

# Offen und sicher

Was bedeutet Offenheit  
für die Sicherheit?

Achim Leitner,  
Linux New Media AG

# Ein Widerspruch in sich?

- ⇒ Offenheit meint *nicht* „jeder darf alles“
- ⇒ Die Technologie ist offen und frei, die Geheimnisse bleiben geheim
- ⇒ 1833: Prinzipien von Kerckhoff (Kryptologie):
  - Die Sicherheit eines Verschlüsselungsverfahrens darf nur von der Geheimhaltung des Schlüssels abhängen, nicht jedoch von der Geheimhaltung des Algorithmus



# Geheim = sicher?

- ⇒ „Security through Obscurity“ ist weit verbreitet
  - Je weniger ein Angreifer weiß, um so schwieriger wird es für ihn, einzubrechen
- ⇒ „Open Source“ heißt, jeder Anwender erhält eine Kopie des Quellcodes. Dabei stoßen Entwickler, die mit Linux arbeiten, häufig auf Sicherheitslücken. Auf Microsoft Windows trifft das nicht zu.

➤ Microsoft Whitepaper zu „Linux im Handel“

# Versteckte Lücken

- ⇒ So lange niemand eine Lücke kennt, ist das System „praktisch sicher“
- ⇒ Woher wissen, dass niemand die Lücke kennt?
- ⇒ Verborgene Lücke bleibt vorhanden
- ⇒ Zeitbombe
- ⇒ Fix ist notwendig

# Sicher weil offen?

- ⇒ Open Source ist keine Wunderwaffe
- ⇒ Wenn der Quellcode eines Programms offen liegt und von zahlreichen kompetenten Fachleuten weltweit eingesetzt, untersucht und geprüft werden kann, erhöht das natürlich die Sicherheit der Software um ein ganz erhebliches Stück.
  - Staatssekretärin des BMWi Margareta Wolf, Eröffnungsrede Linux-Tag 2001

# Wer macht's?

- ⇒ Im Grunde kann jeder Anwender bei Open Source die Sicherheit überprüfen
- ⇒ Im Grunde kann jeder die Bugs selbst beheben
- ⇒ In Wahrheit kann das kaum ein Anwender
- ⇒ In Wahrheit macht das kaum ein Anwender

## Warum?

# Wieso so schwierig?

- ⇒ Sicherheitslücken finden: sehr anspruchsvoll
- ⇒ Programmierer mit Spezialkenntnissen
- ⇒ Problem: Code-Umfang
- ⇒ Problem: „fremder“ Code

Ist damit das ganze Konzept hinfällig?

# Verschiedene Hüte...

⇒ Verbreitete Einteilung der Sicherheitsspezialisten

⇒ Black Hat

- Nutzt Sicherheitslücken selbst aus
- Gibt sie nur an den „Untergrund“ weiter



⇒ White Hat

- Sucht Lücken, um sie zu schließen
- Gibt sie an möglichst alle Betroffenen weiter



# Motivation

⇒ Spekulation, warum nach Lücken gesucht wird



# Mit Quellen einfacher

- ⇒ Suche nach Lücken für *beide* leichter
- ⇒ Nur so: Patches (durch White Hats)
- ⇒ Lücken in proprietärer Software zu suchen verstößt häufig gegen Gesetze oder Verträge (DMCA, Reverse Engineering)
  - Extrembeispiel: Dmitry Skylarov
- ⇒ White Hats bei proprietärer Software im Nachteil

# Image

- ⇒ Sicherheitslücke ⇒ Imageschaden?
  - Lücke nicht zugeben
  - Verharmlosen
- ⇒ Black *und* White Hats „unbeliebt“
- ⇒ Trifft nur die White Hats!
- ⇒ Anwender ist unsicher, weiß es aber nicht

# Vertrauen

- ⇒ Wer vertraut wem?
- ⇒ Bei proprietären Produkten misstraut der Hersteller seinen Kunden
- ⇒ Warum also dem Hersteller vertrauen?
- ⇒ Bei Open Source freie Wahl:
  - Viele Reviewer, auch als Dienstleistung denkbar
  - Wenn ein *einzelner* etwas findet, wird es publik

# Hintertüren

- ⇒ Viele Entwickler ⇒ könnten Hintertüren einbauen, keine Kontrolle
- ⇒ Meisten *gibt* es aber eine Kontrolle jedes neuen Sourcecodes durch wenige Kernentwickler
- ⇒ Trojaner auch in proprietäre Software möglich
- ⇒ ...nur sind sie dort nicht zu finden

# Borland InterBase

## ⇒ Beispiel: Borland InterBase

- Zwischen 92 und 94 Hintertür eingebaut (für Installationsprogramm)
- Blieb über 6 Jahre enthalten
- Quellcode wurde 2000 freigegeben
- Firebird-Projekt arbeitet mit den Quellen und findet die Hintertür im Dezember 2000
- Januar 2001: Advisory
- Wäre durch ASCII-Dump zu entdecken gewesen (verbreiteter Cracker-Trick)

# Wo steckt die (Un)Sicherheit?

- ⇒ Fehler von vornherein vermeiden
  - Design des Systems
  - Sichere Programmierung
  - Konfiguration
- ⇒ Fehler finden, bevor es die Cracker tun
  - Review, Audit
- ⇒ Gefundene Fehler beheben
  - Bug Fixes
  - Advisories

