Introduction to Security - Final Exam

eine Zusammenfassung der Kapitel 10-20 von ri_ba und AllesBeimAlten

WS11

10 Trusted Computer und Multilevel Security

10.1 Formale Modelle für Computer Security

Zwei Fakten gibt es bezüglich Computer Security- erstens: Jedes komplexe System hat Schwachstellen, die behoben werden müssen und zweitens: es ist extrem schwierig, wenn nicht unmöglich ein System zu entwerfen, dass (zumindest) nahezu „unverletzbar“ ist. Bedingt dadurch existiert natürlich der Wunsch formale Modelle zu erstellen um Security Designs und Implementationen zu überprüfen. Intiatoren: U.S. Verteidigungsministerium. Mit steigernder Komplexität der Systeme hat diese Bestrebung jedoch abgenommen, trotzdem liefern diese Modelle immernoch wertvolle Prinzipien und Herangehensweisen.

10.2 Bell-LaPadula-Modell

Das Bell-LaPadula-Modell ist ein formales Modell für Zugriffskontrolle. In dem Modell ist jedes Subjekt und Objekt einer Security Klasse zugewiesen, die im simpelsten Fall als hierarchische Strukturen organisiert sind. Objekte haben Security-Level und Subjekte Security-Clearance-Level. (Wie beim Militär, man Gedenke dem Stoff vom Mid-Term Exam). Die Klassen bestimmen außerdem wie ein Subjekt auf ein Objekt zugreifen darf, original waren 4 (rwx und „append“) Zugriffsarten vorgesehen. Das Konzept ist auch auf andere Bereiche anwendbar, wo Information in grobe Kategorien unterteilt werden kann und Benutzern Rechte gewährt werden, um auf bestimmte Kategorien zuzugreifen.

10.2.1 Exkurs: Multi-Level-Security (Bild)

Sind mehrere Security-Level definiert (Bell-LaPadula-Modell z.B.) so spricht man (logischerweise) von Multi-Level-Security. Multi-Level-Security fordert, dass ein High-Level Subjekt keine Informationen an ein Lower-Level (oder einfach unvergleichbares) Subjekt weitergibt, es sei denn dieser „Datenfluss“ ist von einem autorisierten Benutzer so vorgesehen. Hierbei existieren folgende Eigenschaften:

- Kein aufwärtslesen (Man darf nur gleiche bzw. untere Level lesen) (simple security-property, ss-p)
- Kein abwärtsschreiben (Man darf nur auf gleiche bzw. obere Level schreiben) (star-property, *-p)
- Ein Subjekt kann einem anderen Subjekte Rechte aus seinem Rechte-Satz geben, jedoch unter Beachtung von ss-p und *-p. (ds-property)

Sind Punkt 1 und 2 erfüllt spricht man von Mandatory Acces Control (MAC). Ist Punkt 3 zusätzlich erfüllt so handelt es sich um Discretionary Access Control (DAC). BLP ist DAC.

10.2.2 Formale Beschreibung von BLP

BLP basiert auf den aktuellen Systemzustand beschrieben durch das Quadrupel (Zugriffs-Set b, Zugriffs-Matrix m, Level-Funktion f, Hierarchie h). Für jedes Objekt Si und Oj gelten die DAC Eigenschaften.

- ss-p für jedes Trippel (Si, Oj, read) im Zugriffs-Set b gilt f_c(Si) $\geq$ f_o(Oj)
- *-p für jedes Trippel (Si, Oj, append) in b gilt f_c(Si) $\leq$ f_o(Oj) und für jedes Trippel (Si, Oj, write) in b gilt f_c(Si) = f_o(Oj)
- ds-p: (Si, Oj, Ax) $\Rightarrow$ Ax $\in$ M[Si, Oj]

Durch die formalen Theoreme lässt sich zeigen, dass BLP in der Theorie sicher ist, in der Praxis ist dies aber meist unmöglich.

10.2.3 BLP Regeln (Auszug)

- Get access: Fügt ein Trippel (Si, Oj, Zugriffs-Modus) zum derzeitigen Zugriffs-Set hinzu. (Initiiert Zugriff)
- Release access: Entfernt Trippel (Si, Oj, Zugriffs-Modus) vom derzeitigen Zugriffs-Set. (Beendet Zugriff)
- Change Object Level: Verändert den Wert von f_o(Oj) von einem Objekt Oj
- Change Current Level: Selbiges für Subjekt Si.

Mehr Regeln auf der entsprechenden Folie.

Nun folgt ein sehr spezifisches BLP Beispiel über mehrere Folien, glaube nicht dass es direkt für Prüfung wichtig ist, aber durchlesen ist zum Verständnis dennoch empfohlen. Ich werde es hier nicht genauer behandeln.

10.3 Multics Beispiel (BILD)

Hier ist eigentlich nur interessant, dass es in den 1960ern schon eine Implementierung von Multi-Level-Security bei einem Multics OS gab, genannt Projekt MAC (multiple-acces Computers). In Multics wird jede Art von Speicher per Segmente angesprochen, hierbei gibt es sowohl für Files als auch für Prozesse „Descriptor"-Segmente, die unter anderem Informationen bezüglich dem Zugriffsschutz beinhalten. Diese wurden in der Form von rwx-Bits, die individuell gesetzt werden konnten, abgeleitet von der ACL (die durch den Zugreifer bestimmt wird) des Segments. Außerdem gabs eine Process-Level Tabelle die Security-Clearences jedes Prozesses beinhielt und jedes Segment war mit einem Security-Level versehen, dass im entsprechenden Parent-Verzeichnis definiert wurde. Folie ist mir irgendwie nicht ganz verständlich. Bei Interesse einfach das Bild selbst ansehen.

10.4 BIBA 77

Das BIBA77-Modell beschäftigt sich hauptsächlich mit Integrität. Die Basics von BIBA sind ähnlich dem BLP Modell. Jedes Subjekt und Objekt hier hat ein Integritäts-Level I(Si) bzw. I(Oj). Es gibt eine bestimmte Hierarchische Struktur, in welche es eine strikte Ordnung von hohem zu tiefem Level gibt. Es gelten folgende Regeln:

- Simple Integrität: Subjekte können nur Objekte mit gleichen oder tieferen Integritäts-Level verändern.
- Integritäts Beschränkung: Ein Subjekt kann auf ein Objekt nur dann lesen zugreifen, wenn dessen Integritäts-Level gleich oder höher ist.
- Aufrufs-Eigenschaft: Ein Subjekt kann ein anderes Subjekt nur dann Aufrufen, wenn das Integriertätslevel des Aufrufes größer gleich des aufzurufenden Objekts ist.

Dieser simple Write-Up Verbot verhindert die Kontamination von High-Level Daten.

10.5 Clark-Wilson Integritäts-Modell (BILD

Praktischer, eher auf kommerziellen als auf militärischen Einsatz abgezielt. Beim CWM wird Zertifizierung unter der Beachtung der Sicherheitsrichtlinien von eine Art „Sicherheitsoffizier" durchgeführt, wobei die Ausführung (enforcment) selbst vom System übernommen wird. Beteiligte Elemente: Un- und Eingeschränktes Daten-Items (U/CDI), Integritäts-Verfikation Prozeduren (IVP), Transformations-Prozeduren (TP) und natürlich Benutzer. Es gelten folgende Regeln (Auszug):

- Alle IVPs müssen sicherstellen, dass alle CDIs sich in einem gültigen Zustand befinden während eine IVP ausgeführt wird.
- Alle TPs müssen zertifizier sein gültig zu sein.
- Das System müssen die Identität jedes Nutzers authentifizieren, der versucht eine TP auszuführen.

Komisches Ding, mehr dazu am Bild.

10.6 Das Chinesische-Mauer Modell (BILD)

Hier werden bestimmte Objekte einfach in „Conflict-of-Interest"-Gruppen zusammengefasst, in denen es Unterschiedliche DatenSÄTZE gibt. Greift ein Subjekt auf ein Objekt eines Datensatzes in einer CI-Gruppe zu, baut sich eine Mauer zwischen den Elementen des zugegriffenen Datensatzes und allen anderen Element der CI auf. Somit werden die Zugriffsrechte eines Benutzers von den Informationen die er bereits besitzt bestimmt. „ss-p" hier: Si kann auf Oj nur zugreifen, falls Oj in einem Datensatz einer CI ist, den er bereits besitzt, oder sich in einer CI befindet, auf der er noch überhaupt nicht zugegriffen hat; „*-p" hier: Si kann auf Oj nur schreiben wenn Si auf Oj nach ss-p zugreifen kann und generell alle Oj über die Si verfügt kann im selben Datensatz wie Oj sind.

10.7 Referenz-Monitore (BILD)

Der Referenz-Monitor ist ein Kontroll-Element in der Hardware und im OS eines Computers. Es regelt Zugriffe von Subjekten auf Objekte über dessen Security Parameter. Der Monitor hat Zugriff auf die Kernel Database, das ist eine Liste an Privilegien die jedes Subjekt hat und Schutzattributen über die jedes Objekt verfügt. Der Referenz-Monitor sorgt für die Ausführung der Security Regeln. Jede Form von Zugriff wird überwacht (Complete mediation), Referenz-Monitor und Kernel Database sind geschützt vor jeder Art unerlaubten Zugriffs (Isolation) und die korrekte Funktionalität des Referenz-Monitors muss beweisbar sein (Verifizierbarkeit).

10.8 Verteidigung gegen Trojaner (BILD)

Geschützte Systeme die z.B. die *-Property unterstützen sind hiergegen ziemlich sicher. Das zeigt ein nettes Beispiel wo Jane versucht John mit einem Trojaner auszuspionieren und der Angriff wegen des Zugriffs-Regelwerk (*-p) nicht aufgeht. Ansehen bei Interesse.

10.9 MLS für Rollen-Basierte Zugriffskontrolle

RBAC kann benutzt werden um BLP MLS-Regeln zu implementieren, hierbei muss folgendes umgesetzt werden:

- Sicherheits-Beschränkungen auf Benutzer, für jeden Benutzer u von U wird eine Sec-Clearance L(u) festgelegt.
- Jeder Permission wird mit einer Read oder Write-Erlaubnis auf ein bestimmtes Objekt O verbunden, sodass jedes Objekt eine Read- und eine Write-Erlaubnis hat. Alle Objekte haben damit eine Security-Klassifikation. (Beschränkung der Objekte)
- Defintion des read-level und write-level einer Rolle (Rolle read/write und mit mehren Objekten verbunden, geringste obere Schranke von Objekten = read-level, größte untere Schranke von Objekten = write-level)
- Jede Rolle r hat nun ein defniertes write/read-level. Für jedes User Assignment, muss die clearance des User das r-lvel der Rolle dominieren und vom w-level der Rolle dominiert werden. (Beschränkung der User-Assignments)

SSP: r-level von Rolle = höchstes r-level der referenzierten Objekte, Benutzer darf Rolle nur zugewiesen kriegen, wenn seine Security-Clearance mindestens so hoch ist, wie das r-Level (NO READ UP)
-P: w-level von Rolle = tiefstes w-level der referenzierten Objekte, Benutzer darf Rolle nur zugewiesen kriegen, wenn seine Security-Clearance nicht höher ist, als das w-Level (NO WRITE DOWN)

10.10 MLS bei Datenbank-Security

MLS bei Datenbanken erhöhen die Komplexität der Zugriffskontrolle und des Designs der Datenbank selbst. Ein Problemfeld ist die Granularität der Sicherheits-Klassifkation:

- Komplette Datenbank: Einfachste Herangehensweise, eine ganze Datenbank kann als Sicher/Vertrauenswürdig klassifizert werden (z.B. Personal-Datenbank) bei der Arbeit mit anderen Datenbanken und Files auf einem Server.
- Individuelle Tabellen: Bestimmte Tabellen in der Datenbank werden klassifiziert. zB. Könnte man den Zugriff bestimmte Tabellen als eingeschränkt und nicht eingeschränkt klassifizieren.
- Individuelle Spalten: Es könnte auch von Interesse sein Zugriff auf gewissen Spalten nach Sicherheit zu Klassifizieren. (Mitarbeiter Gehalt)

Die Feinheit kann natürlich noch immer weiter adjustiert werden: Auch interessant: Begrenzter Zugriffs auf bestimmte Datensätze (Reihen) oder sogar auf ein bestimmtes Feld, eines bestimmten Datensatzes.

Read Access: (ssp-Regel) Leicht zu implementieren falls MLS auf Datenbank- oder Tabellen-Level agiert.
Unangenehmer wirds bei feinkörnigeren Varianten: Restriktion auf Gehalt: SELECT name FROM mitarbeiter WHERE gehalt ¿ 50. Unter Umständen wird Ergebnis retourniert obwohl ja eigentlich Zugriffe auf Gehalt verboten wurden. Lösung –¿ nicht nur retournierte Daten überprüfen. Reihen-Restriktionen können NULL-Rückgaben auf sensible Informationen deuten lassen. (Feld-Restriktionen werfen keine **neuen** Probleme auf)

Write Access: (*-p-Regel) Nicht so simpel realisierbar wie ssp. Will ein Nutzer mit einer low-clearance Daten in eine Reihe mit den gleichen Primär-Schlüssel als eine bereits existente high-clearence Reihe einfügen, so hat das DMBS drei Möglichkeiten:

1. Rückmeldung an den Benutzer dass Einfügen nicht Möglich ist. (Schlecht, weil Benutzer damit ja quasi von oberen Datensätzen liest)
2. Kommentarloses überschreiben des High-level Datensatzes. (Schlecht, wegen Datenintegrität)
3. Benutzung von Polyinstantiation: Neue Reihe mit tieferem Security-Level einfügen ohne die bereits existierende High-Level Reihe zu beeinflussen. (Okay, aber nicht so toll. Datenbank nicht mehr eindeutig.)

Selbe Probleme gibts beim Updaten eine Reihe. Bei Granulitäts-Level auf Datenbank- bzw. Tabellen-Ebene gibts wiedermals keine Probleme.

10.11 Trusted Plattform Module

Das Konzept kommt direkt aus der Hölle von der Trusted Computing Group. Das TPM ist ein zentrales (Hardware) Modul in diesem Konzept (=heute: Trusted Computing). Der TPM-Chip wird in einen Computer bestehend aus TC-unterschütztenden Komponenten eingebaut, der mit TC-unterstützender Software betrieben wird. Das TPM generiert einen Schlüssel, den es mit den verwundbaren Komponenten teilt, die Daten im System herumbewegen. (Speichergeräte, ...) Diese Schlüssel können benutzt

werden um verschlüsselte Daten die durch das System „fliessen“ zu entschlüsseln. Die Software auf den System, kann dann davon ausgehen, dass die erhaltenen Daten vertrauenswürdig sind. Und das System kann sicher sein, dass die Software es auch ist.

TC wurde eingeführt um das Vertrauen von Admins und Benutzer in die Sicherheit ihres Systems zu stärken. Die Idee enstand schon in den 1970ern vom U.S. DOD, setzt sich aber wegen Kosten und Performance Problemen wirklich durch. Die Wunsch nach Evaluationskriterien sicherer Systeme der NSA und der US Regierung führten zur „Trusted Computer System Evaluation Criteria (TCSEC)“, daraus entstand dann später die „Common Criteria“

Die drei basis-Dienste von TC sind:

- **Authentifizierter Boot**-Service: Sorgt für den kompletten (sicheren) Start eines Betriebssystems. Das Betriebssysteme baut sich beim starten Stück für Stück auf. Das die TC-Hardware (das TPM?) checkt bei jedem dieser Schritt auf die Integrität der einzelnen Stücke und führt dabei ein fälschungs-sicheres Log (Hashfunktion). Ist das System fertig gebootet kann die „Chain of Trust“ um vertraute Hard- und Software erweitert werden. (TPM-Liste bestimmt was vertrauenswürdig)
- **Zertifizierung**s-Dienst: Hat man eine fertige TC-Konfiguration am start, kann das TPM diese für andere Parteien zertifizieren. Das TPM liefert hierbei eine mit ihrem private-key verschlüsselte Beschreibung der Konfiguration, wodurch dieser voll vertraut werden kann. (TPM an sich vertrauenswürdig, was auch immer es privat signiert ist auch vertrauenswürdig). Die Zeitliche-Korrekheit der Konfiguration kann vom Dritt-Anbierte durch Challenge/Response sicher gestellt werden. TPM = hierarchische Vertrauensstruktur: Traue TPM, traue System, trause OS, traue Software)
- **Verschlüsselung**s-Dienst: Der Verschlüsselungservice verschlüsselt Daten so, dass nur eine bestimmte Maschine in einem bestimmten Zustand diese entschlüsseln kann. Das TPM verfügt über einen geheimen, einzigartigen Master-Schlüssel. Aus diesem generiert es sich Schlüssel für alle unterstützten Konfigurationen eines Systems. Werden Daten unter einer laufenden Konfiguration erstellt, so können diese von einer späteren, veränderten Konfiguration nicht mehr genutzt werden. Kann auch insofern erweitert werden, dass Schlüssel für bestimmte Anwendungen auf bestimmen OS generiert werden.

10.11.1 TPM-Funktionen (BILD)

: Komponenten von der neuesten TPM-Spezifikation (Auszug):

- Crypto-Graphischer Co-Prozessor: Für RSA-Rntschlüsselung
- Key generation: Erstellt RSA public/private-Paare und symmetrische Keys.
- HMAC engine: Der Algorithmus (HMAC) wird in diversen Authentifizierungsprotokollen benutzt.

Weitere auf der Folie.

