Union), *Arbeitspapier über vertrauenswürdige Rechnerplattformen und insbesondere die Tätigkeit der Trusted Computing Group (TCG)*, Brüssel, 23.01.2004

[Gol99] Dieter Gollmann, *Computer Security*, John Wiley & Sons Ltd., Chichester, 1999

[Him03] Gerald Himmelein, *Blick ins Schloss - Details zu Palladium/NGSCB*, c´t 12/2003, S. 192 ff.

[IAPR01] Naomraru Itoi, W. Arbaugh, Samuela J. Pollack and Daniel M. Reeves, *Personal Secure Booting*, Lecture Notes in Computer Science 2119 (2001)

[LBO] LinuxBIOS *http://www.linuxbios.org/*

[Sch96] Bruce Schneier, *Applied Cryptography*, 2nd Edition, John Wiley & Sons, Chichester, 1996

[SST03] Ahmad-Reza Sadeghi, Christian Stüble, *Vertrauen ist gut - Sinn und Unsinn von TCPA und Palladium*, c´t 13/2003, S. 232 ff.

[TCG03] Trusted Computing Group, *https://www.trustedcomputinggroup.org*, *RSA Presentation Final (2003), TCG Organization (2003), TCG Backgrounder (2003)*