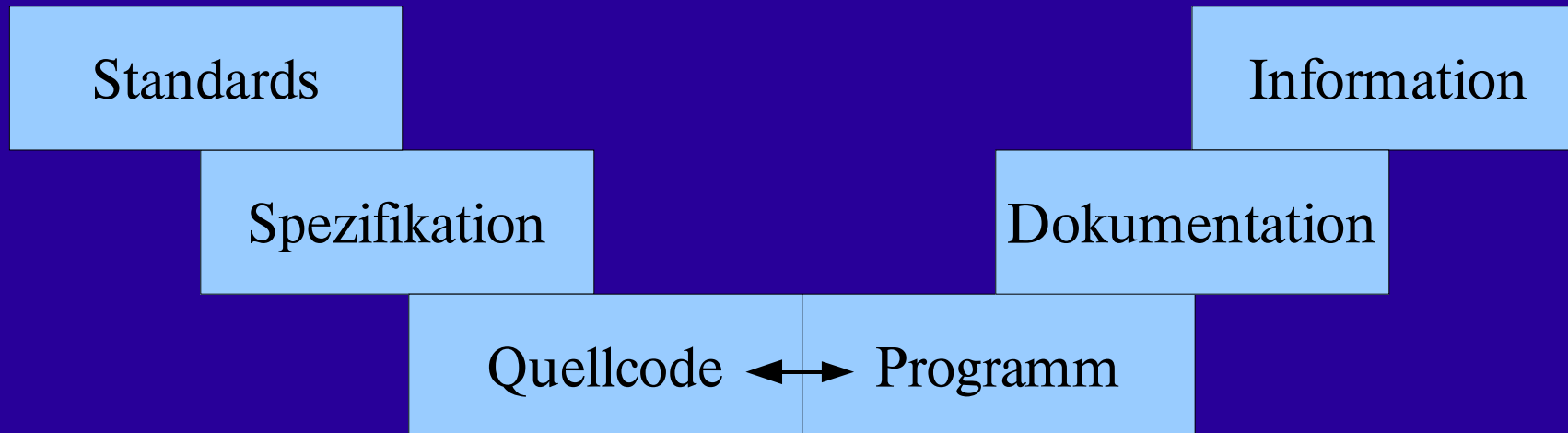
# Freie Software

- ⇒ Verfügbarkeit der Quellen nicht ausreichend
- ⇒ Recht, die Quellen zu ändern
  - Bugfixes
- ⇒ Recht, die Änderungen weiterzugeben
  - Falls Autor/Hersteller dies nicht tut
- ⇒ Recht, die Verbesserungen auch in anderen Programmen einzusetzen



# Offenheit an welchen Stellen?

⇒ Offenheit heißt mehr als nur offene Quellen:



# Offene Standards

- ⇒ Bekannte Vorteile
  - Interoperabilität
  - das Rad nicht ständig neu erfinden
- ⇒ Bekannte Fehler nicht wiederholen
- ⇒ Review-Prozess: Lücken bekannt und verstanden
- ⇒ Gilt in besonderem Maß für Krypto-Algorithmen

# Offene Spezifikation

- ⇒ Sichere Programme tun nur das tun, was sie auch sollen
- ⇒ Gewünschter Funktionsumfang bekannt?
- ⇒ Fehler kann schon in der Architektur stecken

# Offene Quellen

- ⇒ Der zentrale Punkt...
- ⇒ Disassemblieren, Reverse-Engineering unnötig
- ⇒ Reviews/Audits
  - möglich
  - legal
  - erwünscht
- ⇒ Bug Fixes sind möglich, legal und erwünscht

# Besserer Code

- ⇒ Open Source fördert lesbaren Code
  - Nur dann helfen andere Entwickler
  - Image des Entwicklers
- ⇒ Einhaltung der Standards
  - Direkt überprüfbar
- ⇒ Weniger Fehler
- ⇒ Review/Audit einfacher

# Offene Dokumentation

## ⇒ Gern unterschätzter Aspekt

- Undokumentierter Code ist meist unverständlich
- Fehler sind kaum zu finden

⇒ Je schwieriger ein Programm zu untersuchen ist, um so unwahrscheinlicher ist es, dass es jemand unbezahlt untersucht

# Offene Information

- ⇒ Sicherheitsinfos müssen vorhanden und leicht auffindbar sein
  - Konfigurationshinweise
  - Aktuelle Lücken / Patches
- ⇒ Detaillierte Beschreibung der Lücken
  - „Full Disclosure“ ⇒ verbreitet das Wissen
  - Eventuell mit Exploit („Proof of Concept“)
  - Aber: mit Vorwarnung, Patches *vor* der Veröffentlichung



# Fazit

- ⇒ Offenheit macht ein Programm nicht automatisch sicherer
- ⇒ Reviews und Audits müssen auch tatsächlich stattfinden
- ⇒ Wissen über die Entwicklung sicherer Programme muss stärker verbreitet werden
- ⇒ Dazu auch detaillierte Beschreibung der Lücken
- ⇒ Gefundene Fehler müssen behoben werden