**Security Engineering**

**Secure Coding Guidelines:**

Es gibt 2 Klassen von Guidelines: allgemeine (general) oder sprachabhängige (language specific)

**General Guidelines:**

* white lists sind besser als black lists zur Validierung der Benutzereingaben
* KISS-Principle
* Versions- und Konfigurationskontrolle
* Benutzung von Security Design Patterns
* keine Nutzung von unsicheren Funktionen (z.B. gets())
* ...

**Security Patterns:**

**Struktur:**

* Kontext

- Beispielszenario, Annahmen über das IT-System, ...

* Problem

- Gefahren, gegen die man sich schützen möchte

* Anforderungen
* Lösungen

- verschiedene Lösungen werden vorgestellt (Diskussion Vor- und Nachteile)

- Ziel: Risikominimierung

* Verwandte Muster

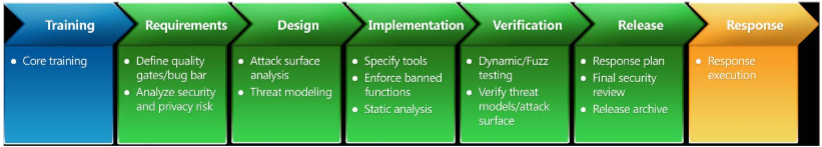
**Klassen:**

* Architectural-level patterns
* Design-level patterns
* Implementation-level patterns

**Privilege Seperation:**

* Code in Teile, die limitierte Privilegien benötigen und Teile, die spezielle Privilegien benötigen, aufgeteilt
* nur die Teile, die spezielle Privilegien benötigen, werden mit diesen Privilegien auf dem Server ausgeführt, der Rest mit den limitierten

**SDL:**

****

1. Training
2. Requirements

* Spezifikation der Sicherheits- und Privatsphäreanforderungen
* Defining Quality Gates: Qualitätskriterien, die über die Freigabe des nächsten Projektschritts entscheiden
* Security and Privacy Risk Assessment: Identifizieren funktionale Aspekte, die eine genauere Überprüfung erfordern

1. Desing

* Definiert und dokumentiert Sicherheitsarchitektur

1. Implementation

* Durchsetzung von Sicherheitspraktiken, um sichere Softwareentwicklung sicherzustellen

1. Verification

* Es wird getestet, ob die Software die spezifizierten Sicherheits- und Privatsphäreanforderungen erfüllt

1. Release

* Software wird auf Auslieferung vorbereitet

1. Response

**Buffer Overflow:**

**Segmente:**

* Text/Code-Segment (Programmcode)
* Data-Segment (initialisierte globale und statische lokale Variablen)
* BSS-Segment (globale nicht initialisierte Variablen)
* Heap-Segment (malloc/new und free/delete)
* Stack-Segment (lokale nicht statische Variablen, Argumente und Rücksprungadressen)

Schritte eines Funktionsaufrufs:

* Prolog (speichern des aktuellen Stacks vor Funktionsaufruf und Speicherallokation)
* Operationen (alle Operationen außer dem Funktionsaufruf zwischen Prolog und Epilog)
* Funktionsaufruf (der eigentliche Funktionsaufruf)
* Epilog (bevor zu der aufrufenden Funktion zurückgekehrt wird, wird der alte Stack wiederhergestellt)

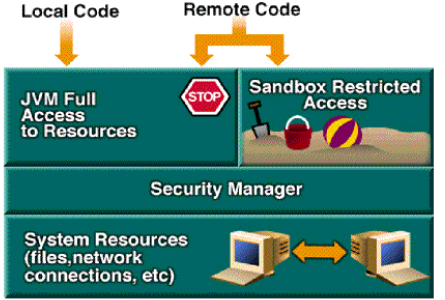
**typischer Prolog:** push %ebp mov %esp, %ebp sub $(durch 4 teilbare Zahl), %esp

* legt Stack-Frame-Pointer auf Stack
* allokiert Speicher für Variablen

(genaueres zu Buffer Overflows kommt noch)

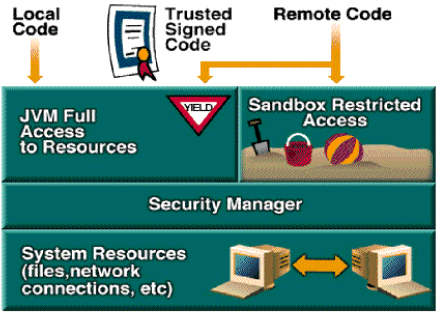
**Java Security:**

**Sandbox Model:**



* Ziel: Ausführung von nicht vertrauenswürdigem Code (über Netzwerk gedownloadet) soll verhindert werden
* lokaler Code ist vertrauenswürdig und kann vollen Zugriff auf wichtige Ressourcen bekommen
* gedownloadeter Code ist nicht vertrauenswürdig und kann keine Aktionen außerhalb seines begrenzten Bereichs ausführen

**Trusted Applets:**



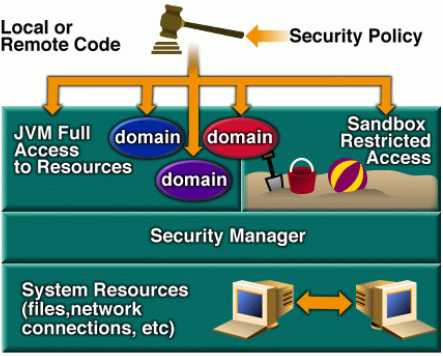
* Es gibt die Möglichkeit signierte Applets außerhalb der Sandbox zu starten
* 3 Teile (sequentiell ausgeführt):

- Verifier: Typ-Sicherheit

- Class Loader: lädt und "entlädt" dynamisch Klassen der "Java Runtime Environment"

- Security Manager: bietet Schutz vor potentiell gefährlichen Funktionalitäten

**fine-grained security:**



* Security Policies für lokalen und "remote" Code

- definiert Genehmigungen

- Konfiguration durch Benutzer oder Systemadministrator

- spezifiziert genehmigten Zugriff auf bestimmte Ressourcen

* Laufzeitsystem gliedert Code in Domänen

- Menge von Klassen, welche die gleiche Menge von Genehmigungen zugeteilt bekommt

- kann so konfiguriert werden, dass das Applet wie in einer Sandbox ausgeführt wird