10.11.2 Geschützt-Speicherungs-Funktion

Ein wichtiges feature von TC ist das ein geschütztes Objekt „verschlossen“ werden kann, um nur in einem bestimmten Software-Status genutzt zu werden. Wenn TPM das Objekt „aufschließt“ (und das sicher, innerhalb der TPM), überprüft das TPM ob der indizierte Software-Status den aktuellen entspricht. Wenn ja ist zugriff erlaubt, sonst nicht. Ein konkretes Beispiel hier zu auf der Folie.

10.12 Common Criteria

Common Criteria sind ISO-Standards um Security-Anforderungen und Evaluationskriterien zu spezifizieren, die bei der Software-Entwicklung und Weiterentwicklung eingesetzt werden können. Wie TPM zielt es ab vertrauen in IT-Systeme (hier: durch bestimmte Aktionen die bei Entwicklung getroffen werden) zu stärken. Werden die Anforderungen eines IT-Produkts erfolgreich evaluiert so kann es von Organisationen wie die NIST/NSA als CC-zertifiziert bestätigt werden.

10.12.1 Anforderungen

Der Ausdruck „Target of Evaluation" TOE beschreibt die Teile des Produkts, die Subjekt der Evaluation sind. Die Anforderungen fallen in die Kategorien Funktionale-Anforderungen (Definieren das gewollte Security-Verhalten standardmäßig) und Versicherungs-Anforderungen(Formen die Basis zur Sicherstellung, dass die gewollten Sicherheitsmaßnahmen auch effektiv und korrekt implementiert wurden). Beide Kategorien werden in Klassen organisiert. Diese Klassen sind Sammlungen an Anforderungen bestimmten Fokusses bzw. bestimmter Intention.

10.12.2 CC Profile und Ziele (BILD)

Der CC definiert außerdem zwei Dokumentarten, die CC-definierte Anforderungen nutzen.

- Protection-Profiles (PPs): Beschreiben einen implementations-unabhängigen Satz an Anforderungen und Zielen für bestimmte Produktkategorien. (welche ähnliche Anforderungen an Sicherheit haben) PPs sollen wiederverwendbar sein und sollen sozusagen bei der Identifikation benötigter Anforderungen für gesetzte (Sicherheits-)Ziele helfen. PP=Zeigt Sicherheitsanforderungen von Benutzer.
- Security-Targets (STs): Enthalten Ziele und Anforderungen eines bestimmten Identifizierten TOE and definieren Maßnahmen um diese zu befriedigen. Ein ST kann konform mit mehreren PPs sein und formt die Basis für deren Evaluation. ST=Wird von Hersteller/Entwickler abgeliefert.

Für eine PP kann ein Nutzer nun eine Rehe an (möglicherweise vorgegebenen, können auch ganze Packages sein) Komponenten auswählen und so die Anforderungen für das gewünscht Produkt definieren. Ähnlich, kann der Hersteller/Entwickler auch eine Nummer an Packages und Komponenten auswählen um eine ST zu definieren.

Beispiel: Smartcard PP: Wird von einer SmartCard Security User-Group entwickelt. Beschreibt Anforderungen der SmartCard um mit sensitiven Applikationen(Banking, ...) sicher genutzt werden zu können. Das Zusicherungs-Level von diesem PP ist EAL 4. Die PP listet Bedrohungen auf, die von Produkten, die von sich behaupten die PP umzusetzen, abgedeckt sein müssen. (z.B. Physikalische Sondierung, Ungültiger Input, ...) Danach folgt eine Liste an Security-Zielen, die dazu dienen den identifizieren Threats entgegen zu wirken und mit geltenden (organisationsabhängigen) Security-Richtlinien einher zu gehen. Das PP listet nun Anforderungen in (grob) drei Bereichen: TOE Funktionale-Anforderungen, TOE Zusicherungs-Anforderungen und Anforderungen für die IT Umgebung. Das letzte Kapitel des PP ist einzig eine Begründung für alle Auswahlen und Definitionen die in diesem PP getroffen/gelistet wurden.

10.12.3 Zusicherung (en. Assurance)

Security-Zusicherung ist der Grad an Vertrauen, die jemand in die korrekte Operation und den Schutz von Security-Steuerungselementen hat, jedoch keine absolute Garantie dafür, dass die Maßnahmen genau so funktionieren wie vorgesehen. Zusicherung betrifft die Anforderungen, Security-Richtlinien, Produkt Design und Implementation und Betrieb eines IT-System. Es gibt mehrere Herangehensweisen

um Zusicherung(en) zu realisieren (Auszug): Analyse von Prozessen und Prozeduren und Überprüfung ob diese auch angewendet werden, Penetrations Tests, Unabhängige Funktional-Test, Analyse der Korrespondenz zwischen TOE-Design Repräsentationen ...
Es gibt 7 Stufen zur Bewertung von Zusicherung, von am wenigsten Präzision und Weitblick (EAL 1) zum Meisten (EAL7). Die Levels lauten wie folgt (Auszug):

- EAL1: Funktional getestet: involviert unabhängige Produkt-Überprüfung ohne Input der Entwickler um ein Level an Vertrauen in korrekte Funktionsweise zu bieten
- EAL2: Strukturiell getestet: (EAL1) plus ein Review einer high-level Design und Schwachstellen Analyse von einem Produkt-Entwickler
- EAL3: Methodisch getestet: inkludiert Anforderungen, dass das Design Security-bezogene Komponenten von anderen trennt und, dass das Design spezifiziert wie Security letztendlich ausgeführt wird
- EAL7: Formal Verifiziertes Design (ergänzt durch eine Formale Präsentation der Funktionalen-Spezifikationen und High-Level Design) und getestet.

Umso höher das Level ist ums so umfassender und zeitintensiver fällt die Evaluation aus.

10.12.4 Evaluation

Das Ziel einer Evaluation einer TOE ist sicherzustellen, dass die Security Features des TOEs korrekt und effektiv arbeiten, und keine Verwundbarkeiten mehr zeigt. Der Evaluations Prozess kann sowohl während als auch nach der Entwicklung (Festlegung) des TOES durchgeführt werden, abhängig vom Verbunden Zusicherungs-Level. Der erhoffte Ausgang einer Evaluation ist, dass die Sicherheit für die TOE ausreicht und von einem technischen Evaluations-Report (Dokument) bestätigt ist. Einbezogen wird hier unter anderem High- und Low-Level Design, Funktionale Spezifikationen, Implementation des Source Codes etc. Bei höheren EAL-Level müssen wie erwähnt auch (semi-)formale Modelle entwickelt werden.

Beteiligte Partien und Phasen der Evaluation: Involviert: Sponsor (Hersteller oder Kunde, Entwickler (bietet Beweise zur Evaluation), Evaluator (bestätigt das Anforderungen befriedigt sind), Zertifikator (Regierungsbehörde die Evaluationsprozess beobachtet, national unterschiedlich). Es gibt drei Phasen:

1. Vorbereitung (Initialer Kontant zwischen Sponsor, Entwickler und Evaluator, inkludiert ein Review der Ziele und andere Evaluationsmöglichkeiten)
2. Durchführung der Evaluation (Strukturisierter, formaler Prozess in welchem der Evaluator eine Serie von Aktivitäten (spezifiziert von CC) durchführt)
3. Konklusion (Evaluator liefert die fertige technischen Evaluations Report an den Zertifikator ab, der den Prozess validiert und für einen öffentlichen Zertifikations-Report vorbereitet)

11 Buffer Overflow

Dieses Kapitel widmet sich voll und ganz Buffer-Overflows, einen sehr gebräuchlichen aber auch altbekannten Angriffs Mechanismus. Buffer-Overflows können bei Programmen ausgenutzt werden die (mehr oder weniger) gedankenlos programmiert wurden. Obwohl altbekannt, tritt es selbst heute noch recht häufig auf, dank schlecht programmierten Programmen und alten Code-Teilen die heute noch in Systemen Einzug finden.

11.1 Basics

Ein Buffer-Overflow ensteht wenn durch einen Programmier-Fehler Daten über die Limitation eines „Buffers“ mit strikter Größe geschrieben, und dadurch benachbarte Speicherbereiche modifiziert werden. Die Konsequenzen so eines Fehlers reichen von unerwarteten Rechte- bzw. Datentransfer (acces violation) bis hin zur (sogar recht wahrscheinlichen) Termination des Programms. Wird's richtig angestellt, kann der Angriff womöglich sogar fremden Code mit dem Privilegien des angegriffen Prozesses ausführen.

In diesem Kapitel gib es einige Code-Beispiel die ich NICHT herausschreiben werden. Diese bitte einfach selbst ansehen, ich werde sie maximal kurz beschreiben.

CodeBSP: Ein C-Code wo zwei Strings vergleicht werden. Beide sind 8 chars lang. Der erste kriegt einen festen Wert zugewiesen (z.B.) „(“START) und der zweite kriegt per Standardinput zeichen rein. Das Problem ist, dass der Standardinput garnicht aufpasst wieviele Zeichen er reinfrisst, gibt man mehr als 8 chars ein, wird in benachbarte Speicherbereiche (in dem Beispiel String 1) einfach hineingeschrieben. Wie und wo (also auch die Nachbarn im Speicher) gespeichert wird hängt von Programmiersprache, Compiler, Maschinen-Architektur und Variablen-Sichtbarkeit ab. Auf der nächsten Folie wird genau dieses Beispiel sozusagen im Speicher dargestellt. Dazu wird noch vermerkt, dass falls die zu vergleichenden Werte hier gleichwertig überschrieben wurden, zu einem Programmablauf führen könnten, der so verläuft als wäre der Vergleich erfolgreich gewesen. z.B "BADINPUTBADINPUT"(16 Zeichen, 0-7 und 8-15 werden verglichen) Ginge es hier um sowas wie eine Passwortabfrage: kritisch.

Um Buffer-Overflows **erfolgreich nutzen** zu können, muss der Angreifer über folgendes **Wissen verfügen**:

1. Er muss über eine Buffer-Overflow Verwundbarkeit bescheid wissen, die mit Hilfe extern eingefügter Daten (unter Kontrolle des Angreifers) ausgelöst werden kann
2. Er muss verstehen wie der Buffer wo gespeichert ist, und welches Potential für Angriffe sich im umliegenden Speicher befindet.

Dieses Wissen erlangt man durch Inspektion des Source-Codes, Aufzeichnen und Durchdenken der Programmabläufe wenn sie übergroßen Input erhalten, oder durch Techniken wie Fuzzing (Tool erzeugt zufällig Daten die in Input-Mög. von Programm gesteckt werden).

11.2 Ein bisschen Programmier-Sprachen-Geschichte

Daten die vom Prozesser auszuführen sind werden im Register oder im Hauptspeicher verwahrt. Die Daten sind einfach Byte-Felder. Wie sie zu interpretieren sind hängt nur von der Funktionweise der Instruktionen mit denen auf sie zugegriffen wird. Moderne Sprachen wie z.B. Java haben eine starke Auffassung vom Typ der Variablen und was man mit ihnen anstellen darf, daher sind sie nicht sehr anfällig für Buffer-Overflows. Diese erhöhte Sicherheit resultiert natürlich in erhöhter Systemauslastung. Dazwischen existieren Sprachen wie z.B. C, die viele moderne Konzepte auffassen, jedoch immer noch Möglichkeiten bieten um auf Speicher direkt zuzugreifen. Hier hat sich jedoch gezeigt das viele Funktionen aus Standardbibliotheken recht alt und somit verwundbar für Buffer-Overflows sind.

11.3 Funktions Aufrufe, Stack-Frames und Stack-Overflows

Wird eine Funktion aufgerufen muss Rückkehr-Adresse, Parameter die der Funktion übergeben werden und möglicherweise Werte die gespeichert werden, da man nach dem Aufruf mit diesen weiter operieren möchte gesichert werden. Diese Werte werden meistens auf einem Stack(-Frame) gespeichert. Ein (BILD) stellt das ganze graphisch dar. Wie's genau aussieht hängt natürlich wieder von Programmiersprache etc. ab.

Ein **Stack-Buffer-Overflow** entsteht nun wenn der Ziel-Buffer auf einem Stack liegt, meist als lokale Variable in dem Stack einer Funktion. Diese lokalen Variablen sind meist unter dem Frame-Pointer bzw. der Rückkehr-Adresse gespeichert. Durch diesen Umstand kann man es Schaffen durch einen Buffer-Overflow diese Link-Adressen zu überschreiben und damt auf jede beliebige Ressource im Speicher oder/aber auch System-Libaries zu verweisen. In der „(“ursprünglichen) Heransgehensweise wurde allerdings einfach auf den Code im exploiteten Buffer verwiesen. Diese Angriffsart wird auch Stack-Smashing genannt.

Ein kurzer Blick auf den Aufbau von Prozessen im Speicher: Wird ein Progamm gestartet, erstellt das OS automatisch einen Prozess mit eigenen virtuellen Speicherbereich. Im (BILD) im Beispiel hier sind die Inhalte der Executable-File (globale Daten, Relocation Table und Programm und Code Segmente) ganz am „Boden“ des Adress-Bereichs. Drüber ist der Programm-Heap der nach oben wächst, dann ein wenig Freiraum und schließlich Platz für den bereits erwähnten Stack.
Stack-Overflow CodeBSP: Einfach eine simple Implementation von Bereits erwähnten über mehre Folien (2 Bps) mit entsprechenden Speicher-Bildern

11.4 Shellcode

Eine essentielle Komponente vieler Buffer-Overflow attacken ist der Transfer von Code des Angreifers in das Programm, meist im selben Buffer gespeichert, der exploitet wird. Der Code ist bekannt als shellcode, weil er traditionell dazu eingesetzt wird eine Shell (Eingabeaufforderung) am System zu installieren. Dieser Shellcode kann auch vorinstallierte Shells ausnutzen bzw. aufrufen. Damit das funktioniert gibt es aber einige Dinge bei der Programmierung zu beachten. Der Shell-Code muss z.b. position-indendent sein, da er nicht wissen kann, wo er sich nun genau im Speicher befindet.

Stack-Overflow CodeBSP: Beispiel zu Stack-Overflow der Shell-Code installiert, über mehrere Folien.

11.5 Weitere Stack-Overflow Varianten

Ziel muss nicht unbedingt eine vertrautes System-Werkzeug sein. Ein anderes mögliches Ziel ist einen Programm das Netzwerk-Dienste anbietet. Angriff könnte dann mit Daten über das Netz passieren. Oder ein Programm bzw. Bibilotek das Datenformate wie GIF, JPEG oder ähnliches lädt (Durch Injektion von Code in das darzustellende Bild, gilt auch für jedes andere Medien/Dokument-Formate). Der Angreifer könnte auch bessere Funktionalität als nur Shellcode realisieren wollen z.B. ein Service für eine Remote-Shell, oder eine Reverse-Shell auch Firewall-Regelsätze könnten gelöscht werden. Der Fantasie sind wahrlich keine Grenzen gesetzt.

Replace Stack-Frame ist eine Variante des klassischen Stack-Buffer Overflows überschreibt nur den Buffer und die saved-Frame-Pointer Adresse. Der saved-Frame-Pointer wird so überschrieben, dass er über die Spitze des überschriebenen Buffers, wo ein „dummy-Stack“ erstellt wurde, zeigt. Dieser Stack verfügt nun über eine Return-Adress die auf den Shellcode weiter unten im Buffer zeigt. Kann benutzt werden wenn nur ein limitierter Buffer-Overflow möglich ist. (z.B. überschreiben von Return-Adresse verboten) Jedoch schwierig zu realisieren. Der Angreifer muss wissen wo genau Buffer und Dummy-Stack im Speicher stehen.
„Defenses: Stack-Schutz Mechanismen, nicht-ausführbare Stackblocks, Randomisierung der Stackposition“

Return to System-Call: Find das hier sehr unverständlich geschrieben. Nach Wikipedia-Recherche geht's vermutlich darum: Wir lernen hier später noch in dem Kapitel diverse Schutzmaßnamen, unter anderem nicht-ausführbaren Speicher kennen. Dann darf kein Code mehr der in Variablen etc. steht ausgeführt werden, weil diese sich in nicht-ausführbaren Speicherbereichen befinden. Die Idee hier ist

jetzt die Return Adresse so zu überschreiben, dass sie auf eine generell existente System-Funktion zeigt (aus irgendeiner geeigneten Bibliothek) die dann mit passenden Parametern (die irgendwo anders im Stack platziert werden) aufgerufen wird. So kann man trotz nicht-ausführbaren Speicher immer noch recht coole Sache machen.
„Defenses: Stack-Schutz Mechanismen, Randomisierung von Stack- und Biblothekspositionen"

Heap-Overflow: Der Heap befindet sich typischerweise gleich über den Code und den globalen Programmdaten und wächst nach oben. Informationen werden von Progammen aus dem Heap geholt, für die Nutzung dynamischen Datenstrukturen wie Listen etc. Wenn so ein Heap einen Buffer enthält der verwundbar ist, kann der umgebende Speicher korrupiert werden. Anders als beim Stack gibt's hier keine Return-Adressen, jedoch wenn in der Umgebung auch Pointer zu einer Funktion liegen, kann der Angreifer diesen so überschreiben, dass der Pointer auf Shellcode im exploiteten Buffer zeigt. (Da Heaps ja gerne für Strukturen wie LinkedLists eingesetzt werden, bieten sich die Link-Pointer der einzelnen List-Elemente an) Alternativ können auch „Management Daten-Strukturen" exploitet werden. (? Was auch immer)
„Defenses: nicht-ausführbare Speicherbereiche, Randomisierung des Heaps im Speicher, Checks auf Korruption der Management-Daten"

Heap-Overflow CodeBSP: Wieder ein Beispiel über mehrere Folien.

11.6 Global Data-Overflow

Der zu exploitende Buffer liegt einfach in den Speicherbereich der globalen Variablen des Programmes, also über dem Programmcode. Liegt irgendwo daneben ein Funktionspointer, kann der exploited werden. Komplexere Angriffe dieser Art könnten Versuchen spezifischere Ziele wie „destructor"-Funktionen (eine GCC und C++ extension), globale-Offset-Tabellen (benutzt um Funktion-Referenzen zu dynamischen Bibliotheken aufzulösen, sobald diese geladen sind) und andere Strukturen zu infilitrieren. Ziel ist es auch Pointer zu überschrieben und auf eigenen Programmcode zu linken.
„Defenses: nicht-ausführbare Speicherbereiche, Funktionspointer unter jede Art von Daten platzieren, Guard-Pages zwischen globalen Datenbereiche und anderen Management-Bereichen"

11.7 Verteidigung gegen Buffer-Overflows

Hier gibts Hauptsächlich zwei Typen: Compile-Time Schutz, zielt ab Programme weniger verletztlich gegen solche Angriffe zu machen und Run-Time Schutz, der versucht bösartige Operationen zu erkennen und abzubrechen. Ideen dazu gibt es schon eben lange, aber noch länger gibt es eine große Basis an verwundbarer Software. Insofern ist Run-Time Schutz besonders als Update für ein O/S interessant, da dieses dann selbst verletzlichen Programmen etwas Schutz bieten kann.

11.7.1 Compile-Time Defences

- **Programmiersprache**: Verwendung einer modernen High-Level Programmiersprache die starke Typisierung unterstützt. Compiler führen hier zu meist auch Rangechecks durch und nur bestimmte Operationen werden bestimmten Typen erlaubt. Negativ: Leistung durch höhere Abstraktion nicht so super, außerdem lässt sich letztendlich doch nicht alles mit solchen sprachen realisieren (Devicedriver) und irgendwo muss dann doch zu C o.ä. zurückgegriffen werden.
- **Verwendung sicherer Programmier-Techniken**: Besonders wenn Sprachen wie C benutzt werden, müssen Programmierer aufpassen sicheren Code zu schreiben. (überlegen wo Overflows aufteten könnten und verhindern, sehr „vorsichtig" Programmieren und überall wo notwendig überprüfen und korrigieren bzw. neu schreiben [auch bei Funktionen aus System-Bibliotheken]) Für absolute Sicherheit muss hier möglicherweise Code mehrmals überarbeitet werden.

- **Sprachen-Erweiterung und sichere Bibliotheken**: Unsichere Pointer und Referenzen (wie z.B. in C) haben dazu geführt, dass man begonnen hat Compiler zu modifizieren, so dass diese automatische Rangechecks auf derartige Referenzen durchführen. Funktioniert bei statisch allokierten Arrays supergut, allerdings bei dynamischen Inhalten schon schlechter. Sowas führt auch meist zu Performanceverschlechterungen. Ein Weg um unsichere Funktionen aus Standardbibliotheken zu umgehen ist diese mit sichereren zu ersetzen oder komplett andere, sicherere Bibliotheken zu verwenden. (z.b. Libsafe für C)

- **Stack-Schutz**: Effektiver Schutz ist einfach Entry- und Exitfunktionen zur Stack-Datenstruktur hinzuzufügen die den Frame auf Hinweise von Korruption checken. Wird eine Modifikation ausfindig gemacht, wird das Programm unterbrochen (z.b. mit zufälligen Kanarienvogel (canary, srsly) - zufälliger Wert wird bei Entry unter Frame Pointer Adresse geschrieben und bei Exit gecheckt ob dieser noch unmodifiziert ist. ODER bei Entry einfach die return-Adresse in einer sicheren Region des Speichers sichern und bei Exit mit derzeitigen Stand der return-Adresse vergleichen. Falls korrupt –¿ Abbruch)

11.7.2 Run-Time Defences

- **Nicht-Ausführbarer Adressbereich**: Viele Buffer-Overflows kopieren Maschinen-Code in einen Buffer und transferieren dann die Ausführung darauf. Ein Mittel dagegen ist Ausführung am Datenspeicher (stack/heap/...) einfach zu verbieten. Wird als eine der besten Methoden angesehen.

- **Adressbereich Randomisation**: Mann kann auch die Struktur von Schlüssel-Datenstrukturen manipulieren. Der Angreifer muss letztendlich Vorraussagen können, wo er sich (zumindest in etwa) im Speicher befindet. Eine Möglichkeit das zu Verhindern ist z.B. die Lokation dieser Datenstruktur bei jedem neuen Prozess zufällig festzulegen. Viele Programme brauchen weitaus weniger Speicher als moderne 32- bzw. 64-Bit OS bieten, dadurch ist bei zufälliger Lokation solcher Datenstrukturen ein zufälliges, „glückliches“ gelingen einer solchen Attacke absolut unwahrscheinlich. Auch Heap-Buffer und Standard-Bibliotheken können zufällig allokiert werden.

- **Guard Pages**: Zwischen kritische Speicherbereiche können sogenannte Guard Pages gelegt werden. Die Speicherbereiche dieser Guard Pages sind von der MMU als illegale Adressen markiert, jeder Versuch auf sie zuzugreifen führt dazu das der zugreifende Prozess abgebrochen wird. Solche Guard Pages können auch zwischen Stack-Frames gelegt werden oder verschiedenen Allokationen in einem Heap.

12 Software Security

Viele Schwachstellen ergeben sich durch schlechte Programmierpraktiken. Oft liegt es an fehlender Überprüfung von Input, Daten und Fehlerbehandlung. Beispiele sind Angriffe durch Buffer Overflow, Injections, und Cross-Site Scripting.

12.1 Grundlegendes

12.1.1 Quality vs Security

Während bei der Qualitätssicherung eher versucht wird, die am öftesten auftretenden Bugs zu beseitigen, wird sich ein Hacker genau diese Vorgangsweise zu nutze machen, um durch nicht häufig auftretende Bugs (z.B. mit absolut abwegigem Input) eine Schwachstelle zu finden.

12.1.2 Defensive Programming

Angeblich haben Programmierer normalerweise eher das Ziel im Kopf. *Defensive* oder *Secure Programming* wird im Gegensatz dazu ein Programmierstil genannt, bei dem man darauf achtet, alle möglichen Fehlerfälle abzufangen und alle Annahmen zu überprüfen, damit das Programm auch bei unerwarteter Benutzung fehlerfrei funktioniert.

Da Programme Daten aus verschiedenen Quellen nehmen, auf verschiedene Weisen speichern, womöglich auch mit anderen Programmen kommunizieren, und das auf verschiedenen Architekturen und Betriebssystemen, gibt es hier vieles zu beachten. Da man aber eher dazu gedrängt wird, ein Programm schnell fertigzustellen, bleibt die Sicherheit oft auf der Strecke, wenn sie kein erklärtes Ziel ist.

12.1.3 Security by Design

In allen Ingenieursberufen ist Verlässlichkeit ein grundlegendes Konzept; man würde auch nicht tolerieren, dass eine Brücke ein- oder ein Flugzeug abstürzt. Bei der Softwareentwicklung ist die Messlatte viel niedriger gelegt und größere Fehler gelten als akzeptabel. Allerdings ändert sich das in letzter Zeit, da Sicherheit ein Schlüsselthema wird.

12.2 Handling Program Input

Falsche (oder fehlende) Behandlung von Input ist ein häufiger Schwachpunkt. Dabei beschreibt „Input“ alles, was von außerhalb des Programms kommt: Eingaben vom Keyboard, Daten aus dem Netzwerk, aber auch Konfigurationsdateien oder vom Betriebssystem bereitgestellte Funktionen. Bei all diesem muss die tatsächliche Größe und Art der Informationen mit den erwarteten Werten übereinstimmen, bevor sie weiterverwendet werden.

12.2.1 Input Size

Oft wird beim Einlesen von Daten eine gewisse Größe für allokierten Speicher angenommen, die unter normalen Verhalten auch logisch erscheint, aber dann nicht überprüft, ob die Inputgröße die Größe des reservierten Speichers nicht überschreitet. Wenn dann dieser zu große Input über die reservierten Speicherbereiche hinaus geschrieben wird, und so in Bereichen landet, wo er nie sein dürfte, spricht man von einem **Buffer Overflow**. Beim Testen auf bloße Fehler („Qualitätstest“) könnte man eine solche Lücke nicht finden, da hier ein unnatürlich großer Input benutzt werden würde. Deshalb muss man beim Einlesen von externen Daten immer misstrauisch sein.

12.2.2 Interpretation of Input

Ein anderer wichtiger Punkt ist die Interpretation der ankommenden Daten. Der Input kann binär kodiert sein, in diesem Fall muss aber die spezielle Kodierung bekannt sein, also ist es normalerweise „application specific“. Verbreiteter ist es, den Input als Text zu behandeln, der z.B. in ASCII kodiert ist. Durch die Internationalisierung ist dieser Teil der Erkennung schwerer geworden, da auch immer mehr andere Zeichensätze verwendet werden.

Außerdem müssen, in beiden Fällen, die enthaltenen Datentypen erkannt werden. Es wäre fatal, z.B. statt einer Zahl eine URL anzunehmen. Dies muss daher auch immer überprüft werden.

12.3 Injection Attacks

Angriffe, die diese Schwachstellen in der Input-Behandlung ausnutzen, um den Programmablauf zu verändern, werden **Injection Attacks** genannt. Oft werden zu diesem Zweck Daten verändert, die als Parameter an andere Programme übergeben werden.

Da bei Scriptsprachen gefördert wird, auf andere Programme und Systemroutinen zuzugreifen, um Code einzusparen, sind diese ein besonders häufiges Opfer.

12.3.1 Command Injection

Kurze Erläuterung des Beispiels: Hier soll über ein Perl-Script die Unix-Funktion `finger` verfügbar gemacht werden (d.h. mit HTML-Output). Diese stellt einfach nur Informationen über den als Argument angegebenen User zur Verfügung. Theoretisch funktioniert es, wenn man einen gültigen Usernamen eingibt. Praktisch lässt sich aber so auch *jedes* Programm mit den Rechten des Webservers ausführen, wenn man durch `;` getrennt neue Unix-Befehle eingibt und das alles als Argument übergibt. Das nennt sich dann **Command Injection**.

Dies kann ganz einfach gelöst werden, indem man den Input validiert. Lässt man keine Sonderzeichen im Argument zu, kann eine solche Attacke auch nicht stattfinden.

12.3.2 SQL Injection

Ähnlich wie bei der Command Injection wird hier ebenfalls durch ein sogenanntes *Funktions-* bzw. *Metazeichen* so getan, als sei das Statement beendet, und ein anderer Befehl ausgeführt. Bei SQL ist dieses Zeichen ein `'`, und die Verhinderung basiert auf dem gleichen Prinzip.

12.3.3 Code Injection

Eine Variante, die oft Teil eines Angriffes mit Buffer Overflow ist. Aber es geht auch eigenständig, so z.B. mit PHP, was folgende Gründe hat:

- Es hat selber keine Zugangsbeschränkungen, und diese müssen daher im Server konfiguriert werden.
- Der Wert einer Variable, die den Wert eines HTTP-Requests aufnimmt, wird automatisch einer globalen Variable mit demselben Namen wie dem des Input-Felds zugewiesen.
- Durch `include` kann Code von *überall* geladen und ausgeführt werden.

12.3.4 Cross Site Scripting Attacks

Dieser Begriff beschreibt eine Klasse von Angriffen, bei der der Input des einen Users (also des Hackers) den Output von anderen Usern beeinflusst. Das ist ein Versagen an zwei Fronten, da weder der Input noch der Output richtig kontrolliert werden.

Das passiert oft auf Webseiten und in von Browsern unterstützten Skriptsprachen wie Javascript, ActiveX oder Flash. Dabei wird deren Annahme ausgenutzt, dass jeglicher Code von einer Seite gleich einzustufen ist und daher auch, wie im Beispiel, Schadcode von einer sonst sicheren Seite ausgeführt wird.

Beispiel: Jemand trägt in einem Gästebuch einen Kommentar ein, der, wenn er den anderen Usern angezeigt wird, als Script interpretiert wird somit jeglicher Code ausgeführt werden kann, in diesem Fall wird ein anderes Script geladen. Dadurch könnte der Angreifer z.B. Cookies von allen Leuten sammeln, die diese Seite aufrufen. Dies wird als **XSS reflection** bezeichnet. Um die Entdeckung zu erschweren, könnte man übrigens z.B. auch alle Zeichen in HTML escapen.

12.4 Validating Input Syntax

Wie eben gezeigt, ist es notwendig, den Input zu überprüfen. Dabei setzt man am besten fest, was erlaubt ist, und vergleicht den Input damit, anstatt nach bösartigen Eingaben zu schauen, da solche vielleicht noch nicht entdeckt wurden (d.h. Whitelist anstatt von Blacklist). Meistens werden für diese Spezifizierung reguläre Ausdrücke verwendet.

Hat man nicht erlaubten Input gefunden, kann man ihn entweder ganz verwerfen, oder ihn verändern. So könnte man z.B. Metazeichen escapen, sodass Command Injections nicht mehr funktionieren.

12.4.1 Alternate Encodings

Einige Zeichen könnten innerhalb eines Zeichensatzes mehrere Darstellungsarten besitzen, was ebenfalls eine Fehlerquelle darstellt, da z.B. einem Programm, das einen lokalen Pfad erwartet, irgendein absoluter gegeben werden kann, wenn man die Überprüfung durch eine alternative Darstellung entsprechend austrickst. Deshalb muss die Darstellung **kanonisiert**, d.h. standardisiert, werden, bevor der Input weiterverarbeitet wird.

12.4.2 Validating Numeric Input

Man muss vor allem beim Überführen einer Zahl von einem Typ in einen anderen drauf achten, dass dies auch konsequent überprüft wird, denn vorzeichenlose Zahlen anders dargestellt werden als vorzeichenbehaftete.

12.4.3 Input Fuzzing

Input Fuzzing ist der Name einer recht guten Methode, um Programme auf Bugs zu testen. Dabei werden bloß extrem viele verschiedene zufällige Eingabedaten für das Programm generiert, um zu überprüfen, ob es trotzdem richtig funktioniert.

Man kann diese zufälligen Daten auch nach bestimmten Mustern erzeugen, um bestimmte Problemfälle abzufangen, sollte aber keinesfalls auf die rein zufällige Methode verzichten.

12.5 Writing Safe Program Code

Nach dem Input folgt das Programm selbst. Da dieser Befehle ausführt und Daten im Speicher ändert, ist es wichtig, dass der Algorithmus korrekt umgesetzt ist und die Daten nur auf sinnvolle und richtige Weise geändert werden.

12.5.1 Correct Algorithm Implementation

Erklärt sich irgendwie von selbst. Ein Algorithmus soll das tun, was er tun soll. Als Beispiel wird hier der Random Number Generator von Netscape genannt, der aber nicht gerade random war, und so die Session Keys berechnet werden konnten.

Auch ist es ein Problem, wenn Testcode im Endprodukt vergessen wird, da es natürlich ein Risiko darstellt.

12.5.2 Correct Machine Language

Heutzutage geht man davon aus, dass Compiler und Interpreter richtigen Maschinencode erzeugen. Das stimmt auch bis auf abstruse Ausnahmefälle, aber für Programme mit ganz hohen Sicherheitsanforderungen muss man dann trotzdem den Maschinencode durchgehen.

12.5.3 Correct Data Interpretation

Auch muss man darauf achten, dass Datentypen richtig interpretiert werden (Character, String, Integer, Pointer, ...). Unter C kann man Integers ganz leicht zu Pointern machen und Chaos veranstalten. Deswegen gilt, dass Programmiersprachen mit starker Typisierung verwendet werden.

12.5.4 Correct Use of Memory

Neuere Programmiersprachen achten selber auf die Memory Allocation, aber bei älteren muss man den Speicher explizit wieder freigeben, ansonsten geht einem einfach irgendwann der Speicher aus und das Programm oder sogar das System stürzt ab. Das nennt sich **memory leak** und kann auch von einem Angreifer genutzt werden.

12.5.5 Race Conditions in Shared Memory

Wenn Prozesse gleichzeitig auf eine Ressource zugreifen, kann es zu mehreren Problemen kommen. Ohne Kontrolle können sie sich gegenseitig ihre Ergebnisee überschreiben, was zu Inkonsistenzen und falschen Daten führt (**race conditions**). Ein anderes Problem ist aber auch die *richtige* Synchronisierung, denn wenn z.B. zwei Prozesse eine Ressource vom jeweils anderen haben wollen, gibt es einen **Deadlock** und es geht nicht weiter, ohne dass ein Programm terminiert werden muss. Das können sich Angreifer auch zu Nutzen machen.

12.6 Interacting with the OS

Und als dritten großen Punkt gibt es da noch das Betriebssystem, das die Ressourcen unter den Programmen verteilt und diesen eine Umgebung bereitstellt, z.B. mit Umgebungsvariablen. Probleme hierbei ist die eben erwähnte Synchronisation beim gleichzeitigen Zugriff auf Ressourcen, aber auch die Rechtevergabe. Programme brauch eine bestimmte Menge an Rechten, um funktionieren zu können, man sollte ihnen aber auch nicht zu viele geben, damit nicht gleich das ganze System beschädigt werden kann, falls das Programm sich fehlerhaft verhält.

12.6.1 Environment Variables

Umgebungsvariablen, wie sie auf deutsch heißen, sind eine Sammlung von Strings, die vom bei der Erstellung vom übergeordneten Prozess geerbt werden. Diese können aber jederzeit geändert werden, womit die veränderten Werte an die Kinder weitergegeben werden, und man kann bei der Prozesserstellung auch explizit andere Werte einsetzen, weswegen man hier auch validieren muss.
Oft wird versucht, durch Skripte, die viele Rechte haben, selber diese Rechte zu erlangen.

Beispiel: Ein privilegiertes Shell-Skript wird ausgeführt, aber die `PATH`-Variable so verändert, dass aufgerufene Befehle, hier `grep`, sich nicht auf die Standard-UNIX-Programme beziehen, sondern auf das selbstgeschriebene Programm des Angreifers. So kann wirklich *jedes* Programm privilegiert ausgeführt werden (**privilege escalation**, und es ist auch nicht zu verhindern, weswegen von privilegierten Shell-Skripts abgeraten wird.

12.6.2 Vulnerable Compiled Programs

Auch schon kompilierte Programme sind auf diese Weise angreifbar, wenn sie die `PATH`- oder `LD_LIBRARY_PATH`-Variablen benutzen. Ersteres kann durch das Zurücksetzen auf einen bekannten, sicheren Wert verhindert werden, zweiteres durch statisches Linken (anstelle des dynamischen, was durch die Tabellen im Angegebenen Pfad ermöglicht wird).

12.6.3 Use of Least Privilege

Genau deshalb sollte man Programme gerade mal mit so vielen Rechten ausführen, wie für das Funktionieren nötig sind und ihnen nur Zugang zu Verzeichnissen genehmigen, die sie auch wirklich brauchen. Bei der Wahl der richtigen Benutzer- und Gruppenprivilegien sollte man zuerst damit anfangen, letzeres zu erhöhen, es könnte aber nicht ausreichend sein.

Viele Programme, die root-Reche benötigen, tun das nur direkt beim Start, wie zum Beispiel irgendein Server, der sich an die Ports bindet. Deshalb sollte man Programme auch in kleinere Module aufteilen, für die jeweils einzeln die Privilegien verteilt werden.

12.6.4 System Calls and Standard Library Functions

Programme funktionieren über diese Funktionen, die das Betriebssystem bereitstellt. Aber es laufen meist mehrere Programme nebeneinander, sodass das OS versuchen wird, die Vorgänge zu optimieren, wodurch es wieder zu unerwarteten Fehlern kommen kann.

Beispiel: Das Beispiel hier bezieht sich leider auf das Buch, aber im Grunde geht es darum, dass es schwer ist, ein wirklich sicheres Löschen zu implementieren, da es ziemlich viele unüberprüfte Annahmen über Systemfunktionen gibt.

12.6.5 Race Conditions

Das Prinzip der Race Conditions wurde weiter oben schon beim Speicher behandelt; hier geht es nun um Dateien, auf die gleichzeitig zugegriffen werden soll. Um Inkonsistenzen zu vermeiden, werden solche Dateien gelockt, indem z.B. ein **Lockfile** erstellt wird, das einfach nur andeutet, dass die Datei gerade in Verwendung ist. Andere Prozesse müssen dann erst die Existenz eines Lockfiles überprüfen.

12.6.6 Safe Temporary Files

Das gegenteilige Problem zum vorherigen ist der exklusive Zugriff auf eine Datei, in diesem Fall die temporäre Datei, die viele Programme zum arbeiten benötigen, weil sie meistens in einem Sammelverzeichnis für solche Dateien liegt.

Oft wird einfach der Process Identififier in die Namensgebung mit einbezogen, sodass der Dateiname einzigartig ist. Wenn man nun noch überprüft, ob nicht vielleicht von einem gecrashten Prozess desselben Programms eine temporäre Datei übrig ist, wäre man zwar was die Funktionalität betrifft auf der sicheren Seite, aber nicht vom Standpunkt der Security aus. Der Angreifer könnte den Namen der temporären Datei erraten und zwischen der Überprüfung und der eigentlichen Erstellung schon seine eigene Datei einschleusen.

12.6.7 Other Program Interaction

Auch die Interaktion zwischen Programmen kann eine Lücke darstellen. Wenn das Programm andere Programme und Dienste aufruft, um den Rückgabewert weiterzuverwenden, müssen, wie bei jedem Input, die Ergebnisse auf ihre Gültigkeit überprüft werden.

Aber auch den Datenfluss sollte überprüft werden, z.B. über die eben besprochenen temporären Dateien, oder bei externen Aufrufen eine sichere Übertragung wie SSL oder SSH benutzt werden.

Außerdem sollten Prozesse auch darauf achten, dass die von ihnen erstellen Prozesse selber korrekt terminieren bzw. die Fehler behandeln.

12.6.8 Handling Program Output

Warum der Output ebenfalls überprüft werden muss, sieht man z.B. bei unserem XSS-Beispiel. Dort sieht man auch, dass durch Veränderung des Outputs nicht das Programm selber, sondern das „output device“ angegriffen wird. Logischerweise müssen Output-Daten auch danach überprüft werden müssen, ob sie auch der Erwartung entsprechen.

13 Physische und Infrastruktur Security

Grundsätzlich gibt es drei Arten von Informations System (IS) Security:

- **Logische Security**: Schutz vor Daten-, Software- und Kommunikationsbezogenen Threats.
- **Physische Security**: Auch Infrastruktur Security, Schutz der IS-bedienenden und wartenden Leuten. Dieser Schutz muss auch jegliche Art physischen Zugriffs verhindern, der die logische Security beinflussen könnte.
- **„Premises“ Security**: Auch Firmen- oder „Fabrik“ Security. Soll Leute und Besitztümer in einem bestimmten Bereich (oder Gebäude) beschützen. (Möglicherweise auch gesetzlich vorgeschrieben) Zugriffskontrolle, Feuermelder, Wachpersonal, ...

13.1 Physische Security

Involviert zwei grundsätzliche Anforderungen:

1. Schaden an systemerhaltenden Komponenten muss verhindert werden. (Dies kann Diverses sein: Data-Processing und Storage Equipment (hardware), Gebäude (physische Einrichtung), Stromversorgung (unterstützende Einrichtungen), Personal ...)
2. Missbrauch der physischen Infrastruktur oder Beschädigung geschützer Daten führt muss verhindert werden.

Das Hauptanliegen physischer Security ist der Schutz der Informations Assets einer Organisation. Ein weiterer Beteiligter ist die Logische Security: Protokoll und Software die Integrität der Informations Assets bewahren.
Die Rolle der physikalischen Security ist hierbei gesondert zu beachten, da sie von Portabilität des System abhängt. (Statisches IS in einer Firma gegen Mobiles System in einem Fahrzeug o.ä.)

13.2 Arten von Bedrohungen

1. **Naturkatastrophen**: „Hurricanes“„Erdbeben“ und auch „Überflutungen“.
2. **Umweltbedingten Bedrohungen**: Ungünstige Temperatur/Feuchtigkeit, Feuer und Rauch, Chemisch-Biologische Verseuchung, Staub und sogar Befall (Ungeziefer, Nagetiere und Schimmel.)

   Maßnahmen zur Schadensbegrenzung: Richtiger Raum und womöglich auch technische Einrichtungen die Temperatur und Luftfeuchtigkeit überwachen und möglicherweiße korregieren / alamieren. Gegen Feuer und Rauch natürlich einhaltung bestimmter Brandschutzbestimmungen und Rauchdetektoren. Gegen Flutungen wieder Sensoren, die entweder Flutungsquelle stoppen oder zumindest die Stromversorgung kappen. Bei Chemisch-Biologisch auch Sensoren. Gegen Staub Filter und Reinung durch entsprechendes Personal. Befall durch Ungeziefer: Kontrollen.
3. **Technische Bedrohung**:

- **elektronische Spannung/Stärke**: **Unterspannung** (Meist kein Schaden, dafür Serviceausfall), **Überspannung** (durch Kabeldefekte oder Blitzeinschlag - Wenn kräftig genug können Silikon-basierte Komponenten wie Speicher etc. beschädigt werden), **Lärm/Störgeräusche** (z.B. von Hochspannungsleitungen, kann Filterungs-Verkabelungen der (internen) Stromversorgung „überstehen“ und darauf mit elektischen Signalen in Geräten interferieren und so Störungen verursachen)
- **elektro-magnetische Interferenz**: Kann von Hochspannungsleitungen, großen elektrischen Geräten und sogar anderen Computern ausgehen. Kann auch frei über Raum übertragen werden und kann mit ebenfalls mit technischen Geräten interferieren.

Maßnahmen zur Schadensbegrenzung: Eine Ununterbrechbare Stromversorgung (UPS en.) kann installiert werden um kürzere Ausfälle zu kompensieren, "Lärmßu Filtern und um als Notstromschalter zu agieren. Gegen elektro-magnetische Interferenzen hilft logischerweise richtige Abschirmung und Filterung.

4. **menschen-verursachte Bedrohungen**: Sind schwerer zu vermeiden, da sie weniger Vorraussagbar sind und meist vom bedrohenden Akteur so geplant sind, dass sie möglicherweise existente Schutzmaßnahmen umgehen.
   Unteranderem: Unerlaubter PHYSISCHER Zugriff (z.b. nur autorisierte Leute in den Server-Räumen, KANN ZU WEITEREN MENSCHEN-VERURSACHTEN BEDROHUNGEN FÜHREN), Diebstahl von Equipment oder Daten, Vandalismus von Equipment und Daten, Missbrauch von Ressourcen (Autorisiert oder auch nicht).
   Alle menschen-verursachten Bedrohungen können von In- als auch Außenseitern ausgehen.
   Maßnahmen zur Schadensbegrenzung: Physikalische Zugangskontrolle (Wachen, Abgeschlossene Türen, Überwachungsmaßnahmen), feste Installation technischer Hardware (gegen Diebstahl), twingend portable Geräte sollten über Tracking-Funktionalität verfügen, Bewegungssensoren, Alarmanlagen und alles was sonst noch sinnvoll erscheint.

13.3 Recovery im Falle eines physikalischen Einbruchs

- **Redundanz**: Im Falle von Beschädigung oder Diebstahl sind Kopien von Daten bzw. Ersatzsystem essenziell. Idealerweise off-site, und aktualisierung der Daten so oft wie verkraftbar. Dank breitband Internet möglicherweise per VPN über das Internet hals "batch"verschlüsseltes BackUp auf einem entfernten Rechner. EXTREM (FUCK YEA.) LÖSUNG: Eine "hot site"kann erstellt werden, die wenn benötigt sofort die Rolle der betroffenen Daten / des betroffenen Gerätes einnehmen kann.
- **Equipment Schadens-Recovery**: Falls durch Feuer oder sonstwas Equipment beschädigt wurde, sollte dieses natürlich entfernt werden. Womöglich sind dafür besondere Reinigungskräfte von Nöten.

13.4 Phyiscal Security - Plan

13.4.1 Threat Begutachtung

Der erste Schritt zur entwicklung eines Physical Security - Planes ist sich Gedanken über mögliche System-Schwachstellen zu machen. Hierzu sollte folgendes Realisiert werden:

1. Erstellung eines Steuerungs-Komitees (Unter Einbezugnahme aller am IS beteiligten Personen)
2. Beschaffung von Information und Unterstützung (Expertenbefragung, Sammeln von Daten früherer „Vorkomnisse“/Threats)

3. Identifizierung aller möglicher Threats
4. Bestimmung der Wahrscheinlichkeit der Ausnutzung aller identifzierter Threats (mit Hilfe von Informationen aus Punkt 2)
5. Durchführen einer (direkten) Kostenabschätzung
6. Erwägung kaskadierender Kosten (ausgehend von daraus folgenden (?, consequential) Threats, die weitere Kosten verursachen)
7. Priorisierung bestimmter Threats
8. Fertigstellung des Gutachtens

13.4.2 Planung und Implementierung

Wenn das Gutachten erstellt wurde kann ein Plan zur Vermeidung, Milderung und Recovery bei PS-Verletzungen entwickelt werden. Typischerweise erfolgt dies in folgenden Schritten:

1. Beurteilung innerer und äußerer Ressourcen (Ich denke hier sind mit „(“Ressourcen) mögliche Ansätze/Techniken gemeint um Angriffe zu Verhindern) z.B. nach Schulnoten (von 1 - starke Möglichkeit zur Verhinderung/Milderung, bis 5)
2. Identifizierung von Aufgaben und priorisierung von „(“Aktionen) (Festlegung welche Schwachstellen wie beseitigt werden sollen, Aufstellen von angestrebten Zielen und Meilensteinen, Erstellung einer Liste was von wem erledigt werden soll)
3. Entwicklung eines Plans (Der Plan enthält (sozusagen) alle „(“Dinge) die man zur Sicherung der Einrichtung und des IS nun ergreifen will: Support-Dokumente, Notfalls-Anruflisten, Karten von Gebäuden und Standorten, Ressourcen Listen und Listen über mögliches Equipment zur Vermeidung, Milderung und Recovery)
4. Implementations des Plans, benötigt regelmäßige Aktualisierung (Anschaffung benötigten Equipments, Vergabe von Verantwortungen, ...)

13.5 BILD: Example Policy

Beispiel zu so einer Policy, kann man sich ja ansehen wenn man mag. Wichtig ist es nicht.

13.6 Physische/Logische Security Integration

Physische Security verfügt meist über eine Unzahl an Sensoren und Alarmen, „(“Vermeidung-Devices) etc. Hier gibt es natürlich viel Raum für Automatisierung und Integration von Computersystemen. Sehr interessant auch zur Realisierung eines zentralen Standorts von dem aus das System kontrolliert werden kann. (Sowohl für Kosten als auch für Sicherheit relevant) Am besten ist sowas natürlich im Rahmen von Zugriffs und Zugangskontrolle möglich. Für so Dinger wie Smart-Card Leser, Zugriffskontroll-Formate etc. gibt es eine Reihe an unterschiedlichen Herstellern. Eine Standarisierung ist nun natürlich anzustreben. Wichtiger Schritt in die Richtung: FIPS-201-1 „(“Personal Identity Verification (PIV) of Federal Employees and Contractors).

13.6.1 BILD: PIV

Das PIV Front-End ist gegen einen Benutzer gerichtet der Zugang zu bestimmte Logische Daten bzw. Bereichen oder Räumen einer Einrichtungen erlangen will. Das PIV Front-End Subsystem unterstützt dreifache Authentifizierung, wobei der tatsächliche Einsatz der einzelnen Komponenten von dem Sicherheitslevel des gewollten Zugangs abhängt. Unterstützt wird Smart-Card (PIV-Card), PIN-Eingabe und Biometrische-Scans (im derzeitigen Standard ein Fingerabdruck-leser).
Die andere Hauptkomponente des PIV-Systems ist das PIV-Kartenausgabe und Verwaltung Subsystem. Dieses Subsystem kümmert sich um die Registrierung neuer Benutzer und um die Identitätsüberprüfung und damit verbundene Services (z.B. public key infrastructure [PKI] directory, certificate status servers). PIV interagiert sodann mit einem Zugriffskontroll-Subsystem, was letztendlich den Zugriff eines bestimmten PIV-Cardholders auf die jeweiligen Ressourcen bestimmt.
Ein weiteres Bild stellt nun dein Aufbau eines solchen PIV-Systemes dar. Inhaltlich nicht wirklich wertvoll, aber ganz interessant da es diverse Akteure in einem solchen System aufzeigt, wobei das PIV-System in der „(“Mitte) agiert.

14 Menschliche Faktoren

14.1 Security Bewusstsein, Training und Ausbildung

Wird prominent in diversen Standards und damit verbunden Dokumenten behandelt. (z.B. ISO 17799). Vorteil zieht man hierbei aus der Verbesserung des Verhalten der Mitarbeiter, der Erhöhung der Fähigkeit Mitarbeiter für ihre Taten verantwortlich zu machen (Milderung der Haftung der Firmen) und dem Entsprechen von Regulationen und vertraglich festgelegten Verpflichtungen.

Die Lernziele für einen Mitarbeiter in Bezug auf Sicherheit hängen von dessen Rolle ab. Lernprogramme starten mit Bewusstsein, bauen auf zu Training und evolvieren in Ausbildung. Ein Modell hierzu (BILD) besteht aus vier Schichten die von der NIST wie folgt zusammengefasst werden (von unten nach oben):

1. Security Bewusstsein, notwendig für alle Mitarbeiter
2. Security Basics und Bildung ist eine transitionale Stufe zwischen Bewusstsein und Training. Bildet sozusagen die Basis für folgendes Training.
3. Die Schicht „Rollen und Verantwortlichkeiten relativ zu IT Systemen“ fokussiert an der Bereitstellung von Wissen, Fähigkeit und Möglichkeiten für Indivuden in deren Jobpositionen
4. Ausbildung und Erfahrung fokussiert die Fähigkeiten zu entwickeln, die gebraucht werden, um mit weiteren technologischen Entwicklungen Schritt zu halten

14.1.1 Bewusstsein

Soll die Mitarbeiter über Security Problemfelder (Verwundbarkeiten, Verantwortung, ...) in der Organisation unterrichten. Wie Bewusstsein-Programme aussehen, richtet sich natürlich nach Organisation und Audienz. (Manager, normaler Mitarbeiter) Außerdem sollten diese Programme in unterschiedlichen Formen an die Mitarbeiter gebracht werden, z.B. als Evente, Newsletter, Poster oder Briefings. Um die Wichtigkeit von Security Bewusstsein zu konkretisieren sollte eine Organisation über ein Security-Bewusstseins-Richtlinien(Policy)-Dokument verfügen.

14.1.2 Training

Ein Security Trainig Porgramm ist ausgelegt, um Mitarbeitern die Fähigkeiten zu geben ihre IS-bezogenen Aufgaben sicherer zu erledigen. Wie dieses Training abläuft hängt wieder von der Rolle des Benutzers ab der dieses erhält. Kann vom simplen Computer Fähigkeiten (**normaler Mitarbeiter**: Wie schützt man Passwort möglichst gut? Wie meldet man Security Verletzungen oder Vorfälle?) bis zu super-1337en-hax0r Skillz (**Programmierer, Entwickler und Adminstratoren**: zeigen wie man Security in Entwicklungs-Zyklen integriert, mögliche Angriffsarten unterrichten, usw.) reichen. Passiert aber auch abseits im **Management** bzw. in der **Exekutive**, wo möglicherweise Risiken, Kosten und Vorteile von Security und Security Evaluations-Techniken behandelt werden.

14.1.3 Ausbildung

Security Ausbildung geht am meisten in die Tiefe und zielt eher auf Security Professional ab. Daher geschieht diese Ausbildung meistens im Rahmen von Mitarbeiter-Karrierenentwicklungs-Programmen.

14.2 Betriebs Security-Richtlinien(Policies

Definition: Eine Security-Richtline(Policy) ist eine formaler Ausdruck der Regeln, denen Personen die Zugriff auf Technologien und Assets der Organisation haben folgeleisten müssen. Wird aber auch in andern Kontext verwendet.

Ein schriftlich verfasstes Security-Richitlinien-Dokument ist fundamental um erwartetes Verhalten zu definieren. Außerdem bietet es unterstützung für IT-staff und lower-level Manager, um high-level Manager für möglichen Bedarf an Erweiterung bzw. Anschaffung von Ressourcen zu überzeugen. Eine solche Richtlinien macht klar was warum beschützt wird, definiert Prozeduren und Standards innerhalb der Organisation und beschreibt eindeutig die Verantwortlichkeiten. Somit ist auch eine Basis auf welcher spätere Konflike interpretiert und gelöst werden können geboten.
Um diese Rollen zu erfüllen muss die Security-Richtline Entscheidungen des Exekutiv-Managements reflektieren. Daher müssen relevante Gesetze und kritische bzw. sensitive Komponenten identifiziert und Ziele und Mechanismen um diese zu befriedigen definiert werden.

Der **Lebenszyklus einer Sicherheits-Richtlinie** (BILD) sieht wie folgt aus: Zuerst wird eine Risiko-Analyse betrieben, dann wird darauf basierend eine Sicherheits-Richtlinie entwickelt, welche dann dem Exekutiv-Management und bestimmten Abteilungen (IS-Technischer Staff, Admins, User-Gruppen) vorgetragen wird und von diesen bestätigt werden muss, dann wird das Sicherheits-Bewusstsein (um die Richtlinie) „ehoeht“ und letztendlich wird sie implementiert (durch Prozeduren und Mechanismen die in ihr selbst verfasst sind).
Nun ist die Richtlinie jedoch nicht finalisiert sondern Bedarf regelmäßiger Neubewertung.

Eine Security-Richtlinie sollte eine Reihe an Themen und Gebieten behandeln. Einige **substantielle Fragen** sollten hierbei beantwortet werden (Auszug): Warum wird die Policy entwickelt? Wer entwickelt die Policy? Wer bestätigt sie? Welche Autorität trägt sie? Auf welchen Gesetzen bzw. Regulationen beruht sie? Wer wird die Policy durchsetzen? Wenn betrifft sie? ..

Behandelte Themen (wieder ein Auszug - riesen Aufzählung über zwei Folien): Prinzipien z.b. eine Definiton von Information Security und die „overall“-Ziele; Bereiche und Wichtigkeit von Security, Physical Security - also physischer Schutz von Geräten und Einrichtungen (siehe Kapitel 13); Hardware - bestimmte Richtlinien die bestimmte Hardware favorisieren, verbieten und vorschreiben, weile diese z.B. bestimmte über technologische Security-Maßnahmen verfügen muss; auch Richtlinien für den Kauf von Software; Richtlinien auch in Bezug auf Anstellung und Feuerung von Mitarbeiten; .... Datenschutz

- sinnvolle Erwartungen an Datenschutz wie Überwachung von Mail etc. sollen klar definiert werden, Richtlinien zur Meldung von Security-Verletzung UND SO WEITER ... am besten selbst einmal durchlesen.

Es gibt einige populäre **Ressourcen** an denen man sich bei der Erstellung von Richtlinien orientieren bzw. diese direkt übernehmen kann, wie z.B. der bereits erwähnte ISO 17799 oder der „Standard of Good Practice for Information Security" vom Information Security Forum.

14.3 Personal Security

Personal Security beschäftigt sich mit Anstellung, Training, Verhaltensbeobachtung und Umgang mit Entlassung. Mitarbeiter können selbst auch in Security-Verletzungen involviert sein und dies sowohl absichtlich als auch unabsichtlich. Threats die von Mitarbeitern ausgehen: Veränderung von Daten, Zerstörung bzw. „crashing" von Systemen, unautorisiertes Erlangen von Rechten bzw. unautorisierte Weitergabe, ...

Folgende Ziele sollten (nach ISO 17799) **bei der Anstellung** berücksichtigt werden: Es soll versichert werden, dass Mitarbeiter, Vertrager (?, contractors) und dritt-Beteiligte ihre Verantwortungen verstehen, passend für die Rollen sind, für die sie in Betracht gezogen werden und das Risiko von Diebstahl, Betrug oder Missbrauch in den Fabriken/Einrichtungen reduzieren.
Für Security stellt die Anstellung neuer Mitarbeiter eine große Herausforderung dar. Bewerber stellen sich in ihrem Resumees gerne besser bzw. qualifzierter dar als sie eigentlich sind, außerdem äußern sich frühere Arbeitgeber der Bewerber, selbst wenn diese ausgesprochen inkompetent waren, ungern negativ über ihre ehemaligen Mitarbeiter aus Angst vor Anklagen. Um Sicherheitsrisiken zu vermeiden muss der Werber ausführlich und detailliert über frühere Anstellungen und Ausbildung befragt werden. Diese Informationen sollten dann so weit vertretbar überprüft werden. Außerdem sollten erfahren Mitarbeiter die Möglichkeit haben Kandiation selbst zu interviewen und Unstimmigkeiten zu diskutieren.
Wird eine Person letzendlich angestellt soll sie vertraglich bestätigen, dass sie mit den Termen und Konditionen des Anstellungsvertrags einverstanden sind. Dieser Vertrag behandelt einige Themengebiete wie z.b. die Security-Richtlinie (Policy), Geheimhaltungsversprechen, etc ...

Während der Anstellung sollte sichergestellt werden das Mitarbeiter, Verträger (contractors) und Dritt-Personen sich über Information-Security-Threats und damit einhergehenden Verpflichten ihrer Position bewusst sind. Verbesserung durch Förderung des Security-Bewusstseins und Training etc. Sonst gibt es bestimmte Prinzipien die bei Personal Security eingehalten werden sollten:

- Least privlege (Jeder hat gerade so viel (Zugriffs-)Rechte wie er braucht, um die seiner Anstellung entsprechenden Aufgaben zu Erfüllen)
- Seperation of duties: (Verteilung der Pflichten, (wirkt eher wie ein Beispiel) (Arbeits-)pflichten sollten so vorsichtig aufgeteilt werden, dass die Leute die die korrekte Ausführung einer Arbeit überprüfen, nicht selbst bei deren Ausführung beteiligt sind.)
- Limitiertes Vertrauen in Schlüsselpersonen (Kein Mitarbeiter einer Firma sollte unersetzlich sein. Es sollte keinen Mitarbeiter geben mit absolut einzigartigen Fähigkeiten. [Man denke hier an PC-System, fällt eine Komponente aus, ist es gut noch einen Ersatz zu haben])

Bei der **Entlassung** sollte sicher gestellt werden, dass alle Entitäten die die Firma verlassen oder deren Anstellung sich verändert diesen Übergang geordnet vollziehen: Komplettes Equipment zurückgegeben und alle Zugriffs(-Rechte) der Person im Betrieb zurückziehen. (auch Informierung des Wachpersonals, etc.)

14.4 Email und Internet Richtlinien

Aus der (natürlicherweise notwendigen) Benutzung von Email und Internet können einige Probleme resultieren wie z.B. schlechtere Effizienz am Arbeitsplatz (rumsurfen) und erhöhtes Risiko, dass bösartige Software ins System gelangt etc.

Auch für Richtlinien wieder einige Empfehlungen:

- Bussiness use only - nur geschäftliche Nutzung
- Datenschutz: Mitarbeiter dürfen keine Privatsphäre erwarten, wenn sie diese Medien innerhalb der Firma benutzen
- Nur vertretbare persönliche Nutzung: Mitarbeiter dürfen diese Medien auch persönlich für sich Nutzen, wenn diese nicht mit ihrer Aufgabe in der Firma interferieren, keine andere Richtlinien verletzen und keine Einrichtungen der Firma übermäßig belasten.
- Verbot gesetzeswidriger Benutzung

15 Rechtliche und ethische Aspekte

15.1 Cyber- und Computerverbrechen

Definition von Computerverbrechen: Verbrechen in denen Computer bzw. Computernetzwerke als Werkzeug, Ziel oder Handlungsraum dienen. (Cyberverbrechen = nur Netzwerk)

Dies geschieht als ... Das U.S. Rechtsministerium unterteilt Computerverbrechen in drei Sparten: Computer als ...

- Ziel (Information stehlen, unautorisierter Zugriff, Modifikation von Daten - Angriff auf Integrität)
- Speichergerät („Passiv“, Speicherung gestohlener Passwörter, Warez, ...)
- Kommunikationswerkzeug (Verkauf von Drogen, Betrug, Verteilung Kinderpornographischer Inhalte ... alles natürlich online)

15.2 BILD: Herausforderungen des/beim Gesetzvollzug

Ob Leute abgeschreckt sind Computervebrechen zu begehen, hängt natürlich vom Erfolg der Exekutive ab. Allerdings ist es für die Exekutive ziemlich schwer bei Computerverbrechen zu ermitteln. (Internationalität der Verbrechen, benötigte Experten und Technologien, etc.) Eine daraus resultierende Auswirkung ist die Anstieg der Kühnheit und Anzahl von Computerverbrechen.
Dieser „Erfolg“ der Computerverbrecher schwächt das Vertrauen der Opfer in die Exekutivorgane, daher kommt es auch kaum zu Berichterstattungeg und eheren Einwilligungen auf mögliche Forderungen von Computerverbrechern.

15.3 BILD: Geistiges Eigentum

Das U.S. Rechtssystem, bzw. Rechtssysteme generell unterscheiden primär zwischen drei Typen von Besitzrechten: echt, persönlich und geistig. Geistiges Eigentum an sich wird wieder in drei Kategorien gesplittet:

- Copyrights (Verbot unautorisierter Benutzung)

- Patente (Verbot unautorisierter Produktion, Verkauf und Benutzung)
- Trademarks (Handelsmarken) (Verbot unautorisierter Benutzung und „angeblicher Limitation“ [colorable limitation ... !?])

15.3.1 Copyright

Schützt den festgelegten, konkreten (Ja, genau ...) Ausdruck einer Idee, nicht die Idee selbst. Gilt in einem Land die Berner Konvention so ist jeder Ausdruck einer Idee sofort geschützt solang diese „original“ ist und vom Schaffer in eine konkrete Form gepackt wurde. (Schriftlich, Multimedia-Form, Software) Ist dem so, sind dem Schaffer folgende Rechte gegeben:

- Reproduktion
- Modifkation (darf Derivate „entwickeln“)
- Distribution (darf publizieren, verkaufen ...)
- Öffentliche-Vorstellung (Hauptsächlich bei Live-Vorstellungen [Vermutlich Shows oder sowas])
- Öffentliche-„Darstellung“ (Man denke an Filme, etc.)

15.3.2 Patente

Ein Patent (=für Erfindungen) verleiht dem Erfinder Besitzrechte zu seiner Erfindung. **Definition:** Nach U.S. Gesetzesbestimmung: „Das Recht andere von der Produktion, Benutzung, Verkaufsanbietung oder Verkauf auszuschließen“.
Es gibt drei Arten von Patenten:

- Nützlichkeits-Patente: Für jeden der etwas sinnvolles/nützliches entdeckt bzw. erfindet (Maschine, Prozess ...)
- Design-Patente: Für jeden der ein neues, originales Design erfindet.
- Pflanzen-Patente: Für jeden der eine neue Pflanzen-Art entdeckt oder erfindet und asexuell reproduziert.

15.3.3 Handelsmarke

Ist Wort, Name, Symbol der Gerät das von jemanden Benutzt wurde um beim Handel von Gütern sich als Urheber zu identifizieren, und von den anderen abzugrenzen. Servicmark = fast dasselbe wie trademark nur für Dienste und weniger für Produkte. Schützen echt nur das „Symbol“, wenn jemand gleiches Produkt unter anderem Zeichen setzt, ist das nach Handelsmarke okay.

15.4 Problemfelder geistigen Eigentums und Computer Security

Für Computer Security interessant:

- Software: Betrifft Hersteller von kommerzieller Software, als auch von Freeware, Shareware und Open-Source Zeug. Können auch einzelne Individuen sein. Zum Schutz meist Copyright interessant, ab und zu womöglich auch Patente.
- Datenbanken: Gesammelte Daten könnten womöglich auch geschützt werden wollen. Z.b. Datenbank die Wirtschaftsprognosen enthält könnte geCopyrighted werden.
- Digitaler Content: Filme, Audo Dateien etc.
- Algorithmen: Z.b. RSA public key. Patente!

15.4.1 U.S. Digital Millennium Copyright ACT (DMCA)

Der DMCA hat große Auswirkungen auf den Schutz digitalen Contents weltweit gehabt. Der DMCA soll die bereits früher gezeichneten Verträge der World Intellectual Property Organization (WIPO) Verträge implementieren. In seiner Essenz verstärkt der DMCA einfach digitale Copyrights. DMCA ermudigt Copyright-Besitzer Technologien einzusetzen um seine Rechte zu schützen. Hier wird zwischen zwei Kategorien separiert:

- Maßnahmen die Zugriff auf den Content verhindern
- Maßnahmen die Kopierung des Contens verhindern

Außerdem verbietete das Gesetz die Entwicklung von Technologien die versuchen derartige Schutzmaßnahmen zu verhindern. Ein paar Aktionen sind aber ausgenommen vom DMCA:

- Fair Use: Nicht genau definiert: Bestimmte Teile des Werkes können zitiert, vorgeführt bzw und oder kopiert werden. Z.b. Für Reviews, Kommentare oder Diskussionen
- Reverse engineering: Ist erlaubt solange der Anwender eine legale Kopie des Programm hat und nicht darauf abzielt die Funktion des Programmes zu kopieren o.ä.
- Encryption Research: Erlaubt solang man nur die Qualität bzw. den Fortschritt solcher Algorithmen verbessern will. (FOR SIENCE!!!)
- Security testing: wenn man nur in guten Willen nach Sicherheitslücken testet
- Privatssphäre: Man darf Schutzmaßnahmen umgehen, wenns die einzige vertretbare Möglichkeit ist private Daten zu schützen.

Trotzdem gibt's ziemliche Bedenken, dass der DMCA legitimen security/crypto-research verhindert.

15.4.2 Digital Rights Management - DRM

DRM referiert Systeme und Prozeduren die sicher stellen das die jeweiligen Copyright-Halter eindeutig identifiziert werden und die entsprechende „Bezahlung" für ihre Werke erhalten haben. DRM kann die Nutzung digitaler Inhalte natürlich weiter einschränken. DRM Standards oder Architekturen gibt's nicht. DRM steht eben für eine Vielzahl an Herangehensweisen.
Ziel von DRM ist es nur autorisierten Personen(und womöglich auch nur limitiert) Zugriff auf den Content zu geben, und dieses über eine Vielzahl unterschiedlicher Plattformen, Medien und Content-typen.

Die **Komponenten (BILD)** bei DRM fallen wie folgt aus: Es gibt einen **Content-Provider** der die Rechte besitzt und diese auch geschützt haben will, einen **Distributor** der den Content unters Volk bringt, einen **Konsumenten** der über das System auf die geschützen digitalen Werke (von Distributor) zugreift und dessen Player/Viewer-Programm sich um die Lizensierung mit Hilfe der **Verrechnungsstelle**, die sich um eben um Lizenzierung und Bezahlung von Provider und Distributor sorgt, kümmert.

Bei der **Architektur (BILD)** wird das auf System von drei Parteien in drei Rollen zugegriffen. Copyright-Halter sind wiedermals die, die den Content gemacht bzw. sich Rechte dazu geholt haben. Service-Provide sind die Distributoren und Verrechnungsstellen. Konsumenten sind die, die das Recht kaufen auf bestimmten Content auf eine bestimmte Art zuzugreifen. Das DRM-System bietet drei Schnittstellen zu seinen Diensten: Identitäts-Management (Mechanismen für einzigartige Entitäten, z.B. bestimmte Parteien oder Content), Content-Management (Funktionen um den Content-Lebenzyklus zu verwalten) und Rechts-Management (Funkitonen um Rechte und Rechtvergebung zu verwalten). Unter dieser Schnittstelle sind die gewohnten System-Funktionen zur Verschlüsselung, Autorisierung, Bezahlung und Zustellung.

15.5 Datenschutz

Überlappt mit Computer Security, da Daten jedes Einzelnen heutzutage eben von Regierungen und Wirtschaft stärker gesammelt und gespeichert werden. Zugleich sind sich indviduelle Personen auch stärker über die Verfügbarkeit ihrer Persönlichen Daten bewusst geworden.

15.5.1 Datenschutz-Gesetze

In der **EU** wurde 1998 die Data Protection Directive beschlossen die sicherstellt, dass Mitgliedstaaten bei der Verarbeitung persönlicher Informationen die fundamentalen Rechte der Privatsphäre und das Recht auf freien persönlicher Informationsfluss in der EU respektieren. Im konkreten müssen Benutzer bescheid wissen welche Daten gesammelt werden und in der Lage sein selbst zu steuern wie mit diesen umgegangen wird.

In den **U.S.** gibts den Privacy Act seit 1974, welcher über den Umgang von Bundesorganen mit persönlichen Daten regelt. Inhaltlich der von der EU sehr ähnlich. In den U.S. gibts nun eine Reihe weiterer (ähnlicher) Gesetze die Datenschutz auch in der Medizin, Elektronischen Kommunikation etc. regeln.

15.5.2 Die „Common Criteria Privatsphäre/Datenschutz-Klasse“ (BILD)

Die Common Criteria Spezifikation definiert einen Satz funktionaler Anforderungen, die in einem vertrauenswürdigen System implementiert werden sollten. Sie soll einen Nutzer vor Entdeckung bzw. Missbrauch seiner Identität von anderen Nutzern schützen. Hierbei geht's aber eher um den Schutz der System-Ressourcen eines Benutzers, nicht der persönlichen Informationen.

Hierbei gibt's vier Hauptbereiche:

- **Anonymität**: Versichert, dass ein Benutzer Systemressourcen verwenden kann, ohne dabei seine Identität zu verraten.
- **Pseudonymität**: Trotz dem vorherigen Punkt sollen Benutzer aber immer noch „Verantwortlich“ (accountable) für ihre Aktionen gemacht werden können.
- **Unlinkability**: Benutzer können diverse Ressourcen und Services benutzen ohne das andere diese „zusammenlinken“ können.
- **Unobservability**: Benutzer können diverse Ressourcen und Services nutzen, ohne dass jemand (ein Dritter) in der Lage ist zu observieren ob die entsprechende Ressource benutzt wird, oder nicht.

15.5.3 Datenschutz und Datenüberwachung (BILD)

Die Anforderungen von Heimatschutz and Anti-Terorismus haben neuartige Bedrohungsformen in Sachen Datenschutz entstehen lassen. Firmen und Regierungen haben begonnen wesentlich agressiver Daten zu sammeln ud zu überwachen.Im Sinne einer technischen Herangehensweise befinden wir uns hierbei im Bereich der Datenbank Security.

Das hier gegebene Bild demonstriert eine Datenschutz-Vorrichtung die zwischen Datenbank und Zugriffsschnittstelle operiert. Es verfügt über eine Reihe von Funktionen: **Daten-Transformation** = Cryptet bestimmte Portionen der Daten, so dass Datenschutz sichergestellt ist, aber die Daten noch gut benutzbar sind; **Anonymisierung** = entfernt bestimmte Informationen aus Query-Resultaten und ersetzt diese mit einzigerartigen, jedoch anonymen Identifiern; **UNVERÄNDERLICHES HARDCORE**

Audit/Logging = trivial; **Assoziativer Speicher** = Software die Muster bzw. Speicher zwischen Datenstücken erkennt, die der Menschen vielleicht nicht erkannt hat.

15.6 Ethische Problemfelder

Aufgrund der Allgegenwärtigkeit und Wichtigkeit von Informationssystemen gibt es auch viele Möglichkeiten für Missbrauch. Die ethische Frage beim Umgang mit viel Infomation ist nicht neu und die gängigen Prinzipien können angewendet werden. Allerdings muss man beachten, dass durch Informationssysteme eine ganz andere Größenordnung von Speicherung, Datenaustausch und vor allem der Durchsuchung dieser Daten ermöglicht wird, sodass auch der Missbrauch erleichtert wird. Außerdem gibt es für viele neue Erfindungen noch keine ethischen Regeln, wie z.B. bei Datenbanken, Cookies und Chats.

Nun einige ethische Problemfelder die durch Computertechnologie entstehen:

- **Repositorien und Prozessoren persönlicher Information**: Unautorisierte Nutzung der Daten bzw. der Datenverarbeitung.
- **Produzenten neuer Formen und Typen von Assets**: Z.B. Computerprogramme die völlig neue Arten an Assets sind, die Möglicherweise nicht den selben„Konzepten des Besitzes“ unterliegen als andere Assets.
- **Werkzeuge von Taten (?, instrument of acts**: Bis zu welchen Grad müssen Computer Dienste und Benutzer von Computer, Daten, etc. verantwortlich für die Integrität des computer outputs sein?
- **Symbole von Bedrohung und Betrug**: Der Computer sollte als Symbol der Denkmaschine und absoluter Wahrheitsproduzent und als menschenähnlicher Ersatz von Menschen vorsichtig betrachtet werden.

15.6.1 Ethische Hierarchie (BILD)

Leute mit speziellem Wissen oder Fähigkeiten haben immer besondere Verpflichtungen gegenüber den (ich sag hier mal) normalen Bürger. Hierzu kann man eine tolle Pyramdie erstellen: An der Spitze sind die typischen ethischen Werte die Professionelle mit allen Menschen teilen, wie Integrität, Fairness und Gerechtigkeit. Wenn man ein Professioneller (generell) ist hat man besondere Verpflichtung gegenüber denen, die vom eigenen Schaffen betroffen sind. Letztlich hat jede Profession (speziell) seine eigenen ethischen Werte und Verplichtungen die mit der bestimmten Art von Wissen einhergehen.

15.6.2 Beispiele ethischer Fragen

Zwei Beispiele sind hier gegeben: Beim ersten geht's um einen IS-Professional der in seiner Firma mit ethischen Pflichten in Konflikt zur Loyalität als Angestellter kommt. (Z.B. weil es zu einer ethisch nicht vereinbaren Entwicklung im Betrieb kommt) Hier kann es dazu kommen das der Professionelle sich entscheidet die „whistle zu blowen“. Das andere Beispiel betrifft einen potentielle Interessenkonflikt. Zum Beispiel, wenn ein Konsultant finanzielles Interesse für einen bestimmten Hersteller hat, sollte das jedem Klienten offenbart werden, wenn die Produkte des Herstellers möglicherweise von den Konsulanten empfohlen werden. KEINE AHNUNG WAS DAMIT GEMEINT IST! (For example, if a consultant has a financial interest in a certain vendor, this should be revealed to any client if that vendor's products or services might be recommended by the consultant.)

15.6.3 Verhaltensmaßregeln oder Codes of Conduct

Ethik kann nicht auf präzise Gesetze oder Fakt-sätze reduziert werden. Trotzdem sollte der Klient eines Professionellen von diesem Erwarten können, dass er über einen bestimmten moralischen Kompass

verfügt. Einige Profesionellen-Gesellschaften (IEEE, ACM, AITP) haben bestimmte ethische Verhaltensmaßregeln adaptiert. Diese Verhaltensmaßregeln dienen als positiverAnsporn für ethisches agieren der Profesionellen, verstärken das öffentliche Bild der Profession und geben Professionellen-Gesellschaften eine Möglichkeit den Kodex als Rechtfertigung, um Mitgliedschaft bzw. die Vergegebung Professioneller-Lizenzen zu verweigern.

Solche Kodizes behandeln meist folgende Felder:

- Würde und Wert andere Leute
- Persönliche Integrität und Ehrlichkeit
- Verantwort zur verrichteten Arbeit
- Diskretion von Information
- Öffentliche Sicherheit, Gesundheit und Wohlergehen
- Beteiligung in Professionellen-Gesellschaften um Standards der Profession zu Verbessern
- Die Ansicht das öffentliches Wissen und Zugang zu Technologie gleichgestellt mit sozialer Macht ist

Die Kodizes legen ihren Schwerpunkt auf die Verantwortung Professioneller gegenüber anderen Menschen.

16 Security Auditing

„Eine unabhängige Untersuchung der Aktivitäten eines Systems, um sicherzustellen, dass es sich richtig verhält und keine Sicherheitslöcher aufweist. Außerdem sollen auch sicherheitsrelevante Änderungen vorgeschlagen werden.
Zu diesem Zweck soll ein **Audit Trail** *angelegt werden, um die Vorgänge des Systems auf Angriffe analysieren zu können.“*

Audit Trail: „Ein chronologischer Bericht über die Systemvorgänge, der die Rekonstruktion und Überprüfung eines Systems zu dem Zeitpunkt einer bestimmten Operation zulässt.“

16.1 Security Audit Architecture

An dieser Stelle ist ein Bild in den Folien, was man sich ansehen sollte. Trotzdem hier die Aufzählung der einzelnen Bestandteile eines solchen Systems:

- **Event Discriminator:** Eine Logik, die die Systemaktivitäten überwacht und sicherheitsrelevante Ereignisse, auf deren Entdeckung sie programmiert ist, meldet.
- **Audit Recorder:** Empfängt Meldungen des Event Discriminators.
- **Alarm Processor:** Einige Ereignisse sind aber als Alarm Events deklariert und werden an den Alarm Processor gesendet, der auf eine vorbestimmte Weise tätig wird.
- **Security Audit Trail:** Eine Liste der Ereignisse, erstellt vom Audit Recorder.
- **Audit Analyzer:** Analysiert die daten im Audit Trail und erstellt eventuell neue Regeln für wichtige Ereignisse und Alarme.
- **Audit Archiver:** Archiviert die Daten des Audit Trail.

- **Audit Provider:** Gibt auf Anfrage Daten aus dem Audit Trail aus.
- **Audit Trail Examiner:** Nimmt über den Audit Provider Daten aus dem Audit Trail und analysiert diese nach bestimmten Gesichtspunkten, was in einen *Security Report* geschrieben wird.

16.1.1 Distributed Audit Trail Model

Auf verteilten Systemen kann auch eine Software, die verteilt auf diesen arbeitet, sinnvoll sein (wie schon z.B. bei den Intrusion Detection Systems). Dazu braucht man zusätzlich zwei Komponenten, die eigentlich logisch sind:

- **Audit Dispatcher:** Dieses auf jedem System installierte Modul überträgt die Berichte an einen zentralen Server.
- **Audit Trail Collector:** Das ist das zugehörige Modul auf dem zentralen Server, das die Daten der Audit Dispatcher sammelt und zusammeführt.

16.1.2 Security Auditing Functions

Auch hier gibt es ein Bild. Bloß ein alternatives Modell (*Common Criteria Specification*).

- **Data Generation:** Stellt fest, zu welchem Level überprüft wird, und listet alle überprüfbaren Ereignisse auf.
- **Event Selection:** Hier wird festgesetzt, welche Vorgänge überprüft werden, und welche nicht.
- **Event Storage:** Erstellung und Verwaltung des Secure Audit Trail, darunter fällt das Verfügbarmachen und Backups.
- **Automatic Response:** Die zu tätigenden Aktionen, falls ein möglicher Eindringungsversuch entdeckt wird.
- **Audit Analysis:** Automatisches Untersuchen der Systemaktivitäten auf vorher definierte verdächtige Ereignisse.
- **Audit Review:** Unterstützt autorisierte Benutzer beim Durchsuchen der Aufzeichnungen, z.B. durch Filtern.

16.1.3 Event Definition

Man muss natürlich spezifizieren, was genau geloggt werden soll. Gängige aufzuzeichnende Ereignisse sind das Erstellen und Löschen von Objekten, Änderung von Zugriffsrechten, Authentifikationen, „security-related actions“ und Import/Export von Daten von/nach externen Speichern. Fernzugriff, das Benutzen ganz bestimmter Applikationen, System Calls und auch Ereignisse von IDS oder Firewalls können auch ein aufzuzeichnendes Ereignis sein.

16.1.4 Other Audit Requirements

Zusammenfassend kann man sagen, dass um ein solches Überwachungssystem umsetzen zu können, natürlich einige Voraussetzungen erfüllt sein müssen. Es muss Möglichkeiten zur **event detection** geben und zum **event recording**, diverse Tools und Interfaces zur **event and audit trail analysis**, sowie eine gewisse Sicherheit dafür. Man sollte die Logs so speichern, dass sie nicht verändert werden können, aber auch die Software im generellen sollte unzugänglich sein. Letztlich ist es auch ein Vorteil, wenn es die Funktionalität nicht beeinträchtigt.

16.1.5 Implementation Requirements

An dieser Stelle erfolgt eine Aufzählung von Implementierungsempfehlungen aus dem ISO-Standard, aber das ist entweder logisch oder schon genannt worden.

16.2 What to Collect

Da große zu sammelnde (und somit laufend zu analysierende) Datenmengen eine Verlangsamung des Systems bedeuten und die Security Reports unüberschaubar machen, sollte man genau auswählen, was zu überwachen ist. Beispiele für Events finden sich weiter oben. Auch ist es sinnvoll, die Logs in verschiedene Kategorien einzuteilen.

16.2.1 System-Level Audit Trails

Mit solchen Logs wird eher versucht, die Performanz zu verbessern, kann aber auch der Sicherheit dienen. Festgehalten werden normalerweise Logins oder Loginversuche, benutzte Geräte und Betriebssystemfunktionen, aber Netzwerkverhalten könnte z.B. auch von Interesse sein.

16.2.2 Application-Level Audit Trails

Solche Logs sind, was der Name vielleicht nicht gleich sagt, sogar *applikationsspezifisch*. Aufgezeichnete Events sind also vom Programm abhängig. Bei einem Datenbankserver könnten es der User und die Anfragen sein; bei einem Mailserver Absender, Empfänger und Anhänge. Dies ist auch sinnvoll, um eventuelle Lücken des Programms aufzudecken.

16.2.3 User-Level Auti Trails

Hiermit wird das Verhalten einzelner User aufgezeichnet. Damit kann man sie zur Verantwortung ziehen, aber auch ein Analysetool füttern, das wissen soll, wie normales Userverhalten aussieht. Im Log kann alles stehen; von benutzten Befehlen und Programmen über Authentifikationsversuche bis hin zu jedem Zugriff auf Dateien.

16.2.4 Physical-Level Audit Trails

Wie der Name hier ebenfalls schon sagt, werden hier Ereignisse von physischen Zugriffskontrollen geloggt, also z.B. von Chipkartensystemen. Dabei können die genaue Zeit, der Ort und der Benutzer gespeichert und bei einem mehrfach fehlgeschlagenen Versuch gleich ein Alarm ausgegeben werden. Dazugehörige Systeme, die u.a. die Rechte verteilen, sollten auch überwacht werden.

16.2.5 Audit Trail Storage Alternatives

Es gibt verschiedene Arten der Speicherung, jede mit ihren Vor- und Nachteilen. Integrität und Geheimhaltung (Integrity and Confidentiality) müssen gegeben sein, z.B. durch Verschlüsselung und Access Control.

- Die Speicherung der Datei auf dem gleichen System ist am schnellsten und einfachsten. Durch den sofortigen Zugriff kann man auch gleich auf Angriffe reagieren. Jedoch kann ein Angreifer auch Zugriff erlangen.
- Das Speichern auf einem externen Medium wie z.B. einer CD ist am sichersten, da hier nichts mehr überschrieben werden kann (automatisch Integrität gegeben), aber es ist umständlich und führt logischerweise zu Verzögerungen beim Zugriff; abgesehen von dem konstanten Bedarf an Speichermedien.

- Man kann die Logs auch einfach ausdrucken, allerdings kann man diese Daten nur noch schwer analysieren.

16.3 Implementing Logging

Die Implementierung ist natürlich Abhängig vom Betriebssystem, aber auf jeden Fall muss es „hooks“ (auch „capture points“) geben, die das Loggen eines Ereignisses auslösen.

16.3.1 Windows Event Log

Unter Windows besteht ein Eintrag aus einer ID, den Attributen, und zusätzlichen optionalen Daten. Es gibt drei verschiedene Logs: den **system event log** für Treiber und andere wichtige Programme, den ungesichterten **application event log** für alle anderen Programme zusammen, und den **security event log**, der der Audit Log ist. Unwichtiges Beispiel in den Folien.

Event Categories: Aufzählung selbsterklärender Begriffe, die ich einfach mal übernehme.
account logon events, account management, directory service access, logon events, object access, policy changes, privilege use, process tracking, system events

16.3.2 UNIX Syslog

Syslog ist der Name des Logging-Mechanismus, der sich, mit einigen Abweichungen, auf allen UNIX-Systemen wiederfindet. Jeder Eintrag hat eine *facility*, das ist die Herkunft der Meldung, und eine *severity*, das ist die Wichtigkeit.

Zu den Grundfunktionen gehört eben das Loggen von ausgewählten Ereignissen, die Speicherung dieser Logs, und ein Protokoll zum Übertragen an einen zentralen Server. Erweiterungen können aber erweiterte Filtermöglichkeiten, Analysetools, programmierbare Reaktionen, Verschlüsselung, und noch mehr an Funktionalität hinzufügen.

In den Folien noch eine genauere Beschreibung des Protokolls, die ich nicht für notwendig erachte, und ein Beispiel. Drüberlesen schadet trotzdem nicht.

16.4 Logging at Application Level

Programme, die viele Rechte benötigen, also *privileged* sind, können Sicherheitslücken haben, die sich nicht in den Logs finden lassen, aber aufgrund deren Rechten verheerend sind. Solche Lücken sind Fehler im Algorithmus oder das Nicht-Validieren von Input.

Aufgrund dessen muss bei solchen Programmen das genaue Verhalten festgehalten werden können. Entweder sind sie dann selber dazu programmiert, ihr *audit data* zu erstellen, oder man muss es mit einer der folgenden zwei Methoden machen, zu denen in den Folien auch hilfreiche Diagramme sind.

Interposable Libraries: Bei der Erstellung des Prozesses wird auch eine Libary erstellt, die *zwischengeschaltet* wird und anstelle der Shared Library die Aufrufe entgegennimmt, sodass sie geloggt werden können. Das funktioniert nur mit dynamisch gelinkten Libraries.

Dynamic Binary Rewriting: Bei dieser Technik wird der Aufruf eines überwachten Programms abgefangen und an einen Service weitergeleitet. Dieser fügt den Code, der die Applikation um die gewünschte Funktionalität erweitert, in ein Speicherabbild des Programmcodes ein, und führt diesen dann aus. Logischerweise funktioniert das mit beiden Typen von Libraries.

16.5 Audit Trail Analysis

Um die Logs richtig analysieren zu können, muss man sie verstehen. Dazu gehört das Wissen um den Kontext eines Eintrags, also an welcher Stelle sich weitere Informationen zu diesem Ereignis finden lassen; aber auch die Fähigkeit, wahrscheinliche false positives zu erkennen.

Auch wenn der Administrator dabei wahrscheinlich von den Programmen unterstützt wird, bleiben trotzdem viele Informationen übrig, die man als Laie nicht entziffern könnte. Um „normale“ Einträge von Abweichungen unterscheiden zu können, hilft es, sich öfters ein Bild der Vorgänge zu verschaffen.

16.5.1 Types of Audit Trail Analysis

Natürlich kann man die Logs auf verschiedene Arten auswerten, was auch ein bisschen vom Zeitpunkt abhängt.

- Nach einem Ereignis kann man versuchen, den Grund herauszufinden und diesen zu beseitigen.
- Durch eine periodische Überprüfung der Logs kann man problematisches Verhalten entdecken.
- Eine Echtzeitanalyse könnte ein Teil eines IDS sein.

16.5.2 Audit Review

Dieser Begriff bezeichnet bloß das Anschauen von ganz bestimmten, vorher rausgefilterten Daten, wie z.b. den Aktionen eines bestimmten Users oder Programms.

16.5.3 Approaches to Data Analysis

Basic Alterting: Es wird einfach nur angezeigt, dass etwas bestimmtes passiert ist.

Baselining: Hier wieder Anomaly Detection bzw. Thresholding (siehe IDS), d.h. es werden „normale“ Werte festgelegt und Abweichungen auf bestimmte Arten registriert.

Windowing: Meldung von Ereignissen, die innerhalb/außerhalb eines bestimmten „Fensters“ von Parametern passieren (z.B. Einloggen, wenn alle schon Feierabend haben).

Correlation: Hierbei wird nach Zusammenhängen zwischen Ereignissen gesucht.

16.5.4 Integrated Aproaches

Da die Menge an Informationen explodierte, brauchte man ein System, um sie zu bändigen. So wurden die Security Information Management (**SIM**) bzw. System Information and Event Management (**SIEM**) Systene erschaffen. Diese können Logs aus den verschiedensten Quellen zusammenfassen, analysieren und auf bestimmte Ereignisse reagieren. Dabei gibt es zwei Ansätze, die entweder *agentless* oder *agent-based* arbeiten. Das gibt bloß an, ob auf dem Hostsystem ein spezielles Programm benötigt wird.

Als Beispiel wird hier MARS von Cisco genannt, das ein agentless SIEM mit einem zentralen Server ist.

17 IT Security Management and Risk Assessment

Um zu wissen, wie genau man vorgehen muss, muss man sich im Klaren darüber sein, was genau man verteidigt, welche Bedrohungen es dafür gibt, und wie genau diesem Bedrohungen entgegentritt.

17.1 IT Security Management

Das ist das Erreichen und Aufrechterhalten eines ausreichenden Levels an *confidentiality, integrity, availability, accountability, authenticity, reliability*.

Dazu werden Regeln erstellt, die Arbeiter geschult, und nötige Systeme installiert, um mal ein paar Punkte zu nennen. Reaktion auf entsprechende Bedrohungen und Ereignisse gehört ebenfalls zum Aufgabenbereich.

Es folgt eine Folie mit den ISO-Standards, die wohl keiner braucht.

17.1.1 IT Security Management Process

Es gibt in den Folien ein schönes Bild dazu, aber im Grunde ist es Folgendes: Die Sicherheit muss ein fester Bestandteil der Firma sein und auch als solcher gehandhabt werden. Es ist nämlich kein einmaliger Prozess, sondern ein zyklischer, bei der immer wieder überprüft und beim ersten Schritt angefangen werden muss.

17.1.2 Plan - Do - Check - Act

Diese zyklische Vorgangsweise ist auch Teil irgendeiner ISO-Richtlinie. Die genauen Schritte lassen sich wie folgt zusammenfassen:

Plan: Policies aufstellen, Ziele erkennen, sicherheitsrelevante Prozesse aufstellen und verbessern.

Do: Dies implementieren.

Check: Die bisherigen Ergebnisse bewerten.

Act: Bekannte Fehler verbessern, um die Sicherheit immer weiter zu erhöhen.

17.1.3 Organizational Context and Security Policy

Wie schon gesagt, muss man erst die Ziele festhalten, dann die Vorgehensweise planen und erkennen, wie in Zukunft gearbeitet werden muss, um es zu erhalten. Das bestehende System muss man ständig überprüfen.

17.1.4 Security Policy Topics

An dieser Stelle wieder eine hässliche Aufzählung. Die Policies, also die *Grundsätze*, müssen folgende Bereiche ansprechen:

Den Sinn dieser Sache, die Voraussetzungen, die Verantwortungen, „Security Awareness“, Strafbarkeit, das Schema der Zugangsrechte, ...

17.1.5 Management Support

Es ist wichtig, dass die Security auch von den Obermotzen anerkannt wird, denn dann ist es viel einfacher, da sie von allen ernstgenommen wird. Es ist zu empfehlen, dass es einen „IT Security Officer“ gibt, der das alles koordiniert.

17.2 Security Risk Assessment

Man muss genau herausfinden, wo die Gefahren sind, damit nicht an einer unnötigen Stelle Geld verschwendet wird. Idealerweise würde man alles überprüfen, das ist aber in der Praxis nicht möglich, da viel zu aufwendig. Stattdessen benutzt man eine der vier folgenden Strategien.

17.2.1 Baseline Approach

Bei diesem Ansatz wird einfach ohne Überprüfung für alles der „best practice“-Ansatz genommen. Je nach Unternehmen kann das aber schon zu viel sein, oder aber zu wenig. Da es für die meisten großen Sicherheitslücken reicht, schnell geht und billig ist, ist diese Lösung wohl eher für kleinere Unternehmen geeignet.

17.2.2 Informal Approach

Das heißt einfach nur, dass EXPERTEN sich da ransetzen und ohne große Analyse einfach nach bestem Wissen und Gewissen arbeiten. Auch relativ einfach und billig, und eben nicht so sicher. Aber schonmal besser als der letzte Ansatz.

17.2.3 Detailed Risk Analysis

Das wäre der „Formal Approach“, da hier nach einer bestimmten Struktur vorgegangen wird, um möglichst alle Bedrohungen, aber auch die besten Möglichkeiten für jetzt und später ausfindig zu machen. Das dauert dementsprechend länger und ist teurer, aber könnte für bestimmte Aufgabenfelder Voraussetzung sein

17.2.4 Combined Approach

Wie der Name schon sagt, werden hier die drei Ansätze kombiniert. Zuerst wird, wie beim Baseline Approach, eine Grundsicherheit nach „best practice“ implementiert, danach erste Schwachstellen per informeller Analyse herausgefunden und schließlich dieses System nochmal gründlich überprüft.

Auf diese Art und Weise hat man Anfangs schon eine sinnvolle Stufe erreicht, weshalb die nachfolgenden Schritte nicht mehr so aufwändig sind; es ist also weit effektiver, auch wenn anfangs noch Schwachstellen offenbleiben könnten.

Insgesamt ist dies nach ISO-Norm für die meisten Unternehmen empfohlen.

17.3 Detailed Risk Analysis Process

Der schon erwähnte formelle Prozess, der die Risiken am gründlichsten Erkennt, wird jetzt untersucht. In den Folien ist ein unlesbares Diagramm.

17.3.1 Establish Context

Eigentlich selbsterlärend. Es wird festgestellt, welche Ziele das Unternehmen hinsichtlich der Security hat, und dessen „political / social environment“ sowie die Risikofreudigkeit. Auch werden rechtliche Grenzen in Erfahrung gebracht.

Nun kann man den Bereich für die Einschätzung der Risiken abstecken.

17.3.2 Asset Identification

Beim zweiten Schritt wird dann in Erfahrung gebracht, was genau geschützt werden muss und welchen Wert es für das Unternehmen hat. Ein Asset ist ein Gut des Unternehmens, egal ob greifbar oder nicht.

Da man vorher kein Wissen über die internen Vorgänge im Unternehmen hat, versucht man sich bei Angestellten zu erkundigen.

17.3.3 Threat Identification

Um Bedrohungen zu erkennen, muss man zu jedem Gut zwei Dinge fragen:

1. Wer oder was könnte ihm schaden?
2. Wie könnte man ihm schaden?

17.3.4 Threat Sources

Eine Bedrohung kann natürlich sein, wie z.B. eine Naturkatastrophe, aber auch künstlich sein. Hier lässt sich nochmal unterteilen. Ein Angestellter, der ausversehen falsche Daten eingibt, ist als unabsichtlich zu klassifizieren; ein Angestellter, der Daten verkauft (passiv) oder ein Hacker (aktiv) aber nicht.

Bei menschlichen Angreifern müssen daher noch die Motivation, Fähigkeiten, Ressourcen wie z.B. Zeit und Geld, Wahrscheinlichkeit eines Angriffs und zu guter Letzt das Level der Abschreckung für einen Angreifern betrachtet werden.

Gute Anhaltspunkte für die Häufigkeit einer bestimmten Bedrohung liefern Statistiken.

17.3.5 Vulnerability Identification

Logischerweise werden hierbei Schwachstellen in den Systemen und Prozessen des Unternehmens erkannt und deren Gewicht festgestellt. Es muss aber zu der Schwachstelle auch eine Bedrohung geben, bevor sie beachtet wird. Hier kann man wieder Statistiken zu Rate ziehen.

17.3.6 Analyse Risks

Für jede Bedrohung müssen die Wahrscheinlichkeit und die Konsequenzen herausgefunden werden, was natürlich auch schwer festzusetzen ist, weswegen eher qualitative anstatt von quantitativen Methoden benutzt werden, was natürlich auch nicht einfach ist. "Wahrscheinlich"würde also z.B. heißen, dass ein solcher Vorfall sich schon einmal ereignet hat; bei den Konsequenzen könnte eine Einordnung "katastrophal"lauten. In den Folien sind lustige Tabellen dazu. Jedenfalls gilt:

$$Risiko = Auftrittswahrscheinlichkeit * VerlustderFirma \tag{1}$$

17.3.7 Risk Register

Die Ergebnisse aus der Risikorechnung werden in ein **Risk Register** geschrieben und der Verwaltung vorgelegt, damit diese Entscheidungen fällen kann. Unwichtige Sachen könnten dabei ignoriert, und von den wichtigen Sachen die leichter umzusetzenden auch wirklich zuerst umgesetzt werden. Insgesamt ist es aber immer eine Kosten/Nutzen-Rechnung.

17.3.8 Risk Treatment Alternatives

Es gibt verschiedene Wege, erkannte Risiken zu behandeln. Man nimmt die Konsequenzen in Kauf (**risk acceptance**), stoppt die Gefahrenquelle und geht lieber Umwege (**risk avoidance**) oder überträgt die Verantwortung an Dritte oder teilt sie mit einem anderen Unternehmen (**risk transferal**).

Man könnte auch die Schwere der Konsequenzen vermindern, z.B. durch ein Backup. Mit der Wahrscheinlichkeit lässt sich das auch tun, z.B. durch den Einsatz einer Firewall.

Komisches Beispiel in den Folien, durchlesen?

18 IT Security Controls, Plans and Procedures

Logische Fortsetzung des letzten Kapitels.

18.1 Controls or Safeguards

Die Schutzmechanismen können sehr verschieden sein: Umsetzung bestimmter Methoden, Verminderung der Verletzbarkeit, ...

Sie lassen sich grob in drei Klassen einteilen:

- **Management Control:** Angelegenheiten, die die Verwaltung regeln kann. Dazu gehören z.B. Richtlinien und Planung.
- **Operational Control:** Dieses Stichwort bezieht sich auf Mechanismen, die von Menschen umgesetzt werden, z.B. die Entwicklung eines Notfallplans oder die Wartung.
- **Technical Control:** Hierbei geht es um die Benutzung von Hard- und Software zum Schutz des Systems.

18.2 Cost-Benefit Analysis

Sehr wahrscheinlich werden nicht alle möglichen Schutzmechanismen eingebaut, da man kein Geld dazu hat. An dieser Stelle wird dann abgewägt, wie man mit den gegebenen Mitteln den größten Vorteil erhält. Man möchte z.B. das Risiko auch nicht um mehr senken, als nötig, da dies auch mehr Geld kostet.

18.3 IT Security Plan

In diesem Plan steht genau, was getan wird, was dafür benötigt wird, und wer dafür verantwortlich ist. Außerdem sollten die Risiken, die empfohlenen (und daraus ausgewählten) Schutzmechanismen, die benötigten Ressourcen, die Verantwortlichen und zu guter Letzt ein zeitlicher Ablauf (**implementation plan**) angegeben werden.

18.3.1 Security Plan Implementation

Selbsterklärend. Die Aufgaben werden verteilt und die Leute überwacht, am Ende muss nochmal die Benutzung des neuen Systems genehmigt werden.

18.3.2 Security Training / Awareness

Die Zuständigen müssen natürlich von den Änderungen unterrichtet werden und müssen wissen, wie man damit umgeht. Das ist allerdings auch ein guter Moment, das Sicherheitsbewusstsein zu schulen.

Dazu gehört es, zu verstehen, warum bestimmte Schutzmechanismen existieren und ihre Wichtigkeit, wer wofür zuständig ist, und die generellen Sicherheitsziele des Unternehmens zu kennen.

18.4 Implementation Followup

Wie wir schon gelernt haben, ist es ein zyklischer Prozess; deshalb darf man seine Arbeit nicht als getan betrachten, sondern muss die neuen Mechanismen überwachen und auswerten. Das führt eventuell zu neuen Änderungen.

18.4.1 Maintenance

Die implementierten Mechanismen bedürfen regelmäßiger Überprüfung und Wartung, um Funktionstüchtigkeit, aber auch Angemessenheit in diesem Einsatzbereich sicherzustellen. Dies ist vor allem bei einer Änderung der Fall.

18.4.2 Security Compliance

Die Einhaltung des Security Plan muss natürlich auch überprüft werden; das passiert meistens im Zuge einer generellen Überprüfung, indem Leute mit Checklisten rumrennen.

18.4.3 Change and Configuration Management

Änderungen am System, z.B. Bugfixes für die OSe, müssen natürlich auf Nebeneffekte überprüft werden. Um im Falle, dass etwas schiefgeht, das System wieder zum Laufen zu kriegen, werden alle Änderungen an der Konfiguration geloggt.

18.5 Incident Handling

Die irgendwann sicher irgendwas passiert, braucht man auch Pläne, wie man mit den Ereignissen umgeht. So kann man im Falle eines Falles wesentlich schneller reagieren.

18.5.1 Types of Security Incidents

Als „security incident“ wird alles bezeichnet, was confidentiality, integrity, ... angreift. Grob kann man in unerlaubtern Zugriff zum System (als anderer User einloggen, Zugriffsbeschränkungen umgehen , ...) und unerlaubter Modifikation von Daten (unerlaubtes Ändern von Daten, ...) unterscheiden.

18.5.2 Managing Security Incidents

Nach dem Entdecken wird versucht, die Art des Angriffs zu erkennen, um entsprechend reagieren zu können. Nachdem die Gefahr gebannt ist, wird die Sicherheitslücke dokumentiert, damit soetwas nicht mehr passiert.

18.5.3 Detecting Incidents

Zum einen gibt es die offensichtliche Möglichkeit, dass man von einem Benutzer etwas Ungewöhnliches gemeldet bekommt. Solches Verhalten sollte übrigens gefördert werden. Andererseits gibt es da auch automatische Tools wie z.B. IDS, IPS und Log Analysis Tools. Die Effektivität solcher Tools hängt natürlich stark von ihrer Konfiguration ab.

18.5.4 Responding to Incidents

Sobald ein möglicher Vorfall entdeckt wurde, muss man auf eine dokumentierte Vorgehensweise zurückgreifen können, wie man am besten den Grund herausfindet und die Situation rettet. Natürlich ist es nicht möglich, alle möglichen Arten zu kennen, aber meist lassen sich Angriffe in bestimmte Kategorien einteilen. Es ist auch hilfreich, wenn man auch Ansprechpartner für wichtige Entscheidungen genannt bekommt, oder sogar ob ein CERT oder die Polizei zu alarmieren ist.

18.5.5 Documenting Incidents

Damit in der Zukunft nicht nocheinmal derselbe Angriff gelingt, sollte man immer dokumentieren, wo genau die Schwachstelle liegt und wie man das in Zukunft verhindern kann. Eine Einschätzung der Auswirkungen auf das System und eine eventuelle Veränderung des Risikos gehört auch dazu, das machen dann aber meist die Experten (wieder Schritt 1 des Zyklus).

Wieder komisches Beispiel.

19 Symmetric Encryption and Message Confidentiality

Kryptographische Systeme werden generell Anhand folgender drei Punkte bewertet:

1. **Die Art der Operationen, um Plaintext in Ciphertext umzuwandeln.** Alle Verschlüsselungsalgortihmen basieren auf zwei Prinzipien: Einmal die Umwandlung eines Elements (bit, Buchstabe, Gruppe von Buchstaben, ...) des Plaintexts in ein anderes Element, und die Vertauschung dieser neuen Elemente. Eine Grundvoraussetzung dabei ist jedoch die Umkehrbarkeit.

2. **Die Anzahl der verwendeten Schlüssel.** Bei einem Schlüssel spricht man von *symmetrischer*, *single-key* oder *konventioneller* Verschlüsselung. Wenn Sender und Empfänger einen anderen Schlüssel benutzen, spricht man von *asymmetrischer* oder *public-key* Verschlüsselung.

3. **Die Art der Abarbeitung des Plaintext.** Hierbei unterscheiden man zwischen *block cipher*, bei dem der Input blockweise bearbeitet wird, und *stream cipher*, bei dem die Elemente hintereinander bearbeitet werden.

In diesem Kapitel geht es aber erstmal um die single-key encryption, die auch immernoch am meisten verbreitet ist. Zur Erinnerung: Die „Zutaten“ sind der Plaintext, der Verschlüsselungsalgorithmus, der Key, der Ciphtertext, und der Entschlüsselungsalgorithmus.

19.1 Cryptanalysis

Der Prozess, an den Plaintext oder den Key zu kommen, nennt sich Cryptanalysis. Die Attacken können, aufgrund der gegebenen Umstände, unterschiedlich geartet sein:

- **Ciphertext Only** Das ist der schwerste Angriff. Hier kann man eigentlich nur den Key durch Brute-Forcen erlangen, es sei denn, man hat einige Informationen über den Plaintext, wie z.B. die Sprache, denn dann könnte man sich statistische Analysen zur Hilfe ziehen.

- **Known Plaintext** Der Angreifer könnte vorher schon den Plaintext und den dazugehörigen Ciphertext abgefangen haben, oder wissen, dass ein bestimmtes Textstück vorkommt.

- **Chosen Plaintext** Wenn der Angreifer Zugriff auf das System hat und selber irgendeinen Text verschlüsseln kann, kann er jeglichen Plaintext verwenden, um damit auf die Struktur des Schlüssels zu schließen.

- Außerdem gibt es noch **Chosen Ciphertext** und **Chosen Text**, aber diese sind relativ selten und werden nicht weiter erläutert.

Nur äußerst schwache Algorithmen lassen sich durch einen Ciphertext-Only-Angriff knacken. Normalerweise sind die Verschlüsselungsalgorithmen darauf ausgelegt, einer Known-Plaintext-Attacke standzuhalten.

19.1.1 Computationally Secure Algorithms

Eine Verschlüsselung gilt als rechnerisch sicher, wenn die Kosten, sie zu knacken, den Wert der Informationen übersteigen, und die Zeit, die dafür benötigt wird, die (sinnvolle) Lebensdauer der Information übersteigt.

Allerdings lässt sich nur bei einer Brute-Force-Attacke der Aufwand abschätzen.

19.1.2 Feistel Cipher Structure

Viele symmetrische Blockverschlüsselungsverfahren, inklusive DES, setzen auf diese Struktur, die man sich aber am besten auf dem Diagramm ansieht. Der Plaintext wird in zwei Hälften geteilt, die rechte Seite wird mit dem Schlüssel als Argument an eine Funktion F gefüttert und mit der linken Seite XOR-Verknüpft. Dann gehts über Kreuz (R wird L etc.) und durch einen speziellen Algorithmus wird ein Subkey generiert, mit dem dann der Spaß von vorne losgeht.

Zum Entschlüsseln braucht man auch nur diese Algorithmus, allerdings muss man mit dem letzten Subkey anfangen und sich hocharbeiten.

19.2 Block Cipher Structure

Jede Blockverschlüsselung hat eine ähnliche Struktur wie die eben beschriebene. Die verschiedenen Verfahren unterscheiden sich in der Größe der Blöcke, der Größe des Keys, der Anzahl der Runden, dem Algorithmus zur Generierung des Subkeys, der Rundenfunktion (die jede Runde angewendet wird...).

In die Entwicklung eines solchen Algorithmus fließt auch die schnelle Ver/- und Entschlüsselung für die Benutzung in Programmen sowie eine einfache Analyse in die Überlegnungen mit ein. (Wenn man einen Algorithmus leicht analysieren kann, lassen sich Schwächen schnell finden und er sich darauf besser gegen Cryptanalysis verteidigen.)

19.2.1 Data Encryption Standard (DES)

Gibt nicht viel zu sagen, ist bloß ein Beispiel. Plaintext ist ein 64-bit-Blöcken, der Key hat 56 bit. Es gibt 16 Runden. Wer mag, kann sich das Diagramm ansehen.

19.2.2 Triple DES (3DES)

Heutzutage ist eine Schlüssellänge von 56 bit nicht mehr ausreichend, deshalb wurde 3DES entwickelt. Es ist einfach nur DES drei Mal hintereinander mit drei verschiedenen Schlüsseln ausgeführt (daher die effektive Schlüssellänge von 168 bit), nur ist die zweite Ausführung eine Entschlüsselung. Das hat anscheinend den Vorteil, dass auf diese Weise Texte, die in DES verschlüsselt wurden, aucht mit 3DES entschlüsselt werden können. Durch die dreifache Ausführung ist es aber auch entsprechend langsamer geworden.

19.2.3 Advanced Encryption Standard (AES)

DES war unsicher geworden und 3DES nur eine (langsame) Notlösung, also kam AES. Die Blocklänge beträgt 128 bit, und der Key 128, 192 oder 256 bite. Es gibt auch ein Diagramm, aber es erscheint mir unwichtig. Auf der nächsten Folie ist zum Diagramm der Kommentar völlig kaputt.

19.2.4 Substitute Bytes

Erklärung laut ihm eh zu schwer, aber es ist halt eine der Methoden, die AES benutzt.

19.2.5 Shift Rows

Ebenfalls eine Methode, die von AES verwendet wird. Die i-te Reihe (i = 0..3) der quadratischen State-Matrix wird um i Zeichen nach links geshiftet (circular).

19.2.6 Mix Columns

Jedes Byte einer Spalte wird auf einene Wert gemappt, das die Funktion für alle vier Bytes im der Spalte ist. So soll der Inhalt gut durchgemischt werden.

19.2.7 Add Round Key

Die 128 bit der State-Matrix werden bitweise mit den 128 bit des Rundenschlüssels geXORt. Da XOR seine eigene Umkehrfunktion ist, funktioniert das auf dem Rückweg genauso.

19.3 Stream Ciphers

Solche Verschlüsselungen benutzen keine Blöcke, sondern sind kontinuierlich. Dabei wird ein Pseudozufallszahlengenerator mit dem Passwort gefüttert und der Output bitweise mit dem Plaintext XOR-verknüpft. So wird ein Stream mit annähernd zufälligen Zahlen erzeugt, aber heutzutage sollte man auch hier eher auf einen Schlüssel mit 128 bit Länge zurückgreifen.

19.3.1 RC4

Ein weit verbreiteter Algorithmus, der z.B. in SSL und WPA Verwendung findet. Das Diagramm auf der Folie hilft stark beim Verständnis. Es wird jedenfalls ein Array mit einer Permutation der Zahlen 0..255 genommen und bitweise I DON'T GET IT

19.4 Modes of Operation

Blockverschlüsselungen müssen immer die volle Blocklänge haben, und evtl. aufsplitten und/oder auffüllen. Dann gibt es irgendwie fünf verschiedene Modi, die jetzt folgen.

19.4.1 Electronic Codebook (ECB)

Das ist der einfachste Modus, er teilt dein Plaintext einfach auf die Blöcke auf und verschlüsselt jeden Block mit demselben Key. Das ist für längere Texte ungeeignet, da sich wiederholender Plaintext auch sich wiederholenden Ciphertext ergibt.

19.4.2 Cipher Block Chaining (CBC)

Wie auch hübsch auf dem Diagramm zu erkennen ist, wird der der Plaintext eines Blocks mit dem Ciphertext des Blocks vorher ge-XOR-d; der Schlüssel bleibt gleich. Für den ersten Block wird daher noch ein IV, ein Initialisierungsvektor, gebraucht, der ebenfalls ausgetauscht werden muss.

Für das entschlüsseln wird der Output mit dem Ciphertext des Blocks davor ge-XOR-d, um den Plaintext zu erhalten. Auch hier wird am Anfang der IV benutzt.

19.4.3 Cipher Feedback (CFB)

Durch diesen Modus kann jeder Block Cipher in einen Stream Cipher umgewandelt werden. Am besten sieht man sich hier das Diagramm an: Der Key wird mit einem IV verschlüsselt und dann von links so viele Bits genommen, wie der Block groß ist. Jetzt wird bitweise XOR-Verknüpft und fertig ist der Stream Cipher. Dieser Ciphertext wird jetzt als Input für das Shift-Register benutzt, mit dem Das Verschlüsselungsalgorithmus zusammen mit dem Key gefüttert wird.

19.4.4 Counter (CTR)

Diese Methode ist sehr einfach, aber auch hier hilft das Diagramm. Der Schlüssel wird mit irgendeinem Startwert verschlüsselt und mit dem ersten Block an Plaintext ge-XOR-d. Für den zweiten Block muss der Counter bloß *irgendeinen* anderen Wert haben, also kann man ihn auch um ein erhöhen. Ja, hier wird nichts verkettet. Trotzdem ist diese Methode mindestens so sicher wie die anderen, hat aber auch noch Vorteile, wie die Effizienz, die dadurch entsteht, dass man die Blöcke problemlos parallel berechnen kann, und die Einfachheit.

19.5 Location of Encryption

Hierzu auch wieder ein Diagramm. Die am weitesten verbreitete Möglichkeit, ein Netzwerk zu sichern, ist Verschlüsselung; davon gibt es aber verschiedene Arten. Bei der **Link Encryption** ist jede Verbindung für sich verschlüsselt. Deswegen müssen die Daten auch entschlüsselt werden, bevor sie weitergeschickt werden können, da man sonst nicht wüsste, wohin damit. Bei der **End-to-End Encryption** wir nur der Datenteil des Pakets verschlüsselt und das Paket auf den Weg geschickt, bis es beim Ziel ankommt und entschlüsselt werden kann.

19.6 Key Distribution

Bei der symmetrischen Verschlüsselung brauchen beide Partner denselben Key. Um das zu erreichen, gibt es folgende Möglichkeiten:

- A wählt den Key aus und bringt ihn zu B.
- C wählt den Key und bringt ihn zu A und B, was schonmal besser ist, aber nicht bei einer großen Anzahl von Benutzern.
- A wählt neuen Key, bringt ihn B, mit dem alten verschlüsselt.
- C wählt neuen Key und schickt ihn verschlüsselt an A und B. Das ist logischerweise die beste Variante.

Eine mögliche Implementierung des Beispiels C finde sich eine Folie weiter. Dabei gibt es neben einem Permanent Key noch einen Session Key, der danach zerstört wird, ein Key Distribution Center, das die Keys austeilt, und in jedem Host noch ein Security Service Module (SSM), das die End-to-End-Encryption durchführt.

20 Public-Key Cryptography and Message Authentication

20.1 Simple Hash Functions

Eine *Secure Hash Function* funktioniert nur in eine Richtung; aus ihr lässt sich der ursprüngliche Wert nicht wiederherstellen.

Alle Hash-Funktionen funktionieren nach dem gleichen Prinzip. Der Input wird als Sequenz von Blöcken mit gleicher Länge gesehen und Block für Block bearbeitet.

Um die Datenintegrität festzustellen, sind solche Hashfunktionen sehr gut geeignet; aber falls der Angreifer auch nur ein bisschen Plaintext hat, ist es ein leichtes, eine neue Nachricht mit demselben Hashcode zu generieren.

20.1.1 SHA Secure Hash Functions

Wurde vom National Institute of Standards and Technology 1993 entwickelt und produzierte einen Hashwert der Länge 160 bit. Auch hier ist diese Schlüssellänge nicht mehr ganz sicher wie gedacht. 2002 kamen drei neue Versionen von SHA, jeweils mit einer Schlüssellänge von 256, 284 und 512.

20.1.2 SHA-512 Structure

Dieser Algorithmus produziert einen Output von 512 bit und erwartet Input in 1024bit-Blöcken.

Im ersten Schritt wird der Input so aufgefüllt, dass die Länge kongruent zu 896 modulo 1024 ist, danach ein Block der Länge 128 bit hinten drangehängt, der die Länge der Originalnachricht enthält. Wie die 64-bit-Integers initialisiert werden, steht "nur im Buch", aber danach werden iterativ 1024bit-Blöcke durch wildeste Berechnungen geschickt, die glaube ich irrelevant sind.

20.1.3 Other Secure Hash Functions

Die meisten Hashfunktionen sind ebenfalls iterierte, da wenn die Funktion für einen einzelnen Block, die **Kompressionsfunktion**, kollisionsresistent ist, ist es auch der ganze Output.

Zum Beispiel gibt es **MD5**, das überall benutzt wurde. Dieser Algorithmus arbeitet aber nur auf Blöcken mit der Länge 512 bit, was anscheinend kombiniert mit dem Output von 128 bit zu kurz ist.

Außerdem gibt es da noch **Whirlpool**, das stark AES ähnelt. Es verwendet ebenfalls 512-bit-Blöcke und liefert einen Output von ebenfalls 512 bit.

20.1.4 HMAC

Ist ein MAC (Message Authentication Code). Daran bestand Interesse, da Hashfunktionen meist schnell sind. Das Problem war nur noch, einen Key mit reinzubekommen.

HMAC scheint das besonders gut zu lösen, indem es die Hashfunktion als Blackbox behandelt, sie also gut austauschbar ist. So gut, dass es z.B. für IPSec und TLS eingesetzt wird.

20.1.5 HMAC Structure

Da ich bezweifle, dass wir irgendwas vorhashen müssen, lass ich mal die Funktionsweise von HMAC weg. Da sind vor allem irgendwelche Konstanten, die man sich merken müsste, etc.

20.1.6 Security of HMAC

Die Sicherheit hängt natürlich vom Hashalgorithmus ab, aber generell gibt es nur zwei mögliche Wege für einen erfolgreichen Angriff. Der erste ist, den Key zu bruteforcen, um einen gleichen Output zu erzeugen, oder er findet eine Kollision in der Hashfunktion, also einen zweiten Wert, der denselben Output hat. Für beides ist der Aufwand exponentiell; für HMAC ist also auch MD5 sicher.

20.2 RSA Public-KEy Encryption

Wurde von Rivest, Shamir und Adleman 1977 am MIT erfunden; ist der bekannteste und am weitesten verbreitete Public-Key-Verschlüsselungsalgorithmus.

20.2.1 RSA Algorithm

Auch hier bezweifle ich wieder, dass wir Algorithmen auswendig lernen müssen. Aber ein Beispiel mit kleinen Zahlen ist auch auf den Folien.

20.2.2 Attacks on RSA

- Wie bei allem, kann man auch hier versuchen, den Private Key per **Bruteforce** zu errechnen.
- Man kann versuchen, die Primfaktorzerlegung, auf der diese Verschlüsselung basiert, **mathematisch** zu knacken. Da die Algorithmen hier auch immer besser werden, sollte man einen Schlüssel von mindestens 1024 bit benutzen.
- Eine nette Analogie für die **Timing Attacks** in den Folien ist ein Einbrecher, der beim Safe aufmachen zuschaut, und anhand der Dauer der Drehung versucht, die Kombination zu erraten. Anscheinend ist das ziemlich effektiv, aber auch durch zufällige Verzögerungen ziemlich leicht zu kontern.
- **Chosen Ciphertext Attacks** sind laut Folien zu kompliziert für uns.

20.3 Diffie-Hellman Key Exchange

Das erste Mal, dass Public-Key-Encryption vorgeschlagen wurde, aber die beiden haben dazu nur ein Schema zum sicheren Schlüsselaustausch auf die Beine gebracht, das aber heute noch recht viel genutzt wird. Für Details soll man ins Buch gucken, was wir leider nicht besitzen. Irgendwas mit "discrete logarithms".

20.3.1 Diffie-Hellman Algorithm

Ich denke wieder, dass man keine Algorithmen auswendig lernen muss. Ein Beispiel steht aber wieder dort.

20.3.2 Man-in-the-Middle Attack

Ich müsste es einfach abtippen, das spare ich mir an dieser Stelle.

20.4 Other Public-Key Algorithms

Da gäbe es **DDS**, den Digital Signature Standard, der 1991 vorgeschlagen und 1996 zuletzt verbessert wurde. Er ist ausschließlich für digitale Signaturen zu gebrauchen und benutzt den SHA-1 Hashalgorithmus.

In letzter Zeit werden für sicheres RSA immer längere Schlüssel benötigt, wodurch es auch zu längeren Rechenzeiten kommt. Auch wenn RSA im Moment am weitesten verbreitet und unangefochten ist, scheint **ECC**, elliptic curve cryptography, langsam an Unterstützung zu gewinnen. Es verspricht, die gleiche Sicherheit bei kleineren Schlüssellängen zu bieten, aber da es noch kaum genutzt wird, stand diese Art der Verschlüsselung noch nicht so sehr auf dem Prüfstand.